1. What is the concept of an abstract superclass?

Ans: - An abstract superclass in Python is a class that cannot be instantiated and is meant to be subclassed by other classes1. It allows you to define a set of methods that must be created within any child classes built from the abstract class1. A class that contains one or more abstract methods is called an abstract class.

```
from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):
    def noofsides(self):
        print("I have 5 sides")

R = Triangle()
R.noofsides()  # Outputs: I have 3 sides
```

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 creates a class attribute. This attribute is shared by all instances of the class.

```
class MyClass:
    my_class_attribute = "This is a class attribute"

print(MyClass.my_class_attribute)  # Outputs: This is a class attribute
```

3. Why does a class need to manually call a superclass's __init__ method?

Ans: - A class needs to manually call a superclass's __init__ method to ensure that the initialization code in the superclass gets executed3. This is important because the superclass's __init__ method might set up variables or state that is necessary for the methods in the superclass.

```
class Base:
    def __init__(self):
        print("Base created")

class Child(Base):
    def __init__(self):
        Base.__init__(self)

c = Child()  # Outputs: Base created
```

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

Ans: - You can augment an inherited method by calling the method from the superclass within the overridden method in the subclass. This allows you to add additional functionality in the subclass's method without losing the behavior defined in the superclass's method.

```
class Parent:
    def inherited_method(self):
        print("This is the inherited method from the parent class.")

class Child(Parent):
    def inherited_method(self):
        # Call the original inherited method
        super().inherited_method()
        # Add additional functionality
        print("This is the additional functionality in the child class.")

c = Child()
c.inherited_method()
# Outputs:
# This is the inherited method from the parent class.
# This is the additional functionality in the child class.
```

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

Ans: - In Python, the local scope of a function includes all variables defined within the function. These variables are only accessible within the function. On the other hand, the local scope of a class includes all variables and methods defined within the class. These can be accessed by all methods within the class and by instances of the class.