**1. What is the concept of an abstract superclass?**

**ANS:-**

**A class is called an Abstract class if it contains one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and its abstract methods must be implemented by its subclasses.**

**import abc**

**class Shape(metaclass=abc.ABCMeta):**

**@abc.abstractmethod**

**def area(self):**

**pass**

**class Rectangle(Shape):**

**def \_\_init\_\_(self, x,y):**

**self.l = x**

**self.b=y**

**def area(self):**

**return self.l\*self.b**

**r = Rectangle(10,20)**

**print ('area: ',r.area())**

**area: 200**

**The 'abc' module in Python library provides the infrastructure for defining custom abstract base classes.**

**'abc' works by marking methods of the base class as abstract. This is done by @absttractmethod decorator. A concrete class which is a sub class of such abstract base class then implements the abstract base by overriding its abstract methods.**

**The abc module defines ABCMeta class which is a metaclass for defining abstract base class. Following example defines Shape class as an abstract base class using ABCMeta. The shape class has area() method decorated by abstractmethod.**

**A Rectangle class now uses above Shape class as its parent and implementing the abstract area() method. Since it is a concrete class, it can be instantiated and imlemented area() method can be called.**

**2. What happens when a class statement's top level contains a basic assignment statement?**

**ANS:-**

**When class statement's top level contains a basic assignment statement, it is considered as class attribute. Change in the value of class attribute will affect all the instances of the class.**

**class Sample:**

**some\_value = 1234**

**def \_\_init\_\_(self, value1):**

**self.value1 = value1**

**s = Sample(2)**

**s1 = Sample(3)**

**print(s.some\_value)**

**print(s.value1)**

**print(s1.value1)**

**Sample.some\_value = 567**

**print(s.some\_value) *#observe value is changed for both the objects***

**print(s1.some\_value)**

**1234**

**2**

**3**

**567**

**567**

**3. Why does a class need to manually call a superclass's \_\_init\_\_ method?**

**ANS:-**

**By doing so,we can access those methods of the super-class (parent class) which have been overridden in a sub-class (child class) that inherits from it.**

**4. How can you augment, instead of completely replacing, an inherited method?**

**ANS:-**

**The way to do that in Python is by calling to the original version directly, with augmented arguments.**

**5. How is the local scope of a class different from that of a function?**

**ANS:-**

**In class, if the variable is declared without self then it is accessible within that function only, kinda local variable. However if it declared using self like self.variable\_name = 'somevalue', then it is accessible via any object but not via the class name.**

**Whereas, if a variable is declared within a function then it is a local variable and is accessible to that function only.**