1. What is the concept of an abstract superclass?

Ans. An abstract superclass is one way to provide re-usable code. You can extend the abstract class and inherit the code. This is sometimes more convenient than using static methods or object composition to share code. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and its abstract methods must be implemented by its subclasses.

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 is usually treated as a class attribute or class level variable, whereas assignment statements inside methods are treated as instance attributes or local attributes.

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

Ans. 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. The \_\_init\_\_ method lets the class initialize the object's attributes and serves no other purpose.

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

Ans. To augment instead of completely replacing an inherited method, redefine it in a subclass, but call back to the superclass’s version of the method manually from the new version of the method in the subclass. That is, pass the self instance to the superclass’s version of the method manually, ex: Superclass.method(self, ...).

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

Ans. Declaring a variable in a class (outside of a function): all class functions can access it (basically a public variable)

Declaring a variable inside a function inside a class: only that function can access it (it's in that function's scope)

Declaring a variable with self.(variable name) inside a function inside a class: all class functions can access it

And since there is no private/protected, everything is public, so everything accessible from inside a class is accessible from outside the class.