1. What is the concept of an abstract superclass?

An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes :

By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses. This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

How Abstract Base classes work :

By default, Python does not provide abstract classes. Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC. ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword @abstractmethod.

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

When the class statement’s top level contains a basic assignment statement they are called class attributes. Class attributes are the variables defined directly in the class that are shared by all objects of the class. Class attributes can be accessed using the class name as well as using the objects.

class Student:

schoolName = 'XYZ School'

Above, the schoolName is a class attribute defined inside a class. The value of the schoolName will remain the same for all the objects unless modified explicitly.

>>> Student.schoolName

'XYZ School'

>>> std = Student()

>>> std.schoolName

'XYZ School'

As you can see, a class attribute is accessed by Student.schoolName as well as std.schoolName. Changing the value of class attribute using the class name would change it across all instances. However, changing class attribute value using instance will not reflect to other instances or class.

>>> Student.schoolName = 'ABC School' # change attribute value using class name

>>> std = Student()

>>> std.schoolName

'ABC School' # value changed for all instances

>>> std.schoolName = 'My School' # changing instance's attribute

>>> std.schoolName

'My School'

>>> Student.schoolName # instance level change not reflectd to class attribute

'ABC School'

>>> std2 = Student()

>>> std2.schoolName

'ABC School'

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

The main reason for always calling base class \_init\_\_ is that base class may typically create member variable and initialize them to defaults. So if you don't call base class init, none of that code would be executed and you would end up with base class that has no member variables.

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

Each method that is inherited from a superclass can be augmented to perform some different action in the new class. If there is no augmentation then the inherited methods will perform as defined in the superclass.

An inherited method is augmented simply by creating a new definition for the method in your class definition.

A more sophisticated way to augment an inherited method involves forwarding. Message forwarding allows you to augment an inherited method in such a way that it can perform its inherited action and some new action.

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

Declaring a variable in a class (outside of a function): all class functions can access it (basically a public variable). This is like a static variable and can be called using the class name. These variables are available to all functions, any functions can modify it and print it.

Declaring a variable inside a function inside a class: only that function can access it (it's in that function's scope). If the variable is declared without self then it is accessible within that function only, like a local variable. However, if it was declared using self like self.var= 'somevalue', then it is accessible via any object but not via the class name.

Declaring a variable at the top level of the class is like declaring a static or class variable. Qualifying it with self is declaring an instance variable. Class variables can be modified by referring to them by class name (e.g. Class.x = 5) and all instances will inherit these changes. Instance variables are private to an instance and can only be modified by that instance.

You can achieve some level of access control using underscores. See private variables in the Python tutorial. By convention, variables starting with one underscore, e.g. \_foo are non-public parts of an API, and names starting with two underscores e.g. \_\_foo will have it's name mangled to be \_classname\_\_foo.