Python List Slicing

To access a range of items in a list, you need to slice a list. One way to do this is to use the simple slicing operator :

With this operator you can specify where to start the slicing, where to end and specify the step.

Slicing a List

If L is a list, the expression L [start : stop : step] returns the portion of the list from index start to index stop, at a step size step.

Syntax

```
L[start:stop:step]
```

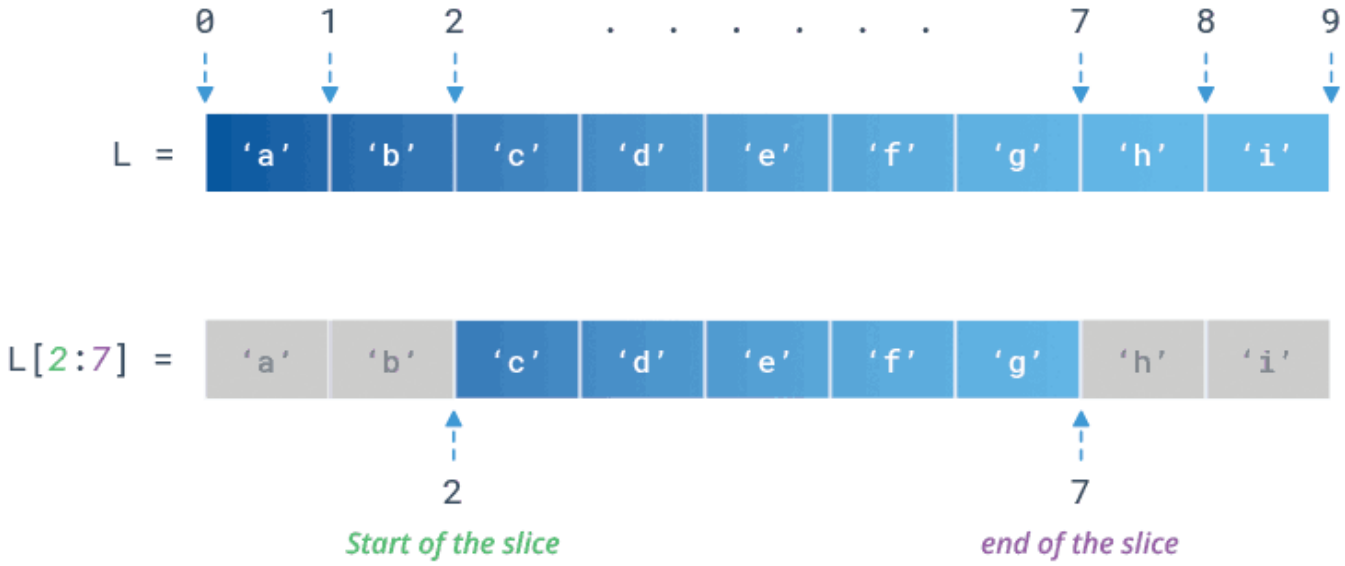
Start position *End position* *The increment*

Basic Example

Here is a basic example of list slicing.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

print(L[2:7])
# Prints ['c', 'd', 'e', 'f', 'g']
```



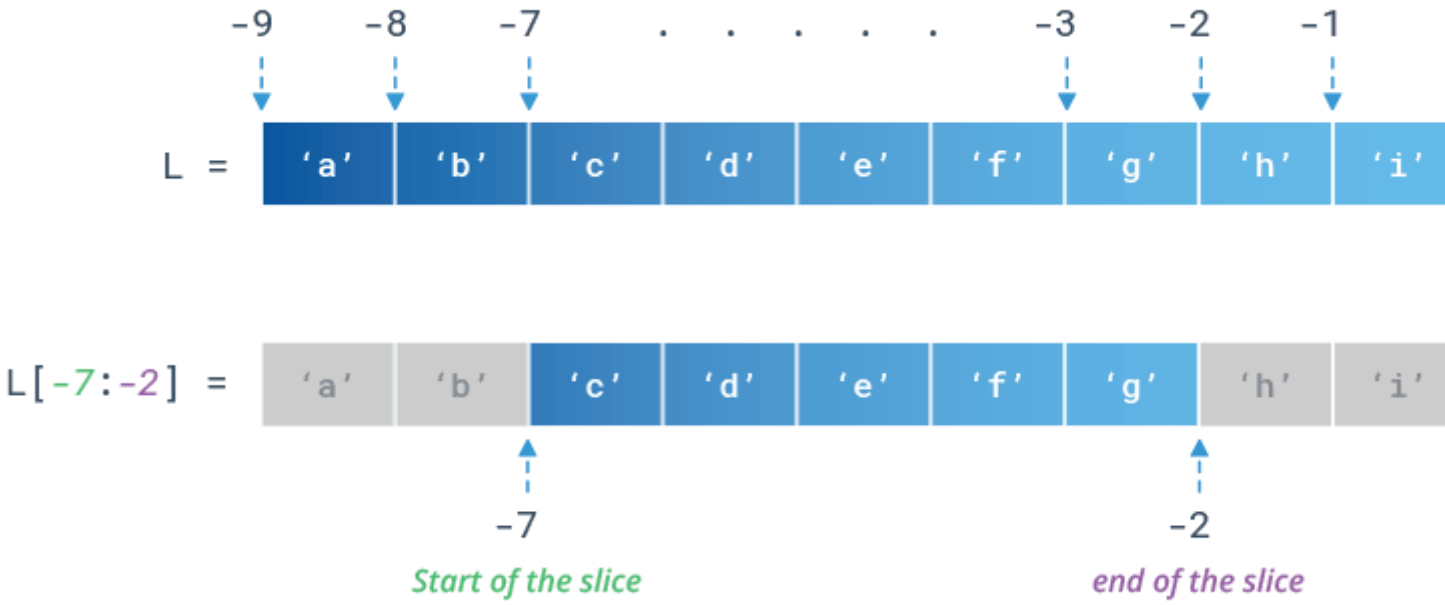
Note that the item at index 7 'h' is not included.

Slice with Negative Indices

You can also specify negative indices while slicing a list.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

print(L[-7:-2])
# Prints ['c', 'd', 'e', 'f', 'g']
```



Slice with Positive & Negative Indices

You can specify both positive and negative indices at the same time.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

print(L[2:-5])
# Prints ['c', 'd']
```

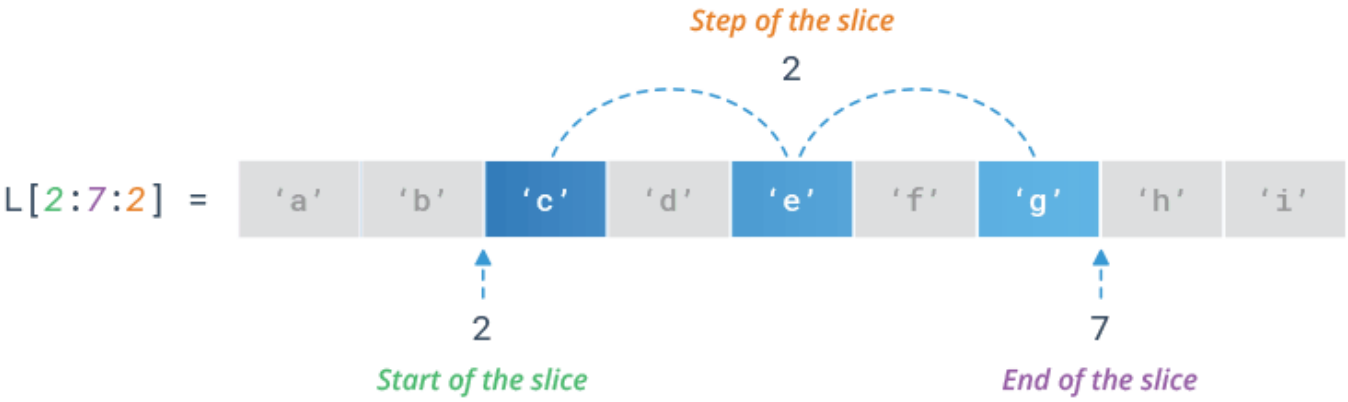
Specify Step of the Slicing

You can specify the step of the slicing using `step` parameter. The `step` parameter is optional and by default 1.

```
# Return every 2nd item between position 2 to 7

L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

print(L[2:7:2])
# Prints ['c', 'e', 'g']
```



Negative Step Size

You can even specify a negative step size.

```
# Return every 2nd item between position 6 to 1
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[6:1:-2])
# Prints ['g', 'e', 'c']
```

Slice at Beginning & End

Omitting the start index starts the slice from the index 0. Meaning, `L[:stop]` is equivalent to `L[0:stop]`

```
# Slice the first three items from the list
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[:3])
# Prints ['a', 'b', 'c']
```

Whereas, omitting the stop index extends the slice to the end of the list. Meaning, `L[start:]` is equivalent to `L[start:len(L)]`

```
# Slice the last three items from the list
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[6:])
# Prints ['g', 'h', 'i']
```

Reverse a List

You can reverse a list by omitting both `start` and `stop` indices and specifying a `step` as -1.

```
L = ['a', 'b', 'c', 'd', 'e']

print(L[::-1])
# Prints ['e', 'd', 'c', 'b', 'a']
```

Modify Multiple List Values

You can modify multiple list items at once with slice assignment. This assignment replaces the specified slice of a list with the items of assigned iterable.

```
# Modify multiple list items

L = ['a', 'b', 'c', 'd', 'e']

L[1:4] = [1, 2, 3]

print(L)
# Prints ['a', 1, 2, 3, 'e']


# Replace multiple elements in place of a single element

L = ['a', 'b', 'c', 'd', 'e']

L[1:2] = [1, 2, 3]

print(L)
# Prints ['a', 1, 2, 3, 'c', 'd', 'e']
```

Insert Multiple List Items

You can insert items into a list without replacing anything. Simply specify a zero-length slice.

```
# Insert at the start

L = ['a', 'b', 'c']

L[:0] = [1, 2, 3]
```

```
print(L)
# Prints [1, 2, 3, 'a', 'b', 'c']


# Insert at the end
L = ['a', 'b', 'c']
L[len(L):] = [1, 2, 3]
print(L)
# Prints ['a', 'b', 'c', 1, 2, 3]
```

You can insert items into the middle of list by keeping both the `start` and `stop` indices of the slice same.

```
# Insert in the middle
L = ['a', 'b', 'c']
L[1:1] = [1, 2, 3]
print(L)
# Prints ['a', 1, 2, 3, 'b', 'c']
```

Delete Multiple List Items

You can delete multiple items out of the middle of a list by assigning the appropriate slice to an empty list.

```
L = ['a', 'b', 'c', 'd', 'e']
L[1:5] = []
print(L)
# Prints ['a']
```

You can also use the del statement with the same slice.

```
L = ['a', 'b', 'c', 'd', 'e']
del L[1:5]
print(L)
# Prints ['a']
```

Clone or Copy a List

When you execute `new_List = old_List`, you don't actually have two lists. The assignment just copies the reference to the list, not the actual list. So, both `new_List` and `old_List` refer to the same list after the assignment.

You can use slicing operator to actually copy the list (also known as a shallow copy).

```
L1 = ['a', 'b', 'c', 'd', 'e']

L2 = L1[:]

print(L2)

# Prints ['a', 'b', 'c', 'd', 'e']

print(L2 is L1)
# Prints False
```