dotty$_{dict}$Documentation

Release 0.1.8

Paweł Zadrożny

May 06, 2017

Contents

Contents:

CHAPTER 1

dotty_dict

Dotty dictionary with support_for['dot.notation.keys'].

Dotty dict-like object allow to access deeply nested keys using dot notation. Create Dotty from dict or other dict-like object to use magic of Dotty.

Ultimate goal is to match all Python dictionary method to work with deeply nested **Dotty** keys.

- Free software: MIT license
- Documentation: https://dotty-dict.readthedocs.io.

Features

- Access and assign deeply nested dictionary key using dot notation
- Return None if key doesn't exist instead of KeyError exception
- Get deeply nested value or default value with .get() method

TODO

- Escape dot char for dictionary keys with dot: dotty_dict['strange.key']
- Check if key exists in deeply nested dictionary: 'deeply.nested' in dotty_dict
- Delete deeply nested keys: del dotty_dict['deeply.nested']

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install dotty_dict, run this command in your terminal:

```
$ pip install dotty_dict
```

This is the preferred method to install dotty_dict, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for dotty_dict can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/pawelzny/dotty_dict
```

Or download the tarball:

```
$ curl  -OL https://github.com/pawelzny/dotty_dict/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

Dotty dictionary follows dict interfaces and all rules applied to dictionaries. Read more in Python documentation

Dotty can be created by placing a standard dictionary with comma-separated list of key: value pairs within **Dotty** constructor, for example:

```
>>> Dotty()
{}
>>> Dotty({'foo': 4098, 'bar': 4127})
{'foo': 4098, 'bar': 4127}
>>> Dotty(foo=4098, bar=4127)
{'foo': 4098, 'bar': 4127}
>>> Dotty([('foo', 4098), ('bar', 4127)])
{'foo': 4098, 'bar': 4127}
>>> Dotty({'foo.bar.baz': 4098, 'foo.bar.fizz': 4127})
{'foo': {'bar': {'baz': 4098, 'fizz': 4127}}}
>>> Dotty([('foo.bar.baz', 4098), ('foo.bar.fizz', 4127)])
{'foo': {'bar': {'baz': 4098, 'fizz': 4127}}}
```

Overview

To use **Dotty** in a project:

```
>>> from dotty_dict import Dotty

>>> dotty = Dotty({'first': {'second': {'deep': 'i am here!'}}})
>>> dotty['first.second.deeper.better.faster.stronger'] = 'ohh!'
>>> dotty
{'first': {'second': {'deep': 'i am here!', 'deeper': {'better': {'faster': {'stronger
↪': 'ohh!'}}}}}}
```

Operations

len(d)

Return the number of items in the **Dotty** dictionary *d*.

d[*'key.key.key'* **]**

Return the item of d with dot notation key. Returns None if key is not in the map.

```
>>> from dotty_dict import Dotty

>>> dotty = Dotty({'foo.bar': 'baz'})
>>> dotty['foo.bar'] += ' & fizz'
>>> dotty
{'foo': {'bar': 'baz & fizz'}}
>>> dotty['foo.bar']
'baz & fizz'
```

d[*'key.key.key'* **] = value**

Set deeply nested *key.key.key* to value

```
>>> from dotty_dict import Dotty
>>> dotty = Dotty()
>>> dotty['foo.bar'] = 'baz'
>>> dotty
{'foo': {'bar': 'baz'}}
```

.get(*'key.key.key'* **[,** *default* **])**

If deeply nested key is in dictionary return it's value, but if key doesn't exist or it's value is None then return optional default value, default defaults to None.

```
>>> from dotty_dict import Dotty
>>> dotty = Dotty()
>>> dotty['foo.bar.baz'] = 'fizz'
>>> value = dotty.get('foo.bar.baz', 'buzz')
>>> value
'fizz'
>>> value = dotty.get('fizz.buzz', 'foo')
>>> value
'foo'
```

.clear()

Removes all items from **Dotty** dict

.copy()

Return a shallow copy of the dictionary.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/pawelzny/dotty_dict/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

dotty_dict could always use more documentation, whether as part of the official dotty_dict docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pawelzny/dotty_dict/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *dotty_dict* for local development.

1. Fork the *dotty_dict* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dotty_dict.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dotty_dict
$ cd dotty_dict/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 dotty_dict tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/pawelzny/dotty_dict/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_dotty_dict
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search