Answer Script

Question No. 01
➔ Write down the differences between class method and static method of Python with proper examples.(at least 3)
Answer No. 01

Class Method	Static Method
A class method is defined using the @classmethod decorator.	A static method is defined using the @staticmethod decorator.
Class methods can be accessed both by the class itself and by its instances.	Static methods can be accessed directly through the class without creating an instance of the class.
It takes the class as the first parameter (self).	It does not take any special parameters.

Example :
 	
@classmethod
def class_method(self):
    print("This is Class method:", self.x)
	Example :
	
	@staticmethod
	def static_method():
	    print("This is Static method")

Question No. 02
➔ Explain with proper examples what is meant by polymorphism in Python.
Answer No. 02
In Python, a generic function name that may be applied to various types is called polymorphic. Operator overloading and method overriding are used to create polymorphism.

1. Operator Overloading : If we want to do some operation between two different class, there will be some error, because we aren't supposed to do operation between classes. For this we should use operator overloading. To do operation, through the operators between classes, we have to make make override the operations manually which is operator overloading.

Example :

```
def _add__(self, other):
    return self.x + other.y
```

this will return the summation of self class's x attribute and other class's y attribute's value. Where self and other refers two different class. And normally, we aren't supposed to add two class's attribute's value.

2. Method Overriding : If there are methods in a class which is also written in its inherited class with the same name, then we use method overriding.

Suppose we created a method in a parent class, if we create that same method with the exact same name in the parent class's derived class's also, then we will get the derived class's methods output what it returns. If no exact same method with same name isn't created in any derived class, then it will return its parent class's method's output. This is the Method Overriding.

Example :

```
class Animal:
    def sound(self):
        print("Animal makes a sound.")

class Dog(Animal):
    def sound(self):
        print("Dog barks.")

class Cat(Animal):
    def __init__(self):
        pass
```

The Dog class will override its parent class's sound method and will return its own class's sound method's output which is "Dog barks". On the other hand, the cat class doesn't have any specific sound method. so it will bring it's parent class's sound method with that output.

Question No. 03

➔ Write a class with three instance variables a,b and c.
Now add the following two methods in that class
   a) **sum()** to get the sum of a,b and c.
   b) **factorial()** to get the factorial of b.

Answer No. 03

```
from math import factorial
class Calculator:
    def __init__(self, a, b, c) -> None:
        self.a = a
        self.b = b
        self.c = c

    def sum(self):
        return self.a + self.b + self.c

    def factorial(self):
        return factorial(self.b)

test = Calculator(2, 5, 3)
print(test.sum())
print(test.factorial())
```

Question No. 04

➜ Explain with proper examples what is meant by multilevel inheritance in Python.

Answer No. 04

Multi-level inheritance :

Inheriting a class that has already inherited by another class is known as multilevel inheritance in the Python. Accordingly, the base class/parent class's features are inherited by the derived/subclass class, and those of the new derived class are inherited by the derived class. This is the Multi-level inheritance.

Example :

```
class Institute:
    def __init__(self) -> None:
        pass

class University(Institute):
    def __init__(self) -> None:
        pass

class Student(University):
    def __init__(self) -> None:
        pass
```

Question No. 05

➜ Write the advantages of using inner functions in Python OOP. Provide specific use cases where inner functions can enhance code readability and organization. (note: answer with proper examples).

Answer No. 05

The advantages of using inner functions in Python OOP :

1. Improves code organization.
2. Enhanced code readability.

3. Increases code reusability.
4. Simplifies passing data as parameter.
5. Increase performance significantly.

Specific use cases of inner functions :

1. Memoization : Inner functions can be used for memoization, which is a technique to cache the results of expensive function calls to avoid repetitive function call to save time.

2. Decorator : we use inner functions in decorator, to enhance code readability and better performance. For this, parameter passing in a nested functions becomes so easy.

Example :

```python
def uppercase(func):
    def inner(name):
        result = func(name)
        return result.upper()
    return inner

@uppercase
def greet(name):
    return f"Hello, {name}!"
print(greet("John"))
```

Here we can see a @uppercase decorator which is made by inner function/nested function, which is much more readable and easy to send parameter.

Question No. 06

➔ Write Python program to solve **Frequency Array**

Answer No. 06

```
n, m = list(map(int, input().split()))
num = list(map(int, input().split()))

ans = {}

for i in num:
    if i not in ans:
        ans[i] = 1
    else:
        ans[i] += 1

for j in range(1, m+1):
    if j in ans:
        print(ans[j])
    else:
        print(0)
```

Question No. 07

➔

```
class Person:
    def __init__(self, name, age, height, weight) -> None:
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight


class Cricketer(Person):
    def __init__(self, name, age, height, weight) -> None:
        super().__init__(name, age, height, weight)
```

Sakib = Cricketer('Sakib', 38, 68, 91)
Mushfiq = Cricketer('Mushfiq', 36, 55, 82)
Mustafiz = Cricketer('Mustafiz', 27, 69, 86)
Riyad = Cricketer('Riyad', 39, 72, 92)

Modify the Cricketer class to find the youngest player using the concept of operator overloading and lastly print his name.

Answer No. 07

```python
class Person:
    def __init__(self, name, age, height, weight) -> None:
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight


class Cricketer(Person):
    def __init__(self, name, age, height, weight) -> None:
        super().__init__(name, age, height, weight)

    def __lt__(self, other):
        if self.age < other.age:
            return True
        else:
            return False

Sakib = Cricketer('Sakib', 38, 68, 91)
Mushfiq = Cricketer('Mushfiq', 36, 55, 82)
Mustafiz = Cricketer('Mustafiz', 27, 69, 86)
Riyad = Cricketer('Riyad', 39, 72, 92)

players = [Sakib, Mushfiq, Mustafiz, Riyad]


min = 99999
```

```python
for i in range(len(players)):
    for j in range(len(players)):
        if players[j].age < players[i].age and (players[j].age <
min):
            min = players[j].age
            player = players[j]


print(f'The youngest player is : {player.name}. His age is : {min}
years')
```