Python Lists

Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Lists in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplication of elements in the list, with each element having its own distinct place and credibility.

Note- Lists are a useful tool for preserving a sequence of data and further iterating over it.

Table of content:

- *List Comprehension*
- *Operations on List*
- *List Methods*

Creating a List

Lists in Python can be created by just placing the sequence inside the square brackets[]. Unlike Sets, a list doesn't need a built-in function for the creation of a list.

Note – Unlike Sets, the list may contain mutable elements.

- Python3

```
# Python program to demonstrate
# Creation of List

# Creating a List
List = []
print("Blank List: ")
print(List)

# Creating a List of numbers
List = [10, 20, 14]
print("\nList of numbers: ")
print(List)

# Creating a List of strings and
accessing
# using index
List = ["Geeks", "For", "Geeks"]
print("\nList Items: ")
print(List[0])
print(List[2])

# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Geeks', 'For'] ,
['Geeks']]
print("\nMulti-Dimensional List: ")
print(List)
```

Output:

```
Blank List:
[]

List of numbers:
[10, 20, 14]

List Items
Geeks
Geeks
```

```
Multi-Dimensional List:
[['Geeks', 'For'], ['Geeks']]
```

Creating a list with multiple distinct or duplicate elements

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

- Python3

```
# Creating a List with
# the use of Numbers
# (Having duplicate values)
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers:
")
print(List)

# Creating a List with
# mixed type of values
# (Having numbers and strings)
List = [1, 2, 'Geeks', 4, 'For', 6,
'Geeks']
print("\nList with the use of Mixed
Values: ")
print(List)
```

Output:

```
List with the use of Numbers:
[1, 2, 4, 4, 3, 3, 3, 6, 5]

List with the use of Mixed Values:
[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
```

Knowing the size of List

- Python3

```
# Creating a List
List1 = []
print(len(List1))

# Creating a List of
numbers
List2 = [10, 20, 14]
print(len(List2))
```

Output:

```
0
3
```

Adding Elements to a List

Using append() method

Elements can be added to the List by using the built-in **append()** function. Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used. Tuples can also be added to the list with the use of the append method because tuples are immutable. Unlike Sets, Lists can also be added to the existing list with the use of the append() method.

- Python3

```
# Python program to demonstrate
# Addition of elements in a List

# Creating a List
List = []
print("Initial blank List: ")
print(List)

# Addition of Elements
# in the List
List.append(1)
List.append(2)
List.append(4)
print("\nList after Addition of Three
elements: ")
print(List)

# Adding elements to the List
# using Iterator
for i in range(1, 4):
    List.append(i)
print("\nList after Addition of elements from
1-3: ")
print(List)

# Adding Tuples to the List
List.append((5, 6))
print("\nList after Addition of a Tuple: ")
print(List)

# Addition of List to a List
List2 = ['For', 'Geeks']
List.append(List2)
print("\nList after Addition of a List: ")
print(List)
```

Output:

```
Initial blank List:
[]
```

```
List after Addition of Three elements:
[1, 2, 4]

List after Addition of elements from 1-3:
[1, 2, 4, 1, 2, 3]

List after Addition of a Tuple:
[1, 2, 4, 1, 2, 3, (5, 6)]

List after Addition of a List:
[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]
```

Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

- Python3

```
# Python program to demonstrate
# Addition of elements in a List

# Creating a List
List = [1,2,3,4]
print("Initial List: ")
print(List)

# Addition of Element at
# specific Position
# (using Insert Method)
List.insert(3, 12)
```

```
List.insert(0, 'Geeks')
print("\nList after performing Insert
Operation: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4]

List after performing Insert Operation:
['Geeks', 1, 2, 3, 12, 4]
```

--------X---------

The insert function inserts the element at a particular index and then reindexes the list accordingly.

Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, **extend()**, this method is used to add multiple elements at the same time at the end of the list.

Note – append() and extend() methods can only add elements at the end.

- Python3

```
# Python program to demonstrate
# Addition of elements in a List

# Creating a List
List = [1,2,3,4]
print("Initial List: ")
print(List)

# Addition of multiple elements
# to the List at the end
# (using Extend Method)
List.extend([8, 'Geeks', 'Always'])
print("\nList after performing Extend
Operation: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4]

List after performing Extend Operation:
[1, 2, 3, 4, 8, 'Geeks', 'Always']
```

---------X-----------X----------

Using the ‘+’ operator to add elements and extend the list

The ‘+’ operator can be used to extend the list and also add elements to the end of the list.

```
mylist = [1,2,3,4]
newlist = [5,6,7,8]
final_list = mylist + newlist
print(final_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

The above output is similar(not identical) to the given output below.

```
mylist = [1,2,3,4]
newlist = [5,6,7,8]
mylist.extend(newlist)
print(mylist)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Similarly, if we want to add a single element, we can convert that element to a 'one-element list' and add it to the main list.

```
mylist = [1,2,3,4]
item = [5]
final_list = mylist + item
print(final_list)
```

```
[1, 2, 3, 4, 5]
```

Using the '*' operator in the List

The '*' operator can be used to create a repetitive list.
For example :

```
mylist = [1]
print(mylist*4)
newlist = [1,2,3,4]
print(newlist*4)
```

```
[1, 1, 1, 1]
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

However, this technique should not be used in two-dimensional lists.
Explanation through an example below :

```
lst = [[1]*4]*4
lst[1][1] = 0
print(lst)
```

```
[[1, 0, 1, 1], [1, 0, 1, 1], [1, 0, 1, 1], [1, 0, 1, 1]]
```

We created a two dimensional list of all 1-s'. However, when we want to modify only the 1st indexed element of the 1st indexed row, it changes the 1st indexed element of all the rows.
This is because python uses shallow lists and does not create 4 different integer objects, but just 1 integer object. Which results in all the indices of the array pointing to one integer.

Accessing elements from the List

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. The index must be an integer. Nested lists are accessed using nested indexing.

- Python3

```
# Python program to demonstrate
# accessing of element from list

# Creating a List with
# the use of multiple values
List = ["Geeks", "For", "Geeks"]

# accessing a element from the
# list using index number
print("Accessing a element from the list")
print(List[0])
print(List[2])

# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Geeks', 'For'] , ['Geeks']]

# accessing an element from the
# Multi-Dimensional List using
# index number
print("Accessing a element from a
Multi-Dimensional list")
print(List[0][1])
print(List[1][0])
```

Output:

```
Accessing a element from the list
Geeks
Geeks
Accessing a element from a Multi-Dimensional list
```

```
For
Geeks
```

Negative indexing

In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

- Python3

```
List = [1, 2, 'Geeks', 4, 'For', 6,
'Geeks']

# accessing an element using
# negative indexing
print("Accessing element using negative
indexing")

# print the last element of list
print(List[-1])

# print the third last element of list
print(List[-3])
```

Output:

```
Accessing element using negative indexing
Geeks
For
```

Removing Elements from the List

Using remove() method

Elements can be removed from the List by using the built-in **remove()** function but an Error arises if the element doesn't exist in the set. Remove() method only removes one element at a time, to remove a range of elements, the iterator is used. The remove() method removes the specified item.

Note – Remove method in List will only remove the first occurrence of the searched element.

- Python3

```
# Python program to demonstrate
# Removal of elements in a List

# Creating a List
List = [1, 2, 3, 4, 5, 6,
        7, 8, 9, 10, 11, 12]
print("Initial List: ")
print(List)

# Removing elements from List
# using Remove() method
List.remove(5)
List.remove(6)
print("\nList after Removal of two elements:
")
print(List)

# Removing elements from List
# using iterator method
for i in range(1, 5):
    List.remove(i)
print("\nList after Removing a range of
elements: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

List after Removal of two elements:
[1, 2, 3, 4, 7, 8, 9, 10, 11, 12]

List after Removing a range of elements:
[7, 8, 9, 10, 11, 12]
```

Using pop() method

Pop() function can also be used to remove and return an element from the set, but by default it removes only the last element of the set, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

- Python3

```
List = [1,2,3,4,5]

# Removing element from the
# Set using the pop() method
List.pop()
print("\nList after popping an element: ")
print(List)

# Removing element at a
# specific location from the
# Set using the pop() method
List.pop(2)
print("\nList after popping a specific
element: ")
print(List)
```

Output:

```
List after popping an element:
[1, 2, 3, 4]

List after popping a specific element:
[1, 2, 4]
```

---------X----------X

If we want to get the last element of the list and remove it as well, we will pass a variable along with the pop function.

Example :

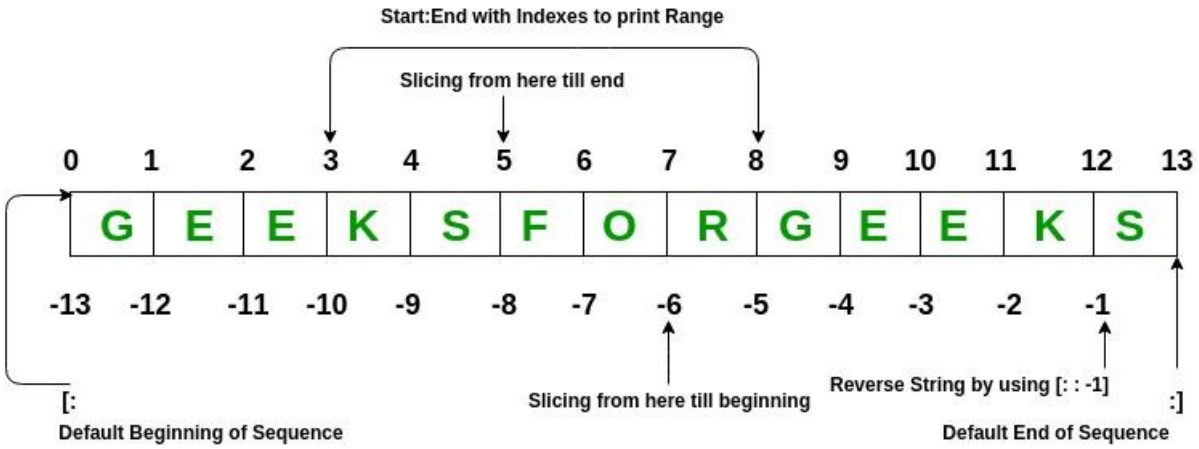
```
mylist = [1,2,3,4,5]
var = mylist.pop()
print('New list is = ',mylist)
print('Popped element is = ',var)
```

```
New list is =  [1, 2, 3, 4]
Popped element is =  5
```

Slicing of a List

In Python List, there are multiple ways to print the whole List with all the elements, but to print a specific range of elements from the list, we use the Slice operation. Slice operation is performed on Lists with the use of a colon(:). To print elements from beginning to a range use [: Index], to print elements from end-use [:-Index], to print elements from specific Index till the end use [Index:], to print elements within a range, use [Start Index:End Index +1] and to print the whole List with the use of slicing operation, use [:]. Further, to print the whole List in reverse order, use [::-1].

Note – To print elements of List from rear-end, use Negative Indexes.


Start:End with Indexes to print Range
Slicing from here till end
0 1 2 3 4 5 6 7 8 9 10 11 12 13
G E E K S F O R G E E K S
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
[:
Default Beginning of Sequence
Slicing from here till beginning
Reverse String by using [: : -1]
:]
Default End of Sequence

- Python3

```
# Python program to demonstrate
# Removal of elements in a List

# Creating a List
List = ['G','E','E','K','S','F',
        'O','R','G','E','E','K','S']
print("Initial List: ")
print(List)

# Print elements of a range
# using Slice operation
Sliced_List = List[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)

# Print elements from a
# pre-defined point to end
Sliced_List = List[5:]
print("\nElements sliced from 5th "
      "element till the end: ")
print(Sliced_List)

# Printing elements from
# beginning till end
Sliced_List = List[:]
print("\nPrinting all elements using slice
operation: ")
print(Sliced_List)
```

Output:

```
Initial List:
['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K',
'S']

Slicing elements in a range 3-8:
['K', 'S', 'F', 'O', 'R']

Elements sliced from 5th element till the end:
['F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
```

```
Printing all elements using slice operation:
['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K',
'S']
```

Negative index List slicing

- Python3

```
# Creating a List
List = ['G','E','E','K','S','F',
        'O','R','G','E','E','K','S']
print("Initial List: ")
print(List)

# Print elements from beginning
# to a pre-defined point using Slice
Sliced_List = List[:-6]
print("\nElements sliced till 6th element from
last: ")
print(Sliced_List)

# Print elements of a range
# using negative index List slicing
Sliced_List = List[-6:-1]
print("\nElements sliced from index -6 to -1")
print(Sliced_List)

# Printing elements in reverse
# using Slice operation
Sliced_List = List[::-1]
print("\nPrinting List in reverse: ")
print(Sliced_List)
```

Output:

```
Initial List:
```

```
['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K',
'S']

Elements sliced till 6th element from last:
['G', 'E', 'E', 'K', 'S', 'F', 'O']

Elements sliced from index -6 to -1
['R', 'G', 'E', 'E', 'K']

Printing List in reverse:
['S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E',
'G']
```

----------X----------X

If we are slicing a range of the list, then it is not necessary to give the correct final index. For example :

```
mylist = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
print(mylist[3:21])
```

```
['K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
```

In this case, we entered a list index from 3 to 21. The list does not contain indices upto 21, so it will automatically assume the value till the maximum available index.

List Comprehension

List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.

A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

Syntax:

newList = [expression(element) for element in oldList if condition]

Example:

- Python3

```
# Python program to demonstrate list
# comprehension in Python

# below list contains square of all
# odd numbers from range 1 to 10
odd_square = [x ** 2 for x in range(1, 11) if x %
2 == 1]
print (odd_square)
```

Output:

```
[1, 9, 25, 49, 81]
```

For better understanding, the above code is similar to –

- Python3

```
# for understanding, above generation is
same as,
odd_square = []

for x in range(1, 11):
    if x % 2 == 1:
        odd_square.append(x**2)

print (odd_square)
```

Output:

```
[1, 9, 25, 49, 81]
```

Operations on List

- Find length of list
  - The len() function gives the length of the list.

```
mylist = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
print(len(mylist))
```

```
13
```

- Concatenating two lists in Python

-Explained through ‘+’ operator and the extend function above.

List Methods

Function Description

Append()	Add an element to the end of the list
Extend()	Add all elements of a list to another list
Insert()	Insert an item at the defined index
Remove()	Removes an item from the list
Pop()	Removes and returns an element at the given index
Clear()	Removes all items from the list
Index()	Returns the index of the first matched item

Count()	Returns the count of the number of items passed as an argument
Sort()	Sort items in a list in ascending order
Reverse()	Reverse the order of items in the list
copy()	Returns a copy of the list

Built-in functions with List

Function	Description
reduce()	apply a particular function passed in its argument to all of the list elements stores the intermediate result and only returns the final summation value

sum()	Sums up the numbers in the list
ord()	Returns an integer representing the Unicode code point of the given Unicode character
cmp()	This function returns 1 if the first list is “greater” than the second list
max()	return maximum element of a given list
min()	return minimum element of a given list
all()	Returns true if all element is true or if the list is empty
any()	return true if any element of the list is true. if the list is empty, return false

len()	Returns length of the list or size of the list
enumerate()	Returns enumerate object of the list
accumulate()	apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
filter()	tests if each element of a list is true or not
map()	returns a list of the results after applying the given function to each item of a given iterable
lambda()	This function can have any number of arguments but only one expression, which is evaluated and returned.

Recent articles on Lists

More videos on Python Lists:

Python List – Set 2

More on Python List –

- Creating a 3D List
- Iterate over a list in Python
- Iterate over multiple lists simultaneously
- Internal working of list in Python
- Python Slicing
- Python List Comprehensions vs Generator Expressions
- List Methods in Python – Set 1 Set 2
- Lambda expression and filter function

Useful Links:

- Recent Articles on Python List
- Python Tutorials
- Python Output Programs in List: Set 6, Set 11, Set 12, Set 13
- Multiple Choice Questions
- All articles in Python Category