**Section A – Short Answer Questions**

1. **Define Object-Oriented Programming. What are its main principles?**

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which contain data and code to manipulate that data. **Main principles:**

* **Encapsulation**: Combining data and methods into one unit.
* **Abstraction**: Hiding internal details and showing only essential features.
* **Inheritance**: Reusing code by inheriting from existing classes.
* **Polymorphism**: One interface, many implementations.

1. **Differentiate between class and object with an example.**

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

class Car:

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

self.model = model

my\_car = Car("Toyota") # my\_car is an object

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

* **Public**: Accessible everywhere.
* **Protected (\_var)**: Accessible within the class and subclasses.
* **Private (\_\_var)**: Accessible only within the class.

1. **What is a constructor in Python? What is its special method name?**

A constructor is a special method used to initialize objects.

The special method is \_\_init\_\_().

1. **What is the purpose of the super() function in inheritance?**

super() allows access to methods of a parent class. It helps in reusing the parent class functionality.

1. **Explain the difference between method overloading and method overriding.**

* **Overloading**: Multiple methods with the same name but different parameters (not directly supported in Python).
* **Overriding**: Redefining a parent class method in a child class.

1. **What is polymorphism in OOP? Give a simple example.**

Polymorphism allows different classes to define methods with the same name.

class Bird:

def sound(self):

print("Some sound")

class Parrot(Bird):

def sound(self):

print("Squawk")

obj = Parrot()

obj.sound() # Output: Squawk

1. **Can you call a private method outside its class in Python? If yes, how?**

Yes, using name mangling: \_ClassName\_\_methodName

class Test:

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

self.\_\_secret()

def \_\_secret(self):

print("This is private")

obj = Test()

obj.\_Test\_\_secret()

**Section B – Coding-Based Questions**

1. **Student class with attributes and display method**

class Student:

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

self.name = name

self.marks = marks

def display(self):

print("Name:", self.name)

print("Marks:", self.marks)

s = Student("Alice", 90)

s.display()

1. **Inheritance and method overriding**

class Vehicle:

def start(self):

print("Vehicle started")

class Car(Vehicle):

def start(self):

super().start()

print("Car started")

c = Car()

c.start()

1. **Demonstrate public, protected, and private variables**

class AccessDemo:

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

self.public\_var = "I am public"

self.\_protected\_var = "I am protected"

self.\_\_private\_var = "I am private"

def show(self):

print(self.public\_var)

print(self.\_protected\_var)

print(self.\_\_private\_var)

obj = AccessDemo()

obj.show()

print(obj.public\_var)

print(obj.\_protected\_var)

print(obj.\_AccessDemo\_\_private\_var)

1. **Animal-Dog-Cat hierarchy with polymorphism**

class Animal:

def make\_sound(self):

print("Some generic sound")

class Dog(Animal):

def make\_sound(self):

print("Woof!")

class Cat(Animal):

def make\_sound(self):

print("Meow!")

animals = [Dog(), Cat()]

for animal in animals:

animal.make\_sound()

**Section C – Application-Based Questions**

1. **BankAccount with deposit, withdraw, and balance display**

class BankAccount:

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

self.\_\_balance = 0

def deposit(self, amount):

if amount > 0:

self.\_\_balance += amount

print(f"Deposited: {amount}")

def withdraw(self, amount):

if amount <= self.\_\_balance:

self.\_\_balance -= amount

print(f"Withdrawn: {amount}")

else:

print("Insufficient balance!")

def display\_balance(self):

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

acc = BankAccount()

acc.deposit(1000)

acc.withdraw(500)

acc.withdraw(600)

acc.display\_balance()

1. **Employee and Manager with inheritance and method overriding**

class Employee:

def \_\_init\_\_(self, name, emp\_id, salary):

self.name = name

self.emp\_id = emp\_id

self.salary = salary

def display(self):

print("Name:", self.name)

print("ID:", self.emp\_id)

print("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("Department:", self.department)

mgr = Manager("John", 101, 50000, "HR")

mgr.display()