1 Python Lists

Note: This is a copy of the GeeksforGeeks tutorial on Python lists as a Jupyter Notebook. All information contained can be found here: https://www.geeksforgeeks.org/python-lists/?ref=lbp

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

1.1 Example of list in Python

Here we are creating Python **List** using [].

[1]:
```
Var = ["Geeks", "for", "Geeks"]
print("Output:\n")
print(Var)
```

```
Output:

['Geeks', 'for', 'Geeks']
```

Lists are the simplest containers that are an integral part of the Python language. 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.

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**

[2]:
```
# Python program to demonstrate
# Creation of List

print("Output:\n")

# 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
```

Complexities for Creating Lists
Time Complexity: O(1)

Space Complexity: O(n) ### **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.

[3]:
```
# Creating a List with
# the use of Numbers
# (Having duplicate values)

print("Output:\n")

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']
```

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.

1.1.1 Example 1: Accessing elements from list

[5]:
```python
# Python program to demonstrate
# accessing of element from list

print("Output:\n")

# 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
```

1.1.2 Example 2: Accessing elements from a multi-dimensional list

[7]:
```python
# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Geeks', 'For'], ['Geeks']]

print("Output:\n")

# 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
```

1.1.3 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.

[9]:
```python
List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']

print("Output:\n")

# 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
```

Complexities for Accessing elements in a Lists:
Time Complexity: O(1)

Space Complexity: O(1)

Getting the size of Python list

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

[10]:
```python
# Creating a List

print("Output:\n")

List1 = []
print(len(List1))

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

```
Output:

0
3
```

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.

1.1.4 Example 1:

[14]:
```python
# Python program to take space
# separated input as a string
# split and store it to a list
# and print the string list

print("Output:\n")

# 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 (Space-Separated):  a b c d e

The list is: ['a', 'b', 'c', 'd', 'e']
```

1.1.5 Example 2:

[18]:
```python
print("Output:\n")

# 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)
```

```
Output:

Enter the size of list :  3
Enter the integer elements: 123 2 3
```

```
The list is: [123, 2, 3]
```

Adding Elements to a Python List

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.

[20]:
```
# Python program to demonstrate
# Addition of elements in a List

print("Output:\n")

# 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']]
```

Complexities for Adding elements in a Lists(append() method):
Time Complexity: O(1)

Space Complexity: O(1)

1.1.6 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).

[21]:
```python
# Python program to demonstrate
# Addition of elements in a List

print("Output:\n")

# 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]
```

Complexities for Adding elements in a Lists(insert() method):
Time Complexity: O(n)

Space Complexity: O(1)

1.1.7 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.*

[22]:
```python
# Python program to demonstrate
# Addition of elements in a List

print("Output:\n")

# 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']
```

Complexities for Adding elements in a Lists(extend() method):
Time Complexity: O(n)

Space Complexity: O(1)

Reversing a List

1.1.8 Method 1: A list can be reversed by using the reverse() method in Python.

[23]:
```python
# Reversing a list

print("Output:\n")
```

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

```
Output:

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

1.1.9 Method 2: Using the reversed() function:

The reversed() function returns a reverse iterator, which can be converted to a list using the list() function.

[24]:
```
print("Output:\n")

my_list = [1, 2, 3, 4, 5]
reversed_list = list(reversed(my_list))
print(reversed_list)
```

```
Output:

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

Removing Elements from the List

1.1.10 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.*

1.1.11 Example 1:

[25]:
```
# Python program to demonstrate
# Removal of elements in a List

print("Output:\n")

# 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]
```

1.1.12 Example 2:

[26]:
```
# Creating a List
List = [1, 2, 3, 4, 5, 6,
                7, 8, 9, 10, 11, 12]

print("Output:\n")

# 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]
```

Complexities for Deleting elements in a Lists(remove() method):
Time Complexity: O(n)

Space Complexity: O(1)

1.1.13 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.

[28]:
```
List = [1, 2, 3, 4, 5]

print("Output:\n")

# Removing element from the
# Set using the pop() method
List.pop()
print("List 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]
```

Complexities for Deleting elements in a Lists(pop() method):
Time Complexity: O(1)/O(n) (O(1) for removing the last element, O(n) for removing the first and middle elements)

Space Complexity: O(1)

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

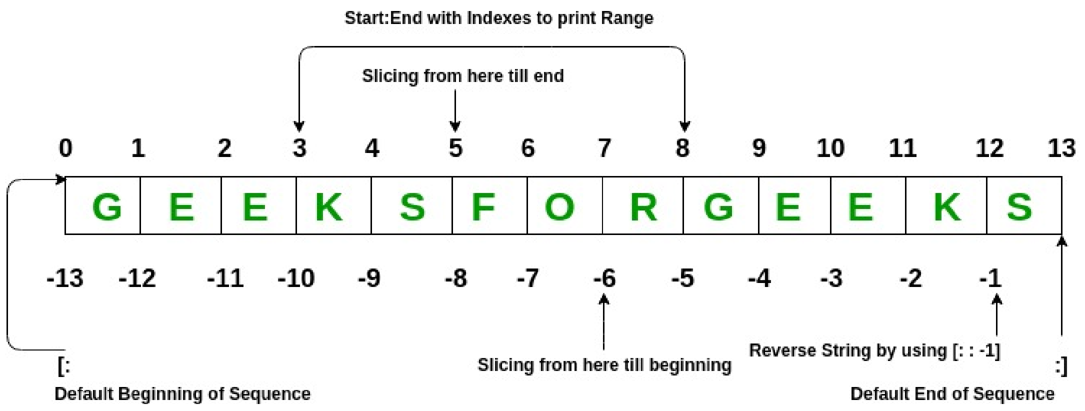
[Index:]

To print the whole list in reverse order, use

[::-1]

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

[2]:
```
from IPython.display import display, Image
display(Image(filename="ListSlicing.jpg", height=400, width=800))
```


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

UNDERSTANDING SLICING OF LISTS:

- pr[0] accesses the first item, 2.
- pr[-4] accesses the fourth item from the end, 5.
- pr[2:] accesses [5, 7, 11, 13], a list of items from third to last.
- pr[:4] accesses [2, 3, 5, 7], a list of items from first to fourth.
- pr[2:4] accesses [5, 7], a list of items from third to fifth.
- pr[1::2] accesses [3, 7, 13], alternate items, starting from the second item.

[29]:
```
# Python program to demonstrate
# Removal of elements in a List

print("Output:\n")

# 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']
```

1.1.14 Negative index List slicing

[30]:
```
print("Output:\n")

# 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']
```

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]

Example:

[31]:
```python
# Python program to demonstrate list
# comprehension in Python

print("Output:\n")

# 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 as follows:

[32]:
```python
# for understanding, above generation is same as,

print("Output:\n")

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]
```

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
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
pop()	Removes and returns the item at the specified index. If no index is provided, it removes and returns the last item.

The operations mentioned above modify the list Itself.

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

Function	Description
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.

[ ]:

```
```