15. Python - List Data Structure

Table of Contents

15. Python - List Data Structure

1. Why should we learn about data structures?

- ✓ The common requirement in any real time project is like, creating, updating, retrieving, and deleting elements.
- ✓ Few more real times operations are below,

    - Storing
    - Searching
    - Retrieving
    - Deleting
    - Processing

    - Duplicate

    - Ordered
    - Unordered

    - Size
    - Capacity

    - Sorting
    - Un-sorting
    - Random access

    - Keys
    - Values
    - Key – value pairs

- ✓ So, to understand above operations where to use and how to use then we need to learn about data structures.

2. Python Data structures

- ✓ If you wanted to store a group of individual objects in a single entity, then you should go for data structures.

3. Sequence of elements

- ✓ Data structure also called as sequence.
- ✓ A sequence is a datatype that can contains a group of elements.
- ✓ The purpose of any sequence is, to store and process a group of elements.
- ✓ In python, strings, lists, tuples, set and dictionaries are very important sequence datatype.

4. list data structure

- ✓ We can **create** list by using,
  - square brackets **[]** symbols
  - list() predefined function.

- ✓ A list can **store** group of objects or elements.
  - A list can store **same** (Homogeneous) type of elements.
  - A list can store **different** type (Heterogeneous) of elements.

- ✓ A list **size** will increase dynamically.

- ✓ In list **insertion** order is preserved or **fixed**.
  - If we insert elements into 10, 20, 30 then output also will display as 10, 20, 30 then this is called as insertion order is preserved or fixed
  - Example

Input	=> [10, 20, 30]
Output	=> [10, 20, 30]

- ✓ **Duplicate** elements are allowed.
- ✓ List having **mutable** nature.
  - Mutable means once we create a list object then we can change or modify the content of list object.

- ✓ Store elements by using **index**.
  - A list data structure supports both positive and negative indexes.
  - Positive index means from left to right
  - Negative index means right to left

Note:

- ✓ Inside list every object can be separated by comma separator.

Make a note

- ✓ list is a predefined class in python.
- ✓ Once if we create list object means internally object is creating for list class.
- ✓ What is a class how class works, we will learn in OOPs chapter.

5. Creating list

- ✓ We can create list by using square brackets **[]**
- ✓ Inside list, elements will be separated by comma separator

5.1. Creating empty list

- ✓ An empty list is valid

Program Name: Creating empty list
demo1.py

```
a = []
print(a)
print(type(a))
```

Output

```
[]
<class 'list'>
```

5.2. Creating list with elements

- ✓ We can create list directly with elements.

Program Name: Creating list with same type of elements
demo2.py

```
numbers = [10, 20, 30, 40]
print(numbers)
```

Output

```
[10, 20, 30, 40]
```

Program Name

Creating list with same type of elements with duplicates

demo3.py

```
numbers = [10, 20, 30, 40, 10, 20, 30, 40]
print(numbers)
```

Output

```
[10, 20, 30, 40, 10, 20, 30, 40]
```

Program Name

creating list with same type of elements

demo4.py

```
names = ["Daniel", "Prasad", "Ramesh", "Daniel"]
print(names)
```

Output

```
["Daniel", "Prasad", "Ramesh", "Daniel"]
```

Program Name

Creating list with different type of elements

demo5.py

```
student_info = ["Daniel", 10, 35.9]
print(student_info)
```

Output

```
["Daniel", 10, 35.9]
```

Note

- ✓ Observe the above programs output,
  - Order is preserved
  - Duplicates are allowed.

5.3. Creating list by using list(p) predefined function

- ✓ list(p) is a predefined function in python.
- ✓ By using this function we can create list.
- ✓ list(p) function takes only one parameter.
- ✓ This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

Program Creating list by using list(p) function
Name demo6.py

```
r = range(0, 10)
a = list(r)
print(a)
```

output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

6. list having mutable nature

- ✓ Once we created a list object then we can change or modify the elements in the existing list object.
- ✓ So, list having mutable nature.

Program Name: list having mutable nature
demo7.py

```
a = [1, 2, 3, 4, 5]
print(a)
print("Before modifying a[0] : ", a[0])

a[0] = 20
print("After modifying a[0] : ", a[0])
print(a)
```

output

```
[1, 2, 3, 4, 5]
Before modifying a[0] :  1
After modifying a[0] :  20
[20, 2, 3, 4, 5]
```

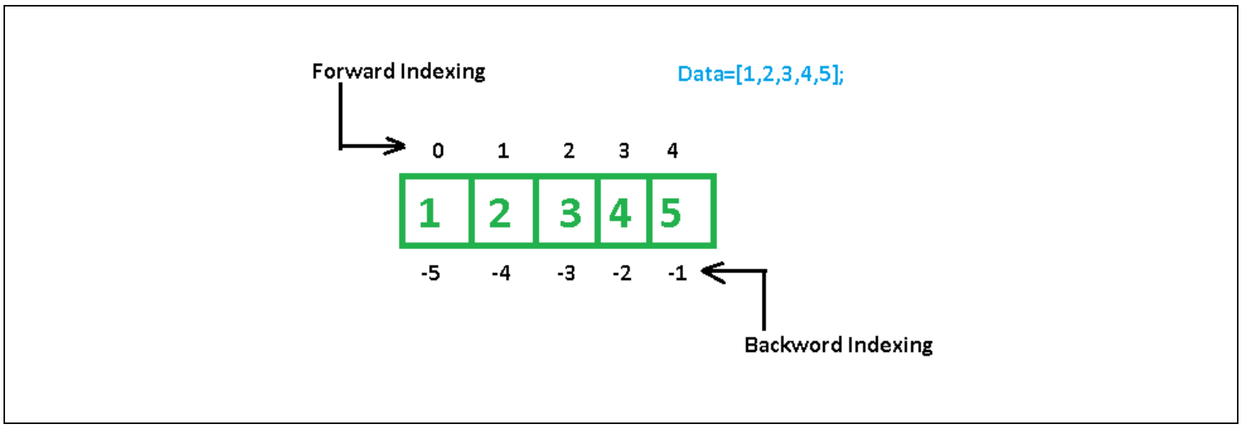
7. Accessing elements from list

- ✓ We can access elements from list by using,

    1. Index.
    2. Slice operator.
    3. loops

7.1. By using index

- ✓ index represents accessing the elements by their position numbers in the list.



- ✓ Indexing represents accessing the elements by their position numbers in the list.
- ✓ Index starts from 0 onwards.

- ✓ List supports both positive and negative indexes.
    - Positive index represents from left to right direction
    - Negative index represents from right to left.

- ✓ If we are trying to access beyond the range of list index, then we will get error like IndexError.

Program Name

list indexing
demo8.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))
```

output

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
```

Program Name	list indexing
demo9.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))

print(type(names[0]))
print(type(names[1]))
print(type(names[2]))
```

output

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
<class 'str'>
<class 'str'>
<class 'str'>
```

Program Name

IndexError: list index out of range
demo10.py

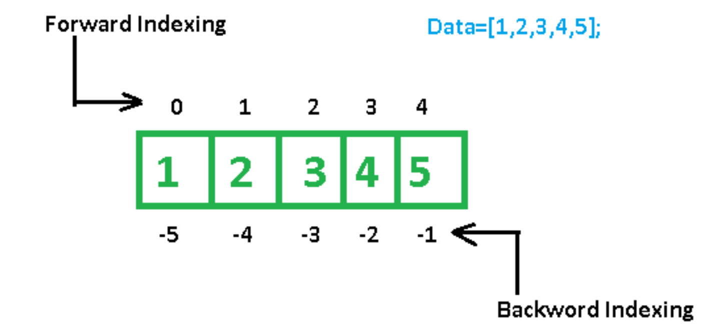
```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)
print(names[30])
```

output

```
['Daniel', 'Prasad', 'Ramesh']
IndexError: list index out of range
```

7.2. Slicing

- ✓ Slicing represents extracting a piece of the list from already created list



Syntax

[start: stop: stepsize]

- ✓ start
  - o It indicates the index where slice can start.
  - o Default value is 0
- ✓ stop
  - o It indicates the index where slice can end.
  - o Default value is max allowed index of list i.e. length of the list
- ✓ Step size
  - o Increment value.
  - o Default value is 1

Program Name

Slice example
demo11.py

```
n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(n)
print(n[:])
print(n[::])
print(n[0:5:])
```

output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
```

7.3. Accessing list by using for loop

- ✓ We can access elements from list by using for loop.

Program Name: accessing elements from list by using for loop
demo12.py

```
values = [100, 200, 300, 400]

for value in values:
        print(value)
```

output

```
100
200
300
400
```

8. len(p) function

- ✓ By using len(p) predefined function we can find the length of list.
- ✓ This function returns the number of elements present in the list.

Program To find length of list
Name demo13.py

```
values = [10, 20, 30, 40, 50]
print(len(values))
```

Output

```
5
```

9. Methods in list data structure

- ✓ As discussed, list is a predefined class.
- ✓ So, list class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using dir(parameter1) predefined function.

- ✓ So, internally list class contains two types of methods,

    - With underscore symbol methods.
        - We no need to focus

    - Without underscore symbol methods.
        - We need to focus much on these

Program Name	Printing list data structure methods by using dir(list) function demo14.py

```
print(dir(list))
```

output

```
[

'__add__', ...... , '__subclasshook__',

Important methods

'append', 'count', 'insert', 'remove', 'reverse', 'sort'

]
```

Important point

- ✓ As per object-oriented principle,
    - If we want to access instance methods, then we should access by using object name.
- ✓ So, all list methods we can access by using list object.

Important methods in list

- ✓ count(p) method
- ✓ append(p) method
- ✓ insert() method
- ✓ remove() method
- ✓ reverse() method
- ✓ sort() method

9.1. count(p)method

- ✓ count(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method returns the number of occurrences of specific value in the list.

Program	To find count of specific value in list
Name	demo15.py

```
n = [1, 2, 3, 4, 5, 5, 5, 3]
print(n.count(5))
print(n.count(2))
```

output

```
3
1
```

9.2. append(p)method

- ✓ append(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method adds object or element to the existing list object.
- ✓ This method will add the object to list at the end of the list.

Program Name appending elements into list
demo16.py

```
a = []
a.append(10)
a.append(20)
a.append(30)
print(a)
```

output

```
[10, 20, 30]
```

Program Name appending elements into list
demo17.py

```
a = [10, 20, "Daniel"]

a.append("Naresh")
a.append("Veeru")
print(a)
```

output

```
[10, 20, "Daniel", "Naresh", "Veeru"]
```

9.3. insert(p1, p2) method:

- ✓ insert(p1, p2) is a predefined method in list class.
- ✓ We should access this method by using list object.
- ✓ By using this method we can insert value at specific position in list.

Program Name	inserting elements into list demo18.py

```
a = [10, 20, 30, 40, 50]

a.insert(0, 76)
print(a)
```

output

```
[76, 10, 20, 30, 40, 50]
```

append(element)	**insert(index, element)**
✓ This method adds element at last position.	✓ This method adds element at specific index position.

9.4. remove(p) method:

- ✓ remove(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can remove value from list.

Program Removing element from list
Name demo19.py

```
a = [10, 20, 30]

a.remove(10)
print(a)
```

output

```
[20, 30]
```

Ordering elements of List:

9.5. reverse():

- ✓ reverse() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can reverse values in list.

Program Name: reverse of the list
demo20.py

```
a = [10, 20, 30, 40]

print(a)
a.reverse()
print(a)
```

output

```
[10, 20, 30, 40]
[40, 30, 20, 10]
```

9.6. sort() method:

- ✓ sort() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By default insertion order is fixed.
- ✓ By using this method we can sort values in list.
  - For numbers the order is ascending order.
  - For strings the order is alphabetical order

Program sorting the numbers and names
Name demo21.py

```
a = [10, 40, 50, 20, 30]
a.sort()
print(a)

b = ['Daniel', 'Ramesh', 'Arjun']
b.sort()
print(b)
```

output

```
[10, 20, 30, 40, 50]
['Arjun', 'Daniel', 'Ramesh']
```

10. Mathematical + and * operators

10.1. Concatenation operator +

- ✓ '+' operator concatenate two list objects to join them and returns single list.

Program Name	+ operator concatenates the lists demo22.py

```
a = [10, 20, 30]
b = [40, 50, 60]
c = a + b

print(c)
```

output

```
[10, 20, 30, 40, 50, 60]
```

10.2 Repetition operator *

- ✓ **'*'** operator works to repetition of elements in the list.

Program Name	* operator repetition the lists demo23.py

```
a = [10, 20, 30]

print(a)
print(a*2)
```

output

```
[10, 20, 30]
[10, 20, 30, 10, 20, 30]
```

11. Membership operators

- ✓ We can check if the element is a member of a list or not by using membership operators those are,
  - **in** operator
  - **not in** operator
- ✓ If the element is member of list, then **in** operator returns True otherwise False.
- ✓ If the element is not in the list, then **not in** operator returns True otherwise False

Program Name: Membership operators
demo24.py

```
a = [10, 20, 30, 40, 50]

print(20 in a)          # True
print(20 not in a)      # False

print(90 in a)          # False
print(90 not in a)      # True
```

output

```
True
False
False
True
```

12. list comprehension

- ✓ List comprehensions represents creating new lists from Iterable object like a list, set, tuple, dictionary and range.
- ✓ List comprehension takes input as iterable, we can apply conditional logic on every item and returns new list.
- ✓ List comprehensions code is very concise way.

Syntax

```
list = [expression for item1 in iterable1 if statement]
```

- ✓ Here Iterable represents a list, set, tuple, dictionary or range object.
- ✓ The result of list comprehension is new list based on the applying conditions.

Program Name: list comprehension example
demo25.py

```
values = [10, 20, 30]
result = [value+2 for value in values]

print(values)
print(result)
```

output

```
[10, 20, 30]
[12, 22, 32]
```

Program Name

list comprehension example
demo26.py

```
values = [10, 20, 30]
result = [value*3 for value in values]

print(values)
print(result)
```

output

```
[10, 20, 30]
[30, 60, 90]
```

Program Name

list comprehension example
demo27.py

```
values = [10, 20, 30, 40, 50, 60, 70, 80, 90]
result = [value for value in values if value <= 50]

print(values)
print(result)
```

output

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
[10, 20, 30, 40, 50]
```

Program Name	square numbers from 1 to 10 by using list comprehension demo28.py

```
values = range(1, 11)
squares = [value*2 for value in values]
print(squares)
```

output

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```