

Python



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.

```
Var = [ "Geeks" , "for" , "Geeks" ]  
print ( Var )
```

Output:

```
["Geeks" , "for" , "Geeks"]
```

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.

```
# 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])
```

Output

Blank List:

[]

List of numbers:

[10, 20, 14]

List Items:

Geeks

Geeks

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 = ["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])
```

Output

```
Accessing a element from the list
Geeks
Geeks
```

Example 2: Accessing elements from a multi-dimensional list

Python3

```
# 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 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
```


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:

Python3

```
# 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): ")

# split the strings and store it to a list
lst = string.split()

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

Output:

Enter elements: GEEKS FOR GEEKS

The list is: ['GEEKS', 'FOR', 'GEEKS']

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

# 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']]
```

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).

```
# 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]

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

# 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)
```


Initial List:

[1, 2, 3, 4]

List after performing Extend Operation:

[1, 2, 3, 4, 8, 'Geeks', 'Always']

Reversing a List

A list can be reversed by using the [reverse\(\) method in Python](#).

Python3

```
# Reversing a list
mylist = [1, 2, 3, 4, 5, 'Geek', 'Python']
mylist.reverse()
print(mylist)
```

Output

```
['Python', 'Geek', 5, 4, 3, 2, 1]
```

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.

```
# 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)
```

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]
```

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

```
List after Removing a range of elements:
[5, 6, 7, 8, 9, 10, 11, 12]
```

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]

# 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]

Slicing of a List

We can get substrings and sublists using a slice. 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 a specific Index till the end use

```
[Index:]
```

To print the whole list in reverse order, use

```
[::-1]
```


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)
```

Output

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

List Methods

Function	Description
<u>Append()</u>	Add an element to the end of the list
<u>Extend()</u>	Add all elements of a list to another list
<u>Insert()</u>	Insert an item at the defined index
<u>Remove()</u>	Removes an item from the list
<u>Clear()</u>	Removes all items from the list
<u>Index()</u>	Returns the index of the first matched item
<u>Count()</u>	Returns the count of the number of items passed as an argument
<u>Sort()</u>	Sort items in a list in ascending order
<u>Reverse()</u>	Reverse the order of items in the list
<u>copy()</u>	Returns a copy of the list

Built-in functions with List

Function	Description
<code>reduce()</code>	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
<code>sum()</code>	Sums up the numbers in the list
<code>ord()</code>	Returns an integer representing the Unicode code point of the given Unicode character
<code>cmp()</code>	This function returns 1 if the first list is “greater” than the second list
<code>max()</code>	return maximum element of a given list
<code>min()</code>	return minimum element of a given list
<code>all()</code>	Returns true if all element is true or if the list is empty
<code>any()</code>	return true if any element of the list is true. if the list is empty, return false
<code>len()</code>	Returns length of the list or size of the list
<code>enumerate()</code>	Returns enumerate object of the list
<code>accumulate()</code>	apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
<code>filter()</code>	tests if each element of a list is true or not
<code>map()</code>	returns a list of the results after applying the given function to each item of a given iterable
<code>lambda()</code>	This function can have any number of arguments but only one expression, which is evaluated and returned.