Dictionaries: How to keep keys/values in same order as declared?

Ask Question

I have a dictionary that I declared in a particular order and want to keep it in that order all the time. The keys/values can't really be kept in order based on their value, I just want it in the order that I declared it.

So if I have the dictionary:

```
d = {'ac': 33, 'gw': 20, 'ap': 102, 'za': 321, 'bs': 10}
```

It isn't in that order if I view it or iterate through it, is there any way to make sure Python will keep the explicit order that I declared the keys/values in?

python dictionary order

edited Jun 4 at 3:04

martineau
59.9k 7 83 154

asked Dec 8 '09 at 15:53

roflwaffle
9,066 20 60 89

9 Can you clarify *why* you want to keep this "in order"? – Peter Hansen Dec 8 '09 at 16:01

3 I agree with Peter Hansen here. A dictionary is not meant to store order, but rather to store key/value pairs and have different access/add times to normal lists. – Zoran Pavlovic Jan 28 '13 at 9:24

26 @ZoranPavlovic: It seems Python's

science. – Michael Schlepel Jun 22 '16 at 15:53

2 Though still an implementation detail (meaning you should use `OrderedDict` until it is guaranteed), dicts in Python 3.6+ remember insertion order, see Dictionaries are ordered in Python 3.6+. – Jim Fasarakis Hilliard Aug 1 '17 at 13:06

10 Answers

```
from collections import OrderedDict
OrderedDict((word, True) for word in w
```

contains

```
OrderedDict([('He', True), ('will', Tr
True)])
```

If the values are `True` (or any other immutable object), you can also use:

```
OrderedDict.fromkeys(words, True)
```

answered Jun 11 '12 at 14:26

eumiro

115k 13 208 223

2 Worth noting, of course, that the 'immutable' part isn't a hard and fast rule that Python will enforce - its "only" a good idea. – lvc Jun 11 '12 at 14:37

7 be aware that solutions like: `OrderedDict(FUTURE=[], TODAY=[], PAST=[])` wont't work, when mentioned aproach: `OrderedDict([('FUTURE', []), ('TODAY', []), ('PAST', [])])` will keep order. – andilabs Jun 5 '14 at 13:47

1 @andi I got another problem,when using jsonify, the OrderedDict seems lost it's order when generate the json data.Anyway to solve this? – tyan Mar 31 '16 at 2:57

github.com/pallets/flask/issues/974 this

can be used to solve the problem.. – tyan Mar 31 '16 at 3:01

Rather than explaining the theoretical part, I'll give a simple example.

```
>>> from collections import OrderedDic
>>> my_dictionary=OrderedDict()
>>> my_dictionary['foo']=3
>>> my_dictionary['aol']=1
>>> my_dictionary
OrderedDict([('foo', 3), ('aol', 1)])
```

edited May 3 '16 at 18:59

twasbrillig
5,345 4 24 49

answered Jan 30 '15 at 7:29

Mohit Dabas
1,504 1 8 10

10 Is there a way to mass assign OrderedDict like the Dict type? – tyan Mar 31 '16 at 2:50

1 `OrderedDict` indeed solves the problem, but... in this particular example you get exactly the same result using a standard dictionary – Tonechas Apr 25 '16 at 19:08

2 @Tonechas: I just tried the example with a standard dictionary, and got `{'aol': 1, 'foo': 3}` So I think it's a good illustrative example. – twasbrillig May 3 '16 at 20:13

3 There's a lesson for everyone: it was discovered (I think around the 2.4 release) that Python's predictable hashing might give rise to security vulnerabilities, so now there's no guarantee that even two different runs of the same code will give the same ordering in a standard dict. – holdenweb Sep 1 '16 at 8:19

Why can't we pass some format of the order we want to mass initialize with our values? Instead of assigning in each line one value? (python2.7) – JavaSa Jun 6 '17 at 11:43

insertion order by default.

Defining

```
d = {'ac':33, 'gw':20, 'ap':102, 'za':
```

will result in a dictionary with the keys in the order listed in the source code.

This was achieved by using a simple array with integers for the sparse hash table, where those integers index into another array that stores the key-value pairs (plus the calculated hash). That latter array just happens to store the items in insertion order, and the whole combination actually uses less memory than the implementation used in Python 3.5 and before. See the original idea post by Raymond Hettinger for details.

In 3.6 this was still considered an implementation detail; see the *What's New in Python 3.6* documentation:

> The order-preserving aspect of this new implementation is considered an implementation detail and should not be relied upon (this may change in the future, but it is desired to have this new dict implementation in the language for a few releases before changing the language spec to mandate order-preserving semantics for all current and future Python implementations; this also helps preserve backwards-compatibility with older versions of the language where random iteration order is still in effect, e.g. Python 3.5).

Python 3.7 elevates this implementation detail to a *language specification*, so it is now mandatory that `dict` preserves order in all Python implementations compatible with that version or newer. See the pronouncement by the BDFL

additional functionality on top of the standard `dict` type. Such as as being reversible (this extends to the view objects), and supporting reordering (via the `move_to_end()` method).

edited Apr 9 at 21:24

answered Sep 16 '16 at 17:40

Martijn Pieters ♦
652k 111 2164 2066

That nice (change of) behaviour does not seem to be documented in docs.python.org/3/library/stdtypes.html#dict - I was looking for a hint on the order. docs.python.org/3/tutorial/datastructures.html#dictionaries mentions "unordered", "arbitrary order". – handle May 16 '17 at 10:17

1 @handle and that's because this is an implementation detail, as Chris notes. The What's new in Python 3.6 docs mention it. – Martijn Pieters ♦ May 16 '17 at 10:20

1 This solved my issue. I was using Python to sort and align a ton of things from a spreadsheet that was feeding another system. The order was critical. I updated to 3.6.4 (Latest Hombrew Version) and it came out in order without messing with it! – Wrenbjor Jan 15 at 16:57

1 @Chris_Rands: quite right, added that in. – Martijn Pieters ♦ Apr 9 at 21:24

python dictionaries are unordered. If you want an ordered dictionary, try collections.OrderedDict.

Note that OrderedDict was introduced into the standard library in python 2.7. If you have an older version of python, you can find recipes for

answered Jun 11 '12 at 14:25

mgilson
195k 38 372 491

see @martijn's post above. From python 3.6 onwards, dict supports insertion ordering. – tpk Jul 17 '17 at 10:28

1 @2943 Careful. So far, that's only for CPython and it's documented to be an implementation detail. `pypy` has been using this optimization for a little while ... Note that this also means that when other implementors (e.g. Jython) get to v3.6, they do not have to preserve insertion order in order to be compliant. Finally, it means that order preservation could be dropped when Raymond Hettinger comes up with the next generation of optimization for `dict` . IOW, for now, if you want an ordered dict, you should use OrderedDict. – mgilson Jul 18 '17 at 13:53

thanks for the suggestion – tpk Jul 19 '17 at 7:39

Dictionaries will use an order that makes searching efficient, and you cant change that,

You could just use a list of objects (a 2 element tuple in a simple case, or even a class), and append items to the end. You can then use linear search to find items in it.

Alternatively you could create or use a different data structure created with the intention of maintaining order.

answered Dec 8 '09 at 15:57

Fire Lancer
14.6k 21 84 149

I came across this post while trying to figure out how to get OrderedDict to work. PyDev for Eclipse couldn't find

to output my list, I just iterated through the tuple's values and plugged the iterated 'key' from the tuple into the dictionary to retrieve my values in the order I needed them.

example:

```
test_dict = dict( val1 = "hi", val2 =
test_tuple = ( 'val1', 'val2', 'val3',
for key in test_tuple: print(test_dict
```

It's a tad cumbersome, but I'm pressed for time and it's the workaround I came up with.

note: the list of lists approach that somebody else suggested does not really make sense to me, because lists are ordered and indexed (and are also a different structure than dictionaries).

edited Jun 27 '14 at 2:23

answered Jun 16 '14 at 21:17

smushface
81 1 2

Great solution. I will use it to write json to file, always in the same order. – Hrvoje T Apr 27 at 21:53

You can't really do what you want with a dictionary. You already have the dictionary `d = {'ac':33, 'gw':20, 'ap':102, 'za':321, 'bs':10}` created. I found there was no way to keep in order once it is already created. What I did was make a json file instead with the object:

```
{"ac":33,"gw":20,"ap":102,"za":321,"bs
```

I used:

```
r = json.load(open('file.json'), objec
```

to verify.

answered Sep 2 '16 at 1:05

nealous3

145 2 9

So why not start with an OrderedDict from a list? The JSON file doesn't really add anything here. – Martijn Pieters ♦ Jun 13 '17 at 6:21

Yes, list is more useful to keep order but the answer was in regards to the question about ordering dictionaries. Just letting people know about the limitations of using a dictionary and giving them a possible work around if they need to use a dictionary for some reason. – nealous3 Jun 14 '17 at 10:33

1 But that part is already covered by much older answers, dating back to 2012. – Martijn Pieters ♦ Jun 14 '17 at 10:38

Generally, you can design a class that behaves like a dictionary, mainly be implementing the methods `__contains__`, `__getitem__`, `__delitem__`, `__setitem__` and some more. That class can have any behaviour you like, for example prividing a sorted iterator over the keys ...

answered Dec 8 '09 at 16:09

Rasmus Kaj

3,032 14 20

I had a similar problem when developing a Django project. I couldn't use OrderedDict, because I was running an old version of python, so the simple solution was to use Django's SortedDict class:

https://code.djangoproject.com/wiki/SortedDict

if you would like to have a dictionary in a specific order, you can also create a list of lists, where the first item will be the key, and the second item will be the value and will look like this example

```
>>> list =[[1,2],[2,3]]
>>> for i in list:
...     print i[0]
...     print i[1]

1
2
2
3
```

edited Jul 4 '13 at 21:48

eyquem
18.7k 4 25 36

answered Jul 4 '13 at 20:47

pelos
447 2 5 15

5 That is not a "dictionary" because you cannot lookup items by their key without searching through the entire collection (taking O(n) time). – BHSPitMonkey May 1 '14 at 0:09

1 Yes, it is not a dictionary, but, depending on the situation, it could provide a valid solution the problem of the original poster. – SunSparc Aug 25 '15 at 21:21

he didn't say exactly how we wanted, just that want to be able to order them =), as always there is plenty of ways to do one thing. – pelos Mar 22 at 15:54