Student Name: _____ Roll No: _____ Section: _____

Lab No. 13

Lab 13 – Introduction to Interface in Python and operator overloading

Objectives:

- Introduction to Interface
- Introduction to Interface in Python
- Operator Overloading
- Operator Overloading in Python
-

1. Introduction to Interface

n object-oriented languages like Python, the interface is a collection of method signatures that should be provided by the implementing class. Implementing an interface is a way of writing an organized code and achieve abstraction.

There's no interface keyword in Python. The Java / C# way of using interfaces is not available here. In the dynamic language world, things are more implicit. We're more focused on how an object behaves, rather than it's type/class

- If abstract class contains only abstract methods then it is interface.
- Abstract class can contain both abstract methods and concrete methods.

2. Python Interface

Interface acts as a blueprint for designing classes. Like classes, interfaces define methods. Unlike classes, these methods are abstract. An abstract method is one that the interface simply defines. It doesn't implement the methods. This is done by classes, which then implement the interface and give concrete meaning to the interface's abstract methods.

Python's approach to interface design is somewhat different when compared to languages like Java, Go, and C++. These languages all have an interface keyword, while Python does not. Python further deviates from other languages in one other aspect. It doesn't require the class that's implementing the interface to define all of the interface's abstract methods

Student Name: _____ Roll No: _____ Section: _____

Exercise 1:

```
from abc import ABC, abstractmethod

class NetworkInterface(ABC):

    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def transfer(self):
        pass


class RealNetwork(NetworkInterface):

    def connect(self):
      print("Network is connect")

    def transfer(self):
        print("All Data is tranfser")

R1= RealNetwork()
R1.connect()
R1.transfer()
```

Output:

Exercise 2:

```
from abc import ABC, abstractmethod

class NetworkInterface(ABC):

    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def transfer(self):
        pass
```

```
class RealNetwork(NetworkInterface):

    def connect(self):
      print("Network is connect")

    def transfer(self):
        print("All Data is tranfser")

class FakeNetwork(NetworkInterface):

    def connect(self):
       print("Network is connect")

R1= RealNetwork()
R1.connect()
R1.transfer()

F1= FakeNetwork()
F1.connect()
```

Output:

Can't instantiate abstract class FakeNetwork with abstract methods transfer that's the abstract base class enforcing the interface. As long as FakeNetwork is missing transfer, we can't create an instance of it.

3. Operator Overloading

Operator overloading in Python is the ability of a single operator to perform more than one operation based on the class (type) of operands.

For example, the + operator can be used to add two numbers, concatenate two strings or merge two lists. This is possible because the + operator is overloaded with int and str classes.

Similarly, you can define additional methods for these operators to extend their functionality to various new classes and this process is called Operator overloading.

Student Name: _____ Roll No: _____ Section: _____

4. Python Operator Overloading

Python operators work for built-in classes. But same operator behaves differently with different types.

This feature in Python, that allows same operator to have different meaning according to the context is called operator overloading.

You have already seen you can use + (addition) operation operator overloading.

Some other operator are as follows.

Operator	**Description**	**Method**
Addition	p1 + p2	p1._add_(p2)
Subtraction	p1 - p2	p1._sub_(p2)
Multiplication	p1 * p2	p1._mul_(p2)
Power	p1 ** p2	p1._pow_(p2)
Division	p1 / p2	p1._truediv_(p2)
Floor Division	p1 // p2	p1._floordiv_(p2)
Remainder (modulo)	p1 % p2	p1._mod_(p2)
Bitwise Left Shift	p1 << p2	p1._lshift_(p2)
Bitwise Right Shift	p1 >> p2	p1._rshift_(p2)
Bitwise AND	p1 & p2	p1._and_(p2)
Bitwise OR	p1 | p2	p1._or_(p2)
Bitwise XOR	p1 ^ p2	p1._xor_(p2)
Bitwise NOT	~p1	p1._invert_()

Relational Operators in python

Operator	**Description**	**Method**
>	Greater than	__gt_(self, other)
>=	Greater than or equal to	__ge_(self, other)
<	Less than	__lt_(self, other)
<=	Less than or equal to	__le_(self, other)
==	Equal to	__eq_(self, other)
!=	Not equal to	__ne_(self, other)

Student Name: _____ Roll No: _____ Section: _____

Exercise 3:

```
class A:
    # defining init method for class
    def_init_(self, x,y):
        self.x = x
        self.y = y


    # overloading the add operator using special function
    def_add_(self, other):
        x = self.x + other.y
        y = self.x + other.y #2+
        return( x,y)
c1 = A(5,3)

c2 = A(2,3)
print("sum = ",c1+c2)
```

Output:

Exercise 4:

```
class A:
    # defining init method for class
    def_init_(self, x):
        self.x = x

    def_lt__(self, other):
        if self.x < other.y:
            return True
```

```
        else:
             return False


class B:
    # defining init method for class
    def __init__(self, y):
        self.y = y


c1 = A(2)
c2 = B(4)
print("Is c1 less than c2: ",c1<c2)
```

Output:

Exercise 5:

```
class A:
    def __init__(self, a):
        self.a = a
    def __lt__(self, other):
        if(self.a<other.a):
            return "ob1 is less than ob2"
        else:
            return "ob2 is less than ob1"
    def __eq__(self, other):
        if(self.a == other.a):
            return "Both are equal"
        else:
            return "Not equal"
```

```
ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)


ob3 = A(4)
ob4 = A(4)
print(ob3 == ob4)
```

Output:

Student Name:_____ Roll No:_____ Section: _____

Programming Exercise (Python)

Task 1: Create a class Point having X and Y axis then perform an operator overloading (Overload all relation operator).

Task 2:

Find out one real world example of interface and implement all abstract method by using python n code.