1. What is the concept of an abstract superclass?

1. Abstract Super Class. A common superclass for several subclasses. Factor up common behavior. Define the methods they all respond to. Methods that subclasses should implement are declared abstract.

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

A) top level statement and classical way both in your code. You can follow the approach suggested by him or just remove the below part from your code. And closing } of Program class and Main method .

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

A) In old-style classes, you could only call the super init if the super class actually had an init defined (which it often doesn't). Therefore, people typically think about calling super method, rather than doing it out of principle.

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

A)

In object-oriented programming, if you want to augment or extend the functionality of an inherited method without completely replacing it, you can achieve this through method overriding and calling the parent method within the child class.

class Parent:

def method(self):

print("Parent method")

class Child(Parent):

def method(self):

# Augmenting the functionality

print("Child method")

# Calling the parent method

super().method()

# Creating an instance of Child

child\_obj = Child()

# Calling the overridden method

child\_obj.method()

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

A) Local Scope of a Function:

Encapsulation: In a function, variables defined within it are accessible only within that function, unless explicitly passed or returned.

Lifetime: Variables declared within a function have a lifetime that spans the execution of the function. They are created when the function is called and destroyed when the function exits.

Access: Functions can access variables from the enclosing scope if they are not redefined within the function.

Local Scope of a Class:

Encapsulation: In a class, variables defined within it (attributes or instance variables) are accessible throughout the class methods via the self (Python) or this (Java) reference, and usually can be accessed outside the class instance through dot notation.

Lifetime: Class variables exist as long as the object instance to which they belong exists. They are created when an instance of the class is created (\_\_init\_\_ method in Python) and destroyed when the object is garbage collected.

Access: Class methods (functions defined within a class) can access class variables (defined within the class but outside any method) directly using the self or this reference. Class variables are shared among all instances of the class.