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

A:  **\_\_iter\_\_() and \_\_next\_\_()** .

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

A:

listA = ['a','e','i','o','u']

iter\_listA = iter(listA)

try:

print( next(iter\_listA))

print( next(iter\_listA))

print( next(iter\_listA))

print( next(iter\_listA))

print( next(iter\_listA))

print( next(iter\_listA)) #StopIteration error

except:

pass

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

A: class demo:

sliced ='abcde'.\_\_getitem\_\_(slice(0, 3, 1))

print(sliced)

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

A: class demo:

import operator

# Initializing values

x = 5

y = 6

a = 5

b = 6

# using add() to add the arguments passed

z = operator.add(a,b)

# using iadd() to add the arguments passed

p = operator.iadd(x,y)

# printing the modified value

print ("Value after adding using normal operator : ",end="")

print (z)

# printing the modified value

print ("Value after adding using Inplace operator : ",end="")

print (p)

# printing value of first argument

# value is unchanged

print ("Value of first argument using normal operator : ",end="")

print (a)

# printing value of first argument

# value is unchanged

print ("Value of first argument using Inplace operator : ",end="")

print (x)

Q5. When is it appropriate to use operator overloading?

A:

When one or both operands are of a user-defined class or structure type, operator overloading makes it easier to specify user-defined implementation for such operations.