1. What is the concept of an abstract superclass?

Answer :-

The concept of an abstract superclass in object-oriented programming refers to a class that is designed to be inherited from but cannot be instantiated on its own. It serves as a blueprint or template for subclasses, providing common attributes and methods that can be shared among multiple related classes.

In Python, abstract superclasses are created using the abc module, which stands for Abstract Base Classes. Abstract base classes are defined by inheriting from the ABC class provided by the abc module and using the @abstractmethod decorator to mark specific methods as abstract

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

@abstractmethod

def perimeter(self):

pass

class Circle(Shape):

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

self.radius = radius

def area(self):

return 3.14 \* self.radius \* self.radius

def perimeter(self):

return 2 \* 3.14 \* self.radius

circle = Circle(5)

print(circle.area()) # Output: 78.5

print(circle.perimeter()) # Output: 31.4

1. 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, it creates a class-level attribute that is shared among all instances of the class.

class MyClass:

class\_attribute = 10

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

self.instance\_attribute = instance\_attribute

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

Answer :-

A class needs to manually call a superclass's \_\_init\_\_ method when it wants to ensure that the initialization code from the superclass is executed before executing its own initialization code.

In object-oriented programming, inheritance allows a class (subclass) to inherit attributes and methods from another class (superclass). When a subclass is instantiated, it typically needs to perform some initialization specific to itself, but it also may require the initialization code of its superclass to properly set up the inherited attributes or perform other necessary operations.

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

Answer:-

Here are the steps to augment an inherited method:

1. Define the method in the subclass with the same name as the method in the superclass.
2. Within the subclass method, you can call the superclass's method using the super() function. This allows you to execute the superclass's implementation before or after adding your own code.
3. Add the specific code in the subclass method to augment or extend the behavior of the inherited method.

class Superclass:

def method(self):

print("Superclass method")

class Subclass(Superclass):

def method(self):

super().method() # Call superclass's method

print("Subclass method")

my\_instance = Subclass()

my\_instance.method()

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

Answer:-

The local scope of a class is used to define and access class attributes and methods throughout the class and its instances. It exists as long as the class remains in memory.

The local scope of a function is used to define and access variables specific to that function during its execution. It exists only during the execution of the function.