**Q1. What is the concept of an abstract superclass?**

Ans-An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class

**Q2. What happens when a class statement’s top level contains a basic assignment statement?**

Ans-In Python, when a class statement's top level contains a basic assignment statement, it creates a class variable that is shared among all instances (objects) of that class. A class variable is a variable that belongs to the class itself and not to any specific instance of the class. This means that the variable will have the same value across all instances of the class.

It's important to understand this distinction between class variables and instance variables to avoid unexpected behavior when working with Python classes. Class variables are useful when you want to store data that is common to all instances of the class.

**Q3. Why does a class need to manually call a superclass’s \_\_init\_\_ method?**

Ans-In Python, a class needs to manually call a superclass's **\_\_init\_\_** method to properly initialize the superclass's attributes and perform any necessary setup that is defined in the superclass's constructor. This is particularly important when the subclass overrides the **\_\_init\_\_** method of the superclass.

When a subclass defines its own **\_\_init\_\_** method, it will override the **\_\_init\_\_** method of the superclass. This means that the superclass's **\_\_init\_\_** method won't be automatically called when you create an instance of the subclass.

To ensure that the superclass's initialization is executed and its attributes are properly set up, you need to explicitly call the superclass's **\_\_init\_\_** method from within the subclass's **\_\_init\_\_** method using the **super()** function. The **super()** function returns a temporary object of the superclass, which allows you to access its methods.

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

Ans-To augment, rather than completely replace, an inherited method in Python, you can follow a few steps to achieve method overriding while still leveraging the functionality of the superclass's method. Method overriding allows a subclass to provide a specific implementation for a method that is already defined in its superclass.

Here's the typical process to augment an inherited method:

Define the subclass, making sure it inherits from the superclass.

In the subclass, define a method with the same name as the method you want to augment from the superclass. This is known as method overriding.Inside the overridden method, you can choose to call the superclass's method using **super()** along with the method name. This way, you can execute the superclass's method before or after performing any additional operations.

**Q5. How is the local scope of a class different from that of a function?**

Ans-The local scope of a class and that of a function are different in terms of the variables they contain and how they are accessed within the context of their respective structures.

**Local Scope in a Function:** In a function, the local scope refers to the area within the function where variables are defined and accessible. Variables defined inside a function are called local variables, and they are limited in scope to that particular function. These variables can only be accessed within the function and are not visible outside of it. Once the function finishes executing, the local variables are destroyed, and their values are no longer available.

**Local Scope in a Class:** In a class, the local scope refers to the area within the methods (functions) of the class where variables are defined and accessible. Variables defined inside a method are also known as local variables and are limited in scope to that specific method. Similar to local variables in a function, local variables in a class can only be accessed within the method in which they are defined. They are not visible in other methods of the class or outside the class.

**Key Differences:**

**Scope Hierarchy:** The local scope of a function is within the function itself, while the local scope of a class is within each method of the class.

**Accessibility:** Local variables in a function are not visible outside the function, and local variables in a method are not visible in other methods or outside the class.

**Lifetime:** The lifetime of local variables in both cases is limited to the execution duration of the function or method. Once the function or method finishes executing, the local variables are destroyed.

It's essential to understand these differences when working with functions and classes to manage variable scope effectively and avoid potential naming conflicts or unintended behavior.