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

**ANS:-**

**add() and mul() methods can be used for both integers and string data objects. String is iterable object. When two strings are passed to + operator, it will return concatenated string. When \* (number) is preceeded by string, then that string is repeated those many number of time.**

**print("akash" + 'deep')**

**print("pqr" \* 5)**

**akashdeep**

**pqrpqrpqrpqrpqr**

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

**ANS:-**

**It depends on the input parameter result is printed. Example, If both inputs are string function will print string output.**

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

**ANS:-**

**import profile**

**import sys**

**class InterceptedList2(list):**

**def \_\_setitem\_\_(self, key, value):**

**print ('saving')**

**list.\_\_setitem\_\_(self, key, value)**

**def \_\_delitem\_\_(self, key):**

**print ('saving')**

**list.\_\_delitem\_\_(self, key)**

**l2 = InterceptedList2()**

**l2.extend([1,2,3,4])**

**profile.run("l2[3:] = [5]")**

**profile.run("l2[2:6] = [12,4]")**

**profile.run("l2[-1:] = [42]")**

**profile.run("l2[::2] = [6,6]")**

saving

40 function calls in 0.000 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

3 0.000 0.000 0.000 0.000 :0(acquire)

3 0.000 0.000 0.000 0.000 :0(append)

1 0.000 0.000 0.000 0.000 :0(exec)

2 0.000 0.000 0.000 0.000 :0(getpid)

2 0.000 0.000 0.000 0.000 :0(isinstance)

1 0.000 0.000 0.000 0.000 :0(print)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.000 0.000 0.000 0.000 <ipython-input-3-e65e972bf339>:8(\_\_setitem\_\_)

1 0.000 0.000 0.000 0.000 <string>:1(<module>)

3 0.000 0.000 0.000 0.000 iostream.py:197(schedule)

2 0.000 0.000 0.000 0.000 iostream.py:310(\_is\_master\_process)

2 0.000 0.000 0.000 0.000 iostream.py:323(\_schedule\_flush)

2 0.000 0.000 0.000 0.000 iostream.py:386(write)

3 0.000 0.000 0.000 0.000 iostream.py:93(\_event\_pipe)

1 0.000 0.000 0.000 0.000 profile:0(l2[3:] = [5])

0 0.000 0.000 profile:0(profiler)

3 0.000 0.000 0.000 0.000 socket.py:432(send)

3 0.000 0.000 0.000 0.000 threading.py:1017(\_wait\_for\_tstate\_lock)

3 0.000 0.000 0.000 0.000 threading.py:1071(is\_alive)

3 0.000 0.000 0.000 0.000 threading.py:513(is\_set)

saving

32 function calls in 0.000 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

2 0.000 0.000 0.000 0.000 :0(acquire)

2 0.000 0.000 0.000 0.000 :0(append)

1 0.000 0.000 0.000 0.000 :0(exec)

2 0.000 0.000 0.000 0.000 :0(getpid)

2 0.000 0.000 0.000 0.000 :0(isinstance)

1 0.000 0.000 0.000 0.000 :0(print)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.000 0.000 0.000 0.000 <ipython-input-3-e65e972bf339>:8(\_\_setitem\_\_)

1 0.000 0.000 0.000 0.000 <string>:1(<module>)

2 0.000 0.000 0.000 0.000 iostream.py:197(schedule)

2 0.000 0.000 0.000 0.000 iostream.py:310(\_is\_master\_process)

2 0.000 0.000 0.000 0.000 iostream.py:323(\_schedule\_flush)

2 0.000 0.000 0.000 0.000 iostream.py:386(write)

2 0.000 0.000 0.000 0.000 iostream.py:93(\_event\_pipe)

1 0.000 0.000 0.000 0.000 profile:0(l2[2:6] = [12,4])

0 0.000 0.000 profile:0(profiler)

2 0.000 0.000 0.000 0.000 socket.py:432(send)

2 0.000 0.000 0.000 0.000 threading.py:1017(\_wait\_for\_tstate\_lock)

2 0.000 0.000 0.000 0.000 threading.py:1071(is\_alive)

2 0.000 0.000 0.000 0.000 threading.py:513(is\_set)

saving

32 function calls in 0.000 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

2 0.000 0.000 0.000 0.000 :0(acquire)

2 0.000 0.000 0.000 0.000 :0(append)

1 0.000 0.000 0.000 0.000 :0(exec)

2 0.000 0.000 0.000 0.000 :0(getpid)

2 0.000 0.000 0.000 0.000 :0(isinstance)

1 0.000 0.000 0.000 0.000 :0(print)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.000 0.000 0.000 0.000 <ipython-input-3-e65e972bf339>:8(\_\_setitem\_\_)

1 0.000 0.000 0.000 0.000 <string>:1(<module>)

2 0.000 0.000 0.000 0.000 iostream.py:197(schedule)

2 0.000 0.000 0.000 0.000 iostream.py:310(\_is\_master\_process)

2 0.000 0.000 0.000 0.000 iostream.py:323(\_schedule\_flush)

2 0.000 0.000 0.000 0.000 iostream.py:386(write)

2 0.000 0.000 0.000 0.000 iostream.py:93(\_event\_pipe)

1 0.000 0.000 0.000 0.000 profile:0(l2[-1:] = [42])

0 0.000 0.000 profile:0(profiler)

2 0.000 0.000 0.000 0.000 socket.py:432(send)

2 0.000 0.000 0.000 0.000 threading.py:1017(\_wait\_for\_tstate\_lock)

2 0.000 0.000 0.000 0.000 threading.py:1071(is\_alive)

2 0.000 0.000 0.000 0.000 threading.py:513(is\_set)

saving

32 function calls in 0.000 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

2 0.000 0.000 0.000 0.000 :0(acquire)

2 0.000 0.000 0.000 0.000 :0(append)

1 0.000 0.000 0.000 0.000 :0(exec)

2 0.000 0.000 0.000 0.000 :0(getpid)

2 0.000 0.000 0.000 0.000 :0(isinstance)

1 0.000 0.000 0.000 0.000 :0(print)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.000 0.000 0.000 0.000 <ipython-input-3-e65e972bf339>:8(\_\_setitem\_\_)

1 0.000 0.000 0.000 0.000 <string>:1(<module>)

2 0.000 0.000 0.000 0.000 iostream.py:197(schedule)

2 0.000 0.000 0.000 0.000 iostream.py:310(\_is\_master\_process)

2 0.000 0.000 0.000 0.000 iostream.py:323(\_schedule\_flush)

2 0.000 0.000 0.000 0.000 iostream.py:386(write)

2 0.000 0.000 0.000 0.000 iostream.py:93(\_event\_pipe)

1 0.000 0.000 0.000 0.000 profile:0(l2[::2] = [6,6])

0 0.000 0.000 profile:0(profiler)

2 0.000 0.000 0.000 0.000 socket.py:432(send)

2 0.000 0.000 0.000 0.000 threading.py:1017(\_wait\_for\_tstate\_lock)

2 0.000 0.000 0.000 0.000 threading.py:1071(is\_alive)

2 0.000 0.000 0.000 0.000 threading.py:513(is\_set)

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

**ANS:-**

**Python in its definition provides methods to perform inplace operations, i.e doing assignment and computation in a single statement using “operator” module. For example,**

**x += y is equivalent to x = operator.iadd(x, y)**

**iadd() function is used to assign and add the current value. This operation does “a+=b” operation. Assigning is not performed in case of immutable containers, such as strings, numbers and tuples.**

**import operator**

**x = 2**

**y = 3**

**x = operator.iadd(x,y)**

**print(x)**

**5**

**Q5. When is it appropriate to use operator overloading?**

**ANS:-**

**When we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary ‘+’ operator it throws an error, because compiler don’t know how to add two objects. So we define a method for an operator and that process is called operator overloading.**