Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: Assignment 2**	
Title of LAB Assignment: Modularization and Classes		
DOP: 31-10-2023	**DOS:02-10-2023**	
CO Mapped:	**PO Mapped:**	**Signature:**

Q1. How do we implement abstract methods in Python? Give an example for the same.

In Python, you can implement abstract methods using abstract base classes (ABCs) from the abc module. To define an abstract method, you use the @abstractmethod decorator, and then you must provide a concrete implementation in a subclass. Here's an example:

Code:

```
from abc import ABC, abstractmethod
```

```
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

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

    def area(self):
        return 3.14 * self.radius ** 2

circle = Circle(5)
print("Area of the circle:", circle.area())
```

In this example, we define an abstract class Shape with an abstract method area(). Then, we create a subclass Circle and provide a concrete implementation of the area() method. Attempting to create an instance of the abstract class directly will result in an error, and you should always instantiate concrete subclasses instead.

Q2. Is function overloading supported by Python? Give reasons.

Python does not support function overloading in the traditional sense as seen in languages like C++ or Java, where you can define multiple functions with the same name but different parameter lists. In Python, the last defined function with a given name will override any previous definitions with the same name.

However, Python provides a flexible way to handle different argument types and numbers using default arguments and variable-length argument lists. You can define a single function that handles different cases based on the arguments passed. This makes explicit function overloading unnecessary.

Q3. Explain the importance of self in Python classes.

In Python, self is a convention used as the first parameter in the methods of a class. It refers to the instance of the class itself, allowing you to access and manipulate the instance's attributes and methods. Here's the importance of self in Python classes:

Accessing Instance Variables: You can use self to access and modify the instance's attributes. For example, self.variable refers to an instance variable.

Calling Other Methods: self allows you to call other methods within the class using self.method().

Creating Multiple Instances: Each instance of a class has its own set of attributes, and self helps distinguish which instance is being accessed or modified.

Maintaining State: It helps maintain the state of an object and ensures that changes are applied to the correct instance.

Q4. Explain the usage of the keyword 'pass' in class definition.

In Python, the pass keyword is a placeholder statement. It does nothing and serves as a way to create empty code blocks or class definitions without causing a syntax error. It is often used when you want to create a class, function, or code structure that you intend to implement later. For example, when defining a class with no methods or attributes yet, you can use pass:

Code:

```
class MyClass:
    pass
```