
nested_dict Documentation

Release 1.61

Leo Goodstadt

June 18, 2015

CONTENTS

1	Drop in replacement for <code>dict</code>	3
2	Specifying the contained type	5
2.1	<i>dict</i> of <code>lists</code>	5
2.2	<i>dict</i> of <code>sets</code>	5
2.3	<i>dict</i> of <code>ints</code>	5
2.4	<i>dict</i> of <code>strs</code>	5
3	Iterating through <code>nested_dict</code>	7
4	Converting to / from dictionaries	9
5	Updating with another dictionary	11
6	<code>defaultdict</code>	13
6.1	<code>nested_dict</code>	13
	Python Module Index	17
	Index	19

Note:

- Source code at <https://github.com/bunbun/nested-dict>
 - Documentation at <http://nested-dict.readthedocs.org>
-

`nested_dict` extends `defaultdict` to support python `dict` with multiple levels of nested-ness:

DROP IN REPLACEMENT FOR `DICT`

```
>>> from nested_dict import nested_dict
>>> nd= nested_dict()
>>> nd["one"] = "1"
>>> nd[1]["two"] = "1 / 2"
>>> nd["uno"][2]["three"] = "1 / 2 / 3"
>>>
... for keys_as_tuple, value in nd.items_flat():
...     print ("%20s == %r" % (keys_as_tuple, value))
...
('one',)           == '1'
(1, 'two')         == '1 / 2'
('uno', 2, 'three') == '1 / 2 / 3'
```


SPECIFYING THE CONTAINED TYPE

If you want the nested dictionary to hold

- a collection (like the `set` in the first example) or
- a scalar with useful default values such as `int` or `str`.

2.1 *dict* of lists

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

2.2 *dict* of sets

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

2.3 *dict* of ints

```
# 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}}
```

2.4 *dict* of strs

```
#    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'}}
```

ITERATING THROUGH NESTED_DICT

Iterating through deep or unevenly nested dictionaries is a bit of a pain without recursion. `nested_dict` allows you to **flatten** the nested levels into **tuples** before iteration.

You do not need to know beforehand how many levels of nesting you have:

```
from nested_dict import nested_dict
nd= nested_dict()
nd["one"] = "1"
nd[1]["two"] = "1 / 2"
nd["uno"][2]["three"] = "1 / 2 / 3"

for keys_as_tuple, value in nd.items_flat():
    print ("%20s == %r" % (keys_as_tuple, value))

#  (1, 'two')           == '1 / 2'
#  ('one',)             == '1'
#  ('uno', 2, 'three')  == '1 / 2 / 3'
```

nested_dict provides

- `items_flat()`
- `keys_flat()`
- `values_flat()`

(`iteritems_flat()`, `iterkeys_flat()`, and `itervalues_flat()` are python 2.7-style synonyms.)

CONVERTING TO / FROM DICTIONARIES

The magic of `nested_dict` sometimes gets in the way (of [pickling](#) for example).

We can convert to and from a vanilla python dict using

- `nested_dict.to_dict()`
- `nested_dict` constructor

```
>>> from nested_dict import nested_dict
>>> nd= nested_dict()
>>> nd["one"] = 1
>>> nd[1]["two"] = "1 / 2"

#
#   convert nested_dict -> dict and pickle
#
>>> nd.to_dict()
{1: {'two': '1 / 2'}, 'one': 1}
>>> import pickle
>>> binary_representation = pickle.dumps(nd.to_dict())

#
#   convert dict -> nested_dict
#
>>> normal_dict = pickle.loads(binary_representation)
>>> new_nd = nested_dict(normal_dict)
>>> nd == new_nd
True
```


UPDATING WITH ANOTHER DICTIONARY

You can use the `nested_dict.update(other)` method to merge in the contents of another dictionary.

If the `nested_dict` has a fixed level of nestedness and a `value_type`, then this is ignored for the key/value pairs from `other` but otherwise preserved as far as possible.

For example, given a three level nested dictionary of integers:

```
>>> d1 = nested_dict.nested_dict(3, int)
>>> d1[1][2][3] = 4
>>> d1[1][2][4] = 5

>>> # integers have a default value of zero
>>> default_value = d1[1][2][5]
>>> print (default_value)
0
>>> print (d1.to_dict())
{1: {2: {3: 4, 4: 5, 5: 0}}}
```

We can update this with any dictionary, not necessarily a three level `nested_dict` of `int`.

```
>>> # some other nested_dict
>>> d2 = nested_dict.nested_dict()
>>> d2[2][3][4][5] = 6
>>> d1.update(d2)
>>> print (d1.to_dict())
{1: {2: {3: 4, 4: 5, 5: 0}}, 2: {3: {4: {5: 6}}}}
```

However, the rest of the dictionary still has the same default value type at the specified level of nestedness

```
>>> print (d1[2][3][4][5])
6
>>> # integers have a default value of zero
>>> print (d1[2][3][5])
0
```


DEFAULTDICT

`nested_dict` extends `collections.defaultdict`

You can get arbitrarily-nested “auto-vivifying” dictionaries using `defaultdict`.

```
from collections import defaultdict
nested_dict = lambda: defaultdict(nested_dict)
nd = nested_dict()
nd[1][2]["three"][4] = 5
nd["one"]["two"]["three"][4] = 5
```

However, only `nested_dict` supports a dict of dict of sets etc.

6.1 nested_dict

6.1.1 Class documentation

`class nested_dict.nested_dict`

`nested_dict.__init__([existing_dict | nested_level, value_type])`

Parameters

- **existing_dict** – an existing dict to be converted into a `nested_dict`
- **nested_level** – the level of nestedness in the dictionary
- **value_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
```

`nested_dict.update(other)`

Updates the dictionary recursively with the key/value pairs from *other*, overwriting existing keys.
Return None.

If the `nested_dict` has a fixed level of nestedness and a `value_type`, then this is ignored for the key/value pairs from *other* but otherwise preserved as far as possible.

`nested_dict.iteritems_flat()`

python 2.7 style synonym for `items_flat()`

`nested_dict.items_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.items_flat())
```

Produces:

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

`nested_dict.iterkeys_flat()`

python 2.7 style synonym for `keys_flat()`

`nested_dict.keys_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.keys_flat())
```

Produces:

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

`nested_dict.itervalues_flat()`

python 2.7 style synonym for `values_flat()`

`nested_dict.values_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.values_flat())
```

Produces:

```
[3, 15]
```

`nested_dict.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}}
```

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

6.1.2 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

6.1.3 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, [13](#)

Symbols

`__init__()` (`nested_dict.nested_dict` method), 13
`__str__()` (in module `nested_dict`), 15

I

`items_flat()` (in module `nested_dict`), 14
`iteritems_flat()` (in module `nested_dict`), 14
`iterkeys_flat()` (in module `nested_dict`), 14
`itervalues_flat()` (in module `nested_dict`), 14

K

`keys_flat()` (in module `nested_dict`), 14

N

`nested_dict` (class in `nested_dict`), 13
`nested_dict` (module), 13

T

`to_dict()` (in module `nested_dict`), 15

U

`update()` (in module `nested_dict`), 14

V

`values_flat()` (in module `nested_dict`), 14