**Python Advance Assignment 2**

**Q1. What is the relationship between classes and modules?**

The difference between a class and a module in python is that a class is used to define a blueprint for a given object, whereas a module is used to reuse a given piece of code inside another program.

**Q2. How do you make instances and classes?**

There are two kinds of objects in Python's OOP model: class objects and instance objects, which is quite different from other OOP languages.

Class objects provide default behavior and serve as factories for generating instance objects. Instance objects are the real objects created by your application. An instance object has its own namespace. It copies all the names from the class object from which it was created.

The class statement creates a class object of the given class name. Within the class definition, you can create class variables via assignment statements, which are shared by all the instances. You can also define methods, via the defs, to be shared by all the instances.

When an instance is created, a new namespace is created, which is initially empty. It clones the class object and attaches all the class attributes. The \_\_init\_\_() is then invoked to create (initialize) instance variables, which are only available to this particular instance.

**Q3. Where and how should be class attributes created?**

A class attribute is a Python variable that belongs to a class rather than a particular object. It is shared between all the objects of this class and it is defined outside the constructor function, \_\_init\_\_(self,...), of the class.

**Q4. Where and how are instance attributes created?**

An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object and it is defined inside the constructor function, \_\_init\_\_(self,..) of the class.

**Q5. What does the term “self” in a Python class mean?**

“self” represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.  
The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

**Q6. How does a Python class handle operator overloading?**

[Python operators](https://www.programiz.com/python-programming/operators) work for built-in classes. But the same operator behaves differently with different types.Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

**Q7. When do you consider allowing operator overloading of your classes?**

Suppose the user has two objects which are the physical representation of a user-defined data type class. The user has to add two objects using the "+" operator, and it gives an error. This is because the compiler does not know how to add two objects. So, the user has to define the function for using the operator, and that process is known as "operator overloading". The user can overload all the existing operators by they cannot create any new operator. Python provides some special functions, or we can say magic functions for performing operator overloading, which is automatically invoked when it is associated with that operator. Such as, when the user uses the "+" operator, the magic function \_\_add\_\_ will automatically invoke in the command where the "+" operator will be defined.

**Q8. What is the most popular form of operator overloading?**

A very popular and convenient example is the **Addition (+) operator**.

Just think how the ‘+’ operator operates on two numbers and the same operator operates on two strings. It performs **“Addition”** on numbers whereas it performs **“Concatenation”** on strings. Operators in Python work for **built-in** classes, like **int**, **str**, **list**, etc. But you can extend their **operability** such that they work on objects of user-defined classes too.

**Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?**

In order to develop robust and well-designed software products with Python, it is essential to obtain a comprehensive understanding of OOP. Two key concepts of OOP are inheritance and polymorphism. Both inheritance and polymorphism are key ingredients for designing robust, flexible, and easy-to-maintain software.