Q1. What is the relationship between classes and modules?

A module in python is simply a way to organize the code, and it contains either python classes or just functions. If you need those classes or functions in your project, you just import them. For instance, the math module in python contains just a bunch of functions, and you just call those needed ( math. sin ).

Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Q2. How do you make instances and classes?

To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts.

In Python, a class can be created by using the keyword class, followed by the class name. each class is associated with a documentation string which can be accessed by using <class-name>.\_\_doc\_\_. A class contains a statement suite including fields, constructor, function, etc. definition. The self is used as a reference variable, which refers to the current class object. It is always the first argument in the function definition. However, using self is optional in the function call. The self-parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class.

Q3. Where and how should be class attributes created?

Class attributes are the variables defined directly in the class that are shared by all objects of the class. They are accessed using class name as well as using object with dot notation, e.g. classname.class\_attribute or object.class\_attribute. Changing value by using classname.class\_attribute = value will be reflected to all the objects. The following example demonstrates the use of class attribute count.

class Student:

count = 0

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

Student.count += 1

count is an attribute in the Student class. Whenever a new object is created, the value of count is incremented by 1. You can now access the count attribute after creating the objects.

>>> std1=Student()

>>> Student.count

1

>>> std2 = Student()

>>> Student.count

2

Q4. Where and how are instance attributes created?

Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor. They are accessed using object dot notation e.g. object.instance\_attribute. Changing value of instance attribute will not be reflected to other objects. The following demonstrates the instance attributes.

class Student:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Now, you can specify the values while creating an instance, as shown below.

>>> std = Student('Bill',25)

>>> std.name

'Bill'

>>> std.age

25

Q5. What does the term "self" in a Python class mean?

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

Q6. How does a Python class handle operator overloading?

The operator overloading in Python means provide extended meaning beyond their predefined operational meaning. Such as, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. We can achieve this as the "+" operator is overloaded by the "int" class and "str" class. This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.

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

Operator overloading is mostly useful when you're making a new class that falls into an existing "Abstract Base Class" (ABC) -- indeed, many of the ABCs in standard library module collections rely on the presence of certain special methods (and special methods, one with names starting and ending with double underscores

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.

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

Both inheritance and polymorphism are fundamental concepts of object oriented programming. These concepts help us to create code that can be extended and easily maintainable.