
nested_dict Documentation

Release 1.5.0

Leo Goodstadt

June 05, 2015

CONTENTS

1	Working without <i>nested_dict</i>	3
2	How to use <i>nested_dict</i>	5
2.1	Flexible levels of nesting	5
2.2	Fixed levels of nesting and set types	5
2.3	Set maximum nesting	6
3	Iterating <i>nested_dict()</i>	7
4	Converting back to dictionaries	9
4.1	<i>nested_dict</i>	9
	Python Module Index	13
	Index	15

nested_dict provides dictionaries with multiple levels of nested-ness:

```
from nested_dict import *  
  
nd = nested_dict()  
  
nd["a"]["b"]["c"] = 311  
nd["d"]["e"] = 311
```

Each nested level is create magically when accessed, a process known as “auto-vivification” in perl.

WORKING WITHOUT *NESTED_DICT*

`defaultdict` from the python `collections` module provides for one or (with some effort) two levels of nestedness For example, here is a dictionary of `sets` with `defaultdict` :

For one level of nesting:

```
from collections import defaultdict
one_level_dict = defaultdict(set)
one_level_dict["1st group"].add(3)
one_level_dict["2nd group"].add(4)
one_level_dict["2nd group"].add(5)
```

For two levels of nesting:

```
from collections import defaultdict
two_level_dict = defaultdict(lambda: defaultdict(set))
two_level_dict["1st group"]["A"].add(3)
two_level_dict["2nd group"]["B"].add(4)
two_level_dict["2nd group"]["C"].add(5)
```

The syntax becomes rapidly more ugly with additional levels of nesting, and it is difficult to mix dictionaries with different levels of nestedness.

HOW TO USE *NESTED_DICT*

2.1 Flexible levels of nesting

```
from nested_dict import nested_dict
nd= nested_dict()
nd["mouse"]["chr1"]["+"] = 311
nd["mouse"]["chromosomes"]="completed"
nd["mouse"]["chr2"] = "2nd longest"
nd["mouse"]["chr3"] = "3rd longest"

for k, v in nd.iteritems_flat():
    print "%-30s==%20r" % (k,v)
```

Gives:

```
('mouse', 'chr3')          == '3rd longest'
('mouse', 'chromosomes')   == 'completed'
('mouse', 'chr2')          == '2nd longest'
('mouse', 'chr1', '+')     == 311
```

2.2 Fixed levels of nesting and set types

This is necessary if you want the nested dictionary to hold a collection (like the `set` in the first example) or scalar such as `int` or `str` with useful default values.

```
# nested dict of lists
nd = nested_dict(2, list)
nd["mouse"]["2"].append(12)
nd["human"]["1"].append(12)

# nested dict of sets
nd = nested_dict(2, set)
nd["mouse"]["2"].add("a")
nd["human"]["1"].add("b")

# nested dict of ints
nd = nested_dict(2, int)
nd["mouse"]["2"] += 4
nd["human"]["1"] += 5
nd["human"]["1"] += 6

nd.to_dict()
```

```
#{'human': {'1': 11}, 'mouse': {'2': 4}}

#   nested dict of strings
nd = nested_dict(2, str)
nd["mouse"] ["2"] += "a" * 4
nd["human"] ["1"] += "b" * 5
nd["human"] ["1"] += "c" * 6

nd.to_dict()
#{'human': {'1': 'bbbbbbcccccc'}, 'mouse': {'2': 'aaaa'}}
```

2.3 Set maximum nesting

You can also specify a maximum level of nesting even if you do not want to specify the stored type. For example, if you know beforehand that your data involves a **maximum** of four nested sub levels, you can add this (very minimal) constraint ahead of time:

```
nd4 = nested_dict(4)
# OK: Assign to "string"
nd4[1][2][3][4]="a"

# Bad: Five levels is one too many
nd4[1][2][3]["four"][5]="b"
#
# KeyError
# ----> nd4[1][2][3]["four"][5]="b"
#
# KeyError: 'four'
#

# OK: Assign to fewer levels is fine
nd4[1]["two"] = 3

# But like with normal dicts, you can't "extend a value" later
nd4[1]["two"][4] = 3

# TypeError
# ----> nd4[1]["two"][4] = 3
#
# TypeError: 'int' object does not support item assignment
```

ITERATING *NESTED_DICT()*

You can use nested iterators to iterate through *nested_dict* just like ordinary python *dicts*

```
from nested_dict import nested_dict
nd= nested_dict()
nd["mouse"]["chr1"]["+"] = 311
nd["mouse"]["chromosomes"]="completed"
nd["mouse"]["chr2"] = "2nd longest"
nd["mouse"]["chr3"] = "3rd longest"

for key1, value1 in nd.items():
    for key2, value2 in value1.items():
        print (key1, key2, str(value2))

# ('mouse', 'chr3', '3rd longest')
# ('mouse', 'chromosomes', 'completed')
# ('mouse', 'chr2', '2nd longest')
# ('mouse', 'chr1', '{ "+": 311}')
```

This is less useful if you do not know beforehand how many levels of nesting you have.

Instead, you can use *iteritems_flat()*, *iterkeys_flat()*, and *itervalues_flat()*. The *_flat()* functions are just like their normal counterparts except they compress all the nested keys into *tuples*:

```
from nested_dict import nested_dict
nd= nested_dict()
nd["mouse"]["chr1"]["+"] = 311
nd["mouse"]["chromosomes"]="completed"
nd["mouse"]["chr2"] = "2nd longest"
nd["mouse"]["chr3"] = "3rd longest"

for keys_as_tuple, value in nd.iteritems_flat():
    print ("%30s == %20r" % (keys_as_tuple, value))
# ('mouse', 'chr3') == '3rd longest'
# ('mouse', 'chromosomes') == 'completed'
# ('mouse', 'chr2') == '2nd longest'
# ('mouse', 'chr1', '+') == 311
```


CONVERTING BACK TO DICTIONARIES

It is often useful to convert away the magic of *nested_dict*, for example, to [pickle](#) the dictionary.

Use *nested_dict.to_dict()*

```
from nested_dict import nested_dict
nd= nested_dict()
nd["mouse"]["chr1"]["+"] = 311
nd["mouse"]["chromosomes"]="completed"
nd.to_dict()
# {'mouse': {'chr1': {'+': 311}, 'chromosomes': 'completed'}}
```

4.1 nested_dict

4.1.1 nested_dict

class nested_dict.**nested_dict**

__init__ ([*nested_level*, *value_type*])

Parameters

- **nested_level** – the level of nestedness in the dictionary
- **collection_type** – the type of the values held in the dictionary

For example,

```
a = nested_dict(3, list)
a['level 1']['level 2']['level 3'].append(1)

b = nested_dict(2, int)
b['level 1']['level 2']+=3
```

If *nested_level* and *value_type* are not defined, the degree of nested-ness is not fixed. For example,

```
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15
```

iteritems_flat ()

iterate through values with nested keys flattened into a tuple

For example,

```
from nested_dict import nested_dict
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15
```

```
print list(a.iteritems_flat())
```

Produces:

```
[      (('1', '2', '3'), 3),
      (('A', 'B'), 15)
]
```

iterkeys_flat()

iterate through values with nested keys flattened into a tuple

For example,

```
from nested_dict import nested_dict
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15

print list(a.iterkeys_flat())
```

Produces:

```
[('1', '2', '3'), ('A', 'B')]
```

intervalues_flat()

iterate through values as a single list, without considering the degree of nesting

For example,

```
from nested_dict import nested_dict
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15

print list(a.intervalues_flat())
```

Produces:

```
[3, 15]
```

to_dict()

Converts the nested dictionary to a nested series of standard dict objects

For example,

```
from nested_dict import nested_dict
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15

print a.to_dict()
```

Produces:

```
{'1': {'2': {'3': 3}}, 'A': {'B': 15}}
```

`__str__([indent])`

The dictionary formatted as a string

Parameters *indent* – The level of indentation for each nested level

For example,

```
from nested_dict import nested_dict
a = nested_dict()
a['1']['2']['3'] = 3
a['A']['B'] = 15

print a
print a.__str__(4)
```

Produces:

```
{
  "1": {
    "2": {
      "3": 3
    }
  },
  "A": {
    "B": 15
  }
}
```

4.1.2 Acknowledgements

Inspired in part from ideas in: <http://stackoverflow.com/questions/635483/what-is-the-best-way-to-implement-nested-dictionaries-in-python> contributed by nosklo

Many thanks

4.1.3 Copyright

The code is licensed under the MIT Software License <http://opensource.org/licenses/MIT>

This essentially only asks that the copyright notices in this code be maintained for **source** distributions.

n

nested_dict, 9

Symbols

`__init__()` (`nested_dict.nested_dict` method), 9
`__str__()` (`nested_dict.nested_dict` method), 11

I

`iteritems_flat()` (`nested_dict.nested_dict` method), 9
`iterkeys_flat()` (`nested_dict.nested_dict` method), 10
`itervalues_flat()` (`nested_dict.nested_dict` method), 10

N

`nested_dict` (class in `nested_dict`), 9
`nested_dict` (module), 9

T

`to_dict()` (`nested_dict.nested_dict` method), 10