Python Advanced Assignment-3

1. What is the concept of an abstract superclass?

Ans.

An abstract superclass is a class that is not meant to be instantiated directly. It serves as a template for other classes and may contain abstract methods that must be implemented by subclasses. In Python, you can define an abstract superclass using the 'abc' module:

```
from abc import ABC, abstractmethod


class AbstractClass(ABC):
    @abstractmethod
    def some_method(self):
        pass
```

Subclasses of this abstract class must implement the `some_method` method; otherwise, they cannot be instantiated.

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:
    class_attribute = 10
```

'class_attribute' is accessible through the class itself ('MyClass.class_attribute') and all instances of the class.

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 superclass is properly initialized. If not called explicitly, the initialization code in the superclass might be skipped, and any attributes or setup defined there won't be available to the subclass.

```
class SuperClass:
    def __init__(self):
```

```
        self.value = 10


class SubClass(SuperClass):
    def __init__(self):
        super().__init__()  # Explicitly calling the superclass's __init__
        self.new_value = 20
```

Using `super()` ensures that the superclass's initialization logic is executed.

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

Ans.

To augment an inherited method, you can call the superclass's version of the method within the subclass method using `super()`, then add additional functionality.

```
class SuperClass:
    def method(self):
        print("SuperClass method")


class SubClass(SuperClass):
    def method(self):
        super().method()  # Call the superclass method
        print("SubClass additional functionality")
```

This way, the superclass method is extended rather than replaced entirely.

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

Ans.

The local scope of a class is different because it does not create a separate namespace for each instance. Class variables (attributes defined at the class level) are shared among all instances, while instance variables (attributes defined within methods using `self`) are unique to each instance.In contrast, the local scope of a function is created anew each time the function is called, and its variables are local to the function call.