**Assignment - 4**

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

Ans: The two operator overloading methods used to support iteration in classes are \_\_iter\_\_() and \_\_next\_\_().

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

Ans: The \_\_str\_\_() method is commonly used to manage printing when the str() function or print() statement is called on an object. Alternatively, the \_\_repr\_\_() method can also control the representation of an object when it is printed, especially in contexts where repr() is used.

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

Ans: To intercept slice operations in a class, you can implement the \_\_getitem\_\_() method. This method allows instances of the class to behave like sequences or mappings and enables slicing functionality when indexing is used.

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

Ans: In a class, in-place addition can be captured by implementing the \_\_iadd\_\_() method. This method is invoked when the += operator is used with an instance of the class, allowing you to define custom behavior for in-place addition operations.

Q5. When is it appropriate to use operator overloading?

Ans: Operator overloading is appropriate when you want to define custom behavior for built-in operators in Python, such as arithmetic operators (+, -, \*, /), comparison operators (<, >, ==, !=), or other operators like indexing ([]) or membership testing (in). It allows you to make your objects behave like built-in types or integrate seamlessly with existing Python syntax, improving code readability and expressiveness. However, it should be used judiciously to avoid confusion and maintain code clarity.