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

Ans:- Two methods cannot have the same name in Python. Method overloading in Python is **a feature that allows the same operator to have different meanings.**

Operator overloading in Python is **the ability of a single operator to perform more than one operation based on the class (type) of operands**. For example, the + operator can be used to add two numbers, concatenate two strings or merge two lists.

Class can support iteration by defined by \_\_getitem or \_\_iter\_\_.

1. \_\_iter\_\_ which returns an object that support the iteation protocol with a \_\_next\_\_ method.

2.\_\_getitems\_\_ by indexing method(which is called repeatedly with successively

Higher indexes.

**\_\_iter\_\_() and \_\_next\_\_()** , collectively called the iterator protocol. An object is called iterable if we can get an iterator from it. Most built-in containers in Python like: list, tuple, string etc. are iterables.

The \_\_iter\_\_ returns the iterator object and is implicitly called at the start of loops. The \_\_next\_\_ method returns the next value and is implicitly called at each loop increment. \_\_next\_\_ raises a StopIteration exception when there are no more value to return, which is implicitly captured by looping constructs to stop iterating.

**class** Counter:

**def** \_\_init\_\_(self, low, high):

self.current = low

self.high = high

**def** \_\_iter\_\_(self):

**return** self

**def** \_\_next\_\_(self):

**if** self.current > self.high:

**raise** StopIteration

**else**:

self.current += 1

**return** self.current - 1

**for** num **in** Counter(5, 15):

**print**(num)

output:-5

6

..

..

15

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

Ans:- The \_\_str\_\_ and \_\_repr\_\_ methods implementing object print displays.The former is called by a print and str built-in-function the later is called by print and str if there is no \_\_str\_\_ and always by the \_\_repr\_\_ built in interactive echos

And nested appearances that is, \_\_repr\_\_is used everywhere, except by print and str when a \_\_str\_\_ is defined.A \_\_str\_\_ is usually used for user friendly displays, \_\_repr\_\_ gives extra details or the objects as code form.

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

Ans:- slicing is caught by the \_\_getitem\_\_ indexing method : it is called with a slice

Object, instead of a simple index.\_\_getslice\_\_ may be used as well.

**Syntax:**

\_\_getitem\_\_(slice(start, stop, step))

**Parameter:**

* **slice() :** constructor to create slice object.
* **start:** An integer number specifying start index.It is optional and default is 0.
* **stop:** An integer number specifying end index.
* **step:** An integer number specifying the step of slicing. It is optional and  
  default is 1.

sliced ='abcde'.\_\_getitem\_\_(slice(0, 2, 1))

print(sliced)

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

Ans:- In- place addition tries \_\_iadd\_\_ first, and \_\_add\_\_ with an assignment second. The same pattern holds true for all binary operator. The\_\_radd\_\_method is also available for right side addition.

**1. iadd()** :- This 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.

**2. iconcat()** :- This function is used to**concat** one string at end of second.

# Python code to demonstrate the working of

# iadd() and iconcat()

# importing operator to handle operator operations

import operator

# using iadd() to add and assign value

x = operator.iadd(2, 3);

# printing the modified value

print ("The value after adding and assigning : ", end="")

print (x)

# initializing values

y = "geeks"

z = "forgeeks"

# using iconcat() to concat the sequences

y = operator.iconcat(y, z)

# using iconcat() to concat sequences

print ("The string after concatenation is : ", end="")

print (y)

Q5. When is it appropriate to use operator overloading?

Ans:- The purpose of operator overloading is to **provide a special meaning of an operator for a user-defined data type** These operators can be overloaded globally or **on a class-by-class basis**. Overloaded operators are implemented as functions and can be member functions or global functions. An overloaded operator is called an operator function. You declare an operator function with the keyword operator preceding the operator.

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used **to add two integers as well as join two strings and merge two lists**. It is achievable because '+' operator is overloaded by int class and str class.