Slicing in Python

List is arguably the most useful and ubiquitous type in Python. One of the reasons it's so handy is Python slice notation. In short, **slicing is a flexible tool to build new lists out of an existing list**.

Python supports slice notation for any sequential data type like lists, strings, tuples, bytes, bytearrays, and ranges. Also, any new data structure can add its support as well. This is greatly used (and abused) in NumPy and Pandas libraries, which are so popular in Machine Learning and Data Science. It's a good example of "learn once, use everywhere".

After getting the list, we can get a part of it using python's slicing operator which has following syntax :

```
list_name [start : stop : steps]
```

Which means that (Note: In left to right traversal)

- slicing will start from index **`start`**
- will go up to **`stop`**(***but not including the stop value***) in step of **`steps`**.
- Default value of **`start`** is **0**, **`stop`** is ***`length of list`*** and for **`steps`** it is **1**

Which means that (Note: In Right to left traversal)

- **`steps`** value is negative.
- slicing will start from index **`start`**
- will go up to **`stop`**(***but not including the stop value***) in step of **`steps`**.
- Default value of **`start`** is **-1**, **`stop`** is **`-(length of list+1)`**

Note:

1. We create a sublist of a list by specifying a range of indices.
2. Semantically specifying the slice is similar to that of range function. But syntactically, it is different.
3. The result of slicing is a new list.

```
# file 1_slice.py
# indices:
#    0    1   2   3   4   5   6
b = [12, 23, 34, 45, 56, 67, 78]
#    -7  -6  -5  -4  -3  -2  -1
print(b[2:5]) # [34, 45, 56]
print(b[:5]) # b[0:5] # [12, 23, 34, 45, 56]
print(b[2:]) # b[2:len(b)] # [34, 45, 56, 67, 78]
print(b[2:6:2]) # init : 2, final value one past the end : 6; step 2 # [34, 56]
print(b[::2])  # init : 0, final value one past the end : len(list); step : 2
               # [12, 34, 56, 78]
```

```
print(b[::-1])  # reverse the elements of the list # [78, 67, 56, 45, 34, 23, 12]
```

Negative value of steps shows right to left traversal instead of left to right traversal that is why [: : -1] **prints list in reverse order.**

```
#    0    1   2   3   4   5   6
b = [12, 23, 34, 45, 56, 67, 78]
#    -7  -6  -5  -4  -3  -2  -1
```

```
print("what : ", b[:5:-1]) # what :  [78],    b[:5:-1]) is same as b[-1:5:-1])
```

Let us try a couple of examples based on slicing of lists.

Example: find the biggest

```
# file : 2_biggest.py
# find the biggest
# algorithm:
# assume the first element in index 0 as the biggest
# walk thru the remaining elements of the list.
#     compare and update big if necessary.
# output biggest
a = [22, 44, 11, 55, 33]
big = a[0]
for e in a[1:]:  # observe : all elements but for the 0th element
      if e > big :
            big = e
print("biggest : ", big)
```

Example: find the total of a slice

```
# file : 3_find_total.py
# find the total of a batsman
# list contains the name of the batsman and his scores in # of innings
# find the total score of Kohli
scores = [ "kohli", 0, 82, 25, 120, 76]
total = 0
for e in scores[1:]:
      total += e
print("total : ", total)
```

```
$ python 3_find_total.py
total :  303
```

Taking n first elements of a list

Slice notation allows you to skip any element of the full syntax. If we skip the start number then it starts from 0 index:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[:5]
[10, 20, 30, 40, 50]
```

So, ***nums[:5] is equivalent to nums[0:5].*** This combination is a handy shortcut to take n first elements of a list.

Taking n last elements of a list

Negative indexes allow us to easily take n-last elements of a list:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]

>>> nums[-3:]
[70, 80, 90]
```

Here, the stop parameter is skipped. That means you take from the start position, till the end of the list. We start from the third element from the end (value 70 with index -3) and take everything to the end.

We can freely mix negative and positive indexes in start and stop positions:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[1:-1]
[20, 30, 40, 50, 60, 70, 80]
>>> nums[-3:8]
[70, 80]
>>> nums[-5:-1]
[50, 60, 70, 80]
```

Taking all but n last elements of a list (Leaving n last elements of a list)

Another good usage of negative indexes:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[:-2]
[10, 20, 30, 40, 50, 60, 70]
```

We take all but the last two elements of original list.

Taking every nth-element of a list

What if we want to have only every 2-nd element of nums? This is where the step parameter comes into play:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[::2]
[10, 30, 50, 70, 90]
```

Here we omit start/stop parameters and use only step. By providing start we can skip some elements:

```
>>> nums[1::2]
[20, 40, 60, 80]
```

And **if we don't want to include some elements at the end**, we can also add the stop parameter:

```
>>> nums[1:-3:2]
[20, 40, 60]
```

Using Negative Step and Reversed List

We can use a negative *step* to obtain a reversed list:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[::-1]
[90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Negative step changes a way, slice notation works. It makes the slice be built from the tail of the list. So, it goes from the last element to the first element. That's why we get a reversed list with a negative step.

Due to this peculiarity, start and stop should be provided from right to left as well. E.g., if you want to have a reversed list which starts from 80:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[-2::-1]
[80, 70, 60, 50, 40, 30, 20, 10]
```

So, we start from the -2 element (value 80) and go from right to left collecting all the elements in a reversed list.

We can use stop value to stop taking before some element. E.g., let's not include 20 and 10 values:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[-2:1:-1]
[80, 70, 60, 50, 40, 30]
```

We use 1 for stop index, which is the element with value 20. So, we go from 80 till 30, not including value 20.

It's a bit baffling that with a negative step, stop index is located before start. Negative step turns everything upside down.

Of course, we can use an arbitrary negative step:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[-2:1:-3]
[80, 50]
```

Slice and Copying

One important thing to notice – is that **list slice creates a shallow copy of the initial list**. That means, we can safely modify the new list and it will not affect the initial list:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> first_five = nums[0:5]
>>> first_five[2] = 33
>>> first_five
[10, 20, 33, 40, 50]
>>> nums
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Despite our mutating the element under index 2, it does not affect the nums list, because first_five list – is a partial copy of nums list.

There is the shortest form of slice notation – just colons *nums[:]*.

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums_copy = nums[:]
>>> nums_copy[0] = 33
>>> nums_copy
[33, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

It creates a shallow copy of the whole list and is a good shorthand when you need a copy of the original list.

Assignment of slice:

When we use to the left of assignment,

- **all the elements in that slice will be removed**
- **all the elements on the right will replace them**

Note :

1. **The right hand side should signify # of elements. It should be an iterable.**
2. **Number of elements on either side need not be same.**

Observe the outputs and comments after each assignment.

```
# file : 4_slice_assignment.py
# assignment of slice
a = [ 10, 20, 30, 40, 50, 60]
# remove the elements on the left
# replace by the elements on the right
a[2:4] = [100, 200]  #right hand side(rhs) should be iterable
print(a)
```

```
# [10, 20, 100, 200, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]
a[2:4] = [1000, 2000, 3000, 4000]
print(a) # list has become bigger
# [10, 20, 1000, 2000, 3000, 4000, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]
a[2:4] = [] # list has become smaller
print(a)
# [10, 20, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]
a[2:4] = "pesu" #  str is iterable
print(a)
# [10, 20, 'p', 'e', 's', 'u', 50, 60]

a = [ 10, 20, 30, 40, 50, 60]
a[2] = "fool"  # a[2] is not a slice;
print(a)
#[10, 20, 'fool', 40, 50, 60]

a = [ 10, 20, 30, 40, 50, 60]
a[2:3] = "fool"  #a[2:3] is a slice
print(a)
# [10, 20, 'f', 'o', 'o', 'l', 40, 50, 60]
```

Short Notes:

Left to right traversal

```
#     0  1  2  3  4  5  6  7  8  9
nums=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
#    -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```

```
nums[:4]   # Taking first n elements of a list
[1, 2, 3, 4]
```

```
nums[4:]   # leaving first n elements of a list
[5, 6, 7, 8, 9, 10]

nums[:-4]   # Leaving last n elements of a list (Taking all but n last elements of a list)
[1, 2, 3, 4, 5, 6]

nums[-4:]   # Taking last n elements of a list
[7, 8, 9, 10]

nums[3:-3]   # Leaving on either side n elements of a list
nums[::2]   # Taking every nth element of a list
```

Right to Left traversal

```
nums[::-1]   # We can use a negative step to obtain a reversed list
nums[::-2]   # Taking every nth element of a list in reverse order
```