How can I perform set operations on Python dictionaries?

While it is incredibly useful to be able to do set operations between the keys of a dictionary, I often wish that I could perform the set operations on the dictionaries themselves.

I found some recipes for taking the difference of two dictionaries but I found those to be quite verbose and felt there must be more pythonic answers.

python dictionary

asked Jul 17 '13 at 7:57

snth
1,683 1 19 36

3 Answers

tl;dr Recipe: `{k:d1.get(k, k in d1 or d2[k]) for k in set(d1) | set(d2)}` and `|` can be replaced with any other set operator.

Based @torek's comment, another recipe that might be easier to remember (while being fully general) is: `{k:d1.get(k,d2.get(k)) for k in set(d1) | set(d2)}`.

Full answer below:

My first answer didn't deal correctly with values that evaluated to False. Here's an improved version which deals with Falsey values:

```
>>> d1 = {'one':1, 'both':3, 'falsey_one':False, 'falsey_both':None}
>>> d2 = {'two':2, 'both':30, 'falsey_two':None, 'falsey_both':False}
>>>
>>> print "d1 - d2:", {k:d1[k] for k in d1 if k not in d2}                 # 0
d1 - d2: {'falsey_one': False, 'one': 1}
>>> print "d2 - d1:", {k:d2[k] for k in d2 if k not in d1}                 # 1
d2 - d1: {'falsey_two': None, 'two': 2}
>>> print "intersection:", {k:d1[k] for k in d1 if k in d2}                      #
2
intersection: {'both': 3, 'falsey_both': None}
>>> print "union:", {k:d1.get(k, k in d1 or d2[k]) for k in set(d1) | set(d2)}   #
3
union: {'falsey_one': False, 'falsey_both': None, 'both': 3, 'two': 2, 'one': 1,
'falsey_two': None}
```

The version for `union` is the most general and can be turned into a function:

```
>>> def dict_ops(d1, d2, setop):
...     """Apply set operation `setop` to dictionaries d1 and d2
...
...     Note: In cases where values are present in both d1 and d2, the value from
...     d1 will be used.
...     """
...     return {k:d1.get(k,k in d1 or d2[k]) for k in setop(set(d1), set(d2))}
...
>>> print "d1 - d2:", dict_ops(d1, d2, lambda x,y: x-y)
d1 - d2: {'falsey_one': False, 'one': 1}
>>> print "d2 - d1:", dict_ops(d1, d2, lambda x,y: y-x)
d2 - d1: {'falsey_two': None, 'two': 2}
>>> import operator as op
>>> print "intersection:", dict_ops(d1, d2, op.and_)
intersection: {'both': 3, 'falsey_both': None}
>>> print "union:", dict_ops(d1, d2, op.or_)
union: {'falsey_one': False, 'falsey_both': None, 'both': 3, 'two': 2, 'one': 1,
'falsey_two': None}
```

Where items are in both dictionaries, the value from `d1` will be used. Of course we can return the value from `d2` instead by changing the order of the function arguments.

```
>>> print "union:", dict_ops(d2, d1, op.or_)
union: {'both': 30, 'falsey_two': None, 'falsey_one': False, 'two': 2, 'one': 1,
'falsey_both': False}
```

edited Jul 17 '13 at 9:33

answered Jul 17 '13 at 8:31

snth
1,683 1 19 36

Heh, `k in d1 or d2[k]` as a default value if `k` isn't in `d1` is pretty cool: it avoids evaluating `d2[k]` exactly whenever `k` is in `d1` so that the second argument to `d1.get` is not needed :-) (Note that `d2.get(k)` would also work but requires looking in `d2` ; not sure if that's really any less efficient in the end.) – torek Jul 17 '13 at 9:07

@torek Thanks. Based on what you said the following also works and might be the easiest to remember: `{k:d1.get(k,d2.get(k)) for k in set(d1) | set(d2)}` .– snth Jul 17 '13 at 9:27

EDIT: The recipes here don't deal correctly with False values. I've submitted another improved answer.

Here are some recipes I've come up with:

```
>>> d1 = {'one':1, 'both':3}
>>> d2 = {'two':2, 'both':30}
>>>
>>> print "d1 only:", {k:d1.get(k) or d2[k] for k in set(d1) - set(d2)}     # 0
d1 only: {'one': 1}
>>> print "d2 only:", {k:d1.get(k) or d2[k] for k in set(d2) - set(d1)}     # 1
d2 only: {'two': 2}
>>> print "in both:", {k:d1.get(k) or d2[k] for k in set(d1) & set(d2)}     # 2
in both: {'both': 3}
>>> print "in either:", {k:d1.get(k) or d2[k] for k in set(d1) | set(d2)}   # 3
in either: {'both': 3, 'two': 2, 'one': 1}
```

While the expressions in #0 and #2 could be made simpler, I like the generality of this expression which allows me to copy and paste this recipe everywhere and simply change the set operation at the end to what I require.

Of course we can turn this into a function:

```
>>> def dict_ops(d1, d2, setop):
...     return {k:d1.get(k) or d2[k] for k in setop(set(d1), set(d2))}
...
>>> print "d1 only:", dict_ops(d1, d2, lambda x,y: x-y)
d1 only: {'one': 1}
>>> print "d2 only:", dict_ops(d1, d2, lambda x,y: y-x)
d2 only: {'two': 2}
>>> import operator as op
>>> print "in both:", dict_ops(d1, d2, op.and_)
in both: {'both': 3}
>>> print "in either:", dict_ops(d1, d2, op.or_)
in either: {'both': 3, 'two': 2, 'one': 1}
>>> print "in either:", dict_ops(d2, d1, lambda x,y: x|y)
in either: {'both': 30, 'two': 2, 'one': 1}
```

edited Dec 22 '13 at 21:28

Benjamin
13.6k 23 103 202

answered Jul 17 '13 at 7:57

snth
1,683 1 19 36

Watch out for cases where `d1[k]` exists but `bool(d1[k])` is `False` , e.g., if `d1['both'] = 0` you get `d2['both']` . This seems entirely valid—if it's in both dictionaries, which value is the "right" one?—but if you're expecting to get the value from `d1` and you *usually* get the values from d1, this could be a surprise. – torek Jul 17 '13 at 8:07

Your answer would be more useful if you labeled the operations the same as the equivalent set operation -- like union, intersection, difference, etc. – martineau Jul 17 '13 at 8:08

@torek You're right about the False values. I've submitted a new answer that hopefully deals with these correctly. I didn't edit this answer because I think the new answer is too different and people had already voted on it. – snth Jul 17 '13 at 8:34

@martineau Thanks, I've relabeled the output in my new answer. – snth Jul 17 '13 at 8:35

Here are some more:

Set addition `d1 + d2`

```
{key: value for key, value in d1.items() + d2.items()}
# here values that are present in `d1` are replaced by values in `d2`
```

Alternatively,

```
d3 = d1.copy()
d3.update(d2)
```

Set difference `d1 - d2`

```
{key: value for key, value in d1.items() if key not in d2}
```

edited May 5 at 14:05

answered Jul 17 '13 at 8:17

Joel Cornett
15.4k 2 32 60

1 I think your `d3` is a union, not an intersection. – snth Jul 17 '13 at 8:32

Yeah, you're right :P – Joel Cornett Jul 17 '13 at 8:35

And your set difference is set intersection. – Asterios May 5 at 8:10

@Asterios thanks for pointing that out! I have fixed it. – Joel Cornett May 5 at 14:08