ASSIGNMENT 3

QUESTION 1. What is the concept of an abstract superclass?
ANSWER :
An abstract superclass can force its child class to create a set of methods, which are defined in superclass using `@abstractmethod` decorator.
For example:

```
from abc import ABC, abstractmethod

class Shape(ABC):
   @abstractmethod
   def printarea(self):
       return 0

class Square(Shape):
   def __init__(self, length):
       self.length = length

   def printarea(self):
       return self.length ** 2

b = Square(5)
print(b.printarea()) -–>25
```

QUESTION 2. What happens when a class statement's top level contains a basic assignment statement?
ANSWER :
When a basic assignment statement appears at the top level of a class statement, it treats it like a class attribute. Like all class attributes, this will be shared by all instances; data attributes are not callable method functions, though.

```
class Books:
   pages = 100

X = Books()
Y = Books()
print(X.pages) -–>100
print(Y.pages) -–>100
```

QUESTION 3. Why does a class need to manually call a superclass's __init__ method?
ANSWER :
When we define an `__init__()` method for a child class, we're replacing the `__init__()` method of its parent class, and the `__init__()` method from the parent class is not called automatically anymore. Therefore, a child class needs to manually call a superclass's `__init__` method using super function if it needs to use superclass attributes.

QUESTION 4. How can you augment, instead of completely replacing, an inherited method?
ANSWER :
To augment instead of completely replacing an inherited method, we redefine the method with the same name in a subclass, then by using the `super()` we can use the attributes we need and we can create new attributes as well.
For example, we can see the method `info` has been inherited from `data` class into `newdata` class and has been augmented by adding the `lastname` attribute to it.

```
class data:
   def info(self, name):
       self.name = name
       self.age = 100

class newdata(data):
   def info(self, name, lastname):
       super().info(name)
       self.lastname = lastname

a = newdata()
a.info('Gaurav', 'Rajput')
print(a.name)
print(a.age)
print(a.lastname)

OUTPUT:
Gaurav
100
Rajput
```

QUESTION 5. How is the local scope of a class different from that of a function?
ANSWER :
- When a variable is defined inside a function, it is available for use within the function only, hence the local scope.
- In case of class, as soon as the compiler reads the keyword `class` a new namespace is created, and used as the local scope and all assignments to local variables go into this new namespace.
- In the example below, `obj = MyClass()` creates a new *instance* of the class and assigns this object to the *local variable* `obj`.

For example,

```
class MyClass:
   var1 = 'Class attribute var1 of Class_Local_Scope.MyClass'
   def __init__(self, var2):
       self.var2 = var2
       self.var3 = 'Instance attribute var3 of Class_Local_Scope.MyClass'
       var4 = 'This variable is local to __init__ of MyClass'
       print(var4)

   def printdata(self):
       print(MyClass.var1)
       print(self.var2)
```

```
        print(self.var3)

obj = MyClass('Instance attribute var2 of Class_Local_Scope.MyClass')
# print(obj.var1)
# print(obj.var2)
# print(obj.var3)
obj.printdata()
```

OUTPUT:

```
This variable is local to __init__ of MyClass
Class attribute var1 of Class_Local_Scope.MyClass
Instance attribute var2 of Class_Local_Scope.MyClass
Instance attribute var3 of Class_Local_Scope.MyClass
```