**Advance Assignment - 03**

1. What is the concept of an abstract superclass?

**Answer :**

* 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.
* The abstract class can "fix" parts of the code (by making it final). This is called the "template method" pattern (and this is not possible with an interface, which cannot provide final methods).
* Of course, you can achieve both with a non-abstract superclass as well.
* An abstract class has the additional benefit that it does not have to provide a complete implementation (that would make sense to instantiate on its own), some parts can be left specified, but unimplemented (the abstract methods).

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

**Answer :**

* Assignment statements are used to (re)bind names to values and to modify attributes or items of mutable objects.
* An assignment statement evaluates the expression list and assigns the single resulting object to each of the target lists, from left to right.
* Assignment creates object references instead of copying the objects.

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

**Answer :**

The main reason for always calling superclass’s \_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?

**Answer :**

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?

**Answer :**

* 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 (how is this different from global (variable name)?)
* And since there is no private/protected, everything is public, so everything accessible from inside a class is accessible from outside the class.