kenjutsu Documentation

Release 0.5.1+0.ge4ba76e.dirty

John Kirkham

Mar 09, 2017

Contents

Contents:

CHAPTER 1

kenjutsu

Python utility functions for slices.

- Free software: BSD 3-Clause
- Documentation: https://kenjutsu.readthedocs.io.

Features

- TODO

Credits

This package was created with Cookiecutter and the nanshe-org/nanshe-cookiecutter project template.

CHAPTER 2

Installation

Stable release

To install kenjutsu, run this command in your terminal:

```
$ pip install kenjutsu
```

This is the preferred method to install kenjutsu, 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 kenjutsu can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/jakirkham/kenjutsu
```

Or download the tarball:

```
$ curl  -OL https://github.com/jakirkham/kenjutsu/tarball/master
```

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

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

CHAPTER 3

Usage

To use kenjutsu in a project:

```
import kenjutsu
```

CHAPTER 4

API

kenjutsu package

Submodules

kenjutsu.blocks module

kenjutsu.blocks.**num_blocks**(*space_shape*, *block_shape*)

Computes the number of blocks.

Takes an array with `space_shape` and `block_shape` for every dimension. From this, it can compute slicings to use for cutting each block out from the original array, HDF5 dataset, or other.

Parameters

- **space_shape** (*tuple*) – Shape of array to slice
- **block_shape** (*tuple*) – Size of each block to take

Returns Number of blocks per dimension

Return type tuple

Examples

```
>>> num_blocks(
...     (2, 3,), (2, 1,)
... )
(1, 3)
```

kenjutsu.blocks.**split_blocks**(*space_shape*, *block_shape*, *block_halo=None*, *index=None*)

Return a list of slicings to cut each block out of an array or other.

Takes an array with `space_shape` and `block_shape` for every dimension and a `block_halo` to extend each block on each side. From this, it can compute slicings to use for cutting each block out from the original array, HDF5 dataset or other.

Note: Blocks on the boundary that cannot extend the full range will be truncated to the largest block that will fit. This will raise a warning, which can be converted to an exception, if needed.

Parameters

- **space_shape** (*tuple*) – Shape of array to slice
- **block_shape** (*tuple*) – Size of each block to take
- **block_halo** (*tuple*) – Halo to tack on to each block
- **index** (*bool*) – Whether to provide an index for each block

Returns Provides tuples of slices for retrieving blocks.

Return type collections.Sequence of tuples of slices

Examples

```
>>> split_blocks(
...     (2, 3,), (1, 1,), (1, 1,), True
... )
([(0, 0),
  (0, 1),
  (0, 2),
  (1, 0),
  (1, 1),
  (1, 2)],

 [(slice(0, 1, 1), slice(0, 1, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
  (slice(0, 1, 1), slice(2, 3, 1)),
  (slice(1, 2, 1), slice(0, 1, 1)),
  (slice(1, 2, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(2, 3, 1))],

 [(slice(0, 2, 1), slice(0, 2, 1)),
  (slice(0, 2, 1), slice(0, 3, 1)),
  (slice(0, 2, 1), slice(1, 3, 1)),
  (slice(0, 2, 1), slice(0, 2, 1)),
  (slice(0, 2, 1), slice(0, 3, 1)),
  (slice(0, 2, 1), slice(1, 3, 1))],

 [(slice(0, 1, 1), slice(0, 1, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(0, 1, 1)),
  (slice(1, 2, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(1, 2, 1))])
```

kenjutsu.core module

The module `core` provides support for working with `slices`.

Overview

The module `core` provides several functions that are useful for working with a Python `slice` or `tuple` of `slices`. This is of particular value when working with NumPy.

API

kenjutsu.format module

`kenjutsu.format.`**`index_to_slice`**(*index*)

Convert an index to a slice.

Note: A single index behaves differently from a length 1 `slice`. When applying the former one reduces that dimension; whereas, applying the latter results in a singleton dimension being retained. Also if an index is out of bounds, one gets an `IndexError`. However, with an out of bounds length 1 `slice`, one simply doesn't get the requested range.

Parameters **`index`** (*int*) – an index to convert to a slice

Returns a slice corresponding to the index

Return type (slice)

Examples

```
>>> index_to_slice(1)
slice(1, 2, 1)
```

```
>>> index_to_slice(-1)
slice(-1, -2, -1)
```

`kenjutsu.format.`**`reformat_slice`**(*a_slice*, *a_length=None*)

Takes a slice and reformats it to fill in as many undefined values as possible.

Parameters

- **`a_slice`** (*slice*) – a slice to reformat.
- **`a_length`** (*int*) – a length to fill for stopping if not provided.

Returns

a new slice with as many values filled in as possible.

Return type (slice)

Examples

```
>>> reformat_slice(slice(2, -1, None))
slice(2, -1, 1)
```

```
>>> reformat_slice(slice(2, -1, None), 10)
slice(2, 9, 1)
```

kenjutsu.format.**reformat_slices**(*slices*, *lengths=None*)

Takes a tuple of slices and reformats them to fill in as many undefined values as possible.

Parameters

- **slices** (*tuple(slice)*) – a tuple of slices to reformat.
- **lengths** (*tuple(int)*) – a tuple of lengths to fill.

Returns

a tuple of slices with all default values filled if possible.

Return type (slice)

Examples

```
>>> reformat_slices(
...     (
...         slice(None),
...         slice(3, None),
...         slice(None, 5),
...         slice(None, None, 2)
...     ),
...     (10, 13, 15, 20)
... )
(slice(0, 10, 1), slice(3, 13, 1), slice(0, 5, 1), slice(0, 20, 2))
```

kenjutsu.format.**split_indices**(*slices*)

Splits slices with multiple indices into multiple splices.

Support of slices with a sequence of indices is varied. Some libraries like NumPy support them without issues. Other libraries like h5py support them as long as they are in sequential order. In still other libraries support is non-existent. However, in all those cases normally a single index is permissible. This converts slices with multiple indices into a list of slices with a single index each. While this still leaves it up to the user to iterate over these and combine the results in some sensible way, it is better than just getting a failure and should extend well to a variety of cases.

Parameters **slices** (*tuple(slice)*) – a tuple of slices to split

Returns a list of a tuple of slices

Return type (list(tuple(slice)))

Examples

```
>>> split_indices(
...     (
...         3,
```

```
...         Ellipsis,
...         [0, 1, 2],
...         slice(2, 5),
...         slice(4, 6, 2)
...     )
... )
[(3, Ellipsis, slice(0, 1, 1), slice(2, 5, 1), slice(4, 6, 2)),
 (3, Ellipsis, slice(1, 2, 1), slice(2, 5, 1), slice(4, 6, 2)),
 (3, Ellipsis, slice(2, 3, 1), slice(2, 5, 1), slice(4, 6, 2))]
```

kenjutsu.kenjutsu module

Warning: The module `kenjutsu.kenjutsu` is deprecated. Please use `kenjutsu.core` instead.

`kenjutsu.kenjutsu.`**`len_slice`**(*a_slice*, *a_length=None*)

Determines how many elements a slice will contain.

Warning: This function is deprecated. Please use `kenjutsu.core.len_slice` instead.

Raises

- `UnknownSliceLengthException` – Will raise an exception if
- a_slice.stop and a_length is None.

Parameters

- **`a_slice`** (*slice*) – a slice to reformat.
- **`a_length`** (*int*) – a length to fill for stopping if not provided.

Returns

a new slice with as many values filled in as possible.

Return type (slice)

Examples

```
>>> len_slice(slice(2, None), 10)
8
```

```
>>> len_slice(slice(2, 6))
4
```

`kenjutsu.kenjutsu.`**`len_slices`**(*slices*, *lengths=None*)

Takes a tuple of slices and reformats them to fill in as many undefined values as possible.

Warning: This function is deprecated. Please use `kenjutsu.core.len_slices` instead.

Parameters

- **slices** (*tuple(slice)*) – a tuple of slices to reformat.
- **lengths** (*tuple(int)*) – a tuple of lengths to fill.

Returns

a tuple of slices with all default values filled if possible.

Return type (slice)

Examples

```
>>> len_slices(
...     (
...         slice(None),
...         slice(3, None),
...         slice(None, 5),
...         slice(None, None, 2)
...     ),
...     (10, 13, 15, 20)
... )
(10, 10, 5, 10)
```

kenjutsu.kenjutsu.**reformat_slice**(*a_slice*, *a_length=None*)

Takes a slice and reformats it to fill in as many undefined values as possible.

Warning: This function is deprecated. Please use `kenjutsu.core.reformat_slice` instead.

Parameters

- **a_slice** (*slice*) – a slice to reformat.
- **a_length** (*int*) – a length to fill for stopping if not provided.

Returns

a new slice with as many values filled in as possible.

Return type (slice)

Examples

```
>>> reformat_slice(slice(2, -1, None))
slice(2, -1, 1)
```

```
>>> reformat_slice(slice(2, -1, None), 10)
slice(2, 9, 1)
```

kenjutsu.kenjutsu.**reformat_slices**(*slices*, *lengths=None*)

Takes a tuple of slices and reformats them to fill in as many undefined values as possible.

Warning: This function is deprecated. Please use `kenjutsu.core.reformat_slices` instead.

Parameters

- **slices** (*tuple(slice)*) – a tuple of slices to reformat.
- **lengths** (*tuple(int)*) – a tuple of lengths to fill.

Returns

a tuple of slices with all default values filled if possible.

Return type (slice)

Examples

```
>>> reformat_slices(
...     (
...         slice(None),
...         slice(3, None),
...         slice(None, 5),
...         slice(None, None, 2)
...     ),
...     (10, 13, 15, 20)
... )
(slice(0, 10, 1), slice(3, 13, 1), slice(0, 5, 1), slice(0, 20, 2))
```

kenjutsu.kenjutsu.**split_blocks**(*space_shape*, *block_shape*, *block_halo=None*)

Return a list of slicings to cut each block out of an array or other.

Takes an array with `space_shape` and `block_shape` for every dimension and a `block_halo` to extend each block on each side. From this, it can compute slicings to use for cutting each block out from the original array, HDF5 dataset or other.

Warning: This function is deprecated. Please use `kenjutsu.core.split_blocks` instead.

Note: Blocks on the boundary that cannot extend the full range will be truncated to the largest block that will fit. This will raise a warning, which can be converted to an exception, if needed.

Parameters

- **space_shape** (*tuple*) – Shape of array to slice
- **block_shape** (*tuple*) – Size of each block to take
- **block_halo** (*tuple*) – Halo to tack on to each block

Returns Provides tuples of slices for retrieving blocks.

Return type collections.Sequence of tuples of slices

Examples

```
>>> split_blocks(
...     (2, 3,), (1, 1,), (1, 1,)
... )
([(slice(0, 1, 1), slice(0, 1, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
```

```
  (slice(0, 1, 1), slice(2, 3, 1)),
  (slice(1, 2, 1), slice(0, 1, 1)),
  (slice(1, 2, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(2, 3, 1))],

 [(slice(0, 2, 1), slice(0, 2, 1)),
  (slice(0, 2, 1), slice(0, 3, 1)),
  (slice(0, 2, 1), slice(1, 3, 1)),
  (slice(0, 2, 1), slice(0, 2, 1)),
  (slice(0, 2, 1), slice(0, 3, 1)),
  (slice(0, 2, 1), slice(1, 3, 1))],

 [(slice(0, 1, 1), slice(0, 1, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
  (slice(0, 1, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(0, 1, 1)),
  (slice(1, 2, 1), slice(1, 2, 1)),
  (slice(1, 2, 1), slice(1, 2, 1))])
```

kenjutsu.measure module

exception kenjutsu.measure.**UnknownSliceLengthException**
Bases: exceptions.Exception

Raised if a slice does not have a known length.

kenjutsu.measure.**len_slice**(*a_slice*, *a_length=None*)
Determines how many elements a slice will contain.

Raises

- *UnknownSliceLengthException* – Will raise an exception if
- a_slice.stop and a_length is None.

Parameters

- **a_slice** (*slice*) – a slice to reformat.
- **a_length** (*int*) – a length to fill for stopping if not provided.

Returns

a new slice with as many values filled in as possible.

Return type (slice)

Examples

```
>>> len_slice(slice(2, None), 10)
8
```

```
>>> len_slice(slice(2, 6))
4
```

kenjutsu.measure.**len_slices**(*slices*, *lengths=None*)
Takes a tuple of slices and reformats them to fill in as many undefined values as possible.

Parameters

- **slices** (*tuple(slice)*) – a tuple of slices to reformat.
- **lengths** (*tuple(int)*) – a tuple of lengths to fill.

Returns

a tuple of slices with all default values filled if possible.

Return type (slice)

Examples

```
>>> len_slices(
...     (
...         slice(None),
...         slice(3, None),
...         slice(None, 5),
...         slice(None, None, 2)
...     ),
...     (10, 13, 15, 20)
... )
(10, 10, 5, 10)
```

kenjutsu.operators module

CHAPTER 5

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/jakirkham/kenjutsu/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

kenjutsu could always use more documentation, whether as part of the official kenjutsu 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/jakirkham/kenjutsu/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 *kenjutsu* for local development.

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

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

3. Install your local copy into an environment. Assuming you have conda installed, this is how you set up your fork for local development (on Windows drop *source*). Replace *"<some version>"* with the Python version used for testing.:

```
$ conda create -n kenjutsuenv python="<some version>"
$ source activate kenjutsuenv
$ 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:

```
$ flake8 kenjutsu tests
$ python setup.py test or py.test
```

   To get flake8, just conda install it into your environment.

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 2.7, 3.4, 3.5, and 3.6. Check https://travis-ci.org/jakirkham/kenjutsu/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_kenjutsu
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index