## Object Oriented Programming Exercises

**Program to implement concepts of Object Oriented Programming such as classes, inheritance and polymorphism.**

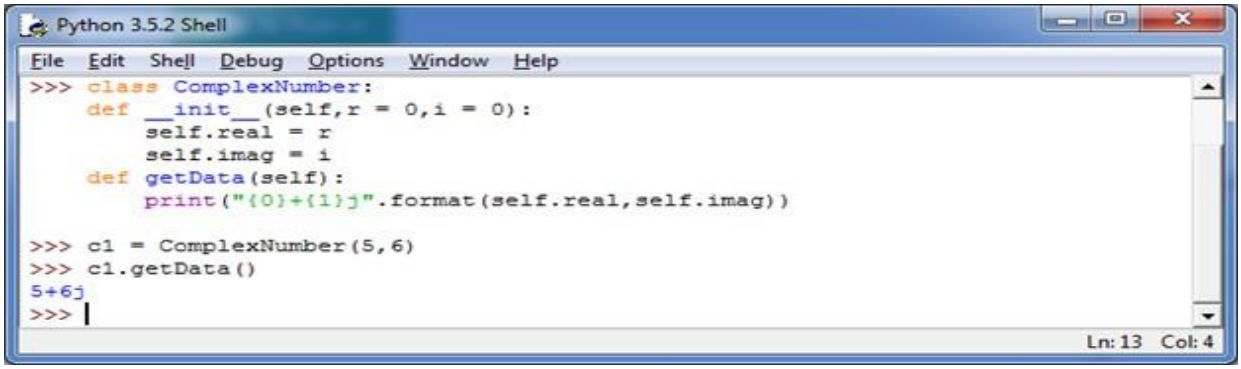
1. **Program to implement an object class.**

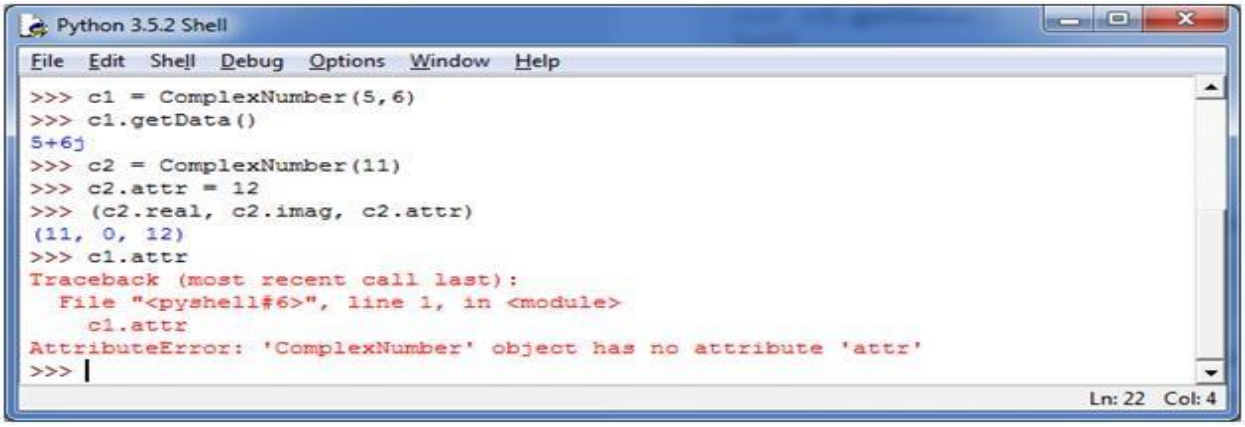
```
class Student:
  def___init__(self,  rollno,  name):
    self.rollno = rollno
    self.name = name
  def displayStudent(self):
    print ("rollno : ", self.rollno,  ", name: ", self.name )
emp1 = Student(121, "Ajeet")
emp2  =  Student(122,  "Sonoo")
emp1.displayStudent()
emp2.displayStudent()
```

**Output:**

**rollno : 121 , name: Ajeet**
**rollno : 122 , name: Sonoo**

2. **Program to demonstrate the concept of constructors**



## 3. Program to demonstrate Single inheritance in Python

```
class Animal:
 def eat(self):
      print  (  'Eating...')
class Dog(Animal):
   def bark(self):
   print ('Barking...')
d=Dog()
d.eat()
d.bark()
```

**Output:**

**Eating...**
**Barking...**

## 4. Program to demonstrate multilevel inheritance in Python

```
class Animal: def
    eat(self):
      print('Eating...'     )
class Dog(Animal):
   def bark(self):
    print ('Barking...')
class BabyDog(Dog):
    def weep(self):
     print( 'Weeping...')
d=BabyDog()
d.eat()
d.bark()
d.weep()
```

**Output:**

**Eating...**
**Barking...**
**Weeping**

## 5. Program to demonstrate multiple inheritance in Python / use of super keyword

```
class First(object):
  def___init___(self):
    super(First, self)._____init___()
    print("first")

class Second(object):
```

```
    def___init___(self):
      super(Second, self).______init___()
      print("second")

  class Third(Second, First):
    def___init___(self):
      super(Third, self)._____init___()
      print("third")

  Third();
```

Output:

```
first
second
third
```

## 6. Program to demonstrate operator overloading

```
import math

class Circle:

 def __init__(self, radius):

    self.__radius = radius


 def setRadius(self, radius):

    self.__radius = radius


 def getRadius(self):

    return self.__radius


 def area(self):

  return math.pi * self.__radius ** 2


 def __add__(self, another_circle):

  return Circle( self.__radius + another_circle.__radius )


c1 = Circle(4)
```

```
print(c1.getRadius())


c2 = Circle(5)

print(c2.getRadius())


c3 = c1 + c2 # This became possible because we have overloaded + operator by adding a
    method named add

print(c3.getRadius())
```

**Expected Output:**

**4**

**5**

**9**

## 7. Program to demonstrate method overriding

```
class A():

    def___init___(self):
            self.___x = 1

    def m1(self):
      print("m1 from A")


class B(A):

    def___init(self):
     self.__y = 1

    def m1(self):
       print("m1 from B")

c = B()
c.m1()
```

**Expected Output:m1 from B**

1. Write a Python class named Rectangle constructed by a length and width and a method which will compute the area of a rectangle
2. Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.