Overview of Python Collections: List: Fundamental List Operations, Accessing, Updating, Deleting, Indexing, Slicing and Built-in list methods in Python.

List:

A list is a collection of different kinds of values or items. Since Python lists are mutable, we can change their elements after forming. The comma (,) and the square brackets [enclose the List's items] serve as separators.

List Declaration

```
# a simple list
list1 = [1, 2, "Python", "Program", 15.9]
list2 = ["Amy", "Ryan", "Henry", "Emma"]

# printing the list
print(list1)
print(list2)

# printing the type of list
print(type(list1))
print(type(list2))
```

Characteristics of Lists

The characteristics of the List are as follows:

- The lists are in order.
- The list element can be accessed via the index.
- The mutable type of List is
- The rundowns are changeable sorts.
- The number of various elements can be stored in a list.

Ordered List Checking

Code

```
# example
a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
b = [ 1, 2, 5, "Ram", 3.50, "Rahul", 6 ]
a == b
```

Output:

```
False
```

The indistinguishable components were remembered for the two records; however, the subsequent rundown changed the file position of the fifth component, which is against the rundowns' planned request. False is returned when the two lists are compared.

Code

```
# example
a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
b = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
a == b
```

Output:

```
True
```

Records forever protect the component's structure. Because of this, it is an arranged collection of things.

Let's take a closer look at the list example.

Code

```
# list example in detail
emp = [ "John", 102, "USA"]
Dep1 = [ "CS",10]
Dep2 = [ "IT",11]
HOD_CS = [ 10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data ...")
print(" Name : %s, ID: %d, Country: %s" %(emp[0], emp[1], emp[2]))
print("printing departments ...")
print("Department 1:\nName: %s, ID: %d\n Department 2:\n Name: %s, ID: %s"%( Dep1[0], Dep2[1], Dep2[0], Dep2[1]))
print("HOD Details ....")
print("CS HOD Name: %s, Id: %d" %(HOD_CS[1], HOD_CS[0]))
print("IT HOD Name: %s, Id: %d" %(HOD_IT[1], HOD_IT[0]))
print(type(emp), type(Dep1), type(Dep2), type(HOD_CS), type(HOD_IT))
```

Output:

```
printing employee data...
Name : John, ID: 102, Country: USA
printing departments...
Department 1:
Name: CS, ID: 11
```

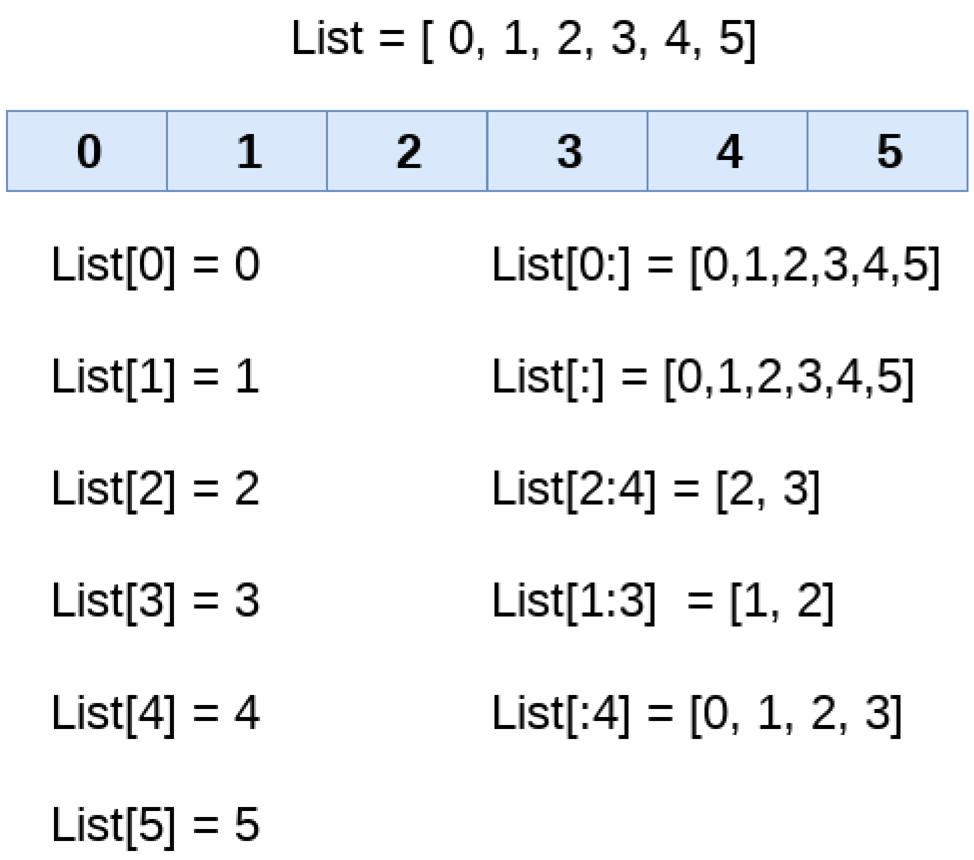
```
Department 2:
Name: IT, ID: 11
HOD Details ....
CS HOD Name: Mr. Holding, Id: 10
IT HOD Name: Mr. Bewon, Id: 11
<class ' list '> <class ' list '> <class ' list '> <class ' list '> <class ' list '>
```

In the preceding illustration, we printed the employee and department-specific details from lists that we had created. To better comprehend the List's concept, look at the code above.

List Indexing and Splitting

The indexing procedure is carried out similarly to string processing. The slice operator [] can be used to get to the List's components.

The index ranges from 0 to length -1. The 0th index is where the List's first element is stored; the 1st index is where the second element is stored, and so on.



```
List = [ 0, 1, 2, 3, 4, 5]

| 0 | 1 | 2 | 3 | 4 | 5 |

List[0] = 0          List[0:] = [0,1,2,3,4,5]

List[1] = 1          List[:] = [0,1,2,3,4,5]

List[2] = 2          List[2:4] = [2, 3]

List[3] = 3          List[1:3]  = [1, 2]

List[4] = 4          List[:4] = [0, 1, 2, 3]

List[5] = 5
```

We can get the sub-list of the list using the following syntax.

1. list_varible(start:stop:step)

- The beginning indicates the beginning record position of the rundown.
- The stop signifies the last record position of the rundown.
- Within a start, the step is used to skip the nth element: stop.

The start parameter is the initial index, the step is the ending index, and the value of the end parameter is the number of elements that are "stepped" through. The default value for the step is one without a specific value. Inside the resultant Sub List, the same with record start would be available, yet the one with the file finish will not. The first element in a list appears to have an index of zero.

Consider the following example:

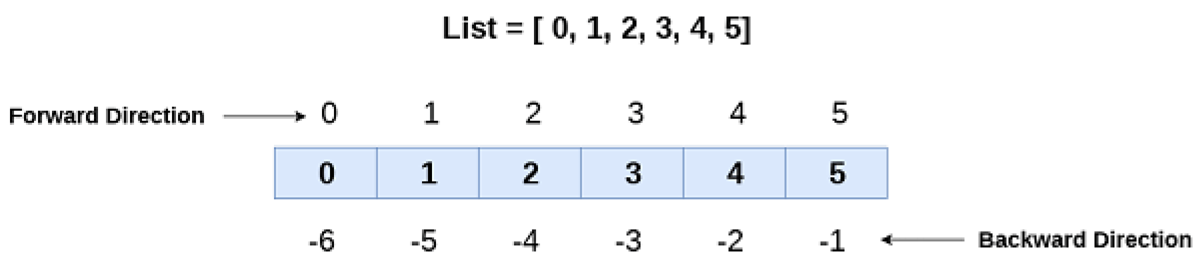
Code

```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
# Slicing the elements
print(list[0:6])
# By default, the index value is 0 so its starts from the 0th element and go for index -1.
print(list[:])
print(list[2:5])
print(list[1:6:2])
```

Output:

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

In contrast to other programming languages, Python lets you use negative indexing as well. The negative indices are counted from the right. The index -1 represents the final element on the List's right side, followed by the index -2 for the next member on the left, and so on, until the last element on the left is reached.



Let's have a look at the following example where we will use negative indexing to access the elements of the list.

Code

```
# negative indexing example
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

Output:

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

Negative indexing allows us to obtain an element, as previously mentioned. The rightmost item in the List was returned by the first print statement in the code above. The second print statement returned the sub-list, and so on.

Updating List Values

Due to their mutability and the slice and assignment operator's ability to update their values, lists are Python's most adaptable data structure. Python's append() and insert() methods can also add values to a list.

Consider the following example to update the values inside the List.

Code

```
# updating list values
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to the second index
list[2] = 10
print(list)
# Adding multiple-element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
```

The list elements can also be deleted by using the **del** keyword. Python also provides us the **remove()** method if we do not know which element is to be deleted from the list.

Consider the following example to delete the list elements.

Code

```
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to second index
list[2] = 10
print(list)
# Adding multiple element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
```

Python List Operations

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings. The different operations of list are

1. Repetition
2. Concatenation
3. Length
4. Iteration
5. Membership

Let's see how the list responds to various operators.

1. Repetition

The redundancy administrator empowers the rundown components to be rehashed on different occasions.

Code

```
# repetition of list
# declaring the list
list1 = [12, 14, 16, 18, 20]
# repetition operator *
l = list1 * 2
print(l)
```

Output:

```
[12, 14, 16, 18, 20, 12, 14, 16, 18, 20]
```

2. Concatenation

It concatenates the list mentioned on either side of the operator.

Code

```
# concatenation of two lists
# declaring the lists
list1 = [12, 14, 16, 18, 20]
list2 = [9, 10, 32, 54, 86]
# concatenation operator +
l = list1 + list2
print(l)
```

Output:

```
[12, 14, 16, 18, 20, 9, 10, 32, 54, 86]
```

3. Length

It is used to get the length of the list

Code

```
# size of the list
# declaring the list
list1 = [12, 14, 16, 18, 20, 23, 27, 39, 40]
# finding length of the list
```

```
len(list1)
```

Output:

```
9
```

4. Iteration

The for loop is used to iterate over the list elements.

Code

```
# iteration of the list
# declaring the list
list1 = [12, 14, 16, 39, 40]
# iterating
for i in list1:
    print(i)
```

Output:

```
12
14
16
39
40
```

5. Membership

It returns true if a particular item exists in a particular list otherwise false.

Code

```
# membership of the list
# declaring the list
list1 = [100, 200, 300, 400, 500]
# true will be printed if value exists
# and false if not

print(600 in list1)
print(700 in list1)
print(1040 in list1)

print(300 in list1)
print(100 in list1)
print(500 in list1)
```

Output:

```
False
False
False
True
True
True
```

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings, which can be iterated as follows.

Code

```python
# iterating a list
list = ["John", "David", "James", "Jonathan"]
for i in list:
    # The i variable will iterate over the elements of the List and contains each element in each iteration.
    print(i)
```

Output:

```
John
David
James
Jonathan
```

Adding Elements to the List

The append() function in Python can add a new item to the List. In any case, the annex() capability can enhance the finish of the rundown.

Consider the accompanying model, where we take the components of the rundown from the client and print the rundown on the control center.

Code

```python
#Declaring the empty list
l =[]
#Number of elements will be entered by the user
n = int(input("Enter the number of elements in the list:"))
# for loop to take the input
for i in range(0,n):
    # The input is taken from the user and added to the list as the item
    l.append(input("Enter the item:"))
```

```python
print("printing the list items..")
# traversal loop to print the list items
for i in l:
    print(i, end = "  ")
```

Output:

```
Enter the number of elements in the list:10
Enter the item:32
Enter the item:56
Enter the item:81
Enter the item:2
Enter the item:34
Enter the item:65
Enter the item:09
Enter the item:66
Enter the item:12
Enter the item:18
printing the list items..
32  56  81  2  34  65  09  66  12  18
```

Removing Elements from the List

The remove() function in Python can remove an element from the List. To comprehend this idea, look at the example that follows.

Example -

Code

```python
list = [0,1,2,3,4]
print("printing original list: ");
for i in list:
    print(i,end=" ")
list.remove(2)
print("\nprinting the list after the removal of first element...")
for i in list:
    print(i,end=" ")
```

Output:

```
printing original list:
0 1 2 3 4
printing the list after the removal of first element...
0 1 3 4
```

Python List Built-in Functions

Python provides the following built-in functions, which can be used with the lists.

1. len()
2. max()
3. min()

len( )

It is used to calculate the length of the list.

Code

```python
# size of the list
# declaring the list
list1 = [12, 16, 18, 20, 39, 40]
# finding length of the list
len(list1)
```

Output:

```
6
```

Max( )

It returns the maximum element of the list

Code

```python
# maximum of the list
list1 = [103, 675, 321, 782, 200]
# large element in the list
print(max(list1))
```

Output:

```
782
```

Min( )

It returns the minimum element of the list

Code

```python
# minimum of the list
list1 = [103, 675, 321, 782, 200]
```

```
# smallest element in the list
print(min(list1))
```

Output:

```
103
```

Let's have a look at the few list examples.

Example: 1- Create a program to eliminate the List's duplicate items.

Code

```
list1 = [1,2,2,3,55,98,65,65,13,29]
# Declare an empty list that will store unique values
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

Output:

```
[1, 2, 3, 55, 98, 65, 13, 29]
```

Example:2- Compose a program to track down the amount of the component in the rundown.

Code

```
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
    sum = sum+i
print("The sum is:",sum)
```

Output:

```
The sum is: 67
In [8]:
```

Example: 3- Compose the program to find the rundowns comprise of somewhere around one normal component.

Code

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
```

```python
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

Output:

```
The common element is: 2
```