Iterating over dictionaries using 'for' loops

Ask Question

2617

586

I am a bit puzzled by the following code:

```
d = {'x': 1, 'y': 2, 'z': 3}
for key in d:
    print key, 'corresponds to', d[key]
```

What I don't understand is the `key` portion. How does Python recognize that it needs only to read the key from the dictionary? Is `key` a special word in Python? Or is it simply a variable?

python python-2.7 dictionary

edited Nov 20 '17 at 21:39
sberry
92.3k 10 103 140

asked Jul 20 '10 at 22:27
TopChef
13.9k 10 22 26

13 Answers

4648

✓

`key` is just a variable name.

```
for key in d:
```

will simply loop over the keys in the dictionary, rather than the keys and values. To loop over both key and value you can use the following:

For Python 2.x:

```
for key, value in d.iteritems():
```

For Python 3.x:

```
for key, value in d.items():
```

For Python 3.x, `iteritems()` has been replaced with simply `items()`, which returns a set-like view backed by the dict, like `iteritems()` but even better. This is also available in 2.7 as `viewitems()`.

The operation `items()` will work for both 2 and 3, but in 2 it will return a list of the dictionary's `(key, value)` pairs, which will not reflect changes to the dict that happen after the `items()` call. If you want the 2.x behavior in 3.x, you can call `list(d.items())`.

edited May 17 '18 at 6:27

answered Jul 20 '10 at 22:29
sberry
92.3k 10 103 140

98 Adding an overlooked reason not to access value like this: d[key] inside the for loop causes the key to be hashed again (to get the value). When the dictionary is large this extra hash will add to the overall time. This is discussed in Raymond Hettinger's tech talk youtube.com/watch?v=anrOzOapJ2E – HarisankarK Jul 28 '17 at 9:43

13 Might make sense to mention that items will be iterated in unpredictable order and `sorted` is needed to stabilize it. – yugr Aug 25 '18 at 9:06

1 @HarisankarKrishnaSwamy what is the alternative? – JoeyC Nov 8 '18 at 4:45

13 Well done variable naming! – Alan Storm Nov 10 '18 at 20:46

3 *"To test yourself, change the word `key` to `poop`."* You've inspired me to do that in all of my scripts now. – connectyourcharger Mar 23 at 11:57

379 It's not that key is a special word, but that dictionaries implement the iterator protocol. You could do this in your class, e.g. see this question for how to build class iterators.

are available in PEP 234. In particular, the section titled "Dictionary Iterators":

> - Dictionaries implement a tp_iter slot that returns an efficient iterator that iterates over the keys of the dictionary. [...] This means that we can write
>
>   ```
>   for k in dict: ...
>   ```
>
>   which is equivalent to, but much faster than
>
>   ```
>   for k in dict.keys(): ...
>   ```
>
>   as long as the restriction on modifications to the dictionary (either by the loop or by another thread) are not violated.
>
> - Add methods to dictionaries that return different kinds of iterators explicitly:
>
>   ```
>   for key in dict.iterkeys(): .
>
>   for value in dict.itervalues(
>
>   for key, value in dict.iterit
>   ```
>
>   This means that `for x in dict` is shorthand for `for x in dict.iterkeys()`.

In Python 3, `dict.iterkeys()`, `dict.itervalues()` and `dict.iteritems()` are no longer supported. Use `dict.keys()`, `dict.values()` and `dict.items()` instead.

edited Nov 30 '18 at 0:11
jpp
103k 21 67 117

answered Jul 20 '10 at 23:52
ars
85.2k 19 124 127

61 In python3 dict.iterkeys(), dict.itervalues() and dict.iteritems() are no longer supported. Use dict.keys(), dict.values() and dict.items() instead. – Sadik Jun 1 '15 at 8:49

Edit: (This is **no longer the case in Python3.6**, but note that it's **not guaranteed** behaviour yet)

```
>>> d = {'x': 1, 'y': 2, 'z': 3}
>>> list(d)
['y', 'x', 'z']
>>> d.keys()
['y', 'x', 'z']
```

For your example, it is a better idea to use `dict.items()` :

```
>>> d.items()
[('y', 2), ('x', 1), ('z', 3)]
```

This gives you a list of tuples. When you loop over them like this, each tuple is unpacked into `k` and `v` automatically:

```
for k,v in d.items():
    print(k, 'corresponds to', v)
```

Using `k` and `v` as variable names when looping over a `dict` is quite common if the body of the loop is only a few lines. For more complicated loops it may be a good idea to use more descriptive names:

```
for letter, number in d.items():
    print(letter, 'corresponds to',
```

It's a good idea to get into the habit of using format strings:

```
for letter, number in d.items():
    print('{0} corresponds to {1}'.
```

edited Oct 31 '17 at 15:12
nescius
32 5

answered Jul 21 '10 at 1:27
John La Rooy
216k 41 279 432

4 From the Python 3.7 release notes: "The insertion-order preservation nature of dict objects is now an official part of the Python language spec." – Gregory Arenius Jul 18 '18 at 16:30

69

`key` is simply a variable.

For **Python2.X**:

... or better,

```
d = {'x': 1, 'y': 2, 'z': 3}
for the_key, the_value in d.iterite
    print the_key, 'corresponds to'
```

For **Python3.X**:

```
d = {'x': 1, 'y': 2, 'z': 3}
for the_key, the_value in d.items()
    print(the_key, 'corresponds to'
```

edited Jun 15 '18 at 10:51

answered Jul 20 '10 at 23:49
ssoler
2,296 2 22 28

46

When you iterate through dictionaries using the `for .. in ..` -syntax, it always iterates over the keys (the values are accessible using `dictionary[key]`).

To iterate over key-value pairs, use `for k,v in s.iteritems()` .

answered Jul 20 '10 at 22:29
Alexander Gessler
39.2k 5 70 113

32 Note that for Python 3, it is `items()` instead of `iteritems()` – Andreas Fester Mar 26 '15 at 11:38

25

This is a very common looping idiom. `in` is an operator. For when to use `for key in dict` and when it must be `for key in dict.keys()` see David Goodger's Idiomatic Python article.

answered Jul 20 '10 at 22:42
chryss
6,088 32 42

As I read these sections about `in` , the operator part is where you check for existence. Maybe the better delete this *`in is an operator`* information. – Wolf May 19 '16 at 12:17

You can use this:

edited Mar 4 '17 at 21:47
Peter Mortensen
14k 19 87 114

answered Jan 14 '17 at 14:42
A H M Forhadul Islam
898 7 10

6 Its a bit old post – Sadi Jan 14 '17 at 14:44

1 @Sadi Is it no longer true? – Basj May 21 '18 at 8:56

11

I have a use case where I have to iterate through the dict to get the key, value pair, also the index indicating where I am. This is how I do it:

```
d = {'x': 1, 'y': 2, 'z': 3}
for i, (key, value) in enumerate(d.
    print(i, key, value)
```

Note that the parentheses around the key, value is important, without the parentheses, you get an ValueError "not enough values to unpack".

edited Jun 2 '17 at 15:37

answered May 25 '17 at 13:42
jdhao
4,849 2 30 50

10

> ## Iterating over dictionaries using 'for' loops
>
> ```
> d = {'x': 1, 'y': 2, 'z': 3}
> for key in d:
>     ...
> ```
>
> How does Python recognize that it needs only to read the key from the dictionary? Is key a special word in Python? Or is it simply a variable?

It's not just `for` loops. The important word here is "iterating".

A dictionary is a mapping of keys to values:

```
d = {'x': 1, 'y': 2, 'z': 3}
```

is only intended to be descriptive - and it is quite apt for the purpose.

This happens in a list comprehension:

```
>>> [k for k in d]
['x', 'y', 'z']
```

It happens when we pass the dictionary to list (or any other collection type object):

```
>>> list(d)
['x', 'y', 'z']
```

The way Python iterates is, in a context where it needs to, it calls the `__iter__` method of the object (in this case the dictionary) which returns an iterator (in this case, a keyiterator object):

```
>>> d.__iter__()
<dict_keyiterator object at 0x7fb17
```

We shouldn't use these special methods ourselves, instead, use the respective builtin function to call it, `iter` :

```
>>> key_iterator = iter(d)
>>> key_iterator
<dict_keyiterator object at 0x7fb17
```

Iterators have a `__next__` method - but we call it with the builtin function, `next` :

```
>>> next(key_iterator)
'x'
>>> next(key_iterator)
'y'
>>> next(key_iterator)
'z'
>>> next(key_iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <modul
StopIteration
```

When an iterator is exhausted, it raises `StopIteration` . This is how Python knows to exit a `for` loop, or a list comprehension, or a generator expression, or any other iterative context. Once an iterator raises `StopIteration` it will always raise it - if you want to iterate again, you need a new one.

```
>>> list(key_iterator)
[]
>>> new_key_iterator = iter(d)
```

Returning to dicts

We've seen dicts iterating in many contexts. What we've seen is that any time we iterate over a dict, we get the keys. Back to the original example:

```
d = {'x': 1, 'y': 2, 'z': 3}
for key in d:
```

If we change the variable name, we still get the keys. Let's try it:

```
>>> for each_key in d:
...     print(each_key, '=>', d[eac
...
x => 1
y => 2
z => 3
```

If we want to iterate over the values, we need to use the `.values` method of dicts, or for both together, `.items` :

```
>>> list(d.values())
[1, 2, 3]
>>> list(d.items())
[('x', 1), ('y', 2), ('z', 3)]
```

In the example given, it would be more efficient to iterate over the items like this:

```
for a_key, corresponding_value in d
    print(a_key, corresponding_valu
```

But for academic purposes, the question's example is just fine.

answered Jun 21 '17 at 2:51
Aaron Hall ♦
185k 53 310 264

9

whatever, today, both python 2.6 and 2.7, as well as 3.x, in my box work well with `items()` :

```
z = {0: 'a', 1: 'b'}
for k, v in z.items(): print(v, k)
```

answered Jan 5 at 18:08
象嘉道
1,635 3 21 40

Don't know why this answer isn't more popular. Uses code to explain itself and works in both major python versions. –

behaviour is *different* as mentioned in the top answer - `z.items()` will return a `list` of `(key, value)` pairs in Python 2.7, effectively a copy of the original dict. – piit79 Feb 25 at 15:51

6

You can check the implementation of CPython's `dicttype` on GitHub. This is the signature of method that implements the dict iterator:

```
_PyDict_Next(PyObject *op, Py_ssize
             PyObject **pvalue, Py_
```

CPython dictobject.c

edited May 19 '18 at 18:38
Peter Mortensen
14k 19 87 114

answered Nov 3 '17 at 5:16
abc
8,914 24 90 147

2

To iterate over keys, it is slower but better to use `my_dict.keys()` . If you tried to do something like this:

```
for key in my_dict:
    my_dict[key+"-1"] = my_dict[key
```

it would create a runtime error because you are changing the keys while the program is running. If you are absolutely set on reducing time, use the `for key in my_dict` way, but you have been warned ;).

answered Dec 31 '15 at 18:39
Neil Chowdhury o_O
160 5

0

For key in my_dict is actually equal to for key in my_dict.keys(). So there if you want to get values of dict you can try two methods.

One:

```
for value in my_dict.values():
    print(value)
```

Two:

answered Mar 11 at 19:38

Amaan Durrani
11 3

protected by Antti Haapala Oct 13 '16 at 12:31

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?