Q1. What is the relationship between classes and modules?

Ans: Classes are blueprints for creating objects with attributes (data) and methods (functions).

Modules are files containing Python code, including functions, variables, and classes.

The relationship: You can define classes within modules, and modules can contain classes along with other code elements. To use a class from a module, you import the module into your code. This relationship helps in organizing and reusing code efficiently.

Q2. How do you make instances and classes?

Ans: To create instances and classes, you need to define a class. Using ‘class’ keyword followed by the class name you can define it. Inside the class, you can describe the behaviour of objects created from the class with the help of attributes(data) and methods(functions).

To create an instance( an object) of a class, call the class as if it were a function and assign the result to a variable. You can also access class members using the dot notation.

Q3. Where and how should be class attributes created?

Ans: Class attributes are attributes that are associated with a class rather than with instances of the class. These attributes are shared among all instances of the class. You can create class attributes inside the class definition itself, typically outside of any methods.

Q4. Where and how are instance attributes created?

Ans: Instance attributes are attributes that are specific to individual instances (objects) of a class. They are created and defined within the class's methods, typically within the constructor method (\_\_init\_\_) or any other instance methods.

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

Ans: In a Python class, the term "self" refers to a convention that represents the instance of the class. It is the first parameter to instance methods in a class, including the constructor method \_\_init\_\_. The use of "self" is a naming convention, and it is not a keyword; you could technically use any other name, but it is strongly recommended to use "self" for clarity and consistency.

Q6. How does a Python class handle operator overloading?

Ans: In Python, operator overloading allows you to define custom behavior for built-in operators (e.g., +, -, \*, /, ==, <, etc.) when applied to instances of your custom classes. This enables you to make your objects work with operators in a way that makes sense for your class. Operator overloading is achieved by defining special methods within your class. These special methods have double underscores (e.g., \_\_add\_\_, \_\_sub\_\_) and are called "magic methods" or "dunder methods."

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

Ans: You should consider allowing operator overloading of your classes when it enhances the clarity, usability, and intuitiveness of your code. Operator overloading is a powerful feature in Python, but it should be used judiciously, and it makes sense in various situations, including semantic clarity, consistency with built-in types, reduced boilerplate, domain-specific language, existing conventions,etc.

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

Ans: The most popular form of operator overloading is likely related to numeric operations and sequences.( +, -, \*, /, ==, !=, <,<=,>,>=, Indexing and slicing, etc.

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

Ans: Classes and Objects, Inheritance and Polymorphism.