Ordered Set Documentation

Release 2.0

Simon Percivall

February 02, 2016

Contents

CHAPTER 1

Contents:

An Ordered Set implementation in Cython. Based on Raymond Hettinger's OrderedSet recipe.

Example:

```
>>> from orderedset import OrderedSet
>>> oset = OrderedSet([1, 2, 3])
>>> oset
OrderedSet([1, 2, 3])
>>> oset | [5, 4, 3, 2, 1]
OrderedSet([1, 2, 3, 5, 4])
```

- Free software: BSD license
- Documentation: http://orderedset.rtfd.org.

1.1 Features

- Works like a regular set, but remembers insertion order;
- Is approximately 5 times faster than the pure Python implementation overall (and 5 times slower than *set*);
- Compatible with Python 2.6 through 3.4.;
- Supports the full set interface;
- Supports some list methods, like *index* and *__getitem__*.
- Supports set methods against iterables.

1.1.1 Installation

At the command line:

```
$ pip install orderedset
```

1.1.2 Usage

To use OrderedSet in a project:

```
from orderedset import OrderedSet

oset = OrderedSet([1, 2, 3])
oset2 = OrderedSet([3, 2, 1])
oset3 = OrderedSet([1, 2, 3, 4])

oset == oset2  # False
oset <= oset2  # True, same as issubset
```

OrderedSet normally compares like a set, but can be made to make ordered sub-/superset comparisons:

```
oset.isorderedsubset(oset2)  # False
oset.isorderedsubset(oset3)  # True
```

OrderedSets work with other sets, and with lists:

```
from orderedset import OrderedSet

oset = OrderedSet([1, 2, 3])
lst = [1, 2, 3]
tes = {1, 2, 3}

oset <= lst  # True
oset <= tes  # True

oset | lst  # OrderedSet([1, 2, 3])
oset | tes  # OrderedSet([1, 2, 3])
```

1.1.3 orderedset package

OrderedSet

class `orderedset.`**`OrderedSet`**

An `OrderedSet` object is an ordered collection of distinct hashable objects.

It works like the `set` type, but remembers insertion order.

It also supports `__getitem__()` and `index()`, like the `list` type.

`__getitem__`(*index*)

Return the *elem* at *index*. Raises `IndexError` if *index* is out of range.

`add`(*self*, *elem*)

Add element *elem* to the set.

`clear`(*self*)

Remove all elements from the *set*.

`copy`(*self*)

Return type *OrderedSet*

Returns a new `OrderedSet` with a shallow copy of self.

`difference`(*self*, *other*)

`OrderedSet - other`

Return type *OrderedSet*

Returns a new `OrderedSet` with elements in the set that are not in the others.

difference_update(*self*, *other*)
`OrderedSet -= other`

Update the `OrderedSet`, removing elements found in others.

discard(*self*, *elem*)
Remove element *elem* from the `OrderedSet` if it is present.

index(*self*, *elem*)
Return the index of *elem*. Rases `ValueError` if not in the OrderedSet.

intersection(*self*, *other*)
`OrderedSet & other`

Return type *OrderedSet*

Returns a new `OrderedSet` with elements common to the set and all others.

intersection_update(*self*, *other*)
`OrderedSet &= other`

Update the `OrderedSet`, keeping only elements found in it and all others.

isdisjoint(*self*, *other*)
Return True if the set has no elements in common with other. Sets are disjoint if and only if their intersection is the empty set.

Return type bool

isorderedsubset(*self*, *other*)

isorderedsuperset(*self*, *other*)

issubset(*self*, *other*)
`OrderedSet <= other`

Return type bool

Test whether the `OrderedSet` is a proper subset of other, that is, `OrderedSet <= other and OrderedSet != other`.

issuperset(*self*, *other*)
`OrderedSet >= other`

Return type bool

Test whether every element in other is in the set.

pop(*self*, *last=True*)
Remove last element. Raises `KeyError` if the `OrderedSet` is empty.

remove(*self*, *elem*)
Remove element *elem* from the `set`. Raises `KeyError` if *elem* is not contained in the set.

symmetric_difference(*self*, *other*)
`OrderedSet ^ other`

Return type *OrderedSet*

Returns a new `OrderedSet` with elements in either the set or other but not both.

symmetric_difference_update(*self*, *other*)
`OrderedSet ^= other`

Update the `OrderedSet`, keeping only elements found in either set, but not in both.

union (*self*, *other*)
`OrderedSet | other`

Return type *OrderedSet*

Returns a new `OrderedSet` with elements from the set and all others.

update (*self*, *other*)
`OrderedSet |= other`

Update the `OrderedSet`, adding elements from all others.

1.1.4 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/simonpercivall/orderedset.

If you are reporting a bug, please include:

- 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" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Ordered Set could always use more documentation, whether as part of the official Ordered Set 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 issueat https://github.com//orderedset/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 *orderedset* for local development.

1. Check out the repository.

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

```
$ flake8 orderedset tests
$ python setup.py test
$ tox
```

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

6. Commit and send the patch or create a pull request.

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 2.7, and 3.3.

1.1.5 Credits

Author

- Simon Percivall <percivall@gmail.com>

Contributors

- Derived from an implementation by Raymond Hettinger
- Set operations from `abcoll.py` in Python core.

1.1.6 History

2.0 - 2016-02-02

- breaking change: All comparisons, other than *eq*, against other ordered sets are now performed unordered; i.e., they are treated as regular sets.
- *isorderedsubset* and *isorderedsuperset* have been added to perform ordered comparisons against other sequences. Using these methods with unordered collections wield yield arbitrary (and depending on Python implementation, unstable) results.

1.2 - 2015-09-29

- bugfix: Set operations only worked with iterables if the OrderedSet was on the left-hand side. They now work both ways.
- bugfix: The order of an intersection was the right-hand side's order. It is now fixed to be the left-hand side's order.

1.1.2 - 2014-10-02

- Make comparisons work with sets and lists, and not crash when compared with None.

1.1.1 - 2014-08-24

- Add pickle/copy support to OrderedSet

1.1 - 2014-06-04

- Make OrderedSets handle slicing in __getitem__().

1.0.2 - 2014-05-14

- Add proper attribution and licenses.

1.0.1 - 2014-05-13

- Don't require Cython to build an sdist.

1.0 - 2014-05-11

- First implementation.

CHAPTER 2

Indices and tables

- genindex

Symbols

A

C

D

I

O

P

R

S

U