Q1. Which two operator overloading methods can you use in your classes to support iteration?

The two operator overloading methods for supporting iteration are \_\_iter\_\_ and \_\_next\_\_. The \_\_iter\_\_ method initializes the iteration, and \_\_next\_\_ defines the next item to be retrieved in each iteration step using the next() function.

Q2. In what contexts do the two operator overloading methods manage printing?

The \_\_str\_\_ method is used for creating a string representation of an object, which is often used when printing instances using print() or str(). The \_\_repr\_\_ method provides an unambiguous representation that's used by the repr() function, often seen in interactive contexts.

Q3. In a class, how do you intercept slice operations?

To intercept slice operations in a class, you can define the \_\_getitem\_\_ method with appropriate slicing logic. It takes arguments for indexing, and you can process these arguments to return the desired sliced portion of your object.

Q4. In a class, how do you capture in-place addition?

To capture in-place addition (e.g., +=) in a class, you can define the \_\_iadd\_\_ method. This method modifies the object in place when the += operator is used. It allows you to customize the behavior of in-place addition for instances of your class.

Q5. When is it appropriate to use operator overloading?

Operator overloading is appropriate when you want to provide a more intuitive and expressive interface for your class instances. It's useful for creating objects that behave like built-in types or other familiar objects. Use it when overloading makes your code clearer and more readable, but avoid overloading operators in ways that might confuse users or violate their expectations.