Dictionary

So far you have learned about string, tuple and list. In this chapter we have another useful built-in data type dictionary. In list, values are indexed by range of numbers, but in dictionary values are indexed by keys. In Python, the dictionary is a sequence of key/value or item pairs separated by commas.

Example

port = {22: "SSH", 23: "Telnet" , 53: "DNS", 80: "HTTP" }

Syntax of dictionary is given as below:

Dictionary\_name = {key: value}

The key, value pair is called item. The key and value are separated by colon (: ) and each item is separated by comma (,). The items are enclosed by curly braces ({ }). An empty dictionary can be created just by using curly braces ({ }).

Key features of the dictionary:

* The key of the Dictionary can be not be changed.
* The string, int or float can be used as key.
* The tuple which does not contain any list can be used as key.
* Keys are unique.
* Values can be anything example list, string, int etc.
* Value can be repeated.
* Values can be changed.
* Dictionary is an unordered collection. The order in which you have entered the items in dictionary may not be retained and you may get the items in different order.

Operations on the dictionary

In this section you will learn different-different python dictionary operation.

Accessing the values of dictionary

In order to access the dictionary’s values you will need the key. Consider a dictionary of networking ports.

Port = {80: “HTTP”, 23 : “Telnet”, 443 : “HTTPS”}

Let us learn by the example.

>>> port = {80: "HTTP", 23 : "Telnet", 443 : "HTTPS"}

>>> port[80]

'HTTP'

>>> port[443]

'HTTPS'

In order to access dictionary’ value, use the square brackets along with the key.

What happen if key is not in the dictionary?

>>> port[21]

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

port[21]

KeyError: 21

>>>

If key is not found then interpreter shows error.

**Delete the item from the dictionary**

By using del keyword you can delete entire dictionary or dictionary’s items.

If you want to delete the dictionary’s item. Use syntax as shown below.

del dict[key]

Example

>>> port = {80: "HTTP", 23 : "Telnet", 443 : "HTTPS"}

>>> del port[23]

>>> port

{80: 'HTTP', 443: 'HTTPS'}

>>>

If you want to delete entire dictionary then use the syntax as shown below.

del dict

Example

>>> port = {80: "HTTP", 23 : "Telnet", 443 : "HTTPS"}

>>> del port

>>> port

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

port

NameError: name 'port' is not defined

>>>

Above error shows that dictionary port has been deleted.

**Updating the values of the dictionary.**

Updating the dictionary is pretty simple, just specify the key in the square bracket along with dictionary name. The syntax is given as

dict[key] = new\_value

Consider the example.

port = {80: "HTTP", 23 : "SMTP”, 443 : "HTTPS"}

In the above dictionary the value of port 23 is “SMTP” but in reality port number 23 is for telnet protocol.

Let us update the above dictionary.

>>> port = {80: "HTTP", 23 : "SMTP", 443 : "HTTPS"}

>>> port

{80: 'HTTP', 443: 'HTTPS', 23: 'SMTP'}

>>> port[23] = "Telnet"

>>> port

{80: 'HTTP', 443: 'HTTPS', 23: 'Telnet'}

>>>

**Adding item to the dictionary**

To add an item to the dictionary is very simple, just specify a new key in square brackets along with the dictionary. The syntax is given as.

dict[new\_key] = value

Example

>>> port = {80: "HTTP", 23 : "Telnet"}

>>> port[110]="POP"

>>> port

{80: 'HTTP', 110: 'POP', 23: 'Telnet'}

>>>

In above example we have added protocol “POP”.

Dictionary functions

In this section we will explore the built-in functions available in python which can be applied on dictionary.

In order to find number of items that are present in the dictionary you can use len() function.

See the example below.

>>> port = {80: "http", 443: "https", 23:"telnet"}

>>> len(port)

3

>>>

Consider situation where you want to convert a dictionary into string there you can use str() function.

Syntax is given as:

str(dict)

Example

>>> port = {80: "http", 443: "https", 23:"telnet"}

>>> port

{80: 'http', 443: 'https', 23: 'telnet'}

>>> str(port)

"{80: 'http', 443: 'https', 23: 'telnet'}"

>>>

You can easily see the double quotes around the dictionary.

Let us get the max key from the dictionary using max() function.

Syntax is given as

max(dict)

>>> dict1 = {1:"abc",5:"hj", 43:"Dhoni", ("a","b"):"game", "hj":56}

>>> max(dict1)

('a', 'b')

So the max function gives the tuple as max value. Similarly to get minimum key you can use min() function.

Syntax is given as

min(dict)

>>> dict1 = {1:"abc",5:"hj", 43:"Dhoni", ("a","b"):"game", "hj":56, (1,3):"kl"}

>>> dict1

{1: 'abc', (1, 3): 'kl', 5: 'hj', 43: 'Dhoni', 'hj': 56, ('a', 'b'): 'game'}

>>> min(dict1)

1

>>>

Let us convert list or tuple into dictionary. In order to convert list or tuple into dictionary the format should be like as shown below

port = [[80,"http"],[20,"ftp"],[23,"telnet"],[443,"https"],[53,"DNS"]]

or

port = [(80,"http"),(20,"ftp"),(23,"telnet"),(443,"https"),(53,"DNS")]

We need pairs of two values.

By using dict function we can convert the above list into dictionary

Syntax is given as

dict(port)

Let us see the conversion by example

>>> port = [[80,"http"],[20,"ftp"],[23,"telnet"],[443,"https"],[53,"DNS"]]

>>> port

[[80, 'http'], [20, 'ftp'], [23, 'telnet'], [443, 'https'], [53, 'DNS']]

>>> dict(port)

{80: 'http', 443: 'https', 20: 'ftp', 53: 'DNS', 23: 'telnet'}

>>>

>>> port = [(80,"http"),(20,"ftp"),(23,"telnet"),(443,"https"),(53,"DNS")]

>>> dict(port)

{80: 'http', 443: 'https', 20: 'ftp', 53: 'DNS', 23: 'telnet'}

>>>

In operator on python dictionary

The in operator can be used to find existence of key in the dictionary consider the example program inkey.py

port1 = {21: "FTP", 22:"SSH", 23: "telnet", 80: "http"}

key = int(raw\_input("Enter the key "))

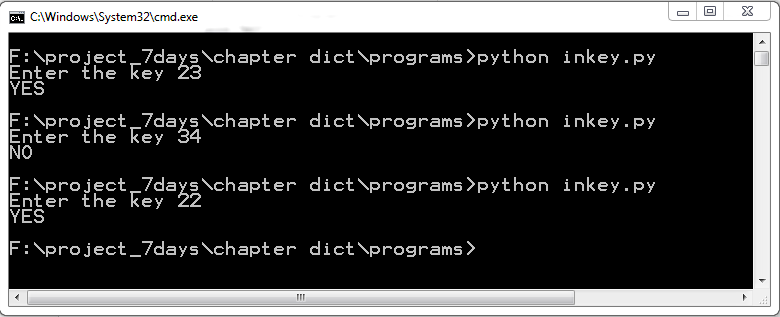
if key in port1:

print "YES"

else :

print "NO"

See the below figure for output.



Output of in program

Similarly you can use not in operator.

Example

>>> port1 = {21: "FTP", 22 :"SSH", 23: "telnet", 80: "http"}

>>> if 21 not in port1:

print "yes"

>>>

Nothing get printed.

Dictionary methods

In this section we will discuss the dictionary methods one by one.

Consider you want to create a copy of existing dictionary, you can use copy() method.

Syntax is given as

dict.copy()

See the example below.

>>> Avengers ={'iron-man':"Tony", "CA":"Steve","BW":"Natasha"}

>>> Avengers

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>> Avengers2 = Avengers.copy()

>>> Avengers2

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>>

You can see that Avengers2 is the exact copy of Avengers. Do not be confused with copy with assignment operator.

Let us see the example below

>>> A1 = {'iron-man':"Tony", "CA":"Steve","BW":"Natasha"}

>>> A2= A1

>>>

>>> A2

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>>

>>> CW= A1.copy()

>>> CW

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>>

Variable A1, A2 holds same dictionary but variable CW hold different dictionary. You can check memory address of A1, A2 and CW.

>>> id(A1)

46395728

>>> id(A2)

46395728

>>> id(CW)

46136896

>>>

We can do one more thing, let us add one more member to A1 dictionary.

>>> A1

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>> A1["hulk"]= "Bruce-Banner"

>>> A1

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> A2

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> CW

{'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha'}

>>>

We have changed the A1 dictionary, and the changes would also be reflected by A2, since both hold same memory address, whereas CW is holding different dictionary.

Consider you have a dictionary and you want to access the key which does not exist in the dictionary. Interpreter shows a KeyError as shown below.

>>> A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> A1["panther"]

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

A1["panther"]

KeyError: 'panther'

>>>

In above code you can clearly see the error, if this happen in running code, your code would not get fully executed. In order to deal with this situation, we will use get() method.

Syntax is given as.

dict.get(key, default=None)

The method get() is used to get the value of given key from the dictionary. If key is not found then default value or message would return.

See the example below

If key is present

>>> A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> A1.get('iron-man',"not found")

'Tony'

In above example, Since the key is found the custom message “not found” does not get printed.

Let see another example.

>>> A1.get('panther',"not found")

'not found'

>>> A1.get("Black")

>>>

If custom message is not set then nothing would be return. There is another method setdefault() very much similar to get() method with little different functionality. Let us discuss the method with examples.

The syntax is given as:

dict.setdefault(key, message)

key -- This is the key to be searched.

Default -- In case key is not found then message would be returned and added to the dictionary.\

Let us see the example.

>>> port1.setdefault(23, "Unknown")

'Telnet'

>>> port1

{80: 'http', 22: 'SSH', 23: 'Telnet'}

>>> port1.setdefault(19, "Unknown")

'Unknown'

>>> port1

{80: 'http', 19: 'Unknown', 22: 'SSH', 23: 'Telnet'}

If message has been not been set then it return and add default value none. See the example below.

>>> port1.setdefault(18)

>>> port1

{80: 'http', 18: None, 19: 'Unknown', 22: 'SSH', 23: 'Telnet'}

>>>

To avoid the keyError we can use get() method, but we can add one more check to avoid the KeyError. The has\_key() method facilitate you to check the given key exist or not.

The syntax is given as:

dict.has\_key(key)

key -- This is the Key to be searched in the dictionary dict. Let us see the example.

The method has\_key() return the True or false.

See the example below.

>>> port1 = {80: 'http', 18: None, 19: 'Unknown', 22: 'SSH', 23: 'Telnet'}

>>> port1.has\_key(80)

True

>>>

>>> port1.has\_key(20)

False

>>>

Consider a situation where you want to do some operation on dictionary’s keys and want to get all keys in the different lists. In this situation you can use the method keys() method.

Syntax is given as:

dict.keys()

Let us consider the example.

A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

In the above dictionary, we want superhero’s characters means all keys.

>>> A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> A1.keys()

['iron-man', 'CA', 'BW', 'hulk']

>>>

The above method return a list of all keys.

Similarly if we want all values in separate list we can use values() methods.

Syntax is given as:

dict.values()

Let us consider the example.

A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

In the above dictionary, we want to get all real name of our heroes.

>>> A1 = {'iron-man': 'Tony', 'CA': 'Steve', 'BW': 'Natasha', 'hulk': 'Bruce-Banner'}

>>> A1.values()

['Tony', 'Steve', 'Natasha', 'Bruce-Banner']

>>>

Sometimes, we need to add one dictionary to another dictionary. Consider an example.

port1 = {22: "SSH", 23: "telnet", 80: "Http”}

We have another dictionary of ports that is port2 as shown below.

port2 = {53 :"DNS", 443 : "https"}

In order to update port1 with port2, we can take the advantage update() method.

Syntax is given as

dict.update(dict2)

dict2-- This is the dictionary to be added.

Example

>>> port1 = {22: "SSH", 23: "telnet", 80: "Http" }

>>>

>>> port2 = {53 :"DNS", 443 : "https"}

>>>

>>> port1.update(port2)

>>> port1

{80: 'Http', 443: 'https', 53: 'DNS', 22: 'SSH', 23: 'telnet'}

>>>

Be careful to use above method because if dict and dict2 dictionaries contain same key then dict dictionary’s keys would be replaced by key of dict1 dictionary.

Our next method is items() the syntax is given below.

dict.items()

The method items() returns the list of dict's (key, value) tuple pairs.

>>> dict1 = d={1:'one',2:'two',3:'three'}

>>> dict1.items()

[(1, 'one'), (2, 'two'), (3, 'three')]

>>>

Sometimes we need to delete all items of dictionary, these can be done by using clear() methods.

Syntax is given as:

dict.clear()

Let us see the example

>>> dict1={1:'one',2:'two',3:'three'}

>>> dict1

{1: 'one', 2: 'two', 3: 'three'}

>>> dict1.clear()

>>> dict1

{}

>>>

**Python dictionary with for loop**

In this section we will learn how to use for loop with dictionary. If you have not done for loop than you can skip the section and get back after learning of for loop.

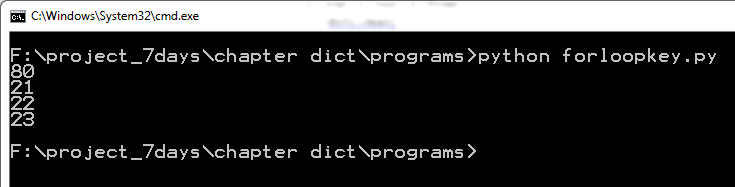
Let us apply for loop on dictionary see the below program name forloopkey.py

See the program below.

port1 = {21: "FTP", 22 :"SSH", 23: "telnet", 80: "http"}

for each in port1:

print each



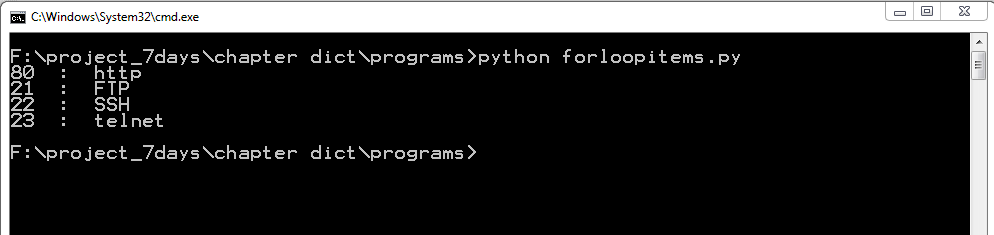
Output of program forloopkey.

The above program prints only the keys of dictionary. If you want to print key as well as value then you can use items () method. See the program forloopitems.py below.

port1 = {21: "FTP", 22 :"SSH", 23: "telnet", 80: "http"}

for k,v in port1.items():

print k," : ", v



Output of program forloopitems.py

The above program seems difficult to understand in first sight. Let us break the program in to parts. The port1.items() returns a list of tuples pair as shown below.

>>> port1 = {21: "FTP", 22:"SSH", 23: "telnet", 80: "http"}

>>> port1.items ()

[(80, 'http'), (21, 'FTP'), (22, 'SSH'), (23, 'telnet')]

>>>

The for loop send each tuple to variable k and v. See the one iteration of for loop below.

>>> k,v = (80, 'http')

>>> k

80

>>> v

'http'

>>>

Now you have learnt how to iterate dictionary. Consider a dictionary of millions of items, and you want to iterate the dictionary, the items () method returns a list of key, value pairs. The newly created list will also take the lot of memory. Python dictionary has very beautiful memory saving method called iteritems()

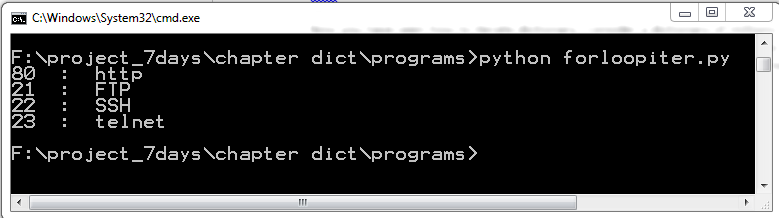
Let us understand by example code forloopiter.py

port1 = {21: "FTP", 22 :"SSH", 23: "telnet", 80: "http"}

for k,v in port1.iteritems():

print k," : ", v

See the out put below



Out put of forloopiter.py program

The above code does not create a list. Let us understand by python shell.

>>> port1 = {21: "FTP", 22 :"SSH", 23: "telnet", 80: "http"}

>>> port1.iteritems()

<dictionary-itemiterator object at 0x02A66210>

>>>

The above code returns a dictionary object not huge list. We will not delve into detail of object.

You can replace key() by iterkeys() and values() by itervalues().

When to use iteritems() and items() ?

If we need the corresponding list and have to do some operations of the list like slicing, indexing then items() method would be the most suitable.

But if our need is just to iterate the dictionary then iteritems() would be the most suitable.

Practical program:

Let us make some program in the dictionary to

1. Make a dictionary from two lists. Both the list are of equal length. Take the list as shown below.

list1 = [1, 2, 3, 4, 5]

list2 = ["a", "b", "c","d", "e"]

The list1 values act as keys of dictionary and list2 values acts as values.

Porgram.

list1 = [1, 2, 3, 4, 5]

list2 = ["a", "b", "c","d", "e"]

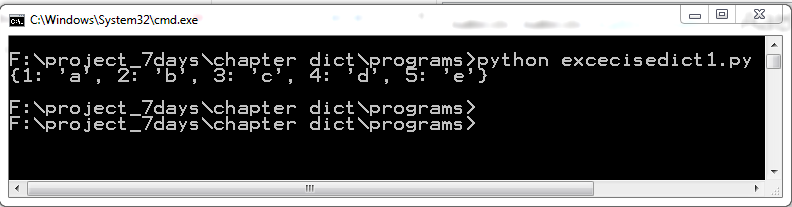
dict1 = {}

for index1 in xrange(len(list1)):

dict1 [list1 [index1]] = list2 [index1]

print dict1

Output.



Output of exercise 1

Let us do the above exercise in one line.

>>> list1 = [1,2,3,4,5]

>>> list2 = ["a", "b", "c","d", "e"]

>>> dict1 = dict([k for k in zip(list1,list2)])

>>> dict1

{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}

>>>

Just one liner code can make a dictionary from 2 lists. So in the above example we have used zip() function , zip() is the built-in function which takes two list and returns a list of 2-tuples like ( [(key, value)].

Let us see one example of zip() function.

>>> list1 = [1,2,3]

>>> list2 = ["a","b","c"]

>>> zip(list1, list2)

[(1, 'a'), (2, 'b'), (3, 'c')]

>>>

Exercise:

1. Find the number of ways to find that whether a key in dictionary exist or not.
2. Use the following dictionary

port1 = {21: "FTP", 22:"SSH", 23: "telnet", 80: "http"}

and make a new dictionary in which key become values and values become key as shown below.

Port2 = {“FTP":21, "SSH":22, “telnet":23, "http": 80}

Summary

In this chapter we learned about dictionary, how to create empty dictionary, adding items to the dictionary, accessing values and deleting values from a dictionary. In order to find number of items in the dictionary we used len() function. There are other useful functions like max(), min() to find max value and min value from dictionary respectively. In the dictionary method we have learned different methods of dictionary such as copy(), keys(), items(). By using items() we can iterate the dictionary. In the end we learned memory saving methods such as iteritems(), iterkeys().