Q1. What is the relationship between classes and modules?

Classes are blueprints for creating objects with attributes and methods. They define the structure and behavior of objects within a program. Classes provide a way to encapsulate data and functionality into reusable components. Modules are files containing Python code, including functions, classes, and variables. They serve as a way to organize related code into separate files for better code organization and reusability. Modules can contain classes along with other code elements.

Q2. How do you make instances and classes?

Use the class keyword to define a class. Inside the class, you can define attributes (variables) and methods (functions) that describe the class's behavior.
To create instances (objects) of a class, call the class like a function. This invokes the class's __init__ method to initialize the instance.

Q3. Where and how should be class attributes created?

Class attributes are created within the class definition and are shared among all instances of that class. They are defined outside of any method and are usually placed at the top of the class definition.

Q4. Where and how are instance attributes created?

Instance attributes are created within the __init__ method of a class and are specific to each instance of that class. They are assigned using the self keyword within the constructor.

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

In Python, self is a conventionally used name for the first parameter of instance methods within a class. It refers to the instance of the class itself. When you call a method on an instance, Python automatically passes the instance as the first argument to the method using the self parameter. This allows the method to access and manipulate the instance's attributes and methods.

Q6. How does a Python class handle operator overloading?

In Python, operator overloading allows you to define special methods(dunder/magic methods) in a class that determine how instances of that class behave with built-in operators.

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

One should consider allowing operator overloading in your classes when it enhances the clarity, readability, and intuitiveness of your code. Operator overloading can make your code more natural and expressive, especially when your class models a concept that naturally corresponds to a specific operator's behavior.

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

One of the most popular forms of operator overloading in Python is for arithmetic operations. Overloading operators like **+**, **-**, *****, and **/** allows objects to behave in a mathematically intuitive way, making code more concise and readable when dealing with mathematical concepts. This is particularly common in classes that model numbers, vectors, matrices, and other mathematical entities.

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

The two most important concepts to grasp in order to comprehend Python OOP code are classes & objects, inheritance & polymorphism