GIZMOFACTS

PYTHON LIST AND TUPLES

Kaustav Ghosh Dastidar

www.gizmofacts.com

Python
Lists and Tuples

Preface

Always wanted to have a go at programming? No more excuses, Python is the ideal way to get started!

Python is an excellent programming language for both novices and experts. It is designed with code readability in mind, making it an excellent choice for beginners learning various programming concepts.

The language is widely used and has many libraries, allowing programmers to accomplish a lot with little code.

I hope you enjoyed my previous three modules, Python Fundamentals, Python Decision Making and Controls & Python Numbers and Strings. If not, please proceed on rush for a quick revision. Here is the fourth module and the most important topics I will cover here i.e. Python Lists and Tuples.

List is an ordered collection of elements. By ordered we mean that the elements are stored one after the other. Lists are mutable this means that the content of the list can be modified by adding, removing or changing objects during the execution of the program.

Just like list tuple is also an ordered collection of elements. However, unlike lists tuples are immutable. Which means that once a tuple is created, we cannot add, delete, modify the elements of the tuple.

This book is written to help you to learn Python programming quickly and effectively.

With Warm Regards,

Kaustav Ghosh Dastidar

Author & Founder of Gizmofacts

About The Author

The man behind **Gizmofacts, Kaustav Ghosh Dastidar** holds a Master Degree in Computer Science. He is usually turned on (metaphorically) by technology and gadgets. Born and raised in a quintessential middle class family he has been well aware of the ignorance the mass harbours about technology. Through Gizmofacts he wants to reach out to all those people, who he believes just need a little push to get into this unique and amazing world of science and software.

Moreover, Kaustav is well aware that nurturing an interest in gadgets doesn't come cheap. Hence he wants to also be an enabler who would provide all the 'need to know' financial details of different gadgets so that people can live their dreams remaining in their budget.

To know more about tips and tricks of softwares, gizmos and mobile apps, follow him in ***Twitter***, ***Facebook*** and Google+.

You may also subscribe to Gizmofacts in ***Youtube*** for getting more information about software tips & tricks.

Disclaimer

This eBook may not be copied or distributed without permission in any way. This publication's content is offered solely for informational reasons. The usage or misuse of this eBook, as well as any financial damage incurred by individuals or property as a direct or indirect result of using this eBook, are not the author's responsibility.

We are unable to guarantee your success or outcomes in the future due to some unforeseeable risks associated with doing business online. You accept that the author is not responsible for any success or failure of your business that is related in any way to the download and use of our information and that the use of our information should be based on your due diligence.

Without the author's prior written consent, no portion of this eBook may be copied or otherwise distributed in any way, including electronically, mechanically, by photocopy, recording, or any other method

Table of Content

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]
```

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]
```

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
```

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
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 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
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	[‘Spam’, ‘SPAM!’]	Slicing fetches sections

Built-in List Functions & Methods

Python includes the following list functions −

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

Python List cmp() Method

Description

Python list method **cmp()** compares elements of two lists.

Syntax

Following is the syntax for **cmp()** method −

```
cmp(list1, list2)
```

Parameters

- **list1** − This is the first list to be compared.
- **list2** − This is the second list to be compared.

Return Value

If elements are of the same type, perform the compare and return the result. If elements are different types, check to see if they are numbers.

- If numbers, perform numeric coercion if necessary and compare.
- If either element is a number, then the other element is "larger" (numbers are "smallest").
- Otherwise, types are sorted alphabetically by name.

If we reached the end of one of the lists, the longer list is "larger." If we exhaust both lists and share the same data, the result is a tie, meaning that 0 is returned.

Example

The following example shows the usage of cmp() method.

```
#!/usr/bin/python

list1, list2 = [123, 'xyz'], [456, 'abc']
print cmp(list1, list2)
print cmp(list2, list1)
list3 = list2 + [786];
print cmp(list2, list3)
```

When we run above program, it produces following result −

```
-1
1
-1
```

Python List len() Method

Description

Python list method **len()** returns the number of elements in the *list.*

Syntax

Following is the syntax for **len()** method −

```
len(list)
```

Parameters

- **list1** − This is a list for which number of elements to be counted.

Return Value

This method returns the number of elements in the list.

Example

The following example shows the usage of len() method.

```
#!/usr/bin/python

list1, list2 = [123, 'xyz', 'zara'], [456, 'abc']
print "First list length : ", len(list1)
print "Second list length : ", len(list2)
```

When we run above program, it produces following result −

```
First list length :  3
Second list length :  2
```

Python List max() Method

Description

Python list method max returns the elements from the list with maximum value.

Syntax

Following is the syntax for **max()** method −

```
max(list)
```

Parameters

- **list1** − This is a list from which max valued element to be returned.

Return Value

This method returns the elements from the list with maximum value.

Example

The following example shows the usage of cmp() method.

```
#!/usr/bin/python

list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 700, 200]
print "Max value element : ", max(list1)
print "Max value element : ", max(list2)
```

When we run above program, it produces following result −

```
Max value element :  zara
Max value element :  700
```

Python List min() Method

Description

Python list method **min()** returns the elements from the *list* with minimum value.

Syntax

Following is the syntax for **min()** method −

```
min(list)
```

Parameters

- **list1** − This is a list from which min valued element to be returned.

Return Value

This method returns the elements from the list with minimum value.

Example

The following example shows the usage of min() method.

```
#!/usr/bin/python

list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 700, 200]
print "min value element : ", min(list1)
print "min value element : ", min(list2)
```

When we run above program, it produces following result −

```
min value element :  123
min value element :  200
```

Python List list () Method

Description

Python list method **list()** takes sequence types and converts them to lists. This is used to convert a given tuple into list.

Note − Tuple are very similar to lists with only difference that element values of a tuple can not be changed and tuple elements are put between parentheses instead of square bracket.

Syntax

Following is the syntax for **list()** method −

```
list( seq )
```

- **seq** − This is a tuple to be converted into list.

Return Value

This method returns the list.

Example

The following example shows the usage of list() method.

```
#!/usr/bin/python

aTuple = (123, 'xyz', 'zara', 'abc');
aList = list(aTuple)
print "List elements : ", aList
```

When we run above program, it produces following result −

```
List elements :  [123, 'xyz', 'zara', 'abc']
```

Python includes following list methods

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

Python List append() Method

Description

Python list method **append()** appends a passed obj into the existing list.

Syntax

Following is the syntax for **append()** method −

```
list.append(obj)
```

Parameters

- **obj** − This is the object to be appended in the list.

Return Value

This method does not return any value but updates existing list.

Example

The following example shows the usage of append() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc'];
aList.append( 2009 );
print "Updated List : ", aList
```

When we run above program, it produces following result −

```
Updated List :  [123, 'xyz', 'zara', 'abc', 2009]
```

Python List count() Method

Description

Python list method **count()** returns count of how many times *obj* occurs in list.

Syntax

Following is the syntax for **count()** method −

```
list.count(obj)
```

Parameters

- **obj** − This is the object to be counted in the list.

Return Value

This method returns count of how many times obj occurs in list.

Example

The following example shows the usage of count() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc', 123];
print "Count for 123 : ", aList.count(123)
print "Count for zara : ", aList.count('zara')
```

When we run above program, it produces following result −

```
Count for 123 :  2
Count for zara :  1
```

Python List extend() Method

Description

Python list method **extend()** appends the contents of *seq* to list.

Syntax

Following is the syntax for **extend()** method −

```
list.extend(seq)
```

Parameters

- **seq** − This is the list of elements

Return Value

This method does not return any value but add the content to existing list.

Example

The following example shows the usage of extend() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc', 123];
bList = [2009, 'manni'];
aList.extend(bList)
print "Extended List : ", aList
```

When we run above program, it produces following result −

```
Extended List :  [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

Python List index() Method

Description

Python list method **index()** returns the lowest index in list that *obj* appears.

Syntax

Following is the syntax for **index()** method −

```
list.index(obj)
```

Parameters

- **obj** − This is the object to be find out.

Return Value

This method returns index of the found object otherwise raise an exception indicating that value does not find.

Example

The following example shows the usage of index() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc'];
print "Index for xyz : ", aList.index( 'xyz' )
print "Index for zara : ", aList.index( 'zara' )
```

When we run above program, it produces following result −

```
Index for xyz :  1
Index for zara :  2
```

Python List insert() Method

Description

Python list method **insert()** inserts object *obj* into list at offset *index*.

Syntax

Following is the syntax for **insert()** method −

```
list.insert(index, obj)
```

Parameters

- **index** − This is the Index where the object obj need to be inserted.
- **obj** – This is the Object to be inserted into the given list.

Return Value

This method does not return any value but it inserts the given element at the given index.

Example

The following example shows the usage of insert() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc']
aList.insert( 3, 2009)
print "Final List : ", aList
```

When we run above program, it produces following result −

```
Final List : [123, 'xyz', 'zara', 2009, 'abc']
```

Python List pop() Method

Description

Python list method **pop()** removes and returns last object or *obj* from the list.

Syntax

Following is the syntax for **pop()** method −

```
list.pop(obj = list[-1])
```

Parameters

- **obj** − This is an optional parameter, index of the object to be removed from the list.

Return Value

This method returns the removed object from the list.

Example

The following example shows the usage of pop() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc'];
print "A List : ", aList.pop()
print "B List : ", aList.pop(2)
```

When we run above program, it produces following result −

```
A List :  abc
B List :  zara
```

Python List remove() Method

Description

Python list method remove() searches for the given element in the list and removes the first matching element.

Syntax

Following is the syntax for **remove()** method −

```
list.remove(obj)
```

Parameters

- **obj** – This is the object to be removed from the list.

Return Value

This Python list method does not return any value but removes the given object from the list.

Example

The following example shows the usage of remove() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.remove('xyz');
print "List : ", aList
aList.remove('abc');
print "List : ", aList
```

When we run above program, it produces following result −

```
List :  [123, 'zara', 'abc', 'xyz']
List :  [123, 'zara', 'xyz']
```

Python List reverse() Method

Description

Python list method reverse() reverses objects of list in place.

Syntax

Following is the syntax for **reverse()** method −

```
list.reverse()
```

Parameters

- **NA**

Return Value

This method does not return any value but reverse the given object from the list.

Example

The following example shows the usage of reverse() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.reverse();
print "List : ", aList
```

When we run above program, it produces following result −

```
List :  ['xyz', 'abc', 'zara', 'xyz', 123]
```

Python List sort() Method

Description

Python list method **sort()** sorts objects of list, use compare *func* if given.

Syntax

Following is the syntax for **sort()** method −

```
list.sort([func])
```

Parameters

- **NA**

Return Value

This method does not return any value but it changes from the original list.

Example

The following example shows the usage of sort() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.sort();
print "List : ", aList
```

When we run above program, it produces following result −

```
List :  [123, 'abc', 'xyz', 'xyz', 'zara']
```

Python - Lists

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];
```

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 −

```
#!/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 −

```
#!/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 −

```
#!/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	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

Python Tuple cmp() Method

Description

Python tuple method cmp() compares elements of two tuples.

Syntax

Following is the syntax for **cmp()** method −

```
cmp(tuple1, tuple2)
```

Parameters

- **tuple1** −This is the first tuple to be compared
- **tuple2** − This is the second tuple to be compared

Return Value

If elements are of the same type, perform the compare and return the result. If elements are different types, check to see if they are numbers.

- If numbers, perform numeric coercion if necessary and compare.
- If either element is a number, then the other element is "larger" (numbers are "smallest").
- Otherwise, types are sorted alphabetically by name.

If we reached the end of one of the tuples, the longer tuple is "larger." If we exhaust both tuples and share the same data, the result is a tie, meaning that 0 is returned.

Example

The following example shows the usage of cmp() method.

```
#!/usr/bin/python

tuple1, tuple2 = (123, 'xyz'), (456, 'abc')
print cmp(tuple1, tuple2)
print cmp(tuple2, tuple1)
tuple3 = tuple2 + (786,);
print cmp(tuple2, tuple3)
```

When we run above program, it produces following result −

```
-1
1
-1
```

Python Tuple len() Method

Description

Python tuple method **len()** returns the number of elements in the tuple.

Syntax

Following is the syntax for **len()** method −

```
len(tuple)
```

Parameters

- **tuple** − This is a tuple for which number of elements to be counted.

Return Value

This method returns the number of elements in the tuple.

Example

The following example shows the usage of len() method.

```
#!/usr/bin/python

tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')
print "First tuple length : ", len(tuple1)
print "Second tuple length : ", len(tuple2)
```

When we run above program, it produces following result −

```
First tuple length :  3
Second tuple length :  2
```

Python Tuple max() Method

Description

Python tuple method **max()** returns the elements from the tuple with maximum value.

Syntax

Following is the syntax for **max()** method −

```
max(tuple)
```

Parameters

- **tuple** – This is a tuple from which max valued element to be returned.

Return Value

This method returns the elements from the tuple with maximum value.

Example

The following example shows the usage of max() method.

```
#!/usr/bin/python

tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)
print "Max value element : ", max(tuple1)
print "Max value element : ", max(tuple2)
```

When we run above program, it produces following result −

```
Max value element :  zara
Max value element :  700
```

Python Tuple min() Method

Description

Python tuple method **min()** returns the elements from the tuple with minimum value.

Syntax

Following is the syntax for **min()** method −

```
min(tuple)
```

Parameters

- **tuple** – This is a tuple from which min valued element to be returned.

Return Value

This method returns the elements from the tuple with minimum value.

Example

The following example shows the usage of min() method.

```
#!/usr/bin/python

tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)
print "min value element : ", min(tuple1)
print "min value element : ", min(tuple2)
```

When we run above program, it produces following result −

```
min value element :  123
min value element :  200
```

Python Tuple tuple() Method

Description

Python tuple method **tuple()** converts a list of items into tuples

Syntax

Following is the syntax for **tuple()** method −

```
tuple( seq )
```

Parameters

- **seq** − This is a sequence to be converted into tuple.

Return Value

This method returns the tuple.

Example

The following example shows the usage of tuple() method.

```
#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc']
aTuple = tuple(aList)
print "Tuple elements : ", aTuple
```

When we run above program, it produces following result −

```
Tuple elements :  (123, ‘xyz’, ‘zara’, ‘abc’)
```

Python - Dictionary

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

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

```
dict['Name']:  Zara
dict['Age']:  7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Alice']: ", dict['Alice']
```

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

```
dict['Alice']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

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

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −

```
dict['Age']:
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

Note − del() method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

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

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

```
#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

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

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

Built-in Dictionary Functions & Methods

Python includes the following dictionary functions −

Sr.No.	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python dictionary cmp() Method

Description

Python dictionary method **cmp()** compares two dictionaries based on key and values.

Syntax

Following is the syntax for **cmp()** method −

```
cmp(dict1, dict2)
```

Parameters

- **dict1** − This is the first dictionary to be compared with dict2.
- **dict2** − This is the second dictionary to be compared with dict1.

Return Value

This method returns 0 if both dictionaries are equal, -1 if dict1 < dict2 and 1 if dict1 > dic2.

Example

The following example shows the usage of cmp() method.

```
#!/usr/bin/python

dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Abid', 'Age': 27};
dict4 = {'Name': 'Zara', 'Age': 7};
print "Return Value : %d" %  cmp (dict1, dict2)
print "Return Value : %d" %  cmp (dict2, dict3)
print "Return Value : %d" %  cmp (dict1, dict4)
```

When we run above program, it produces following result −

```
Return Value : -1
Return Value : 1
Return Value : 0
```

Python dictionary len() Method

Description

Python dictionary method len() gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

Syntax

Following is the syntax for **len()** method −

```
len(dict)
```

Parameters

- **dict1** − This is the dictionary, whose length needs to be calculated.

Return Value

This method returns the length.

Example

The following example shows the usage of len() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};
print "Length : %d" % len (dict)
```

When we run above program, it produces following result −

```
Length : 2
```

Python dictionary str() Method

Description

Python dictionary method **str()** produces a printable string representation of a dictionary.

Syntax

Following is the syntax for **str()** method −

```
str(dict)
```

Parameters

- **dict** − This is the dictionary.

Return Value

This method returns string representation.

Example

The following example shows the usage of str() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};
print "Equivalent String : %s" % str (dict)
```

When we run above program, it produces following result −

```
Equivalent String : {'Age': 7, 'Name': 'Zara'}
```

Python dictionary type() Method

Description

Python dictionary method **type()** returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type.

Syntax

Following is the syntax for **type()** method −

```
type(dict)
```

Parameters

- **dict** − This is the dictionary.

Return Value

This method returns the type of the passed variable.

Example

The following example shows the usage of type() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};
print "Variable Type : %s" %  type (dict)
```

When we run above program, it produces following result −

```
Variable Type : <type 'dict'>
```

Python includes following dictionary methods −

Sr.No.	Function with Description
1	dict.clear() Removes all elements of dictionary dict
2	dict.copy() Returns a shallow copy of dictionary dict
3	dict.fromkeys() Create a new dictionary with keys from seq and values set to value.
4	dict.get(key, default=None) For key key, returns value or default if key not in dictionary
5	dict.has_key(key) Returns true if key in dictionary dict, false otherwise
6	dict.items() Returns a list of dict's (key, value) tuple pairs
7	dict.keys() Returns list of dictionary dict's keys
8	dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if key is not already in dict
8	dict.update(dict2) Adds dictionary dict2's key-values pairs to dict
10	dict.values() Returns list of dictionary dict's values

Python dictionary clear() Method

Description

Python dictionary method **clear()** removes all items from the dictionary.

Syntax

Following is the syntax for **clear()** method −

```
dict.clear()
```

Parameters

- **NA**

Return Value

This method does not return any value.

Example

The following example shows the usage of clear() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};
print "Start Len : %d" %  len(dict)
dict.clear()
print "End Len : %d" %  len(dict)
```

When we run above program, it produces following result −

```
Start Len : 2
End Len : 0
```

Python dictionary copy() Method

Description

Python dictionary method **copy()** returns a shallow copy of the dictionary.

Syntax

Following is the syntax for **copy()** method −

```
dict.copy()
```

Parameters

- **NA**

Return Value

This method returns a shallow copy of the dictionary.

Example

The following example shows the usage of copy() method.

```
#!/usr/bin/python

dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1.copy()
print "New Dictionary : %s" %  str(dict2)
```

When we run above program, it produces following result −

```
New Dictionary : {'Age': 7, 'Name': 'Zara'}
```

Python dictionary fromkeys() Method

Description

Python dictionary method **fromkeys()** creates a new dictionary with keys from seq and values set to value.

Syntax

Following is the syntax for **fromkeys()** method −

```
dict.fromkeys(seq[, value])
```

Parameters

- **seq** − This is the list of values which would be used for dictionary keys preparation.
- **value** − This is optional, if provided then value would be set to this value

Return Value

This method returns the list.

Example

The following example shows the usage of fromkeys() method.

```
#!/usr/bin/python

seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print "New Dictionary : %s" %  str(dict)

dict = dict.fromkeys(seq, 10)
print "New Dictionary : %s" %  str(dict)
```

When we run above program, it produces following result −

```
New Dictionary : {'age': None, 'name': None, 'sex': None}
New Dictionary : {'age': 10, 'name': 10, 'sex': 10}
```

Python dictionary get() Method

Description

Python dictionary method get() returns a value for the given key. If key is not available then returns default value None.

Syntax

Following is the syntax for **get()** method −

```
dict.get(key, default = None)
```

Parameters

- **key** − This is the Key to be searched in the dictionary.
- **default** − This is the Value to be returned in case key does not exist.

Return Value

This method return a value for the given key. If key is not available, then returns default value None.

Example

The following example shows the usage of get() method.

```
#!/usr/bin/python

dict = {'Name': 'Zabra', 'Age': 7}
print "Value : %s" %  dict.get('Age')
print "Value : %s" %  dict.get('Education', "Never")
```

When we run above program, it produces following result −

```
Value : 7
Value : Never
```

Python dictionary has_key() Method

Description

Python dictionary method has_key() returns true if a given key is available in the dictionary, otherwise it returns a false.

Syntax

Following is the syntax for **has_key()** method −

```
dict.has_key(key)
```

Parameters

- **key** − This is the Key to be searched in the dictionary.

Return Value

This method return true if a given key is available in the dictionary, otherwise it returns a false.

Example

The following example shows the usage of has_key() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
print "Value : %s" %  dict.has_key('Age')
print "Value : %s" %  dict.has_key('Sex')
```

When we run above program, it produces following result −

```
Value : True
Value : False
```

Python dictionary items() Method

Description

Python dictionary method **items()** returns a list of dict's (key, value) tuple pairs

Syntax

Following is the syntax for **items()** method −

```
dict.items()
```

Parameters

- **NA**

Return Value

This method returns a list of tuple pairs.

Example

The following example shows the usage of items() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
print "Value : %s" %  dict.items()
```

When we run above program, it produces following result −

```
Value : [('Age', 7), ('Name', 'Zara')]
```

Python dictionary keys() Method

Description

Python dictionary method **keys()** returns a list of all the available keys in the dictionary.

Syntax

Following is the syntax for **keys()** method −

```
dict.keys()
```

Parameters

- **NA**

Return Value

This method returns a list of all the available keys in the dictionary.

Example

The following example shows the usage of keys() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
print "Value : %s" %  dict.keys()
```

When we run above program, it produces following result −

```
Value : ['Age', 'Name']
```

Python dictionary setdefault() Method

Description

Python dictionary method **setdefault()** is similar to get(), but will set dict[key]=default if key is not already in dict.

Syntax

Following is the syntax for **setdefault()** method −

```
dict.setdefault(key, default=None)
```

Parameters

- **key** − This is the key to be searched.
- **default** − This is the Value to be returned in case key is not found.

Return Value

This method returns the key value available in the dictionary and if given key is not available then it will return provided default value.

Example

The following example shows the usage of setdefault() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
print "Value : %s" %  dict.setdefault('Age', None)
print "Value : %s" %  dict.setdefault('Sex', None)
```

When we run above program, it produces following result −

```
Value : 7
Value : None
```

Python dictionary update() Method

Description

Python dictionary method update() adds dictionary dict2's key-values pairs in to dict. This function does not return anything.

Syntax

Following is the syntax for **update()** method −

```
dict.update(dict2)
```

Parameters

- **dict2** − This is the dictionary to be added into dict.

Return Value

This method does not return any value.

Example

The following example shows the usage of update() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
dict2 = {'Sex': 'female' }

dict.update(dict2)
print "Value : %s" %  dict
```

When we run above program, it produces following result −

```
Value : {'Age': 7, 'Name': 'Zara', 'Sex': 'female'}
```

Python dictionary values() Method

Description

Python dictionary method **values()** returns a list of all the values available in a given dictionary.

Syntax

Following is the syntax for **values()** method −

```
dict.values()
```

Parameters

- **NA**

Return Value

This method returns a list of all the values available in a given dictionary.

Example

The following example shows the usage of values() method.

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}
print "Value : %s" %  dict.values()
```

When we run above program, it produces following result −

```
Value : [7, 'Zara']
```

Next Steps?

Have any questions while reading the eBook? Want to discuss programming, gadgets and gaming? You can email us (kaustav@gizmofacts.com) where you can ask any questions you have in min2d and subscribe to our Newsletter and visit our Facebook Page.

You can also check our blog "Gizmofacts" where you can find a ton of useful "how-to-guides", tutorials around blogging, programming, gadget reviews and gaming.