nested_dict Documentation

Release 1.5.1

Leo Goodstadt

June 08, 2015

CONTENTS

Source code at https://github.com/bunbun/nested-dict

`nested_dict` provides dictionaries with multiple levels of nested-ness:

```
from nested_dict import nested_dict

nd = nested_dict()

nd["a"]["b"]["c"] = 311
nd["d"]["e"] = 311
```

Each nested level is created magically when accessed, a process known as "auto-vivification" in perl.

CHAPTER

ONE

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 defaultdict of sets:

```
    # One level of nesting
    from collections import defaultdict
    one_level_dict = defaultdict(set)
    one_level_dict["1st group"].add(3)

    # Two levels of nesting
    two_level_dict = defaultdict(lambda: defaultdict(set))
    two_level_dict["1st group"]["A"].add(3)
```

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

CHAPTER

TWO

HOW TO USE `nested_dict`

Just use `nested_dict` as a drop in replacement for `dict`

2.1 Flexible levels of nesting

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

for k, v in nd.items_flat():
    print "%-50s==%20r" % (k,v)
```

Gives:

```
('mouse', 'chr3')                                   ==       '3rd longest'
('mouse', 'chromosomes', 'Y', 'Male')               ==                True
('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
- a scalar with useful default values such as `int` or `str`.

```
#   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': 'bbbbbccccccc'}, '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:

```
from nested_dict import nested_dict

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
```

CHAPTER

THREE

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 `items_flat()`, `keys_flat()`, and `values_flat()`. (`iteritems_flat()`, `iterkeys_flat()`, and `itervalues_flat()` are python2.7 style synonyms.) 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.items_flat():
    print ("%-30s == %20r" % (keys_as_tuple, value))
#   ('mouse', 'chr3')              ==        '3rd longest'
#   ('mouse', 'chromosomes')       ==          'completed'
#   ('mouse', 'chr2')              ==        '2nd longest'
#   ('mouse', 'chr1', '+')         ==                 311
```

CHAPTER

FOUR

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 Class documentation

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`()

python 2.7 style synonym for `items_flat()`

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)
]
```

iterkeys_flat()
python 2.7 style synonym for `keys_flat()`

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')]
```

itervalues_flat()
python 2.7 style synonym for `values_flat()`

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]
```

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}}
{
    "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

n

Symbols

I

K

N

T

V