Lesson-14: Miscellaneous Concepts In Python

Achieving Method Overloading:

NOTE: There is no method overloading in python

Program 1:

#This is a very famous interview question.

```
class MyMath:
    def sum(self, a=None, b=None, c=None):
        if(a != None and b != None and c != None):
            return (a + b + c)

        elif(a != None and b != None ):
            return (a + b)

m = MyMath()
print(m.sum(1,2,3))
print(m.sum(1.5, 2.5))
```

Output:

```
6
4.0
```

Operator Overloading

Program 2:

```
class classy:

    def __init__(self, a, b ):
        self.__a = a
        self.__b = b

    def __add__(self, obj ):
        ta = self.__a + obj.__a
        tb = self.__b + obj.__b
        cc = classy(ta, tb)
        return cc

    def __str__(self):
        return "a = " + str(self.__a) + " b = " + str(self.__b)

cobjone = classy(1,2)
cobjtwo = classy(1,2)
cobjthree = cobjone + cobjtwo
print(cobjthree)
```

Output:

```
a = 2
b = 4
```

Program 3:

```
class classy:

    def __init__(self, a, b ):
        self.__a = a
        self.__b = b

    def __sub__(self, obj ):
        return classy((self.__a - obj.__a), (self.__b - obj.__b))

    def __str__(self):
        return "a = " + str(self.__a) + " b = " + str(self.__b)

cobjone = classy(1,2)
cobjtwo = classy(1,2)
cobjthree = cobjone - cobjtwo
print(cobjthree)
```

Output:

```
a = 0
b = 0
```

Program 4:

```
class classy:

    def __init__(self, a, b ):
        self.__a = a
        self.__b = b

    def __mul__(self, obj ):
        return classy((self.__a * obj.__a), (self.__b * self.__b))

    def __str__(self):
        return "a = " + str(self.__a) + " b = " + str(self.__b)

cobjone = classy(1,2)
cobjtwo = classy(1,2)
cobjthree = cobjone * cobjtwo
print(cobjthree)
```

Output:

```
a = 1
b = 4
```

Program 5:

```
class classy:

    def __init__(self, a, b ):
        self.__a = a
        self.__b = b

    def __neg__(self ):
        return classy(-(self.__a ), -(self.__b))

    def __str__(self):
        return "a = " + str(self.__a) + " b = " + str(self.__b)

cobjone = classy(1,2)
cobjtwo = -cobjone
print(cobjtwo)
```

Output:

```
a = -1
b = -2
```

Program 6:

```
class classy:
    def __init__(self, a ):
        self.__a = a

    def __lt__(self, obj):
        return self.__a < obj.__a

    def __le__(self, obj):
        return self.__a <= obj.__a

    def __ge__(self, obj):
        return self.__a >= obj.__a

    def __gt__(self, obj):
        return self.__a > obj.__a

    def __ne__(self,obj):
        return self.__a != obj.__a

    def __eq__(self,obj):
        return self.__a == obj.__a

cobjone = classy(1)
cobjtwo = classy(2)

print(cobjone < cobjtwo)
print(cobjone <= cobjtwo)
```

Output:

```
True
True
False
False
False
True
```

```
print(cobjone >= cobjtwo)
print(cobjone > cobjtwo)
cobjthree = classy(0)
cobjfour  = classy(0)
print(cobjthree != cobjfour)
print(cobjthree == cobjfour)
```

Command Line Arguments:

Program 7:

```
import sys

print("The length of CLA is = ", len(sys.argv))
print("The values of CLA are:")

for value in sys.argv:
        print(value)
```

Output:

```
C:\> py -3 clacla.py subhash loves india
The length of CLA is =  4
The values of CLA are:
clacla.py
subhash
loves
india
C:\>
```

Program 8:

```
import sys

print("Adding two numbers through CLA")
print(int(sys.argv[1]) + int(sys.argv[2]))
```

Output:

```
C:\> py -3 clacla.py 2 3
Adding two numbers through CLA
5
C:\>
```

Assignments:

Write a program to accept two files names as input through command line and do a copy operation.

Program 9:

```
import argparse

parser = argparse.ArgumentParser(description="This program will add two floats")
parser.add_argument("numone", type=float, help = "Enter a float type number")
parser.add_argument("numtwo", type=float, help = "Enter a float type number")
args = parser.parse_args( )

result = args.numone + args.numtwo

print(result)
```

Output:

```
C:\> py -3 clacla.py 1 2
3
C:\>
```

Program 10:

```
import argparse

parser = argparse.ArgumentParser(description="This program will add three numbers")
parser.add_argument("num",nargs=3,type=int, help="Pass two values")
args = parser.parse_args( )

result = int(args.num[0]) + int(args.num[1]) + int(args.num[2])

print(result)
```

Output:

```
C:\> py -3 clacla.py 1 2 3
6
C:\>
```

Program 11:

```
import argparse

parser = argparse.ArgumentParser(description="This program will accept all arguments")
```

```
parser.add_argument("values", nargs = '*', help="Enter anything")
args = parser.parse_args()

for val in args.values:
    print(val)
```

Output:

```
C:\> py -3 clacla.py Subhash 10 2.3 [1,2,3]
subhash
10
2.3
[1,2,3]
C:\>
```

<u>Program 12:</u>

```
import argparse

parser = argparse.ArgumentParser(description="This program will accept 1 or more arguments")
parser.add_argument("values", nargs='+', help="Enter one or more arguments")
args = parser.parse_args()

for val in args.values:
    print(val)
```

Output:

```
C:\> py -3 clacla.py 1 subhash [1,2,3]
1
subhash
[1,2,3]
C:\>
```