1. What is the concept of an abstract superclass?

Answer : We define a superclass or an abstract superclass with the aim that it should serve as a definition and the subclasses inheriting it will implement this definition to make something significant. It allows to create set of methods that must be created within any child classes built from abstract class.

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

Answer : When a class statement's top level contains a basic assignment statement, then assignment variable is called a class attribute.

E.g.

```
class Person:
        kind= 'homo_sapiens' #this is class attribute
        def __init__(self, name, surname, year_of_birth):
                self.name = name
                self.surname = surname
                self.year_of_birth = year_of_birth
```

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

Answer : If a subclass has the __init__method, then it will not inherit the__init__method of the superclass. In other words, the __init__method of the subclass overrides the __init__ method of the superclass. So we need to manually call a superclass's __init__method.

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

Answer: super() is used to augment, instead of completely replacing, an inherited method.

E.g.

```
class Person:
        def __init__(self, name, surname, year_of_birth):
                self._name = name
                 self._surname = surname
                self._year_of_birth = year_of_birth

        def age(self, current_year):
                return current_year - self._year_of_birth

        def __str__(self):
                return "%s %s and was born %d." \% (self._name, self._surname,
        self._year_of_birth)

class Student(Person):
        def __init__(self, student_id, *args, **kwargs):
                super(Student, self).__init__(*args, **kwargs)
                self._student_id = student_id
        def __str__(self):
                return super(Student, self).__str__() + " And has ID: %d" % self._student_id
```

We defined __str__ again augmenting the one wrote in Person, but we wanted to extend it, so we used super to achieve our goal.

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

Answer : A variable which is defined inside a function is local to that function. It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.

The inside of a class body is also a local variable scope. Variables which are defined in the class body (but outside any class method) are called class attributes. They can be referenced by their bare names within the same scope, but they can also be accessed from outside this scope if we use the attribute access operator (.) on a class or an instance (an object which uses that class as its type).