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

Answer: __iter__ can be called to to return an iterator object which allows the iteration, __iter__ is initiated at the start of a loop. The __next__ method returns the next value in the iterable , the __next__ method is called implicitly at each loop increment. __next__ also raises a Stop Iteration exception to signify the last element in the iterable. The stop iteration exception is implicitly captured by the loop during runtime.

Q2. In what contexts do the two operator overloading methods manage printing?
Answer: __str__ method is called when the print function is executed. The __str__ function can be overloaded with its own definition, Also str() is the built-in method that casts an instance of an object into a string representation.

```
class Vector:
   def __init__(self, x_comp, y_comp):
      self.x_comp = x_comp
      self.y_comp = y_comp

   def __str__(self):
       # By default, sign of +ve number is not displayed
       # Using `+`, sign is always displayed
       return f'{self.x_comp}i{self.y_comp:+}j'
```

The + sign is always displayed by the str method but the +ve symbol of positive numbers is not displayed

Q3. In a class, how do you intercept slice operations?
Answer: The slice operation in a class returns a slice object. The slice function works on data types such as tuples, list, string etc or objects on which __getItem__ and __len__ functions can be implemented. The slice() function returns a slice object that can slice any tuple,list,string, range data type.

```
class sliceOperation:
   def __init__(self,listObj):
```

```
        self.listObj = listObj

    def cut(self):
        s1 = slice(1,3)
        s2 = slice(1,3,2)
        print(self.listObj[s1])
        print(self.listObj[s2])

string = "Helio"
op = sliceOperation(string)
op.cut()
```

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

```
class Data:
   def __init__(self, data):
      self.data = data

   def __iadd__(self, other):
      self.data += other.data
      return self
x = Data(40)
y = Data(2)
x += y

print(x.data)
# 42
```

Q5. When is it appropriate to use operator overloading?
Answer: Operator overloading is required when user wants to apply the operator on objects which represent user defined data types, but the operator does not know how

to work on those objects , hence the user has to define a function that does the necessary operation on the objects using the operator.
For instance using + operator we can add two numbers and also concatenate two strings, this is because + is overloaded by str and int class.