List Comprehension

```
"""
List comprehension offers a shorter syntax when you want to create a new list
based on the values of an existing list.

If you want a new list, containing only the fruits with the letter "a" in the
name.

Without list comprehension you will have to write a for statement with a
condition
 """
```

Example 0:

```
# to print all elements of a list
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits]
print(newlist)
```

Example 1:

```
# Create a new list out of the existing one which matches the letter 'e'
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "e" in x:
    newlist.append(x)

print(newlist)
```

Example 2:

```
# Above example using list-comprehension
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "e" in x]
print(newlist)
```

Example 3:

```
# To filter the values which does not match the string 'mango'
fruits = ["apple", "banana", "cherry", "kiwi", "mango", "apple"]
print(fruits)

newlist = [x for x in fruits if x != "mango"]
print(newlist)
```

Example 4:

```
# Generating numbers using list-comprehension
newlist = [x for x in range(1,11)]
print(newlist)
```

Example 5:

```
# To get list of even numbers from 1 to 10
newlist = [x for x in range(1,11) if x%2==0]
print(newlist)
```

Example 6:

```
# Using title() function of a list of strings using list comprehension
fruits = ["apple", "banana", "cherry", "kiwi", "mango", "apple"]
newlist = [x.title() for x in fruits]
print(newlist)
```

Example 7:

```
# Return the item if it is not banana, if it is banana return orange
fruits = ["apple", "banana", "cherry", "kiwi", "mango", "banana"]
newlist = [x if x != "banana" else "orange" for x in fruits]
print(newlist)
```

Example 8:

```
# Tredetional way to convert a string characters to a list of char's
l = []

# Traditional approach of iterating
for character in 'Firstman Computers':
    l.append(character)

# Display list
print(l)
```

Example 9:

```
# Using list comprehension to iterate through loop
l = [x for x in 'Firstman Computer Training']

# Displaying list
print(l)
```

Example 10:

```
matrix = []

for i in range(3):

    # Append an empty sublist inside the list
    matrix.append([])

    for j in range(5):
        matrix[i].append(j)

print(matrix)
```

Example 11:

```
# Above prog using Nested list comprehension
matrix = [[j for j in range(5)] for i in range(3)]
print(matrix)
```

Example 12:

```
# Python program to remove multiple
# elements from a list using a condition

li=[10,13,15,20,25,24,28,30,32]
for i in list(li):
    if i % 5 == 0:
        li.remove(i)
print("List after removing elements ",li)

# list() is a fun used to typecast any group of values to a list.
x = 1,2,3,4,5
print(x)
print(type(x))
# convert the above tupel to a list
l = list(x)
```

```
print(l)
print(type(l))
```

Example 13:

```
# Python program to remove multiple
# elements from a list using list comprehension

li=[10,13,15,20,25,24,28,30,32]
li=[num for num in li if num%5!=0]

print("List after removing elements ",li)
```

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

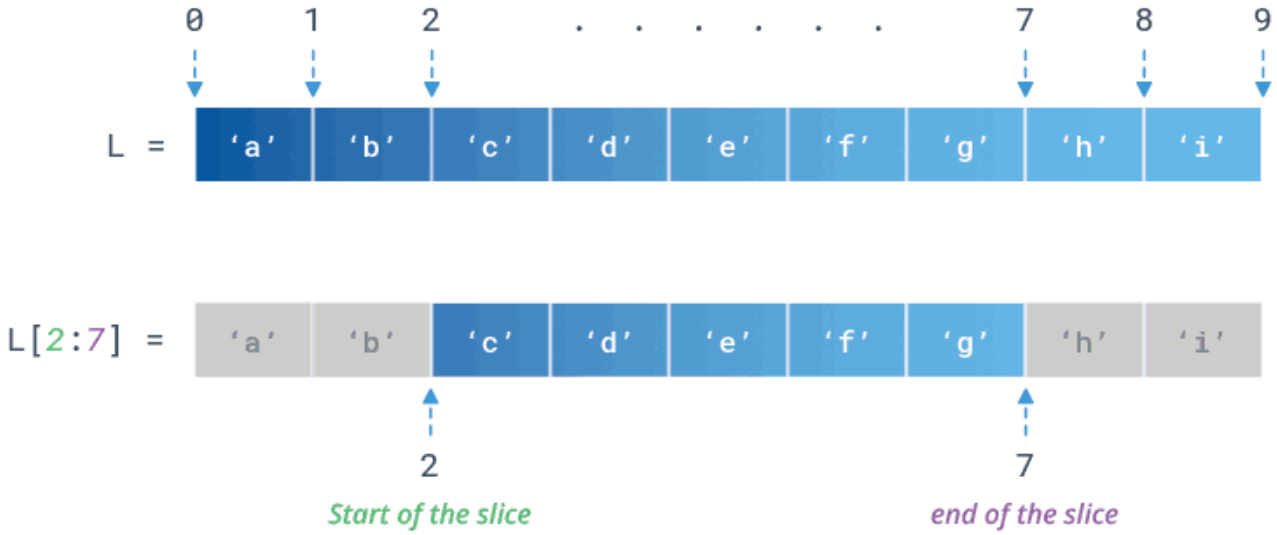
```
L[start:stop:step]
```

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']
```

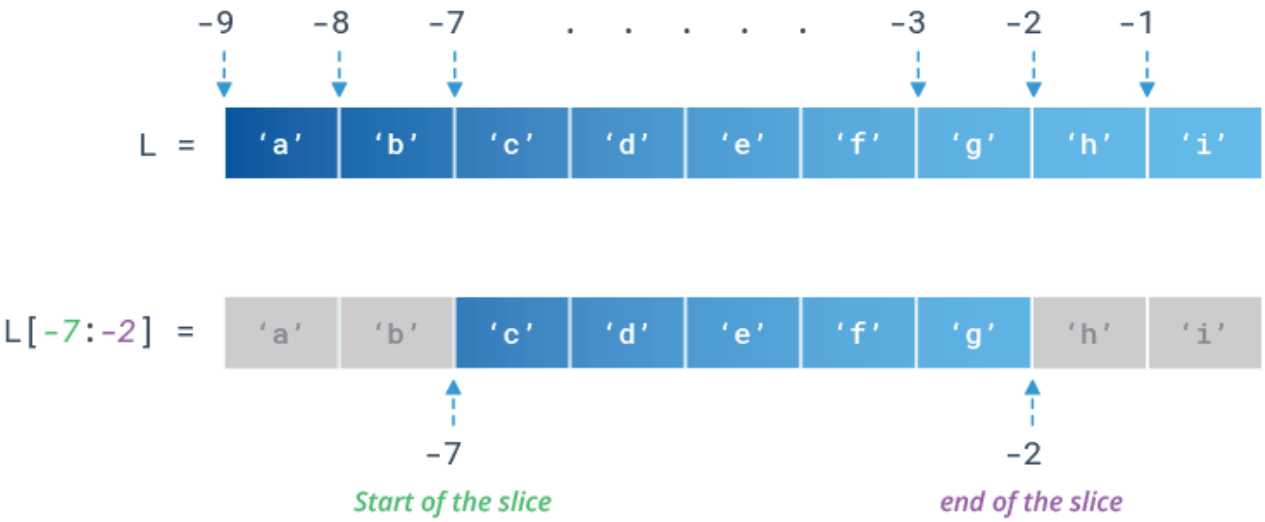


Note that the item at index 7 'h' is not included.

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

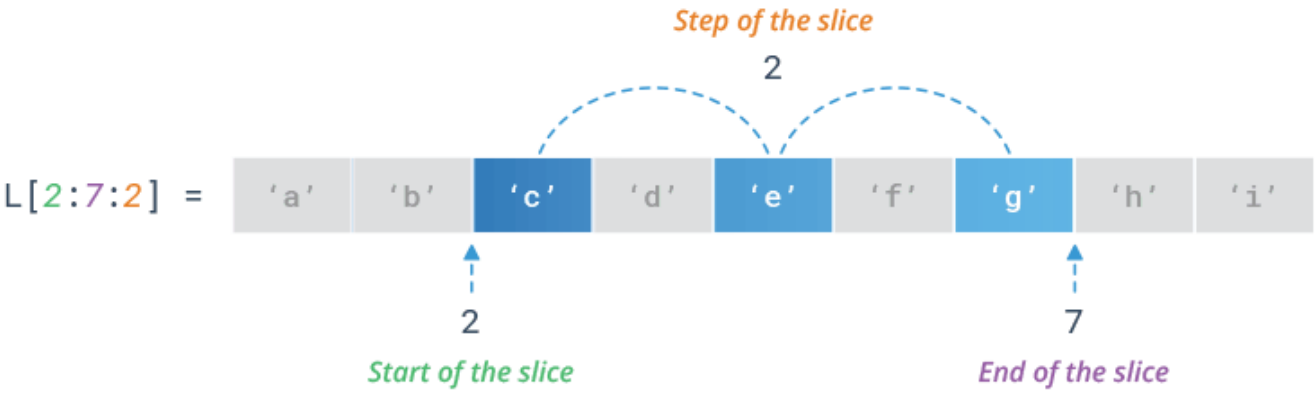
You can specify both positive and negative indices at the same time.

```
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
```