Q1. What is the relationship between classes and modules?

Classes and modules are both organizational units in Python. A module is a file containing Python code, while a class is a blueprint for creating objects. Classes can be defined within modules, and modules can contain multiple classes. Modules can also import classes defined in other modules.

Q2. How do you make instances and classes?

Instances of classes are created by calling the class name followed by parentheses, optionally passing any required arguments to the class's \_\_init\_\_ method. Classes themselves are created using the class keyword followed by the class name and a colon, with the class body containing attributes and methods.

Q3. Where and how should class attributes be created?

Class attributes should be created directly within the class body but outside of any method definitions. They are typically defined at the top of the class body and are accessible to all instances of the class.

Q4. Where and how are instance attributes created?

Instance attributes are created within the \_\_init\_\_ method of a class. They are assigned to self (the instance) using dot notation within the \_\_init\_\_ method. Instance attributes can also be created dynamically outside of \_\_init\_\_, but it's a common practice to initialize them in the constructor.

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

In Python classes, self refers to the instance of the class itself. It is a convention (not a keyword) used as the first parameter of instance methods. When you call an instance method, Python automatically passes the instance (self) as the first argument.

Q6. How does a Python class handle operator overloading?

Python classes can define special methods (e.g., \_\_add\_\_, \_\_sub\_\_, \_\_mul\_\_, etc.) to override the behavior of built-in operators. This process is known as operator overloading. When an operator is used with instances of a class that defines these special methods, Python invokes the appropriate method to perform the desired operation.

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

Operator overloading is considered when you want to provide intuitive and meaningful behavior for operators when applied to instances of your class. It can make code more readable and expressive, especially for mathematical or custom data types.

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

The most popular form of operator overloading in Python is arithmetic operator overloading, where classes define methods like \_\_add\_\_, \_\_sub\_\_, \_\_mul\_\_, etc., to override the behavior of arithmetic operators like +, -, \*, etc.

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

The two most important concepts in Python OOP are classes and objects (instances). Understanding how classes define the structure and behavior of objects, and how objects interact with each other and with the wider program, is essential for comprehending Python OOP code.