Q1. What is the relationship between classes and modules?

**ANS**

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.

A class can have its own instance, but a module cannot be instantiated. We use the ‘class’ keyword to define a class, whereas to use modules, we use the ‘import’ keyword. We can inherit a particular class and modify it using inheritance. But while using modules, it is simply a code containing variables, functions, and classes.

Modules are files present inside a package, whereas a class is used to encapsulate data and functions together inside the same unit.

Q2. How do you make instances and classes?

**ANS**

The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon as follows −

class ClassName:

'Optional class documentation string'

class\_suite

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

Q3. Where and how should be class attributes created?

**ANS**

Class attributes are attributes which are owned by the class itself. They will be shared by all the instances of the class. Therefore they have the same value for every instance. We define class attributes outside all the methods, usually they are placed at the top, right below the class header.

Q4. Where and how are instance attributes created?

**ANS**

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?

**ANS**

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.

Q6. How does a Python class handle operator overloading?

**ANS**

Python operators work for built-in classes. But the same operator behaves differently with different types. For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

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?

**ANS**

The classic operator-overloading example in Python is the plus sign, a binary (i.e., two operands) operator that not only adds a pair of numbers, but also concatenates a pair of lists or strings. The asterisk is similarly overloaded as not only a multiplier for numbers, but also as a repetition operator for lists or strings. Comparison operators (such as >, ==, or !=) exhibit similar behavior

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

**ANS**

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?

**ANS**

In order to develop robust and well-designed software products with Python, it is essential to obtain a comprehensive understanding of OOP. In this article, we will elaborate on two key concepts of OOP which are inheritance and polymorphism.

Both inheritance and polymorphism are key ingredients for designing robust, flexible, and easy-to-maintain software