UNIT 5

1 Mark Questions and Answers

1. **What is a class in Python?**
   → A class is a blueprint for creating objects, bundling attributes (variables) and methods (functions).
2. **What is the purpose of the `__init__()` method in a class?**
   → It is a constructor used to initialize object attributes when an object is created.
3. **What does the `self` keyword represent?**
   → `self` represents the current instance of the class.
4. **Write the syntax to create an object of a class.**

   A) obj = ClassName()

5. Differentiate between class attribute and instance attribute.
→ Class attribute is shared by all objects of a class, whereas instance attribute is unique to each object.

6. **Can a class have multiple constructors in Python? (Yes/No)**
→ No, Python does not support multiple constructors directly.

7. Which keyword is used to define a class in Python?
→ `class`

8. What is the default return type of a method if `return` is not used?
→ `None`

Questions from Inheritance:

9. What is inheritance in Python?
→ Inheritance allows a class (child) to acquire properties and methods from another class (parent).

10. Which keyword is used for inheritance in Python?
→ `class Child(Parent):`

11. What type of inheritance allows a class to inherit from more than one parent?
→ Multiple Inheritance.

12. What is method overriding?
→ When a child class defines a method with the same name as the parent class, it overrides the parent's method.

13. Does Python support method overloading? (Yes/No)
→ No, but it can be achieved using default arguments or variable-length arguments.

14. What is the base class of all classes in Python?
→ `object`

`Encapsulation`

1. **What is encapsulation in Python?**
   → Encapsulation is the process of restricting direct access to class variables and methods.
2. **Which symbols are used for private and protected attributes in Python?**
   → `_var` for protected and `__var` for private.
3. **Can private variables be accessed directly outside a class? (Yes/No)**
   → No.
4. **What are getter and setter methods?**
   → Special methods used to access (get) and update (set) private attributes.

`Polymorphism`

5. **What is polymorphism in Python?**
   → The ability of a function/method to take on many forms (same name, different behavior).
6. **Which OOP principle allows method overriding?**
   → Polymorphism.
7. **Give an example of a built-in polymorphic function in Python.**
   → `len()` works for strings, lists, tuples, etc.

---

2 Mark Questions and Answers

1. **Explain the difference between a class and an object with an example.**
   → A class is a blueprint, and an object is an instance of the class.

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


c1 = Car("Tesla")  # c1 is an object
```

2. Write a simple Python class Car with attributes brand and model.

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

```
        self.model = model
```

3. **How is self used inside a class? Give an example.**
→ It is used to access instance variables and methods.

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

4. Define a class `Student` with a constructor that initializes `name` and `marks`.

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

5. Explain the role of methods in classes with an example.
→ Methods define the behavior of objects.

```
class Calculator:
   def add(self, a, b):
      return a + b
```

Questions from Inheritance:

6. Explain single inheritance with an example.
→ A child inherits from one parent class.

```
class Parent:
   def show(self): print("I am Parent")

class Child(Parent):
   pass

c = Child()
c.show()
```

7. Differentiate between single and multiple inheritance.

- **Single Inheritance** → Child inherits from one parent.
- **Multiple Inheritance** → Child inherits from two or more parents.

8. What is multilevel inheritance? Give an example.
→ In multilevel inheritance, a class is derived from another derived class.

```
class A: pass
class B(A): pass
class C(B): pass
```

9. Explain hierarchical inheritance.
→ Multiple child classes inherit from the same parent class.

10. What is the advantage of inheritance?
→ Code reusability and extensibility.

Encapsulation

1. **Differentiate between public, protected, and private members in Python.**
    - Public: Accessible anywhere.
    - Protected (`_var`): Accessible within class and subclasses.
    - Private (`__var`): Accessible only within the class.
2. **Write a simple program to demonstrate private variables.**

```
class Student:
    def __init__(self, name):
        self.__name = name  # private

    def get_name(self):
        return self.__name

s = Student("John")
print(s.get_name())
```

Polymorphism

3. **Differentiate between method overloading and method overriding.**
    - Overloading: Same method name, different parameters (not directly supported in Python).
    - Overriding: Child class redefines parent class method.
4. **Write an example of polymorphism using a common method.**

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

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

for animal in (Cat(), Dog()):
    animal.sound()
```

5. Explain the difference between encapsulation and polymorphism.

- **Encapsulation** → Bundles data and methods, restricts access.

- **Polymorphism** → Same function name, different behavior.
  Example: `len("abc")` → 3, `len([1,2,3])` → 3.

5 Mark Questions and Answers

1. **Explain the concept of class and object in Python with an example program.**
   → A class is a blueprint; objects are instances. Example:

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

    def display(self):
        print(f"Car: {self.brand}, Model: {self.model}")

c1 = Car("Tesla", "Model S")
c1.display()
```

2. Write a Python program using a class `Employee` with attributes `name` and `salary`. Include a method to display employee details.

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

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}")
```

3. Write a program using a class `Calculator` to perform addition, subtraction, multiplication, and division.

```
class Calculator:
    def add(self, a, b): return a + b
    def sub(self, a, b): return a - b
    def mul(self, a, b): return a * b
    def div(self, a, b): return a / b if b != 0 else "Error: Division by zero"

calc = Calculator()
print(calc.add(10, 5))
print(calc.sub(10, 5))
print(calc.mul(10, 5))
print(calc.div(10, 0))
```

4. Discuss the difference between class variables and instance variables with examples.

- **Class variable** → Shared among all objects.

- **Instance variable** → Unique to each object.

```
class Student:
    school = "ABC School"  # Class variable
    def __init__(self, name):
        self.name = name  # Instance variable
s1 = Student("John")
s2 = Student("Emma")
print(s1.school, s1.name)
print(s2.school, s2.name)
```

5. Create a class `Book` with attributes `title`, `author`, and `price`. Write methods to display the book details and check if the book is expensive (price > 500).

```
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def display(self):
        print(f"Title: {self.title}, Author: {self.author}, Price: {self.price}")

    def is_expensive(self):
        return self.price > 500

b1 = Book("Python Basics", "Guido", 650)
b1.display()
print("Expensive:", b1.is_expensive())
```

Questions from Inheritance:

6. **Explain the different types of inheritance in Python with examples.**
   - **Single Inheritance** – One parent, one child.
   - **Multiple Inheritance** – Child inherits from many parents.
   - **Multilevel Inheritance** – Inheritance across multiple levels.
   - **Hierarchical Inheritance** – Many children inherit from one parent.

```
# Example: Multiple Inheritance
class A:
    def methodA(self): print("Class A")
class B:
    def methodB(self): print("Class B")
class C(A, B):
    pass

obj = C()
obj.methodA()
```

```
obj.methodB()
```

7. **Explain encapsulation and polymorphism in relation to inheritance.**
   - **Encapsulation** → Restricts direct access to class variables (e.g., using private variables `__var`).
   - **Polymorphism** → Same function name behaves differently in different classes.

```
class Bird:
   def fly(self): print("Bird can fly")
class Penguin(Bird):
   def fly(self): print("Penguin cannot fly")

b1, b2 = Bird(), Penguin()
b1.fly()
b2.fly()
```

8. Write a program using multilevel inheritance where `Person` → `Employee` → `Manager`.

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

class Employee(Person):
   def __init__(self, name, emp_id):
      super().__init__(name)
      self.emp_id = emp_id

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

   def display(self):
      print(f"Name: {self.name}, ID: {self.emp_id}, Dept: {self.dept}")

m = Manager("Alice", 101, "HR")
m.display()
```

Polymorphism

9. Write a program to demonstrate polymorphism using method overriding.

```
class Shape:
   def area(self):
      print("Area not defined")

class Circle(Shape):
   def __init__(self, r):
      self.r = r
```

```
    def area(self):
        return 3.14 * self.r * self.r

class Square(Shape):
    def __init__(self, s):
        self.s = s
    def area(self):
        return self.s * self.s

for shape in (Circle(5), Square(4)):
    print(shape.area())
```

10. Discuss how Python achieves polymorphism without method overloading. Give an example.
→ Python achieves it by using default arguments or `*args`.

```
class Math:
    def add(self, a, b=0, c=0):
        return a + b + c

m = Math()
print(m.add(5))        # 1 argument
print(m.add(5, 10))    # 2 arguments
print(m.add(5, 10, 15))   # 3 arguments
```