1. What is the concept of an abstract superclass?

Sol. An abstract superclass is a class that calls a method, but does not inherit or define

it—it expects the method to be filled in by a subclass. This is often used as a way to generalize classes when behavior cannot be predicted until a more specific subclass is coded. OOP frameworks also use this as a way to dispatch to client-defined, customizable operations.

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

Sol. When a simple assignment statement (X = Y) appears at the top level of a class statement, it attaches a data attribute to the class (*Class*.X). Like all class attributes, this will be shared by all instances; data attributes are not callable method functions,though

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

Sol. A class must manually call the \_\_init\_\_ method in a superclass if it defines an \_\_init\_\_ constructor of its own, but it also must still kick off the superclass’s construction code. Python itself automatically runs just one constructor—the lowest one in the tree. Superclass constructors are called through the class name, passing in the self instance manually: *Superclass*.\_\_init\_\_(self, ...).

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

Sol. To augment instead of completely replacing an inherited method, redefine it in asubclass, 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: *Superclass*.*method*(self, ...).

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

Sol.

.