



# python

Course Id :INT 213

# List

A list is a linear data structure, meaning that its elements have a linear ordering. That is, there is a first element, a second element, and so on. Figure depicts a list storing the marks for each student of a given class having 7 students, in which each item is linearly stored and each item in the list is identified by its index value. The location at index 0 stores the marks of first student (roll no.1), the location at index 1 stores the marks of next student, and so on. It is customary in programming languages to begin numbering sequences of items with an index value of 0 rather than 1. This is referred to as zero-based indexing.

Index	Marks
0	78
1	67
2	89
3	46
4	90
5	67
6	91

# List

Operations commonly performed on lists include retrieve, update, insert, delete (remove) and append. Figure below depicts these operations on a list of integers.

**a.) Retrieve** - Get marks at index 2 (89)

Index	Marks
0	78
1	64
2	<b>89</b>
3	46
4	90
5	67
6	91

**b.) Replace**- Update marks at index 0 with 79

(Marks changed from 78 to 79)

Index	Marks
0	<b>79</b>
1	64
2	89
3	46
4	90
5	67
6	91

**c.) Insert** – Insert marks 77 at index 2

Index	Marks
0	79
1	64
2	<b>77</b>
3	89
4	46
5	90
6	67
7	91

**d.) Remove** – Remove marks 67 from List

Index	Marks
0	79
1	64
2	77
3	89
4	46
5	90
6	91

# List

**Append** – Append marks 98 to end of list

Index	Marks
0	79
1	64
2	77
3	89
4	46
5	90
6	91
7	<b>98</b>

The operation depicted in **(a) retrieves** elements of a list by index value. Thus, the marks 89 is retrieved at index 2 (the third item in the list). **The replace operation in (b)** updates the current marks 78 at index 0, with 79. **The insert operation in (c)** inserts the new marks 77 at index 2, thus shifting down all elements below that point and lengthening the list by one. **In (d), the remove operation** deletes the element at index 6, thus shifting up all elements below that point and shortening the list by one. Finally, **the append operation in (e)** adds a new mark, 98, to the end of the list.

# List in Python

- **List**

A list is a mutable, linear data structure of Python that can store a sequence of values belonging to any type.

```
L = [1, 2, 2.5, 'hello']
```

L is a list of mixed values types i.e. two integers (1, 2), one float (2.5) and one string ('hello').

Mutable means that the contents of the list may be altered.

Now, `L = [2.8, 2.5, 'welcome']` contains two float values and one string ('welcome').

A list is a sequence defined by the list class. It contains the methods for creating, manipulating, and processing lists. Elements in a list can be accessed through an index.

# Creating Lists

The list class defines lists. To create a list, you can use list's constructor, as follows:

```
list1 = list ()           # Create an empty list
list2 = list ([2, 3, 4])  # Create a list with elements 2, 3, 4
list3 = list (["red", "green", "blue"]) # Create a list with strings
list4 = list (range (3, 6)) # Create a list with elements 3, 4, 5
list5 = list ("abcd")     # Create a list with characters a, b, c, d
```

You can also create a list by using the following syntax, which is a little simpler:

```
list1 = []                # same as list ()
list2 = [2, 3, 4]         # same as list ([2, 3, 4])
list3 = ["red", "green"]  # same as list (["red", "green"])
```

The elements in a list are separated by commas and are enclosed by a pair of brackets ([]).

# Access values in Lists

- To access values in lists, square brackets are used to slice along with the index or indices to get values stored at that index.
- `seq = Name of the List[ start : end : step]`
- `seq = List[ : : 2]`
- `Seq = List [ 1: :2]`

# Accessing the elements of the list using slice operation



\*lis.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/lis.py (3.7.4)\*

File Edit Format Run Options Window Help

```
l= [1,2,3,4,5,6,7,8,9,10]
print("elements of the list:",l)
print()
print("first element in the list:",l[0])
print()
print(l[2:5])
print()
print(l[ : :2])
print()
print(l[1 : :3])
```



# Some List modification operations

Suppose we have a List l (small L) = ['virat', 'rohit', 'dhoni', 'yuvi']

- a) **Replace** – It is used to replace any element present in the list with the new element with the help of index value. In the example given below, value at index 1 is 'rohit', it is replaced with new value 'rahul'.

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> l = ['virat','rohit','dhoni','yuvi']
>>> l[1]='rahul' # replace rohit with rahul i.e rohit at index 1 is placed with new value rahul
>>> l
['virat', 'rahul', 'dhoni', 'yuvi']
```

- b) **Delete** – It is used to delete any element present in the list with the help of index value. In the example given below, value at index 1 is 'rohit', it is deleted using operation `del l[1]`.

```
>>> del l[1] # delete the value at index 1 i.e. rahul
>>> l # new list
['virat', 'dhoni', 'yuvi']
```

# Some List modification operations

- c) **Insert**– It is used to insert any new element in the list. In the example given below, we inserted 'rahul' at index value 2, now the list l = ['virat', 'dhoni', 'rahul', 'yuvi']

```
>>> l.insert(2, 'rahul')    # insert value rahul at index 2
>>> l                      # new list l
['virat', 'dhoni', 'rahul', 'yuvi']
>>> l[2]
'rahul'
```

- d) **Append** - It is used to add the new element in the list (at the last). In the example given below, we append 'raina' in the list ,now the list l = ['virat', 'dhoni', 'rahul', 'yuvi', 'raina']

```
>>> l.append('raina')      # add new value raina at the last
>>> l                     # new list
['virat', 'dhoni', 'rahul', 'yuvi', 'raina']
```

# Some List modification operations

e) **Sort** - It is used sort the list in ascending order.

```
>>> l.sort()
>>> l                                     # list after sorting
['dhoni', 'rahul', 'raina', 'virat', 'yuvi']
```

f) **Reverse** – It is used to reverse the order of all the elements present in the list.

```
>>> l.reverse()
>>> l
['yuvi', 'virat', 'raina', 'rahul', 'dhoni']
>>> |
```

# Basic List operations

Operation	Description	Example	Output
len	Returns length of list	<code>len([1,2,3,4,5,6,7,8,9,10])</code>	10
concatenation	Joins two lists	<code>[1,2,3,4,5] + [6,7,8,9,10]</code>	<code>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</code>
repetition	Repeats elements in the list	<code>"Hello", "World"*2</code>	<code>['Hello', 'World', 'Hello', 'World']</code>
in	Checks if the value is present In the list	<code>'a' in ['a', 'e', 'i', 'o', 'u']</code>	True
not in	Checks if the value is not present In the list	<code>3 not in [0,2,4,6,8]</code>	True
max	Returns maximum value in the list	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; print(max(num_list))</pre>	9
min	Returns minimum value in the list	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; print(min(num_list))</pre>	0
sum	Adds the values in the list that has numbers	<pre>num_list = [1,2,3,4,5,6,7,8,9,10] print("SUM = ", sum(num_list))</pre>	SUM = 55



# Basic List operations

all	Returns True if all elements of the list are true (or if the list is empty)	<pre>&gt;&gt;&gt; num_list = [0,1,2,3] &gt;&gt;&gt; print(all(num_list))</pre>	False
any	Returns True if any element of the list is true. If the list is empty, returns False	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; print(any(num_list))</pre>	True
list	Converts an iterable (tuple, string, set, dictionary) to a list	<pre>&gt;&gt;&gt; list1 = list("HELLO") &gt;&gt;&gt; print(list1)</pre>	['H', 'E', 'L', 'L', 'O']
sorted	Returns a new sorted list. The original list is not sorted.	<pre>&gt;&gt;&gt; list1 = [3,4,1,2,7,8] &gt;&gt;&gt; list2 = sorted(list1) &gt;&gt;&gt; print(list2)</pre>	[1, 2, 3, 4, 7, 8]

# List Methods

Method	Description	Syntax	Example	Output
<code>append()</code>	Appends an element to the list. In <code>insert()</code> , if the index is 0, then element is inserted as the first element and if we write, <code>list.insert(len(list), obj)</code> , then it inserts <code>obj</code> as the last element in the list. That is, if <code>index= len(list)</code> then <code>insert()</code> method behaves exactly same as <code>append()</code> method.	<code>list. append(obj)</code>	<pre>num_list = [6,3,7,0,1,2,4,9] num_list. append(10) print (num_list)</pre>	<pre>[6,3,7, 0,1,2, 4,9,10]</pre>
<code>count()</code>	Counts the number of times an element appears in the list.	<code>list.count (obj)</code>	<pre>print(num_list. count(4))</pre>	<pre>1</pre>
<code>index()</code>	Returns the lowest index of <code>obj</code> in the list. Gives a <code>ValueError</code> if <code>obj</code> is not present in the list.	<code>list. index(obj)</code>	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,3,7,6,0] &gt;&gt;&gt; print(num_ list.index(7))</pre>	<pre>2</pre>



# List methods

Method	Description	Syntax	Example	Output
<code>insert()</code>	Inserts obj at the specified index in the list.	<code>list. insert(index, obj)</code>	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,3,7,6,0] &gt;&gt;&gt; num_list. insert(3, 100) &gt;&gt;&gt; print(num_list)</pre>	<pre>[6,3,7, 100,0, 3,7,6, 0]</pre>
<code>pop()</code>	Removes the element at the specified index from the list. Index is an optional parameter. If no index is specified, then removes the last object (or element) from the list.	<code>list. pop([index])</code>	<pre>num_list = [6,3,7,0,1,2,4,9] print(num_list. pop()) print(num_list)</pre>	<pre>9 [6, 3, 7, 0, 1, 2, 4]</pre>
<code>remove()</code>	Removes or deletes obj from the list. ValueError is generated if obj is not present in the list. If multiple copies of obj exists in the list, then the first value is deleted.	<code>list. remove(obj)</code>	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; num_list. remove(0) &gt;&gt;&gt; print(num_list)</pre>	<pre>[6,3,7, 1,2,4, 9]</pre>
<code>reverse()</code>	Reverse the elements in the list.	<code>list. reverse()</code>	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; num_list. reverse() &gt;&gt;&gt; print(num_list)</pre>	<pre>[9, 4, 2, 1, 7, 3, 6]</pre>
<code>sort()</code>	Sorts the elements in the list.	<code>list.sort()</code>	<pre>&gt;&gt;&gt; num_list = [6,3,7,0,1,2,4,9] &gt;&gt;&gt; num_list. sort() &gt;&gt;&gt; print(num_list)</pre>	<pre>[9, 4, 2, 1, 0, 7, 3, 6]</pre>
<code>extend()</code>	Adds the elements in a list to the end of another list. Using + or += on a list is similar to using extend().	<code>list1. extend(list2)</code>	<pre>&gt;&gt;&gt; num_list1 = [1,2,3,4,5] &gt;&gt;&gt; num_list2 = [6,7,8,9,10] &gt;&gt;&gt; num_list1. extend(num_list2) &gt;&gt;&gt; print(num_ list1)</pre>	<pre>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</pre>

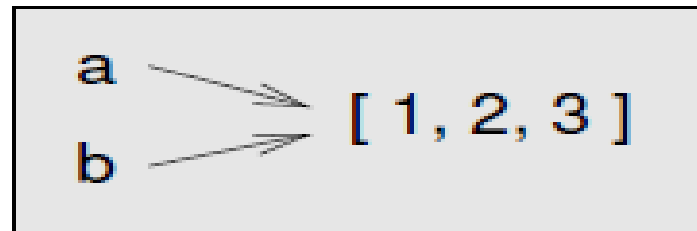
# Aliasing

Since variables refer to objects, if we assign one variable to another, both variables refer to the same object. **For example:**

```
>>> a = [1, 2, 3]
```

```
>>> b=a
```

In this case, the state diagram looks like this:



Because the same list has two different names, a and b, we say that it is aliased. Changes made with one alias affect the other. **For example:**

```
>>> b[0] = 5
```

```
>>> print a
```

```
[5, 2, 3]
```



# Cloning lists

If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference. This process is sometimes called cloning, to avoid the ambiguity of the word “copy”.

The easiest way to clone a list is to use the slice operator. For example:

```
>>> a = [1, 2, 3]
```

```
>>> b = a[:]
```


```
>>> print b
```

```
[1, 2, 3]
```

Taking any slice of **a** creates a new list. In this case the slice happens to consist of the whole list.

# Nested List

- Nested list means a list within another list
- `L = [1,"a","abc",[2,3,4,5],8,9]`

 \*lis.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/lis.py (3.7.4)\*

File Edit Format Run Options Window Help

```
L = [1, 'a', "abc", [2,3,4,5], 8, 9]
```

```
i = 0
```

```
while i < (len(L)) :
```

```
    print("L", "[" , i, "]" , L[i])
```

```
    i+=1
```

# Two Dimensional List

- A **two-dimensional list** is a list that contains other lists as its elements.
- A two-dimensional list consists of a list of one-dimensional lists, you can use a list to store two-dimensional data, such as a matrix or a table, as well.
- For example, the following table, which provides the distances between cities, can be stored in a list named distances.



	Amritsar	Beas	Jalandhar	Phagwara	Chandigarh	Ludhiana	Ambala	New Delhi
Amritsar	0	44	82	103	229	142	251	466
Beas	44	0	38	61	181	100	209	424
Jalandhar	82	38	0	23	143	61	170	386
Phagwara	103	61	23	0	127	40	148	364
Chandigarh	229	181	143	127	0	107	46	261
Ludhiana	142	100	61	40	107	0	109	325
Ambala	251	209	170	148	46	109	0	218
New Delhi	466	424	386	364	261	325	218	0

distances = [

[0, 44, 82, 103, 229, 142, 251, 466],

[44, 0, 38, 61, 181, 100, 209, 424],

[82, 38, 0, 23, 143, 61, 170, 386],

[103, 61, 23, 0, 127, 40, 148, 364],

[229, 181, 143, 127, 0, 107, 46, 261],

[142, 100, 61, 40, 107, 0, 109, 325],

[251, 209, 170, 148, 46, 109, 0, 218]

[466, 424, 386, 364, 261, 325, 218, 0]

]

distances [0] = [0, 44, 82, 103, 229, 142, 251, 466]

distances [1] = [44, 0, 38, 61, 181, 100, 209, 424]

distances [0] [0] = 0

# Three Dimensional List

A **three-dimensional list** consists of a list of two-dimensional lists. In the preceding sections, you used two-dimensional lists to represent a matrix or a table. Occasionally, you need to represent n-dimensional data. You can create n-dimensional lists for any integer n. For example, you can use a three-dimensional list to store exam scores for a class of six students with five exams, and each exam has two parts (multiple-choice and essay). The following syntax creates a three-dimensional list named scores.

```
scores = [  
    [[11.5, 20.5], [11.0, 22.5], [15, 33.5], [13, 21.5], [15, 2.5]],  
    [[4.5, 21.5], [11.0, 22.5], [15, 34.5], [12, 20.5], [14, 11.5]],  
    [[6.5, 30.5], [11.4, 11.5], [11, 33.5], [11, 23.5], [10, 2.5]],  
    [[6.5, 23.5], [11.4, 32.5], [13, 34.5], [11, 20.5], [16, 11.5]],  
    [[8.5, 26.5], [11.4, 52.5], [13, 36.5], [13, 24.5], [16, 2.5]],  
    [[11.5, 20.5], [11.4, 42.5], [13, 31.5], [12, 20.5], [16, 6.5]]]
```

scores [0][1][0] refers to the multiple-choice score for the first student's second exam, which is 11.0. scores [0][1][1] refers to the essay score for the first student's second exam, which is 22.5. The following figure depicts the meaning of the values in the list.



# Searching an element in List

loop.py - C:/Users/dipen/AppData/Local/Programs/Python/Py

File Edit Format Run Options Window Help

```
nums = [10,20,30]
k = 0
item_to_find = 40
found_item = False

while k < len (nums) and not found_item:
    if nums[k] == item_to_find:
        found_item = True
    else:
        k = k+1
if found_item:
    print('item found')
else:
    print('item not found')
```

Python 3.6.1 Shell

File Edit Shell Debug Options Window Help

Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.

```
>>>
= RESTART: C:/Users/dipen/AppData/Local/Programs/Python/Python36-32/loop.py =
item not found
>>> |
```