1. What is the concept of an abstract superclass?

ANSWER.

An abstract superclass, also known as an abstract base class (ABC), is a class in object-oriented programming that is designed to be subclassed but not instantiated directly. It serves as a blueprint or template for creating other classes, providing a common interface or set of methods that its subclasses must implement.

In Python, abstract base classes are typically created using the `abc` module. This module provides the `ABC` class as a base class for defining abstract base classes, as well as the `abstractmethod` decorator for marking methods as abstract.

Key concepts related to abstract superclasses include:

1. Abstract Methods: An abstract method is a method declared in an abstract superclass that does not have an implementation. Subclasses of the abstract superclass must provide implementations for all abstract methods, or else they will also be considered abstract and cannot be instantiated.

2. Forcing Subclass Implementation: Abstract superclasses help enforce a contract between the superclass and its subclasses, ensuring that subclasses implement certain methods with specific signatures. This promotes code consistency and helps prevent errors by guaranteeing that subclasses adhere to a common interface.

3. Partial Implementation: Abstract superclasses can also provide concrete implementations for certain methods while leaving others abstract. This allows for a combination of shared behavior across subclasses and subclass-specific implementations.

4. Code Organization: Abstract superclasses help organize code by grouping related classes together and defining a common interface for them. This promotes code reuse and simplifies maintenance by encapsulating shared functionality in one place.

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

ANSWER.

When a class statement's top level contains a basic assignment statement, such as assigning a value to a variable without being inside any method or nested block, Python treats it as a class attribute.

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

ANSWER.

In many object-oriented programming languages, including Python, a class needs to manually call its superclass's `\_\_init\_\_` method if it wants to initialize attributes or perform other setup tasks defined in the superclass's constructor. This is because when you define a constructor (`\_\_init\_\_` method) in a subclass, it overrides the constructor of the superclass, and the superclass's constructor is not automatically called.

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

ANSWER.

You can augment an inherited method in Python by calling the superclass's method from within the subclass's method and then adding additional functionality before or after the call. This process is known as method overriding with a call to the superclass's method.

Here's how you can augment an inherited method in Python:

1. Override the Method: Define a method with the same name as the method you want to augment in the superclass.

2. Call the Superclass's Method: Inside the subclass's method, call the superclass's method using the `super()` function.

3. Add Additional Functionality: Add any additional functionality before or after the call to the superclass's method.

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

ANSWER.

The local scope of a class and that of a function in Python are different in several ways:

1. Namespace:

- In a function, the local scope is created when the function is called, and it exists only within the function body. Variables defined within the function are local to that function and are not accessible outside of it.

- In a class, the local scope is defined within methods and constructors. Variables defined within methods are local to those methods and are not accessible outside of them. However, class attributes (variables defined directly within the class body) are accessible from within any method of the class.

2. Lifetime:

- The local scope of a function exists only for the duration of the function call. Once the function completes execution, its local variables are destroyed, and the local scope is discarded.

- The local scope of a class exists as long as the instance of the class exists. It is created when an instance of the class is created and destroyed when the instance is garbage collected. However, class attributes persist for the entire lifetime of the class.

3. Access to Attributes:

- In a function, local variables are accessible only within the function body. They cannot be accessed outside of the function.

- In a class, local variables (defined within methods) are accessible only within the method where they are defined. However, class attributes (defined directly within the class body) are accessible from any method of the class.