Add new keys to a dictionary?

Ask Question

2206

405

Is it possible to add a key to a Python dictionary after it has been created? It doesn't seem to have an `.add()` method.

python dictionary

edited Apr 6 '17 at 1:47

martineau
70.6k 10 93 187

asked Jun 21 '09 at 22:07

lfaraone
18.1k 15 45 66

4 If you are here trying to figure out how to add a key *and* return a *new* dictionary, you can do this on python3.5+: `{**mydict, 'new_key': new_val}` . – cs95 Dec 19 '18 at 13:17

15 Answers

```
print(d)
# {'mynewkey': '
```

dited Jan 4 at 13:44

Nick Larsen ♦
14.7k 6 55 85

nswered Jun 21 '09 at 22:09

Paolo Bergantino
386k 74 494 426

48 The explanation is: you create a new key\value pair on a dictionary by assigning a value to that key. If the key doesn't exist, it's added and points to that value. If it exists, the current value it points to is overwritten. – R. Navega May 31 '18 at 1:22

2 What is the difference between this and the `.update()` method? Which is better when? – hegash Jan 16 at 19:14

2 @hegash the `d[key]=val` syntax as it is shorter and can handle any object as key (as long it is hashable), and only sets one value, whereas the `.update(key1=val1, key2=val2)` is nicer if you want to set

strings (since kwargs are converted to strings). `dict.update` can also take another dictionary, but I personally prefer not to explicitly create a new dictionary in order to update another one. – bgusach Feb 13 at 8:38

922

```
>>> x = {1:2}
>>> print x
{1: 2}

>>> x.update({3:4
>>> print x
{1: 2, 3: 4}
```

nswered Jul 22 '09 at 14:48

user142911

10 this is soooo inefficient to create a dictionary just to update one key. Do this only if you have more than 1 key (there may be a threshold above which it's better to create a dict) – Jean-François Fa Aug 1 '18 at 21:42

@Jean-FrançoisFabre

covering all cases. – Robert Brisita Mar 27 at 5:59

it gives the wrong impression that it's the preferred way to add one key. – Jean-François Fa Mar 27 at 6:14

773

I feel like consolidating info about Python dictionaries:

Creating an empty dictionary

```
data = {}
# OR
data = dict()
```

Creating a dictionary with initial values

```
data = {'a':1,'b'
# OR
data = dict(a=1,
# OR
data = {k: v for
```

Inserting/Updating a single value

```
data['a']=1  # Up
# OR
data.update({'a':
# OR
data.update(dict(
# OR
data.update(a=1)
```

Inserting/Updati

```
data.update({'c':
```

Creating a merged dictionary without modifying originals

```
data3 = {}
data3.update(data
data3.update(data
```

Deleting items in dictionary

```
del data[key]  # 
data.pop(key)  # 
data.clear()  # C
```

Check if a key is already in dictionary

```
key in data
```

Iterate through pairs in a dictionary

```
for key in data: 
for key, value in
for key in d.keys
for value in d.va
```

Create a dictionary from 2 lists

```
data = dict(zip(l
```

Feel free to add more!

dited May 11 '18 at 19:28

Community ♦

23.8k 35 111 180

From python 3.5 you can also create a merged dictionary like this: `data3 = {**data1, **data2}` . – Sławomir Górawsl Feb 12 at 10:41

150

Yeah, it's pretty easy. Just do the following:

```
dict["key"] = "va
```

nswered Jun 21 '09 at 22:09

John Slavick
7,707 3 20 21

122

"Is it possible to add a key to a Python dictionary after it has been created? It doesn't seem to have an .add() method."

Yes it is possible, and it does have a method that implements this,

To demonstrate how and how not to use it, let's create an empty dict with the dict literal, `{}` :

```
my_dict = {}
```

Best Practice 1: Subscript notation

To update this dict with a single new key and value, you can use the subscript notation (see Mappings here) that provides for item assignment:

```
my_dict['new key'
```

`my_dict` is now:

```
{'new key': 'new '
```

Best Practice 2: The `update` method - 2 ways

We can also update the dict with multiple values efficiently as well using the `update` method. We may be unnecessarily creating an extra `dict` here, so we hope our `dict` has already been created and came from or was used for another purpose:

`my_dict` is now:

```
{'key 2': 'value
```

Another efficient way of doing this with the update method is with keyword arguments, but since they have to be legitimate python words, you can't have spaces or special symbols or start the name with a number, but many consider this a more readable way to create keys for a dict, and here we certainly avoid creating an extra unnecessary `dict`:

```
my_dict.update(fo
```

and `my_dict` is now:

```
{'key 2': 'value
 'foo': 'bar', 'f
```

So now we have covered three Pythonic ways of updating a `dict`.

Magic method, `__setitem__`, and why it should be avoided

There's another

which uses the `__setitem__` method. Here's an example of how one might use the `__setitem__` method to add a key-value pair to a `dict` , and a demonstration of the poor performance of using it:

```
>>> d = {}
>>> d.__setitem__
>>> d
{'foo': 'bar'}
```

```
>>> def f():
...     d = {}
...     for i in 
...         d['fo
...
>>> def g():
...     d = {}
...     for i in 
...         d.__s
...
>>> import timeit
>>> number = 100
>>> min(timeit.re
0.002088069915771
>>> min(timeit.re
0.005071878433227
```

So we see that using the subscript notation is actually much faster than using `__setitem__` . Doing the Pythonic thing, that is, using the language in the way it was intended to be used, usually is both more readable and computationally efficient.

dited Jun 17 '16 at 11:28

Aaron Hall ♦
185k 53 310 264

79

```
dictionary[key] =
```

nswered Jun 21 '09 at 22:08

Jason Creighton
13.8k 8 30 32

36 Vanuan: speed is not the only requirement for a good answer. An appropriate level of detail is also vital. – naught101 Dec 23 '13 at 23:20

50

If you want to add a dictionary within a dictionary you can do it this way.

Example: Add a new entry to your dictionary & sub dictionary

```
dictionary = {}
dictionary["new k
dictionary["dicti
dictionary["dicti
"dictionary"}
print (dictionary
```

Output:

```
{'new key': 'some
{'other': 'dictio
```

NOTE: Python requires that you first add a sub

```
dictionary["dicti
```

dited Nov 10 '17 at 20:38
user8909115

nswered Apr 26 '12 at 19:04
htmlfarmer
1,574 4 20 33

10 this is as irrelevant to the question asked as most of the comments in php.net manual pages... – Erik Allik Jun 1 '12 at 21:05

1 Nothing to stop you doing this on one line: `dictionary = {"dictionary_within_a_dictionary": {"sub_dict": {"other" : "dictionary"}}}` (or if `dictionary` is already a dict, `dictionary["dictionary_within_a_dictionary"] = {"sub_dict": {"other" : "dictionary"}}`) – Chris Dec 27 '17 at 16:07

36

The orthodox syntax is `d[key] = value` , but if your keyboard is missing the square bracket keys you could do:

```
d.__setitem__(key
```

In fact, defining `__getitem__` and `__setitem__`

the square bracket syntax. See http://www.diveintopython.net/object_oriented_framework/special_class_methods.html

dited Sep 15 '14 at 8:56

nswered Apr 14 '13 at 0:58

Colonel Panic
82.6k 62 306 396

12 I would find it extremely difficult to program with python without the bracket keys on my keyboard. – Bobort Oct 27 '16 at 14:10

3 Or most programming languages when dealing with arrays... – Shadow Dec 8 '17 at 3:33

1 This was the only way I could find to set dictionary values within a list comprehension. Thanks – chris stevens May 18 '18 at 13:29

2 @chrisstevens if you want to set a value in a comprehension a hack I've used is `[a for a in my_dict if my_dict.update({'a': 1}) is None]` . – Jeremy Logan Sep 13 '18 at 22:45

30

you can create one

```
class myDict(dict

    def __init__(
        self = di

    def add(self,
        self[key]

## example

myd = myDict()
myd.add('apples',
myd.add('bananas'
print(myd)
```

gives

```
>>>
{'apples': 6, 'ba
```

nswered May 25 '13 at 13:33

octoback
14.3k 29 107 184

30

This popular question addresses *functional* methods of merging dictionaries `a` and `b`.

Here are some of the more straightforward methods (tested in Python 3)...

```
c = dict( a, **b
c = dict( list(a.
c = dict( i for d
```

Note: The first method above only works if the keys in `b` are strings.

To add or modify a single element, the `b` dictionary

This is equivalent to...

```
def functional_di
    temp = diction
    temp[key] = va
    return temp

c = functional_di
```

dited May 23 '17 at 11:55

Community ♦
1 1

nswered Aug 17 '13 at 23:04

nobar
27.4k 10 87 100

6 Interesting comment about the first method from Python's BDFL (here). – nobar Aug 17 '13 at 23:09

18

Let's pretend you want to live in the immutable world and do NOT want to modify the original but want to create a new `dict` that is the result of adding a new key to the original.

In Python 3.5+ you can do:

```
params = {'a': 1,
new_params = {**p
```

The Python 2 equivalent is:

```
params = {'a': 1,
new_params = dict
```

`params` is still equal to `{'a': 1, 'b': 2}`

and

`new_params` is equal to `{'a': 1, 'b': 2, 'c': 3}`

There will be times when you don't want to modify the original (you only want the result of adding to the original). **I find this a refreshing alternative to the following:**

```
params = {'a': 1,
new_params = para
new_params['c'] =
```

or

```
params = {'a': 1,
new_params = para
new_params.update
```

Reference: https://stackoverflow.com/a/2255892/514866

nswered Jan 12 '18 at 19:31

campeterson
2,321 1 18 24

1 In a lengthy conversation with a pro-functional programming colleague of mine a good point was brought up. One downside to the above approach is that if someone reading the code is not familiar with the `**` in

There are times when you'll favor a less functional approach for better readability. – campeterson Jan 30 '18 at 17:05

3 We cannot foresee what subset of the Python language our readers know, so it's fair to assume they know the entire language so they search on the docs for the parts they don't. – Gabriel Mar 7 '18 at 20:56

12

So many answers and still everybody forgot about the strangely named, oddly behaved, and yet still handy `dict.setdefault()`

This

```
value = my_dict.s
```

basically just does this:

```
try:
    value = my_di
except KeyError:
    value = my_di
```

e.g.

```
>>> mydict = {'a'
>>> mydict.setdef
4 # returns new v
>>> print(mydict)
{'a':1, 'b':2, 'c
# but see what ha
>>> mydict.setdef
```

nswered Oct 23 '17 at 14:03

Michael Ekoka
9,467 5 51 65

4

You can do by `dict.update(Iterable_Sequence of key:value)`

Example:

```
wordFreqDic.updat
```

nswered Dec 12 '18 at 6:12

Lokesh Soni
51 4

If you are using literals, I think it is better to stick to setting keys directly, as you don't have to create intermediate dictionaries. – bgusach Feb 13 at 8:40

2

If you're not joining two dictionaries, but adding new key-value pairs to a dictionary, then using the subscript notation seems like the best way.

```
import timeit

timeit.timeit('di
123123, "asd": 23
>> 0.495825052261

timeit.timeit('di
123123; dictionar
>> 0.207828998565
```

However, if you'd

key-value pairs, you should consider using the `update()` method.

dited Oct 11 '18 at 9:03

nswered Oct 11 '18 at 8:57

Burak Özdemir
166 12

2

first to check whether the key already exists

```
a={1:2,3:4}
a.get(1)
2
a.get(5)
None
```

then you can add the new key and value

nswered Mar 25 at 13:01

Agus Mathew
162 11

protected by Marcin Sep 20 '13 at 19:06

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