

Python Dictionary

Advertisements

⏪ Previous Page

Next Page ⏩

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: list objects are unhashable
```

Built-in Dictionary Functions & Methods –

Python includes the following dictionary functions –

SN	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 includes following dictionary methods –

SN	Methods with Description
1	dict.clear() Removes all elements of dictionary <i>dict</i>
2	dict.copy() Returns a shallow copy of dictionary <i>dict</i>
3	dict.fromkeys() Create a new dictionary with keys from seq and values set to <i>value</i> .
4	dict.get(key, default=None) For key <i>key</i> , returns value or default if key not in dictionary
5	dict.has_key(key) Returns <i>true</i> if key in dictionary <i>dict</i> , <i>false</i> otherwise
6	dict.items() Returns a list of <i>dict</i> 's (key, value) tuple pairs
7	dict.keys() Returns list of dictionary <i>dict</i> 's keys
8	dict.setdefault(key, default=None) Similar to <i>get()</i> , but will set <i>dict[key]=default</i> if <i>key</i> is not already in <i>dict</i>
9	dict.update(dict2) Adds dictionary <i>dict2</i> 's key-values pairs to <i>dict</i>
10	dict.values() Returns list of dictionary <i>dict</i> 's values



[Write for us](#) [FAQ's](#) [Helping](#) [Contact](#)

© Copyright 2017. All Rights Reserved.