

Python - Tuples

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);  
tup2 = (1, 2, 3, 4, 5 );  
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing –

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value –

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
#!/usr/bin/python  
  
tup1 = ('physics', 'chemistry', 1997, 2000);  
tup2 = (1, 2, 3, 4, 5, 6, 7 );  
print "tup1[0]: ", tup1[0];  
print "tup2[1:5]: ", tup2[1:5];
```

[Live Demo](#)

When the above code is executed, it produces the following result –

```
tup1[0]: physics  
tup2[1:5]: [2, 3, 4, 5]
```

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

[Live Demo](#)

```
#!/usr/bin/python  
  
tup1 = (12, 34.56);  
tup2 = ('abc', 'xyz');  
  
# Following action is not valid for tuples  
# tup1[0] = 100;  
  
# So let's create a new tuple as follows  
tup3 = tup1 + tup2;  
print tup3;
```

When the above code is executed, it produces the following result –

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example –

[Live Demo](#)

```
#!/usr/bin/python  
  
tup = ('physics', 'chemistry', 1997, 2000);  
print tup;  
del tup;  
print "After deleting tup : ";  
print tup;
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more –

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter –

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!') * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input –

```
L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

No Enclosing Delimiters

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short examples –

[Live Demo](#)

```
#!/usr/bin/python

print 'abc', -4.24e93, 18+6.6j, 'xyz';
x, y = 1, 2;
print "Value of x , y : ", x,y;
```

When the above code is executed, it produces the following result –

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y : 1 2
```

Built-in Tuple Functions

Python includes the following tuple functions –

Sr.No.	Function with Description
1	<code>cmp(tuple1, tuple2)</code> Compares elements of both tuples.
2	<code>len(tuple)</code> Gives the total length of the tuple.
3	<code>max(tuple)</code> Returns item from the tuple with max value.
4	<code>min(tuple)</code> Returns item from the tuple with min value.
5	<code>tuple(seq)</code> Converts a list into tuple.

Useful Video Courses



Python Online Training

187 Lectures 17.5 hours

Malhar Lathkar

[More Detail](#)



Python - Lists

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences, but the most common ones are lists and tuples, which we would see in this tutorial.

There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000];  
list2 = [1, 2, 3, 4, 5 ];  
list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
#!/usr/bin/python  
  
list1 = ['physics', 'chemistry', 1997, 2000];  
list2 = [1, 2, 3, 4, 5, 6, 7 ];  
print "list1[0]: ", list1[0]  
print "list2[1:5]: ", list2[1:5]
```

[Live Demo](#)

When the above code is executed, it produces the following result –

```
list1[0]: physics  
list2[1:5]: [2, 3, 4, 5]
```

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the `append()` method. For example –

```
#!/usr/bin/python  
  
list = ['physics', 'chemistry', 1997, 2000];  
print "Value available at index 2 : "  
print list[2]  
list[2] = 2001;  
print "New value available at index 2 : "  
print list[2]
```

[Live Demo](#)

Note – `append()` method is discussed in subsequent section.

When the above code is executed, it produces the following result –

```
Value available at index 2 :  
1997  
New value available at index 2 :  
2001
```

Delete List Elements

To remove a list element, you can use either the `del` statement if you know exactly which element(s) you are deleting or the `remove()` method if you do not know. For example –

```
#!/usr/bin/python  
  
list1 = ['physics', 'chemistry', 1997, 2000];  
print list1  
del list1[2];  
print "After deleting value at index 2 : "  
print list1
```

[Live Demo](#)

When the above code is executed, it produces following result –

```
['physics', 'chemistry', 1997, 2000]  
After deleting value at index 2 :  
['physics', 'chemistry', 2000]
```

Note – `remove()` method is discussed in subsequent section.

Basic List Operations

Lists respond to the `+` and `*` operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

Python Expression	Results	Description
<code>len([1, 2, 3])</code>	3	Length
<code>[1, 2, 3] + [4, 5, 6]</code>	<code>[1, 2, 3, 4, 5, 6]</code>	Concatenation
<code>['Hi!'] * 4</code>	<code>['Hi!', 'Hi!', 'Hi!', 'Hi!']</code>	Repetition
<code>3 in [1, 2, 3]</code>	True	Membership
<code>for x in [1, 2, 3]: print x,</code>	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.

Assuming following input –

```
L = ['spam', 'Spam', 'SPAM!']
```

Python Expression	Results	Description
<code>L[2]</code>	SPAM!	Offsets start at zero
<code>L[-2]</code>	Spam	Negative: count from the right
<code>L[1:]</code>	<code>['Spam', 'SPAM!']</code>	Slicing fetches sections

Built-in List Functions & Methods

Python includes the following list functions –

Sr.No.	Function with Description
1	<code>cmp(list1, list2)</code> Compares elements of both lists.
2	<code>len(list)</code> Gives the total length of the list.
3	<code>max(list)</code> Returns item from the list with max value.
4	<code>min(list)</code> Returns item from the list with min value.
5	<code>list(seq)</code> Converts a tuple into list.

Python includes following list methods

Sr.No.	Methods with Description
1	<code>list.append(obj)</code> Appends object <code>obj</code> to list
2	<code>list.count(obj)</code> Returns count of how many times <code>obj</code> occurs in list
3	<code>list.extend(seq)</code> Appends the contents of <code>seq</code> to list
4	<code>list.index(obj)</code> Returns the lowest index in list that <code>obj</code> appears
5	<code>list.insert(index, obj)</code> Inserts object <code>obj</code> into list at offset <code>index</code>
6	<code>list.pop(obj=list[-1])</code> Removes and returns last object or <code>obj</code> from list
7	<code>list.remove(obj)</code> Removes object <code>obj</code> from list
8	<code>list.reverse()</code> Reverses objects of list in place
9	<code>list.sort([func])</code> Sorts objects of list, use compare <code>func</code> if given

Useful Video Courses

