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

***Ans:***

1.‘\_\_iter\_\_(self)’: this method is called when you use the build-in ‘iter()’ function on an instance of your class.

2. ‘\_\_next\_\_(self)’: this method is called by the iterator object returned from “\_\_iter\_\_()”. It should return the next value in the iteration, or raise a ‘stopiteration’ exception if there are no more values.

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

***Ans:***

1. ‘\_\_str\_\_(self)’: this method is called by the built-in ‘str()’ function and ‘print()’ function when you want to print an object.

2.’\_\_repr\_\_(self)’: this method is called by the built-in ‘repr()’ function when you want to print a string representation of an object that can be used to recreate the object.

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

***Ans:***

to intercept slice operations in a class, we can define the ‘\_\_getitem\_\_()’ method within the class. The ‘\_\_getitem\_\_()’ method is a special method in python that allows objects to be indexed or sliced using square bracket [].

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

***Ans:***

to capture in place addition in a class. We can define the ‘\_\_iadd\_\_()’ method within the class. The ‘\_\_iadd\_\_()’ method is a special method in python that allows objects to support the ‘+=’ operator.

Q5. When is it appropriate to use operator overloading?

***Ans:***

operator overloading is appropriate when you want to define custom behaviour for operators in a class or a data type.