Python Lists

Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.
The list is a sequence data type which is used to store the collection of data. Tuples and String are other types of sequence data types.
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.

Creating a List in Python

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 its creation of a list.

Note: *Unlike Sets, the list may contain mutable elements.*

Example 1: Creating a list in Python

```
# Python program to demonstrate

# Creation of 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 = ["Disha", "Computer", "Institute"]

print("\nList Items: ")

print(List[0])

print(List[2])
```

Example 2: 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.

```
# 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, 'Disha', 4, 'Computer', 6, 'Institute']

print("\nList with the use of Mixed Values: ")

print(List)
```

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.

Example 1: Accessing elements from list

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

# Creating a List with the use of multiple values

List = ["Disha", "Computer", "Institute"]

# accessing a element from the list using index number

print("Accessing a element from the list")

print(List[0])

print(List[2])
```

Example 2: Accessing elements from a multi-dimensional list

```
# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Disha', 'Computer'], ['Institute']]


# 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])
```

Negative indexing

```
List = [1, 2, 'Disha', 4, 'Computer', 6, 'Institute']


# 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])
```

Getting the size of Python list
Python len() is used to get the length of the list.

```
        # Creating a List
        List1 = []
        print(len(List1))
        # Creating a List of numbers
        List2 = [10, 20, 14]
        print(len(List2))
```

Taking Input of a Python List
We can take the input of a list of elements as string, integer, float, etc. But the default one is a string.

Example 1:

```
# Python program to take space separated input as a string split and store it to a list and print the string list

# input the list as string

string = input("Enter elements (Space-Separated): ")

lst = string.split()

print('The list is:', lst) # printing the list
```

Example 2:

```
# input size of the list

n = int(input("Enter the size of list : "))

# store integers in a list using map,

# split and strip functions

lst = list(map(int, input("Enter the integer\

elements:").strip().split()))[:n]


# printing the list

print('The list is:', lst)
```

Adding Elements to a Python List

Method 1: 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.

```
#Python program to demonstrate  Addition of elements in a List
        List = []
        print("Initial blank List: ")
        print(List)
        List.append(1)
        List.append(2)
        List.append(4)
        print("\nList after Addition of Three elements: ")
        print(List)
for i in range(1, 4):
                List.append(i)
        print("\nList after Addition of elements from 1-3: ")
        print(List)

        List.append((5, 6))
        print("\nList after Addition of a Tuple: ")
        print(List)

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

Method 2: 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).

```
List = [1,2,3,4]
print("Initial List: ")
```

```
print(List)

List.insert(3, 12)

List.insert(0, 'Geeks')

print("\nList after performing Insert Operation: ")

print(List)
```

Method 3: 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.*

```
# Python program to demonstrate  Addition of elements in 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, 'Smile', 'Always'])

        print("\nList after performing Extend Operation: ")

        print(List)
```

Reversing a List

A list can be reversed by using the reverse() method in Python.

```
# Reversing a list

mylist = [1, 2, 3, 4, 5, 'Java', 'Python']

mylist.reverse()

print(mylist)
```

Removing Elements from the List

Method 1: 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

list. 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.*
Example 1:

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

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

print("Initial List: ")

print(List)

List.remove(5)

List.remove(6)

print("\nList after Removal of two elements: ")

print(List)
```

Method 2: Using pop() method

pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, 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.

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

List.pop()

print("\nList after popping an element: ")

print(List)

List.pop(2)

print("\nList after popping a specific element: ")

print(List)
```

Slicing of a List

```
# Python program to demonstrate Removal of elements in a List
List = ['D', 'I', 'S', 'H', 'A', 'C','O', 'M', 'P', 'U', 'T', 'E', 'R']
print("Initial List: ")
print(List)
Sliced_List = List[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)
Sliced_List = List[5:]
print("\nElements sliced from 5th ""element till the end: ")
print(Sliced_List)
Sliced_List = List[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_List)
```

List Comprehension

Python 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]

```
# 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)
```

```
        # 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)
```