Python Lists with Examples

Lists in Python are ordered. It is modifiable and changeable, unlike Tuples. Tuples in Python are immutable, whereas Lists are mutable. In other words, the Lists can be modified even after it's created.

How to create a Python List

To create a List, you need to set all the items in square brackets []. Each element is separated by commas i.e. comma separated values. Lists offset begins from index 0. Following is an example with 10 integer elements in a List:

```
mylist = [25, 50, 75, 100, 125, 150, 175, 200 ];
```

Another example showing 5 string elements in a List:

```
mylist = ["Nathan", "Darren", "Blair", "Bryce", "Stacey" ];
```

Let us see another list with type float:

```
mylist = [5.7, 8.2, 9. 3.8, 2.9]
```

List can also have mixed datatype:

```
mylist = ["Blair", 10, 5.8, 8.9, "Nathan"]
```

Following is an example with output displaying 3 Lists with different datatypes;

```
#Creating List1
mylist1 = ["Blair", 10, 5.8, 8.9, "Nathan"]
print(mylist1)

#Creating List2
```

```
mylist2 = [5.7, 8.2, 3.8, 2.9]
print(mylist2)

#Creating List3
mylist3 = [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
print(mylist3)
```

The output is as follows:

```
['Blair', 10, 5.8, 8.9, 'Nathan']
[5.7, 8.2, 3.8, 2.9]
[3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
```

Access values from a List

Refer the index number of the specific value you want to access from a List, for example, to access the element at 3rd position, set index 2,

```
print(mylist[2])
```

Let us now see an example to access values from a Python List:

```
#Creating List1
mylist1 = ["Blair", 10, 5.8, 8.9, "Nathan"]
print("List1 = ",mylist1)
print("List1 element at position 3rd = ",mylist1[2])

#Creating List2
mylist2 = [5.7, 8.2, 3.8, 2.9]
print("List2 = ",mylist2)
print("List2 element at position 4th = ",mylist2[3])

#Creating List3
mylist3 = [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
print("List3 = ",mylist3)
print("List3 element at position 7th = ",mylist3[6])
```

The output is as follows:

```
List1 =  ['Blair', 10, 5.8, 8.9, 'Nathan']
List1 element at position 3rd =  5.8
List2 =  [5.7, 8.2, 3.8, 2.9]
List2 element at position 4th =  2.9
List3 =  [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
List3 element at position 7th =  12
```

How to update list values

Lists are mutable and can be easily updated. Use the slice & assignment operator to set the element's index on its left and the new value on the right, for example:

```
mylist[2] = 50;
```

Above, we updated the 3rd position (index x2) with the value 50 using slice and assignment operator. Next, let us now see an example wherein we will be updating three different lists in a single Python program:

```
#Updating element from 3 Lists

#Creating List1
mylist1 = ["Blair", 10, 5.8, 8.9, "Nathan"]
print("List1 = ",mylist1)
print("List1 element at position 3rd = ",mylist1[2])

#updating mylist1
mylist1[2] = 7.2;

print("Updated List1 = ",mylist1)
print("Updated List1 element at position 3rd = ",mylist1[2])
print("--------------------------")

#Creating List2
mylist2 = [5.7, 8.2, 3.8, 2.9]
print("List2 = ",mylist2)
print("List2 element at position 4th = ",mylist2[3])

#updating mylist2
mylist2[3] = 8.6;

print("Updated List2 = ",mylist2)
print("Updated List2 element at position 4th = ",mylist2[3])
print("--------------------------")

#Creating List3
mylist3 = [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
print("List3 = ",mylist3)
print("List3 element at position 7th = ",mylist3[6])

#updating List3
mylist3[6] = 25;

print("Updated List3 = ",mylist3)
print("Updated List3 element at position 7th = ",mylist3[6])
```

The output is as follows:

```
List1 =  ['Blair', 10, 5.8, 8.9, 'Nathan']
List1 element at position 3rd =  5.8
Updated List1 =  ['Blair', 10, 7.2, 8.9, 'Nathan']
Updated List1 element at position 3rd =  7.2
```

```
-------------------------
List2 =  [5.7, 8.2, 3.8, 2.9]
List2 element at position 4th =  2.9
Updated List2 =  [5.7, 8.2, 3.8, 8.6]
Updated List2 element at position 4th =  8.6
-------------------------
List3 =  [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
List3 element at position 7th =  12
Updated List3 =  [3, 1, 6, 7, 9, 15, 25, 18, 17, 16, 20]
Updated List3 element at position 7th =  25
```

Remove elements from the List

To remove elements from List, use the **del** keyword or the **remove()** method, for example:

```
del mylist(3)
```

Above, deletes the element at index 4.

Let us see an example and remove elements from the List:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#deleting an element
del mylist[4]

#updated list
print("Updated List after deleting an element = ",mylist)

#deleting another element
del mylist[1]

#final list
print("Updated List after deleting another element = ",mylist)
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
Updated List after deleting an element =  ['Squash', 'Football'
Updated List after deleting another element =  ['Squash', 'Hock
```

Remove element from a specific position (index) in the List

To remove an element from a specific position in List, the **del** keyword is used. Set the index in the square brackets as in the

below example:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#deleting an element
del mylist[2]

#updated list
print("Updated List after deleting an element = ",mylist)
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
Updated List after deleting an element =  ['Squash', 'Football'
```

Remove a specific element from a Python List

To remove a specific element from a list, set the element to be deleted in the remove() method:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#deleting an element
mylist.remove("Hockey")

#updated list
print("Updated List after deleting an element = ",mylist)

#deleting another element
mylist.remove("Archery")

#updated final list
print("Updated List after deleting another element = ",mylist)
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
Updated List after deleting an element =  ['Squash', 'Football'
Updated List after deleting another element =  ['Squash', 'Foot
```

Remove all elements from the List (Delete the entire List)

To remove all elements from List i.e. deleting the entire List is possible with the **del** keyword. Let us see an example wherein we will delete all the elements from the List:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#delete entire List
del mylist

#error (mylist not defined) since the List is now deleted
print(mylist)
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
Traceback (most recent call last):
  File "./prog.py", line 9, in <module>
NameError: name 'mylist' is not defined
```

Above, we are getting an error since we are trying to display the List which we deleted using the **del** keyword.

Remove multiple elements from a List in Python

To remove/delete multiple elements from a Python List, use the **del** keyword as in the below example. Set the range of elements to be deleted:

```
del mylist[2:4]
```

Above, we deleted elements from 2 (including) to 4 (excluding) indexes. Therefore, we deleted two elements i.e. index 2 and index 3.

Let us now see an example to delete multiple elements:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#delete multiple elements
```

```
del mylist[2:4]

#Updated List
print("Update List after deleting multiple items = ",mylist)
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
Update List after deleting multiple items =  ['Squash', 'Footba
```

Indexing in Lists

For indexing in Lists, use the index operator []. The indexing in Lists begins from 0. The last item is tuple length – 1. Set the index in square brackets and **fetch a specific element** at the same index. With that, if you want to count from the right, use **negative indexing**. The **slice operator** can also be used to fetch a specific set of sub-elements. This is achieved with the colon operator and allows to fetch multiple items. Let us see them one by one.

Fetch a specific element with indexing in Lists

To fetch a specific element, just set the index in a square bracket i.e. refer to the index number:

```
Fetch a specific element at 1st position: mylist[0]
Fetch a specific element at 2nd position: mylist[1]
Fetch a specific element at 3rd position: mylist[2]
Fetch a specific element at 4th position: mylist[3]
```

Above, we fetched elements using the index numbers. Let us see an example and fetch a specific element with indexing in Lists:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#fetch 1st element
print("1st element = ",mylist[0])

#fetch 2nd element
print("2nd element = ",mylist[1])
```

```
#fetch 3rd element
print("3rd element = ",mylist[2])
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
1st element =  Squash
2nd element =  Football
3rd element =  Hockey
```

Negative Indexing in Lists

Left to right traversal of List elements are possible with negative indexing. For example, index -1 is the last item. Let us see an example of negative indexing in List:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#fetch 1st element
print("1st element = ",mylist[0])
#fetch 2nd element
print("2nd element = ",mylist[1])
#fetch 3rd element
print("3rd element = ",mylist[2])

#negative indexing
print("------------------------")

#fetch last element
print("Last element = ",mylist[-1])
#fetch 2nd last element
print("2nd last element = ",mylist[-2])
#fetch 3rd last element
print("3rd last element = ",mylist[-3])
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
1st element =  Squash
2nd element =  Football
3rd element =  Hockey
------------------------
Last element =  Archery
2nd last element =  Cricket
3rd last element =  Hockey
```

Slice Lists

Slice Lists in Python using the slice operator (:). The **slice operator** is used to fetch a specific set of sub-elements i.e.

slicing. Let us see an example:

```
#Creating List
mylist = ["Squash", "Football", "Hockey", "Cricket", "Archery"]
print("List = ",mylist)

#fetch 1st element
print("1st element = ",mylist[0])
#fetch 2nd element
print("2nd element = ",mylist[1])
#fetch 3rd element
print("3rd element = ",mylist[2])

#slice
print(mylist[2:4])

#elements beginning from 2nd to end
print(mylist[2:])
```

The output is as follows:

```
List =  ['Squash', 'Football', 'Hockey', 'Cricket', 'Archery']
1st element =  Squash
2nd element =  Football
3rd element =  Hockey
['Hockey', 'Cricket']
['Hockey', 'Cricket', 'Archery']
```

In the above example, [2:4] means the search starts from index 2 (including) and ends at index 4 (excluding). In the same way, [2:] means the search starts from 2nd to end.

Range of indexes

By specifying the start and end range, we can specify the range of indexes in Python. Let us see an example:

```
print(mylist[1:2])
print(mylist[1:4])
print(mylist[1:5])
print(mylist[3:5])
```

Let us now see an example:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

print(mylist[1:2])
print(mylist[1:4])
print(mylist[1:5])
```

```
print(mylist[3:5])
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
['Billy']
['Billy', 'Lucas', 'Albert']
['Billy', 'Lucas', 'Albert', 'Eva']
['Albert', 'Eva']
```

In the above example, [1:2] means the search starts from index 1 (including) and ends at index 2 (excluding). In the same way, [3:5] means the search starts from 2 (including) and ends at index 5 (excluding):

Iterate through the List

The for loop is used to iterate through a List in Python as in the below example:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#Printing the List
print("Iterating though the List...")

#Iterating through the List
for v in mylist:
  print(v)
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Iterating though the List...
Teun
Billy
Lucas
Albert
Eva
```

Add elements to the List

To add elements to the List, use the append() method as in the below example:

```
#Creating List
```

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#Printing the List
print("Iterating though the List...")

#Iterating through the List
for v in mylist:
  print(v)

mylist.append("Sam")

#Printing the updated List
print("Iterating though the updated List...")

#Iterating through the List
for v in mylist:
  print(v)
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Iterating though the List...
Teun
Billy
Lucas
Albert
Eva
Iterating though the updated List...
Teun
Billy
Lucas
Albert
Eva
Sam
```

Add element at a specific index in List

The insert() method is used to add element at a specific index in List as in the below example:

```
mylist.insert(3, "Brandon")
```

Above, the first parameter in the insert() method is the index and the second parameter is the element to be inserted.

Finally, let us see a complete example:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#Printing the List
print("Iterating though the List...")
```

```
print("Iterating though the List...")

#Iterating through the List
for v in mylist:
  print(v)

mylist.insert(4,"Brandon")

#Printing the updated List
print("Iterating though the updated List...")

#Iterating through the List
for v in mylist:
  print(v)
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Iterating though the List...
Teun
Billy
Lucas
Albert
Eva
Iterating though the updated List...
Teun
Billy
Lucas
Albert
Brandon
Eva
```

Check if a specific item exists in a List

To check if a specific element exists in a List, the **in** keyword is used:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#Iterating through the List
for v in mylist:
  print(v)

#checking for element Lucas
if "Lucas" in mylist:
  print("Found!")

#checking for element Gary
if "Gary" in mylist:
  print("Found!")
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Teun
Billy
```

```
Billy
Lucas
Albert
Eva
Found!
```

Extend the List by adding another List to the end

The **extend()** method is used to extend the list by adding another List to the end. Let us see an example wherein we are extending a List:

```
#Creating List
mylist = [10, 20, 30, 40]
print("List = ",mylist)

#Iterating through the List
for v in mylist:
  print(v)

#Creating another List
mylist2 = [50, 60, 70]

mylist.extend(mylist2)

#Display the updated List
print("List = ",mylist)

#Iterating through the updated List
for v in mylist:
  print(v)
```

The output is as follows:

```
List =  [10, 20, 30, 40]
10
20
30
40
List =  [10, 20, 30, 40, 50, 60, 70]
10
20
30
40
50
60
70
```

Get the length of the List

To get the length of the List, the **len()** method is used in Python. Let us see an example to get the total length of the List:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#List length
print("Length List = ",len(mylist));

print("Iterating List elements......");

#Iterating through the List
for v in mylist:
  print(v)

#checking for element Lucas
if "Lucas" in mylist:
  print("Found!")
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Length List =  5
Iterating List elements......
Teun
Billy
Lucas
Albert
Eva
Found!
```

Get maximum value from the List

To get the maximum value from the List, the max() method is to be used. Let us see an example:

```
#Creating List1
mylist1 = [5.7, 8.2, 3.8, 2.9]
print("List1 = ",mylist1)
print("Maximum value in List1 = ",max(mylist1))

#Creating List2
mylist2 = [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
print("List2 = ",mylist2)
print("Maximum value in List2 = ",max(mylist2))
```

The output is as follows:

```
List1 =  [5.7, 8.2, 3.8, 2.9]
Maximum value in List1 =  8.2
List2 =  [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
Maximum value in List2 =  20
```

Get minimum value from the List

To get the minimum value from the List, the min() method is to be used. Let us see an example:

```
#Creating List1
mylist1 = [5.7, 8.2, 3.8, 2.9]
print("List1 = ",mylist1)
print("Maximum value in List1 = ",max(mylist1))
print("Minimum value in List1 = ",min(mylist1))


#Creating List2
mylist2 = [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
print("List2 = ",mylist2)
print("Maximum value in List2 = ",max(mylist2))
print("Minimum value in List2 = ",min(mylist2))
```

The output is as follows:

```
List1 =  [5.7, 8.2, 3.8, 2.9]
Maximum value in List1 =  8.2
Minimum value in List1 =  2.9
List2 =  [3, 1, 6, 7, 9, 15, 12, 18, 17, 16, 20]
Maximum value in List2 =  20
Minimum value in List2 =  1
```

Make a copy of the List

To make a copy of an already created list, the copy() method is used. Let us see an example:

```
#Creating List1
mylist1 = [5.7, 8.2, 3.8, 2.9]
print("List1 = ",mylist1)
print("Maximum value in List1 = ",max(mylist1))
print("Minimum value in List1 = ",min(mylist1))

#Making a copy
print("Copying List1 to List2......")

mylist2 = mylist1.copy()
print("List2 = ",mylist2)
```

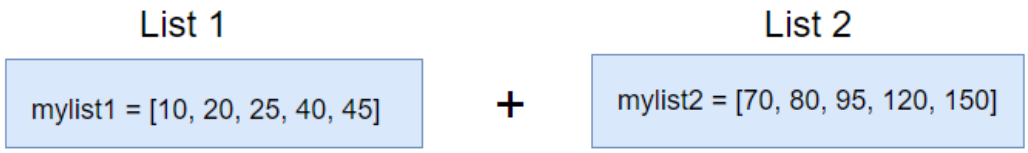
The output is as follows:

```
List1 =  [5.7, 8.2, 3.8, 2.9]
Maximum value in List1 =  8.2
Minimum value in List1 =  2.9
Copying List1 to List2......
List2 =  [5.7, 8.2, 3.8, 2.9]
```

Join Two Lists in Python

Join Two Lists in Python

The + operator is used to join two lists in Python as in the below illustration:



Let us now see an example wherein we will join two lists:

```
#Creating List1
mylist1 = [10, 20, 25, 40, 45]
print("List1 = ",mylist1)
print("Maximum value in List1 = ",max(mylist1))
print("Minimum value in List1 = ",min(mylist1))

print("-----------------------------------");

#Creating List1
mylist2 = [70, 80, 95, 120, 150]
print("List2 = ",mylist2)
print("Maximum value in List2 = ",max(mylist2))
print("Minimum value in List2 = ",min(mylist2))

print("-----------------------------------");

#Joining two Lists
mylist3 = mylist1 + mylist2
print("Joining List1 and List2 =",mylist3)
```

The output is as follows:

```
List1 =  [10, 20, 25, 40, 45]
Maximum value in List1 =  45
Minimum value in List1 =  10
-----------------------------------
List2 =  [70, 80, 95, 120, 150]
Maximum value in List2 =  150
Minimum value in List2 =  70
-----------------------------------
Joining List1 and List2 = [10, 20, 25, 40, 45, 70, 80, 95, 120,
```

Concatenate Two Lists in Python

The + operator is used to concatenate two lists in Python as in the below example:

```
#Creating List1
mylist1 = [20, 40, 60, 80, 100]
print("List1 = ",mylist1)
print("Maximum value in List1 = ",max(mylist1))
print("Minimum value in List1 = ",min(mylist1))

print("-----------------------------------");

#Creating List1
mylist2 = [120, 140, 160, 180, 200]
print("List2 = ",mylist2)
print("Maximum value in List2 = ",max(mylist2))
print("Minimum value in List2 = ",min(mylist2))

print("-----------------------------------");

#Concatenating two Lists
mylist3 = mylist1 + mylist2
print("Concatenation Result of List1 and List2 =",mylist3)
```

The output is as follows:

```
List1 =  [20, 40, 60, 80, 100]
Maximum value in List1 =  100
Minimum value in List1 =  20
-----------------------------------
List2 =  [120, 140, 160, 180, 200]
Maximum value in List2 =  200
Minimum value in List2 =  120
-----------------------------------
Concatenation Result of List1 and List2 = [20, 40, 60, 80, 100,
```

Sort List in Python

The **sort()** method is used in Python to sort List as in the below example:

```
#Creating List
mylist = ["Teun", "Billy", "Lucas", "Albert", "Eva"]
print("List = ",mylist)

#Printing the List
print("Iterating though the List...")

#Iterating through the List
for v in mylist:
  print(v)

#sorting the List
mylist.sort()

print("Sorted List...");
for v in mylist:
  print(v)
```

The output is as follows:

```
List =  ['Teun', 'Billy', 'Lucas', 'Albert', 'Eva']
Iterating though the List...
Teun
Billy
Lucas
Albert
Eva
Sorted List...
Albert
Billy
Eva
Lucas
Teun
```

Reverse the order of the List

To reverse the order of elements in a List, the **reverse()** method is to be used. Let us now see an example:

```
#Creating List
mylist = [10, 20, 25, 40, 45, 60, 80, 90, 100,200]
print("List = ",mylist)

#Reverse List
mylist.reverse()

print("Reverse List = ",mylist);
```

The output is as follows:

```
List =  [10, 20, 25, 40, 45, 60, 80, 90, 100, 200]
Reverse List =  [200, 100, 90, 80, 60, 45, 40, 25, 20, 10]
```

Multi-Dimensional Python List

Lists within Lists are known as Multi-Dimensional Lists in Python. Let us first see how to create a Multi-Dimensional Python List:

```
mylist = [[10, 20, 35, 40,], [50, 60], [80], [90, 100, 200]]
```

Above, we created 4 integer lists in a List i.e. Multi-dimensional List. To access a list in a multi-dimensional list, use square brackets, like Multi-Dimensional Arrays with:

```
list[row_size][column_size]
```

We are accessing the first and second elements in the 1st list of our Multi-Dimensional List:

```
print(mylist[0][0])
print(mylist[0][1])
```

To explain the concept, we have represented Multi-dimensional List graphically below. Similarly, we represent the left index as row number and right index as column number:

Finally, let us see an example wherein we will be creating a Multi-Dimensional List and learn to display all the elements or any specific element:

```
#Creating List mylist1
mylist = [10, 20, 25, 40, 45, 60, 80, 90, 100,200]
print("Printing the entire List1")

#Creating multi-dimensional List mylist2
mylist2 = [[10, 20, 35, 40], [50, 60], [80], [90, 100, 200]]
print("List2 = ",mylist2)

print("Printing elements (row1) from multi-dimensional List2");
print(mylist2[0][0])
print(mylist2[0][1])
print(mylist2[0][2])
print(mylist2[0][3])

print("Printing elements (row2) from multi-dimensional List2");
print(mylist2[1][0])
print(mylist2[1][1])

print("Printing element (row3) from multi-dimensional List2");
print(mylist2[2][0])
```

```
print( "Printing elements (row4) from multi-dimensional List2 ");
print(mylist2[3][0])
print(mylist2[3][1])
print(mylist2[3][2])

print("Printing the entire multi-dimensional List2");

for res in mylist2:
  print(res)
```

The output is as follows:

```
Printing the entire List1
List2 =  [[10, 20, 35, 40], [50, 60], [80], [90, 100, 200]]
Printing elements (row1) from multi-dimensional List2
10
20
35
40
Printing elements (row2) from multi-dimensional List2
50
60
Printing element (row3) from multi-dimensional List2
80
Printing elements (row4) from multi-dimensional List2
90
100
200
Printing the entire multi-dimensional List2
[10, 20, 35, 40]
[50, 60]
[80]
[90, 100, 200]
```

Access elements in a Multi-Dimensional Python List

To access elements in a Multi-Dimensional List, use the number of rows and number of columns. For example, to access the 1st element of the first row set the index of rows and columns i.e. for the 1st element:

```
list[0][0]
```

For the 2nd element in the first row:

```
list[0][1]
```

In the same way, for the 1st element in the second row:

```
list[1][0]
```

Finally, let us see an example to access elements in a Multi-Dimensional List with 4 lists:

```
#Creating multi-dimensional List
mylist = [[10], [50, 60], [80], [90, 100, 200]]
print("List = ",mylist)

print("Printing elements (row1) from multi-dimensional List");
print(mylist[0][0])

print("Printing elements (row2) from multi-dimensional List");
print(mylist[1][0])
print(mylist[1][1])

print("Printing element (row3) from multi-dimensional List2");
print(mylist[2][0])

print("Printing elements (row4) from multi-dimensional List2");
print(mylist[3][0])
print(mylist[3][1])
print(mylist[3][2])

print("Printing the entire multi-dimensional List");

for res in mylist:
  print(res)
```

The output is as follows:

```
List =  [[10], [50, 60], [80], [90, 100, 200]]
Printing elements (row1) from multi-dimensional List
10
Printing elements (row2) from multi-dimensional List
50
60
Printing element (row3) from multi-dimensional List2
80
Printing elements (row4) from multi-dimensional List2
90
100
200
Printing the entire multi-dimensional List
[10]
[50, 60]
[80]
[90, 100, 200]
```

Add elements to the end of the Multi-Dimensional List

Adding elements to the end of Multi-Dimensional List is possible with the **append()** method as in the below example:

```
#Creating multi-dimensional List
mylist = [[10], [50, 60], [80], [90, 100, 200]]
print("List = ",mylist)

print("Printing the multi-dimensional List");

for res in mylist:
  print(res)

#Adding elements to the end
mylist.append([250, 275, 350])

print("Printing the updated multi-dimensional List");

for res in mylist:
  print(res)
```

The output is as follows:

```
List =  [[10], [50, 60], [80], [90, 100, 200]]
Printing the multi-dimensional List
[10]
[50, 60]
[80]
[90, 100, 200]
Printing the updated multi-dimensional List
[10]
[50, 60]
[80]
[90, 100, 200]
[250, 275, 350]
```

In this tutorial, we learned what are Lists in Python. Moreover, we worked around creating Python Lists and their operations like adding, deleting, extending, appending, etc. Clearly, it's quite easy to work on Python Lists. Apart from this, if you are aware of a topic we missed in the List, then kindly mention it in the comment section.

Recommend Posts

- Type Conversion in Python
- Strings in Python
- Functions in Python
- Python Tuples Tutorial
- Python Dictionary
- DateTime Tutorial in Python
- Python Multiple Choice Questions (MCQs)