1. What is the concept of an abstract superclass?

Ans. An abstract superclass, also known as an abstract base class (ABC), is a class in object-oriented programming that is designed to serve as a blueprint or template for other classes, rather than being instantiated directly. An abstract superclass defines methods that must be implemented by its subclasses, but it may also provide default implementations for some methods.

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, it defines a class-level attribute rather than a local variable within a method. This means that the assigned value becomes an attribute of the class itself, rather than being specific to instances of the class.

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

Ans. when a subclass defines its own \_\_init\_\_ method, it does not automatically call the \_\_init\_\_ method of its superclass. Instead, the subclass is responsible for explicitly calling the \_\_init\_\_ method of its superclass if it wants to initialize the inherited attributes and perform any additional initialization defined in the superclass.

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

Ans. You can augment, rather than completely replace, an inherited method by calling the superclass's method from within the subclass's method and then adding or modifying functionality before or after the call. This process is known as method overriding.

5. 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 differ in several key ways:

Access to Variables:

In a function, variables defined within the function's scope are considered local variables and are accessible only within that function.

In a class, variables defined within methods are considered local to those methods and are accessible only within the scope of the method in which they are defined. However, variables defined at the class level (outside of any method) are accessible throughout the class.

Visibility:

In a function, local variables are visible and accessible only within the function's body. They cannot be accessed from outside the function.

In a class, local variables defined within a method are visible and accessible only within that method. They cannot be accessed from outside the method. However, instance variables (attributes) defined within a method using self are accessible from other methods of the class.

Lifetime:

The lifetime of local variables in a function is limited to the duration of the function call. They are created when the function is called and destroyed when the function returns.

In a class, the lifetime of local variables (defined within methods) is limited to the duration of the method invocation. They are created when the method is called and destroyed when the method returns. Instance variables (defined with self) persist as long as the instance exists.

Scoping Rules:

Functions have function-level scope, meaning that variables defined within a function are visible only within that function.

Classes have method-level scope, meaning that variables defined within a method are visible only within that method. Additionally, class-level variables (defined outside methods) are accessible throughout the class.

Namespace:

Functions have their own local namespace, where local variables are stored during the function's execution.

Classes have their own namespace, where instance variables and class variables are stored. Each instance of a class also has its own namespace.