Python OOP Questions & Answers (Set 3)

Q1. What is the concept of an abstract superclass?

An abstract superclass defines a common interface and shared behavior but is not meant to be instantiated directly.
It serves as a blueprint for subclasses.
In Python, abstract classes are defined using the abc module:

```
from abc import ABC, abstractmethod

class Shape(ABC):
@abstractmethod
def area(self):
pass
```

Any subclass must implement the abstract methods before instantiation.

Q2. What happens when a class statement's top level contains a basic assignment statement?

Any assignment inside the class body (but outside methods) creates a class attribute.
Example:

```
class Car:
wheels = 4 # class attribute
```

All instances of the class share this attribute unless shadowed by an instance attribute.

Q3. Why does a class need to manually call a superclass's __init__ method?

Python does not automatically call superclass __init__ methods when subclassing.
If the subclass defines its own __init__, the base class's initializer must be explicitly invoked using super() or by calling the base class directly.
Example:

```
class A:
def __init__(self):
print('A init')

class B(A):
def __init__(self):
super().__init__()
print('B init')
```

Q4. How can you augment, instead of completely replacing, an inherited method?

Use super() (or explicit base class call) to extend behavior instead of replacing it.
Example:

```
class Parent:
```

```
def greet(self):
print('Hello from Parent')

class Child(Parent):
def greet(self):
super().greet()
print('Hello from Child')
```

This way, both parent and child behaviors run.

Q5. How is the local scope of a class different from that of a function?

Function scope: Local variables are created dynamically when the function is called, and disappear when it returns.
Class scope: The class body executes once when the class is defined. The resulting namespace becomes the class's attribute dictionary.
- Variables assigned at the top level → class attributes.
- Methods defined inside → functions that get turned into methods.

Thus, class scope is persistent (attributes live as long as the class), unlike a function's ephemeral scope.