NumPy String-Indexed Documentation

Release latest

May 01, 2022

CONTENTS:

1 Basic functionality **3**

2 Installation **5**

3 Documentation **7**

3.1 User's guide 7
3.2 Array initialization and slicing 9
3.3 Array operations 12
3.4 Computing arrays 19
3.5 Formatting arrays for output 20

4 Discussion and support **23**

Index **25**

NumPy String-Indexed is a NumPy extension that allows arrays to be indexed using descriptive string labels, rather than conventional zero-indexing. When an `ndarray` (AKA a friendly matrix) instance is initialized, labels are assigned to each array index and each dimension, and they stick to the array after NumPy-style operations such as transposing, concatenating, and aggregating. This prevents Python programmers from having to keep track mentally of what each axis and each index represents, instead making each reference to the array in code naturally self-documenting.

NumPy String-Indexed is especially useful for applications like machine learning, scientific computing, and data science, where there is heavy use of multidimensional arrays.

The friendly matrix object is implemented as a lightweight wrapper around a NumPy `ndarray`. It's easy to add to a new or existing project to make it easier to maintain code, and has negligible memory and performance overhead compared to the size of array ($O(x + y + z)$ vs. $O(xyz)$).

CHAPTER

ONE

BASIC FUNCTIONALITY

It's recommended to import NumPy String-Indexed idiomatically as `fm`:

```
import friendly_matrix as fm
```

Labels are provided during object construction and can optionally be used in place of numerical indices for slicing and indexing.

The example below shows how to construct a friendly matrix containing an image with three color channels:

```
image = fm.ndarray(
        numpy_ndarray_image,  # np.ndarray with shape (3, 100, 100)
        dim_names=['color_channel', 'top_to_bottom', 'left_to_right'],
        color_channel=['R', 'G', 'B'])
```

The array can then be sliced like this:

```
# friendly matrix with shape (100, 100)
r_channel = image(color_channel='R')

# an integer
g_top_left_pixel_value = image('G', 0, 0)

# friendly matrix with shape (2, 100, 50)
br_channel_left_half = image(
        color_channel=('B', 'R'),
        left_to_right=range(image.dim_length('left_to_right') // 2))
```

CHAPTER

TWO

INSTALLATION

```
pip install numpy-string-indexed
```

NumPy String-Indexed is listed in PyPI and can be installed with `pip`.

Prerequisites: NumPy String-Indexed 0.0.3 requires Python 3 and a compatible installation of the NumPy Python package.

CHAPTER

THREE

DOCUMENTATION

3.1 User's guide

Below is an overview of the extensions NumPy String-Indexed offers. Functionality can be categorized into: array operations, computing arrays, and formatting arrays.

The examples below build on those from Index.

3.1.1 Array operations

Friendly matrix objects can be operated on just like NumPy `ndarray` s with minimal overhead. The package contains separate implementations of most of the relevant NumPy `ndarray` operations, taking advantage of labels. For example:

```
side_by_side = fm.concatenate((image1, image2), axis='left_to_right')
```

An optimized alternative is to perform label-less operations, by adding `"_A"` (for "array") to the operation name:

```
side_by_side_arr = fm.concatenate_A((image1, image2), axis='left_to_right')
```

If it becomes important to optimize within a particular scope, it's recommended to shed labels before operating:

```
for image in huge_list:
        image_processor(image.A)
```

3.1.2 Computing arrays

A friendly matrix is an ideal structure for storing and retrieving the results of computations over multiple variables. The `compute_ndarray()` function executes computations over all values of the input arrays and stores them in a new friendly matrix `ndarray` instance in a single step:

```
'''Collect samples from a variety of normal distributions'''

import numpy as np

n_samples_list = [1, 10, 100, 1000]
mean_list = list(range(-21, 21))
var_list = [1E1, 1E0, 1E-1, 1E-2, 1E-3]

results = fm.compute_ndarray(
```

(continues on next page)

(continued from previous page)

```
        ['# Samples', 'Mean', 'Variance']
        n_samples_list,
        mean_list,
        var_list,
        normal_sampling_function,
        dtype=np.float32)

# friendly matrices can be sliced using dicts
print(results({
        '# Samples': 100,
        'Mean': 0,
        'Variance': 1,
}))
```

3.1.3 Formatting arrays

The `formatted()` function displays a friendly matrix as a nested list. This is useful for displaying the labels and values of smaller arrays or slice results:

```
mean_0_results = results({
        '# Samples': (1, 1000),
        'Mean': 0,
        'Variance': (10, 1, 0.1),
})
formatted = fm.formatted(
        mean_0_results,
        formatter=lambda n: round(n, 1))

print(formatted)

'''
Example output:

# Samples = 1:
        Variance = 10:
                2.2
        Variance = 1:
                -0.9
        Variance = 0.1:
                0.1
# Samples = 1000:
        Variance = 10:
                -0.2
        Variance = 1:
                -0.0
        Variance = 0.1:
                0.0
'''
```

3.2 Array initialization and slicing

3.2.1 friendly_matrix.ndarray methods

class friendly_matrix.**ndarray**(*array*[, *dim_names=None*[, **args_dim_arrays*[, ***kwargs_dim_arrays*]]])

A structure for matrix-like data, which stores the data as a classic NumPy `ndarray`, and provides the option to reference by human-readable values.

This class, plus the other functions exposed in the friendly_matrix package, are designed as substitutes for the NumPy `ndarray`, with comparable performance benchmarks and familiar, NumPy-style usage patterns.

Labels do not need to be specified for every dimension and index. There are four ways to initialize a `friendly_matrix.ndarray` instance using the constructor, all of which involve assigning new labels to an existing NumPy `ndarray`. The other main way to create a new `friendly_matrix.ndarray` is by calling *friendly_matrix.compute_ndarray()*. The four ways are demonstrated below. In the examples, we assume the array `array` consists of two dimensions, for *size* and *n_passengers*. Dimension *size* has length 3, for *small*, *medium*, and *large*, and dimension *n_passengers* goes from 0 to 4.

1. Casting:

```
rockets = friendly_matrix.ndarray(array)
```

Note: This creates an unlabeled `friendly_matrix.ndarray` instance.

2. Dimension arrays as arguments:

```
rockets = friendly_matrix.ndarray(
        array,
        ['size', 'n_passengers'],
        ['small', 'medium', 'large'])
```

3. Dimension arrays as dict:

```
dim_arrays = {
        'size': ['small', 'medium', 'large']
}
rockets = friendly_matrix.ndarray(
        array,
        ['size', 'n_passengers'],
        dim_arrays)
```

4. Dimension arrays as keyword arguments:

```
rockets = friendly_matrix.ndarray(
        array,
        ['size', 'n_passengers'],
        size=['small', 'medium', 'large'])
```

Parameters

- **array** – NumPy array to wrap
- **dim_names** – label for each dimension
- ***args_dim_arrays** – index labels for each dimension, or single dict mapping each dimension label to its corresponding index labels

- ****kwargs_dim_arrays** – index labels for each dimension (only if specified dimensions are argument- and keyword-friendly)

property ndim

Type int

Number of dimensions

property shape

Type tuple

Length of each dimension

property size

Type int

Total number of elements in the array

property dtype

Type type

Data type of the array

property itemsize

Type int

Length of one element of the array in bytes

dim_length(*dim*) → int

Parameters **dim** – dimension label or index

Returns length of that dimension

take(**args*, ***kwargs*) → *friendly_matrix.ndarray*

Takes a slice of the array according to the specified labels.

Parameters

- ***args** – index labels to select for each dimension, or single dict mapping each dimension label to its corresponding index labels
- ****kwargs** – index labels for each dimension (only if specified dimensions are argument- and keyword-friendly)

If no labels are specified for a dimension, the entire dimension is selected. If a single label not wrapped in a list is specified for a dimension, that dimension is dropped in the result. If all labels specified are single labels, the result is equivalent to calling `get()`.

A take operation can also be performed by calling a `friendly_matrix.ndarray` instance directly. It's recommended to use this shorthand for style.

The three ways of using `take()` are demonstrated below. In the examples, we assume the array `rockets` consists of two dimensions, *size* and *n_passengers*. Dimension *size* has indices named *small*, *medium*, and *large*, and dimension *n_passengers* goes from 0 to 4.

1. Dimension arrays as arguments:

```
rockets('large', [2, 3])
```

The value `None` can be passed in as a shorthand for selecting all indices in a dimension.

2. Dimension arrays as dict:

```
rockets({
        'size': 'large',
        'n_passengers': [2, 3]
})
```

3. Dimension arrays as keyword arguments:

```
rockets(size='large', n_passengers=[2, 3])
```

Note: In the above examples, the shape of the result is `(2,)`, because passing in the single value `'large'` for the first dimension causes the dimension to be dropped from the result. Passing in `['large']` instead would result in a shape of `(1, 2)`.

Returns A new `friendly_matrix.ndarray` instance containing the filtered array

take_A(**args*, ***kwargs*) → numpy.ndarray

Same as *friendly_matrix.ndarray.take()*, except returns only the NumPy array.

get(**args*, ***kwargs*) → object

Gets the single element by its labels.

Parameters

- ***args** – index labels to select for each dimension, or single dict mapping each dimension label to its corresponding index labels
- ****kwargs** – index labels for each dimension (only if specified dimensions are argument- and keyword-friendly)

A get operation can also be performed by calling a `friendly_matrix.ndarray` directly.

Returns The element

set(*val*, **args*, ***kwargs*) → None

Sets the single element by its labels.

Parameters

- **val** – the updated value
- ***args** – index labels to select for each dimension, or single dict mapping each dimension label to its corresponding index labels
- ****kwargs** – index labels for each dimension (only if specified dimensions are argument- and keyword-friendly)

copy() → *friendly_matrix.ndarray*

Creates a deep copy of the current object.

3.2.2 Module functions

friendly_matrix.**take**(*friendly*, **args*, ***kwargs*) → *friendly_matrix.ndarray*

Equivalent to `friendly.take(*args, **kwargs)`.

See *friendly_matrix.ndarray.take()*.

friendly_matrix.**take_A**(*friendly*, **args*, **kwargs*) → numpy.ndarray

Equivalent to `friendly.take_A(*args, **kwargs)`.

See *friendly_matrix.ndarray.take_A()*.

friendly_matrix.**get**(*friendly*, **args*, **kwargs*) → *friendly_matrix.ndarray*

Equivalent to `friendly.get(*args, **kwargs)`.

See *friendly_matrix.ndarray.get()*.

friendly_matrix.**set**(*friendly*, **args*, **kwargs*) → *friendly_matrix.ndarray*

Equivalent to `friendly.set(*args, **kwargs)`.

See *friendly_matrix.ndarray.set()*.

friendly_matrix.**copy**(*friendly*) → *friendly_matrix.ndarray*

Equivalent to `friendly.copy()`.

See *friendly_matrix.ndarray.copy()*.

3.3 Array operations

3.3.1 `friendly_matrix.ndarray` methods

class friendly_matrix.**ndarray**

moveaxis(*dim*, *new_dim*) → *friendly_matrix.ndarray*

Performs a NumPy-style moveaxis operation on the `friendly_matrix.ndarray` instance. The ordering of dimensions is changed by moving one dimension to the position of another dimension.

Parameters

- **dim** – the dimension to move
- **new_dim** – the dimension whose place *dim* will take

Returns The new `friendly_matrix.ndarray` instance

moveaxis_A(*dim*, *new_dim*) → numpy.ndarray

Same as *friendly_matrix.ndarray.moveaxis()*, except returns only the NumPy array.

swapaxes(*dim1*, *dim2*) → *friendly_matrix.ndarray*

Performs a NumPy-style swapaxes operation on the `friendly_matrix.ndarray` instance. The ordering of dimensions is changed by swapping the positions of two dimensions.

Parameters

- **dim1** – dimension
- **dim2** – dimension

Returns The new `friendly_matrix.ndarray` instance

swapaxes_A(*dim1*, *dim2*) → numpy.ndarray

Same as `friendly_matrix.ndarray.swapaxis()`, except returns only the NumPy array.

transpose() → *friendly_matrix.ndarray*

Performs a NumPy-style transpose operation on the `friendly_matrix.ndarray` instance. The ordering of the first two dimensions are swapped.

Returns The new `friendly_matrix.ndarray` instance

transpose_A() → numpy.ndarray

Same as *friendly_matrix.ndarray.transpose()*, except returns only the NumPy array.

friendly_matrix.ndarray.T -> friendly_matrix.ndarray

Same as *friendly_matrix.ndarray.transpose()*.

friendly_matrix.ndarray.T_A -> numpy.ndarray

Same as *friendly_matrix.ndarray.transpose_A()*.

mean(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style mean computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the mean(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

mean_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.mean()*, except returns only the NumPy array.

std(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style std computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the standard deviation(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

std_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.std()*, except returns only the NumPy array.

var(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style var computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the variance(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

var_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.var()*, except returns only the NumPy array.

sum(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style sum computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the sum(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

sum_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.sum()*, except returns only the NumPy array.

prod(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style prod computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the product(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

prod_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.prod()*, except returns only the NumPy array.

min(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style min computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating minimum value(s) along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

min_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.min()*, except returns only the NumPy array.

argmin(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style argmin computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the index or indices of the minimum value along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

argmin_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.argmin()*, except returns only the NumPy array.

all(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style all computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating whether all the values along that dimension are truthy.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

all_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.all()*, except returns only the NumPy array.

any(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style any computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the any of the values along that dimension are truthy.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance

any_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.any()*, except returns only the NumPy array.

cumsum(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style cumsum computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the cumulative sum along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance (with the same shape as the original)

cumsum_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.cumsum()*, except returns only the NumPy array.

cumprod(*axis*) → *friendly_matrix.ndarray*

Performs a NumPy-style cumprod computation on the `friendly_matrix.ndarray` instance. Aggregates over a given dimension by calculating the cumulative product along that dimension.

Parameters **axis** – dimension

Returns The new `friendly_matrix.ndarray` instance (with the same shape as the original)

cumprod_A(*axis*) → numpy.ndarray

Same as *friendly_matrix.ndarray.cumprod()*, except returns only the NumPy array.

squeeze() → *friendly_matrix.ndarray*

Removes any length 1 dimensions in the `friendly_matrix.ndarray` instance by aggregating over them.

Returns The new `friendly_matrix.ndarray` instance

squeeze_A() → numpy.ndarray

Same as *friendly_matrix.ndarray.squeeze()*, except returns only the NumPy array.

3.3.2 Module functions

friendly_matrix.**concatenate**(*friendlies*, *axis=0*) → *friendly_matrix.ndarray*

Performs a NumPy-style concatenate operation on the `friendly_matrix.ndarray` instance. Concatenates the provided `friendly_matrix.ndarray` instances along the specified dimension.

Parameters

- **friendlies** – `friendly_matrix.ndarray` instances
- **axis** – the dimension along which to concatenate *friendlies*

Returns The new `friendly_matrix.ndarray` instance

friendly_matrix.**concatenate_A**(*friendlies*, *axis=0) numpy.ndarray*)

Same as *friendly_matrix.concatenate()*, except returns only the NumPy array.

friendly_matrix.**stack**(*friendlies*, *axis_name*, *axis_array*, *axis=0*) → *friendly_matrix.ndarray*

Performs a NumPy-style stack operation on the `friendly_matrix.ndarray` instances. Stacks the provided `friendly_matrix.ndarray` instances along a new dimension.

Parameters

- **friendlies** – `friendly_matrix.ndarray` instances
- **axis_name** – label for the new dimension
- **axis_array** – index labels for the new dimension
- **axis** – the dimension where the new dimension will be inserted

The `axis_array` argument should have the same length as `friendlies`.

friendly_matrix.**stack_A**(*friendlies*, *axis_name=None*, *axis_array=None*, *axis=None*) → *friendly_matrix.ndarray*

Same as *friendly_matrix.stack()*, except returns only the NumPy array.

friendly_matrix.**vstack**(*friendlies*) → *friendly_matrix.ndarray*

Equivalent to `concatenate(friendlies, axis=0)`. Can't be performed on one-dimensional arrays`.

See *friendly_matrix.concatenate()*.

friendly_matrix.**vstack_A**(*friendlies*) → numpy.ndarray

Same as *friendly_matrix.vstack()*, except returns only the NumPy array.

friendly_matrix.**hstack**(*friendlies*) → *friendly_matrix.ndarray*

Equivalent to `concatenate(friendlies, axis=1)`.

See *friendly_matrix.concatenate()*.

friendly_matrix.**hstack_A**(*friendlies*) → numpy.ndarray

Same as *friendly_matrix.hstack()*, except returns only the NumPy array.

friendly_matrix.**flip**(*friendly*, *axis=None*) → *friendly_matrix.ndarray*

Performs a NumPy-style flip operation on the `friendly_matrix.ndarray` instances. Reverses the order of elements along the provided dimension(s).

Parameters

- **friendly** – `friendly_matrix.ndarray` instance
- **axis** – dimension(s) along which to flip elements

The default value for `axis` of `None` results in a flip along all dimensions.

friendly_matrix.**flip_A**(*friendly*, *axis=None*) → numpy.ndarray

Same as *friendly_matrix.flip()*, except returns only the NumPy array.

friendly_matrix.**fliplr**(*friendly*) → *friendly_matrix.ndarray*

Equivalent to `friendly_matrix.flip(friendly, axis=0)`.

See *friendly_matrix.flip()*.

friendly_matrix.**fliplr_A**(*friendly*) → numpy.ndarray

Same as *friendly_matrix.fliplr()*, except returns only the NumPy array.

friendly_matrix.**flipud**(*friendly*) → *friendly_matrix.ndarray*

Equivalent to `friendly_matrix.flip(friendly, axis=1)`.

See *friendly_matrix.flip()*.

friendly_matrix.**flipud_A**(*friendly*) → numpy.ndarray

Same as *friendly_matrix.flipud()*, except returns only the NumPy array.

friendly_matrix.**moveaxis**(*friendly*, *dim*, *new_dim*) → *friendly_matrix.ndarray*

Equivalent to `friendly.moveaxis(axis)`.

See *friendly_matrix.ndarray.moveaxis()*.

friendly_matrix.**moveaxis_A**(*friendly*, *dim*, *new_dim*) → numpy.ndarray

Equivalent to `friendly.moveaxis_A(axis)`.

See *friendly_matrix.ndarray.moveaxis_A()*.

friendly_matrix.**swapaxes**(*friendly*, *dim1*, *dim2*) → *friendly_matrix.ndarray*

Equivalent to `friendly.swapaxes(axis)`.

See *friendly_matrix.ndarray.swapaxes()*.

friendly_matrix.**swapaxes_A**(*friendly*, *dim1*, *dim2*) → numpy.ndarray

Equivalent to `friendly.swapaxes_A(axis)`.

See *friendly_matrix.ndarray.swapaxes_A()*.

friendly_matrix.**transpose**(*friendly*) → *friendly_matrix.ndarray*

Equivalent to `friendly.transpose(axis)`.

See *friendly_matrix.ndarray.transpose()*.

friendly_matrix.**transpose_A**(*friendly*) → numpy.ndarray

Equivalent to `friendly.transpose_A(axis)`.

See *friendly_matrix.ndarray.transpose_A()*.

friendly_matrix.**mean**(*axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.mean(axis)`.

See *friendly_matrix.ndarray.mean()*.

friendly_matrix.**mean_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.mean_A(axis)`.

See *friendly_matrix.ndarray.mean_A()*.

friendly_matrix.**std**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.std(axis)`.

See *friendly_matrix.ndarray.std()*.

friendly_matrix.**std_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.std_A(axis)`.

See *friendly_matrix.ndarray.std_A()*.

friendly_matrix.**var**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.var(axis)`.

See *friendly_matrix.ndarray.var()*.

friendly_matrix.**var_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.var_A(axis)`.

See *friendly_matrix.ndarray.var_A()*.

friendly_matrix.**sum**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.sum(axis)`.

See *friendly_matrix.ndarray.sum()*.

friendly_matrix.**sum_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.sum_A(axis)`.

See *friendly_matrix.ndarray.sum_A()*.

friendly_matrix.**prod**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.prod(axis)`.

See *friendly_matrix.ndarray.prod()*.

friendly_matrix.**prod_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.prod_A(axis)`.

See *friendly_matrix.ndarray.prod_A()*.

friendly_matrix.**min**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.min(axis)`.

See *friendly_matrix.ndarray.min()*.

friendly_matrix.**min_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.min_A(axis)`.

See *friendly_matrix.ndarray.min_A()*.

friendly_matrix.**argmin**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.argmin(axis)`.

See *friendly_matrix.ndarray.argmin()*.

friendly_matrix.**argmin_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.argmin_A(axis)`.

See *friendly_matrix.ndarray.argmin_A()*.

friendly_matrix.**all**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.all(axis)`.

See *friendly_matrix.ndarray.all()*.

friendly_matrix.**all_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.all_A(axis)`.

See *friendly_matrix.ndarray.all_A()*.

friendly_matrix.**any**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.any(axis)`.

See *friendly_matrix.ndarray.any()*.

friendly_matrix.**any_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.any_A(axis)`.

See *friendly_matrix.ndarray.any_A()*.

friendly_matrix.**cumsum**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.cumsum(axis)`.

See *friendly_matrix.ndarray.cumsum()*.

friendly_matrix.**cumsum_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.cumsum_A(axis)`.

See *friendly_matrix.ndarray.cumsum_A()*.

friendly_matrix.**cumprod**(*friendly*, *axis*) → *friendly_matrix.ndarray*

Equivalent to `friendly.cumprod(axis)`.

See *friendly_matrix.ndarray.cumprod()*.

friendly_matrix.**cumprod_A**(*friendly*, *axis*) → numpy.ndarray

Equivalent to `friendly.cumprod_A(axis)`.

See *friendly_matrix.ndarray.cumprod_A()*.

friendly_matrix.**squeeze**(*friendly*) → *friendly_matrix.ndarray*

Equivalent to `friendly.squeeze()`.

See *friendly_matrix.ndarray.squeeze()*.

friendly_matrix.**squeeze_A**(*friendly*) → numpy.ndarray

Equivalent to `friendly.squeeze_A()`.

See *friendly_matrix.ndarray.squeeze_A()*.

3.4 Computing arrays

friendly_matrix.**compute_ndarray**(*dim_names*, **args*[, *dtype=numpy.float32*]) → *friendly_matrix.ndarray*

Generates a `friendly_matrix.ndarray` object by computing it and having it indexable the same way it's computed: using embedded loops over human-readable lists of values.

A friendly matrix is an ideal structure for storing and retrieving the results of computations over multiple variables. The *compute_ndarray()* function executes computations over all values of the input arrays and stores them in a new `friendly_matrix.ndarray` instance in a single step.

Parameters

- **dim_names** – the name of each dimension
- ***args** – iterables or callables specifying how to calculate results
- **dtype** – the data type of the computed results

The *args* arguments should contain iterables or callables, which constitute a complete set of instructions for computing the result. The first argument must be an iterable, and the last argument must be a callable. A group of one or more consecutive iterable arguments are iterated over via their Cartesian product. The next argument, which is a callable, takes the values from the current iteration as arguments to run some user-defined code, which can optionally yield precomputations for use in subsequent callables further up the stack.

Any intermediate callables should assemble precomputations in a dictionary, which is returned, in order to make them available to subsequent callables. For subsequent callables to access these precomputations, these callables should accept them as keyword arguments.

The final callable should return a value, which gets stored at a location in the `friendly_matrix.ndarray` specified by the values from the current iteration.

The `dim_names` argument should match the order of `args`.

The dim index labels in the result are set as the values of each iterable provided in `args`.

Below is a bare-bones example of how `compute_ndarray()` can be used:

```
iterable_a = [1, 2, 3]
iterable_b = [40, 50, 60]
iterable_c = [.7, .8, .9]

def callable_1(val_a, val_b):
        precomputations = {
```

(continues on next page)

(continued from previous page)

```
                'intermediate_sum': val_a + val_b
        }
        return precomputations

def callable_2(val_a, val_b, val_c, **precomputations):
        final_result = precomputations['intermediate_sum'] * val_c
        return final_result

result_friendly_matrix = friendly_matrix.compute_ndarray(
        ['a', 'b', 'c'],
        iterable_a,
        iterable_b,
        callable_1,
        iterable_c,
        callable_2)  # shape is (3, 3, 3)
```

friendly_matrix.**compute_ndarray_A**(*dim_names*, **args*[, *dtype=numpy.float32*]) → *friendly_matrix.ndarray*

Same as *compute_ndarray()*, except returns only the NumPy array.

3.5 Formatting arrays for output

3.5.1 friendly_matrix.ndarray methods

class friendly_matrix.**ndarray**

formatted([*topological_order=None*[, *formatter=None*[, *display_dim_names=True*]]]) → str

Formats the friendly_matrix.ndarray instance as a nested list. All elements in the array are listed linearly under their dim index labels. The order in which dimensions are traversed can be set, as well as whether dim names are displayed alongside dim index labels, and how elements should be formatted before being appended to the result.

This is useful for displaying the labels and values of smaller matrices or slice results.

Parameters

- **topological_order** – iterable representing the order in which dimensions should be traversed for output
- **formatter** – callable that formats an element for output
- **display_dim_names** – whether to display dim names alongside dim array labels

Example usage:

```
prices.formatted(topological_order=["Year", "Size"],
        formatter=price_formatter,
        display_dim_names=True)

'''
Example output:

Year = 2010:
        Size = small:
```

(continues on next page)

(continued from previous page)

```
                $1.99
        Size = large:
                $2.99
Year = 2020:
        Size = small:
                $2.99
        Size = large:
                $3.99
'''
```

3.5.2 Module functions

`friendly_matrix.`**`formatted`**(*friendly*[, *topological_order=None*[, *formatter=None*[, *display_dim_names=True*]]]) → str

Equivalent to *friendly.formatted(topological_order, formatter, display_dim_names)*.

`friendly_matrix.`**`from_formatted`**(*formatted_friendly*[, *dtype=numpy.str*]) → *friendly_matrix.ndarray*

Deserializes a string representation of a `friendly_matrix.ndarray` instance back into a new `friendly_matrix.ndarray` instance.

Parameters

- **`formatted_friendly`** – the formatted `friendly_matrix.ndarray` instance
- **`dtype`** – the data type of the result *friendly_matrix.ndarray*

Assumes a valid string is provided.

Returns The new `friendly_matrix.ndarray` instance

`friendly_matrix.`**`from_formatted_A`**(*formatted_friendly*[, *dtype=numpy.str*]) → *friendly_matrix.ndarray*

Same as *friendly_matrix.from_formatted()*, except returns only the NumPy array.

- genindex
- modindex
- search

CHAPTER

FOUR

DISCUSSION AND SUPPORT

NumPy String-Indexed is available under the MIT License.

INDEX

A

all() (*friendly_matrix.ndarray method*), 14
all_A() (*friendly_matrix.ndarray method*), 14
any() (*friendly_matrix.ndarray method*), 14
any_A() (*friendly_matrix.ndarray method*), 14
argmin() (*friendly_matrix.ndarray method*), 14
argmin_A() (*friendly_matrix.ndarray method*), 14

B

built-in function
    friendly_matrix.all(), 18
    friendly_matrix.all_A(), 18
    friendly_matrix.any(), 18
    friendly_matrix.any_A(), 18
    friendly_matrix.argmin(), 18
    friendly_matrix.argmin_A(), 18
    friendly_matrix.compute_ndarray(), 19
    friendly_matrix.compute_ndarray_A(), 20
    friendly_matrix.concatenate(), 15
    friendly_matrix.concatenate_A(), 15
    friendly_matrix.copy(), 12
    friendly_matrix.cumprod(), 18
    friendly_matrix.cumprod_A(), 18
    friendly_matrix.cumsum(), 18
    friendly_matrix.cumsum_A(), 18
    friendly_matrix.flip(), 16
    friendly_matrix.flip_A(), 16
    friendly_matrix.fliplr(), 16
    friendly_matrix.fliplr_A(), 16
    friendly_matrix.flipud(), 16
    friendly_matrix.flipud_A(), 16
    friendly_matrix.formatted(), 21
    friendly_matrix.from_formatted(), 21
    friendly_matrix.from_formatted_A(), 21
    friendly_matrix.get(), 12
    friendly_matrix.hstack(), 16
    friendly_matrix.hstack_A(), 16
    friendly_matrix.mean(), 17
    friendly_matrix.mean_A(), 17
    friendly_matrix.min(), 18
    friendly_matrix.min_A(), 18
    friendly_matrix.moveaxis(), 16
    friendly_matrix.moveaxis_A(), 16
    friendly_matrix.prod(), 17
    friendly_matrix.prod_A(), 17
    friendly_matrix.set(), 12
    friendly_matrix.squeeze(), 19
    friendly_matrix.squeeze_A(), 19
    friendly_matrix.stack(), 15
    friendly_matrix.stack_A(), 15
    friendly_matrix.std(), 17
    friendly_matrix.std_A(), 17
    friendly_matrix.sum(), 17
    friendly_matrix.sum_A(), 17
    friendly_matrix.swapaxes(), 16
    friendly_matrix.swapaxes_A(), 16
    friendly_matrix.take(), 12
    friendly_matrix.take_A(), 12
    friendly_matrix.transpose(), 17
    friendly_matrix.transpose_A(), 17
    friendly_matrix.var(), 17
    friendly_matrix.var_A(), 17
    friendly_matrix.vstack(), 15
    friendly_matrix.vstack_A(), 16

C

copy() (*friendly_matrix.ndarray method*), 11
cumprod() (*friendly_matrix.ndarray method*), 15
cumprod_A() (*friendly_matrix.ndarray method*), 15
cumsum() (*friendly_matrix.ndarray method*), 14
cumsum_A() (*friendly_matrix.ndarray method*), 14

D

dim_length() (*friendly_matrix.ndarray method*), 10
dtype (*friendly_matrix.ndarray property*), 10

F

formatted() (*friendly_matrix.ndarray method*), 20
friendly_matrix.all()
    built-in function, 18
friendly_matrix.all_A()
    built-in function, 18
friendly_matrix.any()
    built-in function, 18

friendly_matrix.any_A()
    built-in function, 18
friendly_matrix.argmin()
    built-in function, 18
friendly_matrix.argmin_A()
    built-in function, 18
friendly_matrix.compute_ndarray()
    built-in function, 19
friendly_matrix.compute_ndarray_A()
    built-in function, 20
friendly_matrix.concatenate()
    built-in function, 15
friendly_matrix.concatenate_A()
    built-in function, 15
friendly_matrix.copy()
    built-in function, 12
friendly_matrix.cumprod()
    built-in function, 18
friendly_matrix.cumprod_A()
    built-in function, 18
friendly_matrix.cumsum()
    built-in function, 18
friendly_matrix.cumsum_A()
    built-in function, 18
friendly_matrix.flip()
    built-in function, 16
friendly_matrix.flip_A()
    built-in function, 16
friendly_matrix.fliplr()
    built-in function, 16
friendly_matrix.fliplr_A()
    built-in function, 16
friendly_matrix.flipud()
    built-in function, 16
friendly_matrix.flipud_A()
    built-in function, 16
friendly_matrix.formatted()
    built-in function, 21
friendly_matrix.from_formatted()
    built-in function, 21
friendly_matrix.from_formatted_A()
    built-in function, 21
friendly_matrix.get()
    built-in function, 12
friendly_matrix.hstack()
    built-in function, 16
friendly_matrix.hstack_A()
    built-in function, 16
friendly_matrix.mean()
    built-in function, 17
friendly_matrix.mean_A()
    built-in function, 17
friendly_matrix.min()
    built-in function, 18
friendly_matrix.min_A()
    built-in function, 18
friendly_matrix.moveaxis()
    built-in function, 16
friendly_matrix.moveaxis_A()
    built-in function, 16
friendly_matrix.ndarray (*built-in class*), 9
friendly_matrix.prod()
    built-in function, 17
friendly_matrix.prod_A()
    built-in function, 17
friendly_matrix.set()
    built-in function, 12
friendly_matrix.squeeze()
    built-in function, 19
friendly_matrix.squeeze_A()
    built-in function, 19
friendly_matrix.stack()
    built-in function, 15
friendly_matrix.stack_A()
    built-in function, 15
friendly_matrix.std()
    built-in function, 17
friendly_matrix.std_A()
    built-in function, 17
friendly_matrix.sum()
    built-in function, 17
friendly_matrix.sum_A()
    built-in function, 17
friendly_matrix.swapaxes()
    built-in function, 16
friendly_matrix.swapaxes_A()
    built-in function, 16
friendly_matrix.take()
    built-in function, 12
friendly_matrix.take_A()
    built-in function, 12
friendly_matrix.transpose()
    built-in function, 17
friendly_matrix.transpose_A()
    built-in function, 17
friendly_matrix.var()
    built-in function, 17
friendly_matrix.var_A()
    built-in function, 17
friendly_matrix.vstack()
    built-in function, 15
friendly_matrix.vstack_A()
    built-in function, 16

G

get() (*friendly_matrix.ndarray method*), 11

I

itemsize (*friendly_matrix.ndarray property*), 10

M

mean() (*friendly_matrix.ndarray method*), 13
mean_A() (*friendly_matrix.ndarray method*), 13
min() (*friendly_matrix.ndarray method*), 14
min_A() (*friendly_matrix.ndarray method*), 14
moveaxis() (*friendly_matrix.ndarray method*), 12
moveaxis_A() (*friendly_matrix.ndarray method*), 12

N

ndim (*friendly_matrix.ndarray property*), 10

P

prod() (*friendly_matrix.ndarray method*), 13
prod_A() (*friendly_matrix.ndarray method*), 14

S

set() (*friendly_matrix.ndarray method*), 11
shape (*friendly_matrix.ndarray property*), 10
size (*friendly_matrix.ndarray property*), 10
squeeze() (*friendly_matrix.ndarray method*), 15
squeeze_A() (*friendly_matrix.ndarray method*), 15
std() (*friendly_matrix.ndarray method*), 13
std_A() (*friendly_matrix.ndarray method*), 13
sum() (*friendly_matrix.ndarray method*), 13
sum_A() (*friendly_matrix.ndarray method*), 13
swapaxes() (*friendly_matrix.ndarray method*), 12
swapaxes_A() (*friendly_matrix.ndarray method*), 12

T

take() (*friendly_matrix.ndarray method*), 10
take_A() (*friendly_matrix.ndarray method*), 11
transpose() (*friendly_matrix.ndarray method*), 13
transpose_A() (*friendly_matrix.ndarray method*), 13

V

var() (*friendly_matrix.ndarray method*), 13
var_A() (*friendly_matrix.ndarray method*), 13