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

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)

**Sample output of above program.**

5

6

..

..

15

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

Operator Overloading is the phenomenon of giving alternate/different meaning to an action performed by an operator beyond their predefined operational function. Operator overloading is also called **Operator Ad-hoc Polymorphism**.

Python operators work for built-in classes. But the same operator expresses differently with different types. For example, The + operator will perform arithmetic addition on two numbers, merge two lists and concatenate two strings. Python allows the same operator to have different meanings according to the referring context.

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

setslice" and "delslice" are deprecated, if you want to do the interception you need to **work with python slice objects passed to "setitem" and "delitem"**. If you want to intecept both slices and ordinary accesses this code works perfectly in python 2.6. 2.

# Python slice() function

The slice() method returns a portion of an iterable as an object of the slice class based on the specified range. It can be used with [string](https://www.tutorialsteacher.com/python/python-string), [list](https://www.tutorialsteacher.com/python/python-list), [tuple](https://www.tutorialsteacher.com/python/python-tuple), [set](https://www.tutorialsteacher.com/python/python-set), bytes, or [range](https://www.tutorialsteacher.com/python/range-method) objects or [custom class](https://www.tutorialsteacher.com/python/python-class) object that implements sequence methods \_\_getitem\_\_() and \_\_len\_\_() methods.

### Syntax:

slice(stop)

slice(start, stop, step)

#### Parameters:

1. start: (Optional) Starting index where the slicing of the iterable starts. Deault value is none.
2. stop: Ending index where the slicing should end.
3. step: (Optional) An integer to increment starting index. Defaults to None.

#### Return Value:

Returns an object of the slice class.

The following examples demonstrates the slice() method with a string value.

Example: slice() with String

 Copy

mystr = 'TutorialsTeacher'

nums = [1,2,3,4,5,6,7,8,9,10]

portion1 = slice(9)

portion2 = slice(2, 8, 2)

print('slice: ', portion1)

print('String value: ', mystr[portion1])

print('List value: ', nums[portion1])

print('slice: ', portion2)

print('String value: ', mystr[portion2])

print('List value: ', nums[portion2])

Output

slice: slice(None, 9, None)

String value: Tutorials

List value: [1,2,3,4,5,6,7,8,9]

slice: slice(2, 8, 2)

String value: tra

List value: [3, 5, 7]

Above, slice(9) returns the slice object as slice(None, 9, None), which you can pass to any iterable object to get that index portion.

The slice() method can be used with string, list, tuple, set, bytes, or range. The following example fetch the portion of the list object.

Example: slice() with List, Tuple

 Copy

nums = [1,2,3,4,5,6,7,8,9,10]

odd\_portion = slice(0, 10, 2)

print(nums[odd\_portion])

even\_portion = slice(1, 10, 2)

print(nums[even\_portion])

Output

[1, 3, 5, 7, 9]

[2, 4, 6, 8, 10]

## Using Negative Index

The slice() method also supports negative indexing.

Example: slice() with Negative Indexing

 Copy

nums = [1,2,3,4,5,6,7,8,9,10]

even\_portion = slice(-9, -1, 2)

print(nums[even\_portion])

Output

[2, 4, 6, 8]

## Using Indexing Syntax

You can use iterable\_obj[start:stop:step] shortcut syntax for slicing instead of using the slice() method.

Example: obj[start:stop:step] Syntax

 Copy

nums = [1,2,3,4,5,6,7,8,9,10]

print('Odd nums: ', nums[0:10:2]) # start 0, stop: 10, step:2

print('Even nums: ', nums[1:10:2]) # start 1, stop: 10, step:2

mystr = 'TutorialsTeacher'

print(mystr[0:9]) # start 0, stop: 9, step:1

print(mystr[9:]) # start 9, stop: end of string, step:1

print(mystr[9::2]) # start 9, stop: end of string, step:2

print(mystr[::2]) # start 0, stop: end of string, step:2

Output

Odd nums: [1, 3, 5, 7, 9]

Even nums: [2, 4, 6, 8, 10]

Tutorials

Teacher

Tahr

Ttrasece

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

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 operands 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.

Q5. When is it appropriate to use operator overloading?

Operator overloading is mostly useful **when you're making a new class that falls into an existing "Abstract Base Class" (ABC)** -- indeed, many of the ABCs in standard library module collections rely on the presence of certain special methods (and special methods, one with names starting and ending with double underscores.