1. What is the concept of an abstract superclass?

Ans:

Abstract superclasses are used to define a common interface for a group of related classes, while allowing each subclass to provide its own implementation of the interface. This can make it easier to write and maintain code that works with a set of related objects, since you can rely on the fact that each object will provide a certain set of methods with a certain interface.

In Python, you can define an abstract superclass by using the abc module. To create an abstract superclass, you define a class that inherits from the ABC class provided by the abc module, and you decorate any abstract methods with the @abstractmethod decorator.

Ex:

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

@abstractmethod

def perimeter(self):

pass

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

Ans:

When a class statement's top level contains a basic assignment statement, the assignment statement creates a class-level variable or class attribute.

Class-level variables are shared by all instances of the class, and can be accessed using either the class name or an instance of the class.

For example:

class MyClass:

class\_var = 10

def \_\_init\_\_(self, instance\_var):

self.instance\_var = instance\_var

my\_obj1 = MyClass(20)

my\_obj2 = MyClass(30)

print(MyClass.class\_var) # Output: 10

print(my\_obj1.class\_var) # Output: 10

print(my\_obj2.class\_var) # Output: 10

In this example, the class\_var variable is defined at the class level and is shared by all instances of the MyClass class. When we access class\_var using the class name or an instance of the class, we get the same value (10) in all cases.

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

Ans:

A class needs to manually call a superclass's \_\_init\_\_ method in order to properly initialize the superclass's state and behavior. When a subclass inherits from a superclass, it can override methods or attributes defined in the superclass, but it still needs to ensure that the superclass is properly initialized.

By convention, the \_\_init\_\_ method of a subclass should begin by calling the \_\_init\_\_ method of the superclass. This allows the superclass to perform any necessary initialization, such as setting default attribute values or registering the subclass with a global registry.

Ex:

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

self.breed = breed

super().\_\_init\_\_(name)

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

Ans:

There are two ways to call a superclass's method from a subclass:

1. Call the method directly on the superclass, passing in self as the first argument:

2. Use the super function to get a reference to the superclass's method and call it explicitly

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

Ans:

1. Access to variables: In a class, variables declared at the class level are in the local scope of the class and can be accessed from any method of the class. In contrast, variables declared within a function are in the local scope of that function and can only be accessed within that function.

2. Persistence of variables: In a class, variables declared at the class level persist across method calls and are shared by all instances of the class. In contrast, variables declared within a function are local to that function and are destroyed when the function exits.

3. Method access: In a class, methods defined within the class can access variables in the class's local scope. In contrast, functions defined outside of a class cannot access variables in the class's local scope unless they are passed in as arguments.

4. Namespace: Each class has its own namespace, which is separate from the global namespace and any other class namespaces. In contrast, functions do not have their own namespace, but rather access variables in the global namespace and the local scope.

Overall, the local scope of a class is more flexible and persistent than the local scope of a function, since it allows variables to be shared across methods and instances, and accessed by other methods within the class.