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

To support iteration in your classes, you can use the following two operator overloading methods:

\_\_iter\_\_(self): This method is called when an iterator is requested for an object. It should return an iterator object.

\_\_next\_\_(self): This method is called to get the next value from the iterator. It should raise StopIteration when there are no more items to return.

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

The two operator overloading methods manage printing in the following contexts:

\_\_str\_\_(self): This method is called by the str() built-in function and by the print() function. It should return a string representation of the object.

\_\_repr\_\_(self): This method is called by the repr() built-in function to get a string representation that, ideally, should be unambiguous and could be used to recreate the object.

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

To intercept slice operations in a class, you can use the \_\_getitem\_\_(self, key) method. This method is called to retrieve the value associated with the specified key or slice.

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

To capture in-place addition in a class, you can use the \_\_iadd\_\_(self, other) method. This method is called for the += operator.

Q5. When is it appropriate to use operator overloading?

Operator overloading is appropriate when you want to define custom behaviour for built-in operators in your classes. It allows you to make your objects behave like built-in types and provides a way to define meaningful operations for objects of your class. Common use cases include supporting arithmetic operations, comparisons, iteration, printing, and more, depending on the desired behaviour of your class instances.