Python

UNIT – 3

Two Marks Questions :

1. **What is Class in Python ?**

Ans : A **class** in Python is a blueprint for creating objects. It defines a **data structure** that groups related **attributes (variables)** and **methods (functions)** together. A class provides a way to **model real-world entities** with properties and behaviors.

2. **What is the purpose of __init__ methods in python ?**

Ans : The __init__ method is a constructor in python classes. It is automatically called when a new object (instance) of a class is created. it's primary purpose is to initialize attributes of the object.

Syntax of __init__ :

```
class ClassName:
    def __init__(self, param1, param2):
        self.param1 = param1  # Assign parameter to instance variable
        self.param2 = param2
```

3. **Explain the concept of inheritance in Python.**

Ans : Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a new class (**child class**) to inherit attributes and methods from an existing class (**parent class**). This promotes **code reusability** and **hierarchical structuring.**

4. What is the difference between a class method and an instance method ?

Ans :

Instance Method	Class Method
A method that operates on an instance of a class.	A method that operates on the class itself.
No decorator (default)	@classmethod
self (refers to the instance)	**cls** (refers to the class)
Access using Instance attributes	Access using Class Attributes
we use it, when working with instance-specific data	we use it, when the method affects the class or all instances.

Instance Method Example (optional) :

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand  # Instance attribute
        self.model = model
    def display_info(self):  # Instance method
        return f"Car: {self.brand} {self.model}"
# Creating an instance
car1 = Car("Toyota", "Corolla")
print(car1.display_info())
```

OUTPUT :

Car : Toyota Corolla

Class Method Example (Optional) :

```
class Car:
    brand = "Toyota"  # Class attribute
    @classmethod
    def change_brand(cls, new_brand):  # Class method
        cls.brand = new_brand  # Modifies class attribute
# Calling the class method
Car.change_brand("Honda")
# Checking the change
print(Car.brand)
```

OUTPUT :

Honda

5. **Write a Python class Circle with an attribute radius and a method area() to calculate the area of the circle.**

Ans :

```python
import math  # Importing math module for π (pi)

class Circle:
    def __init__(self, radius):  # Constructor to initialize radius
        self.radius = radius

    def area(self):  # Method to calculate area
        return math.pi * self.radius ** 2  # Formula: πr²

# Example usage:
circle1 = Circle(5)  # Creating an instance with radius 5
print(f"Area of the circle: {circle1.area():.2f}")  # Output: 78.54
```

Output :

78.54

6. Create a class Student with attributes name and roll_no. Add a method display() to print the student's details.

Ans :

```python
class Student:
    def __init__(self, name, roll_no):  # Constructor to initialize attributes
        self.name = name
        self.roll_no = roll_no

    def display(self):  # Method to print student details
        print(f"Student Name: {self.name}, Roll No: {self.roll_no}")

# Example usage:
student1 = Student("Alice", 101)  # Creating an instance
student1.display()  # Output: Student Name: Alice, Roll No: 101
```

OUTPUT :

Student Name : Alice, Roll No : 101

7. **What is the purpose of the super() function in Python ? Provide an example.**

Ans : The **Super()** function is used to call a method from a **parent class (super class)** in a **child class (subclass).** It allows for : **Accessing parent class methods, Code reusability, Supporting multiple inheritance.**

Example :

```python
class Parent:
    def show(self):
        print("This is the Parent class")

class Child(Parent):
    def show(self):
        super().show()  # Calling Parent's method
        print("This is the Child class")

# Creating an object of Child class
obj = Child()
obj.show()
```

Output :

This is the Parent Class

This is the Child Class

8. Explain the concept of data hiding in Python with an example.

Ans : Data hiding is a principle in **object-oriented programming (OOP)** that restricts access to certain attributes or methods of a class. This is done to prevent **accidental modification** of critical data and to enforce **encapsulation**.

- **Using a Single underscore _attribute :** Indicates a protected attribute (Convention only, still accessible).
- **Using double underscore __attribute :** Makes an attribute private (name mangling is applied).

Example :

class BankAccount:

```
    def __init__(self, balance):
        self.__balance = balance  # Private attribute
    def deposit(self, amount):
        self.__balance += amount
    def get_balance(self):  # Public method to access balance
        return self.__balance
account = BankAccount(1000)
account.deposit(500)
print("Balance:", account.get_balance())
```

Output :

```
Balance : 1500
# trying to access private attribute directly ( will cause an error )
```

9. Why is abstraction important in OOP? Provide an example.

Ans : Abstraction is one of the fundamental principles of **Object-Oriented Programming (OOP)** that hides unnecessary details from the user and only shows essential features.

Importance of Abstraction :

- Hides Complexity.
- Improves Code Maintainability.
- Enhances Security.
- Encourages Modular Code.

Example :

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

dog = Dog()
print(dog.sound())  # Output: Woof!
```

Output :

Woof!

10. What is the advantage of using polymorphism in Python ? Provide an Example.

Ans : Polymorphism allows the same method name to be used for different types of objects.

- **Code Reusability :** The same method can be used for different objects.
- **Flexibility :** Objects of different classes can be treated uniformly.
- **Scalability :** Makes it easier to extend code without modifying existing functionality.

- **Readability :** Reduces redundant code and improves program structure.

Example :

```
class Cat:
    def sound(self):
        return "Meow!"

class Dog:
    def sound(self):
        return "Woof!"

# Using polymorphism
for animal in [Cat(), Dog()]:
    print(animal.sound())
```

Output :

Meow !

Woof !