1. An abstract class/superclass 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.

2. 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.

Example: **class** Person:

species **=** 'Homosapiens' *# class attribute*

**def** \_\_init\_\_(self, name, gender):

self**.**name **=** name *# instance attributes*

self**.**gender **=** gender

3. If a child class has **\_\_init\_\_** method, then it will not inherit the **\_\_init\_\_** method of the parent class. In other words the **\_\_init\_\_** method of the child class overrides the **\_\_init\_\_** method of the parent class. So we have to manually call a parent superclass’s \_\_init\_\_ using super() method.

4. **super()** method can be used to augment, instead of completely replacing, an inherited method.

Example: **class** Person:

**def** \_\_init\_\_(self, name, gender):

self**.**name **=** name

self**.**gender **=** gender

**class** Employee(Person):

**def** \_\_init\_\_(self, name, gender, salary):

super()**.**\_\_init\_\_(name, gender)

self**.**salary **=** salary

emp\_1 **=** Employee('Ram','Male',20000)

print(emp\_1**.**\_\_dict\_\_)

Output: {'name': 'Ram', 'gender': 'Male', 'salary': 20000}

5. 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 existing.

Similarly a variable inside of a class also has a local variable scope. Variables which are defined in the class body (but outside all methods) are called as class level variables or 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 of the class.

Example: **def** hello(name):

name **=** name

print(f' your name is {name}')

hello('Ram')

**try**:

name

**except** NameError:

print('Name variable is not available outside hello function scope')

**class** Person:

species **=** "HomoSapiens"

**def** \_\_init\_\_(self):

**pass**

print(Person**.**species) *# Accessing species using class name*

Male **=** Person()

print(Male**.**species) *# Accessing species using instance of class*

Output: your name is Ram

Name variable is not available outside hello function scope

HomoSapiens

HomoSapiens