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

The \_\_iter\_\_ returns the iterator object and is implicitly called at the start of loops. The \_\_next\_\_ method returns the next value and is implicitly called at each loop increment. \_\_next\_\_ raises a StopIteration exception when there are no more value to return, which is implicitly captured by looping constructs to stop iterating.

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

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

There are getter and setter methods as a part of these magical methods. They are implemented by \_\_getitem\_\_() and \_\_setitem\_\_() methods. But, these methods are used only in indexed attributes like arrays, dictionaries, lists e.t.c. Instead of directly accessing and manipulating class attributes, it provides such methods, so these attributes can be modified only by its own instances and thus implements abstraction.

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

Python provides the operator x += y to add two objects in-place by calculating the sum x + y and assigning the result to the first operands variable name x. You can set up the in-place addition behavior for your own class by overriding the magic “dunder” method \_\_iadd\_\_(self, other) in your class definition.

Q5. When is it appropriate to use operator overloading?

Improves code readability by allowing the use of familiar operators.

Ensures that objects of a class behave consistently with built-in types and other user-defined types.

Makes it simpler to write code, especially for complex data types.

Allows for code reuse by implementing one operator method and using it for other operators.