Help on package numpy:

NAME

numpy

DESCRIPTION

NumPy

=====

Provides

1. An array object of arbitrary homogeneous items

2. Fast mathematical operations over arrays

3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation

----------------------------

Documentation is available in two forms: docstrings provided

with the code, and a loose standing reference guide, available from

`the NumPy homepage <https://www.scipy.org>`\_.

We recommend exploring the docstrings using

`IPython <https://ipython.org>`\_, an advanced Python shell with

TAB-completion and introspection capabilities. See below for further

instructions.

The docstring examples assume that `numpy` has been imported as `np`::

>>> import numpy as np

Code snippets are indicated by three greater-than signs::

>>> x = 42

>>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

>>> help(np.sort)

... # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help. This is

particularly true if you see the line "Help on ufunc object:" at the top

of the help() page. Ufuncs are implemented in C, not Python, for speed.

The native Python help() does not know how to view their help, but our

np.info() function does.

To search for documents containing a keyword, do::

>>> np.lookfor('keyword')

... # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts

of numpy are available under the ``doc`` sub-module::

>>> from numpy import doc

>>> help(doc)

... # doctest: +SKIP

Available subpackages

---------------------

doc

Topical documentation on broadcasting, indexing, etc.

lib

Basic functions used by several sub-packages.

random

Core Random Tools

linalg

Core Linear Algebra Tools

fft

Core FFT routines

polynomial

Polynomial tools

testing

NumPy testing tools

f2py

Fortran to Python Interface Generator.

distutils

Enhancements to distutils with support for

Fortran compilers support and more.

Utilities

---------

test

Run numpy unittests

show\_config

Show numpy build configuration

dual

Overwrite certain functions with high-performance Scipy tools

matlib

Make everything matrices.

\_\_version\_\_

NumPy version string

Viewing documentation using IPython

-----------------------------------

Start IPython with the NumPy profile (``ipython -p numpy``), which will

import `numpy` under the alias `np`. Then, use the ``cpaste`` command to

paste examples into the shell. To see which functions are available in

`numpy`, type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use

``np.\*cos\*?<ENTER>`` (where ``<ENTER>`` refers to the ENTER key) to narrow

down the list. To view the docstring for a function, use

``np.cos?<ENTER>`` (to view the docstring) and ``np.cos??<ENTER>`` (to view

the source code).

Copies vs. in-place operation

-----------------------------

Most of the functions in `numpy` return a copy of the array argument

(e.g., `np.sort`). In-place versions of these functions are often

available as array methods, i.e. ``x = np.array([1,2,3]); x.sort()``.

Exceptions to this rule are documented.

PACKAGE CONTENTS

\_\_config\_\_

\_distributor\_init

\_globals

\_pytesttester

compat (package)

conftest

core (package)

ctypeslib

distutils (package)

doc (package)

dual

f2py (package)

fft (package)

lib (package)

linalg (package)

ma (package)

matlib

matrixlib (package)

polynomial (package)

random (package)

setup

testing (package)

tests (package)

version

SUBMODULES

\_mat

char

emath

rec

CLASSES

builtins.DeprecationWarning(builtins.Warning)

ModuleDeprecationWarning

builtins.IndexError(builtins.LookupError)

AxisError(builtins.ValueError, builtins.IndexError)

builtins.RuntimeError(builtins.Exception)

TooHardError

builtins.RuntimeWarning(builtins.Warning)

ComplexWarning

builtins.UserWarning(builtins.Warning)

RankWarning

VisibleDeprecationWarning

builtins.ValueError(builtins.Exception)

AxisError(builtins.ValueError, builtins.IndexError)

builtins.bytes(builtins.object)

bytes\_(builtins.bytes, character)

builtins.object

builtins.int

builtins.str

str\_(builtins.str, character)

DataSource

MachAr

broadcast

busdaycalendar

dtype

finfo

flatiter

format\_parser

generic

bool\_

datetime64

flexible

character

bytes\_(builtins.bytes, character)

void

record

number

inexact

complexfloating

complex128(complexfloating, builtins.complex)

complex128

complex64

floating

float16

float32

float64(floating, builtins.float)

float64

integer

signedinteger

int16

int32

int32

int64

int8

timedelta64

unsignedinteger

uint16

uint32

uint32

uint64

uint8

object\_

iinfo

ndarray

chararray

matrix

memmap

recarray

ndenumerate

ndindex

nditer

poly1d

ufunc

vectorize

contextlib.ContextDecorator(builtins.object)

errstate

class AxisError(builtins.ValueError, builtins.IndexError)

| AxisError(axis, ndim=None, msg\_prefix=None)

|

| Axis supplied was invalid.

|

| Method resolution order:

| AxisError

| builtins.ValueError

| builtins.IndexError

| builtins.LookupError

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Methods defined here:

|

| \_\_init\_\_(self, axis, ndim=None, msg\_prefix=None)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.ValueError:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

class ComplexWarning(builtins.RuntimeWarning)

| The warning raised when casting a complex dtype to a real dtype.

|

| As implemented, casting a complex number to a real discards its imaginary

| part, but this behavior may not be what the user actually wants.

|

| Method resolution order:

| ComplexWarning

| builtins.RuntimeWarning

| builtins.Warning

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.RuntimeWarning:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.RuntimeWarning:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

class DataSource(builtins.object)

| DataSource(destpath='.')

|

| DataSource(destpath='.')

|

| A generic data source file (file, http, ftp, ...).

|

| DataSources can be local files or remote files/URLs. The files may

| also be compressed or uncompressed. DataSource hides some of the

| low-level details of downloading the file, allowing you to simply pass

| in a valid file path (or URL) and obtain a file object.

|

| Parameters

| ----------

| destpath : str or None, optional

| Path to the directory where the source file gets downloaded to for

| use. If `destpath` is None, a temporary directory will be created.

| The default path is the current directory.

|

| Notes

| -----

| URLs require a scheme string (``http://``) to be used, without it they

| will fail::

|

| >>> repos = np.DataSource()

| >>> repos.exists('www.google.com/index.html')

| False

| >>> repos.exists('http://www.google.com/index.html')

| True

|

| Temporary directories are deleted when the DataSource is deleted.

|

| Examples

| --------

| ::

|

| >>> ds = np.DataSource('/home/guido')

| >>> urlname = 'http://www.google.com/'

| >>> gfile = ds.open('http://www.google.com/')

| >>> ds.abspath(urlname)

| '/home/guido/www.google.com/index.html'

|

| >>> ds = np.DataSource(None) # use with temporary file

| >>> ds.open('/home/guido/foobar.txt')

| <open file '/home/guido.foobar.txt', mode 'r' at 0x91d4430>

| >>> ds.abspath('/home/guido/foobar.txt')

| '/tmp/.../home/guido/foobar.txt'

|

| Methods defined here:

|

| \_\_del\_\_(self)

|

| \_\_init\_\_(self, destpath='.')

| Create a DataSource with a local path at destpath.

|

| abspath(self, path)

| Return absolute path of file in the DataSource directory.

|

| If `path` is an URL, then `abspath` will return either the location

| the file exists locally or the location it would exist when opened

| using the `open` method.

|

| Parameters

| ----------

| path : str

| Can be a local file or a remote URL.

|

| Returns

| -------

| out : str

| Complete path, including the `DataSource` destination directory.

|

| Notes

| -----

| The functionality is based on `os.path.abspath`.

|

| exists(self, path)

| Test if path exists.

|

| Test if `path` exists as (and in this order):

|

| - a local file.

| - a remote URL that has been downloaded and stored locally in the

| `DataSource` directory.

| - a remote URL that has not been downloaded, but is valid and

| accessible.

|

| Parameters

| ----------

| path : str

| Can be a local file or a remote URL.

|

| Returns

| -------

| out : bool

| True if `path` exists.

|

| Notes

| -----

| When `path` is an URL, `exists` will return True if it's either

| stored locally in the `DataSource` directory, or is a valid remote

| URL. `DataSource` does not discriminate between the two, the file

| is accessible if it exists in either location.

|

| open(self, path, mode='r', encoding=None, newline=None)

| Open and return file-like object.

|

| If `path` is an URL, it will be downloaded, stored in the

| `DataSource` directory and opened from there.

|

| Parameters

| ----------

| path : str

| Local file path or URL to open.

| mode : {'r', 'w', 'a'}, optional

| Mode to open `path`. Mode 'r' for reading, 'w' for writing,

| 'a' to append. Available modes depend on the type of object

| specified by `path`. Default is 'r'.

| encoding : {None, str}, optional

| Open text file with given encoding. The default encoding will be

| what `io.open` uses.

| newline : {None, str}, optional

| Newline to use when reading text file.

|

| Returns

| -------

| out : file object

| File object.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class MachAr(builtins.object)

| MachAr(float\_conv=<class 'float'>, int\_conv=<class 'int'>, float\_to\_float=<class 'float'>, float\_to\_str=<function MachAr.<lambda> at 0x07C32CD8>, title='Python floating point number')

|

| Diagnosing machine parameters.

|

| Attributes

| ----------

| ibeta : int

| Radix in which numbers are represented.

| it : int

| Number of base-`ibeta` digits in the floating point mantissa M.

| machep : int

| Exponent of the smallest (most negative) power of `ibeta` that,

| added to 1.0, gives something different from 1.0

| eps : float

| Floating-point number ``beta\*\*machep`` (floating point precision)

| negep : int

| Exponent of the smallest power of `ibeta` that, subtracted

| from 1.0, gives something different from 1.0.

| epsneg : float

| Floating-point number ``beta\*\*negep``.

| iexp : int

| Number of bits in the exponent (including its sign and bias).

| minexp : int

| Smallest (most negative) power of `ibeta` consistent with there

| being no leading zeros in the mantissa.

| xmin : float

| Floating point number ``beta\*\*minexp`` (the smallest [in

| magnitude] usable floating value).

| maxexp : int

| Smallest (positive) power of `ibeta` that causes overflow.

| xmax : float

| ``(1-epsneg) \* beta\*\*maxexp`` (the largest [in magnitude]

| usable floating value).

| irnd : int

| In ``range(6)``, information on what kind of rounding is done

| in addition, and on how underflow is handled.

| ngrd : int

| Number of 'guard digits' used when truncating the product

| of two mantissas to fit the representation.

| epsilon : float

| Same as `eps`.

| tiny : float

| Same as `xmin`.

| huge : float

| Same as `xmax`.

| precision : float

| ``- int(-log10(eps))``

| resolution : float

| ``- 10\*\*(-precision)``

|

| Parameters

| ----------

| float\_conv : function, optional

| Function that converts an integer or integer array to a float

| or float array. Default is `float`.

| int\_conv : function, optional

| Function that converts a float or float array to an integer or

| integer array. Default is `int`.

| float\_to\_float : function, optional

| Function that converts a float array to float. Default is `float`.

| Note that this does not seem to do anything useful in the current

| implementation.

| float\_to\_str : function, optional

| Function that converts a single float to a string. Default is

| ``lambda v:'%24.16e' %v``.

| title : str, optional

| Title that is printed in the string representation of `MachAr`.

|

| See Also

| --------

| finfo : Machine limits for floating point types.

| iinfo : Machine limits for integer types.

|

| References

| ----------

| .. [1] Press, Teukolsky, Vetterling and Flannery,

| "Numerical Recipes in C++," 2nd ed,

| Cambridge University Press, 2002, p. 31.

|

| Methods defined here:

|

| \_\_init\_\_(self, float\_conv=<class 'float'>, int\_conv=<class 'int'>, float\_to\_float=<class 'float'>, float\_to\_str=<function MachAr.<lambda> at 0x07C32CD8>, title='Python floating point number')

| float\_conv - convert integer to float (array)

| int\_conv - convert float (array) to integer

| float\_to\_float - convert float array to float

| float\_to\_str - convert array float to str

| title - description of used floating point numbers

|

| \_\_str\_\_(self)

| Return str(self).

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class ModuleDeprecationWarning(builtins.DeprecationWarning)

| Module deprecation warning.

|

| The nose tester turns ordinary Deprecation warnings into test failures.

| That makes it hard to deprecate whole modules, because they get

| imported by default. So this is a special Deprecation warning that the

| nose tester will let pass without making tests fail.

|

| Method resolution order:

| ModuleDeprecationWarning

| builtins.DeprecationWarning

| builtins.Warning

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.DeprecationWarning:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.DeprecationWarning:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

class RankWarning(builtins.UserWarning)

| Issued by `polyfit` when the Vandermonde matrix is rank deficient.

|

| For more information, a way to suppress the warning, and an example of

| `RankWarning` being issued, see `polyfit`.

|

| Method resolution order:

| RankWarning

| builtins.UserWarning

| builtins.Warning

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.UserWarning:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.UserWarning:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

class TooHardError(builtins.RuntimeError)

| Unspecified run-time error.

|

| Method resolution order:

| TooHardError

| builtins.RuntimeError

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.RuntimeError:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.RuntimeError:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

class VisibleDeprecationWarning(builtins.UserWarning)

| Visible deprecation warning.

|

| By default, python will not show deprecation warnings, so this class

| can be used when a very visible warning is helpful, for example because

| the usage is most likely a user bug.

|

| Method resolution order:

| VisibleDeprecationWarning

| builtins.UserWarning

| builtins.Warning

| builtins.Exception

| builtins.BaseException

| builtins.object

|

| Data descriptors defined here:

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.UserWarning:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.UserWarning:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.BaseException:

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| with\_traceback(...)

| Exception.with\_traceback(tb) --

| set self.\_\_traceback\_\_ to tb and return self.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from builtins.BaseException:

|

| \_\_cause\_\_

| exception cause

|

| \_\_context\_\_

| exception context

|

| \_\_dict\_\_

|

| \_\_suppress\_context\_\_

|

| \_\_traceback\_\_

|

| args

bool8 = class bool\_(generic)

| Boolean type (True or False), stored as a byte.

| Character code: ``'?'``.

| Alias: ``np.bool8``.

|

| Method resolution order:

| bool\_

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class bool\_(generic)

| Boolean type (True or False), stored as a byte.

| Character code: ``'?'``.

| Alias: ``np.bool8``.

|

| Method resolution order:

| bool\_

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class broadcast(builtins.object)

| Produce an object that mimics broadcasting.

|

| Parameters

| ----------

| in1, in2, ... : array\_like

| Input parameters.

|

| Returns

| -------

| b : broadcast object

| Broadcast the input parameters against one another, and

| return an object that encapsulates the result.

| Amongst others, it has ``shape`` and ``nd`` properties, and

| may be used as an iterator.

|

| See Also

| --------

| broadcast\_arrays

| broadcast\_to

|

| Examples

| --------

|

| Manually adding two vectors, using broadcasting:

|

| >>> x = np.array([[1], [2], [3]])

| >>> y = np.array([4, 5, 6])

| >>> b = np.broadcast(x, y)

|

| >>> out = np.empty(b.shape)

| >>> out.flat = [u+v for (u,v) in b]

| >>> out

| array([[5., 6., 7.],

| [6., 7., 8.],

| [7., 8., 9.]])

|

| Compare against built-in broadcasting:

|

| >>> x + y

| array([[5, 6, 7],

| [6, 7, 8],

| [7, 8, 9]])

|

| Methods defined here:

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| reset(...)

| reset()

|

| Reset the broadcasted result's iterator(s).

|

| Parameters

| ----------

| None

|

| Returns

| -------

| None

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.index

| 0

| >>> next(b), next(b), next(b)

| ((1, 4), (2, 4), (3, 4))

| >>> b.index

| 3

| >>> b.reset()

| >>> b.index

| 0

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| index

| current index in broadcasted result

|

| Examples

| --------

| >>> x = np.array([[1], [2], [3]])

| >>> y = np.array([4, 5, 6])

| >>> b = np.broadcast(x, y)

| >>> b.index

| 0

| >>> next(b), next(b), next(b)

| ((1, 4), (1, 5), (1, 6))

| >>> b.index

| 3

|

| iters

| tuple of iterators along ``self``'s "components."

|

| Returns a tuple of `numpy.flatiter` objects, one for each "component"

| of ``self``.

|

| See Also

| --------

| numpy.flatiter

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> row, col = b.iters

| >>> row.next(), col.next()

| (1, 4)

|

| nd

| Number of dimensions of broadcasted result. For code intended for NumPy

| 1.12.0 and later the more consistent `ndim` is preferred.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.nd

| 2

|

| ndim

| Number of dimensions of broadcasted result. Alias for `nd`.

|

| .. versionadded:: 1.12.0

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.ndim

| 2

|

| numiter

| Number of iterators possessed by the broadcasted result.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.numiter

| 2

|

| shape

| Shape of broadcasted result.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.shape

| (3, 3)

|

| size

| Total size of broadcasted result.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> y = np.array([[4], [5], [6]])

| >>> b = np.broadcast(x, y)

| >>> b.size

| 9

class busdaycalendar(builtins.object)

| busdaycalendar(weekmask='1111100', holidays=None)

|

| A business day calendar object that efficiently stores information

| defining valid days for the busday family of functions.

|

| The default valid days are Monday through Friday ("business days").

| A busdaycalendar object can be specified with any set of weekly

| valid days, plus an optional "holiday" dates that always will be invalid.

|

| Once a busdaycalendar object is created, the weekmask and holidays

| cannot be modified.

|

| .. versionadded:: 1.7.0

|

| Parameters

| ----------

| weekmask : str or array\_like of bool, optional

| A seven-element array indicating which of Monday through Sunday are

| valid days. May be specified as a length-seven list or array, like

| [1,1,1,1,1,0,0]; a length-seven string, like '1111100'; or a string

| like "Mon Tue Wed Thu Fri", made up of 3-character abbreviations for

| weekdays, optionally separated by white space. Valid abbreviations

| are: Mon Tue Wed Thu Fri Sat Sun

| holidays : array\_like of datetime64[D], optional

| An array of dates to consider as invalid dates, no matter which

| weekday they fall upon. Holiday dates may be specified in any

| order, and NaT (not-a-time) dates are ignored. This list is

| saved in a normalized form that is suited for fast calculations

| of valid days.

|

| Returns

| -------

| out : busdaycalendar

| A business day calendar object containing the specified

| weekmask and holidays values.

|

| See Also

| --------

| is\_busday : Returns a boolean array indicating valid days.

| busday\_offset : Applies an offset counted in valid days.

| busday\_count : Counts how many valid days are in a half-open date range.

|

| Attributes

| ----------

| Note: once a busdaycalendar object is created, you cannot modify the

| weekmask or holidays. The attributes return copies of internal data.

| weekmask : (copy) seven-element array of bool

| holidays : (copy) sorted array of datetime64[D]

|

| Examples

| --------

| >>> # Some important days in July

| ... bdd = np.busdaycalendar(

| ... holidays=['2011-07-01', '2011-07-04', '2011-07-17'])

| >>> # Default is Monday to Friday weekdays

| ... bdd.weekmask

| array([ True, True, True, True, True, False, False])

| >>> # Any holidays already on the weekend are removed

| ... bdd.holidays

| array(['2011-07-01', '2011-07-04'], dtype='datetime64[D]')

|

| Methods defined here:

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| holidays

| A copy of the holiday array indicating additional invalid days.

|

| weekmask

| A copy of the seven-element boolean mask indicating valid days.

byte = class int8(signedinteger)

| Signed integer type, compatible with C ``char``.

| Character code: ``'b'``.

| Canonical name: ``np.byte``.

| Alias \*on this platform\*: ``np.int8``: 8-bit signed integer (-128 to 127).

|

| Method resolution order:

| int8

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

bytes0 = class bytes\_(builtins.bytes, character)

| bytes(iterable\_of\_ints) -> bytes

| bytes(string, encoding[, errors]) -> bytes

| bytes(bytes\_or\_buffer) -> immutable copy of bytes\_or\_buffer

| bytes(int) -> bytes object of size given by the parameter initialized with null bytes

| bytes() -> empty bytes object

|

| Construct an immutable array of bytes from:

| - an iterable yielding integers in range(256)

| - a text string encoded using the specified encoding

| - any object implementing the buffer API.

| - an integer

|

| Method resolution order:

| bytes\_

| builtins.bytes

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.bytes:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| capitalize(...)

| B.capitalize() -> copy of B

|

| Return a copy of B with only its first character capitalized (ASCII)

| and the rest lower-cased.

|

| center(...)

| B.center(width[, fillchar]) -> copy of B

|

| Return B centered in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| count(...)

| B.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of subsection sub in

| bytes B[start:end]. Optional arguments start and end are interpreted

| as in slice notation.

|

| decode(self, /, encoding='utf-8', errors='strict')

| Decode the bytes using the codec registered for encoding.

|

| encoding

| The encoding with which to decode the bytes.

| errors

| The error handling scheme to use for the handling of decoding errors.

| The default is 'strict' meaning that decoding errors raise a

| UnicodeDecodeError. Other possible values are 'ignore' and 'replace'

| as well as any other name registered with codecs.register\_error that

| can handle UnicodeDecodeErrors.

|

| endswith(...)

| B.endswith(suffix[, start[, end]]) -> bool

|

| Return True if B ends with the specified suffix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| suffix can also be a tuple of bytes to try.

|

| expandtabs(...)

| B.expandtabs(tabsize=8) -> copy of B

|

| Return a copy of B where all tab characters are expanded using spaces.

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| B.find(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| hex(...)

| B.hex() -> string

|

| Create a string of hexadecimal numbers from a bytes object.

| Example: b'\xb9\x01\xef'.hex() -> 'b901ef'.

|

| index(...)

| B.index(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the subsection is not found.

|

| isalnum(...)

| B.isalnum() -> bool

|

| Return True if all characters in B are alphanumeric

| and there is at least one character in B, False otherwise.

|

| isalpha(...)

| B.isalpha() -> bool

|

| Return True if all characters in B are alphabetic

| and there is at least one character in B, False otherwise.

|

| isascii(...)

| B.isascii() -> bool

|

| Return True if B is empty or all characters in B are ASCII,

| False otherwise.

|

| isdigit(...)

| B.isdigit() -> bool

|

| Return True if all characters in B are digits

| and there is at least one character in B, False otherwise.

|

| islower(...)

| B.islower() -> bool

|

| Return True if all cased characters in B are lowercase and there is

| at least one cased character in B, False otherwise.

|

| isspace(...)

| B.isspace() -> bool

|

| Return True if all characters in B are whitespace

| and there is at least one character in B, False otherwise.

|

| istitle(...)

| B.istitle() -> bool

|

| Return True if B is a titlecased string and there is at least one

| character in B, i.e. uppercase characters may only follow uncased

| characters and lowercase characters only cased ones. Return False

| otherwise.

|

| isupper(...)

| B.isupper() -> bool

|

| Return True if all cased characters in B are uppercase and there is

| at least one cased character in B, False otherwise.

|

| join(self, iterable\_of\_bytes, /)

| Concatenate any number of bytes objects.

|

| The bytes whose method is called is inserted in between each pair.

|

| The result is returned as a new bytes object.

|

| Example: b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'.

|

| ljust(...)

| B.ljust(width[, fillchar]) -> copy of B

|

| Return B left justified in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| lower(...)

| B.lower() -> copy of B

|

| Return a copy of B with all ASCII characters converted to lowercase.

|

| lstrip(self, bytes=None, /)

| Strip leading bytes contained in the argument.

|

| If the argument is omitted or None, strip leading ASCII whitespace.

|

| partition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original bytes

| object and two empty bytes objects.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| B.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| B.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raise ValueError when the subsection is not found.

|

| rjust(...)

| B.rjust(width[, fillchar]) -> copy of B

|

| Return B right justified in a string of length width. Padding is

| done using the specified fill character (default is a space)

|

| rpartition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty bytes

| objects and the original bytes object.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splitting is done starting at the end of the bytes and working to the front.

|

| rstrip(self, bytes=None, /)

| Strip trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip trailing ASCII whitespace.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the bytes, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| B.startswith(prefix[, start[, end]]) -> bool

|

| Return True if B starts with the specified prefix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| prefix can also be a tuple of bytes to try.

|

| strip(self, bytes=None, /)

| Strip leading and trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip leading and trailing ASCII whitespace.

|

| swapcase(...)

| B.swapcase() -> copy of B

|

| Return a copy of B with uppercase ASCII characters converted

| to lowercase ASCII and vice versa.

|

| title(...)

| B.title() -> copy of B

|

| Return a titlecased version of B, i.e. ASCII words start with uppercase

| characters, all remaining cased characters have lowercase.

|

| translate(self, table, /, delete=b'')

| Return a copy with each character mapped by the given translation table.

|

| table

| Translation table, which must be a bytes object of length 256.

|

| All characters occurring in the optional argument delete are removed.

| The remaining characters are mapped through the given translation table.

|

| upper(...)

| B.upper() -> copy of B

|

| Return a copy of B with all ASCII characters converted to uppercase.

|

| zfill(...)

| B.zfill(width) -> copy of B

|

| Pad a numeric string B with zeros on the left, to fill a field

| of the specified width. B is never truncated.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.bytes:

|

| fromhex(string, /) from builtins.type

| Create a bytes object from a string of hexadecimal numbers.

|

| Spaces between two numbers are accepted.

| Example: bytes.fromhex('B9 01EF') -> b'\\xb9\\x01\\xef'.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.bytes:

|

| maketrans(frm, to, /)

| Return a translation table useable for the bytes or bytearray translate method.

|

| The returned table will be one where each byte in frm is mapped to the byte at

| the same position in to.

|

| The bytes objects frm and to must be of the same length.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class bytes\_(builtins.bytes, character)

| bytes(iterable\_of\_ints) -> bytes

| bytes(string, encoding[, errors]) -> bytes

| bytes(bytes\_or\_buffer) -> immutable copy of bytes\_or\_buffer

| bytes(int) -> bytes object of size given by the parameter initialized with null bytes

| bytes() -> empty bytes object

|

| Construct an immutable array of bytes from:

| - an iterable yielding integers in range(256)

| - a text string encoded using the specified encoding

| - any object implementing the buffer API.

| - an integer

|

| Method resolution order:

| bytes\_

| builtins.bytes

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.bytes:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| capitalize(...)

| B.capitalize() -> copy of B

|

| Return a copy of B with only its first character capitalized (ASCII)

| and the rest lower-cased.

|

| center(...)

| B.center(width[, fillchar]) -> copy of B

|

| Return B centered in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| count(...)

| B.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of subsection sub in

| bytes B[start:end]. Optional arguments start and end are interpreted

| as in slice notation.

|

| decode(self, /, encoding='utf-8', errors='strict')

| Decode the bytes using the codec registered for encoding.

|

| encoding

| The encoding with which to decode the bytes.

| errors

| The error handling scheme to use for the handling of decoding errors.

| The default is 'strict' meaning that decoding errors raise a

| UnicodeDecodeError. Other possible values are 'ignore' and 'replace'

| as well as any other name registered with codecs.register\_error that

| can handle UnicodeDecodeErrors.

|

| endswith(...)

| B.endswith(suffix[, start[, end]]) -> bool

|

| Return True if B ends with the specified suffix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| suffix can also be a tuple of bytes to try.

|

| expandtabs(...)

| B.expandtabs(tabsize=8) -> copy of B

|

| Return a copy of B where all tab characters are expanded using spaces.

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| B.find(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| hex(...)

| B.hex() -> string

|

| Create a string of hexadecimal numbers from a bytes object.

| Example: b'\xb9\x01\xef'.hex() -> 'b901ef'.

|

| index(...)

| B.index(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the subsection is not found.

|

| isalnum(...)

| B.isalnum() -> bool

|

| Return True if all characters in B are alphanumeric

| and there is at least one character in B, False otherwise.

|

| isalpha(...)

| B.isalpha() -> bool

|

| Return True if all characters in B are alphabetic

| and there is at least one character in B, False otherwise.

|

| isascii(...)

| B.isascii() -> bool

|

| Return True if B is empty or all characters in B are ASCII,

| False otherwise.

|

| isdigit(...)

| B.isdigit() -> bool

|

| Return True if all characters in B are digits

| and there is at least one character in B, False otherwise.

|

| islower(...)

| B.islower() -> bool

|

| Return True if all cased characters in B are lowercase and there is

| at least one cased character in B, False otherwise.

|

| isspace(...)

| B.isspace() -> bool

|

| Return True if all characters in B are whitespace

| and there is at least one character in B, False otherwise.

|

| istitle(...)

| B.istitle() -> bool

|

| Return True if B is a titlecased string and there is at least one

| character in B, i.e. uppercase characters may only follow uncased

| characters and lowercase characters only cased ones. Return False

| otherwise.

|

| isupper(...)

| B.isupper() -> bool

|

| Return True if all cased characters in B are uppercase and there is

| at least one cased character in B, False otherwise.

|

| join(self, iterable\_of\_bytes, /)

| Concatenate any number of bytes objects.

|

| The bytes whose method is called is inserted in between each pair.

|

| The result is returned as a new bytes object.

|

| Example: b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'.

|

| ljust(...)

| B.ljust(width[, fillchar]) -> copy of B

|

| Return B left justified in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| lower(...)

| B.lower() -> copy of B

|

| Return a copy of B with all ASCII characters converted to lowercase.

|

| lstrip(self, bytes=None, /)

| Strip leading bytes contained in the argument.

|

| If the argument is omitted or None, strip leading ASCII whitespace.

|

| partition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original bytes

| object and two empty bytes objects.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| B.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| B.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raise ValueError when the subsection is not found.

|

| rjust(...)

| B.rjust(width[, fillchar]) -> copy of B

|

| Return B right justified in a string of length width. Padding is

| done using the specified fill character (default is a space)

|

| rpartition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty bytes

| objects and the original bytes object.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splitting is done starting at the end of the bytes and working to the front.

|

| rstrip(self, bytes=None, /)

| Strip trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip trailing ASCII whitespace.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the bytes, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| B.startswith(prefix[, start[, end]]) -> bool

|

| Return True if B starts with the specified prefix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| prefix can also be a tuple of bytes to try.

|

| strip(self, bytes=None, /)

| Strip leading and trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip leading and trailing ASCII whitespace.

|

| swapcase(...)

| B.swapcase() -> copy of B

|

| Return a copy of B with uppercase ASCII characters converted

| to lowercase ASCII and vice versa.

|

| title(...)

| B.title() -> copy of B

|

| Return a titlecased version of B, i.e. ASCII words start with uppercase

| characters, all remaining cased characters have lowercase.

|

| translate(self, table, /, delete=b'')

| Return a copy with each character mapped by the given translation table.

|

| table

| Translation table, which must be a bytes object of length 256.

|

| All characters occurring in the optional argument delete are removed.

| The remaining characters are mapped through the given translation table.

|

| upper(...)

| B.upper() -> copy of B

|

| Return a copy of B with all ASCII characters converted to uppercase.

|

| zfill(...)

| B.zfill(width) -> copy of B

|

| Pad a numeric string B with zeros on the left, to fill a field

| of the specified width. B is never truncated.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.bytes:

|

| fromhex(string, /) from builtins.type

| Create a bytes object from a string of hexadecimal numbers.

|

| Spaces between two numbers are accepted.

| Example: bytes.fromhex('B9 01EF') -> b'\\xb9\\x01\\xef'.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.bytes:

|

| maketrans(frm, to, /)

| Return a translation table useable for the bytes or bytearray translate method.

|

| The returned table will be one where each byte in frm is mapped to the byte at

| the same position in to.

|

| The bytes objects frm and to must be of the same length.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

cdouble = class complex128(complexfloating, builtins.complex)

| cdouble(real=0, imag=0)

|

| Complex number type composed of two double-precision floating-point

| numbers, compatible with Python `complex`.

| Character code: ``'D'``.

| Canonical name: ``np.cdouble``.

| Alias: ``np.cfloat``.

| Alias: ``np.complex\_``.

| Alias \*on this platform\*: ``np.complex128``: Complex number type composed of 2 64-bit-precision floating-point numbers.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.complex

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.complex:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(...)

cfloat = class complex128(complexfloating, builtins.complex)

| cfloat(real=0, imag=0)

|

| Complex number type composed of two double-precision floating-point

| numbers, compatible with Python `complex`.

| Character code: ``'D'``.

| Canonical name: ``np.cdouble``.

| Alias: ``np.cfloat``.

| Alias: ``np.complex\_``.

| Alias \*on this platform\*: ``np.complex128``: Complex number type composed of 2 64-bit-precision floating-point numbers.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.complex

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.complex:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(...)

class character(flexible)

| Abstract base class of all character string scalar types.

|

| Method resolution order:

| character

| flexible

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

class chararray(ndarray)

| chararray(shape, itemsize=1, unicode=False, buffer=None, offset=0, strides=None, order='C')

|

| chararray(shape, itemsize=1, unicode=False, buffer=None, offset=0,

| strides=None, order=None)

|

| Provides a convenient view on arrays of string and unicode values.

|

| .. note::

| The `chararray` class exists for backwards compatibility with

| Numarray, it is not recommended for new development. Starting from numpy

| 1.4, if one needs arrays of strings, it is recommended to use arrays of

| `dtype` `object\_`, `string\_` or `unicode\_`, and use the free functions

| in the `numpy.char` module for fast vectorized string operations.

|

| Versus a regular NumPy array of type `str` or `unicode`, this

| class adds the following functionality:

|

| 1) values automatically have whitespace removed from the end

| when indexed

|

| 2) comparison operators automatically remove whitespace from the

| end when comparing values

|

| 3) vectorized string operations are provided as methods

| (e.g. `.endswith`) and infix operators (e.g. ``"+", "\*", "%"``)

|

| chararrays should be created using `numpy.char.array` or

| `numpy.char.asarray`, rather than this constructor directly.

|

| This constructor creates the array, using `buffer` (with `offset`

| and `strides`) if it is not ``None``. If `buffer` is ``None``, then

| constructs a new array with `strides` in "C order", unless both

| ``len(shape) >= 2`` and ``order='Fortran'``, in which case `strides`

| is in "Fortran order".

|

| Methods

| -------

| astype

| argsort

| copy

| count

| decode

| dump

| dumps

| encode

| endswith

| expandtabs

| fill

| find

| flatten

| getfield

| index

| isalnum

| isalpha

| isdecimal

| isdigit

| islower

| isnumeric

| isspace

| istitle

| isupper

| item

| join

| ljust

| lower

| lstrip

| nonzero

| put

| ravel

| repeat

| replace

| reshape

| resize

| rfind

| rindex

| rjust

| rsplit

| rstrip

| searchsorted

| setfield

| setflags

| sort

| split

| splitlines

| squeeze

| startswith

| strip

| swapaxes

| swapcase

| take

| title

| tofile

| tolist

| tostring

| translate

| transpose

| upper

| view

| zfill

|

| Parameters

| ----------

| shape : tuple

| Shape of the array.

| itemsize : int, optional

| Length of each array element, in number of characters. Default is 1.

| unicode : bool, optional

| Are the array elements of type unicode (True) or string (False).

| Default is False.

| buffer : int, optional

| Memory address of the start of the array data. Default is None,

| in which case a new array is created.

| offset : int, optional

| Fixed stride displacement from the beginning of an axis?

| Default is 0. Needs to be >=0.

| strides : array\_like of ints, optional

| Strides for the array (see `ndarray.strides` for full description).

| Default is None.

| order : {'C', 'F'}, optional

| The order in which the array data is stored in memory: 'C' ->

| "row major" order (the default), 'F' -> "column major"

| (Fortran) order.

|

| Examples

| --------

| >>> charar = np.chararray((3, 3))

| >>> charar[:] = 'a'

| >>> charar

| chararray([[b'a', b'a', b'a'],

| [b'a', b'a', b'a'],

| [b'a', b'a', b'a']], dtype='|S1')

|

| >>> charar = np.chararray(charar.shape, itemsize=5)

| >>> charar[:] = 'abc'

| >>> charar

| chararray([[b'abc', b'abc', b'abc'],

| [b'abc', b'abc', b'abc'],

| [b'abc', b'abc', b'abc']], dtype='|S5')

|

| Method resolution order:

| chararray

| ndarray

| builtins.object

|

| Methods defined here:

|

| \_\_add\_\_(self, other)

| Return (self + other), that is string concatenation,

| element-wise for a pair of array\_likes of str or unicode.

|

| See also

| --------

| add

|

| \_\_array\_finalize\_\_(self, obj)

| None.

|

| \_\_eq\_\_(self, other)

| Return (self == other) element-wise.

|

| See also

| --------

| equal

|

| \_\_ge\_\_(self, other)

| Return (self >= other) element-wise.

|

| See also

| --------

| greater\_equal

|

| \_\_getitem\_\_(self, obj)

| Return self[key].

|

| \_\_gt\_\_(self, other)

| Return (self > other) element-wise.

|

| See also

| --------

| greater

|

| \_\_le\_\_(self, other)

| Return (self <= other) element-wise.

|

| See also

| --------

| less\_equal

|

| \_\_lt\_\_(self, other)

| Return (self < other) element-wise.

|

| See also

| --------

| less

|

| \_\_mod\_\_(self, i)

| Return (self % i), that is pre-Python 2.6 string formatting

| (iterpolation), element-wise for a pair of array\_likes of `string\_`

| or `unicode\_`.

|

| See also

| --------

| mod

|

| \_\_mul\_\_(self, i)

| Return (self \* i), that is string multiple concatenation,

| element-wise.

|

| See also

| --------

| multiply

|

| \_\_ne\_\_(self, other)

| Return (self != other) element-wise.

|

| See also

| --------

| not\_equal

|

| \_\_radd\_\_(self, other)

| Return (other + self), that is string concatenation,

| element-wise for a pair of array\_likes of `string\_` or `unicode\_`.

|

| See also

| --------

| add

|

| \_\_rmod\_\_(self, other)

| Return value%self.

|

| \_\_rmul\_\_(self, i)

| Return (self \* i), that is string multiple concatenation,

| element-wise.

|

| See also

| --------

| multiply

|

| argsort(self, axis=-1, kind=None, order=None)

| a.argsort(axis=-1, kind=None, order=None)

|

| Returns the indices that would sort this array.

|

| Refer to `numpy.argsort` for full documentation.

|

| See Also

| --------

| numpy.argsort : equivalent function

|

| capitalize(self)

| Return a copy of `self` with only the first character of each element

| capitalized.

|

| See also

| --------

| char.capitalize

|

| center(self, width, fillchar=' ')

| Return a copy of `self` with its elements centered in a

| string of length `width`.

|

| See also

| --------

| center

|

| count(self, sub, start=0, end=None)

| Returns an array with the number of non-overlapping occurrences of

| substring `sub` in the range [`start`, `end`].

|

| See also

| --------

| char.count

|

| decode(self, encoding=None, errors=None)

| Calls `str.decode` element-wise.

|

| See also

| --------

| char.decode

|

| encode(self, encoding=None, errors=None)

| Calls `str.encode` element-wise.

|

| See also

| --------

| char.encode

|

| endswith(self, suffix, start=0, end=None)

| Returns a boolean array which is `True` where the string element

| in `self` ends with `suffix`, otherwise `False`.

|

| See also

| --------

| char.endswith

|

| expandtabs(self, tabsize=8)

| Return a copy of each string element where all tab characters are

| replaced by one or more spaces.

|

| See also

| --------

| char.expandtabs

|

| find(self, sub, start=0, end=None)

| For each element, return the lowest index in the string where

| substring `sub` is found.

|

| See also

| --------

| char.find

|

| index(self, sub, start=0, end=None)

| Like `find`, but raises `ValueError` when the substring is not found.

|

| See also

| --------

| char.index

|

| isalnum(self)

| Returns true for each element if all characters in the string

| are alphanumeric and there is at least one character, false

| otherwise.

|

| See also

| --------

| char.isalnum

|

| isalpha(self)

| Returns true for each element if all characters in the string

| are alphabetic and there is at least one character, false

| otherwise.

|

| See also

| --------

| char.isalpha

|

| isdecimal(self)

| For each element in `self`, return True if there are only

| decimal characters in the element.

|

| See also

| --------

| char.isdecimal

|

| isdigit(self)

| Returns true for each element if all characters in the string are

| digits and there is at least one character, false otherwise.

|

| See also

| --------

| char.isdigit

|

| islower(self)

| Returns true for each element if all cased characters in the

| string are lowercase and there is at least one cased character,

| false otherwise.

|

| See also

| --------

| char.islower

|

| isnumeric(self)

| For each element in `self`, return True if there are only

| numeric characters in the element.

|

| See also

| --------

| char.isnumeric

|

| isspace(self)

| Returns true for each element if there are only whitespace

| characters in the string and there is at least one character,

| false otherwise.

|

| See also

| --------

| char.isspace

|

| istitle(self)

| Returns true for each element if the element is a titlecased

| string and there is at least one character, false otherwise.

|

| See also

| --------

| char.istitle

|

| isupper(self)

| Returns true for each element if all cased characters in the

| string are uppercase and there is at least one character, false

| otherwise.

|

| See also

| --------

| char.isupper

|

| join(self, seq)

| Return a string which is the concatenation of the strings in the

| sequence `seq`.

|

| See also

| --------

| char.join

|

| ljust(self, width, fillchar=' ')

| Return an array with the elements of `self` left-justified in a

| string of length `width`.

|

| See also

| --------

| char.ljust

|

| lower(self)

| Return an array with the elements of `self` converted to

| lowercase.

|

| See also

| --------

| char.lower

|

| lstrip(self, chars=None)

| For each element in `self`, return a copy with the leading characters

| removed.

|

| See also

| --------

| char.lstrip

|

| partition(self, sep)

| Partition each element in `self` around `sep`.

|

| See also

| --------

| partition

|

| replace(self, old, new, count=None)

| For each element in `self`, return a copy of the string with all

| occurrences of substring `old` replaced by `new`.

|

| See also

| --------

| char.replace

|

| rfind(self, sub, start=0, end=None)

| For each element in `self`, return the highest index in the string

| where substring `sub` is found, such that `sub` is contained

| within [`start`, `end`].

|

| See also

| --------

| char.rfind

|

| rindex(self, sub, start=0, end=None)

| Like `rfind`, but raises `ValueError` when the substring `sub` is

| not found.

|

| See also

| --------

| char.rindex

|

| rjust(self, width, fillchar=' ')

| Return an array with the elements of `self`

| right-justified in a string of length `width`.

|

| See also

| --------

| char.rjust

|

| rpartition(self, sep)

| Partition each element in `self` around `sep`.

|

| See also

| --------

| rpartition

|

| rsplit(self, sep=None, maxsplit=None)

| For each element in `self`, return a list of the words in

| the string, using `sep` as the delimiter string.

|

| See also

| --------

| char.rsplit

|

| rstrip(self, chars=None)

| For each element in `self`, return a copy with the trailing

| characters removed.

|

| See also

| --------

| char.rstrip

|

| split(self, sep=None, maxsplit=None)

| For each element in `self`, return a list of the words in the

| string, using `sep` as the delimiter string.

|

| See also

| --------

| char.split

|

| splitlines(self, keepends=None)

| For each element in `self`, return a list of the lines in the

| element, breaking at line boundaries.

|

| See also

| --------

| char.splitlines

|

| startswith(self, prefix, start=0, end=None)

| Returns a boolean array which is `True` where the string element

| in `self` starts with `prefix`, otherwise `False`.

|

| See also

| --------

| char.startswith

|

| strip(self, chars=None)

| For each element in `self`, return a copy with the leading and

| trailing characters removed.

|

| See also

| --------

| char.strip

|

| swapcase(self)

| For each element in `self`, return a copy of the string with

| uppercase characters converted to lowercase and vice versa.

|

| See also

| --------

| char.swapcase

|

| title(self)

| For each element in `self`, return a titlecased version of the

| string: words start with uppercase characters, all remaining cased

| characters are lowercase.

|

| See also

| --------

| char.title

|

| translate(self, table, deletechars=None)

| For each element in `self`, return a copy of the string where

| all characters occurring in the optional argument

| `deletechars` are removed, and the remaining characters have

| been mapped through the given translation table.

|

| See also

| --------

| char.translate

|

| upper(self)

| Return an array with the elements of `self` converted to

| uppercase.

|

| See also

| --------

| char.upper

|

| zfill(self, width)

| Return the numeric string left-filled with zeros in a string of

| length `width`.

|

| See also

| --------

| char.zfill

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(subtype, shape, itemsize=1, unicode=False, buffer=None, offset=0, strides=None, order='C')

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_hash\_\_ = None

|

| ----------------------------------------------------------------------

| Methods inherited from ndarray:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| a.\_\_array\_\_(|dtype) -> reference if type unchanged, copy otherwise.

|

| Returns either a new reference to self if dtype is not given or a new array

| of provided data type if dtype is different from the current dtype of the

| array.

|

| \_\_array\_function\_\_(...)

|

| \_\_array\_prepare\_\_(...)

| a.\_\_array\_prepare\_\_(obj) -> Object of same type as ndarray object obj.

|

| \_\_array\_ufunc\_\_(...)

|

| \_\_array\_wrap\_\_(...)

| a.\_\_array\_wrap\_\_(obj) -> Object of same type as ndarray object a.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_copy\_\_(...)

| a.\_\_copy\_\_()

|

| Used if :func:`copy.copy` is called on an array. Returns a copy of the array.

|

| Equivalent to ``a.copy(order='K')``.

|

| \_\_deepcopy\_\_(...)

| a.\_\_deepcopy\_\_(memo, /) -> Deep copy of array.

|

| Used if :func:`copy.deepcopy` is called on an array.

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| Default object formatter.

|

| \_\_iadd\_\_(self, value, /)

| Return self+=value.

|

| \_\_iand\_\_(self, value, /)

| Return self&=value.

|

| \_\_ifloordiv\_\_(self, value, /)

| Return self//=value.

|

| \_\_ilshift\_\_(self, value, /)

| Return self<<=value.

|

| \_\_imatmul\_\_(self, value, /)

| Return self@=value.

|

| \_\_imod\_\_(self, value, /)

| Return self%=value.

|

| \_\_imul\_\_(self, value, /)

| Return self\*=value.

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_ior\_\_(self, value, /)

| Return self|=value.

|

| \_\_ipow\_\_(self, value, /)

| Return self\*\*=value.

|

| \_\_irshift\_\_(self, value, /)

| Return self>>=value.

|

| \_\_isub\_\_(self, value, /)

| Return self-=value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_itruediv\_\_(self, value, /)

| Return self/=value.

|

| \_\_ixor\_\_(self, value, /)

| Return self^=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_matmul\_\_(self, value, /)

| Return self@value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| a.\_\_reduce\_\_()

|

| For pickling.

|

| \_\_reduce\_ex\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmatmul\_\_(self, value, /)

| Return value@self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_setstate\_\_(...)

| a.\_\_setstate\_\_(state, /)

|

| For unpickling.

|

| The `state` argument must be a sequence that contains the following

| elements:

|

| Parameters

| ----------

| version : int

| optional pickle version. If omitted defaults to 0.

| shape : tuple

| dtype : data-type

| isFortran : bool

| rawdata : string or list

| a binary string with the data (or a list if 'a' is an object array)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| a.all(axis=None, out=None, keepdims=False)

|

| Returns True if all elements evaluate to True.

|

| Refer to `numpy.all` for full documentation.

|

| See Also

| --------

| numpy.all : equivalent function

|

| any(...)

| a.any(axis=None, out=None, keepdims=False)

|

| Returns True if any of the elements of `a` evaluate to True.

|

| Refer to `numpy.any` for full documentation.

|

| See Also

| --------

| numpy.any : equivalent function

|

| argmax(...)

| a.argmax(axis=None, out=None)

|

| Return indices of the maximum values along the given axis.

|

| Refer to `numpy.argmax` for full documentation.

|

| See Also

| --------

| numpy.argmax : equivalent function

|

| argmin(...)

| a.argmin(axis=None, out=None)

|

| Return indices of the minimum values along the given axis of `a`.

|

| Refer to `numpy.argmin` for detailed documentation.

|

| See Also

| --------

| numpy.argmin : equivalent function

|

| argpartition(...)

| a.argpartition(kth, axis=-1, kind='introselect', order=None)

|

| Returns the indices that would partition this array.

|

| Refer to `numpy.argpartition` for full documentation.

|

| .. versionadded:: 1.8.0

|

| See Also

| --------

| numpy.argpartition : equivalent function

|

| astype(...)

| a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

|

| Copy of the array, cast to a specified type.

|

| Parameters

| ----------

| dtype : str or dtype

| Typecode or data-type to which the array is cast.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout order of the result.

| 'C' means C order, 'F' means Fortran order, 'A'

| means 'F' order if all the arrays are Fortran contiguous,

| 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur. Defaults to 'unsafe'

| for backwards compatibility.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| subok : bool, optional

| If True, then sub-classes will be passed-through (default), otherwise

| the returned array will be forced to be a base-class array.

| copy : bool, optional

| By default, astype always returns a newly allocated array. If this

| is set to false, and the `dtype`, `order`, and `subok`

| requirements are satisfied, the input array is returned instead

| of a copy.

|

| Returns

| -------

| arr\_t : ndarray

| Unless `copy` is False and the other conditions for returning the input

| array are satisfied (see description for `copy` input parameter), `arr\_t`

| is a new array of the same shape as the input array, with dtype, order

| given by `dtype`, `order`.

|

| Notes

| -----

| .. versionchanged:: 1.17.0

| Casting between a simple data type and a structured one is possible only

| for "unsafe" casting. Casting to multiple fields is allowed, but

| casting from multiple fields is not.

|

| .. versionchanged:: 1.9.0

| Casting from numeric to string types in 'safe' casting mode requires

| that the string dtype length is long enough to store the max

| integer/float value converted.

|

| Raises

| ------

| ComplexWarning

| When casting from complex to float or int. To avoid this,

| one should use ``a.real.astype(t)``.

|

| Examples

| --------

| >>> x = np.array([1, 2, 2.5])

| >>> x

| array([1. , 2. , 2.5])

|

| >>> x.astype(int)

| array([1, 2, 2])

|

| byteswap(...)

| a.byteswap(inplace=False)

|

| Swap the bytes of the array elements

|

| Toggle between low-endian and big-endian data representation by

| returning a byteswapped array, optionally swapped in-place.

|

| Parameters

| ----------

| inplace : bool, optional

| If ``True``, swap bytes in-place, default is ``False``.

|

| Returns

| -------

| out : ndarray

| The byteswapped array. If `inplace` is ``True``, this is

| a view to self.

|

| Examples

| --------

| >>> A = np.array([1, 256, 8755], dtype=np.int16)

| >>> list(map(hex, A))

| ['0x1', '0x100', '0x2233']

| >>> A.byteswap(inplace=True)

| array([ 256, 1, 13090], dtype=int16)

| >>> list(map(hex, A))

| ['0x100', '0x1', '0x3322']

|

| Arrays of strings are not swapped

|

| >>> A = np.array(['ceg', 'fac'])

| >>> A.byteswap()

| Traceback (most recent call last):

| ...

| UnicodeDecodeError: ...

|

| choose(...)

| a.choose(choices, out=None, mode='raise')

|

| Use an index array to construct a new array from a set of choices.

|

| Refer to `numpy.choose` for full documentation.

|

| See Also

| --------

| numpy.choose : equivalent function

|

| clip(...)

| a.clip(min=None, max=None, out=None, \*\*kwargs)

|

| Return an array whose values are limited to ``[min, max]``.

| One of max or min must be given.

|

| Refer to `numpy.clip` for full documentation.

|

| See Also

| --------

| numpy.clip : equivalent function

|

| compress(...)

| a.compress(condition, axis=None, out=None)

|

| Return selected slices of this array along given axis.

|

| Refer to `numpy.compress` for full documentation.

|

| See Also

| --------

| numpy.compress : equivalent function

|

| conj(...)

| a.conj()

|

| Complex-conjugate all elements.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| conjugate(...)

| a.conjugate()

|

| Return the complex conjugate, element-wise.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| copy(...)

| a.copy(order='C')

|

| Return a copy of the array.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout of the copy. 'C' means C-order,

| 'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

| 'C' otherwise. 'K' means match the layout of `a` as closely

| as possible. (Note that this function and :func:`numpy.copy` are very

| similar, but have different default values for their order=

| arguments.)

|

| See also

| --------

| numpy.copy

| numpy.copyto

|

| Examples

| --------

| >>> x = np.array([[1,2,3],[4,5,6]], order='F')

|

| >>> y = x.copy()

|

| >>> x.fill(0)

|

| >>> x

| array([[0, 0, 0],

| [0, 0, 0]])

|

| >>> y

| array([[1, 2, 3],

| [4, 5, 6]])

|

| >>> y.flags['C\_CONTIGUOUS']

| True

|

| cumprod(...)

| a.cumprod(axis=None, dtype=None, out=None)

|

| Return the cumulative product of the elements along the given axis.

|

| Refer to `numpy.cumprod` for full documentation.

|

| See Also

| --------

| numpy.cumprod : equivalent function

|

| cumsum(...)

| a.cumsum(axis=None, dtype=None, out=None)

|

| Return the cumulative sum of the elements along the given axis.

|

| Refer to `numpy.cumsum` for full documentation.

|

| See Also

| --------

| numpy.cumsum : equivalent function

|

| diagonal(...)

| a.diagonal(offset=0, axis1=0, axis2=1)

|

| Return specified diagonals. In NumPy 1.9 the returned array is a

| read-only view instead of a copy as in previous NumPy versions. In

| a future version the read-only restriction will be removed.

|

| Refer to :func:`numpy.diagonal` for full documentation.

|

| See Also

| --------

| numpy.diagonal : equivalent function

|

| dot(...)

| a.dot(b, out=None)

|

| Dot product of two arrays.

|

| Refer to `numpy.dot` for full documentation.

|

| See Also

| --------

| numpy.dot : equivalent function

|

| Examples

| --------

| >>> a = np.eye(2)

| >>> b = np.ones((2, 2)) \* 2

| >>> a.dot(b)

| array([[2., 2.],

| [2., 2.]])

|

| This array method can be conveniently chained:

|

| >>> a.dot(b).dot(b)

| array([[8., 8.],

| [8., 8.]])

|

| dump(...)

| a.dump(file)

|

| Dump a pickle of the array to the specified file.

| The array can be read back with pickle.load or numpy.load.

|

| Parameters

| ----------

| file : str or Path

| A string naming the dump file.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| dumps(...)

| a.dumps()

|

| Returns the pickle of the array as a string.

| pickle.loads or numpy.loads will convert the string back to an array.

|

| Parameters

| ----------

| None

|

| fill(...)

| a.fill(value)

|

| Fill the array with a scalar value.

|

| Parameters

| ----------

| value : scalar

| All elements of `a` will be assigned this value.

|

| Examples

| --------

| >>> a = np.array([1, 2])

| >>> a.fill(0)

| >>> a

| array([0, 0])

| >>> a = np.empty(2)

| >>> a.fill(1)

| >>> a

| array([1., 1.])

|

| flatten(...)

| a.flatten(order='C')

|

| Return a copy of the array collapsed into one dimension.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| 'C' means to flatten in row-major (C-style) order.

| 'F' means to flatten in column-major (Fortran-

| style) order. 'A' means to flatten in column-major

| order if `a` is Fortran \*contiguous\* in memory,

| row-major order otherwise. 'K' means to flatten

| `a` in the order the elements occur in memory.

| The default is 'C'.

|

| Returns

| -------

| y : ndarray

| A copy of the input array, flattened to one dimension.

|

| See Also

| --------

| ravel : Return a flattened array.

| flat : A 1-D flat iterator over the array.

|

| Examples

| --------

| >>> a = np.array([[1,2], [3,4]])

| >>> a.flatten()

| array([1, 2, 3, 4])

| >>> a.flatten('F')

| array([1, 3, 2, 4])

|

| getfield(...)

| a.getfield(dtype, offset=0)

|

| Returns a field of the given array as a certain type.

|

| A field is a view of the array data with a given data-type. The values in

| the view are determined by the given type and the offset into the current

| array in bytes. The offset needs to be such that the view dtype fits in the

| array dtype; for example an array of dtype complex128 has 16-byte elements.

| If taking a view with a 32-bit integer (4 bytes), the offset needs to be

| between 0 and 12 bytes.

|

| Parameters

| ----------

| dtype : str or dtype

| The data type of the view. The dtype size of the view can not be larger

| than that of the array itself.

| offset : int

| Number of bytes to skip before beginning the element view.

|

| Examples

| --------

| >>> x = np.diag([1.+1.j]\*2)

| >>> x[1, 1] = 2 + 4.j

| >>> x

| array([[1.+1.j, 0.+0.j],

| [0.+0.j, 2.+4.j]])

| >>> x.getfield(np.float64)

| array([[1., 0.],

| [0., 2.]])

|

| By choosing an offset of 8 bytes we can select the complex part of the

| array for our view:

|

| >>> x.getfield(np.float64, offset=8)

| array([[1., 0.],

| [0., 4.]])

|

| item(...)

| a.item(\*args)

|

| Copy an element of an array to a standard Python scalar and return it.

|

| Parameters

| ----------

| \\*args : Arguments (variable number and type)

|

| \* none: in this case, the method only works for arrays

| with one element (`a.size == 1`), which element is

| copied into a standard Python scalar object and returned.

|

| \* int\_type: this argument is interpreted as a flat index into

| the array, specifying which element to copy and return.

|

| \* tuple of int\_types: functions as does a single int\_type argument,

| except that the argument is interpreted as an nd-index into the

| array.

|

| Returns

| -------

| z : Standard Python scalar object

| A copy of the specified element of the array as a suitable

| Python scalar

|

| Notes

| -----

| When the data type of `a` is longdouble or clongdouble, item() returns

| a scalar array object because there is no available Python scalar that

| would not lose information. Void arrays return a buffer object for item(),

| unless fields are defined, in which case a tuple is returned.

|

| `item` is very similar to a[args], except, instead of an array scalar,

| a standard Python scalar is returned. This can be useful for speeding up

| access to elements of the array and doing arithmetic on elements of the

| array using Python's optimized math.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.item(3)

| 1

| >>> x.item(7)

| 0

| >>> x.item((0, 1))

| 2

| >>> x.item((2, 2))

| 1

|

| itemset(...)

| a.itemset(\*args)

|

| Insert scalar into an array (scalar is cast to array's dtype, if possible)

|

| There must be at least 1 argument, and define the last argument

| as \*item\*. Then, ``a.itemset(\*args)`` is equivalent to but faster

| than ``a[args] = item``. The item should be a scalar value and `args`

| must select a single item in the array `a`.

|

| Parameters

| ----------

| \\*args : Arguments

| If one argument: a scalar, only used in case `a` is of size 1.

| If two arguments: the last argument is the value to be set

| and must be a scalar, the first argument specifies a single array

| element location. It is either an int or a tuple.

|

| Notes

| -----

| Compared to indexing syntax, `itemset` provides some speed increase

| for placing a scalar into a particular location in an `ndarray`,

| if you must do this. However, generally this is discouraged:

| among other problems, it complicates the appearance of the code.

| Also, when using `itemset` (and `item`) inside a loop, be sure

| to assign the methods to a local variable to avoid the attribute

| look-up at each loop iteration.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.itemset(4, 0)

| >>> x.itemset((2, 2), 9)

| >>> x

| array([[2, 2, 6],

| [1, 0, 6],

| [1, 0, 9]])

|

| max(...)

| a.max(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the maximum along a given axis.

|

| Refer to `numpy.amax` for full documentation.

|

| See Also

| --------

| numpy.amax : equivalent function

|

| mean(...)

| a.mean(axis=None, dtype=None, out=None, keepdims=False)

|

| Returns the average of the array elements along given axis.

|

| Refer to `numpy.mean` for full documentation.

|

| See Also

| --------

| numpy.mean : equivalent function

|

| min(...)

| a.min(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the minimum along a given axis.

|

| Refer to `numpy.amin` for full documentation.

|

| See Also

| --------

| numpy.amin : equivalent function

|

| newbyteorder(...)

| arr.newbyteorder(new\_order='S')

|

| Return the array with the same data viewed with a different byte order.

|

| Equivalent to::

|

| arr.view(arr.dtype.newbytorder(new\_order))

|

| Changes are also made in all fields and sub-arrays of the array data

| type.

|

|

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_arr : array

| New array object with the dtype reflecting given change to the

| byte order.

|

| nonzero(...)

| a.nonzero()

|

| Return the indices of the elements that are non-zero.

|

| Refer to `numpy.nonzero` for full documentation.

|

| See Also

| --------

| numpy.nonzero : equivalent function

|

| prod(...)

| a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True)

|

| Return the product of the array elements over the given axis

|

| Refer to `numpy.prod` for full documentation.

|

| See Also

| --------

| numpy.prod : equivalent function

|

| ptp(...)

| a.ptp(axis=None, out=None, keepdims=False)

|

| Peak to peak (maximum - minimum) value along a given axis.

|

| Refer to `numpy.ptp` for full documentation.

|

| See Also

| --------

| numpy.ptp : equivalent function

|

| put(...)

| a.put(indices, values, mode='raise')

|

| Set ``a.flat[n] = values[n]`` for all `n` in indices.

|

| Refer to `numpy.put` for full documentation.

|

| See Also

| --------

| numpy.put : equivalent function

|

| ravel(...)

| a.ravel([order])

|

| Return a flattened array.

|

| Refer to `numpy.ravel` for full documentation.

|

| See Also

| --------

| numpy.ravel : equivalent function

|

| ndarray.flat : a flat iterator on the array.

|

| repeat(...)

| a.repeat(repeats, axis=None)

|

| Repeat elements of an array.

|

| Refer to `numpy.repeat` for full documentation.

|

| See Also

| --------

| numpy.repeat : equivalent function

|

| reshape(...)

| a.reshape(shape, order='C')

|

| Returns an array containing the same data with a new shape.

|

| Refer to `numpy.reshape` for full documentation.

|

| See Also

| --------

| numpy.reshape : equivalent function

|

| Notes

| -----

| Unlike the free function `numpy.reshape`, this method on `ndarray` allows

| the elements of the shape parameter to be passed in as separate arguments.

| For example, ``a.reshape(10, 11)`` is equivalent to

| ``a.reshape((10, 11))``.

|

| resize(...)

| a.resize(new\_shape, refcheck=True)

|

| Change shape and size of array in-place.

|

| Parameters

| ----------

| new\_shape : tuple of ints, or `n` ints

| Shape of resized array.

| refcheck : bool, optional

| If False, reference count will not be checked. Default is True.

|

| Returns

| -------

| None

|

| Raises

| ------

| ValueError

| If `a` does not own its own data or references or views to it exist,

| and the data memory must be changed.

| PyPy only: will always raise if the data memory must be changed, since

| there is no reliable way to determine if references or views to it

| exist.

|

| SystemError

| If the `order` keyword argument is specified. This behaviour is a

| bug in NumPy.

|

| See Also

| --------

| resize : Return a new array with the specified shape.

|

| Notes

| -----

| This reallocates space for the data area if necessary.

|

| Only contiguous arrays (data elements consecutive in memory) can be

| resized.

|

| The purpose of the reference count check is to make sure you

| do not use this array as a buffer for another Python object and then

| reallocate the memory. However, reference counts can increase in

| other ways so if you are sure that you have not shared the memory

| for this array with another Python object, then you may safely set

| `refcheck` to False.

|

| Examples

| --------

| Shrinking an array: array is flattened (in the order that the data are

| stored in memory), resized, and reshaped:

|

| >>> a = np.array([[0, 1], [2, 3]], order='C')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [1]])

|

| >>> a = np.array([[0, 1], [2, 3]], order='F')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [2]])

|

| Enlarging an array: as above, but missing entries are filled with zeros:

|

| >>> b = np.array([[0, 1], [2, 3]])

| >>> b.resize(2, 3) # new\_shape parameter doesn't have to be a tuple

| >>> b

| array([[0, 1, 2],

| [3, 0, 0]])

|

| Referencing an array prevents resizing...

|

| >>> c = a

| >>> a.resize((1, 1))

| Traceback (most recent call last):

| ...

| ValueError: cannot resize an array that references or is referenced ...

|

| Unless `refcheck` is False:

|

| >>> a.resize((1, 1), refcheck=False)

| >>> a

| array([[0]])

| >>> c

| array([[0]])

|

| round(...)

| a.round(decimals=0, out=None)

|

| Return `a` with each element rounded to the given number of decimals.

|

| Refer to `numpy.around` for full documentation.

|

| See Also

| --------

| numpy.around : equivalent function

|

| searchsorted(...)

| a.searchsorted(v, side='left', sorter=None)

|

| Find indices where elements of v should be inserted in a to maintain order.

|

| For full documentation, see `numpy.searchsorted`

|

| See Also

| --------

| numpy.searchsorted : equivalent function

|

| setfield(...)

| a.setfield(val, dtype, offset=0)

|

| Put a value into a specified place in a field defined by a data-type.

|

| Place `val` into `a`'s field defined by `dtype` and beginning `offset`

| bytes into the field.

|

| Parameters

| ----------

| val : object

| Value to be placed in field.

| dtype : dtype object

| Data-type of the field in which to place `val`.

| offset : int, optional

| The number of bytes into the field at which to place `val`.

|

| Returns

| -------

| None

|

| See Also

| --------

| getfield

|

| Examples

| --------

| >>> x = np.eye(3)

| >>> x.getfield(np.float64)

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

| >>> x.setfield(3, np.int32)

| >>> x.getfield(np.int32)

| array([[3, 3, 3],

| [3, 3, 3],

| [3, 3, 3]], dtype=int32)

| >>> x

| array([[1.0e+000, 1.5e-323, 1.5e-323],

| [1.5e-323, 1.0e+000, 1.5e-323],

| [1.5e-323, 1.5e-323, 1.0e+000]])

| >>> x.setfield(np.eye(3), np.int32)

| >>> x

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

|

| setflags(...)

| a.setflags(write=None, align=None, uic=None)

|

| Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY),

| respectively.

|

| These Boolean-valued flags affect how numpy interprets the memory

| area used by `a` (see Notes below). The ALIGNED flag can only

| be set to True if the data is actually aligned according to the type.

| The WRITEBACKIFCOPY and (deprecated) UPDATEIFCOPY flags can never be set

| to True. The flag WRITEABLE can only be set to True if the array owns its

| own memory, or the ultimate owner of the memory exposes a writeable buffer

| interface, or is a string. (The exception for string is made so that

| unpickling can be done without copying memory.)

|

| Parameters

| ----------

| write : bool, optional

| Describes whether or not `a` can be written to.

| align : bool, optional

| Describes whether or not `a` is aligned properly for its type.

| uic : bool, optional

| Describes whether or not `a` is a copy of another "base" array.

|

| Notes

| -----

| Array flags provide information about how the memory area used

| for the array is to be interpreted. There are 7 Boolean flags

| in use, only four of which can be changed by the user:

| WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED.

|

| WRITEABLE (W) the data area can be written to;

|

| ALIGNED (A) the data and strides are aligned appropriately for the hardware

| (as determined by the compiler);

|

| UPDATEIFCOPY (U) (deprecated), replaced by WRITEBACKIFCOPY;

|

| WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced

| by .base). When the C-API function PyArray\_ResolveWritebackIfCopy is

| called, the base array will be updated with the contents of this array.

|

| All flags can be accessed using the single (upper case) letter as well

| as the full name.

|

| Examples

| --------

| >>> y = np.array([[3, 1, 7],

| ... [2, 0, 0],

| ... [8, 5, 9]])

| >>> y

| array([[3, 1, 7],

| [2, 0, 0],

| [8, 5, 9]])

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : True

| ALIGNED : True

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(write=0, align=0)

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : False

| ALIGNED : False

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(uic=1)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: cannot set WRITEBACKIFCOPY flag to True

|

| sort(...)

| a.sort(axis=-1, kind=None, order=None)

|

| Sort an array in-place. Refer to `numpy.sort` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

| Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

| and 'mergesort' use timsort under the covers and, in general, the

| actual implementation will vary with datatype. The 'mergesort' option

| is retained for backwards compatibility.

|

| .. versionchanged:: 1.15.0.

| The 'stable' option was added.

|

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.sort : Return a sorted copy of an array.

| argsort : Indirect sort.

| lexsort : Indirect stable sort on multiple keys.

| searchsorted : Find elements in sorted array.

| partition: Partial sort.

|

| Notes

| -----

| See `numpy.sort` for notes on the different sorting algorithms.

|

| Examples

| --------

| >>> a = np.array([[1,4], [3,1]])

| >>> a.sort(axis=1)

| >>> a

| array([[1, 4],

| [1, 3]])

| >>> a.sort(axis=0)

| >>> a

| array([[1, 3],

| [1, 4]])

|

| Use the `order` keyword to specify a field to use when sorting a

| structured array:

|

| >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])

| >>> a.sort(order='y')

| >>> a

| array([(b'c', 1), (b'a', 2)],

| dtype=[('x', 'S1'), ('y', '<i8')])

|

| squeeze(...)

| a.squeeze(axis=None)

|

| Remove single-dimensional entries from the shape of `a`.

|

| Refer to `numpy.squeeze` for full documentation.

|

| See Also

| --------

| numpy.squeeze : equivalent function

|

| std(...)

| a.std(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the standard deviation of the array elements along given axis.

|

| Refer to `numpy.std` for full documentation.

|

| See Also

| --------

| numpy.std : equivalent function

|

| sum(...)

| a.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True)

|

| Return the sum of the array elements over the given axis.

|

| Refer to `numpy.sum` for full documentation.

|

| See Also

| --------

| numpy.sum : equivalent function

|

| swapaxes(...)

| a.swapaxes(axis1, axis2)

|

| Return a view of the array with `axis1` and `axis2` interchanged.

|

| Refer to `numpy.swapaxes` for full documentation.

|

| See Also

| --------

| numpy.swapaxes : equivalent function

|

| take(...)

| a.take(indices, axis=None, out=None, mode='raise')

|

| Return an array formed from the elements of `a` at the given indices.

|

| Refer to `numpy.take` for full documentation.

|

| See Also

| --------

| numpy.take : equivalent function

|

| tobytes(...)

| a.tobytes(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| .. versionadded:: 1.9.0

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| tofile(...)

| a.tofile(fid, sep="", format="%s")

|

| Write array to a file as text or binary (default).

|

| Data is always written in 'C' order, independent of the order of `a`.

| The data produced by this method can be recovered using the function

| fromfile().

|

| Parameters

| ----------

| fid : file or str or Path

| An open file object, or a string containing a filename.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| sep : str

| Separator between array items for text output.

| If "" (empty), a binary file is written, equivalent to

| ``file.write(a.tobytes())``.

| format : str

| Format string for text file output.

| Each entry in the array is formatted to text by first converting

| it to the closest Python type, and then using "format" % item.

|

| Notes

| -----

| This is a convenience function for quick storage of array data.

| Information on endianness and precision is lost, so this method is not a

| good choice for files intended to archive data or transport data between

| machines with different endianness. Some of these problems can be overcome

| by outputting the data as text files, at the expense of speed and file

| size.

|

| When fid is a file object, array contents are directly written to the

| file, bypassing the file object's ``write`` method. As a result, tofile

| cannot be used with files objects supporting compression (e.g., GzipFile)

| or file-like objects that do not support ``fileno()`` (e.g., BytesIO).

|

| tolist(...)

| a.tolist()

|

| Return the array as an ``a.ndim``-levels deep nested list of Python scalars.

|

| Return a copy of the array data as a (nested) Python list.

| Data items are converted to the nearest compatible builtin Python type, via

| the `~numpy.ndarray.item` function.

|

| If ``a.ndim`` is 0, then since the depth of the nested list is 0, it will

| not be a list at all, but a simple Python scalar.

|

| Parameters

| ----------

| none

|

| Returns

| -------

| y : object, or list of object, or list of list of object, or ...

| The possibly nested list of array elements.

|

| Notes

| -----

| The array may be recreated via ``a = np.array(a.tolist())``, although this

| may sometimes lose precision.

|

| Examples

| --------

| For a 1D array, ``a.tolist()`` is almost the same as ``list(a)``:

|

| >>> a = np.array([1, 2])

| >>> list(a)

| [1, 2]

| >>> a.tolist()

| [1, 2]

|

| However, for a 2D array, ``tolist`` applies recursively:

|

| >>> a = np.array([[1, 2], [3, 4]])

| >>> list(a)

| [array([1, 2]), array([3, 4])]

| >>> a.tolist()

| [[1, 2], [3, 4]]

|

| The base case for this recursion is a 0D array:

|

| >>> a = np.array(1)

| >>> list(a)

| Traceback (most recent call last):

| ...

| TypeError: iteration over a 0-d array

| >>> a.tolist()

| 1

|

| tostring(...)

| a.tostring(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| trace(...)

| a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

|

| Return the sum along diagonals of the array.

|

| Refer to `numpy.trace` for full documentation.

|

| See Also

| --------

| numpy.trace : equivalent function

|

| transpose(...)

| a.transpose(\*axes)

|

| Returns a view of the array with axes transposed.

|

| For a 1-D array this has no effect, as a transposed vector is simply the

| same vector. To convert a 1-D array into a 2D column vector, an additional

| dimension must be added. `np.atleast2d(a).T` achieves this, as does

| `a[:, np.newaxis]`.

| For a 2-D array, this is a standard matrix transpose.

| For an n-D array, if axes are given, their order indicates how the

| axes are permuted (see Examples). If axes are not provided and

| ``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then

| ``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

|

| Parameters

| ----------

| axes : None, tuple of ints, or `n` ints

|

| \* None or no argument: reverses the order of the axes.

|

| \* tuple of ints: `i` in the `j`-th place in the tuple means `a`'s

| `i`-th axis becomes `a.transpose()`'s `j`-th axis.

|

| \* `n` ints: same as an n-tuple of the same ints (this form is

| intended simply as a "convenience" alternative to the tuple form)

|

| Returns

| -------

| out : ndarray

| View of `a`, with axes suitably permuted.

|

| See Also

| --------

| ndarray.T : Array property returning the array transposed.

| ndarray.reshape : Give a new shape to an array without changing its data.

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> a

| array([[1, 2],

| [3, 4]])

| >>> a.transpose()

| array([[1, 3],

| [2, 4]])

| >>> a.transpose((1, 0))

| array([[1, 3],

| [2, 4]])

| >>> a.transpose(1, 0)

| array([[1, 3],

| [2, 4]])

|

| var(...)

| a.var(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the variance of the array elements, along given axis.

|

| Refer to `numpy.var` for full documentation.

|

| See Also

| --------

| numpy.var : equivalent function

|

| view(...)

| a.view(dtype=None, type=None)

|

| New view of array with the same data.

|

| Parameters

| ----------

| dtype : data-type or ndarray sub-class, optional

| Data-type descriptor of the returned view, e.g., float32 or int16. The

| default, None, results in the view having the same data-type as `a`.

| This argument can also be specified as an ndarray sub-class, which

| then specifies the type of the returned object (this is equivalent to

| setting the ``type`` parameter).

| type : Python type, optional

| Type of the returned view, e.g., ndarray or matrix. Again, the

| default None results in type preservation.

|

| Notes

| -----

| ``a.view()`` is used two different ways:

|

| ``a.view(some\_dtype)`` or ``a.view(dtype=some\_dtype)`` constructs a view

| of the array's memory with a different data-type. This can cause a

| reinterpretation of the bytes of memory.

|

| ``a.view(ndarray\_subclass)`` or ``a.view(type=ndarray\_subclass)`` just

| returns an instance of `ndarray\_subclass` that looks at the same array

| (same shape, dtype, etc.) This does not cause a reinterpretation of the

| memory.

|

| For ``a.view(some\_dtype)``, if ``some\_dtype`` has a different number of

| bytes per entry than the previous dtype (for example, converting a

| regular array to a structured array), then the behavior of the view

| cannot be predicted just from the superficial appearance of ``a`` (shown

| by ``print(a)``). It also depends on exactly how ``a`` is stored in

| memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus

| defined as a slice or transpose, etc., the view may give different

| results.

|

|

| Examples

| --------

| >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

|

| Viewing array data using a different type and dtype:

|

| >>> y = x.view(dtype=np.int16, type=np.matrix)

| >>> y

| matrix([[513]], dtype=int16)

| >>> print(type(y))

| <class 'numpy.matrix'>

|

| Creating a view on a structured array so it can be used in calculations

|

| >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])

| >>> xv = x.view(dtype=np.int8).reshape(-1,2)

| >>> xv

| array([[1, 2],

| [3, 4]], dtype=int8)

| >>> xv.mean(0)

| array([2., 3.])

|

| Making changes to the view changes the underlying array

|

| >>> xv[0,1] = 20

| >>> x

| array([(1, 20), (3, 4)], dtype=[('a', 'i1'), ('b', 'i1')])

|

| Using a view to convert an array to a recarray:

|

| >>> z = x.view(np.recarray)

| >>> z.a

| array([1, 3], dtype=int8)

|

| Views share data:

|

| >>> x[0] = (9, 10)

| >>> z[0]

| (9, 10)

|

| Views that change the dtype size (bytes per entry) should normally be

| avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

|

| >>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)

| >>> y = x[:, 0:2]

| >>> y

| array([[1, 2],

| [4, 5]], dtype=int16)

| >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])

| Traceback (most recent call last):

| ...

| ValueError: To change to a dtype of a different size, the array must be C-contiguous

| >>> z = y.copy()

| >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])

| array([[(1, 2)],

| [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])

|

| ----------------------------------------------------------------------

| Data descriptors inherited from ndarray:

|

| T

| The transposed array.

|

| Same as ``self.transpose()``.

|

| Examples

| --------

| >>> x = np.array([[1.,2.],[3.,4.]])

| >>> x

| array([[ 1., 2.],

| [ 3., 4.]])

| >>> x.T

| array([[ 1., 3.],

| [ 2., 4.]])

| >>> x = np.array([1.,2.,3.,4.])

| >>> x

| array([ 1., 2., 3., 4.])

| >>> x.T

| array([ 1., 2., 3., 4.])

|

| See Also

| --------

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side.

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: C-struct side.

|

| base

| Base object if memory is from some other object.

|

| Examples

| --------

| The base of an array that owns its memory is None:

|

| >>> x = np.array([1,2,3,4])

| >>> x.base is None

| True

|

| Slicing creates a view, whose memory is shared with x:

|

| >>> y = x[2:]

| >>> y.base is x

| True

|

| ctypes

| An object to simplify the interaction of the array with the ctypes

| module.

|

| This attribute creates an object that makes it easier to use arrays

| when calling shared libraries with the ctypes module. The returned

| object has, among others, data, shape, and strides attributes (see

| Notes below) which themselves return ctypes objects that can be used

| as arguments to a shared library.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| c : Python object

| Possessing attributes data, shape, strides, etc.

|

| See Also

| --------

| numpy.ctypeslib

|

| Notes

| -----

| Below are the public attributes of this object which were documented

| in "Guide to NumPy" (we have omitted undocumented public attributes,

| as well as documented private attributes):

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.data

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.shape

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.strides

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.data\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.shape\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.strides\_as

| :noindex:

|

| If the ctypes module is not available, then the ctypes attribute

| of array objects still returns something useful, but ctypes objects

| are not returned and errors may be raised instead. In particular,

| the object will still have the ``as\_parameter`` attribute which will

| return an integer equal to the data attribute.

|

| Examples

| --------

| >>> import ctypes

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.ctypes.data

| 30439712

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long))

| <ctypes.LP\_c\_long object at 0x01F01300>

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long)).contents

| c\_long(0)

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_longlong)).contents

| c\_longlong(4294967296L)

| >>> x.ctypes.shape

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FFD580>

| >>> x.ctypes.shape\_as(ctypes.c\_long)

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides\_as(ctypes.c\_longlong)

| <numpy.core.\_internal.c\_longlong\_Array\_2 object at 0x01F01300>

|

| data

| Python buffer object pointing to the start of the array's data.

|

| dtype

| Data-type of the array's elements.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| d : numpy dtype object

|

| See Also

| --------

| numpy.dtype

|

| Examples

| --------

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.dtype

| dtype('int32')

| >>> type(x.dtype)

| <type 'numpy.dtype'>

|

| flags

| Information about the memory layout of the array.

|

| Attributes

| ----------

| C\_CONTIGUOUS (C)

| The data is in a single, C-style contiguous segment.

| F\_CONTIGUOUS (F)

| The data is in a single, Fortran-style contiguous segment.

| OWNDATA (O)

| The array owns the memory it uses or borrows it from another object.

| WRITEABLE (W)

| The data area can be written to. Setting this to False locks

| the data, making it read-only. A view (slice, etc.) inherits WRITEABLE

| from its base array at creation time, but a view of a writeable

| array may be subsequently locked while the base array remains writeable.

| (The opposite is not true, in that a view of a locked array may not

| be made writeable. However, currently, locking a base object does not

| lock any views that already reference it, so under that circumstance it

| is possible to alter the contents of a locked array via a previously

| created writeable view onto it.) Attempting to change a non-writeable

| array raises a RuntimeError exception.

| ALIGNED (A)

| The data and all elements are aligned appropriately for the hardware.

| WRITEBACKIFCOPY (X)

| This array is a copy of some other array. The C-API function

| PyArray\_ResolveWritebackIfCopy must be called before deallocating

| to the base array will be updated with the contents of this array.

| UPDATEIFCOPY (U)

| (Deprecated, use WRITEBACKIFCOPY) This array is a copy of some other array.

| When this array is

| deallocated, the base array will be updated with the contents of

| this array.

| FNC

| F\_CONTIGUOUS and not C\_CONTIGUOUS.

| FORC

| F\_CONTIGUOUS or C\_CONTIGUOUS (one-segment test).

| BEHAVED (B)

| ALIGNED and WRITEABLE.

| CARRAY (CA)

| BEHAVED and C\_CONTIGUOUS.

| FARRAY (FA)

| BEHAVED and F\_CONTIGUOUS and not C\_CONTIGUOUS.

|

| Notes

| -----

| The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),

| or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag

| names are only supported in dictionary access.

|

| Only the WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be

| changed by the user, via direct assignment to the attribute or dictionary

| entry, or by calling `ndarray.setflags`.

|

| The array flags cannot be set arbitrarily:

|

| - UPDATEIFCOPY can only be set ``False``.

| - WRITEBACKIFCOPY can only be set ``False``.

| - ALIGNED can only be set ``True`` if the data is truly aligned.

| - WRITEABLE can only be set ``True`` if the array owns its own memory

| or the ultimate owner of the memory exposes a writeable buffer

| interface or is a string.

|

| Arrays can be both C-style and Fortran-style contiguous simultaneously.

| This is clear for 1-dimensional arrays, but can also be true for higher

| dimensional arrays.

|

| Even for contiguous arrays a stride for a given dimension

| ``arr.strides[dim]`` may be \*arbitrary\* if ``arr.shape[dim] == 1``

| or the array has no elements.

| It does \*not\* generally hold that ``self.strides[-1] == self.itemsize``

| for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for

| Fortran-style contiguous arrays is true.

|

| flat

| A 1-D iterator over the array.

|

| This is a `numpy.flatiter` instance, which acts similarly to, but is not

| a subclass of, Python's built-in iterator object.

|

| See Also

| --------

| flatten : Return a copy of the array collapsed into one dimension.

|

| flatiter

|

| Examples

| --------

| >>> x = np.arange(1, 7).reshape(2, 3)

| >>> x

| array([[1, 2, 3],

| [4, 5, 6]])

| >>> x.flat[3]

| 4

| >>> x.T

| array([[1, 4],

| [2, 5],

| [3, 6]])

| >>> x.T.flat[3]

| 5

| >>> type(x.flat)

| <class 'numpy.flatiter'>

|

| An assignment example:

|

| >>> x.flat = 3; x

| array([[3, 3, 3],

| [3, 3, 3]])

| >>> x.flat[[1,4]] = 1; x

| array([[3, 1, 3],

| [3, 1, 3]])

|

| imag

| The imaginary part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.imag

| array([ 0. , 0.70710678])

| >>> x.imag.dtype

| dtype('float64')

|

| itemsize

| Length of one array element in bytes.

|

| Examples

| --------

| >>> x = np.array([1,2,3], dtype=np.float64)

| >>> x.itemsize

| 8

| >>> x = np.array([1,2,3], dtype=np.complex128)

| >>> x.itemsize

| 16

|

| nbytes

| Total bytes consumed by the elements of the array.

|

| Notes

| -----

| Does not include memory consumed by non-element attributes of the

| array object.

|

| Examples

| --------

| >>> x = np.zeros((3,5,2), dtype=np.complex128)

| >>> x.nbytes

| 480

| >>> np.prod(x.shape) \* x.itemsize

| 480

|

| ndim

| Number of array dimensions.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> x.ndim

| 1

| >>> y = np.zeros((2, 3, 4))

| >>> y.ndim

| 3

|

| real

| The real part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.real

| array([ 1. , 0.70710678])

| >>> x.real.dtype

| dtype('float64')

|

| See Also

| --------

| numpy.real : equivalent function

|

| shape

| Tuple of array dimensions.

|

| The shape property is usually used to get the current shape of an array,

| but may also be used to reshape the array in-place by assigning a tuple of

| array dimensions to it. As with `numpy.reshape`, one of the new shape

| dimensions can be -1, in which case its value is inferred from the size of

| the array and the remaining dimensions. Reshaping an array in-place will

| fail if a copy is required.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3, 4])

| >>> x.shape

| (4,)

| >>> y = np.zeros((2, 3, 4))

| >>> y.shape

| (2, 3, 4)

| >>> y.shape = (3, 8)

| >>> y

| array([[ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.]])

| >>> y.shape = (3, 6)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: total size of new array must be unchanged

| >>> np.zeros((4,2))[::2].shape = (-1,)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| AttributeError: incompatible shape for a non-contiguous array

|

| See Also

| --------

| numpy.reshape : similar function

| ndarray.reshape : similar method

|

| size

| Number of elements in the array.

|

| Equal to ``np.prod(a.shape)``, i.e., the product of the array's

| dimensions.

|

| Notes

| -----

| `a.size` returns a standard arbitrary precision Python integer. This

| may not be the case with other methods of obtaining the same value

| (like the suggested ``np.prod(a.shape)``, which returns an instance

| of ``np.int\_``), and may be relevant if the value is used further in

| calculations that may overflow a fixed size integer type.

|

| Examples

| --------

| >>> x = np.zeros((3, 5, 2), dtype=np.complex128)

| >>> x.size

| 30

| >>> np.prod(x.shape)

| 30

|

| strides

| Tuple of bytes to step in each dimension when traversing an array.

|

| The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`

| is::

|

| offset = sum(np.array(i) \* a.strides)

|

| A more detailed explanation of strides can be found in the

| "ndarray.rst" file in the NumPy reference guide.

|

| Notes

| -----

| Imagine an array of 32-bit integers (each 4 bytes)::

|

| x = np.array([[0, 1, 2, 3, 4],

| [5, 6, 7, 8, 9]], dtype=np.int32)

|

| This array is stored in memory as 40 bytes, one after the other

| (known as a contiguous block of memory). The strides of an array tell

| us how many bytes we have to skip in memory to move to the next position

| along a certain axis. For example, we have to skip 4 bytes (1 value) to

| move to the next column, but 20 bytes (5 values) to get to the same

| position in the next row. As such, the strides for the array `x` will be

| ``(20, 4)``.

|

| See Also

| --------

| numpy.lib.stride\_tricks.as\_strided

|

| Examples

| --------

| >>> y = np.reshape(np.arange(2\*3\*4), (2,3,4))

| >>> y

| array([[[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]],

| [[12, 13, 14, 15],

| [16, 17, 18, 19],

| [20, 21, 22, 23]]])

| >>> y.strides

| (48, 16, 4)

| >>> y[1,1,1]

| 17

| >>> offset=sum(y.strides \* np.array((1,1,1)))

| >>> offset/y.itemsize

| 17

|

| >>> x = np.reshape(np.arange(5\*6\*7\*8), (5,6,7,8)).transpose(2,3,1,0)

| >>> x.strides

| (32, 4, 224, 1344)

| >>> i = np.array([3,5,2,2])

| >>> offset = sum(i \* x.strides)

| >>> x[3,5,2,2]

| 813

| >>> offset / x.itemsize

| 813

clongdouble = class complex128(complexfloating)

| Complex number type composed of two extended-precision floating-point

| numbers.

| Character code: ``'G'``.

| Canonical name: ``np.clongdouble``.

| Alias: ``np.clongfloat``.

| Alias: ``np.longcomplex``.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

clongfloat = class complex128(complexfloating)

| Complex number type composed of two extended-precision floating-point

| numbers.

| Character code: ``'G'``.

| Canonical name: ``np.clongdouble``.

| Alias: ``np.clongfloat``.

| Alias: ``np.longcomplex``.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class complex128(complexfloating, builtins.complex)

| complex128(real=0, imag=0)

|

| Complex number type composed of two double-precision floating-point

| numbers, compatible with Python `complex`.

| Character code: ``'D'``.

| Canonical name: ``np.cdouble``.

| Alias: ``np.cfloat``.

| Alias: ``np.complex\_``.

| Alias \*on this platform\*: ``np.complex128``: Complex number type composed of 2 64-bit-precision floating-point numbers.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.complex

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.complex:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(...)

class complex64(complexfloating)

| Complex number type composed of two single-precision floating-point

| numbers.

| Character code: ``'F'``.

| Canonical name: ``np.csingle``.

| Alias: ``np.singlecomplex``.

| Alias \*on this platform\*: ``np.complex64``: Complex number type composed of 2 32-bit-precision floating-point numbers.

|

| Method resolution order:

| complex64

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

complex\_ = class complex128(complexfloating, builtins.complex)

| complex\_(real=0, imag=0)

|

| Complex number type composed of two double-precision floating-point

| numbers, compatible with Python `complex`.

| Character code: ``'D'``.

| Canonical name: ``np.cdouble``.

| Alias: ``np.cfloat``.

| Alias: ``np.complex\_``.

| Alias \*on this platform\*: ``np.complex128``: Complex number type composed of 2 64-bit-precision floating-point numbers.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.complex

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.complex:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(...)

class complexfloating(inexact)

| Abstract base class of all complex number scalar types that are made up of

| floating-point numbers.

|

| Method resolution order:

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

csingle = class complex64(complexfloating)

| Complex number type composed of two single-precision floating-point

| numbers.

| Character code: ``'F'``.

| Canonical name: ``np.csingle``.

| Alias: ``np.singlecomplex``.

| Alias \*on this platform\*: ``np.complex64``: Complex number type composed of 2 32-bit-precision floating-point numbers.

|

| Method resolution order:

| complex64

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class datetime64(generic)

| Base class for numpy scalar types.

|

| Class from which most (all?) numpy scalar types are derived. For

| consistency, exposes the same API as `ndarray`, despite many

| consequent attributes being either "get-only," or completely irrelevant.

| This is the class from which it is strongly suggested users should derive

| custom scalar types.

|

| Method resolution order:

| datetime64

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

double = class float64(floating, builtins.float)

| double(x=0, /)

|

| Double-precision floating-point number type, compatible with Python `float`

| and C ``double``.

| Character code: ``'d'``.

| Canonical name: ``np.double``.

| Alias: ``np.float\_``.

| Alias \*on this platform\*: ``np.float64``: 64-bit precision floating-point number type: sign bit, 11 bits exponent, 52 bits mantissa.

|

| Method resolution order:

| float64

| floating

| inexact

| number

| generic

| builtins.float

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| double.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.double(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.double(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.double(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.float:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(self, /)

|

| \_\_trunc\_\_(self, /)

| Return the Integral closest to x between 0 and x.

|

| hex(self, /)

| Return a hexadecimal representation of a floating-point number.

|

| >>> (-0.1).hex()

| '-0x1.999999999999ap-4'

| >>> 3.14159.hex()

| '0x1.921f9f01b866ep+1'

|

| is\_integer(self, /)

| Return True if the float is an integer.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.float:

|

| \_\_getformat\_\_(typestr, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

|

| It exists mainly to be used in Python's test suite.

|

| This function returns whichever of 'unknown', 'IEEE, big-endian' or 'IEEE,

| little-endian' best describes the format of floating point numbers used by the

| C type named by typestr.

|

| \_\_set\_format\_\_(typestr, fmt, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

| fmt

| Must be one of 'unknown', 'IEEE, big-endian' or 'IEEE, little-endian',

| and in addition can only be one of the latter two if it appears to

| match the underlying C reality.

|

| It exists mainly to be used in Python's test suite.

|

| Override the automatic determination of C-level floating point type.

| This affects how floats are converted to and from binary strings.

|

| fromhex(string, /) from builtins.type

| Create a floating-point number from a hexadecimal string.

|

| >>> float.fromhex('0x1.ffffp10')

| 2047.984375

| >>> float.fromhex('-0x1p-1074')

| -5e-324

class dtype(builtins.object)

| dtype(obj, align=False, copy=False)

|

| Create a data type object.

|

| A numpy array is homogeneous, and contains elements described by a

| dtype object. A dtype object can be constructed from different

| combinations of fundamental numeric types.

|

| Parameters

| ----------

| obj

| Object to be converted to a data type object.

| align : bool, optional

| Add padding to the fields to match what a C compiler would output

| for a similar C-struct. Can be ``True`` only if `obj` is a dictionary

| or a comma-separated string. If a struct dtype is being created,

| this also sets a sticky alignment flag ``isalignedstruct``.

| copy : bool, optional

| Make a new copy of the data-type object. If ``False``, the result

| may just be a reference to a built-in data-type object.

|

| See also

| --------

| result\_type

|

| Examples

| --------

| Using array-scalar type:

|

| >>> np.dtype(np.int16)

| dtype('int16')

|

| Structured type, one field name 'f1', containing int16:

|

| >>> np.dtype([('f1', np.int16)])

| dtype([('f1', '<i2')])

|

| Structured type, one field named 'f1', in itself containing a structured

| type with one field:

|

| >>> np.dtype([('f1', [('f1', np.int16)])])

| dtype([('f1', [('f1', '<i2')])])

|

| Structured type, two fields: the first field contains an unsigned int, the

| second an int32:

|

| >>> np.dtype([('f1', np.uint64), ('f2', np.int32)])

| dtype([('f1', '<u8'), ('f2', '<i4')])

|

| Using array-protocol type strings:

|

| >>> np.dtype([('a','f8'),('b','S10')])

| dtype([('a', '<f8'), ('b', 'S10')])

|

| Using comma-separated field formats. The shape is (2,3):

|

| >>> np.dtype("i4, (2,3)f8")

| dtype([('f0', '<i4'), ('f1', '<f8', (2, 3))])

|

| Using tuples. ``int`` is a fixed type, 3 the field's shape. ``void``

| is a flexible type, here of size 10:

|

| >>> np.dtype([('hello',(np.int64,3)),('world',np.void,10)])

| dtype([('hello', '<i8', (3,)), ('world', 'V10')])

|

| Subdivide ``int16`` into 2 ``int8``'s, called x and y. 0 and 1 are

| the offsets in bytes:

|

| >>> np.dtype((np.int16, {'x':(np.int8,0), 'y':(np.int8,1)}))

| dtype((numpy.int16, [('x', 'i1'), ('y', 'i1')]))

|

| Using dictionaries. Two fields named 'gender' and 'age':

|

| >>> np.dtype({'names':['gender','age'], 'formats':['S1',np.uint8]})

| dtype([('gender', 'S1'), ('age', 'u1')])

|

| Offsets in bytes, here 0 and 25:

|

| >>> np.dtype({'surname':('S25',0),'age':(np.uint8,25)})

| dtype([('surname', 'S25'), ('age', 'u1')])

|

| Methods defined here:

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_setstate\_\_(...)

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new dtype with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. The default value ('S') results in swapping the current

| byte order. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The code does a case-insensitive check on the first letter of

| `new\_order` for these alternatives. For example, any of '>'

| or 'B' or 'b' or 'brian' are valid to specify big-endian.

|

| Returns

| -------

| new\_dtype : dtype

| New dtype object with the given change to the byte order.

|

| Notes

| -----

| Changes are also made in all fields and sub-arrays of the data type.

|

| Examples

| --------

| >>> import sys

| >>> sys\_is\_le = sys.byteorder == 'little'

| >>> native\_code = sys\_is\_le and '<' or '>'

| >>> swapped\_code = sys\_is\_le and '>' or '<'

| >>> native\_dt = np.dtype(native\_code+'i2')

| >>> swapped\_dt = np.dtype(swapped\_code+'i2')

| >>> native\_dt.newbyteorder('S') == swapped\_dt

| True

| >>> native\_dt.newbyteorder() == swapped\_dt

| True

| >>> native\_dt == swapped\_dt.newbyteorder('S')

| True

| >>> native\_dt == swapped\_dt.newbyteorder('=')

| True

| >>> native\_dt == swapped\_dt.newbyteorder('N')

| True

| >>> native\_dt == native\_dt.newbyteorder('|')

| True

| >>> np.dtype('<i2') == native\_dt.newbyteorder('<')

| True

| >>> np.dtype('<i2') == native\_dt.newbyteorder('L')

| True

| >>> np.dtype('>i2') == native\_dt.newbyteorder('>')

| True

| >>> np.dtype('>i2') == native\_dt.newbyteorder('B')

| True

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| alignment

| The required alignment (bytes) of this data-type according to the compiler.

|

| More information is available in the C-API section of the manual.

|

| Examples

| --------

|

| >>> x = np.dtype('i4')

| >>> x.alignment

| 4

|

| >>> x = np.dtype(float)

| >>> x.alignment

| 8

|

| base

| Returns dtype for the base element of the subarrays,

| regardless of their dimension or shape.

|

| See Also

| --------

| dtype.subdtype

|

| Examples

| --------

| >>> x = numpy.dtype('8f')

| >>> x.base

| dtype('float32')

|

| >>> x = numpy.dtype('i2')

| >>> x.base

| dtype('int16')

|

| byteorder

| A character indicating the byte-order of this data-type object.

|

| One of:

|

| === ==============

| '=' native

| '<' little-endian

| '>' big-endian

| '|' not applicable

| === ==============

|

| All built-in data-type objects have byteorder either '=' or '|'.

|

| Examples

| --------

|

| >>> dt = np.dtype('i2')

| >>> dt.byteorder

| '='

| >>> # endian is not relevant for 8 bit numbers

| >>> np.dtype('i1').byteorder

| '|'

| >>> # or ASCII strings

| >>> np.dtype('S2').byteorder

| '|'

| >>> # Even if specific code is given, and it is native

| >>> # '=' is the byteorder

| >>> import sys

| >>> sys\_is\_le = sys.byteorder == 'little'

| >>> native\_code = sys\_is\_le and '<' or '>'

| >>> swapped\_code = sys\_is\_le and '>' or '<'

| >>> dt = np.dtype(native\_code + 'i2')

| >>> dt.byteorder

| '='

| >>> # Swapped code shows up as itself

| >>> dt = np.dtype(swapped\_code + 'i2')

| >>> dt.byteorder == swapped\_code

| True

|

| char

| A unique character code for each of the 21 different built-in types.

|

| Examples

| --------

|

| >>> x = np.dtype(float)

| >>> x.char

| 'd'

|

| descr

| `\_\_array\_interface\_\_` description of the data-type.

|

| The format is that required by the 'descr' key in the

| `\_\_array\_interface\_\_` attribute.

|

| Warning: This attribute exists specifically for `\_\_array\_interface\_\_`,

| and is not a datatype description compatible with `np.dtype`.

|

| Examples

| --------

|

| >>> x = np.dtype(float)

| >>> x.descr

| [('', '<f8')]

|

| >>> dt = np.dtype([('name', np.str\_, 16), ('grades', np.float64, (2,))])

| >>> dt.descr

| [('name', '<U16'), ('grades', '<f8', (2,))]

|

| fields

| Dictionary of named fields defined for this data type, or ``None``.

|

| The dictionary is indexed by keys that are the names of the fields.

| Each entry in the dictionary is a tuple fully describing the field::

|

| (dtype, offset[, title])

|

| Offset is limited to C int, which is signed and usually 32 bits.

| If present, the optional title can be any object (if it is a string

| or unicode then it will also be a key in the fields dictionary,

| otherwise it's meta-data). Notice also that the first two elements

| of the tuple can be passed directly as arguments to the ``ndarray.getfield``

| and ``ndarray.setfield`` methods.

|

| See Also

| --------

| ndarray.getfield, ndarray.setfield

|

| Examples

| --------

| >>> dt = np.dtype([('name', np.str\_, 16), ('grades', np.float64, (2,))])

| >>> print(dt.fields)

| {'grades': (dtype(('float64',(2,))), 16), 'name': (dtype('|S16'), 0)}

|

| flags

| Bit-flags describing how this data type is to be interpreted.

|

| Bit-masks are in `numpy.core.multiarray` as the constants

| `ITEM\_HASOBJECT`, `LIST\_PICKLE`, `ITEM\_IS\_POINTER`, `NEEDS\_INIT`,

| `NEEDS\_PYAPI`, `USE\_GETITEM`, `USE\_SETITEM`. A full explanation

| of these flags is in C-API documentation; they are largely useful

| for user-defined data-types.

|

| The following example demonstrates that operations on this particular

| dtype requires Python C-API.

|

| Examples

| --------

|

| >>> x = np.dtype([('a', np.int32, 8), ('b', np.float64, 6)])

| >>> x.flags

| 16

| >>> np.core.multiarray.NEEDS\_PYAPI

| 16

|

| hasobject

| Boolean indicating whether this dtype contains any reference-counted

| objects in any fields or sub-dtypes.

|

| Recall that what is actually in the ndarray memory representing

| the Python object is the memory address of that object (a pointer).

| Special handling may be required, and this attribute is useful for

| distinguishing data types that may contain arbitrary Python objects

| and data-types that won't.

|

| isalignedstruct

| Boolean indicating whether the dtype is a struct which maintains

| field alignment. This flag is sticky, so when combining multiple

| structs together, it is preserved and produces new dtypes which

| are also aligned.

|

| isbuiltin

| Integer indicating how this dtype relates to the built-in dtypes.

|

| Read-only.

|

| = ========================================================================

| 0 if this is a structured array type, with fields

| 1 if this is a dtype compiled into numpy (such as ints, floats etc)

| 2 if the dtype is for a user-defined numpy type

| A user-defined type uses the numpy C-API machinery to extend

| numpy to handle a new array type. See

| :ref:`user.user-defined-data-types` in the NumPy manual.

| = ========================================================================

|

| Examples

| --------

| >>> dt = np.dtype('i2')

| >>> dt.isbuiltin

| 1

| >>> dt = np.dtype('f8')

| >>> dt.isbuiltin

| 1

| >>> dt = np.dtype([('field1', 'f8')])

| >>> dt.isbuiltin

| 0

|

| isnative

| Boolean indicating whether the byte order of this dtype is native

| to the platform.

|

| itemsize

| The element size of this data-type object.

|

| For 18 of the 21 types this number is fixed by the data-type.

| For the flexible data-types, this number can be anything.

|

| Examples

| --------

|

| >>> arr = np.array([[1, 2], [3, 4]])

| >>> arr.dtype

| dtype('int64')

| >>> arr.itemsize

| 8

|

| >>> dt = np.dtype([('name', np.str\_, 16), ('grades', np.float64, (2,))])

| >>> dt.itemsize

| 80

|

| kind

| A character code (one of 'biufcmMOSUV') identifying the general kind of data.

|

| = ======================

| b boolean

| i signed integer

| u unsigned integer

| f floating-point

| c complex floating-point

| m timedelta

| M datetime

| O object

| S (byte-)string

| U Unicode

| V void

| = ======================

|

| Examples

| --------

|

| >>> dt = np.dtype('i4')

| >>> dt.kind

| 'i'

| >>> dt = np.dtype('f8')

| >>> dt.kind

| 'f'

| >>> dt = np.dtype([('field1', 'f8')])

| >>> dt.kind

| 'V'

|

| metadata

|

| name

| A bit-width name for this data-type.

|

| Un-sized flexible data-type objects do not have this attribute.

|

| Examples

| --------

|

| >>> x = np.dtype(float)

| >>> x.name

| 'float64'

| >>> x = np.dtype([('a', np.int32, 8), ('b', np.float64, 6)])

| >>> x.name

| 'void640'

|

| names

| Ordered list of field names, or ``None`` if there are no fields.

|

| The names are ordered according to increasing byte offset. This can be

| used, for example, to walk through all of the named fields in offset order.

|

| Examples

| --------

| >>> dt = np.dtype([('name', np.str\_, 16), ('grades', np.float64, (2,))])

| >>> dt.names

| ('name', 'grades')

|

| ndim

| Number of dimensions of the sub-array if this data type describes a

| sub-array, and ``0`` otherwise.

|

| .. versionadded:: 1.13.0

|

| Examples

| --------

| >>> x = np.dtype(float)

| >>> x.ndim

| 0

|

| >>> x = np.dtype((float, 8))

| >>> x.ndim

| 1

|

| >>> x = np.dtype(('i4', (3, 4)))

| >>> x.ndim

| 2

|

| num

| A unique number for each of the 21 different built-in types.

|

| These are roughly ordered from least-to-most precision.

|

| Examples

| --------

|

| >>> dt = np.dtype(str)

| >>> dt.num

| 19

|

| >>> dt = np.dtype(float)

| >>> dt.num

| 12

|

| shape

| Shape tuple of the sub-array if this data type describes a sub-array,

| and ``()`` otherwise.

|

| Examples

| --------

|

| >>> dt = np.dtype(('i4', 4))

| >>> dt.shape

| (4,)

|

| >>> dt = np.dtype(('i4', (2, 3)))

| >>> dt.shape

| (2, 3)

|

| str

| The array-protocol typestring of this data-type object.

|

| subdtype

| Tuple ``(item\_dtype, shape)`` if this `dtype` describes a sub-array, and

| None otherwise.

|

| The \*shape\* is the fixed shape of the sub-array described by this

| data type, and \*item\_dtype\* the data type of the array.

|

| If a field whose dtype object has this attribute is retrieved,

| then the extra dimensions implied by \*shape\* are tacked on to

| the end of the retrieved array.

|

| See Also

| --------

| dtype.base

|

| Examples

| --------

| >>> x = numpy.dtype('8f')

| >>> x.subdtype

| (dtype('float32'), (8,))

|

| >>> x = numpy.dtype('i2')

| >>> x.subdtype

| >>>

|

| type

| The type object used to instantiate a scalar of this data-type.

class errstate(contextlib.ContextDecorator)

| errstate(\*\*kwargs)

|

| errstate(\*\*kwargs)

|

| Context manager for floating-point error handling.

|

| Using an instance of `errstate` as a context manager allows statements in

| that context to execute with a known error handling behavior. Upon entering

| the context the error handling is set with `seterr` and `seterrcall`, and

| upon exiting it is reset to what it was before.

|

| .. versionchanged:: 1.17.0

| `errstate` is also usable as a function decorator, saving

| a level of indentation if an entire function is wrapped.

| See :py:class:`contextlib.ContextDecorator` for more information.

|

| Parameters

| ----------

| kwargs : {divide, over, under, invalid}

| Keyword arguments. The valid keywords are the possible floating-point

| exceptions. Each keyword should have a string value that defines the

| treatment for the particular error. Possible values are

| {'ignore', 'warn', 'raise', 'call', 'print', 'log'}.

|

| See Also

| --------

| seterr, geterr, seterrcall, geterrcall

|

| Notes

| -----

| For complete documentation of the types of floating-point exceptions and

| treatment options, see `seterr`.

|

| Examples

| --------

| >>> from collections import OrderedDict

| >>> olderr = np.seterr(all='ignore') # Set error handling to known state.

|

| >>> np.arange(3) / 0.

| array([nan, inf, inf])

| >>> with np.errstate(divide='warn'):

| ... np.arange(3) / 0.

| array([nan, inf, inf])

|

| >>> np.sqrt(-1)

| nan

| >>> with np.errstate(invalid='raise'):

| ... np.sqrt(-1)

| Traceback (most recent call last):

| File "<stdin>", line 2, in <module>

| FloatingPointError: invalid value encountered in sqrt

|

| Outside the context the error handling behavior has not changed:

|

| >>> OrderedDict(sorted(np.geterr().items()))

| OrderedDict([('divide', 'ignore'), ('invalid', 'ignore'), ('over', 'ignore'), ('under', 'ignore')])

|

| Method resolution order:

| errstate

| contextlib.ContextDecorator

| builtins.object

|

| Methods defined here:

|

| \_\_enter\_\_(self)

|

| \_\_exit\_\_(self, \*exc\_info)

|

| \_\_init\_\_(self, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from contextlib.ContextDecorator:

|

| \_\_call\_\_(self, func)

| Call self as a function.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from contextlib.ContextDecorator:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class finfo(builtins.object)

| finfo(dtype)

|

| finfo(dtype)

|

| Machine limits for floating point types.

|

| Attributes

| ----------

| bits : int

| The number of bits occupied by the type.

| eps : float

| The smallest representable positive number such that

| ``1.0 + eps != 1.0``. Type of `eps` is an appropriate floating

| point type.

| epsneg : floating point number of the appropriate type

| The smallest representable positive number such that

| ``1.0 - epsneg != 1.0``.

| iexp : int

| The number of bits in the exponent portion of the floating point

| representation.

| machar : MachAr

| The object which calculated these parameters and holds more

| detailed information.

| machep : int

| The exponent that yields `eps`.

| max : floating point number of the appropriate type

| The largest representable number.

| maxexp : int

| The smallest positive power of the base (2) that causes overflow.

| min : floating point number of the appropriate type

| The smallest representable number, typically ``-max``.

| minexp : int

| The most negative power of the base (2) consistent with there

| being no leading 0's in the mantissa.

| negep : int

| The exponent that yields `epsneg`.

| nexp : int

| The number of bits in the exponent including its sign and bias.

| nmant : int

| The number of bits in the mantissa.

| precision : int

| The approximate number of decimal digits to which this kind of

| float is precise.

| resolution : floating point number of the appropriate type

| The approximate decimal resolution of this type, i.e.,

| ``10\*\*-precision``.

| tiny : float

| The smallest positive usable number. Type of `tiny` is an

| appropriate floating point type.

|

| Parameters

| ----------

| dtype : float, dtype, or instance

| Kind of floating point data-type about which to get information.

|

| See Also

| --------

| MachAr : The implementation of the tests that produce this information.

| iinfo : The equivalent for integer data types.

|

| Notes

| -----

| For developers of NumPy: do not instantiate this at the module level.

| The initial calculation of these parameters is expensive and negatively

| impacts import times. These objects are cached, so calling ``finfo()``

| repeatedly inside your functions is not a problem.

|

| Methods defined here:

|

| \_\_repr\_\_(self)

| Return repr(self).

|

| \_\_str\_\_(self)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(cls, dtype)

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class flatiter(builtins.object)

| Flat iterator object to iterate over arrays.

|

| A `flatiter` iterator is returned by ``x.flat`` for any array `x`.

| It allows iterating over the array as if it were a 1-D array,

| either in a for-loop or by calling its `next` method.

|

| Iteration is done in row-major, C-style order (the last

| index varying the fastest). The iterator can also be indexed using

| basic slicing or advanced indexing.

|

| See Also

| --------

| ndarray.flat : Return a flat iterator over an array.

| ndarray.flatten : Returns a flattened copy of an array.

|

| Notes

| -----

| A `flatiter` iterator can not be constructed directly from Python code

| by calling the `flatiter` constructor.

|

| Examples

| --------

| >>> x = np.arange(6).reshape(2, 3)

| >>> fl = x.flat

| >>> type(fl)

| <class 'numpy.flatiter'>

| >>> for item in fl:

| ... print(item)

| ...

| 0

| 1

| 2

| 3

| 4

| 5

|

| >>> fl[2:4]

| array([2, 3])

|

| Methods defined here:

|

| \_\_array\_\_(...)

| \_\_array\_\_(type=None) Get array from iterator

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| copy(...)

| copy()

|

| Get a copy of the iterator as a 1-D array.

|

| Examples

| --------

| >>> x = np.arange(6).reshape(2, 3)

| >>> x

| array([[0, 1, 2],

| [3, 4, 5]])

| >>> fl = x.flat

| >>> fl.copy()

| array([0, 1, 2, 3, 4, 5])

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| base

| A reference to the array that is iterated over.

|

| Examples

| --------

| >>> x = np.arange(5)

| >>> fl = x.flat

| >>> fl.base is x

| True

|

| coords

| An N-dimensional tuple of current coordinates.

|

| Examples

| --------

| >>> x = np.arange(6).reshape(2, 3)

| >>> fl = x.flat

| >>> fl.coords

| (0, 0)

| >>> fl.next()

| 0

| >>> fl.coords

| (0, 1)

|

| index

| Current flat index into the array.

|

| Examples

| --------

| >>> x = np.arange(6).reshape(2, 3)

| >>> fl = x.flat

| >>> fl.index

| 0

| >>> fl.next()

| 0

| >>> fl.index

| 1

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_hash\_\_ = None

class flexible(generic)

| Abstract base class of all scalar types without predefined length.

| The actual size of these types depends on the specific `np.dtype`

| instantiation.

|

| Method resolution order:

| flexible

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

class float16(floating)

| Half-precision floating-point number type.

| Character code: ``'e'``.

| Canonical name: ``np.half``.

| Alias \*on this platform\*: ``np.float16``: 16-bit-precision floating-point number type: sign bit, 5 bits exponent, 10 bits mantissa.

|

| Method resolution order:

| float16

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| half.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.half(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.half(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.half(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class float32(floating)

| Single-precision floating-point number type, compatible with C ``float``.

| Character code: ``'f'``.

| Canonical name: ``np.single``.

| Alias \*on this platform\*: ``np.float32``: 32-bit-precision floating-point number type: sign bit, 8 bits exponent, 23 bits mantissa.

|

| Method resolution order:

| float32

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| single.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.single(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.single(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.single(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class float64(floating, builtins.float)

| float64(x=0, /)

|

| Double-precision floating-point number type, compatible with Python `float`

| and C ``double``.

| Character code: ``'d'``.

| Canonical name: ``np.double``.

| Alias: ``np.float\_``.

| Alias \*on this platform\*: ``np.float64``: 64-bit precision floating-point number type: sign bit, 11 bits exponent, 52 bits mantissa.

|

| Method resolution order:

| float64

| floating

| inexact

| number

| generic

| builtins.float

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| double.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.double(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.double(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.double(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.float:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(self, /)

|

| \_\_trunc\_\_(self, /)

| Return the Integral closest to x between 0 and x.

|

| hex(self, /)

| Return a hexadecimal representation of a floating-point number.

|

| >>> (-0.1).hex()

| '-0x1.999999999999ap-4'

| >>> 3.14159.hex()

| '0x1.921f9f01b866ep+1'

|

| is\_integer(self, /)

| Return True if the float is an integer.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.float:

|

| \_\_getformat\_\_(typestr, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

|

| It exists mainly to be used in Python's test suite.

|

| This function returns whichever of 'unknown', 'IEEE, big-endian' or 'IEEE,

| little-endian' best describes the format of floating point numbers used by the

| C type named by typestr.

|

| \_\_set\_format\_\_(typestr, fmt, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

| fmt

| Must be one of 'unknown', 'IEEE, big-endian' or 'IEEE, little-endian',

| and in addition can only be one of the latter two if it appears to

| match the underlying C reality.

|

| It exists mainly to be used in Python's test suite.

|

| Override the automatic determination of C-level floating point type.

| This affects how floats are converted to and from binary strings.

|

| fromhex(string, /) from builtins.type

| Create a floating-point number from a hexadecimal string.

|

| >>> float.fromhex('0x1.ffffp10')

| 2047.984375

| >>> float.fromhex('-0x1p-1074')

| -5e-324

float\_ = class float64(floating, builtins.float)

| float\_(x=0, /)

|

| Double-precision floating-point number type, compatible with Python `float`

| and C ``double``.

| Character code: ``'d'``.

| Canonical name: ``np.double``.

| Alias: ``np.float\_``.

| Alias \*on this platform\*: ``np.float64``: 64-bit precision floating-point number type: sign bit, 11 bits exponent, 52 bits mantissa.

|

| Method resolution order:

| float64

| floating

| inexact

| number

| generic

| builtins.float

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| double.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.double(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.double(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.double(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.float:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(self, /)

|

| \_\_trunc\_\_(self, /)

| Return the Integral closest to x between 0 and x.

|

| hex(self, /)

| Return a hexadecimal representation of a floating-point number.

|

| >>> (-0.1).hex()

| '-0x1.999999999999ap-4'

| >>> 3.14159.hex()

| '0x1.921f9f01b866ep+1'

|

| is\_integer(self, /)

| Return True if the float is an integer.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.float:

|

| \_\_getformat\_\_(typestr, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

|

| It exists mainly to be used in Python's test suite.

|

| This function returns whichever of 'unknown', 'IEEE, big-endian' or 'IEEE,

| little-endian' best describes the format of floating point numbers used by the

| C type named by typestr.

|

| \_\_set\_format\_\_(typestr, fmt, /) from builtins.type

| You probably don't want to use this function.

|

| typestr

| Must be 'double' or 'float'.

| fmt

| Must be one of 'unknown', 'IEEE, big-endian' or 'IEEE, little-endian',

| and in addition can only be one of the latter two if it appears to

| match the underlying C reality.

|

| It exists mainly to be used in Python's test suite.

|

| Override the automatic determination of C-level floating point type.

| This affects how floats are converted to and from binary strings.

|

| fromhex(string, /) from builtins.type

| Create a floating-point number from a hexadecimal string.

|

| >>> float.fromhex('0x1.ffffp10')

| 2047.984375

| >>> float.fromhex('-0x1p-1074')

| -5e-324

class floating(inexact)

| Abstract base class of all floating-point scalar types.

|

| Method resolution order:

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

class format\_parser(builtins.object)

| format\_parser(formats, names, titles, aligned=False, byteorder=None)

|

| Class to convert formats, names, titles description to a dtype.

|

| After constructing the format\_parser object, the dtype attribute is

| the converted data-type:

| ``dtype = format\_parser(formats, names, titles).dtype``

|

| Attributes

| ----------

| dtype : dtype

| The converted data-type.

|

| Parameters

| ----------

| formats : str or list of str

| The format description, either specified as a string with

| comma-separated format descriptions in the form ``'f8, i4, a5'``, or

| a list of format description strings in the form

| ``['f8', 'i4', 'a5']``.

| names : str or list/tuple of str

| The field names, either specified as a comma-separated string in the

| form ``'col1, col2, col3'``, or as a list or tuple of strings in the

| form ``['col1', 'col2', 'col3']``.

| An empty list can be used, in that case default field names

| ('f0', 'f1', ...) are used.

| titles : sequence

| Sequence of title strings. An empty list can be used to leave titles

| out.

| aligned : bool, optional

| If True, align the fields by padding as the C-compiler would.

| Default is False.

| byteorder : str, optional

| If specified, all the fields will be changed to the

| provided byte-order. Otherwise, the default byte-order is

| used. For all available string specifiers, see `dtype.newbyteorder`.

|

| See Also

| --------

| dtype, typename, sctype2char

|

| Examples

| --------

| >>> np.format\_parser(['<f8', '<i4', '<a5'], ['col1', 'col2', 'col3'],

| ... ['T1', 'T2', 'T3']).dtype

| dtype([(('T1', 'col1'), '<f8'), (('T2', 'col2'), '<i4'), (('T3', 'col3'), 'S5')])

|

| `names` and/or `titles` can be empty lists. If `titles` is an empty list,

| titles will simply not appear. If `names` is empty, default field names

| will be used.

|

| >>> np.format\_parser(['f8', 'i4', 'a5'], ['col1', 'col2', 'col3'],

| ... []).dtype

| dtype([('col1', '<f8'), ('col2', '<i4'), ('col3', '<S5')])

| >>> np.format\_parser(['<f8', '<i4', '<a5'], [], []).dtype

| dtype([('f0', '<f8'), ('f1', '<i4'), ('f2', 'S5')])

|

| Methods defined here:

|

| \_\_init\_\_(self, formats, names, titles, aligned=False, byteorder=None)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class generic(builtins.object)

| Base class for numpy scalar types.

|

| Class from which most (all?) numpy scalar types are derived. For

| consistency, exposes the same API as `ndarray`, despite many

| consequent attributes being either "get-only," or completely irrelevant.

| This is the class from which it is strongly suggested users should derive

| custom scalar types.

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_hash\_\_ = None

half = class float16(floating)

| Half-precision floating-point number type.

| Character code: ``'e'``.

| Canonical name: ``np.half``.

| Alias \*on this platform\*: ``np.float16``: 16-bit-precision floating-point number type: sign bit, 5 bits exponent, 10 bits mantissa.

|

| Method resolution order:

| float16

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| half.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.half(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.half(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.half(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class iinfo(builtins.object)

| iinfo(int\_type)

|

| iinfo(type)

|

| Machine limits for integer types.

|

| Attributes

| ----------

| bits : int

| The number of bits occupied by the type.

| min : int

| The smallest integer expressible by the type.

| max : int

| The largest integer expressible by the type.

|

| Parameters

| ----------

| int\_type : integer type, dtype, or instance

| The kind of integer data type to get information about.

|

| See Also

| --------

| finfo : The equivalent for floating point data types.

|

| Examples

| --------

| With types:

|

| >>> ii16 = np.iinfo(np.int16)

| >>> ii16.min

| -32768

| >>> ii16.max

| 32767

| >>> ii32 = np.iinfo(np.int32)

| >>> ii32.min

| -2147483648

| >>> ii32.max

| 2147483647

|

| With instances:

|

| >>> ii32 = np.iinfo(np.int32(10))

| >>> ii32.min

| -2147483648

| >>> ii32.max

| 2147483647

|

| Methods defined here:

|

| \_\_init\_\_(self, int\_type)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_repr\_\_(self)

| Return repr(self).

|

| \_\_str\_\_(self)

| String representation.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| max

| Maximum value of given dtype.

|

| min

| Minimum value of given dtype.

class inexact(number)

| Abstract base class of all numeric scalar types with a (potentially)

| inexact representation of the values in its range, such as

| floating-point numbers.

|

| Method resolution order:

| inexact

| number

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

int0 = class int32(signedinteger)

| Signed integer type, compatible with C ``int``.

| Character code: ``'i'``.

| Canonical name: ``np.intc``.

| Alias \*on this platform\*: ``np.intp``: Signed integer large enough to fit pointer, compatible with C ``intptr\_t``.

|

| Method resolution order:

| int32

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class int16(signedinteger)

| Signed integer type, compatible with C ``short``.

| Character code: ``'h'``.

| Canonical name: ``np.short``.

| Alias \*on this platform\*: ``np.int16``: 16-bit signed integer (-32768 to 32767).

|

| Method resolution order:

| int16

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class int32(signedinteger)

| Signed integer type, compatible with Python `int` anc C ``long``.

| Character code: ``'l'``.

| Canonical name: ``np.int\_``.

| Alias \*on this platform\*: ``np.int32``: 32-bit signed integer (-2147483648 to 2147483647).

|

| Method resolution order:

| int32

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class int64(signedinteger)

| Signed integer type, compatible with C ``long long``.

| Character code: ``'q'``.

| Canonical name: ``np.longlong``.

| Alias \*on this platform\*: ``np.int64``: 64-bit signed integer (-9223372036854775808 to 9223372036854775807).

|

| Method resolution order:

| int64

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class int8(signedinteger)

| Signed integer type, compatible with C ``char``.

| Character code: ``'b'``.

| Canonical name: ``np.byte``.

| Alias \*on this platform\*: ``np.int8``: 8-bit signed integer (-128 to 127).

|

| Method resolution order:

| int8

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

int\_ = class int32(signedinteger)

| Signed integer type, compatible with Python `int` anc C ``long``.

| Character code: ``'l'``.

| Canonical name: ``np.int\_``.

| Alias \*on this platform\*: ``np.int32``: 32-bit signed integer (-2147483648 to 2147483647).

|

| Method resolution order:

| int32

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

intc = class int32(signedinteger)

| Signed integer type, compatible with C ``int``.

| Character code: ``'i'``.

| Canonical name: ``np.intc``.

| Alias \*on this platform\*: ``np.intp``: Signed integer large enough to fit pointer, compatible with C ``intptr\_t``.

|

| Method resolution order:

| int32

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class integer(number)

| Abstract base class of all integer scalar types.

|

| Method resolution order:

| integer

| number

| generic

| builtins.object

|

| Data descriptors defined here:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

intp = class int32(signedinteger)

| Signed integer type, compatible with C ``int``.

| Character code: ``'i'``.

| Canonical name: ``np.intc``.

| Alias \*on this platform\*: ``np.intp``: Signed integer large enough to fit pointer, compatible with C ``intptr\_t``.

|

| Method resolution order:

| int32

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

long = class int(object)

| int([x]) -> integer

| int(x, base=10) -> integer

|

| Convert a number or string to an integer, or return 0 if no arguments

| are given. If x is a number, return x.\_\_int\_\_(). For floating point

| numbers, this truncates towards zero.

|

| If x is not a number or if base is given, then x must be a string,

| bytes, or bytearray instance representing an integer literal in the

| given base. The literal can be preceded by '+' or '-' and be surrounded

| by whitespace. The base defaults to 10. Valid bases are 0 and 2-36.

| Base 0 means to interpret the base from the string as an integer literal.

| >>> int('0b100', base=0)

| 4

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_ceil\_\_(...)

| Ceiling of an Integral returns itself.

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floor\_\_(...)

| Flooring an Integral returns itself.

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(self, format\_spec, /)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getnewargs\_\_(self, /)

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

| Rounding an Integral returns itself.

| Rounding with an ndigits argument also returns an integer.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_sizeof\_\_(self, /)

| Returns size in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_trunc\_\_(...)

| Truncating an Integral returns itself.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| bit\_length(self, /)

| Number of bits necessary to represent self in binary.

|

| >>> bin(37)

| '0b100101'

| >>> (37).bit\_length()

| 6

|

| conjugate(...)

| Returns self, the complex conjugate of any int.

|

| to\_bytes(self, /, length, byteorder, \*, signed=False)

| Return an array of bytes representing an integer.

|

| length

| Length of bytes object to use. An OverflowError is raised if the

| integer is not representable with the given number of bytes.

| byteorder

| The byte order used to represent the integer. If byteorder is 'big',

| the most significant byte is at the beginning of the byte array. If

| byteorder is 'little', the most significant byte is at the end of the

| byte array. To request the native byte order of the host system, use

| `sys.byteorder' as the byte order value.

| signed

| Determines whether two's complement is used to represent the integer.

| If signed is False and a negative integer is given, an OverflowError

| is raised.

|

| ----------------------------------------------------------------------

| Class methods defined here:

|

| from\_bytes(bytes, byteorder, \*, signed=False) from builtins.type

| Return the integer represented by the given array of bytes.

|

| bytes

| Holds the array of bytes to convert. The argument must either

| support the buffer protocol or be an iterable object producing bytes.

| Bytes and bytearray are examples of built-in objects that support the

| buffer protocol.

| byteorder

| The byte order used to represent the integer. If byteorder is 'big',

| the most significant byte is at the beginning of the byte array. If

| byteorder is 'little', the most significant byte is at the end of the

| byte array. To request the native byte order of the host system, use

| `sys.byteorder' as the byte order value.

| signed

| Indicates whether two's complement is used to represent the integer.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| denominator

| the denominator of a rational number in lowest terms

|

| imag

| the imaginary part of a complex number

|

| numerator

| the numerator of a rational number in lowest terms

|

| real

| the real part of a complex number

longcomplex = class complex128(complexfloating)

| Complex number type composed of two extended-precision floating-point

| numbers.

| Character code: ``'G'``.

| Canonical name: ``np.clongdouble``.

| Alias: ``np.clongfloat``.

| Alias: ``np.longcomplex``.

|

| Method resolution order:

| complex128

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

longdouble = class float64(floating)

| Extended-precision floating-point number type, compatible with C

| ``long double`` but not necessarily with IEEE 754 quadruple-precision.

| Character code: ``'g'``.

| Canonical name: ``np.longdouble``.

| Alias: ``np.longfloat``.

|

| Method resolution order:

| float64

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| longdouble.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.longdouble(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.longdouble(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.longdouble(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

longfloat = class float64(floating)

| Extended-precision floating-point number type, compatible with C

| ``long double`` but not necessarily with IEEE 754 quadruple-precision.

| Character code: ``'g'``.

| Canonical name: ``np.longdouble``.

| Alias: ``np.longfloat``.

|

| Method resolution order:

| float64

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| longdouble.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.longdouble(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.longdouble(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.longdouble(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

longlong = class int64(signedinteger)

| Signed integer type, compatible with C ``long long``.

| Character code: ``'q'``.

| Canonical name: ``np.longlong``.

| Alias \*on this platform\*: ``np.int64``: 64-bit signed integer (-9223372036854775808 to 9223372036854775807).

|

| Method resolution order:

| int64

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class matrix(ndarray)

| matrix(data, dtype=None, copy=True)

|

| matrix(data, dtype=None, copy=True)

|

| .. note:: It is no longer recommended to use this class, even for linear

| algebra. Instead use regular arrays. The class may be removed

| in the future.

|

| Returns a matrix from an array-like object, or from a string of data.

| A matrix is a specialized 2-D array that retains its 2-D nature

| through operations. It has certain special operators, such as ``\*``

| (matrix multiplication) and ``\*\*`` (matrix power).

|

| Parameters

| ----------

| data : array\_like or string

| If `data` is a string, it is interpreted as a matrix with commas

| or spaces separating columns, and semicolons separating rows.

| dtype : data-type

| Data-type of the output matrix.

| copy : bool

| If `data` is already an `ndarray`, then this flag determines

| whether the data is copied (the default), or whether a view is

| constructed.

|

| See Also

| --------

| array

|

| Examples

| --------

| >>> a = np.matrix('1 2; 3 4')

| >>> a

| matrix([[1, 2],

| [3, 4]])

|

| >>> np.matrix([[1, 2], [3, 4]])

| matrix([[1, 2],

| [3, 4]])

|

| Method resolution order:

| matrix

| ndarray

| builtins.object

|

| Methods defined here:

|

| \_\_array\_finalize\_\_(self, obj)

| None.

|

| \_\_getitem\_\_(self, index)

| Return self[key].

|

| \_\_imul\_\_(self, other)

| Return self\*=value.

|

| \_\_ipow\_\_(self, other)

| Return self\*\*=value.

|

| \_\_mul\_\_(self, other)

| Return self\*value.

|

| \_\_pow\_\_(self, other)

| Return pow(self, value, mod).

|

| \_\_rmul\_\_(self, other)

| Return value\*self.

|

| \_\_rpow\_\_(self, other)

| Return pow(value, self, mod).

|

| all(self, axis=None, out=None)

| Test whether all matrix elements along a given axis evaluate to True.

|

| Parameters

| ----------

| See `numpy.all` for complete descriptions

|

| See Also

| --------

| numpy.all

|

| Notes

| -----

| This is the same as `ndarray.all`, but it returns a `matrix` object.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> y = x[0]; y

| matrix([[0, 1, 2, 3]])

| >>> (x == y)

| matrix([[ True, True, True, True],

| [False, False, False, False],

| [False, False, False, False]])

| >>> (x == y).all()

| False

| >>> (x == y).all(0)

| matrix([[False, False, False, False]])

| >>> (x == y).all(1)

| matrix([[ True],

| [False],

| [False]])

|

| any(self, axis=None, out=None)

| Test whether any array element along a given axis evaluates to True.

|

| Refer to `numpy.any` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which logical OR is performed

| out : ndarray, optional

| Output to existing array instead of creating new one, must have

| same shape as expected output

|

| Returns

| -------

| any : bool, ndarray

| Returns a single bool if `axis` is ``None``; otherwise,

| returns `ndarray`

|

| argmax(self, axis=None, out=None)

| Indexes of the maximum values along an axis.

|

| Return the indexes of the first occurrences of the maximum values

| along the specified axis. If axis is None, the index is for the

| flattened matrix.

|

| Parameters

| ----------

| See `numpy.argmax` for complete descriptions

|

| See Also

| --------

| numpy.argmax

|

| Notes

| -----

| This is the same as `ndarray.argmax`, but returns a `matrix` object

| where `ndarray.argmax` would return an `ndarray`.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.argmax()

| 11

| >>> x.argmax(0)

| matrix([[2, 2, 2, 2]])

| >>> x.argmax(1)

| matrix([[3],

| [3],

| [3]])

|

| argmin(self, axis=None, out=None)

| Indexes of the minimum values along an axis.

|

| Return the indexes of the first occurrences of the minimum values

| along the specified axis. If axis is None, the index is for the

| flattened matrix.

|

| Parameters

| ----------

| See `numpy.argmin` for complete descriptions.

|

| See Also

| --------

| numpy.argmin

|

| Notes

| -----

| This is the same as `ndarray.argmin`, but returns a `matrix` object

| where `ndarray.argmin` would return an `ndarray`.

|

| Examples

| --------

| >>> x = -np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, -1, -2, -3],

| [ -4, -5, -6, -7],

| [ -8, -9, -10, -11]])

| >>> x.argmin()

| 11

| >>> x.argmin(0)

| matrix([[2, 2, 2, 2]])

| >>> x.argmin(1)

| matrix([[3],

| [3],

| [3]])

|

| flatten(self, order='C')

| Return a flattened copy of the matrix.

|

| All `N` elements of the matrix are placed into a single row.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| 'C' means to flatten in row-major (C-style) order. 'F' means to

| flatten in column-major (Fortran-style) order. 'A' means to

| flatten in column-major order if `m` is Fortran \*contiguous\* in

| memory, row-major order otherwise. 'K' means to flatten `m` in

| the order the elements occur in memory. The default is 'C'.

|

| Returns

| -------

| y : matrix

| A copy of the matrix, flattened to a `(1, N)` matrix where `N`

| is the number of elements in the original matrix.

|

| See Also

| --------

| ravel : Return a flattened array.

| flat : A 1-D flat iterator over the matrix.

|

| Examples

| --------

| >>> m = np.matrix([[1,2], [3,4]])

| >>> m.flatten()

| matrix([[1, 2, 3, 4]])

| >>> m.flatten('F')

| matrix([[1, 3, 2, 4]])

|

| getA = A(self)

| Return `self` as an `ndarray` object.

|

| Equivalent to ``np.asarray(self)``.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : ndarray

| `self` as an `ndarray`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.getA()

| array([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

|

| getA1 = A1(self)

| Return `self` as a flattened `ndarray`.

|

| Equivalent to ``np.asarray(x).ravel()``

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : ndarray

| `self`, 1-D, as an `ndarray`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.getA1()

| array([ 0, 1, 2, ..., 9, 10, 11])

|

| getH = H(self)

| Returns the (complex) conjugate transpose of `self`.

|

| Equivalent to ``np.transpose(self)`` if `self` is real-valued.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| complex conjugate transpose of `self`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4)))

| >>> z = x - 1j\*x; z

| matrix([[ 0. +0.j, 1. -1.j, 2. -2.j, 3. -3.j],

| [ 4. -4.j, 5. -5.j, 6. -6.j, 7. -7.j],

| [ 8. -8.j, 9. -9.j, 10.-10.j, 11.-11.j]])

| >>> z.getH()

| matrix([[ 0. -0.j, 4. +4.j, 8. +8.j],

| [ 1. +1.j, 5. +5.j, 9. +9.j],

| [ 2. +2.j, 6. +6.j, 10.+10.j],

| [ 3. +3.j, 7. +7.j, 11.+11.j]])

|

| getI = I(self)

| Returns the (multiplicative) inverse of invertible `self`.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| If `self` is non-singular, `ret` is such that ``ret \* self`` ==

| ``self \* ret`` == ``np.matrix(np.eye(self[0,:].size)`` all return

| ``True``.

|

| Raises

| ------

| numpy.linalg.LinAlgError: Singular matrix

| If `self` is singular.

|

| See Also

| --------

| linalg.inv

|

| Examples

| --------

| >>> m = np.matrix('[1, 2; 3, 4]'); m

| matrix([[1, 2],

| [3, 4]])

| >>> m.getI()

| matrix([[-2. , 1. ],

| [ 1.5, -0.5]])

| >>> m.getI() \* m

| matrix([[ 1., 0.], # may vary

| [ 0., 1.]])

|

| getT = T(self)

| Returns the transpose of the matrix.

|

| Does \*not\* conjugate! For the complex conjugate transpose, use ``.H``.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| The (non-conjugated) transpose of the matrix.

|

| See Also

| --------

| transpose, getH

|

| Examples

| --------

| >>> m = np.matrix('[1, 2; 3, 4]')

| >>> m

| matrix([[1, 2],

| [3, 4]])

| >>> m.getT()

| matrix([[1, 3],

| [2, 4]])

|

| max(self, axis=None, out=None)

| Return the maximum value along an axis.

|

| Parameters

| ----------

| See `amax` for complete descriptions

|

| See Also

| --------

| amax, ndarray.max

|

| Notes

| -----

| This is the same as `ndarray.max`, but returns a `matrix` object

| where `ndarray.max` would return an ndarray.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.max()

| 11

| >>> x.max(0)

| matrix([[ 8, 9, 10, 11]])

| >>> x.max(1)

| matrix([[ 3],

| [ 7],

| [11]])

|

| mean(self, axis=None, dtype=None, out=None)

| Returns the average of the matrix elements along the given axis.

|

| Refer to `numpy.mean` for full documentation.

|

| See Also

| --------

| numpy.mean

|

| Notes

| -----

| Same as `ndarray.mean` except that, where that returns an `ndarray`,

| this returns a `matrix` object.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3, 4)))

| >>> x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.mean()

| 5.5

| >>> x.mean(0)

| matrix([[4., 5., 6., 7.]])

| >>> x.mean(1)

| matrix([[ 1.5],

| [ 5.5],

| [ 9.5]])

|

| min(self, axis=None, out=None)

| Return the minimum value along an axis.

|

| Parameters

| ----------

| See `amin` for complete descriptions.

|

| See Also

| --------

| amin, ndarray.min

|

| Notes

| -----

| This is the same as `ndarray.min`, but returns a `matrix` object

| where `ndarray.min` would return an ndarray.

|

| Examples

| --------

| >>> x = -np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, -1, -2, -3],

| [ -4, -5, -6, -7],

| [ -8, -9, -10, -11]])

| >>> x.min()

| -11

| >>> x.min(0)

| matrix([[ -8, -9, -10, -11]])

| >>> x.min(1)

| matrix([[ -3],

| [ -7],

| [-11]])

|

| prod(self, axis=None, dtype=None, out=None)

| Return the product of the array elements over the given axis.

|

| Refer to `prod` for full documentation.

|

| See Also

| --------

| prod, ndarray.prod

|

| Notes

| -----

| Same as `ndarray.prod`, except, where that returns an `ndarray`, this

| returns a `matrix` object instead.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.prod()

| 0

| >>> x.prod(0)

| matrix([[ 0, 45, 120, 231]])

| >>> x.prod(1)

| matrix([[ 0],

| [ 840],

| [7920]])

|

| ptp(self, axis=None, out=None)

| Peak-to-peak (maximum - minimum) value along the given axis.

|

| Refer to `numpy.ptp` for full documentation.

|

| See Also

| --------

| numpy.ptp

|

| Notes

| -----

| Same as `ndarray.ptp`, except, where that would return an `ndarray` object,

| this returns a `matrix` object.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.ptp()

| 11

| >>> x.ptp(0)

| matrix([[8, 8, 8, 8]])

| >>> x.ptp(1)

| matrix([[3],

| [3],

| [3]])

|

| ravel(self, order='C')

| Return a flattened matrix.

|

| Refer to `numpy.ravel` for more documentation.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| The elements of `m` are read using this index order. 'C' means to

| index the elements in C-like order, with the last axis index

| changing fastest, back to the first axis index changing slowest.

| 'F' means to index the elements in Fortran-like index order, with

| the first index changing fastest, and the last index changing

| slowest. Note that the 'C' and 'F' options take no account of the

| memory layout of the underlying array, and only refer to the order

| of axis indexing. 'A' means to read the elements in Fortran-like

| index order if `m` is Fortran \*contiguous\* in memory, C-like order

| otherwise. 'K' means to read the elements in the order they occur

| in memory, except for reversing the data when strides are negative.

| By default, 'C' index order is used.

|

| Returns

| -------

| ret : matrix

| Return the matrix flattened to shape `(1, N)` where `N`

| is the number of elements in the original matrix.

| A copy is made only if necessary.

|

| See Also

| --------

| matrix.flatten : returns a similar output matrix but always a copy

| matrix.flat : a flat iterator on the array.

| numpy.ravel : related function which returns an ndarray

|

| squeeze(self, axis=None)

| Return a possibly reshaped matrix.

|

| Refer to `numpy.squeeze` for more documentation.

|

| Parameters

| ----------

| axis : None or int or tuple of ints, optional

| Selects a subset of the single-dimensional entries in the shape.

| If an axis is selected with shape entry greater than one,

| an error is raised.

|

| Returns

| -------

| squeezed : matrix

| The matrix, but as a (1, N) matrix if it had shape (N, 1).

|

| See Also

| --------

| numpy.squeeze : related function

|

| Notes

| -----

| If `m` has a single column then that column is returned

| as the single row of a matrix. Otherwise `m` is returned.

| The returned matrix is always either `m` itself or a view into `m`.

| Supplying an axis keyword argument will not affect the returned matrix

| but it may cause an error to be raised.

|

| Examples

| --------

| >>> c = np.matrix([[1], [2]])

| >>> c

| matrix([[1],

| [2]])

| >>> c.squeeze()

| matrix([[1, 2]])

| >>> r = c.T

| >>> r

| matrix([[1, 2]])

| >>> r.squeeze()

| matrix([[1, 2]])

| >>> m = np.matrix([[1, 2], [3, 4]])

| >>> m.squeeze()

| matrix([[1, 2],

| [3, 4]])

|

| std(self, axis=None, dtype=None, out=None, ddof=0)

| Return the standard deviation of the array elements along the given axis.

|

| Refer to `numpy.std` for full documentation.

|

| See Also

| --------

| numpy.std

|

| Notes

| -----

| This is the same as `ndarray.std`, except that where an `ndarray` would

| be returned, a `matrix` object is returned instead.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3, 4)))

| >>> x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.std()

| 3.4520525295346629 # may vary

| >>> x.std(0)

| matrix([[ 3.26598632, 3.26598632, 3.26598632, 3.26598632]]) # may vary

| >>> x.std(1)

| matrix([[ 1.11803399],

| [ 1.11803399],

| [ 1.11803399]])

|

| sum(self, axis=None, dtype=None, out=None)

| Returns the sum of the matrix elements, along the given axis.

|

| Refer to `numpy.sum` for full documentation.

|

| See Also

| --------

| numpy.sum

|

| Notes

| -----

| This is the same as `ndarray.sum`, except that where an `ndarray` would

| be returned, a `matrix` object is returned instead.

|

| Examples

| --------

| >>> x = np.matrix([[1, 2], [4, 3]])

| >>> x.sum()

| 10

| >>> x.sum(axis=1)

| matrix([[3],

| [7]])

| >>> x.sum(axis=1, dtype='float')

| matrix([[3.],

| [7.]])

| >>> out = np.zeros((2, 1), dtype='float')

| >>> x.sum(axis=1, dtype='float', out=np.asmatrix(out))

| matrix([[3.],

| [7.]])

|

| tolist(self)

| Return the matrix as a (possibly nested) list.

|

| See `ndarray.tolist` for full documentation.

|

| See Also

| --------

| ndarray.tolist

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.tolist()

| [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

|

| var(self, axis=None, dtype=None, out=None, ddof=0)

| Returns the variance of the matrix elements, along the given axis.

|

| Refer to `numpy.var` for full documentation.

|

| See Also

| --------

| numpy.var

|

| Notes

| -----

| This is the same as `ndarray.var`, except that where an `ndarray` would

| be returned, a `matrix` object is returned instead.

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3, 4)))

| >>> x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.var()

| 11.916666666666666

| >>> x.var(0)

| matrix([[ 10.66666667, 10.66666667, 10.66666667, 10.66666667]]) # may vary

| >>> x.var(1)

| matrix([[1.25],

| [1.25],

| [1.25]])

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(subtype, data, dtype=None, copy=True)

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| A

| Return `self` as an `ndarray` object.

|

| Equivalent to ``np.asarray(self)``.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : ndarray

| `self` as an `ndarray`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.getA()

| array([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

|

| A1

| Return `self` as a flattened `ndarray`.

|

| Equivalent to ``np.asarray(x).ravel()``

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : ndarray

| `self`, 1-D, as an `ndarray`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4))); x

| matrix([[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]])

| >>> x.getA1()

| array([ 0, 1, 2, ..., 9, 10, 11])

|

| H

| Returns the (complex) conjugate transpose of `self`.

|

| Equivalent to ``np.transpose(self)`` if `self` is real-valued.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| complex conjugate transpose of `self`

|

| Examples

| --------

| >>> x = np.matrix(np.arange(12).reshape((3,4)))

| >>> z = x - 1j\*x; z

| matrix([[ 0. +0.j, 1. -1.j, 2. -2.j, 3. -3.j],

| [ 4. -4.j, 5. -5.j, 6. -6.j, 7. -7.j],

| [ 8. -8.j, 9. -9.j, 10.-10.j, 11.-11.j]])

| >>> z.getH()

| matrix([[ 0. -0.j, 4. +4.j, 8. +8.j],

| [ 1. +1.j, 5. +5.j, 9. +9.j],

| [ 2. +2.j, 6. +6.j, 10.+10.j],

| [ 3. +3.j, 7. +7.j, 11.+11.j]])

|

| I

| Returns the (multiplicative) inverse of invertible `self`.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| If `self` is non-singular, `ret` is such that ``ret \* self`` ==

| ``self \* ret`` == ``np.matrix(np.eye(self[0,:].size)`` all return

| ``True``.

|

| Raises

| ------

| numpy.linalg.LinAlgError: Singular matrix

| If `self` is singular.

|

| See Also

| --------

| linalg.inv

|

| Examples

| --------

| >>> m = np.matrix('[1, 2; 3, 4]'); m

| matrix([[1, 2],

| [3, 4]])

| >>> m.getI()

| matrix([[-2. , 1. ],

| [ 1.5, -0.5]])

| >>> m.getI() \* m

| matrix([[ 1., 0.], # may vary

| [ 0., 1.]])

|

| T

| Returns the transpose of the matrix.

|

| Does \*not\* conjugate! For the complex conjugate transpose, use ``.H``.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| ret : matrix object

| The (non-conjugated) transpose of the matrix.

|

| See Also

| --------

| transpose, getH

|

| Examples

| --------

| >>> m = np.matrix('[1, 2; 3, 4]')

| >>> m

| matrix([[1, 2],

| [3, 4]])

| >>> m.getT()

| matrix([[1, 3],

| [2, 4]])

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_array\_priority\_\_ = 10.0

|

| ----------------------------------------------------------------------

| Methods inherited from ndarray:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| a.\_\_array\_\_(|dtype) -> reference if type unchanged, copy otherwise.

|

| Returns either a new reference to self if dtype is not given or a new array

| of provided data type if dtype is different from the current dtype of the

| array.

|

| \_\_array\_function\_\_(...)

|

| \_\_array\_prepare\_\_(...)

| a.\_\_array\_prepare\_\_(obj) -> Object of same type as ndarray object obj.

|

| \_\_array\_ufunc\_\_(...)

|

| \_\_array\_wrap\_\_(...)

| a.\_\_array\_wrap\_\_(obj) -> Object of same type as ndarray object a.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_copy\_\_(...)

| a.\_\_copy\_\_()

|

| Used if :func:`copy.copy` is called on an array. Returns a copy of the array.

|

| Equivalent to ``a.copy(order='K')``.

|

| \_\_deepcopy\_\_(...)

| a.\_\_deepcopy\_\_(memo, /) -> Deep copy of array.

|

| Used if :func:`copy.deepcopy` is called on an array.

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_iadd\_\_(self, value, /)

| Return self+=value.

|

| \_\_iand\_\_(self, value, /)

| Return self&=value.

|

| \_\_ifloordiv\_\_(self, value, /)

| Return self//=value.

|

| \_\_ilshift\_\_(self, value, /)

| Return self<<=value.

|

| \_\_imatmul\_\_(self, value, /)

| Return self@=value.

|

| \_\_imod\_\_(self, value, /)

| Return self%=value.

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_ior\_\_(self, value, /)

| Return self|=value.

|

| \_\_irshift\_\_(self, value, /)

| Return self>>=value.

|

| \_\_isub\_\_(self, value, /)

| Return self-=value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_itruediv\_\_(self, value, /)

| Return self/=value.

|

| \_\_ixor\_\_(self, value, /)

| Return self^=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_matmul\_\_(self, value, /)

| Return self@value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| a.\_\_reduce\_\_()

|

| For pickling.

|

| \_\_reduce\_ex\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmatmul\_\_(self, value, /)

| Return value@self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_setstate\_\_(...)

| a.\_\_setstate\_\_(state, /)

|

| For unpickling.

|

| The `state` argument must be a sequence that contains the following

| elements:

|

| Parameters

| ----------

| version : int

| optional pickle version. If omitted defaults to 0.

| shape : tuple

| dtype : data-type

| isFortran : bool

| rawdata : string or list

| a binary string with the data (or a list if 'a' is an object array)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| argpartition(...)

| a.argpartition(kth, axis=-1, kind='introselect', order=None)

|

| Returns the indices that would partition this array.

|

| Refer to `numpy.argpartition` for full documentation.

|

| .. versionadded:: 1.8.0

|

| See Also

| --------

| numpy.argpartition : equivalent function

|

| argsort(...)

| a.argsort(axis=-1, kind=None, order=None)

|

| Returns the indices that would sort this array.

|

| Refer to `numpy.argsort` for full documentation.

|

| See Also

| --------

| numpy.argsort : equivalent function

|

| astype(...)

| a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

|

| Copy of the array, cast to a specified type.

|

| Parameters

| ----------

| dtype : str or dtype

| Typecode or data-type to which the array is cast.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout order of the result.

| 'C' means C order, 'F' means Fortran order, 'A'

| means 'F' order if all the arrays are Fortran contiguous,

| 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur. Defaults to 'unsafe'

| for backwards compatibility.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| subok : bool, optional

| If True, then sub-classes will be passed-through (default), otherwise

| the returned array will be forced to be a base-class array.

| copy : bool, optional

| By default, astype always returns a newly allocated array. If this

| is set to false, and the `dtype`, `order`, and `subok`

| requirements are satisfied, the input array is returned instead

| of a copy.

|

| Returns

| -------

| arr\_t : ndarray

| Unless `copy` is False and the other conditions for returning the input

| array are satisfied (see description for `copy` input parameter), `arr\_t`

| is a new array of the same shape as the input array, with dtype, order

| given by `dtype`, `order`.

|

| Notes

| -----

| .. versionchanged:: 1.17.0

| Casting between a simple data type and a structured one is possible only

| for "unsafe" casting. Casting to multiple fields is allowed, but

| casting from multiple fields is not.

|

| .. versionchanged:: 1.9.0

| Casting from numeric to string types in 'safe' casting mode requires

| that the string dtype length is long enough to store the max

| integer/float value converted.

|

| Raises

| ------

| ComplexWarning

| When casting from complex to float or int. To avoid this,

| one should use ``a.real.astype(t)``.

|

| Examples

| --------

| >>> x = np.array([1, 2, 2.5])

| >>> x

| array([1. , 2. , 2.5])

|

| >>> x.astype(int)

| array([1, 2, 2])

|

| byteswap(...)

| a.byteswap(inplace=False)

|

| Swap the bytes of the array elements

|

| Toggle between low-endian and big-endian data representation by

| returning a byteswapped array, optionally swapped in-place.

|

| Parameters

| ----------

| inplace : bool, optional

| If ``True``, swap bytes in-place, default is ``False``.

|

| Returns

| -------

| out : ndarray

| The byteswapped array. If `inplace` is ``True``, this is

| a view to self.

|

| Examples

| --------

| >>> A = np.array([1, 256, 8755], dtype=np.int16)

| >>> list(map(hex, A))

| ['0x1', '0x100', '0x2233']

| >>> A.byteswap(inplace=True)

| array([ 256, 1, 13090], dtype=int16)

| >>> list(map(hex, A))

| ['0x100', '0x1', '0x3322']

|

| Arrays of strings are not swapped

|

| >>> A = np.array(['ceg', 'fac'])

| >>> A.byteswap()

| Traceback (most recent call last):

| ...

| UnicodeDecodeError: ...

|

| choose(...)

| a.choose(choices, out=None, mode='raise')

|

| Use an index array to construct a new array from a set of choices.

|

| Refer to `numpy.choose` for full documentation.

|

| See Also

| --------

| numpy.choose : equivalent function

|

| clip(...)

| a.clip(min=None, max=None, out=None, \*\*kwargs)

|

| Return an array whose values are limited to ``[min, max]``.

| One of max or min must be given.

|

| Refer to `numpy.clip` for full documentation.

|

| See Also

| --------

| numpy.clip : equivalent function

|

| compress(...)

| a.compress(condition, axis=None, out=None)

|

| Return selected slices of this array along given axis.

|

| Refer to `numpy.compress` for full documentation.

|

| See Also

| --------

| numpy.compress : equivalent function

|

| conj(...)

| a.conj()

|

| Complex-conjugate all elements.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| conjugate(...)

| a.conjugate()

|

| Return the complex conjugate, element-wise.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| copy(...)

| a.copy(order='C')

|

| Return a copy of the array.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout of the copy. 'C' means C-order,

| 'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

| 'C' otherwise. 'K' means match the layout of `a` as closely

| as possible. (Note that this function and :func:`numpy.copy` are very

| similar, but have different default values for their order=

| arguments.)

|

| See also

| --------

| numpy.copy

| numpy.copyto

|

| Examples

| --------

| >>> x = np.array([[1,2,3],[4,5,6]], order='F')

|

| >>> y = x.copy()

|

| >>> x.fill(0)

|

| >>> x

| array([[0, 0, 0],

| [0, 0, 0]])

|

| >>> y

| array([[1, 2, 3],

| [4, 5, 6]])

|

| >>> y.flags['C\_CONTIGUOUS']

| True

|

| cumprod(...)

| a.cumprod(axis=None, dtype=None, out=None)

|

| Return the cumulative product of the elements along the given axis.

|

| Refer to `numpy.cumprod` for full documentation.

|

| See Also

| --------

| numpy.cumprod : equivalent function

|

| cumsum(...)

| a.cumsum(axis=None, dtype=None, out=None)

|

| Return the cumulative sum of the elements along the given axis.

|

| Refer to `numpy.cumsum` for full documentation.

|

| See Also

| --------

| numpy.cumsum : equivalent function

|

| diagonal(...)

| a.diagonal(offset=0, axis1=0, axis2=1)

|

| Return specified diagonals. In NumPy 1.9 the returned array is a

| read-only view instead of a copy as in previous NumPy versions. In

| a future version the read-only restriction will be removed.

|

| Refer to :func:`numpy.diagonal` for full documentation.

|

| See Also

| --------

| numpy.diagonal : equivalent function

|

| dot(...)

| a.dot(b, out=None)

|

| Dot product of two arrays.

|

| Refer to `numpy.dot` for full documentation.

|

| See Also

| --------

| numpy.dot : equivalent function

|

| Examples

| --------

| >>> a = np.eye(2)

| >>> b = np.ones((2, 2)) \* 2

| >>> a.dot(b)

| array([[2., 2.],

| [2., 2.]])

|

| This array method can be conveniently chained:

|

| >>> a.dot(b).dot(b)

| array([[8., 8.],

| [8., 8.]])

|

| dump(...)

| a.dump(file)

|

| Dump a pickle of the array to the specified file.

| The array can be read back with pickle.load or numpy.load.

|

| Parameters

| ----------

| file : str or Path

| A string naming the dump file.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| dumps(...)

| a.dumps()

|

| Returns the pickle of the array as a string.

| pickle.loads or numpy.loads will convert the string back to an array.

|

| Parameters

| ----------

| None

|

| fill(...)

| a.fill(value)

|

| Fill the array with a scalar value.

|

| Parameters

| ----------

| value : scalar

| All elements of `a` will be assigned this value.

|

| Examples

| --------

| >>> a = np.array([1, 2])

| >>> a.fill(0)

| >>> a

| array([0, 0])

| >>> a = np.empty(2)

| >>> a.fill(1)

| >>> a

| array([1., 1.])

|

| getfield(...)

| a.getfield(dtype, offset=0)

|

| Returns a field of the given array as a certain type.

|

| A field is a view of the array data with a given data-type. The values in

| the view are determined by the given type and the offset into the current

| array in bytes. The offset needs to be such that the view dtype fits in the

| array dtype; for example an array of dtype complex128 has 16-byte elements.

| If taking a view with a 32-bit integer (4 bytes), the offset needs to be

| between 0 and 12 bytes.

|

| Parameters

| ----------

| dtype : str or dtype

| The data type of the view. The dtype size of the view can not be larger

| than that of the array itself.

| offset : int

| Number of bytes to skip before beginning the element view.

|

| Examples

| --------

| >>> x = np.diag([1.+1.j]\*2)

| >>> x[1, 1] = 2 + 4.j

| >>> x

| array([[1.+1.j, 0.+0.j],

| [0.+0.j, 2.+4.j]])

| >>> x.getfield(np.float64)

| array([[1., 0.],

| [0., 2.]])

|

| By choosing an offset of 8 bytes we can select the complex part of the

| array for our view:

|

| >>> x.getfield(np.float64, offset=8)

| array([[1., 0.],

| [0., 4.]])

|

| item(...)

| a.item(\*args)

|

| Copy an element of an array to a standard Python scalar and return it.

|

| Parameters

| ----------

| \\*args : Arguments (variable number and type)

|

| \* none: in this case, the method only works for arrays

| with one element (`a.size == 1`), which element is

| copied into a standard Python scalar object and returned.

|

| \* int\_type: this argument is interpreted as a flat index into

| the array, specifying which element to copy and return.

|

| \* tuple of int\_types: functions as does a single int\_type argument,

| except that the argument is interpreted as an nd-index into the

| array.

|

| Returns

| -------

| z : Standard Python scalar object

| A copy of the specified element of the array as a suitable

| Python scalar

|

| Notes

| -----

| When the data type of `a` is longdouble or clongdouble, item() returns

| a scalar array object because there is no available Python scalar that

| would not lose information. Void arrays return a buffer object for item(),

| unless fields are defined, in which case a tuple is returned.

|

| `item` is very similar to a[args], except, instead of an array scalar,

| a standard Python scalar is returned. This can be useful for speeding up

| access to elements of the array and doing arithmetic on elements of the

| array using Python's optimized math.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.item(3)

| 1

| >>> x.item(7)

| 0

| >>> x.item((0, 1))

| 2

| >>> x.item((2, 2))

| 1

|

| itemset(...)

| a.itemset(\*args)

|

| Insert scalar into an array (scalar is cast to array's dtype, if possible)

|

| There must be at least 1 argument, and define the last argument

| as \*item\*. Then, ``a.itemset(\*args)`` is equivalent to but faster

| than ``a[args] = item``. The item should be a scalar value and `args`

| must select a single item in the array `a`.

|

| Parameters

| ----------

| \\*args : Arguments

| If one argument: a scalar, only used in case `a` is of size 1.

| If two arguments: the last argument is the value to be set

| and must be a scalar, the first argument specifies a single array

| element location. It is either an int or a tuple.

|

| Notes

| -----

| Compared to indexing syntax, `itemset` provides some speed increase

| for placing a scalar into a particular location in an `ndarray`,

| if you must do this. However, generally this is discouraged:

| among other problems, it complicates the appearance of the code.

| Also, when using `itemset` (and `item`) inside a loop, be sure

| to assign the methods to a local variable to avoid the attribute

| look-up at each loop iteration.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.itemset(4, 0)

| >>> x.itemset((2, 2), 9)

| >>> x

| array([[2, 2, 6],

| [1, 0, 6],

| [1, 0, 9]])

|

| newbyteorder(...)

| arr.newbyteorder(new\_order='S')

|

| Return the array with the same data viewed with a different byte order.

|

| Equivalent to::

|

| arr.view(arr.dtype.newbytorder(new\_order))

|

| Changes are also made in all fields and sub-arrays of the array data

| type.

|

|

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_arr : array

| New array object with the dtype reflecting given change to the

| byte order.

|

| nonzero(...)

| a.nonzero()

|

| Return the indices of the elements that are non-zero.

|

| Refer to `numpy.nonzero` for full documentation.

|

| See Also

| --------

| numpy.nonzero : equivalent function

|

| partition(...)

| a.partition(kth, axis=-1, kind='introselect', order=None)

|

| Rearranges the elements in the array in such a way that the value of the

| element in kth position is in the position it would be in a sorted array.

| All elements smaller than the kth element are moved before this element and

| all equal or greater are moved behind it. The ordering of the elements in

| the two partitions is undefined.

|

| .. versionadded:: 1.8.0

|

| Parameters

| ----------

| kth : int or sequence of ints

| Element index to partition by. The kth element value will be in its

| final sorted position and all smaller elements will be moved before it

| and all equal or greater elements behind it.

| The order of all elements in the partitions is undefined.

| If provided with a sequence of kth it will partition all elements

| indexed by kth of them into their sorted position at once.

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'introselect'}, optional

| Selection algorithm. Default is 'introselect'.

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need to be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.partition : Return a parititioned copy of an array.

| argpartition : Indirect partition.

| sort : Full sort.

|

| Notes

| -----

| See ``np.partition`` for notes on the different algorithms.

|

| Examples

| --------

| >>> a = np.array([3, 4, 2, 1])

| >>> a.partition(3)

| >>> a

| array([2, 1, 3, 4])

|

| >>> a.partition((1, 3))

| >>> a

| array([1, 2, 3, 4])

|

| put(...)

| a.put(indices, values, mode='raise')

|

| Set ``a.flat[n] = values[n]`` for all `n` in indices.

|

| Refer to `numpy.put` for full documentation.

|

| See Also

| --------

| numpy.put : equivalent function

|

| repeat(...)

| a.repeat(repeats, axis=None)

|

| Repeat elements of an array.

|

| Refer to `numpy.repeat` for full documentation.

|

| See Also

| --------

| numpy.repeat : equivalent function

|

| reshape(...)

| a.reshape(shape, order='C')

|

| Returns an array containing the same data with a new shape.

|

| Refer to `numpy.reshape` for full documentation.

|

| See Also

| --------

| numpy.reshape : equivalent function

|

| Notes

| -----

| Unlike the free function `numpy.reshape`, this method on `ndarray` allows

| the elements of the shape parameter to be passed in as separate arguments.

| For example, ``a.reshape(10, 11)`` is equivalent to

| ``a.reshape((10, 11))``.

|

| resize(...)

| a.resize(new\_shape, refcheck=True)

|

| Change shape and size of array in-place.

|

| Parameters

| ----------

| new\_shape : tuple of ints, or `n` ints

| Shape of resized array.

| refcheck : bool, optional

| If False, reference count will not be checked. Default is True.

|

| Returns

| -------

| None

|

| Raises

| ------

| ValueError

| If `a` does not own its own data or references or views to it exist,

| and the data memory must be changed.

| PyPy only: will always raise if the data memory must be changed, since

| there is no reliable way to determine if references or views to it

| exist.

|

| SystemError

| If the `order` keyword argument is specified. This behaviour is a

| bug in NumPy.

|

| See Also

| --------

| resize : Return a new array with the specified shape.

|

| Notes

| -----

| This reallocates space for the data area if necessary.

|

| Only contiguous arrays (data elements consecutive in memory) can be

| resized.

|

| The purpose of the reference count check is to make sure you

| do not use this array as a buffer for another Python object and then

| reallocate the memory. However, reference counts can increase in

| other ways so if you are sure that you have not shared the memory

| for this array with another Python object, then you may safely set

| `refcheck` to False.

|

| Examples

| --------

| Shrinking an array: array is flattened (in the order that the data are

| stored in memory), resized, and reshaped:

|

| >>> a = np.array([[0, 1], [2, 3]], order='C')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [1]])

|

| >>> a = np.array([[0, 1], [2, 3]], order='F')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [2]])

|

| Enlarging an array: as above, but missing entries are filled with zeros:

|

| >>> b = np.array([[0, 1], [2, 3]])

| >>> b.resize(2, 3) # new\_shape parameter doesn't have to be a tuple

| >>> b

| array([[0, 1, 2],

| [3, 0, 0]])

|

| Referencing an array prevents resizing...

|

| >>> c = a

| >>> a.resize((1, 1))

| Traceback (most recent call last):

| ...

| ValueError: cannot resize an array that references or is referenced ...

|

| Unless `refcheck` is False:

|

| >>> a.resize((1, 1), refcheck=False)

| >>> a

| array([[0]])

| >>> c

| array([[0]])

|

| round(...)

| a.round(decimals=0, out=None)

|

| Return `a` with each element rounded to the given number of decimals.

|

| Refer to `numpy.around` for full documentation.

|

| See Also

| --------

| numpy.around : equivalent function

|

| searchsorted(...)

| a.searchsorted(v, side='left', sorter=None)

|

| Find indices where elements of v should be inserted in a to maintain order.

|

| For full documentation, see `numpy.searchsorted`

|

| See Also

| --------

| numpy.searchsorted : equivalent function

|

| setfield(...)

| a.setfield(val, dtype, offset=0)

|

| Put a value into a specified place in a field defined by a data-type.

|

| Place `val` into `a`'s field defined by `dtype` and beginning `offset`

| bytes into the field.

|

| Parameters

| ----------

| val : object

| Value to be placed in field.

| dtype : dtype object

| Data-type of the field in which to place `val`.

| offset : int, optional

| The number of bytes into the field at which to place `val`.

|

| Returns

| -------

| None

|

| See Also

| --------

| getfield

|

| Examples

| --------

| >>> x = np.eye(3)

| >>> x.getfield(np.float64)

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

| >>> x.setfield(3, np.int32)

| >>> x.getfield(np.int32)

| array([[3, 3, 3],

| [3, 3, 3],

| [3, 3, 3]], dtype=int32)

| >>> x

| array([[1.0e+000, 1.5e-323, 1.5e-323],

| [1.5e-323, 1.0e+000, 1.5e-323],

| [1.5e-323, 1.5e-323, 1.0e+000]])

| >>> x.setfield(np.eye(3), np.int32)

| >>> x

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

|

| setflags(...)

| a.setflags(write=None, align=None, uic=None)

|

| Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY),

| respectively.

|

| These Boolean-valued flags affect how numpy interprets the memory

| area used by `a` (see Notes below). The ALIGNED flag can only

| be set to True if the data is actually aligned according to the type.

| The WRITEBACKIFCOPY and (deprecated) UPDATEIFCOPY flags can never be set

| to True. The flag WRITEABLE can only be set to True if the array owns its

| own memory, or the ultimate owner of the memory exposes a writeable buffer

| interface, or is a string. (The exception for string is made so that

| unpickling can be done without copying memory.)

|

| Parameters

| ----------

| write : bool, optional

| Describes whether or not `a` can be written to.

| align : bool, optional

| Describes whether or not `a` is aligned properly for its type.

| uic : bool, optional

| Describes whether or not `a` is a copy of another "base" array.

|

| Notes

| -----

| Array flags provide information about how the memory area used

| for the array is to be interpreted. There are 7 Boolean flags

| in use, only four of which can be changed by the user:

| WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED.

|

| WRITEABLE (W) the data area can be written to;

|

| ALIGNED (A) the data and strides are aligned appropriately for the hardware

| (as determined by the compiler);

|

| UPDATEIFCOPY (U) (deprecated), replaced by WRITEBACKIFCOPY;

|

| WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced

| by .base). When the C-API function PyArray\_ResolveWritebackIfCopy is

| called, the base array will be updated with the contents of this array.

|

| All flags can be accessed using the single (upper case) letter as well

| as the full name.

|

| Examples

| --------

| >>> y = np.array([[3, 1, 7],

| ... [2, 0, 0],

| ... [8, 5, 9]])

| >>> y

| array([[3, 1, 7],

| [2, 0, 0],

| [8, 5, 9]])

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : True

| ALIGNED : True

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(write=0, align=0)

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : False

| ALIGNED : False

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(uic=1)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: cannot set WRITEBACKIFCOPY flag to True

|

| sort(...)

| a.sort(axis=-1, kind=None, order=None)

|

| Sort an array in-place. Refer to `numpy.sort` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

| Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

| and 'mergesort' use timsort under the covers and, in general, the

| actual implementation will vary with datatype. The 'mergesort' option

| is retained for backwards compatibility.

|

| .. versionchanged:: 1.15.0.

| The 'stable' option was added.

|

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.sort : Return a sorted copy of an array.

| argsort : Indirect sort.

| lexsort : Indirect stable sort on multiple keys.

| searchsorted : Find elements in sorted array.

| partition: Partial sort.

|

| Notes

| -----

| See `numpy.sort` for notes on the different sorting algorithms.

|

| Examples

| --------

| >>> a = np.array([[1,4], [3,1]])

| >>> a.sort(axis=1)

| >>> a

| array([[1, 4],

| [1, 3]])

| >>> a.sort(axis=0)

| >>> a

| array([[1, 3],

| [1, 4]])

|

| Use the `order` keyword to specify a field to use when sorting a

| structured array:

|

| >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])

| >>> a.sort(order='y')

| >>> a

| array([(b'c', 1), (b'a', 2)],

| dtype=[('x', 'S1'), ('y', '<i8')])

|

| swapaxes(...)

| a.swapaxes(axis1, axis2)

|

| Return a view of the array with `axis1` and `axis2` interchanged.

|

| Refer to `numpy.swapaxes` for full documentation.

|

| See Also

| --------

| numpy.swapaxes : equivalent function

|

| take(...)

| a.take(indices, axis=None, out=None, mode='raise')

|

| Return an array formed from the elements of `a` at the given indices.

|

| Refer to `numpy.take` for full documentation.

|

| See Also

| --------

| numpy.take : equivalent function

|

| tobytes(...)

| a.tobytes(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| .. versionadded:: 1.9.0

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| tofile(...)

| a.tofile(fid, sep="", format="%s")

|

| Write array to a file as text or binary (default).

|

| Data is always written in 'C' order, independent of the order of `a`.

| The data produced by this method can be recovered using the function

| fromfile().

|

| Parameters

| ----------

| fid : file or str or Path

| An open file object, or a string containing a filename.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| sep : str

| Separator between array items for text output.

| If "" (empty), a binary file is written, equivalent to

| ``file.write(a.tobytes())``.

| format : str

| Format string for text file output.

| Each entry in the array is formatted to text by first converting

| it to the closest Python type, and then using "format" % item.

|

| Notes

| -----

| This is a convenience function for quick storage of array data.

| Information on endianness and precision is lost, so this method is not a

| good choice for files intended to archive data or transport data between

| machines with different endianness. Some of these problems can be overcome

| by outputting the data as text files, at the expense of speed and file

| size.

|

| When fid is a file object, array contents are directly written to the

| file, bypassing the file object's ``write`` method. As a result, tofile

| cannot be used with files objects supporting compression (e.g., GzipFile)

| or file-like objects that do not support ``fileno()`` (e.g., BytesIO).

|

| tostring(...)

| a.tostring(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| trace(...)

| a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

|

| Return the sum along diagonals of the array.

|

| Refer to `numpy.trace` for full documentation.

|

| See Also

| --------

| numpy.trace : equivalent function

|

| transpose(...)

| a.transpose(\*axes)

|

| Returns a view of the array with axes transposed.

|

| For a 1-D array this has no effect, as a transposed vector is simply the

| same vector. To convert a 1-D array into a 2D column vector, an additional

| dimension must be added. `np.atleast2d(a).T` achieves this, as does

| `a[:, np.newaxis]`.

| For a 2-D array, this is a standard matrix transpose.

| For an n-D array, if axes are given, their order indicates how the

| axes are permuted (see Examples). If axes are not provided and

| ``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then

| ``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

|

| Parameters

| ----------

| axes : None, tuple of ints, or `n` ints

|

| \* None or no argument: reverses the order of the axes.

|

| \* tuple of ints: `i` in the `j`-th place in the tuple means `a`'s

| `i`-th axis becomes `a.transpose()`'s `j`-th axis.

|

| \* `n` ints: same as an n-tuple of the same ints (this form is

| intended simply as a "convenience" alternative to the tuple form)

|

| Returns

| -------

| out : ndarray

| View of `a`, with axes suitably permuted.

|

| See Also

| --------

| ndarray.T : Array property returning the array transposed.

| ndarray.reshape : Give a new shape to an array without changing its data.

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> a

| array([[1, 2],

| [3, 4]])

| >>> a.transpose()

| array([[1, 3],

| [2, 4]])

| >>> a.transpose((1, 0))

| array([[1, 3],

| [2, 4]])

| >>> a.transpose(1, 0)

| array([[1, 3],

| [2, 4]])

|

| view(...)

| a.view(dtype=None, type=None)

|

| New view of array with the same data.

|

| Parameters

| ----------

| dtype : data-type or ndarray sub-class, optional

| Data-type descriptor of the returned view, e.g., float32 or int16. The

| default, None, results in the view having the same data-type as `a`.

| This argument can also be specified as an ndarray sub-class, which

| then specifies the type of the returned object (this is equivalent to

| setting the ``type`` parameter).

| type : Python type, optional

| Type of the returned view, e.g., ndarray or matrix. Again, the

| default None results in type preservation.

|

| Notes

| -----

| ``a.view()`` is used two different ways:

|

| ``a.view(some\_dtype)`` or ``a.view(dtype=some\_dtype)`` constructs a view

| of the array's memory with a different data-type. This can cause a

| reinterpretation of the bytes of memory.

|

| ``a.view(ndarray\_subclass)`` or ``a.view(type=ndarray\_subclass)`` just

| returns an instance of `ndarray\_subclass` that looks at the same array

| (same shape, dtype, etc.) This does not cause a reinterpretation of the

| memory.

|

| For ``a.view(some\_dtype)``, if ``some\_dtype`` has a different number of

| bytes per entry than the previous dtype (for example, converting a

| regular array to a structured array), then the behavior of the view

| cannot be predicted just from the superficial appearance of ``a`` (shown

| by ``print(a)``). It also depends on exactly how ``a`` is stored in

| memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus

| defined as a slice or transpose, etc., the view may give different

| results.

|

|

| Examples

| --------

| >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

|

| Viewing array data using a different type and dtype:

|

| >>> y = x.view(dtype=np.int16, type=np.matrix)

| >>> y

| matrix([[513]], dtype=int16)

| >>> print(type(y))

| <class 'numpy.matrix'>

|

| Creating a view on a structured array so it can be used in calculations

|

| >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])

| >>> xv = x.view(dtype=np.int8).reshape(-1,2)

| >>> xv

| array([[1, 2],

| [3, 4]], dtype=int8)

| >>> xv.mean(0)

| array([2., 3.])

|

| Making changes to the view changes the underlying array

|

| >>> xv[0,1] = 20

| >>> x

| array([(1, 20), (3, 4)], dtype=[('a', 'i1'), ('b', 'i1')])

|

| Using a view to convert an array to a recarray:

|

| >>> z = x.view(np.recarray)

| >>> z.a

| array([1, 3], dtype=int8)

|

| Views share data:

|

| >>> x[0] = (9, 10)

| >>> z[0]

| (9, 10)

|

| Views that change the dtype size (bytes per entry) should normally be

| avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

|

| >>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)

| >>> y = x[:, 0:2]

| >>> y

| array([[1, 2],

| [4, 5]], dtype=int16)

| >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])

| Traceback (most recent call last):

| ...

| ValueError: To change to a dtype of a different size, the array must be C-contiguous

| >>> z = y.copy()

| >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])

| array([[(1, 2)],

| [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])

|

| ----------------------------------------------------------------------

| Data descriptors inherited from ndarray:

|

| \_\_array\_interface\_\_

| Array protocol: Python side.

|

| \_\_array\_struct\_\_

| Array protocol: C-struct side.

|

| base

| Base object if memory is from some other object.

|

| Examples

| --------

| The base of an array that owns its memory is None:

|

| >>> x = np.array([1,2,3,4])

| >>> x.base is None

| True

|

| Slicing creates a view, whose memory is shared with x:

|

| >>> y = x[2:]

| >>> y.base is x

| True

|

| ctypes

| An object to simplify the interaction of the array with the ctypes

| module.

|

| This attribute creates an object that makes it easier to use arrays

| when calling shared libraries with the ctypes module. The returned

| object has, among others, data, shape, and strides attributes (see

| Notes below) which themselves return ctypes objects that can be used

| as arguments to a shared library.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| c : Python object

| Possessing attributes data, shape, strides, etc.

|

| See Also

| --------

| numpy.ctypeslib

|

| Notes

| -----

| Below are the public attributes of this object which were documented

| in "Guide to NumPy" (we have omitted undocumented public attributes,

| as well as documented private attributes):

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.data

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.shape

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.strides

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.data\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.shape\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.strides\_as

| :noindex:

|

| If the ctypes module is not available, then the ctypes attribute

| of array objects still returns something useful, but ctypes objects

| are not returned and errors may be raised instead. In particular,

| the object will still have the ``as\_parameter`` attribute which will

| return an integer equal to the data attribute.

|

| Examples

| --------

| >>> import ctypes

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.ctypes.data

| 30439712

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long))

| <ctypes.LP\_c\_long object at 0x01F01300>

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long)).contents

| c\_long(0)

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_longlong)).contents

| c\_longlong(4294967296L)

| >>> x.ctypes.shape

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FFD580>

| >>> x.ctypes.shape\_as(ctypes.c\_long)

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides\_as(ctypes.c\_longlong)

| <numpy.core.\_internal.c\_longlong\_Array\_2 object at 0x01F01300>

|

| data

| Python buffer object pointing to the start of the array's data.

|

| dtype

| Data-type of the array's elements.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| d : numpy dtype object

|

| See Also

| --------

| numpy.dtype

|

| Examples

| --------

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.dtype

| dtype('int32')

| >>> type(x.dtype)

| <type 'numpy.dtype'>

|

| flags

| Information about the memory layout of the array.

|

| Attributes

| ----------

| C\_CONTIGUOUS (C)

| The data is in a single, C-style contiguous segment.

| F\_CONTIGUOUS (F)

| The data is in a single, Fortran-style contiguous segment.

| OWNDATA (O)

| The array owns the memory it uses or borrows it from another object.

| WRITEABLE (W)

| The data area can be written to. Setting this to False locks

| the data, making it read-only. A view (slice, etc.) inherits WRITEABLE

| from its base array at creation time, but a view of a writeable

| array may be subsequently locked while the base array remains writeable.

| (The opposite is not true, in that a view of a locked array may not

| be made writeable. However, currently, locking a base object does not

| lock any views that already reference it, so under that circumstance it

| is possible to alter the contents of a locked array via a previously

| created writeable view onto it.) Attempting to change a non-writeable

| array raises a RuntimeError exception.

| ALIGNED (A)

| The data and all elements are aligned appropriately for the hardware.

| WRITEBACKIFCOPY (X)

| This array is a copy of some other array. The C-API function

| PyArray\_ResolveWritebackIfCopy must be called before deallocating

| to the base array will be updated with the contents of this array.

| UPDATEIFCOPY (U)

| (Deprecated, use WRITEBACKIFCOPY) This array is a copy of some other array.

| When this array is

| deallocated, the base array will be updated with the contents of

| this array.

| FNC

| F\_CONTIGUOUS and not C\_CONTIGUOUS.

| FORC

| F\_CONTIGUOUS or C\_CONTIGUOUS (one-segment test).

| BEHAVED (B)

| ALIGNED and WRITEABLE.

| CARRAY (CA)

| BEHAVED and C\_CONTIGUOUS.

| FARRAY (FA)

| BEHAVED and F\_CONTIGUOUS and not C\_CONTIGUOUS.

|

| Notes

| -----

| The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),

| or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag

| names are only supported in dictionary access.

|

| Only the WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be

| changed by the user, via direct assignment to the attribute or dictionary

| entry, or by calling `ndarray.setflags`.

|

| The array flags cannot be set arbitrarily:

|

| - UPDATEIFCOPY can only be set ``False``.

| - WRITEBACKIFCOPY can only be set ``False``.

| - ALIGNED can only be set ``True`` if the data is truly aligned.

| - WRITEABLE can only be set ``True`` if the array owns its own memory

| or the ultimate owner of the memory exposes a writeable buffer

| interface or is a string.

|

| Arrays can be both C-style and Fortran-style contiguous simultaneously.

| This is clear for 1-dimensional arrays, but can also be true for higher

| dimensional arrays.

|

| Even for contiguous arrays a stride for a given dimension

| ``arr.strides[dim]`` may be \*arbitrary\* if ``arr.shape[dim] == 1``

| or the array has no elements.

| It does \*not\* generally hold that ``self.strides[-1] == self.itemsize``

| for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for

| Fortran-style contiguous arrays is true.

|

| flat

| A 1-D iterator over the array.

|

| This is a `numpy.flatiter` instance, which acts similarly to, but is not

| a subclass of, Python's built-in iterator object.

|

| See Also

| --------

| flatten : Return a copy of the array collapsed into one dimension.

|

| flatiter

|

| Examples

| --------

| >>> x = np.arange(1, 7).reshape(2, 3)

| >>> x

| array([[1, 2, 3],

| [4, 5, 6]])

| >>> x.flat[3]

| 4

| >>> x.T

| array([[1, 4],

| [2, 5],

| [3, 6]])

| >>> x.T.flat[3]

| 5

| >>> type(x.flat)

| <class 'numpy.flatiter'>

|

| An assignment example:

|

| >>> x.flat = 3; x

| array([[3, 3, 3],

| [3, 3, 3]])

| >>> x.flat[[1,4]] = 1; x

| array([[3, 1, 3],

| [3, 1, 3]])

|

| imag

| The imaginary part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.imag

| array([ 0. , 0.70710678])

| >>> x.imag.dtype

| dtype('float64')

|

| itemsize

| Length of one array element in bytes.

|

| Examples

| --------

| >>> x = np.array([1,2,3], dtype=np.float64)

| >>> x.itemsize

| 8

| >>> x = np.array([1,2,3], dtype=np.complex128)

| >>> x.itemsize

| 16

|

| nbytes

| Total bytes consumed by the elements of the array.

|

| Notes

| -----

| Does not include memory consumed by non-element attributes of the

| array object.

|

| Examples

| --------

| >>> x = np.zeros((3,5,2), dtype=np.complex128)

| >>> x.nbytes

| 480

| >>> np.prod(x.shape) \* x.itemsize

| 480

|

| ndim

| Number of array dimensions.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> x.ndim

| 1

| >>> y = np.zeros((2, 3, 4))

| >>> y.ndim

| 3

|

| real

| The real part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.real

| array([ 1. , 0.70710678])

| >>> x.real.dtype

| dtype('float64')

|

| See Also

| --------

| numpy.real : equivalent function

|

| shape

| Tuple of array dimensions.

|

| The shape property is usually used to get the current shape of an array,

| but may also be used to reshape the array in-place by assigning a tuple of

| array dimensions to it. As with `numpy.reshape`, one of the new shape

| dimensions can be -1, in which case its value is inferred from the size of

| the array and the remaining dimensions. Reshaping an array in-place will

| fail if a copy is required.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3, 4])

| >>> x.shape

| (4,)

| >>> y = np.zeros((2, 3, 4))

| >>> y.shape

| (2, 3, 4)

| >>> y.shape = (3, 8)

| >>> y

| array([[ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.]])

| >>> y.shape = (3, 6)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: total size of new array must be unchanged

| >>> np.zeros((4,2))[::2].shape = (-1,)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| AttributeError: incompatible shape for a non-contiguous array

|

| See Also

| --------

| numpy.reshape : similar function

| ndarray.reshape : similar method

|

| size

| Number of elements in the array.

|

| Equal to ``np.prod(a.shape)``, i.e., the product of the array's

| dimensions.

|

| Notes

| -----

| `a.size` returns a standard arbitrary precision Python integer. This

| may not be the case with other methods of obtaining the same value

| (like the suggested ``np.prod(a.shape)``, which returns an instance

| of ``np.int\_``), and may be relevant if the value is used further in

| calculations that may overflow a fixed size integer type.

|

| Examples

| --------

| >>> x = np.zeros((3, 5, 2), dtype=np.complex128)

| >>> x.size

| 30

| >>> np.prod(x.shape)

| 30

|

| strides

| Tuple of bytes to step in each dimension when traversing an array.

|

| The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`

| is::

|

| offset = sum(np.array(i) \* a.strides)

|

| A more detailed explanation of strides can be found in the

| "ndarray.rst" file in the NumPy reference guide.

|

| Notes

| -----

| Imagine an array of 32-bit integers (each 4 bytes)::

|

| x = np.array([[0, 1, 2, 3, 4],

| [5, 6, 7, 8, 9]], dtype=np.int32)

|

| This array is stored in memory as 40 bytes, one after the other

| (known as a contiguous block of memory). The strides of an array tell

| us how many bytes we have to skip in memory to move to the next position

| along a certain axis. For example, we have to skip 4 bytes (1 value) to

| move to the next column, but 20 bytes (5 values) to get to the same

| position in the next row. As such, the strides for the array `x` will be

| ``(20, 4)``.

|

| See Also

| --------

| numpy.lib.stride\_tricks.as\_strided

|

| Examples

| --------

| >>> y = np.reshape(np.arange(2\*3\*4), (2,3,4))

| >>> y

| array([[[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]],

| [[12, 13, 14, 15],

| [16, 17, 18, 19],

| [20, 21, 22, 23]]])

| >>> y.strides

| (48, 16, 4)

| >>> y[1,1,1]

| 17

| >>> offset=sum(y.strides \* np.array((1,1,1)))

| >>> offset/y.itemsize

| 17

|

| >>> x = np.reshape(np.arange(5\*6\*7\*8), (5,6,7,8)).transpose(2,3,1,0)

| >>> x.strides

| (32, 4, 224, 1344)

| >>> i = np.array([3,5,2,2])

| >>> offset = sum(i \* x.strides)

| >>> x[3,5,2,2]

| 813

| >>> offset / x.itemsize

| 813

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from ndarray:

|

| \_\_hash\_\_ = None

class memmap(ndarray)

| memmap(filename, dtype=<class 'numpy.uint8'>, mode='r+', offset=0, shape=None, order='C')

|

| Create a memory-map to an array stored in a \*binary\* file on disk.

|

| Memory-mapped files are used for accessing small segments of large files

| on disk, without reading the entire file into memory. NumPy's

| memmap's are array-like objects. This differs from Python's ``mmap``

| module, which uses file-like objects.

|

| This subclass of ndarray has some unpleasant interactions with

| some operations, because it doesn't quite fit properly as a subclass.

| An alternative to using this subclass is to create the ``mmap``

| object yourself, then create an ndarray with ndarray.\_\_new\_\_ directly,

| passing the object created in its 'buffer=' parameter.

|

| This class may at some point be turned into a factory function

| which returns a view into an mmap buffer.

|

| Delete the memmap instance to close the memmap file.

|

|

| Parameters

| ----------

| filename : str, file-like object, or pathlib.Path instance

| The file name or file object to be used as the array data buffer.

| dtype : data-type, optional

| The data-type used to interpret the file contents.

| Default is `uint8`.

| mode : {'r+', 'r', 'w+', 'c'}, optional

| The file is opened in this mode:

|

| +------+-------------------------------------------------------------+

| | 'r' | Open existing file for reading only. |

| +------+-------------------------------------------------------------+

| | 'r+' | Open existing file for reading and writing. |

| +------+-------------------------------------------------------------+

| | 'w+' | Create or overwrite existing file for reading and writing. |

| +------+-------------------------------------------------------------+

| | 'c' | Copy-on-write: assignments affect data in memory, but |

| | | changes are not saved to disk. The file on disk is |

| | | read-only. |

| +------+-------------------------------------------------------------+

|

| Default is 'r+'.

| offset : int, optional

| In the file, array data starts at this offset. Since `offset` is

| measured in bytes, it should normally be a multiple of the byte-size

| of `dtype`. When ``mode != 'r'``, even positive offsets beyond end of

| file are valid; The file will be extended to accommodate the

| additional data. By default, ``memmap`` will start at the beginning of

| the file, even if ``filename`` is a file pointer ``fp`` and

| ``fp.tell() != 0``.

| shape : tuple, optional

| The desired shape of the array. If ``mode == 'r'`` and the number

| of remaining bytes after `offset` is not a multiple of the byte-size

| of `dtype`, you must specify `shape`. By default, the returned array

| will be 1-D with the number of elements determined by file size

| and data-type.

| order : {'C', 'F'}, optional

| Specify the order of the ndarray memory layout:

| :term:`row-major`, C-style or :term:`column-major`,

| Fortran-style. This only has an effect if the shape is

| greater than 1-D. The default order is 'C'.

|

| Attributes

| ----------

| filename : str or pathlib.Path instance

| Path to the mapped file.

| offset : int

| Offset position in the file.

| mode : str

| File mode.

|

| Methods

| -------

| flush

| Flush any changes in memory to file on disk.

| When you delete a memmap object, flush is called first to write

| changes to disk before removing the object.

|

|

| See also

| --------

| lib.format.open\_memmap : Create or load a memory-mapped ``.npy`` file.

|

| Notes

| -----

| The memmap object can be used anywhere an ndarray is accepted.

| Given a memmap ``fp``, ``isinstance(fp, numpy.ndarray)`` returns

| ``True``.

|

| Memory-mapped files cannot be larger than 2GB on 32-bit systems.

|

| When a memmap causes a file to be created or extended beyond its

| current size in the filesystem, the contents of the new part are

| unspecified. On systems with POSIX filesystem semantics, the extended

| part will be filled with zero bytes.

|

| Examples

| --------

| >>> data = np.arange(12, dtype='float32')

| >>> data.resize((3,4))

|

| This example uses a temporary file so that doctest doesn't write

| files to your directory. You would use a 'normal' filename.

|

| >>> from tempfile import mkdtemp

| >>> import os.path as path

| >>> filename = path.join(mkdtemp(), 'newfile.dat')

|

| Create a memmap with dtype and shape that matches our data:

|

| >>> fp = np.memmap(filename, dtype='float32', mode='w+', shape=(3,4))

| >>> fp

| memmap([[0., 0., 0., 0.],

| [0., 0., 0., 0.],

| [0., 0., 0., 0.]], dtype=float32)

|

| Write data to memmap array:

|

| >>> fp[:] = data[:]

| >>> fp

| memmap([[ 0., 1., 2., 3.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.]], dtype=float32)

|

| >>> fp.filename == path.abspath(filename)

| True

|

| Deletion flushes memory changes to disk before removing the object:

|

| >>> del fp

|

| Load the memmap and verify data was stored:

|

| >>> newfp = np.memmap(filename, dtype='float32', mode='r', shape=(3,4))

| >>> newfp

| memmap([[ 0., 1., 2., 3.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.]], dtype=float32)

|

| Read-only memmap:

|

| >>> fpr = np.memmap(filename, dtype='float32', mode='r', shape=(3,4))

| >>> fpr.flags.writeable

| False

|

| Copy-on-write memmap:

|

| >>> fpc = np.memmap(filename, dtype='float32', mode='c', shape=(3,4))

| >>> fpc.flags.writeable

| True

|

| It's possible to assign to copy-on-write array, but values are only

| written into the memory copy of the array, and not written to disk:

|

| >>> fpc

| memmap([[ 0., 1., 2., 3.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.]], dtype=float32)

| >>> fpc[0,:] = 0

| >>> fpc

| memmap([[ 0., 0., 0., 0.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.]], dtype=float32)

|

| File on disk is unchanged:

|

| >>> fpr

| memmap([[ 0., 1., 2., 3.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.]], dtype=float32)

|

| Offset into a memmap:

|

| >>> fpo = np.memmap(filename, dtype='float32', mode='r', offset=16)

| >>> fpo

| memmap([ 4., 5., 6., 7., 8., 9., 10., 11.], dtype=float32)

|

| Method resolution order:

| memmap

| ndarray

| builtins.object

|

| Methods defined here:

|

| \_\_array\_finalize\_\_(self, obj)

| None.

|

| \_\_array\_wrap\_\_(self, arr, context=None)

| a.\_\_array\_wrap\_\_(obj) -> Object of same type as ndarray object a.

|

| \_\_getitem\_\_(self, index)

| Return self[key].

|

| flush(self)

| Write any changes in the array to the file on disk.

|

| For further information, see `memmap`.

|

| Parameters

| ----------

| None

|

| See Also

| --------

| memmap

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(subtype, filename, dtype=<class 'numpy.uint8'>, mode='r+', offset=0, shape=None, order='C')

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_array\_priority\_\_ = -100.0

|

| ----------------------------------------------------------------------

| Methods inherited from ndarray:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| a.\_\_array\_\_(|dtype) -> reference if type unchanged, copy otherwise.

|

| Returns either a new reference to self if dtype is not given or a new array

| of provided data type if dtype is different from the current dtype of the

| array.

|

| \_\_array\_function\_\_(...)

|

| \_\_array\_prepare\_\_(...)

| a.\_\_array\_prepare\_\_(obj) -> Object of same type as ndarray object obj.

|

| \_\_array\_ufunc\_\_(...)

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_copy\_\_(...)

| a.\_\_copy\_\_()

|

| Used if :func:`copy.copy` is called on an array. Returns a copy of the array.

|

| Equivalent to ``a.copy(order='K')``.

|

| \_\_deepcopy\_\_(...)

| a.\_\_deepcopy\_\_(memo, /) -> Deep copy of array.

|

| Used if :func:`copy.deepcopy` is called on an array.

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_iadd\_\_(self, value, /)

| Return self+=value.

|

| \_\_iand\_\_(self, value, /)

| Return self&=value.

|

| \_\_ifloordiv\_\_(self, value, /)

| Return self//=value.

|

| \_\_ilshift\_\_(self, value, /)

| Return self<<=value.

|

| \_\_imatmul\_\_(self, value, /)

| Return self@=value.

|

| \_\_imod\_\_(self, value, /)

| Return self%=value.

|

| \_\_imul\_\_(self, value, /)

| Return self\*=value.

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_ior\_\_(self, value, /)

| Return self|=value.

|

| \_\_ipow\_\_(self, value, /)

| Return self\*\*=value.

|

| \_\_irshift\_\_(self, value, /)

| Return self>>=value.

|

| \_\_isub\_\_(self, value, /)

| Return self-=value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_itruediv\_\_(self, value, /)

| Return self/=value.

|

| \_\_ixor\_\_(self, value, /)

| Return self^=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_matmul\_\_(self, value, /)

| Return self@value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| a.\_\_reduce\_\_()

|

| For pickling.

|

| \_\_reduce\_ex\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmatmul\_\_(self, value, /)

| Return value@self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_setstate\_\_(...)

| a.\_\_setstate\_\_(state, /)

|

| For unpickling.

|

| The `state` argument must be a sequence that contains the following

| elements:

|

| Parameters

| ----------

| version : int

| optional pickle version. If omitted defaults to 0.

| shape : tuple

| dtype : data-type

| isFortran : bool

| rawdata : string or list

| a binary string with the data (or a list if 'a' is an object array)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| a.all(axis=None, out=None, keepdims=False)

|

| Returns True if all elements evaluate to True.

|

| Refer to `numpy.all` for full documentation.

|

| See Also

| --------

| numpy.all : equivalent function

|

| any(...)

| a.any(axis=None, out=None, keepdims=False)

|

| Returns True if any of the elements of `a` evaluate to True.

|

| Refer to `numpy.any` for full documentation.

|

| See Also

| --------

| numpy.any : equivalent function

|

| argmax(...)

| a.argmax(axis=None, out=None)

|

| Return indices of the maximum values along the given axis.

|

| Refer to `numpy.argmax` for full documentation.

|

| See Also

| --------

| numpy.argmax : equivalent function

|

| argmin(...)

| a.argmin(axis=None, out=None)

|

| Return indices of the minimum values along the given axis of `a`.

|

| Refer to `numpy.argmin` for detailed documentation.

|

| See Also

| --------

| numpy.argmin : equivalent function

|

| argpartition(...)

| a.argpartition(kth, axis=-1, kind='introselect', order=None)

|

| Returns the indices that would partition this array.

|

| Refer to `numpy.argpartition` for full documentation.

|

| .. versionadded:: 1.8.0

|

| See Also

| --------

| numpy.argpartition : equivalent function

|

| argsort(...)

| a.argsort(axis=-1, kind=None, order=None)

|

| Returns the indices that would sort this array.

|

| Refer to `numpy.argsort` for full documentation.

|

| See Also

| --------

| numpy.argsort : equivalent function

|

| astype(...)

| a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

|

| Copy of the array, cast to a specified type.

|

| Parameters

| ----------

| dtype : str or dtype

| Typecode or data-type to which the array is cast.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout order of the result.

| 'C' means C order, 'F' means Fortran order, 'A'

| means 'F' order if all the arrays are Fortran contiguous,

| 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur. Defaults to 'unsafe'

| for backwards compatibility.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| subok : bool, optional

| If True, then sub-classes will be passed-through (default), otherwise

| the returned array will be forced to be a base-class array.

| copy : bool, optional

| By default, astype always returns a newly allocated array. If this

| is set to false, and the `dtype`, `order`, and `subok`

| requirements are satisfied, the input array is returned instead

| of a copy.

|

| Returns

| -------

| arr\_t : ndarray

| Unless `copy` is False and the other conditions for returning the input

| array are satisfied (see description for `copy` input parameter), `arr\_t`

| is a new array of the same shape as the input array, with dtype, order

| given by `dtype`, `order`.

|

| Notes

| -----

| .. versionchanged:: 1.17.0

| Casting between a simple data type and a structured one is possible only

| for "unsafe" casting. Casting to multiple fields is allowed, but

| casting from multiple fields is not.

|

| .. versionchanged:: 1.9.0

| Casting from numeric to string types in 'safe' casting mode requires

| that the string dtype length is long enough to store the max

| integer/float value converted.

|

| Raises

| ------

| ComplexWarning

| When casting from complex to float or int. To avoid this,

| one should use ``a.real.astype(t)``.

|

| Examples

| --------

| >>> x = np.array([1, 2, 2.5])

| >>> x

| array([1. , 2. , 2.5])

|

| >>> x.astype(int)

| array([1, 2, 2])

|

| byteswap(...)

| a.byteswap(inplace=False)

|

| Swap the bytes of the array elements

|

| Toggle between low-endian and big-endian data representation by

| returning a byteswapped array, optionally swapped in-place.

|

| Parameters

| ----------

| inplace : bool, optional

| If ``True``, swap bytes in-place, default is ``False``.

|

| Returns

| -------

| out : ndarray

| The byteswapped array. If `inplace` is ``True``, this is

| a view to self.

|

| Examples

| --------

| >>> A = np.array([1, 256, 8755], dtype=np.int16)

| >>> list(map(hex, A))

| ['0x1', '0x100', '0x2233']

| >>> A.byteswap(inplace=True)

| array([ 256, 1, 13090], dtype=int16)

| >>> list(map(hex, A))

| ['0x100', '0x1', '0x3322']

|

| Arrays of strings are not swapped

|

| >>> A = np.array(['ceg', 'fac'])

| >>> A.byteswap()

| Traceback (most recent call last):

| ...

| UnicodeDecodeError: ...

|

| choose(...)

| a.choose(choices, out=None, mode='raise')

|

| Use an index array to construct a new array from a set of choices.

|

| Refer to `numpy.choose` for full documentation.

|

| See Also

| --------

| numpy.choose : equivalent function

|

| clip(...)

| a.clip(min=None, max=None, out=None, \*\*kwargs)

|

| Return an array whose values are limited to ``[min, max]``.

| One of max or min must be given.

|

| Refer to `numpy.clip` for full documentation.

|

| See Also

| --------

| numpy.clip : equivalent function

|

| compress(...)

| a.compress(condition, axis=None, out=None)

|

| Return selected slices of this array along given axis.

|

| Refer to `numpy.compress` for full documentation.

|

| See Also

| --------

| numpy.compress : equivalent function

|

| conj(...)

| a.conj()

|

| Complex-conjugate all elements.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| conjugate(...)

| a.conjugate()

|

| Return the complex conjugate, element-wise.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| copy(...)

| a.copy(order='C')

|

| Return a copy of the array.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout of the copy. 'C' means C-order,

| 'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

| 'C' otherwise. 'K' means match the layout of `a` as closely

| as possible. (Note that this function and :func:`numpy.copy` are very

| similar, but have different default values for their order=

| arguments.)

|

| See also

| --------

| numpy.copy

| numpy.copyto

|

| Examples

| --------

| >>> x = np.array([[1,2,3],[4,5,6]], order='F')

|

| >>> y = x.copy()

|

| >>> x.fill(0)

|

| >>> x

| array([[0, 0, 0],

| [0, 0, 0]])

|

| >>> y

| array([[1, 2, 3],

| [4, 5, 6]])

|

| >>> y.flags['C\_CONTIGUOUS']

| True

|

| cumprod(...)

| a.cumprod(axis=None, dtype=None, out=None)

|

| Return the cumulative product of the elements along the given axis.

|

| Refer to `numpy.cumprod` for full documentation.

|

| See Also

| --------

| numpy.cumprod : equivalent function

|

| cumsum(...)

| a.cumsum(axis=None, dtype=None, out=None)

|

| Return the cumulative sum of the elements along the given axis.

|

| Refer to `numpy.cumsum` for full documentation.

|

| See Also

| --------

| numpy.cumsum : equivalent function

|

| diagonal(...)

| a.diagonal(offset=0, axis1=0, axis2=1)

|

| Return specified diagonals. In NumPy 1.9 the returned array is a

| read-only view instead of a copy as in previous NumPy versions. In

| a future version the read-only restriction will be removed.

|

| Refer to :func:`numpy.diagonal` for full documentation.

|

| See Also

| --------

| numpy.diagonal : equivalent function

|

| dot(...)

| a.dot(b, out=None)

|

| Dot product of two arrays.

|

| Refer to `numpy.dot` for full documentation.

|

| See Also

| --------

| numpy.dot : equivalent function

|

| Examples

| --------

| >>> a = np.eye(2)

| >>> b = np.ones((2, 2)) \* 2

| >>> a.dot(b)

| array([[2., 2.],

| [2., 2.]])

|

| This array method can be conveniently chained:

|

| >>> a.dot(b).dot(b)

| array([[8., 8.],

| [8., 8.]])

|

| dump(...)

| a.dump(file)

|

| Dump a pickle of the array to the specified file.

| The array can be read back with pickle.load or numpy.load.

|

| Parameters

| ----------

| file : str or Path

| A string naming the dump file.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| dumps(...)

| a.dumps()

|

| Returns the pickle of the array as a string.

| pickle.loads or numpy.loads will convert the string back to an array.

|

| Parameters

| ----------

| None

|

| fill(...)

| a.fill(value)

|

| Fill the array with a scalar value.

|

| Parameters

| ----------

| value : scalar

| All elements of `a` will be assigned this value.

|

| Examples

| --------

| >>> a = np.array([1, 2])

| >>> a.fill(0)

| >>> a

| array([0, 0])

| >>> a = np.empty(2)

| >>> a.fill(1)

| >>> a

| array([1., 1.])

|

| flatten(...)

| a.flatten(order='C')

|

| Return a copy of the array collapsed into one dimension.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| 'C' means to flatten in row-major (C-style) order.

| 'F' means to flatten in column-major (Fortran-

| style) order. 'A' means to flatten in column-major

| order if `a` is Fortran \*contiguous\* in memory,

| row-major order otherwise. 'K' means to flatten

| `a` in the order the elements occur in memory.

| The default is 'C'.

|

| Returns

| -------

| y : ndarray

| A copy of the input array, flattened to one dimension.

|

| See Also

| --------

| ravel : Return a flattened array.

| flat : A 1-D flat iterator over the array.

|

| Examples

| --------

| >>> a = np.array([[1,2], [3,4]])

| >>> a.flatten()

| array([1, 2, 3, 4])

| >>> a.flatten('F')

| array([1, 3, 2, 4])

|

| getfield(...)

| a.getfield(dtype, offset=0)

|

| Returns a field of the given array as a certain type.

|

| A field is a view of the array data with a given data-type. The values in

| the view are determined by the given type and the offset into the current

| array in bytes. The offset needs to be such that the view dtype fits in the

| array dtype; for example an array of dtype complex128 has 16-byte elements.

| If taking a view with a 32-bit integer (4 bytes), the offset needs to be

| between 0 and 12 bytes.

|

| Parameters

| ----------

| dtype : str or dtype

| The data type of the view. The dtype size of the view can not be larger

| than that of the array itself.

| offset : int

| Number of bytes to skip before beginning the element view.

|

| Examples

| --------

| >>> x = np.diag([1.+1.j]\*2)

| >>> x[1, 1] = 2 + 4.j

| >>> x

| array([[1.+1.j, 0.+0.j],

| [0.+0.j, 2.+4.j]])

| >>> x.getfield(np.float64)

| array([[1., 0.],

| [0., 2.]])

|

| By choosing an offset of 8 bytes we can select the complex part of the

| array for our view:

|

| >>> x.getfield(np.float64, offset=8)

| array([[1., 0.],

| [0., 4.]])

|

| item(...)

| a.item(\*args)

|

| Copy an element of an array to a standard Python scalar and return it.

|

| Parameters

| ----------

| \\*args : Arguments (variable number and type)

|

| \* none: in this case, the method only works for arrays

| with one element (`a.size == 1`), which element is

| copied into a standard Python scalar object and returned.

|

| \* int\_type: this argument is interpreted as a flat index into

| the array, specifying which element to copy and return.

|

| \* tuple of int\_types: functions as does a single int\_type argument,

| except that the argument is interpreted as an nd-index into the

| array.

|

| Returns

| -------

| z : Standard Python scalar object

| A copy of the specified element of the array as a suitable

| Python scalar

|

| Notes

| -----

| When the data type of `a` is longdouble or clongdouble, item() returns

| a scalar array object because there is no available Python scalar that

| would not lose information. Void arrays return a buffer object for item(),

| unless fields are defined, in which case a tuple is returned.

|

| `item` is very similar to a[args], except, instead of an array scalar,

| a standard Python scalar is returned. This can be useful for speeding up

| access to elements of the array and doing arithmetic on elements of the

| array using Python's optimized math.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.item(3)

| 1

| >>> x.item(7)

| 0

| >>> x.item((0, 1))

| 2

| >>> x.item((2, 2))

| 1

|

| itemset(...)

| a.itemset(\*args)

|

| Insert scalar into an array (scalar is cast to array's dtype, if possible)

|

| There must be at least 1 argument, and define the last argument

| as \*item\*. Then, ``a.itemset(\*args)`` is equivalent to but faster

| than ``a[args] = item``. The item should be a scalar value and `args`

| must select a single item in the array `a`.

|

| Parameters

| ----------

| \\*args : Arguments

| If one argument: a scalar, only used in case `a` is of size 1.

| If two arguments: the last argument is the value to be set

| and must be a scalar, the first argument specifies a single array

| element location. It is either an int or a tuple.

|

| Notes

| -----

| Compared to indexing syntax, `itemset` provides some speed increase

| for placing a scalar into a particular location in an `ndarray`,

| if you must do this. However, generally this is discouraged:

| among other problems, it complicates the appearance of the code.

| Also, when using `itemset` (and `item`) inside a loop, be sure

| to assign the methods to a local variable to avoid the attribute

| look-up at each loop iteration.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.itemset(4, 0)

| >>> x.itemset((2, 2), 9)

| >>> x

| array([[2, 2, 6],

| [1, 0, 6],

| [1, 0, 9]])

|

| max(...)

| a.max(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the maximum along a given axis.

|

| Refer to `numpy.amax` for full documentation.

|

| See Also

| --------

| numpy.amax : equivalent function

|

| mean(...)

| a.mean(axis=None, dtype=None, out=None, keepdims=False)

|

| Returns the average of the array elements along given axis.

|

| Refer to `numpy.mean` for full documentation.

|

| See Also

| --------

| numpy.mean : equivalent function

|

| min(...)

| a.min(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the minimum along a given axis.

|

| Refer to `numpy.amin` for full documentation.

|

| See Also

| --------

| numpy.amin : equivalent function

|

| newbyteorder(...)

| arr.newbyteorder(new\_order='S')

|

| Return the array with the same data viewed with a different byte order.

|

| Equivalent to::

|

| arr.view(arr.dtype.newbytorder(new\_order))

|

| Changes are also made in all fields and sub-arrays of the array data

| type.

|

|

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_arr : array

| New array object with the dtype reflecting given change to the

| byte order.

|

| nonzero(...)

| a.nonzero()

|

| Return the indices of the elements that are non-zero.

|

| Refer to `numpy.nonzero` for full documentation.

|

| See Also

| --------

| numpy.nonzero : equivalent function

|

| partition(...)

| a.partition(kth, axis=-1, kind='introselect', order=None)

|

| Rearranges the elements in the array in such a way that the value of the

| element in kth position is in the position it would be in a sorted array.

| All elements smaller than the kth element are moved before this element and

| all equal or greater are moved behind it. The ordering of the elements in

| the two partitions is undefined.

|

| .. versionadded:: 1.8.0

|

| Parameters

| ----------

| kth : int or sequence of ints

| Element index to partition by. The kth element value will be in its

| final sorted position and all smaller elements will be moved before it

| and all equal or greater elements behind it.

| The order of all elements in the partitions is undefined.

| If provided with a sequence of kth it will partition all elements

| indexed by kth of them into their sorted position at once.

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'introselect'}, optional

| Selection algorithm. Default is 'introselect'.

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need to be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.partition : Return a parititioned copy of an array.

| argpartition : Indirect partition.

| sort : Full sort.

|

| Notes

| -----

| See ``np.partition`` for notes on the different algorithms.

|

| Examples

| --------

| >>> a = np.array([3, 4, 2, 1])

| >>> a.partition(3)

| >>> a

| array([2, 1, 3, 4])

|

| >>> a.partition((1, 3))

| >>> a

| array([1, 2, 3, 4])

|

| prod(...)

| a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True)

|

| Return the product of the array elements over the given axis

|

| Refer to `numpy.prod` for full documentation.

|

| See Also

| --------

| numpy.prod : equivalent function

|

| ptp(...)

| a.ptp(axis=None, out=None, keepdims=False)

|

| Peak to peak (maximum - minimum) value along a given axis.

|

| Refer to `numpy.ptp` for full documentation.

|

| See Also

| --------

| numpy.ptp : equivalent function

|

| put(...)

| a.put(indices, values, mode='raise')

|

| Set ``a.flat[n] = values[n]`` for all `n` in indices.

|

| Refer to `numpy.put` for full documentation.

|

| See Also

| --------

| numpy.put : equivalent function

|

| ravel(...)

| a.ravel([order])

|

| Return a flattened array.

|

| Refer to `numpy.ravel` for full documentation.

|

| See Also

| --------

| numpy.ravel : equivalent function

|

| ndarray.flat : a flat iterator on the array.

|

| repeat(...)

| a.repeat(repeats, axis=None)

|

| Repeat elements of an array.

|

| Refer to `numpy.repeat` for full documentation.

|

| See Also

| --------

| numpy.repeat : equivalent function

|

| reshape(...)

| a.reshape(shape, order='C')

|

| Returns an array containing the same data with a new shape.

|

| Refer to `numpy.reshape` for full documentation.

|

| See Also

| --------

| numpy.reshape : equivalent function

|

| Notes

| -----

| Unlike the free function `numpy.reshape`, this method on `ndarray` allows

| the elements of the shape parameter to be passed in as separate arguments.

| For example, ``a.reshape(10, 11)`` is equivalent to

| ``a.reshape((10, 11))``.

|

| resize(...)

| a.resize(new\_shape, refcheck=True)

|

| Change shape and size of array in-place.

|

| Parameters

| ----------

| new\_shape : tuple of ints, or `n` ints

| Shape of resized array.

| refcheck : bool, optional

| If False, reference count will not be checked. Default is True.

|

| Returns

| -------

| None

|

| Raises

| ------

| ValueError

| If `a` does not own its own data or references or views to it exist,

| and the data memory must be changed.

| PyPy only: will always raise if the data memory must be changed, since

| there is no reliable way to determine if references or views to it

| exist.

|

| SystemError

| If the `order` keyword argument is specified. This behaviour is a

| bug in NumPy.

|

| See Also

| --------

| resize : Return a new array with the specified shape.

|

| Notes

| -----

| This reallocates space for the data area if necessary.

|

| Only contiguous arrays (data elements consecutive in memory) can be

| resized.

|

| The purpose of the reference count check is to make sure you

| do not use this array as a buffer for another Python object and then

| reallocate the memory. However, reference counts can increase in

| other ways so if you are sure that you have not shared the memory

| for this array with another Python object, then you may safely set

| `refcheck` to False.

|

| Examples

| --------

| Shrinking an array: array is flattened (in the order that the data are

| stored in memory), resized, and reshaped:

|

| >>> a = np.array([[0, 1], [2, 3]], order='C')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [1]])

|

| >>> a = np.array([[0, 1], [2, 3]], order='F')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [2]])

|

| Enlarging an array: as above, but missing entries are filled with zeros:

|

| >>> b = np.array([[0, 1], [2, 3]])

| >>> b.resize(2, 3) # new\_shape parameter doesn't have to be a tuple

| >>> b

| array([[0, 1, 2],

| [3, 0, 0]])

|

| Referencing an array prevents resizing...

|

| >>> c = a

| >>> a.resize((1, 1))

| Traceback (most recent call last):

| ...

| ValueError: cannot resize an array that references or is referenced ...

|

| Unless `refcheck` is False:

|

| >>> a.resize((1, 1), refcheck=False)

| >>> a

| array([[0]])

| >>> c

| array([[0]])

|

| round(...)

| a.round(decimals=0, out=None)

|

| Return `a` with each element rounded to the given number of decimals.

|

| Refer to `numpy.around` for full documentation.

|

| See Also

| --------

| numpy.around : equivalent function

|

| searchsorted(...)

| a.searchsorted(v, side='left', sorter=None)

|

| Find indices where elements of v should be inserted in a to maintain order.

|

| For full documentation, see `numpy.searchsorted`

|

| See Also

| --------

| numpy.searchsorted : equivalent function

|

| setfield(...)

| a.setfield(val, dtype, offset=0)

|

| Put a value into a specified place in a field defined by a data-type.

|

| Place `val` into `a`'s field defined by `dtype` and beginning `offset`

| bytes into the field.

|

| Parameters

| ----------

| val : object

| Value to be placed in field.

| dtype : dtype object

| Data-type of the field in which to place `val`.

| offset : int, optional

| The number of bytes into the field at which to place `val`.

|

| Returns

| -------

| None

|

| See Also

| --------

| getfield

|

| Examples

| --------

| >>> x = np.eye(3)

| >>> x.getfield(np.float64)

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

| >>> x.setfield(3, np.int32)

| >>> x.getfield(np.int32)

| array([[3, 3, 3],

| [3, 3, 3],

| [3, 3, 3]], dtype=int32)

| >>> x

| array([[1.0e+000, 1.5e-323, 1.5e-323],

| [1.5e-323, 1.0e+000, 1.5e-323],

| [1.5e-323, 1.5e-323, 1.0e+000]])

| >>> x.setfield(np.eye(3), np.int32)

| >>> x

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

|

| setflags(...)

| a.setflags(write=None, align=None, uic=None)

|

| Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY),

| respectively.

|

| These Boolean-valued flags affect how numpy interprets the memory

| area used by `a` (see Notes below). The ALIGNED flag can only

| be set to True if the data is actually aligned according to the type.

| The WRITEBACKIFCOPY and (deprecated) UPDATEIFCOPY flags can never be set

| to True. The flag WRITEABLE can only be set to True if the array owns its

| own memory, or the ultimate owner of the memory exposes a writeable buffer

| interface, or is a string. (The exception for string is made so that

| unpickling can be done without copying memory.)

|

| Parameters

| ----------

| write : bool, optional

| Describes whether or not `a` can be written to.

| align : bool, optional

| Describes whether or not `a` is aligned properly for its type.

| uic : bool, optional

| Describes whether or not `a` is a copy of another "base" array.

|

| Notes

| -----

| Array flags provide information about how the memory area used

| for the array is to be interpreted. There are 7 Boolean flags

| in use, only four of which can be changed by the user:

| WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED.

|

| WRITEABLE (W) the data area can be written to;

|

| ALIGNED (A) the data and strides are aligned appropriately for the hardware

| (as determined by the compiler);

|

| UPDATEIFCOPY (U) (deprecated), replaced by WRITEBACKIFCOPY;

|

| WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced

| by .base). When the C-API function PyArray\_ResolveWritebackIfCopy is

| called, the base array will be updated with the contents of this array.

|

| All flags can be accessed using the single (upper case) letter as well

| as the full name.

|

| Examples

| --------

| >>> y = np.array([[3, 1, 7],

| ... [2, 0, 0],

| ... [8, 5, 9]])

| >>> y

| array([[3, 1, 7],

| [2, 0, 0],

| [8, 5, 9]])

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : True

| ALIGNED : True

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(write=0, align=0)

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : False

| ALIGNED : False

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(uic=1)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: cannot set WRITEBACKIFCOPY flag to True

|

| sort(...)

| a.sort(axis=-1, kind=None, order=None)

|

| Sort an array in-place. Refer to `numpy.sort` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

| Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

| and 'mergesort' use timsort under the covers and, in general, the

| actual implementation will vary with datatype. The 'mergesort' option

| is retained for backwards compatibility.

|

| .. versionchanged:: 1.15.0.

| The 'stable' option was added.

|

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.sort : Return a sorted copy of an array.

| argsort : Indirect sort.

| lexsort : Indirect stable sort on multiple keys.

| searchsorted : Find elements in sorted array.

| partition: Partial sort.

|

| Notes

| -----

| See `numpy.sort` for notes on the different sorting algorithms.

|

| Examples

| --------

| >>> a = np.array([[1,4], [3,1]])

| >>> a.sort(axis=1)

| >>> a

| array([[1, 4],

| [1, 3]])

| >>> a.sort(axis=0)

| >>> a

| array([[1, 3],

| [1, 4]])

|

| Use the `order` keyword to specify a field to use when sorting a

| structured array:

|

| >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])

| >>> a.sort(order='y')

| >>> a

| array([(b'c', 1), (b'a', 2)],

| dtype=[('x', 'S1'), ('y', '<i8')])

|

| squeeze(...)

| a.squeeze(axis=None)

|

| Remove single-dimensional entries from the shape of `a`.

|

| Refer to `numpy.squeeze` for full documentation.

|

| See Also

| --------

| numpy.squeeze : equivalent function

|

| std(...)

| a.std(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the standard deviation of the array elements along given axis.

|

| Refer to `numpy.std` for full documentation.

|

| See Also

| --------

| numpy.std : equivalent function

|

| sum(...)

| a.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True)

|

| Return the sum of the array elements over the given axis.

|

| Refer to `numpy.sum` for full documentation.

|

| See Also

| --------

| numpy.sum : equivalent function

|

| swapaxes(...)

| a.swapaxes(axis1, axis2)

|

| Return a view of the array with `axis1` and `axis2` interchanged.

|

| Refer to `numpy.swapaxes` for full documentation.

|

| See Also

| --------

| numpy.swapaxes : equivalent function

|

| take(...)

| a.take(indices, axis=None, out=None, mode='raise')

|

| Return an array formed from the elements of `a` at the given indices.

|

| Refer to `numpy.take` for full documentation.

|

| See Also

| --------

| numpy.take : equivalent function

|

| tobytes(...)

| a.tobytes(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| .. versionadded:: 1.9.0

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| tofile(...)

| a.tofile(fid, sep="", format="%s")

|

| Write array to a file as text or binary (default).

|

| Data is always written in 'C' order, independent of the order of `a`.

| The data produced by this method can be recovered using the function

| fromfile().

|

| Parameters

| ----------

| fid : file or str or Path

| An open file object, or a string containing a filename.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| sep : str

| Separator between array items for text output.

| If "" (empty), a binary file is written, equivalent to

| ``file.write(a.tobytes())``.

| format : str

| Format string for text file output.

| Each entry in the array is formatted to text by first converting

| it to the closest Python type, and then using "format" % item.

|

| Notes

| -----

| This is a convenience function for quick storage of array data.

| Information on endianness and precision is lost, so this method is not a

| good choice for files intended to archive data or transport data between

| machines with different endianness. Some of these problems can be overcome

| by outputting the data as text files, at the expense of speed and file

| size.

|

| When fid is a file object, array contents are directly written to the

| file, bypassing the file object's ``write`` method. As a result, tofile

| cannot be used with files objects supporting compression (e.g., GzipFile)

| or file-like objects that do not support ``fileno()`` (e.g., BytesIO).

|

| tolist(...)

| a.tolist()

|

| Return the array as an ``a.ndim``-levels deep nested list of Python scalars.

|

| Return a copy of the array data as a (nested) Python list.

| Data items are converted to the nearest compatible builtin Python type, via

| the `~numpy.ndarray.item` function.

|

| If ``a.ndim`` is 0, then since the depth of the nested list is 0, it will

| not be a list at all, but a simple Python scalar.

|

| Parameters

| ----------

| none

|

| Returns

| -------

| y : object, or list of object, or list of list of object, or ...

| The possibly nested list of array elements.

|

| Notes

| -----

| The array may be recreated via ``a = np.array(a.tolist())``, although this

| may sometimes lose precision.

|

| Examples

| --------

| For a 1D array, ``a.tolist()`` is almost the same as ``list(a)``:

|

| >>> a = np.array([1, 2])

| >>> list(a)

| [1, 2]

| >>> a.tolist()

| [1, 2]

|

| However, for a 2D array, ``tolist`` applies recursively:

|

| >>> a = np.array([[1, 2], [3, 4]])

| >>> list(a)

| [array([1, 2]), array([3, 4])]

| >>> a.tolist()

| [[1, 2], [3, 4]]

|

| The base case for this recursion is a 0D array:

|

| >>> a = np.array(1)

| >>> list(a)

| Traceback (most recent call last):

| ...

| TypeError: iteration over a 0-d array

| >>> a.tolist()

| 1

|

| tostring(...)

| a.tostring(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| trace(...)

| a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

|

| Return the sum along diagonals of the array.

|

| Refer to `numpy.trace` for full documentation.

|

| See Also

| --------

| numpy.trace : equivalent function

|

| transpose(...)

| a.transpose(\*axes)

|

| Returns a view of the array with axes transposed.

|

| For a 1-D array this has no effect, as a transposed vector is simply the

| same vector. To convert a 1-D array into a 2D column vector, an additional

| dimension must be added. `np.atleast2d(a).T` achieves this, as does

| `a[:, np.newaxis]`.

| For a 2-D array, this is a standard matrix transpose.

| For an n-D array, if axes are given, their order indicates how the

| axes are permuted (see Examples). If axes are not provided and

| ``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then

| ``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

|

| Parameters

| ----------

| axes : None, tuple of ints, or `n` ints

|

| \* None or no argument: reverses the order of the axes.

|

| \* tuple of ints: `i` in the `j`-th place in the tuple means `a`'s

| `i`-th axis becomes `a.transpose()`'s `j`-th axis.

|

| \* `n` ints: same as an n-tuple of the same ints (this form is

| intended simply as a "convenience" alternative to the tuple form)

|

| Returns

| -------

| out : ndarray

| View of `a`, with axes suitably permuted.

|

| See Also

| --------

| ndarray.T : Array property returning the array transposed.

| ndarray.reshape : Give a new shape to an array without changing its data.

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> a

| array([[1, 2],

| [3, 4]])

| >>> a.transpose()

| array([[1, 3],

| [2, 4]])

| >>> a.transpose((1, 0))

| array([[1, 3],

| [2, 4]])

| >>> a.transpose(1, 0)

| array([[1, 3],

| [2, 4]])

|

| var(...)

| a.var(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the variance of the array elements, along given axis.

|

| Refer to `numpy.var` for full documentation.

|

| See Also

| --------

| numpy.var : equivalent function

|

| view(...)

| a.view(dtype=None, type=None)

|

| New view of array with the same data.

|

| Parameters

| ----------

| dtype : data-type or ndarray sub-class, optional

| Data-type descriptor of the returned view, e.g., float32 or int16. The

| default, None, results in the view having the same data-type as `a`.

| This argument can also be specified as an ndarray sub-class, which

| then specifies the type of the returned object (this is equivalent to

| setting the ``type`` parameter).

| type : Python type, optional

| Type of the returned view, e.g., ndarray or matrix. Again, the

| default None results in type preservation.

|

| Notes

| -----

| ``a.view()`` is used two different ways:

|

| ``a.view(some\_dtype)`` or ``a.view(dtype=some\_dtype)`` constructs a view

| of the array's memory with a different data-type. This can cause a

| reinterpretation of the bytes of memory.

|

| ``a.view(ndarray\_subclass)`` or ``a.view(type=ndarray\_subclass)`` just

| returns an instance of `ndarray\_subclass` that looks at the same array

| (same shape, dtype, etc.) This does not cause a reinterpretation of the

| memory.

|

| For ``a.view(some\_dtype)``, if ``some\_dtype`` has a different number of

| bytes per entry than the previous dtype (for example, converting a

| regular array to a structured array), then the behavior of the view

| cannot be predicted just from the superficial appearance of ``a`` (shown

| by ``print(a)``). It also depends on exactly how ``a`` is stored in

| memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus

| defined as a slice or transpose, etc., the view may give different

| results.

|

|

| Examples

| --------

| >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

|

| Viewing array data using a different type and dtype:

|

| >>> y = x.view(dtype=np.int16, type=np.matrix)

| >>> y

| matrix([[513]], dtype=int16)

| >>> print(type(y))

| <class 'numpy.matrix'>

|

| Creating a view on a structured array so it can be used in calculations

|

| >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])

| >>> xv = x.view(dtype=np.int8).reshape(-1,2)

| >>> xv

| array([[1, 2],

| [3, 4]], dtype=int8)

| >>> xv.mean(0)

| array([2., 3.])

|

| Making changes to the view changes the underlying array

|

| >>> xv[0,1] = 20

| >>> x

| array([(1, 20), (3, 4)], dtype=[('a', 'i1'), ('b', 'i1')])

|

| Using a view to convert an array to a recarray:

|

| >>> z = x.view(np.recarray)

| >>> z.a

| array([1, 3], dtype=int8)

|

| Views share data:

|

| >>> x[0] = (9, 10)

| >>> z[0]

| (9, 10)

|

| Views that change the dtype size (bytes per entry) should normally be

| avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

|

| >>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)

| >>> y = x[:, 0:2]

| >>> y

| array([[1, 2],

| [4, 5]], dtype=int16)

| >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])

| Traceback (most recent call last):

| ...

| ValueError: To change to a dtype of a different size, the array must be C-contiguous

| >>> z = y.copy()

| >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])

| array([[(1, 2)],

| [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])

|

| ----------------------------------------------------------------------

| Data descriptors inherited from ndarray:

|

| T

| The transposed array.

|

| Same as ``self.transpose()``.

|

| Examples

| --------

| >>> x = np.array([[1.,2.],[3.,4.]])

| >>> x

| array([[ 1., 2.],

| [ 3., 4.]])

| >>> x.T

| array([[ 1., 3.],

| [ 2., 4.]])

| >>> x = np.array([1.,2.,3.,4.])

| >>> x

| array([ 1., 2., 3., 4.])

| >>> x.T

| array([ 1., 2., 3., 4.])

|

| See Also

| --------

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side.

|

| \_\_array\_struct\_\_

| Array protocol: C-struct side.

|

| base

| Base object if memory is from some other object.

|

| Examples

| --------

| The base of an array that owns its memory is None:

|

| >>> x = np.array([1,2,3,4])

| >>> x.base is None

| True

|

| Slicing creates a view, whose memory is shared with x:

|

| >>> y = x[2:]

| >>> y.base is x

| True

|

| ctypes

| An object to simplify the interaction of the array with the ctypes

| module.

|

| This attribute creates an object that makes it easier to use arrays

| when calling shared libraries with the ctypes module. The returned

| object has, among others, data, shape, and strides attributes (see

| Notes below) which themselves return ctypes objects that can be used

| as arguments to a shared library.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| c : Python object

| Possessing attributes data, shape, strides, etc.

|

| See Also

| --------

| numpy.ctypeslib

|

| Notes

| -----

| Below are the public attributes of this object which were documented

| in "Guide to NumPy" (we have omitted undocumented public attributes,

| as well as documented private attributes):

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.data

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.shape

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.strides

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.data\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.shape\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.strides\_as

| :noindex:

|

| If the ctypes module is not available, then the ctypes attribute

| of array objects still returns something useful, but ctypes objects

| are not returned and errors may be raised instead. In particular,

| the object will still have the ``as\_parameter`` attribute which will

| return an integer equal to the data attribute.

|

| Examples

| --------

| >>> import ctypes

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.ctypes.data

| 30439712

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long))

| <ctypes.LP\_c\_long object at 0x01F01300>

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long)).contents

| c\_long(0)

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_longlong)).contents

| c\_longlong(4294967296L)

| >>> x.ctypes.shape

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FFD580>

| >>> x.ctypes.shape\_as(ctypes.c\_long)

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides\_as(ctypes.c\_longlong)

| <numpy.core.\_internal.c\_longlong\_Array\_2 object at 0x01F01300>

|

| data

| Python buffer object pointing to the start of the array's data.

|

| dtype

| Data-type of the array's elements.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| d : numpy dtype object

|

| See Also

| --------

| numpy.dtype

|

| Examples

| --------

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.dtype

| dtype('int32')

| >>> type(x.dtype)

| <type 'numpy.dtype'>

|

| flags

| Information about the memory layout of the array.

|

| Attributes

| ----------

| C\_CONTIGUOUS (C)

| The data is in a single, C-style contiguous segment.

| F\_CONTIGUOUS (F)

| The data is in a single, Fortran-style contiguous segment.

| OWNDATA (O)

| The array owns the memory it uses or borrows it from another object.

| WRITEABLE (W)

| The data area can be written to. Setting this to False locks

| the data, making it read-only. A view (slice, etc.) inherits WRITEABLE

| from its base array at creation time, but a view of a writeable

| array may be subsequently locked while the base array remains writeable.

| (The opposite is not true, in that a view of a locked array may not

| be made writeable. However, currently, locking a base object does not

| lock any views that already reference it, so under that circumstance it

| is possible to alter the contents of a locked array via a previously

| created writeable view onto it.) Attempting to change a non-writeable

| array raises a RuntimeError exception.

| ALIGNED (A)

| The data and all elements are aligned appropriately for the hardware.

| WRITEBACKIFCOPY (X)

| This array is a copy of some other array. The C-API function

| PyArray\_ResolveWritebackIfCopy must be called before deallocating

| to the base array will be updated with the contents of this array.

| UPDATEIFCOPY (U)

| (Deprecated, use WRITEBACKIFCOPY) This array is a copy of some other array.

| When this array is

| deallocated, the base array will be updated with the contents of

| this array.

| FNC

| F\_CONTIGUOUS and not C\_CONTIGUOUS.

| FORC

| F\_CONTIGUOUS or C\_CONTIGUOUS (one-segment test).

| BEHAVED (B)

| ALIGNED and WRITEABLE.

| CARRAY (CA)

| BEHAVED and C\_CONTIGUOUS.

| FARRAY (FA)

| BEHAVED and F\_CONTIGUOUS and not C\_CONTIGUOUS.

|

| Notes

| -----

| The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),

| or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag

| names are only supported in dictionary access.

|

| Only the WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be

| changed by the user, via direct assignment to the attribute or dictionary

| entry, or by calling `ndarray.setflags`.

|

| The array flags cannot be set arbitrarily:

|

| - UPDATEIFCOPY can only be set ``False``.

| - WRITEBACKIFCOPY can only be set ``False``.

| - ALIGNED can only be set ``True`` if the data is truly aligned.

| - WRITEABLE can only be set ``True`` if the array owns its own memory

| or the ultimate owner of the memory exposes a writeable buffer

| interface or is a string.

|

| Arrays can be both C-style and Fortran-style contiguous simultaneously.

| This is clear for 1-dimensional arrays, but can also be true for higher

| dimensional arrays.

|

| Even for contiguous arrays a stride for a given dimension

| ``arr.strides[dim]`` may be \*arbitrary\* if ``arr.shape[dim] == 1``

| or the array has no elements.

| It does \*not\* generally hold that ``self.strides[-1] == self.itemsize``

| for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for

| Fortran-style contiguous arrays is true.

|

| flat

| A 1-D iterator over the array.

|

| This is a `numpy.flatiter` instance, which acts similarly to, but is not

| a subclass of, Python's built-in iterator object.

|

| See Also

| --------

| flatten : Return a copy of the array collapsed into one dimension.

|

| flatiter

|

| Examples

| --------

| >>> x = np.arange(1, 7).reshape(2, 3)

| >>> x

| array([[1, 2, 3],

| [4, 5, 6]])

| >>> x.flat[3]

| 4

| >>> x.T

| array([[1, 4],

| [2, 5],

| [3, 6]])

| >>> x.T.flat[3]

| 5

| >>> type(x.flat)

| <class 'numpy.flatiter'>

|

| An assignment example:

|

| >>> x.flat = 3; x

| array([[3, 3, 3],

| [3, 3, 3]])

| >>> x.flat[[1,4]] = 1; x

| array([[3, 1, 3],

| [3, 1, 3]])

|

| imag

| The imaginary part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.imag

| array([ 0. , 0.70710678])

| >>> x.imag.dtype

| dtype('float64')

|

| itemsize

| Length of one array element in bytes.

|

| Examples

| --------

| >>> x = np.array([1,2,3], dtype=np.float64)

| >>> x.itemsize

| 8

| >>> x = np.array([1,2,3], dtype=np.complex128)

| >>> x.itemsize

| 16

|

| nbytes

| Total bytes consumed by the elements of the array.

|

| Notes

| -----

| Does not include memory consumed by non-element attributes of the

| array object.

|

| Examples

| --------

| >>> x = np.zeros((3,5,2), dtype=np.complex128)

| >>> x.nbytes

| 480

| >>> np.prod(x.shape) \* x.itemsize

| 480

|

| ndim

| Number of array dimensions.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> x.ndim

| 1

| >>> y = np.zeros((2, 3, 4))

| >>> y.ndim

| 3

|

| real

| The real part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.real

| array([ 1. , 0.70710678])

| >>> x.real.dtype

| dtype('float64')

|

| See Also

| --------

| numpy.real : equivalent function

|

| shape

| Tuple of array dimensions.

|

| The shape property is usually used to get the current shape of an array,

| but may also be used to reshape the array in-place by assigning a tuple of

| array dimensions to it. As with `numpy.reshape`, one of the new shape

| dimensions can be -1, in which case its value is inferred from the size of

| the array and the remaining dimensions. Reshaping an array in-place will

| fail if a copy is required.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3, 4])

| >>> x.shape

| (4,)

| >>> y = np.zeros((2, 3, 4))

| >>> y.shape

| (2, 3, 4)

| >>> y.shape = (3, 8)

| >>> y

| array([[ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.]])

| >>> y.shape = (3, 6)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: total size of new array must be unchanged

| >>> np.zeros((4,2))[::2].shape = (-1,)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| AttributeError: incompatible shape for a non-contiguous array

|

| See Also

| --------

| numpy.reshape : similar function

| ndarray.reshape : similar method

|

| size

| Number of elements in the array.

|

| Equal to ``np.prod(a.shape)``, i.e., the product of the array's

| dimensions.

|

| Notes

| -----

| `a.size` returns a standard arbitrary precision Python integer. This

| may not be the case with other methods of obtaining the same value

| (like the suggested ``np.prod(a.shape)``, which returns an instance

| of ``np.int\_``), and may be relevant if the value is used further in

| calculations that may overflow a fixed size integer type.

|

| Examples

| --------

| >>> x = np.zeros((3, 5, 2), dtype=np.complex128)

| >>> x.size

| 30

| >>> np.prod(x.shape)

| 30

|

| strides

| Tuple of bytes to step in each dimension when traversing an array.

|

| The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`

| is::

|

| offset = sum(np.array(i) \* a.strides)

|

| A more detailed explanation of strides can be found in the

| "ndarray.rst" file in the NumPy reference guide.

|

| Notes

| -----

| Imagine an array of 32-bit integers (each 4 bytes)::

|

| x = np.array([[0, 1, 2, 3, 4],

| [5, 6, 7, 8, 9]], dtype=np.int32)

|

| This array is stored in memory as 40 bytes, one after the other

| (known as a contiguous block of memory). The strides of an array tell

| us how many bytes we have to skip in memory to move to the next position

| along a certain axis. For example, we have to skip 4 bytes (1 value) to

| move to the next column, but 20 bytes (5 values) to get to the same

| position in the next row. As such, the strides for the array `x` will be

| ``(20, 4)``.

|

| See Also

| --------

| numpy.lib.stride\_tricks.as\_strided

|

| Examples

| --------

| >>> y = np.reshape(np.arange(2\*3\*4), (2,3,4))

| >>> y

| array([[[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]],

| [[12, 13, 14, 15],

| [16, 17, 18, 19],

| [20, 21, 22, 23]]])

| >>> y.strides

| (48, 16, 4)

| >>> y[1,1,1]

| 17

| >>> offset=sum(y.strides \* np.array((1,1,1)))

| >>> offset/y.itemsize

| 17

|

| >>> x = np.reshape(np.arange(5\*6\*7\*8), (5,6,7,8)).transpose(2,3,1,0)

| >>> x.strides

| (32, 4, 224, 1344)

| >>> i = np.array([3,5,2,2])

| >>> offset = sum(i \* x.strides)

| >>> x[3,5,2,2]

| 813

| >>> offset / x.itemsize

| 813

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from ndarray:

|

| \_\_hash\_\_ = None

class ndarray(builtins.object)

| ndarray(shape, dtype=float, buffer=None, offset=0,

| strides=None, order=None)

|

| An array object represents a multidimensional, homogeneous array

| of fixed-size items. An associated data-type object describes the

| format of each element in the array (its byte-order, how many bytes it

| occupies in memory, whether it is an integer, a floating point number,

| or something else, etc.)

|

| Arrays should be constructed using `array`, `zeros` or `empty` (refer

| to the See Also section below). The parameters given here refer to

| a low-level method (`ndarray(...)`) for instantiating an array.

|

| For more information, refer to the `numpy` module and examine the

| methods and attributes of an array.

|

| Parameters

| ----------

| (for the \_\_new\_\_ method; see Notes below)

|

| shape : tuple of ints

| Shape of created array.

| dtype : data-type, optional

| Any object that can be interpreted as a numpy data type.

| buffer : object exposing buffer interface, optional

| Used to fill the array with data.

| offset : int, optional

| Offset of array data in buffer.

| strides : tuple of ints, optional

| Strides of data in memory.

| order : {'C', 'F'}, optional

| Row-major (C-style) or column-major (Fortran-style) order.

|

| Attributes

| ----------

| T : ndarray

| Transpose of the array.

| data : buffer

| The array's elements, in memory.

| dtype : dtype object

| Describes the format of the elements in the array.

| flags : dict

| Dictionary containing information related to memory use, e.g.,

| 'C\_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

| flat : numpy.flatiter object

| Flattened version of the array as an iterator. The iterator

| allows assignments, e.g., ``x.flat = 3`` (See `ndarray.flat` for

| assignment examples; TODO).

| imag : ndarray

| Imaginary part of the array.

| real : ndarray

| Real part of the array.

| size : int

| Number of elements in the array.

| itemsize : int

| The memory use of each array element in bytes.

| nbytes : int

| The total number of bytes required to store the array data,

| i.e., ``itemsize \* size``.

| ndim : int

| The array's number of dimensions.

| shape : tuple of ints

| Shape of the array.

| strides : tuple of ints

| The step-size required to move from one element to the next in

| memory. For example, a contiguous ``(3, 4)`` array of type

| ``int16`` in C-order has strides ``(8, 2)``. This implies that

| to move from element to element in memory requires jumps of 2 bytes.

| To move from row-to-row, one needs to jump 8 bytes at a time

| (``2 \* 4``).

| ctypes : ctypes object

| Class containing properties of the array needed for interaction

| with ctypes.

| base : ndarray

| If the array is a view into another array, that array is its `base`

| (unless that array is also a view). The `base` array is where the

| array data is actually stored.

|

| See Also

| --------

| array : Construct an array.

| zeros : Create an array, each element of which is zero.

| empty : Create an array, but leave its allocated memory unchanged (i.e.,

| it contains "garbage").

| dtype : Create a data-type.

|

| Notes

| -----

| There are two modes of creating an array using ``\_\_new\_\_``:

|

| 1. If `buffer` is None, then only `shape`, `dtype`, and `order`

| are used.

| 2. If `buffer` is an object exposing the buffer interface, then

| all keywords are interpreted.

|

| No ``\_\_init\_\_`` method is needed because the array is fully initialized

| after the ``\_\_new\_\_`` method.

|

| Examples

| --------

| These examples illustrate the low-level `ndarray` constructor. Refer

| to the `See Also` section above for easier ways of constructing an

| ndarray.

|

| First mode, `buffer` is None:

|

| >>> np.ndarray(shape=(2,2), dtype=float, order='F')

| array([[0.0e+000, 0.0e+000], # random

| [ nan, 2.5e-323]])

|

| Second mode:

|

| >>> np.ndarray((2,), buffer=np.array([1,2,3]),

| ... offset=np.int\_().itemsize,

| ... dtype=int) # offset = 1\*itemsize, i.e. skip first element

| array([2, 3])

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| a.\_\_array\_\_(|dtype) -> reference if type unchanged, copy otherwise.

|

| Returns either a new reference to self if dtype is not given or a new array

| of provided data type if dtype is different from the current dtype of the

| array.

|

| \_\_array\_function\_\_(...)

|

| \_\_array\_prepare\_\_(...)

| a.\_\_array\_prepare\_\_(obj) -> Object of same type as ndarray object obj.

|

| \_\_array\_ufunc\_\_(...)

|

| \_\_array\_wrap\_\_(...)

| a.\_\_array\_wrap\_\_(obj) -> Object of same type as ndarray object a.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_copy\_\_(...)

| a.\_\_copy\_\_()

|

| Used if :func:`copy.copy` is called on an array. Returns a copy of the array.

|

| Equivalent to ``a.copy(order='K')``.

|

| \_\_deepcopy\_\_(...)

| a.\_\_deepcopy\_\_(memo, /) -> Deep copy of array.

|

| Used if :func:`copy.deepcopy` is called on an array.

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_iadd\_\_(self, value, /)

| Return self+=value.

|

| \_\_iand\_\_(self, value, /)

| Return self&=value.

|

| \_\_ifloordiv\_\_(self, value, /)

| Return self//=value.

|

| \_\_ilshift\_\_(self, value, /)

| Return self<<=value.

|

| \_\_imatmul\_\_(self, value, /)

| Return self@=value.

|

| \_\_imod\_\_(self, value, /)

| Return self%=value.

|

| \_\_imul\_\_(self, value, /)

| Return self\*=value.

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_ior\_\_(self, value, /)

| Return self|=value.

|

| \_\_ipow\_\_(self, value, /)

| Return self\*\*=value.

|

| \_\_irshift\_\_(self, value, /)

| Return self>>=value.

|

| \_\_isub\_\_(self, value, /)

| Return self-=value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_itruediv\_\_(self, value, /)

| Return self/=value.

|

| \_\_ixor\_\_(self, value, /)

| Return self^=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_matmul\_\_(self, value, /)

| Return self@value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| a.\_\_reduce\_\_()

|

| For pickling.

|

| \_\_reduce\_ex\_\_(...)

| Helper for pickle.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmatmul\_\_(self, value, /)

| Return value@self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_setstate\_\_(...)

| a.\_\_setstate\_\_(state, /)

|

| For unpickling.

|

| The `state` argument must be a sequence that contains the following

| elements:

|

| Parameters

| ----------

| version : int

| optional pickle version. If omitted defaults to 0.

| shape : tuple

| dtype : data-type

| isFortran : bool

| rawdata : string or list

| a binary string with the data (or a list if 'a' is an object array)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| a.all(axis=None, out=None, keepdims=False)

|

| Returns True if all elements evaluate to True.

|

| Refer to `numpy.all` for full documentation.

|

| See Also

| --------

| numpy.all : equivalent function

|

| any(...)

| a.any(axis=None, out=None, keepdims=False)

|

| Returns True if any of the elements of `a` evaluate to True.

|

| Refer to `numpy.any` for full documentation.

|

| See Also

| --------

| numpy.any : equivalent function

|

| argmax(...)

| a.argmax(axis=None, out=None)

|

| Return indices of the maximum values along the given axis.

|

| Refer to `numpy.argmax` for full documentation.

|

| See Also

| --------

| numpy.argmax : equivalent function

|

| argmin(...)

| a.argmin(axis=None, out=None)

|

| Return indices of the minimum values along the given axis of `a`.

|

| Refer to `numpy.argmin` for detailed documentation.

|

| See Also

| --------

| numpy.argmin : equivalent function

|

| argpartition(...)

| a.argpartition(kth, axis=-1, kind='introselect', order=None)

|

| Returns the indices that would partition this array.

|

| Refer to `numpy.argpartition` for full documentation.

|

| .. versionadded:: 1.8.0

|

| See Also

| --------

| numpy.argpartition : equivalent function

|

| argsort(...)

| a.argsort(axis=-1, kind=None, order=None)

|

| Returns the indices that would sort this array.

|

| Refer to `numpy.argsort` for full documentation.

|

| See Also

| --------

| numpy.argsort : equivalent function

|

| astype(...)

| a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

|

| Copy of the array, cast to a specified type.

|

| Parameters

| ----------

| dtype : str or dtype

| Typecode or data-type to which the array is cast.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout order of the result.

| 'C' means C order, 'F' means Fortran order, 'A'

| means 'F' order if all the arrays are Fortran contiguous,

| 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur. Defaults to 'unsafe'

| for backwards compatibility.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| subok : bool, optional

| If True, then sub-classes will be passed-through (default), otherwise

| the returned array will be forced to be a base-class array.

| copy : bool, optional

| By default, astype always returns a newly allocated array. If this

| is set to false, and the `dtype`, `order`, and `subok`

| requirements are satisfied, the input array is returned instead

| of a copy.

|

| Returns

| -------

| arr\_t : ndarray

| Unless `copy` is False and the other conditions for returning the input

| array are satisfied (see description for `copy` input parameter), `arr\_t`

| is a new array of the same shape as the input array, with dtype, order

| given by `dtype`, `order`.

|

| Notes

| -----

| .. versionchanged:: 1.17.0

| Casting between a simple data type and a structured one is possible only

| for "unsafe" casting. Casting to multiple fields is allowed, but

| casting from multiple fields is not.

|

| .. versionchanged:: 1.9.0

| Casting from numeric to string types in 'safe' casting mode requires

| that the string dtype length is long enough to store the max

| integer/float value converted.

|

| Raises

| ------

| ComplexWarning

| When casting from complex to float or int. To avoid this,

| one should use ``a.real.astype(t)``.

|

| Examples

| --------

| >>> x = np.array([1, 2, 2.5])

| >>> x

| array([1. , 2. , 2.5])

|

| >>> x.astype(int)

| array([1, 2, 2])

|

| byteswap(...)

| a.byteswap(inplace=False)

|

| Swap the bytes of the array elements

|

| Toggle between low-endian and big-endian data representation by

| returning a byteswapped array, optionally swapped in-place.

|

| Parameters

| ----------

| inplace : bool, optional

| If ``True``, swap bytes in-place, default is ``False``.

|

| Returns

| -------

| out : ndarray

| The byteswapped array. If `inplace` is ``True``, this is

| a view to self.

|

| Examples

| --------

| >>> A = np.array([1, 256, 8755], dtype=np.int16)

| >>> list(map(hex, A))

| ['0x1', '0x100', '0x2233']

| >>> A.byteswap(inplace=True)

| array([ 256, 1, 13090], dtype=int16)

| >>> list(map(hex, A))

| ['0x100', '0x1', '0x3322']

|

| Arrays of strings are not swapped

|

| >>> A = np.array(['ceg', 'fac'])

| >>> A.byteswap()

| Traceback (most recent call last):

| ...

| UnicodeDecodeError: ...

|

| choose(...)

| a.choose(choices, out=None, mode='raise')

|

| Use an index array to construct a new array from a set of choices.

|

| Refer to `numpy.choose` for full documentation.

|

| See Also

| --------

| numpy.choose : equivalent function

|

| clip(...)

| a.clip(min=None, max=None, out=None, \*\*kwargs)

|

| Return an array whose values are limited to ``[min, max]``.

| One of max or min must be given.

|

| Refer to `numpy.clip` for full documentation.

|

| See Also

| --------

| numpy.clip : equivalent function

|

| compress(...)

| a.compress(condition, axis=None, out=None)

|

| Return selected slices of this array along given axis.

|

| Refer to `numpy.compress` for full documentation.

|

| See Also

| --------

| numpy.compress : equivalent function

|

| conj(...)

| a.conj()

|

| Complex-conjugate all elements.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| conjugate(...)

| a.conjugate()

|

| Return the complex conjugate, element-wise.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| copy(...)

| a.copy(order='C')

|

| Return a copy of the array.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout of the copy. 'C' means C-order,

| 'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

| 'C' otherwise. 'K' means match the layout of `a` as closely

| as possible. (Note that this function and :func:`numpy.copy` are very

| similar, but have different default values for their order=

| arguments.)

|

| See also

| --------

| numpy.copy

| numpy.copyto

|

| Examples

| --------

| >>> x = np.array([[1,2,3],[4,5,6]], order='F')

|

| >>> y = x.copy()

|

| >>> x.fill(0)

|

| >>> x

| array([[0, 0, 0],

| [0, 0, 0]])

|

| >>> y

| array([[1, 2, 3],

| [4, 5, 6]])

|

| >>> y.flags['C\_CONTIGUOUS']

| True

|

| cumprod(...)

| a.cumprod(axis=None, dtype=None, out=None)

|

| Return the cumulative product of the elements along the given axis.

|

| Refer to `numpy.cumprod` for full documentation.

|

| See Also

| --------

| numpy.cumprod : equivalent function

|

| cumsum(...)

| a.cumsum(axis=None, dtype=None, out=None)

|

| Return the cumulative sum of the elements along the given axis.

|

| Refer to `numpy.cumsum` for full documentation.

|

| See Also

| --------

| numpy.cumsum : equivalent function

|

| diagonal(...)

| a.diagonal(offset=0, axis1=0, axis2=1)

|

| Return specified diagonals. In NumPy 1.9 the returned array is a

| read-only view instead of a copy as in previous NumPy versions. In

| a future version the read-only restriction will be removed.

|

| Refer to :func:`numpy.diagonal` for full documentation.

|

| See Also

| --------

| numpy.diagonal : equivalent function

|

| dot(...)

| a.dot(b, out=None)

|

| Dot product of two arrays.

|

| Refer to `numpy.dot` for full documentation.

|

| See Also

| --------

| numpy.dot : equivalent function

|

| Examples

| --------

| >>> a = np.eye(2)

| >>> b = np.ones((2, 2)) \* 2

| >>> a.dot(b)

| array([[2., 2.],

| [2., 2.]])

|

| This array method can be conveniently chained:

|

| >>> a.dot(b).dot(b)

| array([[8., 8.],

| [8., 8.]])

|

| dump(...)

| a.dump(file)

|

| Dump a pickle of the array to the specified file.

| The array can be read back with pickle.load or numpy.load.

|

| Parameters

| ----------

| file : str or Path

| A string naming the dump file.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| dumps(...)

| a.dumps()

|

| Returns the pickle of the array as a string.

| pickle.loads or numpy.loads will convert the string back to an array.

|

| Parameters

| ----------

| None

|

| fill(...)

| a.fill(value)

|

| Fill the array with a scalar value.

|

| Parameters

| ----------

| value : scalar

| All elements of `a` will be assigned this value.

|

| Examples

| --------

| >>> a = np.array([1, 2])

| >>> a.fill(0)

| >>> a

| array([0, 0])

| >>> a = np.empty(2)

| >>> a.fill(1)

| >>> a

| array([1., 1.])

|

| flatten(...)

| a.flatten(order='C')

|

| Return a copy of the array collapsed into one dimension.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| 'C' means to flatten in row-major (C-style) order.

| 'F' means to flatten in column-major (Fortran-

| style) order. 'A' means to flatten in column-major

| order if `a` is Fortran \*contiguous\* in memory,

| row-major order otherwise. 'K' means to flatten

| `a` in the order the elements occur in memory.

| The default is 'C'.

|

| Returns

| -------

| y : ndarray

| A copy of the input array, flattened to one dimension.

|

| See Also

| --------

| ravel : Return a flattened array.

| flat : A 1-D flat iterator over the array.

|

| Examples

| --------

| >>> a = np.array([[1,2], [3,4]])

| >>> a.flatten()

| array([1, 2, 3, 4])

| >>> a.flatten('F')

| array([1, 3, 2, 4])

|

| getfield(...)

| a.getfield(dtype, offset=0)

|

| Returns a field of the given array as a certain type.

|

| A field is a view of the array data with a given data-type. The values in

| the view are determined by the given type and the offset into the current

| array in bytes. The offset needs to be such that the view dtype fits in the

| array dtype; for example an array of dtype complex128 has 16-byte elements.

| If taking a view with a 32-bit integer (4 bytes), the offset needs to be

| between 0 and 12 bytes.

|

| Parameters

| ----------

| dtype : str or dtype

| The data type of the view. The dtype size of the view can not be larger

| than that of the array itself.

| offset : int

| Number of bytes to skip before beginning the element view.

|

| Examples

| --------

| >>> x = np.diag([1.+1.j]\*2)

| >>> x[1, 1] = 2 + 4.j

| >>> x

| array([[1.+1.j, 0.+0.j],

| [0.+0.j, 2.+4.j]])

| >>> x.getfield(np.float64)

| array([[1., 0.],

| [0., 2.]])

|

| By choosing an offset of 8 bytes we can select the complex part of the

| array for our view:

|

| >>> x.getfield(np.float64, offset=8)

| array([[1., 0.],

| [0., 4.]])

|

| item(...)

| a.item(\*args)

|

| Copy an element of an array to a standard Python scalar and return it.

|

| Parameters

| ----------

| \\*args : Arguments (variable number and type)

|

| \* none: in this case, the method only works for arrays

| with one element (`a.size == 1`), which element is

| copied into a standard Python scalar object and returned.

|

| \* int\_type: this argument is interpreted as a flat index into

| the array, specifying which element to copy and return.

|

| \* tuple of int\_types: functions as does a single int\_type argument,

| except that the argument is interpreted as an nd-index into the

| array.

|

| Returns

| -------

| z : Standard Python scalar object

| A copy of the specified element of the array as a suitable

| Python scalar

|

| Notes

| -----

| When the data type of `a` is longdouble or clongdouble, item() returns

| a scalar array object because there is no available Python scalar that

| would not lose information. Void arrays return a buffer object for item(),

| unless fields are defined, in which case a tuple is returned.

|

| `item` is very similar to a[args], except, instead of an array scalar,

| a standard Python scalar is returned. This can be useful for speeding up

| access to elements of the array and doing arithmetic on elements of the

| array using Python's optimized math.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.item(3)

| 1

| >>> x.item(7)

| 0

| >>> x.item((0, 1))

| 2

| >>> x.item((2, 2))

| 1

|

| itemset(...)

| a.itemset(\*args)

|

| Insert scalar into an array (scalar is cast to array's dtype, if possible)

|

| There must be at least 1 argument, and define the last argument

| as \*item\*. Then, ``a.itemset(\*args)`` is equivalent to but faster

| than ``a[args] = item``. The item should be a scalar value and `args`

| must select a single item in the array `a`.

|

| Parameters

| ----------

| \\*args : Arguments

| If one argument: a scalar, only used in case `a` is of size 1.

| If two arguments: the last argument is the value to be set

| and must be a scalar, the first argument specifies a single array

| element location. It is either an int or a tuple.

|

| Notes

| -----

| Compared to indexing syntax, `itemset` provides some speed increase

| for placing a scalar into a particular location in an `ndarray`,

| if you must do this. However, generally this is discouraged:

| among other problems, it complicates the appearance of the code.

| Also, when using `itemset` (and `item`) inside a loop, be sure

| to assign the methods to a local variable to avoid the attribute

| look-up at each loop iteration.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.itemset(4, 0)

| >>> x.itemset((2, 2), 9)

| >>> x

| array([[2, 2, 6],

| [1, 0, 6],

| [1, 0, 9]])

|

| max(...)

| a.max(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the maximum along a given axis.

|

| Refer to `numpy.amax` for full documentation.

|

| See Also

| --------

| numpy.amax : equivalent function

|

| mean(...)

| a.mean(axis=None, dtype=None, out=None, keepdims=False)

|

| Returns the average of the array elements along given axis.

|

| Refer to `numpy.mean` for full documentation.

|

| See Also

| --------

| numpy.mean : equivalent function

|

| min(...)

| a.min(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the minimum along a given axis.

|

| Refer to `numpy.amin` for full documentation.

|

| See Also

| --------

| numpy.amin : equivalent function

|

| newbyteorder(...)

| arr.newbyteorder(new\_order='S')

|

| Return the array with the same data viewed with a different byte order.

|

| Equivalent to::

|

| arr.view(arr.dtype.newbytorder(new\_order))

|

| Changes are also made in all fields and sub-arrays of the array data

| type.

|

|

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_arr : array

| New array object with the dtype reflecting given change to the

| byte order.

|

| nonzero(...)

| a.nonzero()

|

| Return the indices of the elements that are non-zero.

|

| Refer to `numpy.nonzero` for full documentation.

|

| See Also

| --------

| numpy.nonzero : equivalent function

|

| partition(...)

| a.partition(kth, axis=-1, kind='introselect', order=None)

|

| Rearranges the elements in the array in such a way that the value of the

| element in kth position is in the position it would be in a sorted array.

| All elements smaller than the kth element are moved before this element and

| all equal or greater are moved behind it. The ordering of the elements in

| the two partitions is undefined.

|

| .. versionadded:: 1.8.0

|

| Parameters

| ----------

| kth : int or sequence of ints

| Element index to partition by. The kth element value will be in its

| final sorted position and all smaller elements will be moved before it

| and all equal or greater elements behind it.

| The order of all elements in the partitions is undefined.

| If provided with a sequence of kth it will partition all elements

| indexed by kth of them into their sorted position at once.

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'introselect'}, optional

| Selection algorithm. Default is 'introselect'.

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need to be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.partition : Return a parititioned copy of an array.

| argpartition : Indirect partition.

| sort : Full sort.

|

| Notes

| -----

| See ``np.partition`` for notes on the different algorithms.

|

| Examples

| --------

| >>> a = np.array([3, 4, 2, 1])

| >>> a.partition(3)

| >>> a

| array([2, 1, 3, 4])

|

| >>> a.partition((1, 3))

| >>> a

| array([1, 2, 3, 4])

|

| prod(...)

| a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True)

|

| Return the product of the array elements over the given axis

|

| Refer to `numpy.prod` for full documentation.

|

| See Also

| --------

| numpy.prod : equivalent function

|

| ptp(...)

| a.ptp(axis=None, out=None, keepdims=False)

|

| Peak to peak (maximum - minimum) value along a given axis.

|

| Refer to `numpy.ptp` for full documentation.

|

| See Also

| --------

| numpy.ptp : equivalent function

|

| put(...)

| a.put(indices, values, mode='raise')

|

| Set ``a.flat[n] = values[n]`` for all `n` in indices.

|

| Refer to `numpy.put` for full documentation.

|

| See Also

| --------

| numpy.put : equivalent function

|

| ravel(...)

| a.ravel([order])

|

| Return a flattened array.

|

| Refer to `numpy.ravel` for full documentation.

|

| See Also

| --------

| numpy.ravel : equivalent function

|

| ndarray.flat : a flat iterator on the array.

|

| repeat(...)

| a.repeat(repeats, axis=None)

|

| Repeat elements of an array.

|

| Refer to `numpy.repeat` for full documentation.

|

| See Also

| --------

| numpy.repeat : equivalent function

|

| reshape(...)

| a.reshape(shape, order='C')

|

| Returns an array containing the same data with a new shape.

|

| Refer to `numpy.reshape` for full documentation.

|

| See Also

| --------

| numpy.reshape : equivalent function

|

| Notes

| -----

| Unlike the free function `numpy.reshape`, this method on `ndarray` allows

| the elements of the shape parameter to be passed in as separate arguments.

| For example, ``a.reshape(10, 11)`` is equivalent to

| ``a.reshape((10, 11))``.

|

| resize(...)

| a.resize(new\_shape, refcheck=True)

|

| Change shape and size of array in-place.

|

| Parameters

| ----------

| new\_shape : tuple of ints, or `n` ints

| Shape of resized array.

| refcheck : bool, optional

| If False, reference count will not be checked. Default is True.

|

| Returns

| -------

| None

|

| Raises

| ------

| ValueError

| If `a` does not own its own data or references or views to it exist,

| and the data memory must be changed.

| PyPy only: will always raise if the data memory must be changed, since

| there is no reliable way to determine if references or views to it

| exist.

|

| SystemError

| If the `order` keyword argument is specified. This behaviour is a

| bug in NumPy.

|

| See Also

| --------

| resize : Return a new array with the specified shape.

|

| Notes

| -----

| This reallocates space for the data area if necessary.

|

| Only contiguous arrays (data elements consecutive in memory) can be

| resized.

|

| The purpose of the reference count check is to make sure you

| do not use this array as a buffer for another Python object and then

| reallocate the memory. However, reference counts can increase in

| other ways so if you are sure that you have not shared the memory

| for this array with another Python object, then you may safely set

| `refcheck` to False.

|

| Examples

| --------

| Shrinking an array: array is flattened (in the order that the data are

| stored in memory), resized, and reshaped:

|

| >>> a = np.array([[0, 1], [2, 3]], order='C')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [1]])

|

| >>> a = np.array([[0, 1], [2, 3]], order='F')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [2]])

|

| Enlarging an array: as above, but missing entries are filled with zeros:

|

| >>> b = np.array([[0, 1], [2, 3]])

| >>> b.resize(2, 3) # new\_shape parameter doesn't have to be a tuple

| >>> b

| array([[0, 1, 2],

| [3, 0, 0]])

|

| Referencing an array prevents resizing...

|

| >>> c = a

| >>> a.resize((1, 1))

| Traceback (most recent call last):

| ...

| ValueError: cannot resize an array that references or is referenced ...

|

| Unless `refcheck` is False:

|

| >>> a.resize((1, 1), refcheck=False)

| >>> a

| array([[0]])

| >>> c

| array([[0]])

|

| round(...)

| a.round(decimals=0, out=None)

|

| Return `a` with each element rounded to the given number of decimals.

|

| Refer to `numpy.around` for full documentation.

|

| See Also

| --------

| numpy.around : equivalent function

|

| searchsorted(...)

| a.searchsorted(v, side='left', sorter=None)

|

| Find indices where elements of v should be inserted in a to maintain order.

|

| For full documentation, see `numpy.searchsorted`

|

| See Also

| --------

| numpy.searchsorted : equivalent function

|

| setfield(...)

| a.setfield(val, dtype, offset=0)

|

| Put a value into a specified place in a field defined by a data-type.

|

| Place `val` into `a`'s field defined by `dtype` and beginning `offset`

| bytes into the field.

|

| Parameters

| ----------

| val : object

| Value to be placed in field.

| dtype : dtype object

| Data-type of the field in which to place `val`.

| offset : int, optional

| The number of bytes into the field at which to place `val`.

|

| Returns

| -------

| None

|

| See Also

| --------

| getfield

|

| Examples

| --------

| >>> x = np.eye(3)

| >>> x.getfield(np.float64)

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

| >>> x.setfield(3, np.int32)

| >>> x.getfield(np.int32)

| array([[3, 3, 3],

| [3, 3, 3],

| [3, 3, 3]], dtype=int32)

| >>> x

| array([[1.0e+000, 1.5e-323, 1.5e-323],

| [1.5e-323, 1.0e+000, 1.5e-323],

| [1.5e-323, 1.5e-323, 1.0e+000]])

| >>> x.setfield(np.eye(3), np.int32)

| >>> x

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

|

| setflags(...)

| a.setflags(write=None, align=None, uic=None)

|

| Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY),

| respectively.

|

| These Boolean-valued flags affect how numpy interprets the memory

| area used by `a` (see Notes below). The ALIGNED flag can only

| be set to True if the data is actually aligned according to the type.

| The WRITEBACKIFCOPY and (deprecated) UPDATEIFCOPY flags can never be set

| to True. The flag WRITEABLE can only be set to True if the array owns its

| own memory, or the ultimate owner of the memory exposes a writeable buffer

| interface, or is a string. (The exception for string is made so that

| unpickling can be done without copying memory.)

|

| Parameters

| ----------

| write : bool, optional

| Describes whether or not `a` can be written to.

| align : bool, optional

| Describes whether or not `a` is aligned properly for its type.

| uic : bool, optional

| Describes whether or not `a` is a copy of another "base" array.

|

| Notes

| -----

| Array flags provide information about how the memory area used

| for the array is to be interpreted. There are 7 Boolean flags

| in use, only four of which can be changed by the user:

| WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED.

|

| WRITEABLE (W) the data area can be written to;

|

| ALIGNED (A) the data and strides are aligned appropriately for the hardware

| (as determined by the compiler);

|

| UPDATEIFCOPY (U) (deprecated), replaced by WRITEBACKIFCOPY;

|

| WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced

| by .base). When the C-API function PyArray\_ResolveWritebackIfCopy is

| called, the base array will be updated with the contents of this array.

|

| All flags can be accessed using the single (upper case) letter as well

| as the full name.

|

| Examples

| --------

| >>> y = np.array([[3, 1, 7],

| ... [2, 0, 0],

| ... [8, 5, 9]])

| >>> y

| array([[3, 1, 7],

| [2, 0, 0],

| [8, 5, 9]])

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : True

| ALIGNED : True

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(write=0, align=0)

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : False

| ALIGNED : False

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(uic=1)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: cannot set WRITEBACKIFCOPY flag to True

|

| sort(...)

| a.sort(axis=-1, kind=None, order=None)

|

| Sort an array in-place. Refer to `numpy.sort` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

| Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

| and 'mergesort' use timsort under the covers and, in general, the

| actual implementation will vary with datatype. The 'mergesort' option

| is retained for backwards compatibility.

|

| .. versionchanged:: 1.15.0.

| The 'stable' option was added.

|

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.sort : Return a sorted copy of an array.

| argsort : Indirect sort.

| lexsort : Indirect stable sort on multiple keys.

| searchsorted : Find elements in sorted array.

| partition: Partial sort.

|

| Notes

| -----

| See `numpy.sort` for notes on the different sorting algorithms.

|

| Examples

| --------

| >>> a = np.array([[1,4], [3,1]])

| >>> a.sort(axis=1)

| >>> a

| array([[1, 4],

| [1, 3]])

| >>> a.sort(axis=0)

| >>> a

| array([[1, 3],

| [1, 4]])

|

| Use the `order` keyword to specify a field to use when sorting a

| structured array:

|

| >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])

| >>> a.sort(order='y')

| >>> a

| array([(b'c', 1), (b'a', 2)],

| dtype=[('x', 'S1'), ('y', '<i8')])

|

| squeeze(...)

| a.squeeze(axis=None)

|

| Remove single-dimensional entries from the shape of `a`.

|

| Refer to `numpy.squeeze` for full documentation.

|

| See Also

| --------

| numpy.squeeze : equivalent function

|

| std(...)

| a.std(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the standard deviation of the array elements along given axis.

|

| Refer to `numpy.std` for full documentation.

|

| See Also

| --------

| numpy.std : equivalent function

|

| sum(...)

| a.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True)

|

| Return the sum of the array elements over the given axis.

|

| Refer to `numpy.sum` for full documentation.

|

| See Also

| --------

| numpy.sum : equivalent function

|

| swapaxes(...)

| a.swapaxes(axis1, axis2)

|

| Return a view of the array with `axis1` and `axis2` interchanged.

|

| Refer to `numpy.swapaxes` for full documentation.

|

| See Also

| --------

| numpy.swapaxes : equivalent function

|

| take(...)

| a.take(indices, axis=None, out=None, mode='raise')

|

| Return an array formed from the elements of `a` at the given indices.

|

| Refer to `numpy.take` for full documentation.

|

| See Also

| --------

| numpy.take : equivalent function

|

| tobytes(...)

| a.tobytes(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| .. versionadded:: 1.9.0

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| tofile(...)

| a.tofile(fid, sep="", format="%s")

|

| Write array to a file as text or binary (default).

|

| Data is always written in 'C' order, independent of the order of `a`.

| The data produced by this method can be recovered using the function

| fromfile().

|

| Parameters

| ----------

| fid : file or str or Path

| An open file object, or a string containing a filename.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| sep : str

| Separator between array items for text output.

| If "" (empty), a binary file is written, equivalent to

| ``file.write(a.tobytes())``.

| format : str

| Format string for text file output.

| Each entry in the array is formatted to text by first converting

| it to the closest Python type, and then using "format" % item.

|

| Notes

| -----

| This is a convenience function for quick storage of array data.

| Information on endianness and precision is lost, so this method is not a

| good choice for files intended to archive data or transport data between

| machines with different endianness. Some of these problems can be overcome

| by outputting the data as text files, at the expense of speed and file

| size.

|

| When fid is a file object, array contents are directly written to the

| file, bypassing the file object's ``write`` method. As a result, tofile

| cannot be used with files objects supporting compression (e.g., GzipFile)

| or file-like objects that do not support ``fileno()`` (e.g., BytesIO).

|

| tolist(...)

| a.tolist()

|

| Return the array as an ``a.ndim``-levels deep nested list of Python scalars.

|

| Return a copy of the array data as a (nested) Python list.

| Data items are converted to the nearest compatible builtin Python type, via

| the `~numpy.ndarray.item` function.

|

| If ``a.ndim`` is 0, then since the depth of the nested list is 0, it will

| not be a list at all, but a simple Python scalar.

|

| Parameters

| ----------

| none

|

| Returns

| -------

| y : object, or list of object, or list of list of object, or ...

| The possibly nested list of array elements.

|

| Notes

| -----

| The array may be recreated via ``a = np.array(a.tolist())``, although this

| may sometimes lose precision.

|

| Examples

| --------

| For a 1D array, ``a.tolist()`` is almost the same as ``list(a)``:

|

| >>> a = np.array([1, 2])

| >>> list(a)

| [1, 2]

| >>> a.tolist()

| [1, 2]

|

| However, for a 2D array, ``tolist`` applies recursively:

|

| >>> a = np.array([[1, 2], [3, 4]])

| >>> list(a)

| [array([1, 2]), array([3, 4])]

| >>> a.tolist()

| [[1, 2], [3, 4]]

|

| The base case for this recursion is a 0D array:

|

| >>> a = np.array(1)

| >>> list(a)

| Traceback (most recent call last):

| ...

| TypeError: iteration over a 0-d array

| >>> a.tolist()

| 1

|

| tostring(...)

| a.tostring(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| trace(...)

| a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

|

| Return the sum along diagonals of the array.

|

| Refer to `numpy.trace` for full documentation.

|

| See Also

| --------

| numpy.trace : equivalent function

|

| transpose(...)

| a.transpose(\*axes)

|

| Returns a view of the array with axes transposed.

|

| For a 1-D array this has no effect, as a transposed vector is simply the

| same vector. To convert a 1-D array into a 2D column vector, an additional

| dimension must be added. `np.atleast2d(a).T` achieves this, as does

| `a[:, np.newaxis]`.

| For a 2-D array, this is a standard matrix transpose.

| For an n-D array, if axes are given, their order indicates how the

| axes are permuted (see Examples). If axes are not provided and

| ``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then

| ``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

|

| Parameters

| ----------

| axes : None, tuple of ints, or `n` ints

|

| \* None or no argument: reverses the order of the axes.

|

| \* tuple of ints: `i` in the `j`-th place in the tuple means `a`'s

| `i`-th axis becomes `a.transpose()`'s `j`-th axis.

|

| \* `n` ints: same as an n-tuple of the same ints (this form is

| intended simply as a "convenience" alternative to the tuple form)

|

| Returns

| -------

| out : ndarray

| View of `a`, with axes suitably permuted.

|

| See Also

| --------

| ndarray.T : Array property returning the array transposed.

| ndarray.reshape : Give a new shape to an array without changing its data.

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> a

| array([[1, 2],

| [3, 4]])

| >>> a.transpose()

| array([[1, 3],

| [2, 4]])

| >>> a.transpose((1, 0))

| array([[1, 3],

| [2, 4]])

| >>> a.transpose(1, 0)

| array([[1, 3],

| [2, 4]])

|

| var(...)

| a.var(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the variance of the array elements, along given axis.

|

| Refer to `numpy.var` for full documentation.

|

| See Also

| --------

| numpy.var : equivalent function

|

| view(...)

| a.view(dtype=None, type=None)

|

| New view of array with the same data.

|

| Parameters

| ----------

| dtype : data-type or ndarray sub-class, optional

| Data-type descriptor of the returned view, e.g., float32 or int16. The

| default, None, results in the view having the same data-type as `a`.

| This argument can also be specified as an ndarray sub-class, which

| then specifies the type of the returned object (this is equivalent to

| setting the ``type`` parameter).

| type : Python type, optional

| Type of the returned view, e.g., ndarray or matrix. Again, the

| default None results in type preservation.

|

| Notes

| -----

| ``a.view()`` is used two different ways:

|

| ``a.view(some\_dtype)`` or ``a.view(dtype=some\_dtype)`` constructs a view

| of the array's memory with a different data-type. This can cause a

| reinterpretation of the bytes of memory.

|

| ``a.view(ndarray\_subclass)`` or ``a.view(type=ndarray\_subclass)`` just

| returns an instance of `ndarray\_subclass` that looks at the same array

| (same shape, dtype, etc.) This does not cause a reinterpretation of the

| memory.

|

| For ``a.view(some\_dtype)``, if ``some\_dtype`` has a different number of

| bytes per entry than the previous dtype (for example, converting a

| regular array to a structured array), then the behavior of the view

| cannot be predicted just from the superficial appearance of ``a`` (shown

| by ``print(a)``). It also depends on exactly how ``a`` is stored in

| memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus

| defined as a slice or transpose, etc., the view may give different

| results.

|

|

| Examples

| --------

| >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

|

| Viewing array data using a different type and dtype:

|

| >>> y = x.view(dtype=np.int16, type=np.matrix)

| >>> y

| matrix([[513]], dtype=int16)

| >>> print(type(y))

| <class 'numpy.matrix'>

|

| Creating a view on a structured array so it can be used in calculations

|

| >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])

| >>> xv = x.view(dtype=np.int8).reshape(-1,2)

| >>> xv

| array([[1, 2],

| [3, 4]], dtype=int8)

| >>> xv.mean(0)

| array([2., 3.])

|

| Making changes to the view changes the underlying array

|

| >>> xv[0,1] = 20

| >>> x

| array([(1, 20), (3, 4)], dtype=[('a', 'i1'), ('b', 'i1')])

|

| Using a view to convert an array to a recarray:

|

| >>> z = x.view(np.recarray)

| >>> z.a

| array([1, 3], dtype=int8)

|

| Views share data:

|

| >>> x[0] = (9, 10)

| >>> z[0]

| (9, 10)

|

| Views that change the dtype size (bytes per entry) should normally be

| avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

|

| >>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)

| >>> y = x[:, 0:2]

| >>> y

| array([[1, 2],

| [4, 5]], dtype=int16)

| >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])

| Traceback (most recent call last):

| ...

| ValueError: To change to a dtype of a different size, the array must be C-contiguous

| >>> z = y.copy()

| >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])

| array([[(1, 2)],

| [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| T

| The transposed array.

|

| Same as ``self.transpose()``.

|

| Examples

| --------

| >>> x = np.array([[1.,2.],[3.,4.]])

| >>> x

| array([[ 1., 2.],

| [ 3., 4.]])

| >>> x.T

| array([[ 1., 3.],

| [ 2., 4.]])

| >>> x = np.array([1.,2.,3.,4.])

| >>> x

| array([ 1., 2., 3., 4.])

| >>> x.T

| array([ 1., 2., 3., 4.])

|

| See Also

| --------

| transpose

|

| \_\_array\_finalize\_\_

| None.

|

| \_\_array\_interface\_\_

| Array protocol: Python side.

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: C-struct side.

|

| base

| Base object if memory is from some other object.

|

| Examples

| --------

| The base of an array that owns its memory is None:

|

| >>> x = np.array([1,2,3,4])

| >>> x.base is None

| True

|

| Slicing creates a view, whose memory is shared with x:

|

| >>> y = x[2:]

| >>> y.base is x

| True

|

| ctypes

| An object to simplify the interaction of the array with the ctypes

| module.

|

| This attribute creates an object that makes it easier to use arrays

| when calling shared libraries with the ctypes module. The returned

| object has, among others, data, shape, and strides attributes (see

| Notes below) which themselves return ctypes objects that can be used

| as arguments to a shared library.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| c : Python object

| Possessing attributes data, shape, strides, etc.

|

| See Also

| --------

| numpy.ctypeslib

|

| Notes

| -----

| Below are the public attributes of this object which were documented

| in "Guide to NumPy" (we have omitted undocumented public attributes,

| as well as documented private attributes):

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.data

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.shape

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.strides

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.data\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.shape\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.strides\_as

| :noindex:

|

| If the ctypes module is not available, then the ctypes attribute

| of array objects still returns something useful, but ctypes objects

| are not returned and errors may be raised instead. In particular,

| the object will still have the ``as\_parameter`` attribute which will

| return an integer equal to the data attribute.

|

| Examples

| --------

| >>> import ctypes

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.ctypes.data

| 30439712

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long))

| <ctypes.LP\_c\_long object at 0x01F01300>

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long)).contents

| c\_long(0)

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_longlong)).contents

| c\_longlong(4294967296L)

| >>> x.ctypes.shape

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FFD580>

| >>> x.ctypes.shape\_as(ctypes.c\_long)

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides\_as(ctypes.c\_longlong)

| <numpy.core.\_internal.c\_longlong\_Array\_2 object at 0x01F01300>

|

| data

| Python buffer object pointing to the start of the array's data.

|

| dtype

| Data-type of the array's elements.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| d : numpy dtype object

|

| See Also

| --------

| numpy.dtype

|

| Examples

| --------

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.dtype

| dtype('int32')

| >>> type(x.dtype)

| <type 'numpy.dtype'>

|

| flags

| Information about the memory layout of the array.

|

| Attributes

| ----------

| C\_CONTIGUOUS (C)

| The data is in a single, C-style contiguous segment.

| F\_CONTIGUOUS (F)

| The data is in a single, Fortran-style contiguous segment.

| OWNDATA (O)

| The array owns the memory it uses or borrows it from another object.

| WRITEABLE (W)

| The data area can be written to. Setting this to False locks

| the data, making it read-only. A view (slice, etc.) inherits WRITEABLE

| from its base array at creation time, but a view of a writeable

| array may be subsequently locked while the base array remains writeable.

| (The opposite is not true, in that a view of a locked array may not

| be made writeable. However, currently, locking a base object does not

| lock any views that already reference it, so under that circumstance it

| is possible to alter the contents of a locked array via a previously

| created writeable view onto it.) Attempting to change a non-writeable

| array raises a RuntimeError exception.

| ALIGNED (A)

| The data and all elements are aligned appropriately for the hardware.

| WRITEBACKIFCOPY (X)

| This array is a copy of some other array. The C-API function

| PyArray\_ResolveWritebackIfCopy must be called before deallocating

| to the base array will be updated with the contents of this array.

| UPDATEIFCOPY (U)

| (Deprecated, use WRITEBACKIFCOPY) This array is a copy of some other array.

| When this array is

| deallocated, the base array will be updated with the contents of

| this array.

| FNC

| F\_CONTIGUOUS and not C\_CONTIGUOUS.

| FORC

| F\_CONTIGUOUS or C\_CONTIGUOUS (one-segment test).

| BEHAVED (B)

| ALIGNED and WRITEABLE.

| CARRAY (CA)

| BEHAVED and C\_CONTIGUOUS.

| FARRAY (FA)

| BEHAVED and F\_CONTIGUOUS and not C\_CONTIGUOUS.

|

| Notes

| -----

| The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),

| or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag

| names are only supported in dictionary access.

|

| Only the WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be

| changed by the user, via direct assignment to the attribute or dictionary

| entry, or by calling `ndarray.setflags`.

|

| The array flags cannot be set arbitrarily:

|

| - UPDATEIFCOPY can only be set ``False``.

| - WRITEBACKIFCOPY can only be set ``False``.

| - ALIGNED can only be set ``True`` if the data is truly aligned.

| - WRITEABLE can only be set ``True`` if the array owns its own memory

| or the ultimate owner of the memory exposes a writeable buffer

| interface or is a string.

|

| Arrays can be both C-style and Fortran-style contiguous simultaneously.

| This is clear for 1-dimensional arrays, but can also be true for higher

| dimensional arrays.

|

| Even for contiguous arrays a stride for a given dimension

| ``arr.strides[dim]`` may be \*arbitrary\* if ``arr.shape[dim] == 1``

| or the array has no elements.

| It does \*not\* generally hold that ``self.strides[-1] == self.itemsize``

| for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for

| Fortran-style contiguous arrays is true.

|

| flat

| A 1-D iterator over the array.

|

| This is a `numpy.flatiter` instance, which acts similarly to, but is not

| a subclass of, Python's built-in iterator object.

|

| See Also

| --------

| flatten : Return a copy of the array collapsed into one dimension.

|

| flatiter

|

| Examples

| --------

| >>> x = np.arange(1, 7).reshape(2, 3)

| >>> x

| array([[1, 2, 3],

| [4, 5, 6]])

| >>> x.flat[3]

| 4

| >>> x.T

| array([[1, 4],

| [2, 5],

| [3, 6]])

| >>> x.T.flat[3]

| 5

| >>> type(x.flat)

| <class 'numpy.flatiter'>

|

| An assignment example:

|

| >>> x.flat = 3; x

| array([[3, 3, 3],

| [3, 3, 3]])

| >>> x.flat[[1,4]] = 1; x

| array([[3, 1, 3],

| [3, 1, 3]])

|

| imag

| The imaginary part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.imag

| array([ 0. , 0.70710678])

| >>> x.imag.dtype

| dtype('float64')

|

| itemsize

| Length of one array element in bytes.

|

| Examples

| --------

| >>> x = np.array([1,2,3], dtype=np.float64)

| >>> x.itemsize

| 8

| >>> x = np.array([1,2,3], dtype=np.complex128)

| >>> x.itemsize

| 16

|

| nbytes

| Total bytes consumed by the elements of the array.

|

| Notes

| -----

| Does not include memory consumed by non-element attributes of the

| array object.

|

| Examples

| --------

| >>> x = np.zeros((3,5,2), dtype=np.complex128)

| >>> x.nbytes

| 480

| >>> np.prod(x.shape) \* x.itemsize

| 480

|

| ndim

| Number of array dimensions.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> x.ndim

| 1

| >>> y = np.zeros((2, 3, 4))

| >>> y.ndim

| 3

|

| real

| The real part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.real

| array([ 1. , 0.70710678])

| >>> x.real.dtype

| dtype('float64')

|

| See Also

| --------

| numpy.real : equivalent function

|

| shape

| Tuple of array dimensions.

|

| The shape property is usually used to get the current shape of an array,

| but may also be used to reshape the array in-place by assigning a tuple of

| array dimensions to it. As with `numpy.reshape`, one of the new shape

| dimensions can be -1, in which case its value is inferred from the size of

| the array and the remaining dimensions. Reshaping an array in-place will

| fail if a copy is required.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3, 4])

| >>> x.shape

| (4,)

| >>> y = np.zeros((2, 3, 4))

| >>> y.shape

| (2, 3, 4)

| >>> y.shape = (3, 8)

| >>> y

| array([[ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.]])

| >>> y.shape = (3, 6)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: total size of new array must be unchanged

| >>> np.zeros((4,2))[::2].shape = (-1,)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| AttributeError: incompatible shape for a non-contiguous array

|

| See Also

| --------

| numpy.reshape : similar function

| ndarray.reshape : similar method

|

| size

| Number of elements in the array.

|

| Equal to ``np.prod(a.shape)``, i.e., the product of the array's

| dimensions.

|

| Notes

| -----

| `a.size` returns a standard arbitrary precision Python integer. This

| may not be the case with other methods of obtaining the same value

| (like the suggested ``np.prod(a.shape)``, which returns an instance

| of ``np.int\_``), and may be relevant if the value is used further in

| calculations that may overflow a fixed size integer type.

|

| Examples

| --------

| >>> x = np.zeros((3, 5, 2), dtype=np.complex128)

| >>> x.size

| 30

| >>> np.prod(x.shape)

| 30

|

| strides

| Tuple of bytes to step in each dimension when traversing an array.

|

| The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`

| is::

|

| offset = sum(np.array(i) \* a.strides)

|

| A more detailed explanation of strides can be found in the

| "ndarray.rst" file in the NumPy reference guide.

|

| Notes

| -----

| Imagine an array of 32-bit integers (each 4 bytes)::

|

| x = np.array([[0, 1, 2, 3, 4],

| [5, 6, 7, 8, 9]], dtype=np.int32)

|

| This array is stored in memory as 40 bytes, one after the other

| (known as a contiguous block of memory). The strides of an array tell

| us how many bytes we have to skip in memory to move to the next position

| along a certain axis. For example, we have to skip 4 bytes (1 value) to

| move to the next column, but 20 bytes (5 values) to get to the same

| position in the next row. As such, the strides for the array `x` will be

| ``(20, 4)``.

|

| See Also

| --------

| numpy.lib.stride\_tricks.as\_strided

|

| Examples

| --------

| >>> y = np.reshape(np.arange(2\*3\*4), (2,3,4))

| >>> y

| array([[[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]],

| [[12, 13, 14, 15],

| [16, 17, 18, 19],

| [20, 21, 22, 23]]])

| >>> y.strides

| (48, 16, 4)

| >>> y[1,1,1]

| 17

| >>> offset=sum(y.strides \* np.array((1,1,1)))

| >>> offset/y.itemsize

| 17

|

| >>> x = np.reshape(np.arange(5\*6\*7\*8), (5,6,7,8)).transpose(2,3,1,0)

| >>> x.strides

| (32, 4, 224, 1344)

| >>> i = np.array([3,5,2,2])

| >>> offset = sum(i \* x.strides)

| >>> x[3,5,2,2]

| 813

| >>> offset / x.itemsize

| 813

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_hash\_\_ = None

class ndenumerate(builtins.object)

| ndenumerate(arr)

|

| Multidimensional index iterator.

|

| Return an iterator yielding pairs of array coordinates and values.

|

| Parameters

| ----------

| arr : ndarray

| Input array.

|

| See Also

| --------

| ndindex, flatiter

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> for index, x in np.ndenumerate(a):

| ... print(index, x)

| (0, 0) 1

| (0, 1) 2

| (1, 0) 3

| (1, 1) 4

|

| Methods defined here:

|

| \_\_init\_\_(self, arr)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_iter\_\_(self)

|

| \_\_next\_\_(self)

| Standard iterator method, returns the index tuple and array value.

|

| Returns

| -------

| coords : tuple of ints

| The indices of the current iteration.

| val : scalar

| The array element of the current iteration.

|

| next = \_\_next\_\_(self)

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class ndindex(builtins.object)

| ndindex(\*shape)

|

| An N-dimensional iterator object to index arrays.

|

| Given the shape of an array, an `ndindex` instance iterates over

| the N-dimensional index of the array. At each iteration a tuple

| of indices is returned, the last dimension is iterated over first.

|

| Parameters

| ----------

| `\*args` : ints

| The size of each dimension of the array.

|

| See Also

| --------

| ndenumerate, flatiter

|

| Examples

| --------

| >>> for index in np.ndindex(3, 2, 1):

| ... print(index)

| (0, 0, 0)

| (0, 1, 0)

| (1, 0, 0)

| (1, 1, 0)

| (2, 0, 0)

| (2, 1, 0)

|

| Methods defined here:

|

| \_\_init\_\_(self, \*shape)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_iter\_\_(self)

|

| \_\_next\_\_(self)

| Standard iterator method, updates the index and returns the index

| tuple.

|

| Returns

| -------

| val : tuple of ints

| Returns a tuple containing the indices of the current

| iteration.

|

| ndincr(self)

| Increment the multi-dimensional index by one.

|

| This method is for backward compatibility only: do not use.

|

| next = \_\_next\_\_(self)

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class nditer(builtins.object)

| Efficient multi-dimensional iterator object to iterate over arrays.

| To get started using this object, see the

| :ref:`introductory guide to array iteration <arrays.nditer>`.

|

| Parameters

| ----------

| op : ndarray or sequence of array\_like

| The array(s) to iterate over.

|

| flags : sequence of str, optional

| Flags to control the behavior of the iterator.

|

| \* ``buffered`` enables buffering when required.

| \* ``c\_index`` causes a C-order index to be tracked.

| \* ``f\_index`` causes a Fortran-order index to be tracked.

| \* ``multi\_index`` causes a multi-index, or a tuple of indices

| with one per iteration dimension, to be tracked.

| \* ``common\_dtype`` causes all the operands to be converted to

| a common data type, with copying or buffering as necessary.

| \* ``copy\_if\_overlap`` causes the iterator to determine if read

| operands have overlap with write operands, and make temporary

| copies as necessary to avoid overlap. False positives (needless

| copying) are possible in some cases.

| \* ``delay\_bufalloc`` delays allocation of the buffers until

| a reset() call is made. Allows ``allocate`` operands to

| be initialized before their values are copied into the buffers.

| \* ``external\_loop`` causes the ``values`` given to be

| one-dimensional arrays with multiple values instead of

| zero-dimensional arrays.

| \* ``grow\_inner`` allows the ``value`` array sizes to be made

| larger than the buffer size when both ``buffered`` and

| ``external\_loop`` is used.

| \* ``ranged`` allows the iterator to be restricted to a sub-range

| of the iterindex values.

| \* ``refs\_ok`` enables iteration of reference types, such as

| object arrays.

| \* ``reduce\_ok`` enables iteration of ``readwrite`` operands

| which are broadcasted, also known as reduction operands.

| \* ``zerosize\_ok`` allows `itersize` to be zero.

| op\_flags : list of list of str, optional

| This is a list of flags for each operand. At minimum, one of

| ``readonly``, ``readwrite``, or ``writeonly`` must be specified.

|

| \* ``readonly`` indicates the operand will only be read from.

| \* ``readwrite`` indicates the operand will be read from and written to.

| \* ``writeonly`` indicates the operand will only be written to.

| \* ``no\_broadcast`` prevents the operand from being broadcasted.

| \* ``contig`` forces the operand data to be contiguous.

| \* ``aligned`` forces the operand data to be aligned.

| \* ``nbo`` forces the operand data to be in native byte order.

| \* ``copy`` allows a temporary read-only copy if required.

| \* ``updateifcopy`` allows a temporary read-write copy if required.

| \* ``allocate`` causes the array to be allocated if it is None

| in the ``op`` parameter.

| \* ``no\_subtype`` prevents an ``allocate`` operand from using a subtype.

| \* ``arraymask`` indicates that this operand is the mask to use

| for selecting elements when writing to operands with the

| 'writemasked' flag set. The iterator does not enforce this,

| but when writing from a buffer back to the array, it only

| copies those elements indicated by this mask.

| \* ``writemasked`` indicates that only elements where the chosen

| ``arraymask`` operand is True will be written to.

| \* ``overlap\_assume\_elementwise`` can be used to mark operands that are

| accessed only in the iterator order, to allow less conservative

| copying when ``copy\_if\_overlap`` is present.

| op\_dtypes : dtype or tuple of dtype(s), optional

| The required data type(s) of the operands. If copying or buffering

| is enabled, the data will be converted to/from their original types.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the iteration order. 'C' means C order, 'F' means

| Fortran order, 'A' means 'F' order if all the arrays are Fortran

| contiguous, 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible. This also

| affects the element memory order of ``allocate`` operands, as they

| are allocated to be compatible with iteration order.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur when making a copy

| or buffering. Setting this to 'unsafe' is not recommended,

| as it can adversely affect accumulations.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| op\_axes : list of list of ints, optional

| If provided, is a list of ints or None for each operands.

| The list of axes for an operand is a mapping from the dimensions

| of the iterator to the dimensions of the operand. A value of

| -1 can be placed for entries, causing that dimension to be

| treated as `newaxis`.

| itershape : tuple of ints, optional

| The desired shape of the iterator. This allows ``allocate`` operands

| with a dimension mapped by op\_axes not corresponding to a dimension

| of a different operand to get a value not equal to 1 for that

| dimension.

| buffersize : int, optional

| When buffering is enabled, controls the size of the temporary

| buffers. Set to 0 for the default value.

|

| Attributes

| ----------

| dtypes : tuple of dtype(s)

| The data types of the values provided in `value`. This may be

| different from the operand data types if buffering is enabled.

| Valid only before the iterator is closed.

| finished : bool

| Whether the iteration over the operands is finished or not.

| has\_delayed\_bufalloc : bool

| If True, the iterator was created with the ``delay\_bufalloc`` flag,

| and no reset() function was called on it yet.

| has\_index : bool

| If True, the iterator was created with either the ``c\_index`` or

| the ``f\_index`` flag, and the property `index` can be used to

| retrieve it.

| has\_multi\_index : bool

| If True, the iterator was created with the ``multi\_index`` flag,

| and the property `multi\_index` can be used to retrieve it.

| index

| When the ``c\_index`` or ``f\_index`` flag was used, this property

| provides access to the index. Raises a ValueError if accessed

| and ``has\_index`` is False.

| iterationneedsapi : bool

| Whether iteration requires access to the Python API, for example

| if one of the operands is an object array.

| iterindex : int

| An index which matches the order of iteration.

| itersize : int

| Size of the iterator.

| itviews

| Structured view(s) of `operands` in memory, matching the reordered

| and optimized iterator access pattern. Valid only before the iterator

| is closed.

| multi\_index

| When the ``multi\_index`` flag was used, this property

| provides access to the index. Raises a ValueError if accessed

| accessed and ``has\_multi\_index`` is False.

| ndim : int

| The dimensions of the iterator.

| nop : int

| The number of iterator operands.

| operands : tuple of operand(s)

| The array(s) to be iterated over. Valid only before the iterator is

| closed.

| shape : tuple of ints

| Shape tuple, the shape of the iterator.

| value

| Value of ``operands`` at current iteration. Normally, this is a

| tuple of array scalars, but if the flag ``external\_loop`` is used,

| it is a tuple of one dimensional arrays.

|

| Notes

| -----

| `nditer` supersedes `flatiter`. The iterator implementation behind

| `nditer` is also exposed by the NumPy C API.

|

| The Python exposure supplies two iteration interfaces, one which follows

| the Python iterator protocol, and another which mirrors the C-style

| do-while pattern. The native Python approach is better in most cases, but

| if you need the coordinates or index of an iterator, use the C-style pattern.

|

| Examples

| --------

| Here is how we might write an ``iter\_add`` function, using the

| Python iterator protocol:

|

| >>> def iter\_add\_py(x, y, out=None):

| ... addop = np.add

| ... it = np.nditer([x, y, out], [],

| ... [['readonly'], ['readonly'], ['writeonly','allocate']])

| ... with it:

| ... for (a, b, c) in it:

| ... addop(a, b, out=c)

| ... return it.operands[2]

|

| Here is the same function, but following the C-style pattern:

|

| >>> def iter\_add(x, y, out=None):

| ... addop = np.add

| ... it = np.nditer([x, y, out], [],

| ... [['readonly'], ['readonly'], ['writeonly','allocate']])

| ... with it:

| ... while not it.finished:

| ... addop(it[0], it[1], out=it[2])

| ... it.iternext()

| ... return it.operands[2]

|

| Here is an example outer product function:

|

| >>> def outer\_it(x, y, out=None):

| ... mulop = np.multiply

| ... it = np.nditer([x, y, out], ['external\_loop'],

| ... [['readonly'], ['readonly'], ['writeonly', 'allocate']],

| ... op\_axes=[list(range(x.ndim)) + [-1] \* y.ndim,

| ... [-1] \* x.ndim + list(range(y.ndim)),

| ... None])

| ... with it:

| ... for (a, b, c) in it:

| ... mulop(a, b, out=c)

| ... return it.operands[2]

|

| >>> a = np.arange(2)+1

| >>> b = np.arange(3)+1

| >>> outer\_it(a,b)

| array([[1, 2, 3],

| [2, 4, 6]])

|

| Here is an example function which operates like a "lambda" ufunc:

|

| >>> def luf(lamdaexpr, \*args, \*\*kwargs):

| ... '''luf(lambdaexpr, op1, ..., opn, out=None, order='K', casting='safe', buffersize=0)'''

| ... nargs = len(args)

| ... op = (kwargs.get('out',None),) + args

| ... it = np.nditer(op, ['buffered','external\_loop'],

| ... [['writeonly','allocate','no\_broadcast']] +

| ... [['readonly','nbo','aligned']]\*nargs,

| ... order=kwargs.get('order','K'),

| ... casting=kwargs.get('casting','safe'),

| ... buffersize=kwargs.get('buffersize',0))

| ... while not it.finished:

| ... it[0] = lamdaexpr(\*it[1:])

| ... it.iternext()

| ... return it.operands[0]

|

| >>> a = np.arange(5)

| >>> b = np.ones(5)

| >>> luf(lambda i,j:i\*i + j/2, a, b)

| array([ 0.5, 1.5, 4.5, 9.5, 16.5])

|

| If operand flags `"writeonly"` or `"readwrite"` are used the operands may

| be views into the original data with the `WRITEBACKIFCOPY` flag. In this case

| nditer must be used as a context manager or the nditer.close

| method must be called before using the result. The temporary

| data will be written back to the original data when the `\_\_exit\_\_`

| function is called but not before:

|

| >>> a = np.arange(6, dtype='i4')[::-2]

| >>> with np.nditer(a, [],

| ... [['writeonly', 'updateifcopy']],

| ... casting='unsafe',

| ... op\_dtypes=[np.dtype('f4')]) as i:

| ... x = i.operands[0]

| ... x[:] = [-1, -2, -3]

| ... # a still unchanged here

| >>> a, x

| (array([-1, -2, -3], dtype=int32), array([-1., -2., -3.], dtype=float32))

|

| It is important to note that once the iterator is exited, dangling

| references (like `x` in the example) may or may not share data with

| the original data `a`. If writeback semantics were active, i.e. if

| `x.base.flags.writebackifcopy` is `True`, then exiting the iterator

| will sever the connection between `x` and `a`, writing to `x` will

| no longer write to `a`. If writeback semantics are not active, then

| `x.data` will still point at some part of `a.data`, and writing to

| one will affect the other.

|

| Methods defined here:

|

| \_\_copy\_\_(...)

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_enter\_\_(...)

|

| \_\_exit\_\_(...)

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| close(...)

| close()

|

| Resolve all writeback semantics in writeable operands.

|

| See Also

| --------

|

| :ref:`nditer-context-manager`

|

| copy(...)

| copy()

|

| Get a copy of the iterator in its current state.

|

| Examples

| --------

| >>> x = np.arange(10)

| >>> y = x + 1

| >>> it = np.nditer([x, y])

| >>> next(it)

| (array(0), array(1))

| >>> it2 = it.copy()

| >>> next(it2)

| (array(1), array(2))

|

| debug\_print(...)

| debug\_print()

|

| Print the current state of the `nditer` instance and debug info to stdout.

|

| enable\_external\_loop(...)

| enable\_external\_loop()

|

| When the "external\_loop" was not used during construction, but

| is desired, this modifies the iterator to behave as if the flag

| was specified.

|

| iternext(...)

| iternext()

|

| Check whether iterations are left, and perform a single internal iteration

| without returning the result. Used in the C-style pattern do-while

| pattern. For an example, see `nditer`.

|

| Returns

| -------

| iternext : bool

| Whether or not there are iterations left.

|

| remove\_axis(...)

| remove\_axis(i)

|

| Removes axis `i` from the iterator. Requires that the flag "multi\_index"

| be enabled.

|

| remove\_multi\_index(...)

| remove\_multi\_index()

|

| When the "multi\_index" flag was specified, this removes it, allowing

| the internal iteration structure to be optimized further.

|

| reset(...)

| reset()

|

| Reset the iterator to its initial state.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| dtypes

|

| finished

|

| has\_delayed\_bufalloc

|

| has\_index

|

| has\_multi\_index

|

| index

|

| iterationneedsapi

|

| iterindex

|

| iterrange

|

| itersize

|

| itviews

|

| multi\_index

|

| ndim

|

| nop

|

| operands

| operands[`Slice`]

|

| The array(s) to be iterated over. Valid only before the iterator is closed.

|

| shape

|

| value

class number(generic)

| Abstract base class of all numeric scalar types.

|

| Method resolution order:

| number

| generic

| builtins.object

|

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

object0 = class object\_(generic)

| Any Python object.

| Character code: ``'O'``.

|

| Method resolution order:

| object\_

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_call\_\_(self, /, \*args, \*\*kwargs)

| Call self as a function.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_iadd\_\_(self, value, /)

| Implement self+=value.

|

| \_\_imul\_\_(self, value, /)

| Implement self\*=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class object\_(generic)

| Any Python object.

| Character code: ``'O'``.

|

| Method resolution order:

| object\_

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_call\_\_(self, /, \*args, \*\*kwargs)

| Call self as a function.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_iadd\_\_(self, value, /)

| Implement self+=value.

|

| \_\_imul\_\_(self, value, /)

| Implement self\*=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class poly1d(builtins.object)

| poly1d(c\_or\_r, r=False, variable=None)

|

| A one-dimensional polynomial class.

|

| A convenience class, used to encapsulate "natural" operations on

| polynomials so that said operations may take on their customary

| form in code (see Examples).

|

| Parameters

| ----------

| c\_or\_r : array\_like

| The polynomial's coefficients, in decreasing powers, or if

| the value of the second parameter is True, the polynomial's

| roots (values where the polynomial evaluates to 0). For example,

| ``poly1d([1, 2, 3])`` returns an object that represents

| :math:`x^2 + 2x + 3`, whereas ``poly1d([1, 2, 3], True)`` returns

| one that represents :math:`(x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x -6`.

| r : bool, optional

| If True, `c\_or\_r` specifies the polynomial's roots; the default

| is False.

| variable : str, optional

| Changes the variable used when printing `p` from `x` to `variable`

| (see Examples).

|

| Examples

| --------

| Construct the polynomial :math:`x^2 + 2x + 3`:

|

| >>> p = np.poly1d([1, 2, 3])

| >>> print(np.poly1d(p))

| 2

| 1 x + 2 x + 3

|

| Evaluate the polynomial at :math:`x = 0.5`:

|

| >>> p(0.5)

| 4.25

|

| Find the roots:

|

| >>> p.r

| array([-1.+1.41421356j, -1.-1.41421356j])

| >>> p(p.r)

| array([ -4.44089210e-16+0.j, -4.44089210e-16+0.j]) # may vary

|

| These numbers in the previous line represent (0, 0) to machine precision

|

| Show the coefficients:

|

| >>> p.c

| array([1, 2, 3])

|

| Display the order (the leading zero-coefficients are removed):

|

| >>> p.order

| 2

|

| Show the coefficient of the k-th power in the polynomial

| (which is equivalent to ``p.c[-(i+1)]``):

|

| >>> p[1]

| 2

|

| Polynomials can be added, subtracted, multiplied, and divided

| (returns quotient and remainder):

|

| >>> p \* p

| poly1d([ 1, 4, 10, 12, 9])

|

| >>> (p\*\*3 + 4) / p

| (poly1d([ 1., 4., 10., 12., 9.]), poly1d([4.]))

|

| ``asarray(p)`` gives the coefficient array, so polynomials can be

| used in all functions that accept arrays:

|

| >>> p\*\*2 # square of polynomial

| poly1d([ 1, 4, 10, 12, 9])

|

| >>> np.square(p) # square of individual coefficients

| array([1, 4, 9])

|

| The variable used in the string representation of `p` can be modified,

| using the `variable` parameter:

|

| >>> p = np.poly1d([1,2,3], variable='z')

| >>> print(p)

| 2

| 1 z + 2 z + 3

|

| Construct a polynomial from its roots:

|

| >>> np.poly1d([1, 2], True)

| poly1d([ 1., -3., 2.])

|

| This is the same polynomial as obtained by:

|

| >>> np.poly1d([1, -1]) \* np.poly1d([1, -2])

| poly1d([ 1, -3, 2])

|

| Methods defined here:

|

| \_\_add\_\_(self, other)

|

| \_\_array\_\_(self, t=None)

|

| \_\_call\_\_(self, val)

| Call self as a function.

|

| \_\_div\_\_(self, other)

|

| \_\_eq\_\_(self, other)

| Return self==value.

|

| \_\_getitem\_\_(self, val)

|

| \_\_init\_\_(self, c\_or\_r, r=False, variable=None)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_iter\_\_(self)

|

| \_\_len\_\_(self)

|

| \_\_mul\_\_(self, other)

|

| \_\_ne\_\_(self, other)

| Return self!=value.

|

| \_\_neg\_\_(self)

|

| \_\_pos\_\_(self)

|

| \_\_pow\_\_(self, val)

|

| \_\_radd\_\_(self, other)

|

| \_\_rdiv\_\_(self, other)

|

| \_\_repr\_\_(self)

| Return repr(self).

|

| \_\_rmul\_\_(self, other)

|

| \_\_rsub\_\_(self, other)

|

| \_\_rtruediv\_\_ = \_\_rdiv\_\_(self, other)

|

| \_\_setitem\_\_(self, key, val)

|

| \_\_str\_\_(self)

| Return str(self).

|

| \_\_sub\_\_(self, other)

|

| \_\_truediv\_\_ = \_\_div\_\_(self, other)

|

| deriv(self, m=1)

| Return a derivative of this polynomial.

|

| Refer to `polyder` for full documentation.

|

| See Also

| --------

| polyder : equivalent function

|

| integ(self, m=1, k=0)

| Return an antiderivative (indefinite integral) of this polynomial.

|

| Refer to `polyint` for full documentation.

|

| See Also

| --------

| polyint : equivalent function

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| c

| The polynomial coefficients

|

| coef

| The polynomial coefficients

|

| coefficients

| The polynomial coefficients

|

| coeffs

| The polynomial coefficients

|

| o

| The order or degree of the polynomial

|

| order

| The order or degree of the polynomial

|

| r

| The roots of the polynomial, where self(x) == 0

|

| roots

| The roots of the polynomial, where self(x) == 0

|

| variable

| The name of the polynomial variable

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_hash\_\_ = None

class recarray(ndarray)

| recarray(shape, dtype=None, buf=None, offset=0, strides=None, formats=None, names=None, titles=None, byteorder=None, aligned=False, order='C')

|

| Construct an ndarray that allows field access using attributes.

|

| Arrays may have a data-types containing fields, analogous

| to columns in a spread sheet. An example is ``[(x, int), (y, float)]``,

| where each entry in the array is a pair of ``(int, float)``. Normally,

| these attributes are accessed using dictionary lookups such as ``arr['x']``

| and ``arr['y']``. Record arrays allow the fields to be accessed as members

| of the array, using ``arr.x`` and ``arr.y``.

|

| Parameters

| ----------

| shape : tuple

| Shape of output array.

| dtype : data-type, optional

| The desired data-type. By default, the data-type is determined

| from `formats`, `names`, `titles`, `aligned` and `byteorder`.

| formats : list of data-types, optional

| A list containing the data-types for the different columns, e.g.

| ``['i4', 'f8', 'i4']``. `formats` does \*not\* support the new

| convention of using types directly, i.e. ``(int, float, int)``.

| Note that `formats` must be a list, not a tuple.

| Given that `formats` is somewhat limited, we recommend specifying

| `dtype` instead.

| names : tuple of str, optional

| The name of each column, e.g. ``('x', 'y', 'z')``.

| buf : buffer, optional

| By default, a new array is created of the given shape and data-type.

| If `buf` is specified and is an object exposing the buffer interface,

| the array will use the memory from the existing buffer. In this case,

| the `offset` and `strides` keywords are available.

|

| Other Parameters

| ----------------

| titles : tuple of str, optional

| Aliases for column names. For example, if `names` were

| ``('x', 'y', 'z')`` and `titles` is

| ``('x\_coordinate', 'y\_coordinate', 'z\_coordinate')``, then

| ``arr['x']`` is equivalent to both ``arr.x`` and ``arr.x\_coordinate``.

| byteorder : {'<', '>', '='}, optional

| Byte-order for all fields.

| aligned : bool, optional

| Align the fields in memory as the C-compiler would.

| strides : tuple of ints, optional

| Buffer (`buf`) is interpreted according to these strides (strides

| define how many bytes each array element, row, column, etc.

| occupy in memory).

| offset : int, optional

| Start reading buffer (`buf`) from this offset onwards.

| order : {'C', 'F'}, optional

| Row-major (C-style) or column-major (Fortran-style) order.

|

| Returns

| -------

| rec : recarray

| Empty array of the given shape and type.

|

| See Also

| --------

| rec.fromrecords : Construct a record array from data.

| record : fundamental data-type for `recarray`.

| format\_parser : determine a data-type from formats, names, titles.

|

| Notes

| -----

| This constructor can be compared to ``empty``: it creates a new record

| array but does not fill it with data. To create a record array from data,

| use one of the following methods:

|

| 1. Create a standard ndarray and convert it to a record array,

| using ``arr.view(np.recarray)``

| 2. Use the `buf` keyword.

| 3. Use `np.rec.fromrecords`.

|

| Examples

| --------

| Create an array with two fields, ``x`` and ``y``:

|

| >>> x = np.array([(1.0, 2), (3.0, 4)], dtype=[('x', '<f8'), ('y', '<i8')])

| >>> x

| array([(1., 2), (3., 4)], dtype=[('x', '<f8'), ('y', '<i8')])

|

| >>> x['x']

| array([1., 3.])

|

| View the array as a record array:

|

| >>> x = x.view(np.recarray)

|

| >>> x.x

| array([1., 3.])

|

| >>> x.y

| array([2, 4])

|

| Create a new, empty record array:

|

| >>> np.recarray((2,),

| ... dtype=[('x', int), ('y', float), ('z', int)]) #doctest: +SKIP

| rec.array([(-1073741821, 1.2249118382103472e-301, 24547520),

| (3471280, 1.2134086255804012e-316, 0)],

| dtype=[('x', '<i4'), ('y', '<f8'), ('z', '<i4')])

|

| Method resolution order:

| recarray

| ndarray

| builtins.object

|

| Methods defined here:

|

| \_\_array\_finalize\_\_(self, obj)

| None.

|

| \_\_getattribute\_\_(self, attr)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, indx)

| Return self[key].

|

| \_\_repr\_\_(self)

| Return repr(self).

|

| \_\_setattr\_\_(self, attr, val)

| Implement setattr(self, name, value).

|

| field(self, attr, val=None)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(subtype, shape, dtype=None, buf=None, offset=0, strides=None, formats=None, names=None, titles=None, byteorder=None, aligned=False, order='C')

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from ndarray:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| a.\_\_array\_\_(|dtype) -> reference if type unchanged, copy otherwise.

|

| Returns either a new reference to self if dtype is not given or a new array

| of provided data type if dtype is different from the current dtype of the

| array.

|

| \_\_array\_function\_\_(...)

|

| \_\_array\_prepare\_\_(...)

| a.\_\_array\_prepare\_\_(obj) -> Object of same type as ndarray object obj.

|

| \_\_array\_ufunc\_\_(...)

|

| \_\_array\_wrap\_\_(...)

| a.\_\_array\_wrap\_\_(obj) -> Object of same type as ndarray object a.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_copy\_\_(...)

| a.\_\_copy\_\_()

|

| Used if :func:`copy.copy` is called on an array. Returns a copy of the array.

|

| Equivalent to ``a.copy(order='K')``.

|

| \_\_deepcopy\_\_(...)

| a.\_\_deepcopy\_\_(memo, /) -> Deep copy of array.

|

| Used if :func:`copy.deepcopy` is called on an array.

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| Default object formatter.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_iadd\_\_(self, value, /)

| Return self+=value.

|

| \_\_iand\_\_(self, value, /)

| Return self&=value.

|

| \_\_ifloordiv\_\_(self, value, /)

| Return self//=value.

|

| \_\_ilshift\_\_(self, value, /)

| Return self<<=value.

|

| \_\_imatmul\_\_(self, value, /)

| Return self@=value.

|

| \_\_imod\_\_(self, value, /)

| Return self%=value.

|

| \_\_imul\_\_(self, value, /)

| Return self\*=value.

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_ior\_\_(self, value, /)

| Return self|=value.

|

| \_\_ipow\_\_(self, value, /)

| Return self\*\*=value.

|

| \_\_irshift\_\_(self, value, /)

| Return self>>=value.

|

| \_\_isub\_\_(self, value, /)

| Return self-=value.

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_itruediv\_\_(self, value, /)

| Return self/=value.

|

| \_\_ixor\_\_(self, value, /)

| Return self^=value.

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_matmul\_\_(self, value, /)

| Return self@value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| a.\_\_reduce\_\_()

|

| For pickling.

|

| \_\_reduce\_ex\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmatmul\_\_(self, value, /)

| Return value@self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_setstate\_\_(...)

| a.\_\_setstate\_\_(state, /)

|

| For unpickling.

|

| The `state` argument must be a sequence that contains the following

| elements:

|

| Parameters

| ----------

| version : int

| optional pickle version. If omitted defaults to 0.

| shape : tuple

| dtype : data-type

| isFortran : bool

| rawdata : string or list

| a binary string with the data (or a list if 'a' is an object array)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| a.all(axis=None, out=None, keepdims=False)

|

| Returns True if all elements evaluate to True.

|

| Refer to `numpy.all` for full documentation.

|

| See Also

| --------

| numpy.all : equivalent function

|

| any(...)

| a.any(axis=None, out=None, keepdims=False)

|

| Returns True if any of the elements of `a` evaluate to True.

|

| Refer to `numpy.any` for full documentation.

|

| See Also

| --------

| numpy.any : equivalent function

|

| argmax(...)

| a.argmax(axis=None, out=None)

|

| Return indices of the maximum values along the given axis.

|

| Refer to `numpy.argmax` for full documentation.

|

| See Also

| --------

| numpy.argmax : equivalent function

|

| argmin(...)

| a.argmin(axis=None, out=None)

|

| Return indices of the minimum values along the given axis of `a`.

|

| Refer to `numpy.argmin` for detailed documentation.

|

| See Also

| --------

| numpy.argmin : equivalent function

|

| argpartition(...)

| a.argpartition(kth, axis=-1, kind='introselect', order=None)

|

| Returns the indices that would partition this array.

|

| Refer to `numpy.argpartition` for full documentation.

|

| .. versionadded:: 1.8.0

|

| See Also

| --------

| numpy.argpartition : equivalent function

|

| argsort(...)

| a.argsort(axis=-1, kind=None, order=None)

|

| Returns the indices that would sort this array.

|

| Refer to `numpy.argsort` for full documentation.

|

| See Also

| --------

| numpy.argsort : equivalent function

|

| astype(...)

| a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

|

| Copy of the array, cast to a specified type.

|

| Parameters

| ----------

| dtype : str or dtype

| Typecode or data-type to which the array is cast.

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout order of the result.

| 'C' means C order, 'F' means Fortran order, 'A'

| means 'F' order if all the arrays are Fortran contiguous,

| 'C' order otherwise, and 'K' means as close to the

| order the array elements appear in memory as possible.

| Default is 'K'.

| casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

| Controls what kind of data casting may occur. Defaults to 'unsafe'

| for backwards compatibility.

|

| \* 'no' means the data types should not be cast at all.

| \* 'equiv' means only byte-order changes are allowed.

| \* 'safe' means only casts which can preserve values are allowed.

| \* 'same\_kind' means only safe casts or casts within a kind,

| like float64 to float32, are allowed.

| \* 'unsafe' means any data conversions may be done.

| subok : bool, optional

| If True, then sub-classes will be passed-through (default), otherwise

| the returned array will be forced to be a base-class array.

| copy : bool, optional

| By default, astype always returns a newly allocated array. If this

| is set to false, and the `dtype`, `order`, and `subok`

| requirements are satisfied, the input array is returned instead

| of a copy.

|

| Returns

| -------

| arr\_t : ndarray

| Unless `copy` is False and the other conditions for returning the input

| array are satisfied (see description for `copy` input parameter), `arr\_t`

| is a new array of the same shape as the input array, with dtype, order

| given by `dtype`, `order`.

|

| Notes

| -----

| .. versionchanged:: 1.17.0

| Casting between a simple data type and a structured one is possible only

| for "unsafe" casting. Casting to multiple fields is allowed, but

| casting from multiple fields is not.

|

| .. versionchanged:: 1.9.0

| Casting from numeric to string types in 'safe' casting mode requires

| that the string dtype length is long enough to store the max

| integer/float value converted.

|

| Raises

| ------

| ComplexWarning

| When casting from complex to float or int. To avoid this,

| one should use ``a.real.astype(t)``.

|

| Examples

| --------

| >>> x = np.array([1, 2, 2.5])

| >>> x

| array([1. , 2. , 2.5])

|

| >>> x.astype(int)

| array([1, 2, 2])

|

| byteswap(...)

| a.byteswap(inplace=False)

|

| Swap the bytes of the array elements

|

| Toggle between low-endian and big-endian data representation by

| returning a byteswapped array, optionally swapped in-place.

|

| Parameters

| ----------

| inplace : bool, optional

| If ``True``, swap bytes in-place, default is ``False``.

|

| Returns

| -------

| out : ndarray

| The byteswapped array. If `inplace` is ``True``, this is

| a view to self.

|

| Examples

| --------

| >>> A = np.array([1, 256, 8755], dtype=np.int16)

| >>> list(map(hex, A))

| ['0x1', '0x100', '0x2233']

| >>> A.byteswap(inplace=True)

| array([ 256, 1, 13090], dtype=int16)

| >>> list(map(hex, A))

| ['0x100', '0x1', '0x3322']

|

| Arrays of strings are not swapped

|

| >>> A = np.array(['ceg', 'fac'])

| >>> A.byteswap()

| Traceback (most recent call last):

| ...

| UnicodeDecodeError: ...

|

| choose(...)

| a.choose(choices, out=None, mode='raise')

|

| Use an index array to construct a new array from a set of choices.

|

| Refer to `numpy.choose` for full documentation.

|

| See Also

| --------

| numpy.choose : equivalent function

|

| clip(...)

| a.clip(min=None, max=None, out=None, \*\*kwargs)

|

| Return an array whose values are limited to ``[min, max]``.

| One of max or min must be given.

|

| Refer to `numpy.clip` for full documentation.

|

| See Also

| --------

| numpy.clip : equivalent function

|

| compress(...)

| a.compress(condition, axis=None, out=None)

|

| Return selected slices of this array along given axis.

|

| Refer to `numpy.compress` for full documentation.

|

| See Also

| --------

| numpy.compress : equivalent function

|

| conj(...)

| a.conj()

|

| Complex-conjugate all elements.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| conjugate(...)

| a.conjugate()

|

| Return the complex conjugate, element-wise.

|

| Refer to `numpy.conjugate` for full documentation.

|

| See Also

| --------

| numpy.conjugate : equivalent function

|

| copy(...)

| a.copy(order='C')

|

| Return a copy of the array.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| Controls the memory layout of the copy. 'C' means C-order,

| 'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

| 'C' otherwise. 'K' means match the layout of `a` as closely

| as possible. (Note that this function and :func:`numpy.copy` are very

| similar, but have different default values for their order=

| arguments.)

|

| See also

| --------

| numpy.copy

| numpy.copyto

|

| Examples

| --------

| >>> x = np.array([[1,2,3],[4,5,6]], order='F')

|

| >>> y = x.copy()

|

| >>> x.fill(0)

|

| >>> x

| array([[0, 0, 0],

| [0, 0, 0]])

|

| >>> y

| array([[1, 2, 3],

| [4, 5, 6]])

|

| >>> y.flags['C\_CONTIGUOUS']

| True

|

| cumprod(...)

| a.cumprod(axis=None, dtype=None, out=None)

|

| Return the cumulative product of the elements along the given axis.

|

| Refer to `numpy.cumprod` for full documentation.

|

| See Also

| --------

| numpy.cumprod : equivalent function

|

| cumsum(...)

| a.cumsum(axis=None, dtype=None, out=None)

|

| Return the cumulative sum of the elements along the given axis.

|

| Refer to `numpy.cumsum` for full documentation.

|

| See Also

| --------

| numpy.cumsum : equivalent function

|

| diagonal(...)

| a.diagonal(offset=0, axis1=0, axis2=1)

|

| Return specified diagonals. In NumPy 1.9 the returned array is a

| read-only view instead of a copy as in previous NumPy versions. In

| a future version the read-only restriction will be removed.

|

| Refer to :func:`numpy.diagonal` for full documentation.

|

| See Also

| --------

| numpy.diagonal : equivalent function

|

| dot(...)

| a.dot(b, out=None)

|

| Dot product of two arrays.

|

| Refer to `numpy.dot` for full documentation.

|

| See Also

| --------

| numpy.dot : equivalent function

|

| Examples

| --------

| >>> a = np.eye(2)

| >>> b = np.ones((2, 2)) \* 2

| >>> a.dot(b)

| array([[2., 2.],

| [2., 2.]])

|

| This array method can be conveniently chained:

|

| >>> a.dot(b).dot(b)

| array([[8., 8.],

| [8., 8.]])

|

| dump(...)

| a.dump(file)

|

| Dump a pickle of the array to the specified file.

| The array can be read back with pickle.load or numpy.load.

|

| Parameters

| ----------

| file : str or Path

| A string naming the dump file.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| dumps(...)

| a.dumps()

|

| Returns the pickle of the array as a string.

| pickle.loads or numpy.loads will convert the string back to an array.

|

| Parameters

| ----------

| None

|

| fill(...)

| a.fill(value)

|

| Fill the array with a scalar value.

|

| Parameters

| ----------

| value : scalar

| All elements of `a` will be assigned this value.

|

| Examples

| --------

| >>> a = np.array([1, 2])

| >>> a.fill(0)

| >>> a

| array([0, 0])

| >>> a = np.empty(2)

| >>> a.fill(1)

| >>> a

| array([1., 1.])

|

| flatten(...)

| a.flatten(order='C')

|

| Return a copy of the array collapsed into one dimension.

|

| Parameters

| ----------

| order : {'C', 'F', 'A', 'K'}, optional

| 'C' means to flatten in row-major (C-style) order.

| 'F' means to flatten in column-major (Fortran-

| style) order. 'A' means to flatten in column-major

| order if `a` is Fortran \*contiguous\* in memory,

| row-major order otherwise. 'K' means to flatten

| `a` in the order the elements occur in memory.

| The default is 'C'.

|

| Returns

| -------

| y : ndarray

| A copy of the input array, flattened to one dimension.

|

| See Also

| --------

| ravel : Return a flattened array.

| flat : A 1-D flat iterator over the array.

|

| Examples

| --------

| >>> a = np.array([[1,2], [3,4]])

| >>> a.flatten()

| array([1, 2, 3, 4])

| >>> a.flatten('F')

| array([1, 3, 2, 4])

|

| getfield(...)

| a.getfield(dtype, offset=0)

|

| Returns a field of the given array as a certain type.

|

| A field is a view of the array data with a given data-type. The values in

| the view are determined by the given type and the offset into the current

| array in bytes. The offset needs to be such that the view dtype fits in the

| array dtype; for example an array of dtype complex128 has 16-byte elements.

| If taking a view with a 32-bit integer (4 bytes), the offset needs to be

| between 0 and 12 bytes.

|

| Parameters

| ----------

| dtype : str or dtype

| The data type of the view. The dtype size of the view can not be larger

| than that of the array itself.

| offset : int

| Number of bytes to skip before beginning the element view.

|

| Examples

| --------

| >>> x = np.diag([1.+1.j]\*2)

| >>> x[1, 1] = 2 + 4.j

| >>> x

| array([[1.+1.j, 0.+0.j],

| [0.+0.j, 2.+4.j]])

| >>> x.getfield(np.float64)

| array([[1., 0.],

| [0., 2.]])

|

| By choosing an offset of 8 bytes we can select the complex part of the

| array for our view:

|

| >>> x.getfield(np.float64, offset=8)

| array([[1., 0.],

| [0., 4.]])

|

| item(...)

| a.item(\*args)

|

| Copy an element of an array to a standard Python scalar and return it.

|

| Parameters

| ----------

| \\*args : Arguments (variable number and type)

|

| \* none: in this case, the method only works for arrays

| with one element (`a.size == 1`), which element is

| copied into a standard Python scalar object and returned.

|

| \* int\_type: this argument is interpreted as a flat index into

| the array, specifying which element to copy and return.

|

| \* tuple of int\_types: functions as does a single int\_type argument,

| except that the argument is interpreted as an nd-index into the

| array.

|

| Returns

| -------

| z : Standard Python scalar object

| A copy of the specified element of the array as a suitable

| Python scalar

|

| Notes

| -----

| When the data type of `a` is longdouble or clongdouble, item() returns

| a scalar array object because there is no available Python scalar that

| would not lose information. Void arrays return a buffer object for item(),

| unless fields are defined, in which case a tuple is returned.

|

| `item` is very similar to a[args], except, instead of an array scalar,

| a standard Python scalar is returned. This can be useful for speeding up

| access to elements of the array and doing arithmetic on elements of the

| array using Python's optimized math.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.item(3)

| 1

| >>> x.item(7)

| 0

| >>> x.item((0, 1))

| 2

| >>> x.item((2, 2))

| 1

|

| itemset(...)

| a.itemset(\*args)

|

| Insert scalar into an array (scalar is cast to array's dtype, if possible)

|

| There must be at least 1 argument, and define the last argument

| as \*item\*. Then, ``a.itemset(\*args)`` is equivalent to but faster

| than ``a[args] = item``. The item should be a scalar value and `args`

| must select a single item in the array `a`.

|

| Parameters

| ----------

| \\*args : Arguments

| If one argument: a scalar, only used in case `a` is of size 1.

| If two arguments: the last argument is the value to be set

| and must be a scalar, the first argument specifies a single array

| element location. It is either an int or a tuple.

|

| Notes

| -----

| Compared to indexing syntax, `itemset` provides some speed increase

| for placing a scalar into a particular location in an `ndarray`,

| if you must do this. However, generally this is discouraged:

| among other problems, it complicates the appearance of the code.

| Also, when using `itemset` (and `item`) inside a loop, be sure

| to assign the methods to a local variable to avoid the attribute

| look-up at each loop iteration.

|

| Examples

| --------

| >>> np.random.seed(123)

| >>> x = np.random.randint(9, size=(3, 3))

| >>> x

| array([[2, 2, 6],

| [1, 3, 6],

| [1, 0, 1]])

| >>> x.itemset(4, 0)

| >>> x.itemset((2, 2), 9)

| >>> x

| array([[2, 2, 6],

| [1, 0, 6],

| [1, 0, 9]])

|

| max(...)

| a.max(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the maximum along a given axis.

|

| Refer to `numpy.amax` for full documentation.

|

| See Also

| --------

| numpy.amax : equivalent function

|

| mean(...)

| a.mean(axis=None, dtype=None, out=None, keepdims=False)

|

| Returns the average of the array elements along given axis.

|

| Refer to `numpy.mean` for full documentation.

|

| See Also

| --------

| numpy.mean : equivalent function

|

| min(...)

| a.min(axis=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Return the minimum along a given axis.

|

| Refer to `numpy.amin` for full documentation.

|

| See Also

| --------

| numpy.amin : equivalent function

|

| newbyteorder(...)

| arr.newbyteorder(new\_order='S')

|

| Return the array with the same data viewed with a different byte order.

|

| Equivalent to::

|

| arr.view(arr.dtype.newbytorder(new\_order))

|

| Changes are also made in all fields and sub-arrays of the array data

| type.

|

|

|

| Parameters

| ----------

| new\_order : string, optional

| Byte order to force; a value from the byte order specifications

| below. `new\_order` codes can be any of:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_arr : array

| New array object with the dtype reflecting given change to the

| byte order.

|

| nonzero(...)

| a.nonzero()

|

| Return the indices of the elements that are non-zero.

|

| Refer to `numpy.nonzero` for full documentation.

|

| See Also

| --------

| numpy.nonzero : equivalent function

|

| partition(...)

| a.partition(kth, axis=-1, kind='introselect', order=None)

|

| Rearranges the elements in the array in such a way that the value of the

| element in kth position is in the position it would be in a sorted array.

| All elements smaller than the kth element are moved before this element and

| all equal or greater are moved behind it. The ordering of the elements in

| the two partitions is undefined.

|

| .. versionadded:: 1.8.0

|

| Parameters

| ----------

| kth : int or sequence of ints

| Element index to partition by. The kth element value will be in its

| final sorted position and all smaller elements will be moved before it

| and all equal or greater elements behind it.

| The order of all elements in the partitions is undefined.

| If provided with a sequence of kth it will partition all elements

| indexed by kth of them into their sorted position at once.

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'introselect'}, optional

| Selection algorithm. Default is 'introselect'.

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need to be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.partition : Return a parititioned copy of an array.

| argpartition : Indirect partition.

| sort : Full sort.

|

| Notes

| -----

| See ``np.partition`` for notes on the different algorithms.

|

| Examples

| --------

| >>> a = np.array([3, 4, 2, 1])

| >>> a.partition(3)

| >>> a

| array([2, 1, 3, 4])

|

| >>> a.partition((1, 3))

| >>> a

| array([1, 2, 3, 4])

|

| prod(...)

| a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True)

|

| Return the product of the array elements over the given axis

|

| Refer to `numpy.prod` for full documentation.

|

| See Also

| --------

| numpy.prod : equivalent function

|

| ptp(...)

| a.ptp(axis=None, out=None, keepdims=False)

|

| Peak to peak (maximum - minimum) value along a given axis.

|

| Refer to `numpy.ptp` for full documentation.

|

| See Also

| --------

| numpy.ptp : equivalent function

|

| put(...)

| a.put(indices, values, mode='raise')

|

| Set ``a.flat[n] = values[n]`` for all `n` in indices.

|

| Refer to `numpy.put` for full documentation.

|

| See Also

| --------

| numpy.put : equivalent function

|

| ravel(...)

| a.ravel([order])

|

| Return a flattened array.

|

| Refer to `numpy.ravel` for full documentation.

|

| See Also

| --------

| numpy.ravel : equivalent function

|

| ndarray.flat : a flat iterator on the array.

|

| repeat(...)

| a.repeat(repeats, axis=None)

|

| Repeat elements of an array.

|

| Refer to `numpy.repeat` for full documentation.

|

| See Also

| --------

| numpy.repeat : equivalent function

|

| reshape(...)

| a.reshape(shape, order='C')

|

| Returns an array containing the same data with a new shape.

|

| Refer to `numpy.reshape` for full documentation.

|

| See Also

| --------

| numpy.reshape : equivalent function

|

| Notes

| -----

| Unlike the free function `numpy.reshape`, this method on `ndarray` allows

| the elements of the shape parameter to be passed in as separate arguments.

| For example, ``a.reshape(10, 11)`` is equivalent to

| ``a.reshape((10, 11))``.

|

| resize(...)

| a.resize(new\_shape, refcheck=True)

|

| Change shape and size of array in-place.

|

| Parameters

| ----------

| new\_shape : tuple of ints, or `n` ints

| Shape of resized array.

| refcheck : bool, optional

| If False, reference count will not be checked. Default is True.

|

| Returns

| -------

| None

|

| Raises

| ------

| ValueError

| If `a` does not own its own data or references or views to it exist,

| and the data memory must be changed.

| PyPy only: will always raise if the data memory must be changed, since

| there is no reliable way to determine if references or views to it

| exist.

|

| SystemError

| If the `order` keyword argument is specified. This behaviour is a

| bug in NumPy.

|

| See Also

| --------

| resize : Return a new array with the specified shape.

|

| Notes

| -----

| This reallocates space for the data area if necessary.

|

| Only contiguous arrays (data elements consecutive in memory) can be

| resized.

|

| The purpose of the reference count check is to make sure you

| do not use this array as a buffer for another Python object and then

| reallocate the memory. However, reference counts can increase in

| other ways so if you are sure that you have not shared the memory

| for this array with another Python object, then you may safely set

| `refcheck` to False.

|

| Examples

| --------

| Shrinking an array: array is flattened (in the order that the data are

| stored in memory), resized, and reshaped:

|

| >>> a = np.array([[0, 1], [2, 3]], order='C')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [1]])

|

| >>> a = np.array([[0, 1], [2, 3]], order='F')

| >>> a.resize((2, 1))

| >>> a

| array([[0],

| [2]])

|

| Enlarging an array: as above, but missing entries are filled with zeros:

|

| >>> b = np.array([[0, 1], [2, 3]])

| >>> b.resize(2, 3) # new\_shape parameter doesn't have to be a tuple

| >>> b

| array([[0, 1, 2],

| [3, 0, 0]])

|

| Referencing an array prevents resizing...

|

| >>> c = a

| >>> a.resize((1, 1))

| Traceback (most recent call last):

| ...

| ValueError: cannot resize an array that references or is referenced ...

|

| Unless `refcheck` is False:

|

| >>> a.resize((1, 1), refcheck=False)

| >>> a

| array([[0]])

| >>> c

| array([[0]])

|

| round(...)

| a.round(decimals=0, out=None)

|

| Return `a` with each element rounded to the given number of decimals.

|

| Refer to `numpy.around` for full documentation.

|

| See Also

| --------

| numpy.around : equivalent function

|

| searchsorted(...)

| a.searchsorted(v, side='left', sorter=None)

|

| Find indices where elements of v should be inserted in a to maintain order.

|

| For full documentation, see `numpy.searchsorted`

|

| See Also

| --------

| numpy.searchsorted : equivalent function

|

| setfield(...)

| a.setfield(val, dtype, offset=0)

|

| Put a value into a specified place in a field defined by a data-type.

|

| Place `val` into `a`'s field defined by `dtype` and beginning `offset`

| bytes into the field.

|

| Parameters

| ----------

| val : object

| Value to be placed in field.

| dtype : dtype object

| Data-type of the field in which to place `val`.

| offset : int, optional

| The number of bytes into the field at which to place `val`.

|

| Returns

| -------

| None

|

| See Also

| --------

| getfield

|

| Examples

| --------

| >>> x = np.eye(3)

| >>> x.getfield(np.float64)

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

| >>> x.setfield(3, np.int32)

| >>> x.getfield(np.int32)

| array([[3, 3, 3],

| [3, 3, 3],

| [3, 3, 3]], dtype=int32)

| >>> x

| array([[1.0e+000, 1.5e-323, 1.5e-323],

| [1.5e-323, 1.0e+000, 1.5e-323],

| [1.5e-323, 1.5e-323, 1.0e+000]])

| >>> x.setfield(np.eye(3), np.int32)

| >>> x

| array([[1., 0., 0.],

| [0., 1., 0.],

| [0., 0., 1.]])

|

| setflags(...)

| a.setflags(write=None, align=None, uic=None)

|

| Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY),

| respectively.

|

| These Boolean-valued flags affect how numpy interprets the memory

| area used by `a` (see Notes below). The ALIGNED flag can only

| be set to True if the data is actually aligned according to the type.

| The WRITEBACKIFCOPY and (deprecated) UPDATEIFCOPY flags can never be set

| to True. The flag WRITEABLE can only be set to True if the array owns its

| own memory, or the ultimate owner of the memory exposes a writeable buffer

| interface, or is a string. (The exception for string is made so that

| unpickling can be done without copying memory.)

|

| Parameters

| ----------

| write : bool, optional

| Describes whether or not `a` can be written to.

| align : bool, optional

| Describes whether or not `a` is aligned properly for its type.

| uic : bool, optional

| Describes whether or not `a` is a copy of another "base" array.

|

| Notes

| -----

| Array flags provide information about how the memory area used

| for the array is to be interpreted. There are 7 Boolean flags

| in use, only four of which can be changed by the user:

| WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED.

|

| WRITEABLE (W) the data area can be written to;

|

| ALIGNED (A) the data and strides are aligned appropriately for the hardware

| (as determined by the compiler);

|

| UPDATEIFCOPY (U) (deprecated), replaced by WRITEBACKIFCOPY;

|

| WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced

| by .base). When the C-API function PyArray\_ResolveWritebackIfCopy is

| called, the base array will be updated with the contents of this array.

|

| All flags can be accessed using the single (upper case) letter as well

| as the full name.

|

| Examples

| --------

| >>> y = np.array([[3, 1, 7],

| ... [2, 0, 0],

| ... [8, 5, 9]])

| >>> y

| array([[3, 1, 7],

| [2, 0, 0],

| [8, 5, 9]])

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : True

| ALIGNED : True

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(write=0, align=0)

| >>> y.flags

| C\_CONTIGUOUS : True

| F\_CONTIGUOUS : False

| OWNDATA : True

| WRITEABLE : False

| ALIGNED : False

| WRITEBACKIFCOPY : False

| UPDATEIFCOPY : False

| >>> y.setflags(uic=1)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: cannot set WRITEBACKIFCOPY flag to True

|

| sort(...)

| a.sort(axis=-1, kind=None, order=None)

|

| Sort an array in-place. Refer to `numpy.sort` for full documentation.

|

| Parameters

| ----------

| axis : int, optional

| Axis along which to sort. Default is -1, which means sort along the

| last axis.

| kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

| Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

| and 'mergesort' use timsort under the covers and, in general, the

| actual implementation will vary with datatype. The 'mergesort' option

| is retained for backwards compatibility.

|

| .. versionchanged:: 1.15.0.

| The 'stable' option was added.

|

| order : str or list of str, optional

| When `a` is an array with fields defined, this argument specifies

| which fields to compare first, second, etc. A single field can

| be specified as a string, and not all fields need be specified,

| but unspecified fields will still be used, in the order in which

| they come up in the dtype, to break ties.

|

| See Also

| --------

| numpy.sort : Return a sorted copy of an array.

| argsort : Indirect sort.

| lexsort : Indirect stable sort on multiple keys.

| searchsorted : Find elements in sorted array.

| partition: Partial sort.

|

| Notes

| -----

| See `numpy.sort` for notes on the different sorting algorithms.

|

| Examples

| --------

| >>> a = np.array([[1,4], [3,1]])

| >>> a.sort(axis=1)

| >>> a

| array([[1, 4],

| [1, 3]])

| >>> a.sort(axis=0)

| >>> a

| array([[1, 3],

| [1, 4]])

|

| Use the `order` keyword to specify a field to use when sorting a

| structured array:

|

| >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])

| >>> a.sort(order='y')

| >>> a

| array([(b'c', 1), (b'a', 2)],

| dtype=[('x', 'S1'), ('y', '<i8')])

|

| squeeze(...)

| a.squeeze(axis=None)

|

| Remove single-dimensional entries from the shape of `a`.

|

| Refer to `numpy.squeeze` for full documentation.

|

| See Also

| --------

| numpy.squeeze : equivalent function

|

| std(...)

| a.std(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the standard deviation of the array elements along given axis.

|

| Refer to `numpy.std` for full documentation.

|

| See Also

| --------

| numpy.std : equivalent function

|

| sum(...)

| a.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True)

|

| Return the sum of the array elements over the given axis.

|

| Refer to `numpy.sum` for full documentation.

|

| See Also

| --------

| numpy.sum : equivalent function

|

| swapaxes(...)

| a.swapaxes(axis1, axis2)

|

| Return a view of the array with `axis1` and `axis2` interchanged.

|

| Refer to `numpy.swapaxes` for full documentation.

|

| See Also

| --------

| numpy.swapaxes : equivalent function

|

| take(...)

| a.take(indices, axis=None, out=None, mode='raise')

|

| Return an array formed from the elements of `a` at the given indices.

|

| Refer to `numpy.take` for full documentation.

|

| See Also

| --------

| numpy.take : equivalent function

|

| tobytes(...)

| a.tobytes(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| .. versionadded:: 1.9.0

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| tofile(...)

| a.tofile(fid, sep="", format="%s")

|

| Write array to a file as text or binary (default).

|

| Data is always written in 'C' order, independent of the order of `a`.

| The data produced by this method can be recovered using the function

| fromfile().

|

| Parameters

| ----------

| fid : file or str or Path

| An open file object, or a string containing a filename.

|

| .. versionchanged:: 1.17.0

| `pathlib.Path` objects are now accepted.

|

| sep : str

| Separator between array items for text output.

| If "" (empty), a binary file is written, equivalent to

| ``file.write(a.tobytes())``.

| format : str

| Format string for text file output.

| Each entry in the array is formatted to text by first converting

| it to the closest Python type, and then using "format" % item.

|

| Notes

| -----

| This is a convenience function for quick storage of array data.

| Information on endianness and precision is lost, so this method is not a

| good choice for files intended to archive data or transport data between

| machines with different endianness. Some of these problems can be overcome

| by outputting the data as text files, at the expense of speed and file

| size.

|

| When fid is a file object, array contents are directly written to the

| file, bypassing the file object's ``write`` method. As a result, tofile

| cannot be used with files objects supporting compression (e.g., GzipFile)

| or file-like objects that do not support ``fileno()`` (e.g., BytesIO).

|

| tolist(...)

| a.tolist()

|

| Return the array as an ``a.ndim``-levels deep nested list of Python scalars.

|

| Return a copy of the array data as a (nested) Python list.

| Data items are converted to the nearest compatible builtin Python type, via

| the `~numpy.ndarray.item` function.

|

| If ``a.ndim`` is 0, then since the depth of the nested list is 0, it will

| not be a list at all, but a simple Python scalar.

|

| Parameters

| ----------

| none

|

| Returns

| -------

| y : object, or list of object, or list of list of object, or ...

| The possibly nested list of array elements.

|

| Notes

| -----

| The array may be recreated via ``a = np.array(a.tolist())``, although this

| may sometimes lose precision.

|

| Examples

| --------

| For a 1D array, ``a.tolist()`` is almost the same as ``list(a)``:

|

| >>> a = np.array([1, 2])

| >>> list(a)

| [1, 2]

| >>> a.tolist()

| [1, 2]

|

| However, for a 2D array, ``tolist`` applies recursively:

|

| >>> a = np.array([[1, 2], [3, 4]])

| >>> list(a)

| [array([1, 2]), array([3, 4])]

| >>> a.tolist()

| [[1, 2], [3, 4]]

|

| The base case for this recursion is a 0D array:

|

| >>> a = np.array(1)

| >>> list(a)

| Traceback (most recent call last):

| ...

| TypeError: iteration over a 0-d array

| >>> a.tolist()

| 1

|

| tostring(...)

| a.tostring(order='C')

|

| Construct Python bytes containing the raw data bytes in the array.

|

| Constructs Python bytes showing a copy of the raw contents of

| data memory. The bytes object can be produced in either 'C' or 'Fortran',

| or 'Any' order (the default is 'C'-order). 'Any' order means C-order

| unless the F\_CONTIGUOUS flag in the array is set, in which case it

| means 'Fortran' order.

|

| This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

|

| Parameters

| ----------

| order : {'C', 'F', None}, optional

| Order of the data for multidimensional arrays:

| C, Fortran, or the same as for the original array.

|

| Returns

| -------

| s : bytes

| Python bytes exhibiting a copy of `a`'s raw data.

|

| Examples

| --------

| >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')

| >>> x.tobytes()

| b'\x00\x00\x01\x00\x02\x00\x03\x00'

| >>> x.tobytes('C') == x.tobytes()

| True

| >>> x.tobytes('F')

| b'\x00\x00\x02\x00\x01\x00\x03\x00'

|

| trace(...)

| a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

|

| Return the sum along diagonals of the array.

|

| Refer to `numpy.trace` for full documentation.

|

| See Also

| --------

| numpy.trace : equivalent function

|

| transpose(...)

| a.transpose(\*axes)

|

| Returns a view of the array with axes transposed.

|

| For a 1-D array this has no effect, as a transposed vector is simply the

| same vector. To convert a 1-D array into a 2D column vector, an additional

| dimension must be added. `np.atleast2d(a).T` achieves this, as does

| `a[:, np.newaxis]`.

| For a 2-D array, this is a standard matrix transpose.

| For an n-D array, if axes are given, their order indicates how the

| axes are permuted (see Examples). If axes are not provided and

| ``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then

| ``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

|

| Parameters

| ----------

| axes : None, tuple of ints, or `n` ints

|

| \* None or no argument: reverses the order of the axes.

|

| \* tuple of ints: `i` in the `j`-th place in the tuple means `a`'s

| `i`-th axis becomes `a.transpose()`'s `j`-th axis.

|

| \* `n` ints: same as an n-tuple of the same ints (this form is

| intended simply as a "convenience" alternative to the tuple form)

|

| Returns

| -------

| out : ndarray

| View of `a`, with axes suitably permuted.

|

| See Also

| --------

| ndarray.T : Array property returning the array transposed.

| ndarray.reshape : Give a new shape to an array without changing its data.

|

| Examples

| --------

| >>> a = np.array([[1, 2], [3, 4]])

| >>> a

| array([[1, 2],

| [3, 4]])

| >>> a.transpose()

| array([[1, 3],

| [2, 4]])

| >>> a.transpose((1, 0))

| array([[1, 3],

| [2, 4]])

| >>> a.transpose(1, 0)

| array([[1, 3],

| [2, 4]])

|

| var(...)

| a.var(axis=None, dtype=None, out=None, ddof=0, keepdims=False)

|

| Returns the variance of the array elements, along given axis.

|

| Refer to `numpy.var` for full documentation.

|

| See Also

| --------

| numpy.var : equivalent function

|

| view(...)

| a.view(dtype=None, type=None)

|

| New view of array with the same data.

|

| Parameters

| ----------

| dtype : data-type or ndarray sub-class, optional

| Data-type descriptor of the returned view, e.g., float32 or int16. The

| default, None, results in the view having the same data-type as `a`.

| This argument can also be specified as an ndarray sub-class, which

| then specifies the type of the returned object (this is equivalent to

| setting the ``type`` parameter).

| type : Python type, optional

| Type of the returned view, e.g., ndarray or matrix. Again, the

| default None results in type preservation.

|

| Notes

| -----

| ``a.view()`` is used two different ways:

|

| ``a.view(some\_dtype)`` or ``a.view(dtype=some\_dtype)`` constructs a view

| of the array's memory with a different data-type. This can cause a

| reinterpretation of the bytes of memory.

|

| ``a.view(ndarray\_subclass)`` or ``a.view(type=ndarray\_subclass)`` just

| returns an instance of `ndarray\_subclass` that looks at the same array

| (same shape, dtype, etc.) This does not cause a reinterpretation of the

| memory.

|

| For ``a.view(some\_dtype)``, if ``some\_dtype`` has a different number of

| bytes per entry than the previous dtype (for example, converting a

| regular array to a structured array), then the behavior of the view

| cannot be predicted just from the superficial appearance of ``a`` (shown

| by ``print(a)``). It also depends on exactly how ``a`` is stored in

| memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus

| defined as a slice or transpose, etc., the view may give different

| results.

|

|

| Examples

| --------

| >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

|

| Viewing array data using a different type and dtype:

|

| >>> y = x.view(dtype=np.int16, type=np.matrix)

| >>> y

| matrix([[513]], dtype=int16)

| >>> print(type(y))

| <class 'numpy.matrix'>

|

| Creating a view on a structured array so it can be used in calculations

|

| >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])

| >>> xv = x.view(dtype=np.int8).reshape(-1,2)

| >>> xv

| array([[1, 2],

| [3, 4]], dtype=int8)

| >>> xv.mean(0)

| array([2., 3.])

|

| Making changes to the view changes the underlying array

|

| >>> xv[0,1] = 20

| >>> x

| array([(1, 20), (3, 4)], dtype=[('a', 'i1'), ('b', 'i1')])

|

| Using a view to convert an array to a recarray:

|

| >>> z = x.view(np.recarray)

| >>> z.a

| array([1, 3], dtype=int8)

|

| Views share data:

|

| >>> x[0] = (9, 10)

| >>> z[0]

| (9, 10)

|

| Views that change the dtype size (bytes per entry) should normally be

| avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

|

| >>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)

| >>> y = x[:, 0:2]

| >>> y

| array([[1, 2],

| [4, 5]], dtype=int16)

| >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])

| Traceback (most recent call last):

| ...

| ValueError: To change to a dtype of a different size, the array must be C-contiguous

| >>> z = y.copy()

| >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])

| array([[(1, 2)],

| [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])

|

| ----------------------------------------------------------------------

| Data descriptors inherited from ndarray:

|

| T

| The transposed array.

|

| Same as ``self.transpose()``.

|

| Examples

| --------

| >>> x = np.array([[1.,2.],[3.,4.]])

| >>> x

| array([[ 1., 2.],

| [ 3., 4.]])

| >>> x.T

| array([[ 1., 3.],

| [ 2., 4.]])

| >>> x = np.array([1.,2.,3.,4.])

| >>> x

| array([ 1., 2., 3., 4.])

| >>> x.T

| array([ 1., 2., 3., 4.])

|

| See Also

| --------

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side.

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: C-struct side.

|

| base

| Base object if memory is from some other object.

|

| Examples

| --------

| The base of an array that owns its memory is None:

|

| >>> x = np.array([1,2,3,4])

| >>> x.base is None

| True

|

| Slicing creates a view, whose memory is shared with x:

|

| >>> y = x[2:]

| >>> y.base is x

| True

|

| ctypes

| An object to simplify the interaction of the array with the ctypes

| module.

|

| This attribute creates an object that makes it easier to use arrays

| when calling shared libraries with the ctypes module. The returned

| object has, among others, data, shape, and strides attributes (see

| Notes below) which themselves return ctypes objects that can be used

| as arguments to a shared library.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| c : Python object

| Possessing attributes data, shape, strides, etc.

|

| See Also

| --------

| numpy.ctypeslib

|

| Notes

| -----

| Below are the public attributes of this object which were documented

| in "Guide to NumPy" (we have omitted undocumented public attributes,

| as well as documented private attributes):

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.data

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.shape

| :noindex:

|

| .. autoattribute:: numpy.core.\_internal.\_ctypes.strides

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.data\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.shape\_as

| :noindex:

|

| .. automethod:: numpy.core.\_internal.\_ctypes.strides\_as

| :noindex:

|

| If the ctypes module is not available, then the ctypes attribute

| of array objects still returns something useful, but ctypes objects

| are not returned and errors may be raised instead. In particular,

| the object will still have the ``as\_parameter`` attribute which will

| return an integer equal to the data attribute.

|

| Examples

| --------

| >>> import ctypes

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.ctypes.data

| 30439712

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long))

| <ctypes.LP\_c\_long object at 0x01F01300>

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_long)).contents

| c\_long(0)

| >>> x.ctypes.data\_as(ctypes.POINTER(ctypes.c\_longlong)).contents

| c\_longlong(4294967296L)

| >>> x.ctypes.shape

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FFD580>

| >>> x.ctypes.shape\_as(ctypes.c\_long)

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides

| <numpy.core.\_internal.c\_long\_Array\_2 object at 0x01FCE620>

| >>> x.ctypes.strides\_as(ctypes.c\_longlong)

| <numpy.core.\_internal.c\_longlong\_Array\_2 object at 0x01F01300>

|

| data

| Python buffer object pointing to the start of the array's data.

|

| dtype

| Data-type of the array's elements.

|

| Parameters

| ----------

| None

|

| Returns

| -------

| d : numpy dtype object

|

| See Also

| --------

| numpy.dtype

|

| Examples

| --------

| >>> x

| array([[0, 1],

| [2, 3]])

| >>> x.dtype

| dtype('int32')

| >>> type(x.dtype)

| <type 'numpy.dtype'>

|

| flags

| Information about the memory layout of the array.

|

| Attributes

| ----------

| C\_CONTIGUOUS (C)

| The data is in a single, C-style contiguous segment.

| F\_CONTIGUOUS (F)

| The data is in a single, Fortran-style contiguous segment.

| OWNDATA (O)

| The array owns the memory it uses or borrows it from another object.

| WRITEABLE (W)

| The data area can be written to. Setting this to False locks

| the data, making it read-only. A view (slice, etc.) inherits WRITEABLE

| from its base array at creation time, but a view of a writeable

| array may be subsequently locked while the base array remains writeable.

| (The opposite is not true, in that a view of a locked array may not

| be made writeable. However, currently, locking a base object does not

| lock any views that already reference it, so under that circumstance it

| is possible to alter the contents of a locked array via a previously

| created writeable view onto it.) Attempting to change a non-writeable

| array raises a RuntimeError exception.

| ALIGNED (A)

| The data and all elements are aligned appropriately for the hardware.

| WRITEBACKIFCOPY (X)

| This array is a copy of some other array. The C-API function

| PyArray\_ResolveWritebackIfCopy must be called before deallocating

| to the base array will be updated with the contents of this array.

| UPDATEIFCOPY (U)

| (Deprecated, use WRITEBACKIFCOPY) This array is a copy of some other array.

| When this array is

| deallocated, the base array will be updated with the contents of

| this array.

| FNC

| F\_CONTIGUOUS and not C\_CONTIGUOUS.

| FORC

| F\_CONTIGUOUS or C\_CONTIGUOUS (one-segment test).

| BEHAVED (B)

| ALIGNED and WRITEABLE.

| CARRAY (CA)

| BEHAVED and C\_CONTIGUOUS.

| FARRAY (FA)

| BEHAVED and F\_CONTIGUOUS and not C\_CONTIGUOUS.

|

| Notes

| -----

| The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),

| or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag

| names are only supported in dictionary access.

|

| Only the WRITEBACKIFCOPY, UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be

| changed by the user, via direct assignment to the attribute or dictionary

| entry, or by calling `ndarray.setflags`.

|

| The array flags cannot be set arbitrarily:

|

| - UPDATEIFCOPY can only be set ``False``.

| - WRITEBACKIFCOPY can only be set ``False``.

| - ALIGNED can only be set ``True`` if the data is truly aligned.

| - WRITEABLE can only be set ``True`` if the array owns its own memory

| or the ultimate owner of the memory exposes a writeable buffer

| interface or is a string.

|

| Arrays can be both C-style and Fortran-style contiguous simultaneously.

| This is clear for 1-dimensional arrays, but can also be true for higher

| dimensional arrays.

|

| Even for contiguous arrays a stride for a given dimension

| ``arr.strides[dim]`` may be \*arbitrary\* if ``arr.shape[dim] == 1``

| or the array has no elements.

| It does \*not\* generally hold that ``self.strides[-1] == self.itemsize``

| for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for

| Fortran-style contiguous arrays is true.

|

| flat

| A 1-D iterator over the array.

|

| This is a `numpy.flatiter` instance, which acts similarly to, but is not

| a subclass of, Python's built-in iterator object.

|

| See Also

| --------

| flatten : Return a copy of the array collapsed into one dimension.

|

| flatiter

|

| Examples

| --------

| >>> x = np.arange(1, 7).reshape(2, 3)

| >>> x

| array([[1, 2, 3],

| [4, 5, 6]])

| >>> x.flat[3]

| 4

| >>> x.T

| array([[1, 4],

| [2, 5],

| [3, 6]])

| >>> x.T.flat[3]

| 5

| >>> type(x.flat)

| <class 'numpy.flatiter'>

|

| An assignment example:

|

| >>> x.flat = 3; x

| array([[3, 3, 3],

| [3, 3, 3]])

| >>> x.flat[[1,4]] = 1; x

| array([[3, 1, 3],

| [3, 1, 3]])

|

| imag

| The imaginary part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.imag

| array([ 0. , 0.70710678])

| >>> x.imag.dtype

| dtype('float64')

|

| itemsize

| Length of one array element in bytes.

|

| Examples

| --------

| >>> x = np.array([1,2,3], dtype=np.float64)

| >>> x.itemsize

| 8

| >>> x = np.array([1,2,3], dtype=np.complex128)

| >>> x.itemsize

| 16

|

| nbytes

| Total bytes consumed by the elements of the array.

|

| Notes

| -----

| Does not include memory consumed by non-element attributes of the

| array object.

|

| Examples

| --------

| >>> x = np.zeros((3,5,2), dtype=np.complex128)

| >>> x.nbytes

| 480

| >>> np.prod(x.shape) \* x.itemsize

| 480

|

| ndim

| Number of array dimensions.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3])

| >>> x.ndim

| 1

| >>> y = np.zeros((2, 3, 4))

| >>> y.ndim

| 3

|

| real

| The real part of the array.

|

| Examples

| --------

| >>> x = np.sqrt([1+0j, 0+1j])

| >>> x.real

| array([ 1. , 0.70710678])

| >>> x.real.dtype

| dtype('float64')

|

| See Also

| --------

| numpy.real : equivalent function

|

| shape

| Tuple of array dimensions.

|

| The shape property is usually used to get the current shape of an array,

| but may also be used to reshape the array in-place by assigning a tuple of

| array dimensions to it. As with `numpy.reshape`, one of the new shape

| dimensions can be -1, in which case its value is inferred from the size of

| the array and the remaining dimensions. Reshaping an array in-place will

| fail if a copy is required.

|

| Examples

| --------

| >>> x = np.array([1, 2, 3, 4])

| >>> x.shape

| (4,)

| >>> y = np.zeros((2, 3, 4))

| >>> y.shape

| (2, 3, 4)

| >>> y.shape = (3, 8)

| >>> y

| array([[ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.],

| [ 0., 0., 0., 0., 0., 0., 0., 0.]])

| >>> y.shape = (3, 6)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| ValueError: total size of new array must be unchanged

| >>> np.zeros((4,2))[::2].shape = (-1,)

| Traceback (most recent call last):

| File "<stdin>", line 1, in <module>

| AttributeError: incompatible shape for a non-contiguous array

|

| See Also

| --------

| numpy.reshape : similar function

| ndarray.reshape : similar method

|

| size

| Number of elements in the array.

|

| Equal to ``np.prod(a.shape)``, i.e., the product of the array's

| dimensions.

|

| Notes

| -----

| `a.size` returns a standard arbitrary precision Python integer. This

| may not be the case with other methods of obtaining the same value

| (like the suggested ``np.prod(a.shape)``, which returns an instance

| of ``np.int\_``), and may be relevant if the value is used further in

| calculations that may overflow a fixed size integer type.

|

| Examples

| --------

| >>> x = np.zeros((3, 5, 2), dtype=np.complex128)

| >>> x.size

| 30

| >>> np.prod(x.shape)

| 30

|

| strides

| Tuple of bytes to step in each dimension when traversing an array.

|

| The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`

| is::

|

| offset = sum(np.array(i) \* a.strides)

|

| A more detailed explanation of strides can be found in the

| "ndarray.rst" file in the NumPy reference guide.

|

| Notes

| -----

| Imagine an array of 32-bit integers (each 4 bytes)::

|

| x = np.array([[0, 1, 2, 3, 4],

| [5, 6, 7, 8, 9]], dtype=np.int32)

|

| This array is stored in memory as 40 bytes, one after the other

| (known as a contiguous block of memory). The strides of an array tell

| us how many bytes we have to skip in memory to move to the next position

| along a certain axis. For example, we have to skip 4 bytes (1 value) to

| move to the next column, but 20 bytes (5 values) to get to the same

| position in the next row. As such, the strides for the array `x` will be

| ``(20, 4)``.

|

| See Also

| --------

| numpy.lib.stride\_tricks.as\_strided

|

| Examples

| --------

| >>> y = np.reshape(np.arange(2\*3\*4), (2,3,4))

| >>> y

| array([[[ 0, 1, 2, 3],

| [ 4, 5, 6, 7],

| [ 8, 9, 10, 11]],

| [[12, 13, 14, 15],

| [16, 17, 18, 19],

| [20, 21, 22, 23]]])

| >>> y.strides

| (48, 16, 4)

| >>> y[1,1,1]

| 17

| >>> offset=sum(y.strides \* np.array((1,1,1)))

| >>> offset/y.itemsize

| 17

|

| >>> x = np.reshape(np.arange(5\*6\*7\*8), (5,6,7,8)).transpose(2,3,1,0)

| >>> x.strides

| (32, 4, 224, 1344)

| >>> i = np.array([3,5,2,2])

| >>> offset = sum(i \* x.strides)

| >>> x[3,5,2,2]

| 813

| >>> offset / x.itemsize

| 813

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from ndarray:

|

| \_\_hash\_\_ = None

class record(void)

| A data-type scalar that allows field access as attribute lookup.

|

| Method resolution order:

| record

| void

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_getattribute\_\_(self, attr)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, indx)

| Return self[key].

|

| \_\_repr\_\_(self)

| Return repr(self).

|

| \_\_setattr\_\_(self, attr, val)

| Implement setattr(self, name, value).

|

| \_\_str\_\_(self)

| Return str(self).

|

| pprint(self)

| Pretty-print all fields.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

|

| ----------------------------------------------------------------------

| Methods inherited from void:

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Static methods inherited from void:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from void:

|

| base

| base object

|

| dtype

| dtype object

|

| flags

| integer value of flags

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| data

| pointer to start of data

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

short = class int16(signedinteger)

| Signed integer type, compatible with C ``short``.

| Character code: ``'h'``.

| Canonical name: ``np.short``.

| Alias \*on this platform\*: ``np.int16``: 16-bit signed integer (-32768 to 32767).

|

| Method resolution order:

| int16

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class signedinteger(integer)

| Abstract base class of all signed integer scalar types.

|

| Method resolution order:

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

single = class float32(floating)

| Single-precision floating-point number type, compatible with C ``float``.

| Character code: ``'f'``.

| Canonical name: ``np.single``.

| Alias \*on this platform\*: ``np.float32``: 32-bit-precision floating-point number type: sign bit, 8 bits exponent, 23 bits mantissa.

|

| Method resolution order:

| float32

| floating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| as\_integer\_ratio(...)

| single.as\_integer\_ratio() -> (int, int)

|

| Return a pair of integers, whose ratio is exactly equal to the original

| floating point number, and with a positive denominator.

| Raise OverflowError on infinities and a ValueError on NaNs.

|

| >>> np.single(10.0).as\_integer\_ratio()

| (10, 1)

| >>> np.single(0.0).as\_integer\_ratio()

| (0, 1)

| >>> np.single(-.25).as\_integer\_ratio()

| (-1, 4)

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

singlecomplex = class complex64(complexfloating)

| Complex number type composed of two single-precision floating-point

| numbers.

| Character code: ``'F'``.

| Canonical name: ``np.csingle``.

| Alias: ``np.singlecomplex``.

| Alias \*on this platform\*: ``np.complex64``: Complex number type composed of 2 32-bit-precision floating-point numbers.

|

| Method resolution order:

| complex64

| complexfloating

| inexact

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_complex\_\_(...)

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

str0 = class str\_(builtins.str, character)

| str(object='') -> str

| str(bytes\_or\_buffer[, encoding[, errors]]) -> str

|

| Create a new string object from the given object. If encoding or

| errors is specified, then the object must expose a data buffer

| that will be decoded using the given encoding and error handler.

| Otherwise, returns the result of object.\_\_str\_\_() (if defined)

| or repr(object).

| encoding defaults to sys.getdefaultencoding().

| errors defaults to 'strict'.

|

| Method resolution order:

| str\_

| builtins.str

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.str:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_format\_\_(self, format\_spec, /)

| Return a formatted version of the string as described by format\_spec.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_sizeof\_\_(self, /)

| Return the size of the string in memory, in bytes.

|

| capitalize(self, /)

| Return a capitalized version of the string.

|

| More specifically, make the first character have upper case and the rest lower

| case.

|

| casefold(self, /)

| Return a version of the string suitable for caseless comparisons.

|

| center(self, width, fillchar=' ', /)

| Return a centered string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| count(...)

| S.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of substring sub in

| string S[start:end]. Optional arguments start and end are

| interpreted as in slice notation.

|

| encode(self, /, encoding='utf-8', errors='strict')

| Encode the string using the codec registered for encoding.

|

| encoding

| The encoding in which to encode the string.

| errors

| The error handling scheme to use for encoding errors.

| The default is 'strict' meaning that encoding errors raise a

| UnicodeEncodeError. Other possible values are 'ignore', 'replace' and

| 'xmlcharrefreplace' as well as any other name registered with

| codecs.register\_error that can handle UnicodeEncodeErrors.

|

| endswith(...)

| S.endswith(suffix[, start[, end]]) -> bool

|

| Return True if S ends with the specified suffix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| suffix can also be a tuple of strings to try.

|

| expandtabs(self, /, tabsize=8)

| Return a copy where all tab characters are expanded using spaces.

|

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| S.find(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| format(...)

| S.format(\*args, \*\*kwargs) -> str

|

| Return a formatted version of S, using substitutions from args and kwargs.

| The substitutions are identified by braces ('{' and '}').

|

| format\_map(...)

| S.format\_map(mapping) -> str

|

| Return a formatted version of S, using substitutions from mapping.

| The substitutions are identified by braces ('{' and '}').

|

| index(...)

| S.index(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| isalnum(self, /)

| Return True if the string is an alpha-numeric string, False otherwise.

|

| A string is alpha-numeric if all characters in the string are alpha-numeric and

| there is at least one character in the string.

|

| isalpha(self, /)

| Return True if the string is an alphabetic string, False otherwise.

|

| A string is alphabetic if all characters in the string are alphabetic and there

| is at least one character in the string.

|

| isascii(self, /)

| Return True if all characters in the string are ASCII, False otherwise.

|

| ASCII characters have code points in the range U+0000-U+007F.

| Empty string is ASCII too.

|

| isdecimal(self, /)

| Return True if the string is a decimal string, False otherwise.

|

| A string is a decimal string if all characters in the string are decimal and

| there is at least one character in the string.

|

| isdigit(self, /)

| Return True if the string is a digit string, False otherwise.

|

| A string is a digit string if all characters in the string are digits and there

| is at least one character in the string.

|

| isidentifier(self, /)

| Return True if the string is a valid Python identifier, False otherwise.

|

| Use keyword.iskeyword() to test for reserved identifiers such as "def" and

| "class".

|

| islower(self, /)

| Return True if the string is a lowercase string, False otherwise.

|

| A string is lowercase if all cased characters in the string are lowercase and

| there is at least one cased character in the string.

|

| isnumeric(self, /)

| Return True if the string is a numeric string, False otherwise.

|

| A string is numeric if all characters in the string are numeric and there is at

| least one character in the string.

|

| isprintable(self, /)

| Return True if the string is printable, False otherwise.

|

| A string is printable if all of its characters are considered printable in

| repr() or if it is empty.

|

| isspace(self, /)

| Return True if the string is a whitespace string, False otherwise.

|

| A string is whitespace if all characters in the string are whitespace and there

| is at least one character in the string.

|

| istitle(self, /)

| Return True if the string is a title-cased string, False otherwise.

|

| In a title-cased string, upper- and title-case characters may only

| follow uncased characters and lowercase characters only cased ones.

|

| isupper(self, /)

| Return True if the string is an uppercase string, False otherwise.

|

| A string is uppercase if all cased characters in the string are uppercase and

| there is at least one cased character in the string.

|

| join(self, iterable, /)

| Concatenate any number of strings.

|

| The string whose method is called is inserted in between each given string.

| The result is returned as a new string.

|

| Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

|

| ljust(self, width, fillchar=' ', /)

| Return a left-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| lower(self, /)

| Return a copy of the string converted to lowercase.

|

| lstrip(self, chars=None, /)

| Return a copy of the string with leading whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| partition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original string

| and two empty strings.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| S.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| S.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| rjust(self, width, fillchar=' ', /)

| Return a right-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| rpartition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty strings

| and the original string.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splits are done starting at the end of the string and working to the front.

|

| rstrip(self, chars=None, /)

| Return a copy of the string with trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the string, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| S.startswith(prefix[, start[, end]]) -> bool

|

| Return True if S starts with the specified prefix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| prefix can also be a tuple of strings to try.

|

| strip(self, chars=None, /)

| Return a copy of the string with leading and trailing whitespace remove.

|

| If chars is given and not None, remove characters in chars instead.

|

| swapcase(self, /)

| Convert uppercase characters to lowercase and lowercase characters to uppercase.

|

| title(self, /)

| Return a version of the string where each word is titlecased.

|

| More specifically, words start with uppercased characters and all remaining

| cased characters have lower case.

|

| translate(self, table, /)

| Replace each character in the string using the given translation table.

|

| table

| Translation table, which must be a mapping of Unicode ordinals to

| Unicode ordinals, strings, or None.

|

| The table must implement lookup/indexing via \_\_getitem\_\_, for instance a

| dictionary or list. If this operation raises LookupError, the character is

| left untouched. Characters mapped to None are deleted.

|

| upper(self, /)

| Return a copy of the string converted to uppercase.

|

| zfill(self, width, /)

| Pad a numeric string with zeros on the left, to fill a field of the given width.

|

| The string is never truncated.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.str:

|

| maketrans(x, y=None, z=None, /)

| Return a translation table usable for str.translate().

|

| If there is only one argument, it must be a dictionary mapping Unicode

| ordinals (integers) or characters to Unicode ordinals, strings or None.

| Character keys will be then converted to ordinals.

| If there are two arguments, they must be strings of equal length, and

| in the resulting dictionary, each character in x will be mapped to the

| character at the same position in y. If there is a third argument, it

| must be a string, whose characters will be mapped to None in the result.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class str\_(builtins.str, character)

| str(object='') -> str

| str(bytes\_or\_buffer[, encoding[, errors]]) -> str

|

| Create a new string object from the given object. If encoding or

| errors is specified, then the object must expose a data buffer

| that will be decoded using the given encoding and error handler.

| Otherwise, returns the result of object.\_\_str\_\_() (if defined)

| or repr(object).

| encoding defaults to sys.getdefaultencoding().

| errors defaults to 'strict'.

|

| Method resolution order:

| str\_

| builtins.str

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.str:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_format\_\_(self, format\_spec, /)

| Return a formatted version of the string as described by format\_spec.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_sizeof\_\_(self, /)

| Return the size of the string in memory, in bytes.

|

| capitalize(self, /)

| Return a capitalized version of the string.

|

| More specifically, make the first character have upper case and the rest lower

| case.

|

| casefold(self, /)

| Return a version of the string suitable for caseless comparisons.

|

| center(self, width, fillchar=' ', /)

| Return a centered string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| count(...)

| S.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of substring sub in

| string S[start:end]. Optional arguments start and end are

| interpreted as in slice notation.

|

| encode(self, /, encoding='utf-8', errors='strict')

| Encode the string using the codec registered for encoding.

|

| encoding

| The encoding in which to encode the string.

| errors

| The error handling scheme to use for encoding errors.

| The default is 'strict' meaning that encoding errors raise a

| UnicodeEncodeError. Other possible values are 'ignore', 'replace' and

| 'xmlcharrefreplace' as well as any other name registered with

| codecs.register\_error that can handle UnicodeEncodeErrors.

|

| endswith(...)

| S.endswith(suffix[, start[, end]]) -> bool

|

| Return True if S ends with the specified suffix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| suffix can also be a tuple of strings to try.

|

| expandtabs(self, /, tabsize=8)

| Return a copy where all tab characters are expanded using spaces.

|

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| S.find(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| format(...)

| S.format(\*args, \*\*kwargs) -> str

|

| Return a formatted version of S, using substitutions from args and kwargs.

| The substitutions are identified by braces ('{' and '}').

|

| format\_map(...)

| S.format\_map(mapping) -> str

|

| Return a formatted version of S, using substitutions from mapping.

| The substitutions are identified by braces ('{' and '}').

|

| index(...)

| S.index(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| isalnum(self, /)

| Return True if the string is an alpha-numeric string, False otherwise.

|

| A string is alpha-numeric if all characters in the string are alpha-numeric and

| there is at least one character in the string.

|

| isalpha(self, /)

| Return True if the string is an alphabetic string, False otherwise.

|

| A string is alphabetic if all characters in the string are alphabetic and there

| is at least one character in the string.

|

| isascii(self, /)

| Return True if all characters in the string are ASCII, False otherwise.

|

| ASCII characters have code points in the range U+0000-U+007F.

| Empty string is ASCII too.

|

| isdecimal(self, /)

| Return True if the string is a decimal string, False otherwise.

|

| A string is a decimal string if all characters in the string are decimal and

| there is at least one character in the string.

|

| isdigit(self, /)

| Return True if the string is a digit string, False otherwise.

|

| A string is a digit string if all characters in the string are digits and there

| is at least one character in the string.

|

| isidentifier(self, /)

| Return True if the string is a valid Python identifier, False otherwise.

|

| Use keyword.iskeyword() to test for reserved identifiers such as "def" and

| "class".

|

| islower(self, /)

| Return True if the string is a lowercase string, False otherwise.

|

| A string is lowercase if all cased characters in the string are lowercase and

| there is at least one cased character in the string.

|

| isnumeric(self, /)

| Return True if the string is a numeric string, False otherwise.

|

| A string is numeric if all characters in the string are numeric and there is at

| least one character in the string.

|

| isprintable(self, /)

| Return True if the string is printable, False otherwise.

|

| A string is printable if all of its characters are considered printable in

| repr() or if it is empty.

|

| isspace(self, /)

| Return True if the string is a whitespace string, False otherwise.

|

| A string is whitespace if all characters in the string are whitespace and there

| is at least one character in the string.

|

| istitle(self, /)

| Return True if the string is a title-cased string, False otherwise.

|

| In a title-cased string, upper- and title-case characters may only

| follow uncased characters and lowercase characters only cased ones.

|

| isupper(self, /)

| Return True if the string is an uppercase string, False otherwise.

|

| A string is uppercase if all cased characters in the string are uppercase and

| there is at least one cased character in the string.

|

| join(self, iterable, /)

| Concatenate any number of strings.

|

| The string whose method is called is inserted in between each given string.

| The result is returned as a new string.

|

| Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

|

| ljust(self, width, fillchar=' ', /)

| Return a left-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| lower(self, /)

| Return a copy of the string converted to lowercase.

|

| lstrip(self, chars=None, /)

| Return a copy of the string with leading whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| partition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original string

| and two empty strings.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| S.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| S.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| rjust(self, width, fillchar=' ', /)

| Return a right-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| rpartition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty strings

| and the original string.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splits are done starting at the end of the string and working to the front.

|

| rstrip(self, chars=None, /)

| Return a copy of the string with trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the string, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| S.startswith(prefix[, start[, end]]) -> bool

|

| Return True if S starts with the specified prefix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| prefix can also be a tuple of strings to try.

|

| strip(self, chars=None, /)

| Return a copy of the string with leading and trailing whitespace remove.

|

| If chars is given and not None, remove characters in chars instead.

|

| swapcase(self, /)

| Convert uppercase characters to lowercase and lowercase characters to uppercase.

|

| title(self, /)

| Return a version of the string where each word is titlecased.

|

| More specifically, words start with uppercased characters and all remaining

| cased characters have lower case.

|

| translate(self, table, /)

| Replace each character in the string using the given translation table.

|

| table

| Translation table, which must be a mapping of Unicode ordinals to

| Unicode ordinals, strings, or None.

|

| The table must implement lookup/indexing via \_\_getitem\_\_, for instance a

| dictionary or list. If this operation raises LookupError, the character is

| left untouched. Characters mapped to None are deleted.

|

| upper(self, /)

| Return a copy of the string converted to uppercase.

|

| zfill(self, width, /)

| Pad a numeric string with zeros on the left, to fill a field of the given width.

|

| The string is never truncated.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.str:

|

| maketrans(x, y=None, z=None, /)

| Return a translation table usable for str.translate().

|

| If there is only one argument, it must be a dictionary mapping Unicode

| ordinals (integers) or characters to Unicode ordinals, strings or None.

| Character keys will be then converted to ordinals.

| If there are two arguments, they must be strings of equal length, and

| in the resulting dictionary, each character in x will be mapped to the

| character at the same position in y. If there is a third argument, it

| must be a string, whose characters will be mapped to None in the result.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

string\_ = class bytes\_(builtins.bytes, character)

| bytes(iterable\_of\_ints) -> bytes

| bytes(string, encoding[, errors]) -> bytes

| bytes(bytes\_or\_buffer) -> immutable copy of bytes\_or\_buffer

| bytes(int) -> bytes object of size given by the parameter initialized with null bytes

| bytes() -> empty bytes object

|

| Construct an immutable array of bytes from:

| - an iterable yielding integers in range(256)

| - a text string encoded using the specified encoding

| - any object implementing the buffer API.

| - an integer

|

| Method resolution order:

| bytes\_

| builtins.bytes

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.bytes:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| capitalize(...)

| B.capitalize() -> copy of B

|

| Return a copy of B with only its first character capitalized (ASCII)

| and the rest lower-cased.

|

| center(...)

| B.center(width[, fillchar]) -> copy of B

|

| Return B centered in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| count(...)

| B.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of subsection sub in

| bytes B[start:end]. Optional arguments start and end are interpreted

| as in slice notation.

|

| decode(self, /, encoding='utf-8', errors='strict')

| Decode the bytes using the codec registered for encoding.

|

| encoding

| The encoding with which to decode the bytes.

| errors

| The error handling scheme to use for the handling of decoding errors.

| The default is 'strict' meaning that decoding errors raise a

| UnicodeDecodeError. Other possible values are 'ignore' and 'replace'

| as well as any other name registered with codecs.register\_error that

| can handle UnicodeDecodeErrors.

|

| endswith(...)

| B.endswith(suffix[, start[, end]]) -> bool

|

| Return True if B ends with the specified suffix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| suffix can also be a tuple of bytes to try.

|

| expandtabs(...)

| B.expandtabs(tabsize=8) -> copy of B

|

| Return a copy of B where all tab characters are expanded using spaces.

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| B.find(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| hex(...)

| B.hex() -> string

|

| Create a string of hexadecimal numbers from a bytes object.

| Example: b'\xb9\x01\xef'.hex() -> 'b901ef'.

|

| index(...)

| B.index(sub[, start[, end]]) -> int

|

| Return the lowest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the subsection is not found.

|

| isalnum(...)

| B.isalnum() -> bool

|

| Return True if all characters in B are alphanumeric

| and there is at least one character in B, False otherwise.

|

| isalpha(...)

| B.isalpha() -> bool

|

| Return True if all characters in B are alphabetic

| and there is at least one character in B, False otherwise.

|

| isascii(...)

| B.isascii() -> bool

|

| Return True if B is empty or all characters in B are ASCII,

| False otherwise.

|

| isdigit(...)

| B.isdigit() -> bool

|

| Return True if all characters in B are digits

| and there is at least one character in B, False otherwise.

|

| islower(...)

| B.islower() -> bool

|

| Return True if all cased characters in B are lowercase and there is

| at least one cased character in B, False otherwise.

|

| isspace(...)

| B.isspace() -> bool

|

| Return True if all characters in B are whitespace

| and there is at least one character in B, False otherwise.

|

| istitle(...)

| B.istitle() -> bool

|

| Return True if B is a titlecased string and there is at least one

| character in B, i.e. uppercase characters may only follow uncased

| characters and lowercase characters only cased ones. Return False

| otherwise.

|

| isupper(...)

| B.isupper() -> bool

|

| Return True if all cased characters in B are uppercase and there is

| at least one cased character in B, False otherwise.

|

| join(self, iterable\_of\_bytes, /)

| Concatenate any number of bytes objects.

|

| The bytes whose method is called is inserted in between each pair.

|

| The result is returned as a new bytes object.

|

| Example: b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'.

|

| ljust(...)

| B.ljust(width[, fillchar]) -> copy of B

|

| Return B left justified in a string of length width. Padding is

| done using the specified fill character (default is a space).

|

| lower(...)

| B.lower() -> copy of B

|

| Return a copy of B with all ASCII characters converted to lowercase.

|

| lstrip(self, bytes=None, /)

| Strip leading bytes contained in the argument.

|

| If the argument is omitted or None, strip leading ASCII whitespace.

|

| partition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original bytes

| object and two empty bytes objects.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| B.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| B.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in B where subsection sub is found,

| such that sub is contained within B[start,end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raise ValueError when the subsection is not found.

|

| rjust(...)

| B.rjust(width[, fillchar]) -> copy of B

|

| Return B right justified in a string of length width. Padding is

| done using the specified fill character (default is a space)

|

| rpartition(self, sep, /)

| Partition the bytes into three parts using the given separator.

|

| This will search for the separator sep in the bytes, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty bytes

| objects and the original bytes object.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splitting is done starting at the end of the bytes and working to the front.

|

| rstrip(self, bytes=None, /)

| Strip trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip trailing ASCII whitespace.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the sections in the bytes, using sep as the delimiter.

|

| sep

| The delimiter according which to split the bytes.

| None (the default value) means split on ASCII whitespace characters

| (space, tab, return, newline, formfeed, vertical tab).

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the bytes, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| B.startswith(prefix[, start[, end]]) -> bool

|

| Return True if B starts with the specified prefix, False otherwise.

| With optional start, test B beginning at that position.

| With optional end, stop comparing B at that position.

| prefix can also be a tuple of bytes to try.

|

| strip(self, bytes=None, /)

| Strip leading and trailing bytes contained in the argument.

|

| If the argument is omitted or None, strip leading and trailing ASCII whitespace.

|

| swapcase(...)

| B.swapcase() -> copy of B

|

| Return a copy of B with uppercase ASCII characters converted

| to lowercase ASCII and vice versa.

|

| title(...)

| B.title() -> copy of B

|

| Return a titlecased version of B, i.e. ASCII words start with uppercase

| characters, all remaining cased characters have lowercase.

|

| translate(self, table, /, delete=b'')

| Return a copy with each character mapped by the given translation table.

|

| table

| Translation table, which must be a bytes object of length 256.

|

| All characters occurring in the optional argument delete are removed.

| The remaining characters are mapped through the given translation table.

|

| upper(...)

| B.upper() -> copy of B

|

| Return a copy of B with all ASCII characters converted to uppercase.

|

| zfill(...)

| B.zfill(width) -> copy of B

|

| Pad a numeric string B with zeros on the left, to fill a field

| of the specified width. B is never truncated.

|

| ----------------------------------------------------------------------

| Class methods inherited from builtins.bytes:

|

| fromhex(string, /) from builtins.type

| Create a bytes object from a string of hexadecimal numbers.

|

| Spaces between two numbers are accepted.

| Example: bytes.fromhex('B9 01EF') -> b'\\xb9\\x01\\xef'.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.bytes:

|

| maketrans(frm, to, /)

| Return a translation table useable for the bytes or bytearray translate method.

|

| The returned table will be one where each byte in frm is mapped to the byte at

| the same position in to.

|

| The bytes objects frm and to must be of the same length.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class timedelta64(signedinteger)

| Abstract base class of all signed integer scalar types.

|

| Method resolution order:

| timedelta64

| signedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

ubyte = class uint8(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned char``.

| Character code: ``'B'``.

| Canonical name: ``np.ubyte``.

| Alias \*on this platform\*: ``np.uint8``: 8-bit unsigned integer (0 to 255).

|

| Method resolution order:

| uint8

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class ufunc(builtins.object)

| Functions that operate element by element on whole arrays.

|

| To see the documentation for a specific ufunc, use `info`. For

| example, ``np.info(np.sin)``. Because ufuncs are written in C

| (for speed) and linked into Python with NumPy's ufunc facility,

| Python's help() function finds this page whenever help() is called

| on a ufunc.

|

| A detailed explanation of ufuncs can be found in the docs for :ref:`ufuncs`.

|

| Calling ufuncs:

| ===============

|

| op(\*x[, out], where=True, \*\*kwargs)

| Apply `op` to the arguments `\*x` elementwise, broadcasting the arguments.

|

| The broadcasting rules are:

|

| \* Dimensions of length 1 may be prepended to either array.

| \* Arrays may be repeated along dimensions of length 1.

|

| Parameters

| ----------

| \*x : array\_like

| Input arrays.

| out : ndarray, None, or tuple of ndarray and None, optional

| Alternate array object(s) in which to put the result; if provided, it

| must have a shape that the inputs broadcast to. A tuple of arrays

| (possible only as a keyword argument) must have length equal to the

| number of outputs; use `None` for uninitialized outputs to be

| allocated by the ufunc.

| where : array\_like, optional

| This condition is broadcast over the input. At locations where the

| condition is True, the `out` array will be set to the ufunc result.

| Elsewhere, the `out` array will retain its original value.

| Note that if an uninitialized `out` array is created via the default

| ``out=None``, locations within it where the condition is False will

| remain uninitialized.

| \*\*kwargs

| For other keyword-only arguments, see the :ref:`ufunc docs <ufuncs.kwargs>`.

|

| Returns

| -------

| r : ndarray or tuple of ndarray

| `r` will have the shape that the arrays in `x` broadcast to; if `out` is

| provided, it will be returned. If not, `r` will be allocated and

| may contain uninitialized values. If the function has more than one

| output, then the result will be a tuple of arrays.

|

| Methods defined here:

|

| \_\_call\_\_(self, /, \*args, \*\*kwargs)

| Call self as a function.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| accumulate(...)

| accumulate(array, axis=0, dtype=None, out=None)

|

| Accumulate the result of applying the operator to all elements.

|

| For a one-dimensional array, accumulate produces results equivalent to::

|

| r = np.empty(len(A))

| t = op.identity # op = the ufunc being applied to A's elements

| for i in range(len(A)):

| t = op(t, A[i])

| r[i] = t

| return r

|

| For example, add.accumulate() is equivalent to np.cumsum().

|

| For a multi-dimensional array, accumulate is applied along only one

| axis (axis zero by default; see Examples below) so repeated use is

| necessary if one wants to accumulate over multiple axes.

|

| Parameters

| ----------

| array : array\_like

| The array to act on.

| axis : int, optional

| The axis along which to apply the accumulation; default is zero.

| dtype : data-type code, optional

| The data-type used to represent the intermediate results. Defaults

| to the data-type of the output array if such is provided, or the

| the data-type of the input array if no output array is provided.

| out : ndarray, None, or tuple of ndarray and None, optional

| A location into which the result is stored. If not provided or `None`,

| a freshly-allocated array is returned. For consistency with

| ``ufunc.\_\_call\_\_``, if given as a keyword, this may be wrapped in a

| 1-element tuple.

|

| .. versionchanged:: 1.13.0

| Tuples are allowed for keyword argument.

|

| Returns

| -------

| r : ndarray

| The accumulated values. If `out` was supplied, `r` is a reference to

| `out`.

|

| Examples

| --------

| 1-D array examples:

|

| >>> np.add.accumulate([2, 3, 5])

| array([ 2, 5, 10])

| >>> np.multiply.accumulate([2, 3, 5])

| array([ 2, 6, 30])

|

| 2-D array examples:

|

| >>> I = np.eye(2)

| >>> I

| array([[1., 0.],

| [0., 1.]])

|

| Accumulate along axis 0 (rows), down columns:

|

| >>> np.add.accumulate(I, 0)

| array([[1., 0.],

| [1., 1.]])

| >>> np.add.accumulate(I) # no axis specified = axis zero

| array([[1., 0.],

| [1., 1.]])

|

| Accumulate along axis 1 (columns), through rows:

|

| >>> np.add.accumulate(I, 1)

| array([[1., 1.],

| [0., 1.]])

|

| at(...)

| at(a, indices, b=None)

|

| Performs unbuffered in place operation on operand 'a' for elements

| specified by 'indices'. For addition ufunc, this method is equivalent to

| ``a[indices] += b``, except that results are accumulated for elements that

| are indexed more than once. For example, ``a[[0,0]] += 1`` will only

| increment the first element once because of buffering, whereas

| ``add.at(a, [0,0], 1)`` will increment the first element twice.

|

| .. versionadded:: 1.8.0

|

| Parameters

| ----------

| a : array\_like

| The array to perform in place operation on.

| indices : array\_like or tuple

| Array like index object or slice object for indexing into first

| operand. If first operand has multiple dimensions, indices can be a

| tuple of array like index objects or slice objects.

| b : array\_like

| Second operand for ufuncs requiring two operands. Operand must be

| broadcastable over first operand after indexing or slicing.

|

| Examples

| --------

| Set items 0 and 1 to their negative values:

|

| >>> a = np.array([1, 2, 3, 4])

| >>> np.negative.at(a, [0, 1])

| >>> a

| array([-1, -2, 3, 4])

|

| Increment items 0 and 1, and increment item 2 twice:

|

| >>> a = np.array([1, 2, 3, 4])

| >>> np.add.at(a, [0, 1, 2, 2], 1)

| >>> a

| array([2, 3, 5, 4])

|

| Add items 0 and 1 in first array to second array,

| and store results in first array:

|

| >>> a = np.array([1, 2, 3, 4])

| >>> b = np.array([1, 2])

| >>> np.add.at(a, [0, 1], b)

| >>> a

| array([2, 4, 3, 4])

|

| outer(...)

| outer(A, B, \*\*kwargs)

|

| Apply the ufunc `op` to all pairs (a, b) with a in `A` and b in `B`.

|

| Let ``M = A.ndim``, ``N = B.ndim``. Then the result, `C`, of

| ``op.outer(A, B)`` is an array of dimension M + N such that:

|

| .. math:: C[i\_0, ..., i\_{M-1}, j\_0, ..., j\_{N-1}] =

| op(A[i\_0, ..., i\_{M-1}], B[j\_0, ..., j\_{N-1}])

|

| For `A` and `B` one-dimensional, this is equivalent to::

|

| r = empty(len(A),len(B))

| for i in range(len(A)):

| for j in range(len(B)):

| r[i,j] = op(A[i], B[j]) # op = ufunc in question

|

| Parameters

| ----------

| A : array\_like

| First array

| B : array\_like

| Second array

| kwargs : any

| Arguments to pass on to the ufunc. Typically `dtype` or `out`.

|

| Returns

| -------

| r : ndarray

| Output array

|

| See Also

| --------

| numpy.outer

|

| Examples

| --------

| >>> np.multiply.outer([1, 2, 3], [4, 5, 6])

| array([[ 4, 5, 6],

| [ 8, 10, 12],

| [12, 15, 18]])

|

| A multi-dimensional example:

|

| >>> A = np.array([[1, 2, 3], [4, 5, 6]])

| >>> A.shape

| (2, 3)

| >>> B = np.array([[1, 2, 3, 4]])

| >>> B.shape

| (1, 4)

| >>> C = np.multiply.outer(A, B)

| >>> C.shape; C

| (2, 3, 1, 4)

| array([[[[ 1, 2, 3, 4]],

| [[ 2, 4, 6, 8]],

| [[ 3, 6, 9, 12]]],

| [[[ 4, 8, 12, 16]],

| [[ 5, 10, 15, 20]],

| [[ 6, 12, 18, 24]]]])

|

| reduce(...)

| reduce(a, axis=0, dtype=None, out=None, keepdims=False, initial=<no value>, where=True)

|

| Reduces `a`'s dimension by one, by applying ufunc along one axis.

|

| Let :math:`a.shape = (N\_0, ..., N\_i, ..., N\_{M-1})`. Then

| :math:`ufunc.reduce(a, axis=i)[k\_0, ..,k\_{i-1}, k\_{i+1}, .., k\_{M-1}]` =

| the result of iterating `j` over :math:`range(N\_i)`, cumulatively applying

| ufunc to each :math:`a[k\_0, ..,k\_{i-1}, j, k\_{i+1}, .., k\_{M-1}]`.

| For a one-dimensional array, reduce produces results equivalent to:

| ::

|

| r = op.identity # op = ufunc

| for i in range(len(A)):

| r = op(r, A[i])

| return r

|

| For example, add.reduce() is equivalent to sum().

|

| Parameters

| ----------

| a : array\_like

| The array to act on.

| axis : None or int or tuple of ints, optional

| Axis or axes along which a reduction is performed.

| The default (`axis` = 0) is perform a reduction over the first

| dimension of the input array. `axis` may be negative, in

| which case it counts from the last to the first axis.

|

| .. versionadded:: 1.7.0

|

| If this is `None`, a reduction is performed over all the axes.

| If this is a tuple of ints, a reduction is performed on multiple

| axes, instead of a single axis or all the axes as before.

|

| For operations which are either not commutative or not associative,

| doing a reduction over multiple axes is not well-defined. The

| ufuncs do not currently raise an exception in this case, but will

| likely do so in the future.

| dtype : data-type code, optional

| The type used to represent the intermediate results. Defaults

| to the data-type of the output array if this is provided, or

| the data-type of the input array if no output array is provided.

| out : ndarray, None, or tuple of ndarray and None, optional

| A location into which the result is stored. If not provided or `None`,

| a freshly-allocated array is returned. For consistency with

| ``ufunc.\_\_call\_\_``, if given as a keyword, this may be wrapped in a

| 1-element tuple.

|

| .. versionchanged:: 1.13.0

| Tuples are allowed for keyword argument.

| keepdims : bool, optional

| If this is set to True, the axes which are reduced are left

| in the result as dimensions with size one. With this option,

| the result will broadcast correctly against the original `arr`.

|

| .. versionadded:: 1.7.0

| initial : scalar, optional

| The value with which to start the reduction.

| If the ufunc has no identity or the dtype is object, this defaults

| to None - otherwise it defaults to ufunc.identity.

| If ``None`` is given, the first element of the reduction is used,

| and an error is thrown if the reduction is empty.

|

| .. versionadded:: 1.15.0

|

| where : array\_like of bool, optional

| A boolean array which is broadcasted to match the dimensions

| of `a`, and selects elements to include in the reduction. Note

| that for ufuncs like ``minimum`` that do not have an identity

| defined, one has to pass in also ``initial``.

|

| .. versionadded:: 1.17.0

|

| Returns

| -------

| r : ndarray

| The reduced array. If `out` was supplied, `r` is a reference to it.

|

| Examples

| --------

| >>> np.multiply.reduce([2,3,5])

| 30

|

| A multi-dimensional array example:

|

| >>> X = np.arange(8).reshape((2,2,2))

| >>> X

| array([[[0, 1],

| [2, 3]],

| [[4, 5],

| [6, 7]]])

| >>> np.add.reduce(X, 0)

| array([[ 4, 6],

| [ 8, 10]])

| >>> np.add.reduce(X) # confirm: default axis value is 0

| array([[ 4, 6],

| [ 8, 10]])

| >>> np.add.reduce(X, 1)

| array([[ 2, 4],

| [10, 12]])

| >>> np.add.reduce(X, 2)

| array([[ 1, 5],

| [ 9, 13]])

|

| You can use the ``initial`` keyword argument to initialize the reduction

| with a different value, and ``where`` to select specific elements to include:

|

| >>> np.add.reduce([10], initial=5)

| 15

| >>> np.add.reduce(np.ones((2, 2, 2)), axis=(0, 2), initial=10)

| array([14., 14.])

| >>> a = np.array([10., np.nan, 10])

| >>> np.add.reduce(a, where=~np.isnan(a))

| 20.0

|

| Allows reductions of empty arrays where they would normally fail, i.e.

| for ufuncs without an identity.

|

| >>> np.minimum.reduce([], initial=np.inf)

| inf

| >>> np.minimum.reduce([[1., 2.], [3., 4.]], initial=10., where=[True, False])

| array([ 1., 10.])

| >>> np.minimum.reduce([])

| Traceback (most recent call last):

| ...

| ValueError: zero-size array to reduction operation minimum which has no identity

|

| reduceat(...)

| reduceat(a, indices, axis=0, dtype=None, out=None)

|

| Performs a (local) reduce with specified slices over a single axis.

|

| For i in ``range(len(indices))``, `reduceat` computes

| ``ufunc.reduce(a[indices[i]:indices[i+1]])``, which becomes the i-th

| generalized "row" parallel to `axis` in the final result (i.e., in a

| 2-D array, for example, if `axis = 0`, it becomes the i-th row, but if

| `axis = 1`, it becomes the i-th column). There are three exceptions to this:

|

| \* when ``i = len(indices) - 1`` (so for the last index),

| ``indices[i+1] = a.shape[axis]``.

| \* if ``indices[i] >= indices[i + 1]``, the i-th generalized "row" is

| simply ``a[indices[i]]``.

| \* if ``indices[i] >= len(a)`` or ``indices[i] < 0``, an error is raised.

|

| The shape of the output depends on the size of `indices`, and may be

| larger than `a` (this happens if ``len(indices) > a.shape[axis]``).

|

| Parameters

| ----------

| a : array\_like

| The array to act on.

| indices : array\_like

| Paired indices, comma separated (not colon), specifying slices to

| reduce.

| axis : int, optional

| The axis along which to apply the reduceat.

| dtype : data-type code, optional

| The type used to represent the intermediate results. Defaults

| to the data type of the output array if this is provided, or

| the data type of the input array if no output array is provided.

| out : ndarray, None, or tuple of ndarray and None, optional

| A location into which the result is stored. If not provided or `None`,

| a freshly-allocated array is returned. For consistency with

| ``ufunc.\_\_call\_\_``, if given as a keyword, this may be wrapped in a

| 1-element tuple.

|

| .. versionchanged:: 1.13.0

| Tuples are allowed for keyword argument.

|

| Returns

| -------

| r : ndarray

| The reduced values. If `out` was supplied, `r` is a reference to

| `out`.

|

| Notes

| -----

| A descriptive example:

|

| If `a` is 1-D, the function `ufunc.accumulate(a)` is the same as

| ``ufunc.reduceat(a, indices)[::2]`` where `indices` is

| ``range(len(array) - 1)`` with a zero placed

| in every other element:

| ``indices = zeros(2 \* len(a) - 1)``, ``indices[1::2] = range(1, len(a))``.

|

| Don't be fooled by this attribute's name: `reduceat(a)` is not

| necessarily smaller than `a`.

|

| Examples

| --------

| To take the running sum of four successive values:

|

| >>> np.add.reduceat(np.arange(8),[0,4, 1,5, 2,6, 3,7])[::2]

| array([ 6, 10, 14, 18])

|

| A 2-D example:

|

| >>> x = np.linspace(0, 15, 16).reshape(4,4)

| >>> x

| array([[ 0., 1., 2., 3.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.],

| [12., 13., 14., 15.]])

|

| ::

|

| # reduce such that the result has the following five rows:

| # [row1 + row2 + row3]

| # [row4]

| # [row2]

| # [row3]

| # [row1 + row2 + row3 + row4]

|

| >>> np.add.reduceat(x, [0, 3, 1, 2, 0])

| array([[12., 15., 18., 21.],

| [12., 13., 14., 15.],

| [ 4., 5., 6., 7.],

| [ 8., 9., 10., 11.],

| [24., 28., 32., 36.]])

|

| ::

|

| # reduce such that result has the following two columns:

| # [col1 \* col2 \* col3, col4]

|

| >>> np.multiply.reduceat(x, [0, 3], 1)

| array([[ 0., 3.],

| [ 120., 7.],

| [ 720., 11.],

| [2184., 15.]])

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| identity

| The identity value.

|

| Data attribute containing the identity element for the ufunc, if it has one.

| If it does not, the attribute value is None.

|

| Examples

| --------

| >>> np.add.identity

| 0

| >>> np.multiply.identity

| 1

| >>> np.power.identity

| 1

| >>> print(np.exp.identity)

| None

|

| nargs

| The number of arguments.

|

| Data attribute containing the number of arguments the ufunc takes, including

| optional ones.

|

| Notes

| -----

| Typically this value will be one more than what you might expect because all

| ufuncs take the optional "out" argument.

|

| Examples

| --------

| >>> np.add.nargs

| 3

| >>> np.multiply.nargs

| 3

| >>> np.power.nargs

| 3

| >>> np.exp.nargs

| 2

|

| nin

| The number of inputs.

|

| Data attribute containing the number of arguments the ufunc treats as input.

|

| Examples

| --------

| >>> np.add.nin

| 2

| >>> np.multiply.nin

| 2

| >>> np.power.nin

| 2

| >>> np.exp.nin

| 1

|

| nout

| The number of outputs.

|

| Data attribute containing the number of arguments the ufunc treats as output.

|

| Notes

| -----

| Since all ufuncs can take output arguments, this will always be (at least) 1.

|

| Examples

| --------

| >>> np.add.nout

| 1

| >>> np.multiply.nout

| 1

| >>> np.power.nout

| 1

| >>> np.exp.nout

| 1

|

| ntypes

| The number of types.

|

| The number of numerical NumPy types - of which there are 18 total - on which

| the ufunc can operate.

|

| See Also

| --------

| numpy.ufunc.types

|

| Examples

| --------

| >>> np.add.ntypes

| 18

| >>> np.multiply.ntypes

| 18

| >>> np.power.ntypes

| 17

| >>> np.exp.ntypes

| 7

| >>> np.remainder.ntypes

| 14

|

| signature

| Definition of the core elements a generalized ufunc operates on.

|

| The signature determines how the dimensions of each input/output array

| are split into core and loop dimensions:

|

| 1. Each dimension in the signature is matched to a dimension of the

| corresponding passed-in array, starting from the end of the shape tuple.

| 2. Core dimensions assigned to the same label in the signature must have

| exactly matching sizes, no broadcasting is performed.

| 3. The core dimensions are removed from all inputs and the remaining

| dimensions are broadcast together, defining the loop dimensions.

|

| Notes

| -----

| Generalized ufuncs are used internally in many linalg functions, and in

| the testing suite; the examples below are taken from these.

| For ufuncs that operate on scalars, the signature is `None`, which is

| equivalent to '()' for every argument.

|

| Examples

| --------

| >>> np.core.umath\_tests.matrix\_multiply.signature

| '(m,n),(n,p)->(m,p)'

| >>> np.linalg.\_umath\_linalg.det.signature

| '(m,m)->()'

| >>> np.add.signature is None

| True # equivalent to '(),()->()'

|

| types

| Returns a list with types grouped input->output.

|

| Data attribute listing the data-type "Domain-Range" groupings the ufunc can

| deliver. The data-types are given using the character codes.

|

| See Also

| --------

| numpy.ufunc.ntypes

|

| Examples

| --------

| >>> np.add.types

| ['??->?', 'bb->b', 'BB->B', 'hh->h', 'HH->H', 'ii->i', 'II->I', 'll->l',

| 'LL->L', 'qq->q', 'QQ->Q', 'ff->f', 'dd->d', 'gg->g', 'FF->F', 'DD->D',

| 'GG->G', 'OO->O']

|

| >>> np.multiply.types

| ['??->?', 'bb->b', 'BB->B', 'hh->h', 'HH->H', 'ii->i', 'II->I', 'll->l',

| 'LL->L', 'qq->q', 'QQ->Q', 'ff->f', 'dd->d', 'gg->g', 'FF->F', 'DD->D',

| 'GG->G', 'OO->O']

|

| >>> np.power.types

| ['bb->b', 'BB->B', 'hh->h', 'HH->H', 'ii->i', 'II->I', 'll->l', 'LL->L',

| 'qq->q', 'QQ->Q', 'ff->f', 'dd->d', 'gg->g', 'FF->F', 'DD->D', 'GG->G',

| 'OO->O']

|

| >>> np.exp.types

| ['f->f', 'd->d', 'g->g', 'F->F', 'D->D', 'G->G', 'O->O']

|

| >>> np.remainder.types

| ['bb->b', 'BB->B', 'hh->h', 'HH->H', 'ii->i', 'II->I', 'll->l', 'LL->L',

| 'qq->q', 'QQ->Q', 'ff->f', 'dd->d', 'gg->g', 'OO->O']

uint = class uint32(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned long``.

| Character code: ``'L'``.

| Canonical name: ``np.uint``.

| Alias \*on this platform\*: ``np.uint32``: 32-bit unsigned integer (0 to 4294967295).

|

| Method resolution order:

| uint32

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

uint0 = class uint32(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned int``.

| Character code: ``'I'``.

| Canonical name: ``np.uintc``.

| Alias \*on this platform\*: ``np.uintp``: Unsigned integer large enough to fit pointer, compatible with C ``uintptr\_t``.

|

| Method resolution order:

| uint32

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class uint16(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned short``.

| Character code: ``'H'``.

| Canonical name: ``np.ushort``.

| Alias \*on this platform\*: ``np.uint16``: 16-bit unsigned integer (0 to 65535).

|

| Method resolution order:

| uint16

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class uint32(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned long``.

| Character code: ``'L'``.

| Canonical name: ``np.uint``.

| Alias \*on this platform\*: ``np.uint32``: 32-bit unsigned integer (0 to 4294967295).

|

| Method resolution order:

| uint32

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class uint64(unsignedinteger)

| Signed integer type, compatible with C ``unsigned long long``.

| Character code: ``'Q'``.

| Canonical name: ``np.ulonglong``.

| Alias \*on this platform\*: ``np.uint64``: 64-bit unsigned integer (0 to 18446744073709551615).

|

| Method resolution order:

| uint64

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class uint8(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned char``.

| Character code: ``'B'``.

| Canonical name: ``np.ubyte``.

| Alias \*on this platform\*: ``np.uint8``: 8-bit unsigned integer (0 to 255).

|

| Method resolution order:

| uint8

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

uintc = class uint32(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned int``.

| Character code: ``'I'``.

| Canonical name: ``np.uintc``.

| Alias \*on this platform\*: ``np.uintp``: Unsigned integer large enough to fit pointer, compatible with C ``uintptr\_t``.

|

| Method resolution order:

| uint32

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

uintp = class uint32(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned int``.

| Character code: ``'I'``.

| Canonical name: ``np.uintc``.

| Alias \*on this platform\*: ``np.uintp``: Unsigned integer large enough to fit pointer, compatible with C ``uintptr\_t``.

|

| Method resolution order:

| uint32

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

ulonglong = class uint64(unsignedinteger)

| Signed integer type, compatible with C ``unsigned long long``.

| Character code: ``'Q'``.

| Canonical name: ``np.ulonglong``.

| Alias \*on this platform\*: ``np.uint64``: 64-bit unsigned integer (0 to 18446744073709551615).

|

| Method resolution order:

| uint64

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

unicode = class str(object)

| str(object='') -> str

| str(bytes\_or\_buffer[, encoding[, errors]]) -> str

|

| Create a new string object from the given object. If encoding or

| errors is specified, then the object must expose a data buffer

| that will be decoded using the given encoding and error handler.

| Otherwise, returns the result of object.\_\_str\_\_() (if defined)

| or repr(object).

| encoding defaults to sys.getdefaultencoding().

| errors defaults to 'strict'.

|

| Methods defined here:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_format\_\_(self, format\_spec, /)

| Return a formatted version of the string as described by format\_spec.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_sizeof\_\_(self, /)

| Return the size of the string in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| capitalize(self, /)

| Return a capitalized version of the string.

|

| More specifically, make the first character have upper case and the rest lower

| case.

|

| casefold(self, /)

| Return a version of the string suitable for caseless comparisons.

|

| center(self, width, fillchar=' ', /)

| Return a centered string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| count(...)

| S.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of substring sub in

| string S[start:end]. Optional arguments start and end are

| interpreted as in slice notation.

|

| encode(self, /, encoding='utf-8', errors='strict')

| Encode the string using the codec registered for encoding.

|

| encoding

| The encoding in which to encode the string.

| errors

| The error handling scheme to use for encoding errors.

| The default is 'strict' meaning that encoding errors raise a

| UnicodeEncodeError. Other possible values are 'ignore', 'replace' and

| 'xmlcharrefreplace' as well as any other name registered with

| codecs.register\_error that can handle UnicodeEncodeErrors.

|

| endswith(...)

| S.endswith(suffix[, start[, end]]) -> bool

|

| Return True if S ends with the specified suffix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| suffix can also be a tuple of strings to try.

|

| expandtabs(self, /, tabsize=8)

| Return a copy where all tab characters are expanded using spaces.

|

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| S.find(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| format(...)

| S.format(\*args, \*\*kwargs) -> str

|

| Return a formatted version of S, using substitutions from args and kwargs.

| The substitutions are identified by braces ('{' and '}').

|

| format\_map(...)

| S.format\_map(mapping) -> str

|

| Return a formatted version of S, using substitutions from mapping.

| The substitutions are identified by braces ('{' and '}').

|

| index(...)

| S.index(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| isalnum(self, /)

| Return True if the string is an alpha-numeric string, False otherwise.

|

| A string is alpha-numeric if all characters in the string are alpha-numeric and

| there is at least one character in the string.

|

| isalpha(self, /)

| Return True if the string is an alphabetic string, False otherwise.

|

| A string is alphabetic if all characters in the string are alphabetic and there

| is at least one character in the string.

|

| isascii(self, /)

| Return True if all characters in the string are ASCII, False otherwise.

|

| ASCII characters have code points in the range U+0000-U+007F.

| Empty string is ASCII too.

|

| isdecimal(self, /)

| Return True if the string is a decimal string, False otherwise.

|

| A string is a decimal string if all characters in the string are decimal and

| there is at least one character in the string.

|

| isdigit(self, /)

| Return True if the string is a digit string, False otherwise.

|

| A string is a digit string if all characters in the string are digits and there

| is at least one character in the string.

|

| isidentifier(self, /)

| Return True if the string is a valid Python identifier, False otherwise.

|

| Use keyword.iskeyword() to test for reserved identifiers such as "def" and

| "class".

|

| islower(self, /)

| Return True if the string is a lowercase string, False otherwise.

|

| A string is lowercase if all cased characters in the string are lowercase and

| there is at least one cased character in the string.

|

| isnumeric(self, /)

| Return True if the string is a numeric string, False otherwise.

|

| A string is numeric if all characters in the string are numeric and there is at

| least one character in the string.

|

| isprintable(self, /)

| Return True if the string is printable, False otherwise.

|

| A string is printable if all of its characters are considered printable in

| repr() or if it is empty.

|

| isspace(self, /)

| Return True if the string is a whitespace string, False otherwise.

|

| A string is whitespace if all characters in the string are whitespace and there

| is at least one character in the string.

|

| istitle(self, /)

| Return True if the string is a title-cased string, False otherwise.

|

| In a title-cased string, upper- and title-case characters may only

| follow uncased characters and lowercase characters only cased ones.

|

| isupper(self, /)

| Return True if the string is an uppercase string, False otherwise.

|

| A string is uppercase if all cased characters in the string are uppercase and

| there is at least one cased character in the string.

|

| join(self, iterable, /)

| Concatenate any number of strings.

|

| The string whose method is called is inserted in between each given string.

| The result is returned as a new string.

|

| Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

|

| ljust(self, width, fillchar=' ', /)

| Return a left-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| lower(self, /)

| Return a copy of the string converted to lowercase.

|

| lstrip(self, chars=None, /)

| Return a copy of the string with leading whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| partition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original string

| and two empty strings.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| S.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| S.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| rjust(self, width, fillchar=' ', /)

| Return a right-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| rpartition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty strings

| and the original string.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splits are done starting at the end of the string and working to the front.

|

| rstrip(self, chars=None, /)

| Return a copy of the string with trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the string, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| S.startswith(prefix[, start[, end]]) -> bool

|

| Return True if S starts with the specified prefix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| prefix can also be a tuple of strings to try.

|

| strip(self, chars=None, /)

| Return a copy of the string with leading and trailing whitespace remove.

|

| If chars is given and not None, remove characters in chars instead.

|

| swapcase(self, /)

| Convert uppercase characters to lowercase and lowercase characters to uppercase.

|

| title(self, /)

| Return a version of the string where each word is titlecased.

|

| More specifically, words start with uppercased characters and all remaining

| cased characters have lower case.

|

| translate(self, table, /)

| Replace each character in the string using the given translation table.

|

| table

| Translation table, which must be a mapping of Unicode ordinals to

| Unicode ordinals, strings, or None.

|

| The table must implement lookup/indexing via \_\_getitem\_\_, for instance a

| dictionary or list. If this operation raises LookupError, the character is

| left untouched. Characters mapped to None are deleted.

|

| upper(self, /)

| Return a copy of the string converted to uppercase.

|

| zfill(self, width, /)

| Pad a numeric string with zeros on the left, to fill a field of the given width.

|

| The string is never truncated.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| maketrans(x, y=None, z=None, /)

| Return a translation table usable for str.translate().

|

| If there is only one argument, it must be a dictionary mapping Unicode

| ordinals (integers) or characters to Unicode ordinals, strings or None.

| Character keys will be then converted to ordinals.

| If there are two arguments, they must be strings of equal length, and

| in the resulting dictionary, each character in x will be mapped to the

| character at the same position in y. If there is a third argument, it

| must be a string, whose characters will be mapped to None in the result.

unicode\_ = class str\_(builtins.str, character)

| str(object='') -> str

| str(bytes\_or\_buffer[, encoding[, errors]]) -> str

|

| Create a new string object from the given object. If encoding or

| errors is specified, then the object must expose a data buffer

| that will be decoded using the given encoding and error handler.

| Otherwise, returns the result of object.\_\_str\_\_() (if defined)

| or repr(object).

| encoding defaults to sys.getdefaultencoding().

| errors defaults to 'strict'.

|

| Method resolution order:

| str\_

| builtins.str

| character

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Methods inherited from builtins.str:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_format\_\_(self, format\_spec, /)

| Return a formatted version of the string as described by format\_spec.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_sizeof\_\_(self, /)

| Return the size of the string in memory, in bytes.

|

| capitalize(self, /)

| Return a capitalized version of the string.

|

| More specifically, make the first character have upper case and the rest lower

| case.

|

| casefold(self, /)

| Return a version of the string suitable for caseless comparisons.

|

| center(self, width, fillchar=' ', /)

| Return a centered string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| count(...)

| S.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of substring sub in

| string S[start:end]. Optional arguments start and end are

| interpreted as in slice notation.

|

| encode(self, /, encoding='utf-8', errors='strict')

| Encode the string using the codec registered for encoding.

|

| encoding

| The encoding in which to encode the string.

| errors

| The error handling scheme to use for encoding errors.

| The default is 'strict' meaning that encoding errors raise a

| UnicodeEncodeError. Other possible values are 'ignore', 'replace' and

| 'xmlcharrefreplace' as well as any other name registered with

| codecs.register\_error that can handle UnicodeEncodeErrors.

|

| endswith(...)

| S.endswith(suffix[, start[, end]]) -> bool

|

| Return True if S ends with the specified suffix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| suffix can also be a tuple of strings to try.

|

| expandtabs(self, /, tabsize=8)

| Return a copy where all tab characters are expanded using spaces.

|

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| S.find(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| format(...)

| S.format(\*args, \*\*kwargs) -> str

|

| Return a formatted version of S, using substitutions from args and kwargs.

| The substitutions are identified by braces ('{' and '}').

|

| format\_map(...)

| S.format\_map(mapping) -> str

|

| Return a formatted version of S, using substitutions from mapping.

| The substitutions are identified by braces ('{' and '}').

|

| index(...)

| S.index(sub[, start[, end]]) -> int

|

| Return the lowest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| isalnum(self, /)

| Return True if the string is an alpha-numeric string, False otherwise.

|

| A string is alpha-numeric if all characters in the string are alpha-numeric and

| there is at least one character in the string.

|

| isalpha(self, /)

| Return True if the string is an alphabetic string, False otherwise.

|

| A string is alphabetic if all characters in the string are alphabetic and there

| is at least one character in the string.

|

| isascii(self, /)

| Return True if all characters in the string are ASCII, False otherwise.

|

| ASCII characters have code points in the range U+0000-U+007F.

| Empty string is ASCII too.

|

| isdecimal(self, /)

| Return True if the string is a decimal string, False otherwise.

|

| A string is a decimal string if all characters in the string are decimal and

| there is at least one character in the string.

|

| isdigit(self, /)

| Return True if the string is a digit string, False otherwise.

|

| A string is a digit string if all characters in the string are digits and there

| is at least one character in the string.

|

| isidentifier(self, /)

| Return True if the string is a valid Python identifier, False otherwise.

|

| Use keyword.iskeyword() to test for reserved identifiers such as "def" and

| "class".

|

| islower(self, /)

| Return True if the string is a lowercase string, False otherwise.

|

| A string is lowercase if all cased characters in the string are lowercase and

| there is at least one cased character in the string.

|

| isnumeric(self, /)

| Return True if the string is a numeric string, False otherwise.

|

| A string is numeric if all characters in the string are numeric and there is at

| least one character in the string.

|

| isprintable(self, /)

| Return True if the string is printable, False otherwise.

|

| A string is printable if all of its characters are considered printable in

| repr() or if it is empty.

|

| isspace(self, /)

| Return True if the string is a whitespace string, False otherwise.

|

| A string is whitespace if all characters in the string are whitespace and there

| is at least one character in the string.

|

| istitle(self, /)

| Return True if the string is a title-cased string, False otherwise.

|

| In a title-cased string, upper- and title-case characters may only

| follow uncased characters and lowercase characters only cased ones.

|

| isupper(self, /)

| Return True if the string is an uppercase string, False otherwise.

|

| A string is uppercase if all cased characters in the string are uppercase and

| there is at least one cased character in the string.

|

| join(self, iterable, /)

| Concatenate any number of strings.

|

| The string whose method is called is inserted in between each given string.

| The result is returned as a new string.

|

| Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

|

| ljust(self, width, fillchar=' ', /)

| Return a left-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| lower(self, /)

| Return a copy of the string converted to lowercase.

|

| lstrip(self, chars=None, /)

| Return a copy of the string with leading whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| partition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original string

| and two empty strings.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| S.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Return -1 on failure.

|

| rindex(...)

| S.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in S where substring sub is found,

| such that sub is contained within S[start:end]. Optional

| arguments start and end are interpreted as in slice notation.

|

| Raises ValueError when the substring is not found.

|

| rjust(self, width, fillchar=' ', /)

| Return a right-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| rpartition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty strings

| and the original string.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splits are done starting at the end of the string and working to the front.

|

| rstrip(self, chars=None, /)

| Return a copy of the string with trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| Return a list of the lines in the string, breaking at line boundaries.

|

| Line breaks are not included in the resulting list unless keepends is given and

| true.

|

| startswith(...)

| S.startswith(prefix[, start[, end]]) -> bool

|

| Return True if S starts with the specified prefix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| prefix can also be a tuple of strings to try.

|

| strip(self, chars=None, /)

| Return a copy of the string with leading and trailing whitespace remove.

|

| If chars is given and not None, remove characters in chars instead.

|

| swapcase(self, /)

| Convert uppercase characters to lowercase and lowercase characters to uppercase.

|

| title(self, /)

| Return a version of the string where each word is titlecased.

|

| More specifically, words start with uppercased characters and all remaining

| cased characters have lower case.

|

| translate(self, table, /)

| Replace each character in the string using the given translation table.

|

| table

| Translation table, which must be a mapping of Unicode ordinals to

| Unicode ordinals, strings, or None.

|

| The table must implement lookup/indexing via \_\_getitem\_\_, for instance a

| dictionary or list. If this operation raises LookupError, the character is

| left untouched. Characters mapped to None are deleted.

|

| upper(self, /)

| Return a copy of the string converted to uppercase.

|

| zfill(self, width, /)

| Pad a numeric string with zeros on the left, to fill a field of the given width.

|

| The string is never truncated.

|

| ----------------------------------------------------------------------

| Static methods inherited from builtins.str:

|

| maketrans(x, y=None, z=None, /)

| Return a translation table usable for str.translate().

|

| If there is only one argument, it must be a dictionary mapping Unicode

| ordinals (integers) or characters to Unicode ordinals, strings or None.

| Character keys will be then converted to ordinals.

| If there are two arguments, they must be strings of equal length, and

| in the resulting dictionary, each character in x will be mapped to the

| character at the same position in y. If there is a third argument, it

| must be a string, whose characters will be mapped to None in the result.

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class unsignedinteger(integer)

| Abstract base class of all unsigned integer scalar types.

|

| Method resolution order:

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from generic:

|

| \_\_hash\_\_ = None

ushort = class uint16(unsignedinteger)

| Unsigned integer type, compatible with C ``unsigned short``.

| Character code: ``'H'``.

| Canonical name: ``np.ushort``.

| Alias \*on this platform\*: ``np.uint16``: 16-bit unsigned integer (0 to 65535).

|

| Method resolution order:

| uint16

| unsignedinteger

| integer

| number

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_index\_\_(self, /)

| Return self converted to an integer, if self is suitable for use as an index into a list.

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from integer:

|

| denominator

| denominator of value (1)

|

| numerator

| numerator of value (the value itself)

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_round\_\_(...)

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| base

| base object

|

| data

| pointer to start of data

|

| dtype

| get array data-descriptor

|

| flags

| integer value of flags

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

class vectorize(builtins.object)

| vectorize(pyfunc, otypes=None, doc=None, excluded=None, cache=False, signature=None)

|

| vectorize(pyfunc, otypes=None, doc=None, excluded=None, cache=False,

| signature=None)

|

| Generalized function class.

|

| Define a vectorized function which takes a nested sequence of objects or

| numpy arrays as inputs and returns a single numpy array or a tuple of numpy

| arrays. The vectorized function evaluates `pyfunc` over successive tuples

| of the input arrays like the python map function, except it uses the

| broadcasting rules of numpy.

|

| The data type of the output of `vectorized` is determined by calling

| the function with the first element of the input. This can be avoided

| by specifying the `otypes` argument.

|

| Parameters

| ----------

| pyfunc : callable

| A python function or method.

| otypes : str or list of dtypes, optional

| The output data type. It must be specified as either a string of

| typecode characters or a list of data type specifiers. There should

| be one data type specifier for each output.

| doc : str, optional

| The docstring for the function. If `None`, the docstring will be the

| ``pyfunc.\_\_doc\_\_``.

| excluded : set, optional

| Set of strings or integers representing the positional or keyword

| arguments for which the function will not be vectorized. These will be

| passed directly to `pyfunc` unmodified.

|

| .. versionadded:: 1.7.0

|

| cache : bool, optional

| If `True`, then cache the first function call that determines the number

| of outputs if `otypes` is not provided.

|

| .. versionadded:: 1.7.0

|

| signature : string, optional

| Generalized universal function signature, e.g., ``(m,n),(n)->(m)`` for

| vectorized matrix-vector multiplication. If provided, ``pyfunc`` will

| be called with (and expected to return) arrays with shapes given by the

| size of corresponding core dimensions. By default, ``pyfunc`` is

| assumed to take scalars as input and output.

|

| .. versionadded:: 1.12.0

|

| Returns

| -------

| vectorized : callable

| Vectorized function.

|

| See Also

| --------

| frompyfunc : Takes an arbitrary Python function and returns a ufunc

|

| Notes

| -----

| The `vectorize` function is provided primarily for convenience, not for

| performance. The implementation is essentially a for loop.

|

| If `otypes` is not specified, then a call to the function with the

| first argument will be used to determine the number of outputs. The

| results of this call will be cached if `cache` is `True` to prevent

| calling the function twice. However, to implement the cache, the

| original function must be wrapped which will slow down subsequent

| calls, so only do this if your function is expensive.

|

| The new keyword argument interface and `excluded` argument support

| further degrades performance.

|

| References

| ----------

| .. [1] NumPy Reference, section `Generalized Universal Function API

| <https://docs.scipy.org/doc/numpy/reference/c-api.generalized-ufuncs.html>`\_.

|

| Examples

| --------

| >>> def myfunc(a, b):

| ... "Return a-b if a>b, otherwise return a+b"

| ... if a > b:

| ... return a - b

| ... else:

| ... return a + b

|

| >>> vfunc = np.vectorize(myfunc)

| >>> vfunc([1, 2, 3, 4], 2)

| array([3, 4, 1, 2])

|

| The docstring is taken from the input function to `vectorize` unless it

| is specified:

|

| >>> vfunc.\_\_doc\_\_

| 'Return a-b if a>b, otherwise return a+b'

| >>> vfunc = np.vectorize(myfunc, doc='Vectorized `myfunc`')

| >>> vfunc.\_\_doc\_\_

| 'Vectorized `myfunc`'

|

| The output type is determined by evaluating the first element of the input,

| unless it is specified:

|

| >>> out = vfunc([1, 2, 3, 4], 2)

| >>> type(out[0])

| <class 'numpy.int64'>

| >>> vfunc = np.vectorize(myfunc, otypes=[float])

| >>> out = vfunc([1, 2, 3, 4], 2)

| >>> type(out[0])

| <class 'numpy.float64'>

|

| The `excluded` argument can be used to prevent vectorizing over certain

| arguments. This can be useful for array-like arguments of a fixed length

| such as the coefficients for a polynomial as in `polyval`:

|

| >>> def mypolyval(p, x):

| ... \_p = list(p)

| ... res = \_p.pop(0)

| ... while \_p:

| ... res = res\*x + \_p.pop(0)

| ... return res

| >>> vpolyval = np.vectorize(mypolyval, excluded=['p'])

| >>> vpolyval(p=[1, 2, 3], x=[0, 1])

| array([3, 6])

|

| Positional arguments may also be excluded by specifying their position:

|

| >>> vpolyval.excluded.add(0)

| >>> vpolyval([1, 2, 3], x=[0, 1])

| array([3, 6])

|

| The `signature` argument allows for vectorizing functions that act on

| non-scalar arrays of fixed length. For example, you can use it for a

| vectorized calculation of Pearson correlation coefficient and its p-value:

|

| >>> import scipy.stats

| >>> pearsonr = np.vectorize(scipy.stats.pearsonr,

| ... signature='(n),(n)->(),()')

| >>> pearsonr([[0, 1, 2, 3]], [[1, 2, 3, 4], [4, 3, 2, 1]])

| (array([ 1., -1.]), array([ 0., 0.]))

|

| Or for a vectorized convolution:

|

| >>> convolve = np.vectorize(np.convolve, signature='(n),(m)->(k)')

| >>> convolve(np.eye(4), [1, 2, 1])

| array([[1., 2., 1., 0., 0., 0.],

| [0., 1., 2., 1., 0., 0.],

| [0., 0., 1., 2., 1., 0.],

| [0., 0., 0., 1., 2., 1.]])

|

| Methods defined here:

|

| \_\_call\_\_(self, \*args, \*\*kwargs)

| Return arrays with the results of `pyfunc` broadcast (vectorized) over

| `args` and `kwargs` not in `excluded`.

|

| \_\_init\_\_(self, pyfunc, otypes=None, doc=None, excluded=None, cache=False, signature=None)

| Initialize self. See help(type(self)) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

class void(flexible)

| Abstract base class of all scalar types without predefined length.

| The actual size of these types depends on the specific `np.dtype`

| instantiation.

|

| Method resolution order:

| void

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| base

| base object

|

| dtype

| dtype object

|

| flags

| integer value of flags

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| data

| pointer to start of data

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

void0 = class void(flexible)

| Abstract base class of all scalar types without predefined length.

| The actual size of these types depends on the specific `np.dtype`

| instantiation.

|

| Method resolution order:

| void

| flexible

| generic

| builtins.object

|

| Methods defined here:

|

| \_\_delitem\_\_(self, key, /)

| Delete self[key].

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setitem\_\_(self, key, value, /)

| Set self[key] to value.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| getfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setfield(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| base

| base object

|

| dtype

| dtype object

|

| flags

| integer value of flags

|

| ----------------------------------------------------------------------

| Methods inherited from generic:

|

| \_\_abs\_\_(self, /)

| abs(self)

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_and\_\_(self, value, /)

| Return self&value.

|

| \_\_array\_\_(...)

| sc.\_\_array\_\_(dtype) return 0-dim array from scalar with specified dtype

|

| \_\_array\_wrap\_\_(...)

| sc.\_\_array\_wrap\_\_(obj) return scalar from array

|

| \_\_bool\_\_(self, /)

| self != 0

|

| \_\_copy\_\_(...)

|

| \_\_deepcopy\_\_(...)

|

| \_\_divmod\_\_(self, value, /)

| Return divmod(self, value).

|

| \_\_float\_\_(self, /)

| float(self)

|

| \_\_floordiv\_\_(self, value, /)

| Return self//value.

|

| \_\_format\_\_(...)

| NumPy array scalar formatter

|

| \_\_int\_\_(self, /)

| int(self)

|

| \_\_invert\_\_(self, /)

| ~self

|

| \_\_lshift\_\_(self, value, /)

| Return self<<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_neg\_\_(self, /)

| -self

|

| \_\_or\_\_(self, value, /)

| Return self|value.

|

| \_\_pos\_\_(self, /)

| +self

|

| \_\_pow\_\_(self, value, mod=None, /)

| Return pow(self, value, mod).

|

| \_\_radd\_\_(self, value, /)

| Return value+self.

|

| \_\_rand\_\_(self, value, /)

| Return value&self.

|

| \_\_rdivmod\_\_(self, value, /)

| Return divmod(value, self).

|

| \_\_reduce\_\_(...)

| Helper for pickle.

|

| \_\_rfloordiv\_\_(self, value, /)

| Return value//self.

|

| \_\_rlshift\_\_(self, value, /)

| Return value<<self.

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_ror\_\_(self, value, /)

| Return value|self.

|

| \_\_round\_\_(...)

|

| \_\_rpow\_\_(self, value, mod=None, /)

| Return pow(value, self, mod).

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| \_\_rsub\_\_(self, value, /)

| Return value-self.

|

| \_\_rtruediv\_\_(self, value, /)

| Return value/self.

|

| \_\_rxor\_\_(self, value, /)

| Return value^self.

|

| \_\_setstate\_\_(...)

|

| \_\_sizeof\_\_(...)

| Size of object in memory, in bytes.

|

| \_\_sub\_\_(self, value, /)

| Return self-value.

|

| \_\_truediv\_\_(self, value, /)

| Return self/value.

|

| \_\_xor\_\_(self, value, /)

| Return self^value.

|

| all(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| any(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmax(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argmin(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| argsort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| astype(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| byteswap(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| choose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| clip(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| compress(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| conj(...)

|

| conjugate(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| copy(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumprod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| cumsum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| diagonal(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dump(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| dumps(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| fill(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| flatten(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| item(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| itemset(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| max(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| mean(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| min(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| newbyteorder(...)

| newbyteorder(new\_order='S')

|

| Return a new `dtype` with a different byte order.

|

| Changes are also made in all fields and sub-arrays of the data type.

|

| The `new\_order` code can be any from the following:

|

| \* 'S' - swap dtype from current to opposite endian

| \* {'<', 'L'} - little endian

| \* {'>', 'B'} - big endian

| \* {'=', 'N'} - native order

| \* {'|', 'I'} - ignore (no change to byte order)

|

| Parameters

| ----------

| new\_order : str, optional

| Byte order to force; a value from the byte order specifications

| above. The default value ('S') results in swapping the current

| byte order. The code does a case-insensitive check on the first

| letter of `new\_order` for the alternatives above. For example,

| any of 'B' or 'b' or 'biggish' are valid to specify big-endian.

|

|

| Returns

| -------

| new\_dtype : dtype

| New `dtype` object with the given change to the byte order.

|

| nonzero(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| prod(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ptp(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| put(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ravel(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| repeat(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| reshape(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| resize(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| round(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| searchsorted(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| setflags(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class so as to

| provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sort(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| squeeze(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| std(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| sum(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| swapaxes(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| take(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tobytes(...)

|

| tofile(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tolist(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| tostring(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| trace(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| transpose(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| var(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| view(...)

| Not implemented (virtual attribute)

|

| Class generic exists solely to derive numpy scalars from, and possesses,

| albeit unimplemented, all the attributes of the ndarray class

| so as to provide a uniform API.

|

| See also the corresponding attribute of the derived class of interest.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from generic:

|

| T

| transpose

|

| \_\_array\_interface\_\_

| Array protocol: Python side

|

| \_\_array\_priority\_\_

| Array priority.

|

| \_\_array\_struct\_\_

| Array protocol: struct

|

| data

| pointer to start of data

|

| flat

| a 1-d view of scalar

|

| imag

| imaginary part of scalar

|

| itemsize

| length of one element in bytes

|

| nbytes

| length of item in bytes

|

| ndim

| number of array dimensions

|

| real

| real part of scalar

|

| shape

| tuple of array dimensions

|

| size

| number of elements in the gentype

|

| strides

| tuple of bytes steps in each dimension

FUNCTIONS

\_add\_newdoc\_ufunc(...)

add\_ufunc\_docstring(ufunc, new\_docstring)

Replace the docstring for a ufunc with new\_docstring.

This method will only work if the current docstring for

the ufunc is NULL. (At the C level, i.e. when ufunc->doc is NULL.)

Parameters

----------

ufunc : numpy.ufunc

A ufunc whose current doc is NULL.

new\_docstring : string

The new docstring for the ufunc.

Notes

-----

This method allocates memory for new\_docstring on

the heap. Technically this creates a mempory leak, since this

memory will not be reclaimed until the end of the program

even if the ufunc itself is removed. However this will only

be a problem if the user is repeatedly creating ufuncs with

no documentation, adding documentation via add\_newdoc\_ufunc,

and then throwing away the ufunc.

add\_docstring(...)

add\_docstring(obj, docstring)

Add a docstring to a built-in obj if possible.

If the obj already has a docstring raise a RuntimeError

If this routine does not know how to add a docstring to the object

raise a TypeError

add\_newdoc(place, obj, doc)

Adds documentation to obj which is in module place.

If doc is a string add it to obj as a docstring

If doc is a tuple, then the first element is interpreted as

an attribute of obj and the second as the docstring

(method, docstring)

If doc is a list, then each element of the list should be a

sequence of length two --> [(method1, docstring1),

(method2, docstring2), ...]

This routine never raises an error if the docstring can't be written, but

will raise an error if the object being documented does not exist.

This routine cannot modify read-only docstrings, as appear

in new-style classes or built-in functions. Because this

routine never raises an error the caller must check manually

that the docstrings were changed.

add\_newdoc\_ufunc = \_add\_newdoc\_ufunc(...)

add\_ufunc\_docstring(ufunc, new\_docstring)

Replace the docstring for a ufunc with new\_docstring.

This method will only work if the current docstring for

the ufunc is NULL. (At the C level, i.e. when ufunc->doc is NULL.)

Parameters

----------

ufunc : numpy.ufunc

A ufunc whose current doc is NULL.

new\_docstring : string

The new docstring for the ufunc.

Notes

-----

This method allocates memory for new\_docstring on

the heap. Technically this creates a mempory leak, since this

memory will not be reclaimed until the end of the program

even if the ufunc itself is removed. However this will only

be a problem if the user is repeatedly creating ufuncs with

no documentation, adding documentation via add\_newdoc\_ufunc,

and then throwing away the ufunc.

alen(a)

Return the length of the first dimension of the input array.

Parameters

----------

a : array\_like

Input array.

Returns

-------

alen : int

Length of the first dimension of `a`.

See Also

--------

shape, size

Examples

--------

>>> a = np.zeros((7,4,5))

>>> a.shape[0]

7

>>> np.alen(a)

7

all(a, axis=None, out=None, keepdims=<no value>)

Test whether all array elements along a given axis evaluate to True.

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

axis : None or int or tuple of ints, optional

Axis or axes along which a logical AND reduction is performed.

The default (`axis` = `None`) is to perform a logical AND over all

the dimensions of the input array. `axis` may be negative, in

which case it counts from the last to the first axis.

.. versionadded:: 1.7.0

If this is a tuple of ints, a reduction is performed on multiple

axes, instead of a single axis or all the axes as before.

out : ndarray, optional

Alternate output array in which to place the result.

It must have the same shape as the expected output and its

type is preserved (e.g., if ``dtype(out)`` is float, the result

will consist of 0.0's and 1.0's). See `doc.ufuncs` (Section

"Output arguments") for more details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `all` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

all : ndarray, bool

A new boolean or array is returned unless `out` is specified,

in which case a reference to `out` is returned.

See Also

--------

ndarray.all : equivalent method

any : Test whether any element along a given axis evaluates to True.

Notes

-----

Not a Number (NaN), positive infinity and negative infinity

evaluate to `True` because these are not equal to zero.

Examples

--------

>>> np.all([[True,False],[True,True]])

False

>>> np.all([[True,False],[True,True]], axis=0)

array([ True, False])

>>> np.all([-1, 4, 5])

True

>>> np.all([1.0, np.nan])

True

>>> o=np.array(False)

>>> z=np.all([-1, 4, 5], out=o)

>>> id(z), id(o), z

(28293632, 28293632, array(True)) # may vary

allclose(a, b, rtol=1e-05, atol=1e-08, equal\_nan=False)

Returns True if two arrays are element-wise equal within a tolerance.

The tolerance values are positive, typically very small numbers. The

relative difference (`rtol` \* abs(`b`)) and the absolute difference

`atol` are added together to compare against the absolute difference

between `a` and `b`.

If either array contains one or more NaNs, False is returned.

Infs are treated as equal if they are in the same place and of the same

sign in both arrays.

Parameters

----------

a, b : array\_like

Input arrays to compare.

rtol : float

The relative tolerance parameter (see Notes).

atol : float

The absolute tolerance parameter (see Notes).

equal\_nan : bool

Whether to compare NaN's as equal. If True, NaN's in `a` will be

considered equal to NaN's in `b` in the output array.

.. versionadded:: 1.10.0

Returns

-------

allclose : bool

Returns True if the two arrays are equal within the given

tolerance; False otherwise.

See Also

--------

isclose, all, any, equal

Notes

-----

If the following equation is element-wise True, then allclose returns

True.

absolute(`a` - `b`) <= (`atol` + `rtol` \* absolute(`b`))

The above equation is not symmetric in `a` and `b`, so that

``allclose(a, b)`` might be different from ``allclose(b, a)`` in

some rare cases.

The comparison of `a` and `b` uses standard broadcasting, which

means that `a` and `b` need not have the same shape in order for

``allclose(a, b)`` to evaluate to True. The same is true for

`equal` but not `array\_equal`.

Examples

--------

>>> np.allclose([1e10,1e-7], [1.00001e10,1e-8])

False

>>> np.allclose([1e10,1e-8], [1.00001e10,1e-9])

True

>>> np.allclose([1e10,1e-8], [1.0001e10,1e-9])

False

>>> np.allclose([1.0, np.nan], [1.0, np.nan])

False

>>> np.allclose([1.0, np.nan], [1.0, np.nan], equal\_nan=True)

True

alltrue(\*args, \*\*kwargs)

Check if all elements of input array are true.

See Also

--------

numpy.all : Equivalent function; see for details.

amax(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)

Return the maximum of an array or maximum along an axis.

Parameters

----------

a : array\_like

Input data.

axis : None or int or tuple of ints, optional

Axis or axes along which to operate. By default, flattened input is

used.

.. versionadded:: 1.7.0

If this is a tuple of ints, the maximum is selected over multiple axes,

instead of a single axis or all the axes as before.

out : ndarray, optional

Alternative output array in which to place the result. Must

be of the same shape and buffer length as the expected output.

See `doc.ufuncs` (Section "Output arguments") for more details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `amax` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

initial : scalar, optional

The minimum value of an output element. Must be present to allow

computation on empty slice. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.15.0

where : array\_like of bool, optional

Elements to compare for the maximum. See `~numpy.ufunc.reduce`

for details.

.. versionadded:: 1.17.0

Returns

-------

amax : ndarray or scalar

Maximum of `a`. If `axis` is None, the result is a scalar value.

If `axis` is given, the result is an array of dimension

``a.ndim - 1``.

See Also

--------

amin :

The minimum value of an array along a given axis, propagating any NaNs.

nanmax :

The maximum value of an array along a given axis, ignoring any NaNs.

maximum :

Element-wise maximum of two arrays, propagating any NaNs.

fmax :

Element-wise maximum of two arrays, ignoring any NaNs.

argmax :

Return the indices of the maximum values.

nanmin, minimum, fmin

Notes

-----

NaN values are propagated, that is if at least one item is NaN, the

corresponding max value will be NaN as well. To ignore NaN values

(MATLAB behavior), please use nanmax.

Don't use `amax` for element-wise comparison of 2 arrays; when

``a.shape[0]`` is 2, ``maximum(a[0], a[1])`` is faster than

``amax(a, axis=0)``.

Examples

--------

>>> a = np.arange(4).reshape((2,2))

>>> a

array([[0, 1],

[2, 3]])

>>> np.amax(a) # Maximum of the flattened array

3

>>> np.amax(a, axis=0) # Maxima along the first axis

array([2, 3])

>>> np.amax(a, axis=1) # Maxima along the second axis

array([1, 3])

>>> np.amax(a, where=[False, True], initial=-1, axis=0)

array([-1, 3])

>>> b = np.arange(5, dtype=float)

>>> b[2] = np.NaN

>>> np.amax(b)

nan

>>> np.amax(b, where=~np.isnan(b), initial=-1)

4.0

>>> np.nanmax(b)

4.0

You can use an initial value to compute the maximum of an empty slice, or

to initialize it to a different value:

>>> np.max([[-50], [10]], axis=-1, initial=0)

array([ 0, 10])

Notice that the initial value is used as one of the elements for which the

maximum is determined, unlike for the default argument Python's max

function, which is only used for empty iterables.

>>> np.max([5], initial=6)

6

>>> max([5], default=6)

5

amin(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)

Return the minimum of an array or minimum along an axis.

Parameters

----------

a : array\_like

Input data.

axis : None or int or tuple of ints, optional

Axis or axes along which to operate. By default, flattened input is

used.

.. versionadded:: 1.7.0

If this is a tuple of ints, the minimum is selected over multiple axes,

instead of a single axis or all the axes as before.

out : ndarray, optional

Alternative output array in which to place the result. Must

be of the same shape and buffer length as the expected output.

See `doc.ufuncs` (Section "Output arguments") for more details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `amin` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

initial : scalar, optional

The maximum value of an output element. Must be present to allow

computation on empty slice. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.15.0

where : array\_like of bool, optional

Elements to compare for the minimum. See `~numpy.ufunc.reduce`

for details.

.. versionadded:: 1.17.0

Returns

-------

amin : ndarray or scalar

Minimum of `a`. If `axis` is None, the result is a scalar value.

If `axis` is given, the result is an array of dimension

``a.ndim - 1``.

See Also

--------

amax :

The maximum value of an array along a given axis, propagating any NaNs.

nanmin :

The minimum value of an array along a given axis, ignoring any NaNs.

minimum :

Element-wise minimum of two arrays, propagating any NaNs.

fmin :

Element-wise minimum of two arrays, ignoring any NaNs.

argmin :

Return the indices of the minimum values.

nanmax, maximum, fmax

Notes

-----

NaN values are propagated, that is if at least one item is NaN, the

corresponding min value will be NaN as well. To ignore NaN values

(MATLAB behavior), please use nanmin.

Don't use `amin` for element-wise comparison of 2 arrays; when

``a.shape[0]`` is 2, ``minimum(a[0], a[1])`` is faster than

``amin(a, axis=0)``.

Examples

--------

>>> a = np.arange(4).reshape((2,2))

>>> a

array([[0, 1],

[2, 3]])

>>> np.amin(a) # Minimum of the flattened array

0

>>> np.amin(a, axis=0) # Minima along the first axis

array([0, 1])

>>> np.amin(a, axis=1) # Minima along the second axis

array([0, 2])

>>> np.amin(a, where=[False, True], initial=10, axis=0)

array([10, 1])

>>> b = np.arange(5, dtype=float)

>>> b[2] = np.NaN

>>> np.amin(b)

nan

>>> np.amin(b, where=~np.isnan(b), initial=10)

0.0

>>> np.nanmin(b)

0.0

>>> np.min([[-50], [10]], axis=-1, initial=0)

array([-50, 0])

Notice that the initial value is used as one of the elements for which the

minimum is determined, unlike for the default argument Python's max

function, which is only used for empty iterables.

Notice that this isn't the same as Python's ``default`` argument.

>>> np.min([6], initial=5)

5

>>> min([6], default=5)

6

angle(z, deg=False)

Return the angle of the complex argument.

Parameters

----------

z : array\_like

A complex number or sequence of complex numbers.

deg : bool, optional

Return angle in degrees if True, radians if False (default).

Returns

-------

angle : ndarray or scalar

The counterclockwise angle from the positive real axis on the complex

plane in the range ``(-pi, pi]``, with dtype as numpy.float64.

..versionchanged:: 1.16.0

This function works on subclasses of ndarray like `ma.array`.

See Also

--------

arctan2

absolute

Examples

--------

>>> np.angle([1.0, 1.0j, 1+1j]) # in radians

array([ 0. , 1.57079633, 0.78539816]) # may vary

>>> np.angle(1+1j, deg=True) # in degrees

45.0

any(a, axis=None, out=None, keepdims=<no value>)

Test whether any array element along a given axis evaluates to True.

Returns single boolean unless `axis` is not ``None``

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

axis : None or int or tuple of ints, optional

Axis or axes along which a logical OR reduction is performed.

The default (`axis` = `None`) is to perform a logical OR over all

the dimensions of the input array. `axis` may be negative, in

which case it counts from the last to the first axis.

.. versionadded:: 1.7.0

If this is a tuple of ints, a reduction is performed on multiple

axes, instead of a single axis or all the axes as before.

out : ndarray, optional

Alternate output array in which to place the result. It must have

the same shape as the expected output and its type is preserved

(e.g., if it is of type float, then it will remain so, returning

1.0 for True and 0.0 for False, regardless of the type of `a`).

See `doc.ufuncs` (Section "Output arguments") for details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `any` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

any : bool or ndarray

A new boolean or `ndarray` is returned unless `out` is specified,

in which case a reference to `out` is returned.

See Also

--------

ndarray.any : equivalent method

all : Test whether all elements along a given axis evaluate to True.

Notes

-----

Not a Number (NaN), positive infinity and negative infinity evaluate

to `True` because these are not equal to zero.

Examples

--------

>>> np.any([[True, False], [True, True]])

True

>>> np.any([[True, False], [False, False]], axis=0)

array([ True, False])

>>> np.any([-1, 0, 5])

True

>>> np.any(np.nan)

True

>>> o=np.array(False)

>>> z=np.any([-1, 4, 5], out=o)

>>> z, o

(array(True), array(True))

>>> # Check now that z is a reference to o

>>> z is o

True

>>> id(z), id(o) # identity of z and o # doctest: +SKIP

(191614240, 191614240)

append(arr, values, axis=None)

Append values to the end of an array.

Parameters

----------

arr : array\_like

Values are appended to a copy of this array.

values : array\_like

These values are appended to a copy of `arr`. It must be of the

correct shape (the same shape as `arr`, excluding `axis`). If

`axis` is not specified, `values` can be any shape and will be

flattened before use.

axis : int, optional

The axis along which `values` are appended. If `axis` is not

given, both `arr` and `values` are flattened before use.

Returns

-------

append : ndarray

A copy of `arr` with `values` appended to `axis`. Note that

`append` does not occur in-place: a new array is allocated and

filled. If `axis` is None, `out` is a flattened array.

See Also

--------

insert : Insert elements into an array.

delete : Delete elements from an array.

Examples

--------

>>> np.append([1, 2, 3], [[4, 5, 6], [7, 8, 9]])

array([1, 2, 3, ..., 7, 8, 9])

When `axis` is specified, `values` must have the correct shape.

>>> np.append([[1, 2, 3], [4, 5, 6]], [[7, 8, 9]], axis=0)

array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

>>> np.append([[1, 2, 3], [4, 5, 6]], [7, 8, 9], axis=0)

Traceback (most recent call last):

...

ValueError: all the input arrays must have same number of dimensions

apply\_along\_axis(func1d, axis, arr, \*args, \*\*kwargs)

Apply a function to 1-D slices along the given axis.

Execute `func1d(a, \*args)` where `func1d` operates on 1-D arrays and `a`

is a 1-D slice of `arr` along `axis`.

This is equivalent to (but faster than) the following use of `ndindex` and

`s\_`, which sets each of ``ii``, ``jj``, and ``kk`` to a tuple of indices::

Ni, Nk = a.shape[:axis], a.shape[axis+1:]

for ii in ndindex(Ni):

for kk in ndindex(Nk):

f = func1d(arr[ii + s\_[:,] + kk])

Nj = f.shape

for jj in ndindex(Nj):

out[ii + jj + kk] = f[jj]

Equivalently, eliminating the inner loop, this can be expressed as::

Ni, Nk = a.shape[:axis], a.shape[axis+1:]

for ii in ndindex(Ni):

for kk in ndindex(Nk):

out[ii + s\_[...,] + kk] = func1d(arr[ii + s\_[:,] + kk])

Parameters

----------

func1d : function (M,) -> (Nj...)

This function should accept 1-D arrays. It is applied to 1-D

slices of `arr` along the specified axis.

axis : integer

Axis along which `arr` is sliced.

arr : ndarray (Ni..., M, Nk...)

Input array.

args : any

Additional arguments to `func1d`.

kwargs : any

Additional named arguments to `func1d`.

.. versionadded:: 1.9.0

Returns

-------

out : ndarray (Ni..., Nj..., Nk...)

The output array. The shape of `out` is identical to the shape of

`arr`, except along the `axis` dimension. This axis is removed, and

replaced with new dimensions equal to the shape of the return value

of `func1d`. So if `func1d` returns a scalar `out` will have one

fewer dimensions than `arr`.

See Also

--------

apply\_over\_axes : Apply a function repeatedly over multiple axes.

Examples

--------

>>> def my\_func(a):

... """Average first and last element of a 1-D array"""

... return (a[0] + a[-1]) \* 0.5

>>> b = np.array([[1,2,3], [4,5,6], [7,8,9]])

>>> np.apply\_along\_axis(my\_func, 0, b)

array([4., 5., 6.])

>>> np.apply\_along\_axis(my\_func, 1, b)

array([2., 5., 8.])

For a function that returns a 1D array, the number of dimensions in

`outarr` is the same as `arr`.

>>> b = np.array([[8,1,7], [4,3,9], [5,2,6]])

>>> np.apply\_along\_axis(sorted, 1, b)

array([[1, 7, 8],

[3, 4, 9],

[2, 5, 6]])

For a function that returns a higher dimensional array, those dimensions

are inserted in place of the `axis` dimension.

>>> b = np.array([[1,2,3], [4,5,6], [7,8,9]])

>>> np.apply\_along\_axis(np.diag, -1, b)

array([[[1, 0, 0],

[0, 2, 0],

[0, 0, 3]],

[[4, 0, 0],

[0, 5, 0],

[0, 0, 6]],

[[7, 0, 0],

[0, 8, 0],

[0, 0, 9]]])

apply\_over\_axes(func, a, axes)

Apply a function repeatedly over multiple axes.

`func` is called as `res = func(a, axis)`, where `axis` is the first

element of `axes`. The result `res` of the function call must have

either the same dimensions as `a` or one less dimension. If `res`

has one less dimension than `a`, a dimension is inserted before

`axis`. The call to `func` is then repeated for each axis in `axes`,

with `res` as the first argument.

Parameters

----------

func : function

This function must take two arguments, `func(a, axis)`.

a : array\_like

Input array.

axes : array\_like

Axes over which `func` is applied; the elements must be integers.

Returns

-------

apply\_over\_axis : ndarray

The output array. The number of dimensions is the same as `a`,

but the shape can be different. This depends on whether `func`

changes the shape of its output with respect to its input.

See Also

--------

apply\_along\_axis :

Apply a function to 1-D slices of an array along the given axis.

Notes

------

This function is equivalent to tuple axis arguments to reorderable ufuncs

with keepdims=True. Tuple axis arguments to ufuncs have been available since

version 1.7.0.

Examples

--------

>>> a = np.arange(24).reshape(2,3,4)

>>> a

array([[[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]],

[[12, 13, 14, 15],

[16, 17, 18, 19],

[20, 21, 22, 23]]])

Sum over axes 0 and 2. The result has same number of dimensions

as the original array:

>>> np.apply\_over\_axes(np.sum, a, [0,2])

array([[[ 60],

[ 92],

[124]]])

Tuple axis arguments to ufuncs are equivalent:

>>> np.sum(a, axis=(0,2), keepdims=True)

array([[[ 60],

[ 92],

[124]]])

arange(...)

arange([start,] stop[, step,], dtype=None)

Return evenly spaced values within a given interval.

Values are generated within the half-open interval ``[start, stop)``

(in other words, the interval including `start` but excluding `stop`).

For integer arguments the function is equivalent to the Python built-in

`range` function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not

be consistent. It is better to use `numpy.linspace` for these cases.

Parameters

----------

start : number, optional

Start of interval. The interval includes this value. The default

start value is 0.

stop : number

End of interval. The interval does not include this value, except

in some cases where `step` is not an integer and floating point

round-off affects the length of `out`.

step : number, optional

Spacing between values. For any output `out`, this is the distance

between two adjacent values, ``out[i+1] - out[i]``. The default

step size is 1. If `step` is specified as a position argument,

`start` must also be given.

dtype : dtype

The type of the output array. If `dtype` is not given, infer the data

type from the other input arguments.

Returns

-------

arange : ndarray

Array of evenly spaced values.

For floating point arguments, the length of the result is

``ceil((stop - start)/step)``. Because of floating point overflow,

this rule may result in the last element of `out` being greater

than `stop`.

See Also

--------

linspace : Evenly spaced numbers with careful handling of endpoints.

ogrid: Arrays of evenly spaced numbers in N-dimensions.

mgrid: Grid-shaped arrays of evenly spaced numbers in N-dimensions.

Examples

--------

>>> np.arange(3)

array([0, 1, 2])

>>> np.arange(3.0)

array([ 0., 1., 2.])

>>> np.arange(3,7)

array([3, 4, 5, 6])

>>> np.arange(3,7,2)

array([3, 5])

argmax(a, axis=None, out=None)

Returns the indices of the maximum values along an axis.

Parameters

----------

a : array\_like

Input array.

axis : int, optional

By default, the index is into the flattened array, otherwise

along the specified axis.

out : array, optional

If provided, the result will be inserted into this array. It should

be of the appropriate shape and dtype.

Returns

-------

index\_array : ndarray of ints

Array of indices into the array. It has the same shape as `a.shape`

with the dimension along `axis` removed.

See Also

--------

ndarray.argmax, argmin

amax : The maximum value along a given axis.

unravel\_index : Convert a flat index into an index tuple.

Notes

-----

In case of multiple occurrences of the maximum values, the indices

corresponding to the first occurrence are returned.

Examples

--------

>>> a = np.arange(6).reshape(2,3) + 10

>>> a

array([[10, 11, 12],

[13, 14, 15]])

>>> np.argmax(a)

5

>>> np.argmax(a, axis=0)

array([1, 1, 1])

>>> np.argmax(a, axis=1)

array([2, 2])

Indexes of the maximal elements of a N-dimensional array:

>>> ind = np.unravel\_index(np.argmax(a, axis=None), a.shape)

>>> ind

(1, 2)

>>> a[ind]

15

>>> b = np.arange(6)

>>> b[1] = 5

>>> b

array([0, 5, 2, 3, 4, 5])

>>> np.argmax(b) # Only the first occurrence is returned.

1

argmin(a, axis=None, out=None)

Returns the indices of the minimum values along an axis.

Parameters

----------

a : array\_like

Input array.

axis : int, optional

By default, the index is into the flattened array, otherwise

along the specified axis.

out : array, optional

If provided, the result will be inserted into this array. It should

be of the appropriate shape and dtype.

Returns

-------

index\_array : ndarray of ints

Array of indices into the array. It has the same shape as `a.shape`

with the dimension along `axis` removed.

See Also

--------

ndarray.argmin, argmax

amin : The minimum value along a given axis.

unravel\_index : Convert a flat index into an index tuple.

Notes

-----

In case of multiple occurrences of the minimum values, the indices

corresponding to the first occurrence are returned.

Examples

--------

>>> a = np.arange(6).reshape(2,3) + 10

>>> a

array([[10, 11, 12],

[13, 14, 15]])

>>> np.argmin(a)

0

>>> np.argmin(a, axis=0)

array([0, 0, 0])

>>> np.argmin(a, axis=1)

array([0, 0])

Indices of the minimum elements of a N-dimensional array:

>>> ind = np.unravel\_index(np.argmin(a, axis=None), a.shape)

>>> ind

(0, 0)

>>> a[ind]

10

>>> b = np.arange(6) + 10

>>> b[4] = 10

>>> b

array([10, 11, 12, 13, 10, 15])

>>> np.argmin(b) # Only the first occurrence is returned.

0

argpartition(a, kth, axis=-1, kind='introselect', order=None)

Perform an indirect partition along the given axis using the

algorithm specified by the `kind` keyword. It returns an array of

indices of the same shape as `a` that index data along the given

axis in partitioned order.

.. versionadded:: 1.8.0

Parameters

----------

a : array\_like

Array to sort.

kth : int or sequence of ints

Element index to partition by. The k-th element will be in its

final sorted position and all smaller elements will be moved

before it and all larger elements behind it. The order all

elements in the partitions is undefined. If provided with a

sequence of k-th it will partition all of them into their sorted

position at once.

axis : int or None, optional

Axis along which to sort. The default is -1 (the last axis). If

None, the flattened array is used.

kind : {'introselect'}, optional

Selection algorithm. Default is 'introselect'

order : str or list of str, optional

When `a` is an array with fields defined, this argument

specifies which fields to compare first, second, etc. A single

field can be specified as a string, and not all fields need be

specified, but unspecified fields will still be used, in the

order in which they come up in the dtype, to break ties.

Returns

-------

index\_array : ndarray, int

Array of indices that partition `a` along the specified axis.

If `a` is one-dimensional, ``a[index\_array]`` yields a partitioned `a`.

More generally, ``np.take\_along\_axis(a, index\_array, axis=a)`` always

yields the partitioned `a`, irrespective of dimensionality.

See Also

--------

partition : Describes partition algorithms used.

ndarray.partition : Inplace partition.

argsort : Full indirect sort

Notes

-----

See `partition` for notes on the different selection algorithms.

Examples

--------

One dimensional array:

>>> x = np.array([3, 4, 2, 1])

>>> x[np.argpartition(x, 3)]

array([2, 1, 3, 4])

>>> x[np.argpartition(x, (1, 3))]

array([1, 2, 3, 4])

>>> x = [3, 4, 2, 1]

>>> np.array(x)[np.argpartition(x, 3)]

array([2, 1, 3, 4])

argsort(a, axis=-1, kind=None, order=None)

Returns the indices that would sort an array.

Perform an indirect sort along the given axis using the algorithm specified

by the `kind` keyword. It returns an array of indices of the same shape as

`a` that index data along the given axis in sorted order.

Parameters

----------

a : array\_like

Array to sort.

axis : int or None, optional

Axis along which to sort. The default is -1 (the last axis). If None,

the flattened array is used.

kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

and 'mergesort' use timsort under the covers and, in general, the

actual implementation will vary with data type. The 'mergesort' option

is retained for backwards compatibility.

.. versionchanged:: 1.15.0.

The 'stable' option was added.

order : str or list of str, optional

When `a` is an array with fields defined, this argument specifies

which fields to compare first, second, etc. A single field can

be specified as a string, and not all fields need be specified,

but unspecified fields will still be used, in the order in which

they come up in the dtype, to break ties.

Returns

-------

index\_array : ndarray, int

Array of indices that sort `a` along the specified `axis`.

If `a` is one-dimensional, ``a[index\_array]`` yields a sorted `a`.

More generally, ``np.take\_along\_axis(a, index\_array, axis=axis)``

always yields the sorted `a`, irrespective of dimensionality.

See Also

--------

sort : Describes sorting algorithms used.

lexsort : Indirect stable sort with multiple keys.

ndarray.sort : Inplace sort.

argpartition : Indirect partial sort.

Notes

-----

See `sort` for notes on the different sorting algorithms.

As of NumPy 1.4.0 `argsort` works with real/complex arrays containing

nan values. The enhanced sort order is documented in `sort`.

Examples

--------

One dimensional array:

>>> x = np.array([3, 1, 2])

>>> np.argsort(x)

array([1, 2, 0])

Two-dimensional array:

>>> x = np.array([[0, 3], [2, 2]])

>>> x

array([[0, 3],

[2, 2]])

>>> ind = np.argsort(x, axis=0) # sorts along first axis (down)

>>> ind

array([[0, 1],

[1, 0]])

>>> np.take\_along\_axis(x, ind, axis=0) # same as np.sort(x, axis=0)

array([[0, 2],

[2, 3]])

>>> ind = np.argsort(x, axis=1) # sorts along last axis (across)

>>> ind

array([[0, 1],

[0, 1]])

>>> np.take\_along\_axis(x, ind, axis=1) # same as np.sort(x, axis=1)

array([[0, 3],

[2, 2]])

Indices of the sorted elements of a N-dimensional array:

>>> ind = np.unravel\_index(np.argsort(x, axis=None), x.shape)

>>> ind

(array([0, 1, 1, 0]), array([0, 0, 1, 1]))

>>> x[ind] # same as np.sort(x, axis=None)

array([0, 2, 2, 3])

Sorting with keys:

>>> x = np.array([(1, 0), (0, 1)], dtype=[('x', '<i4'), ('y', '<i4')])

>>> x

array([(1, 0), (0, 1)],

dtype=[('x', '<i4'), ('y', '<i4')])

>>> np.argsort(x, order=('x','y'))

array([1, 0])

>>> np.argsort(x, order=('y','x'))

array([0, 1])

argwhere(a)

Find the indices of array elements that are non-zero, grouped by element.

Parameters

----------

a : array\_like

Input data.

Returns

-------

index\_array : ndarray

Indices of elements that are non-zero. Indices are grouped by element.

See Also

--------

where, nonzero

Notes

-----

``np.argwhere(a)`` is the same as ``np.transpose(np.nonzero(a))``.

The output of ``argwhere`` is not suitable for indexing arrays.

For this purpose use ``nonzero(a)`` instead.

Examples

--------

>>> x = np.arange(6).reshape(2,3)

>>> x

array([[0, 1, 2],

[3, 4, 5]])

>>> np.argwhere(x>1)

array([[0, 2],

[1, 0],

[1, 1],

[1, 2]])

around(a, decimals=0, out=None)

Evenly round to the given number of decimals.

Parameters

----------

a : array\_like

Input data.

decimals : int, optional

Number of decimal places to round to (default: 0). If

decimals is negative, it specifies the number of positions to

the left of the decimal point.

out : ndarray, optional

Alternative output array in which to place the result. It must have

the same shape as the expected output, but the type of the output

values will be cast if necessary. See `doc.ufuncs` (Section

"Output arguments") for details.

Returns

-------

rounded\_array : ndarray

An array of the same type as `a`, containing the rounded values.

Unless `out` was specified, a new array is created. A reference to

the result is returned.

The real and imaginary parts of complex numbers are rounded

separately. The result of rounding a float is a float.

See Also

--------

ndarray.round : equivalent method

ceil, fix, floor, rint, trunc

Notes

-----

For values exactly halfway between rounded decimal values, NumPy

rounds to the nearest even value. Thus 1.5 and 2.5 round to 2.0,

-0.5 and 0.5 round to 0.0, etc. Results may also be surprising due

to the inexact representation of decimal fractions in the IEEE

floating point standard [1]\_ and errors introduced when scaling

by powers of ten.

References

----------

.. [1] "Lecture Notes on the Status of IEEE 754", William Kahan,

https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF

.. [2] "How Futile are Mindless Assessments of

Roundoff in Floating-Point Computation?", William Kahan,

https://people.eecs.berkeley.edu/~wkahan/Mindless.pdf

Examples

--------

>>> np.around([0.37, 1.64])

array([0., 2.])

>>> np.around([0.37, 1.64], decimals=1)

array([0.4, 1.6])

>>> np.around([.5, 1.5, 2.5, 3.5, 4.5]) # rounds to nearest even value

array([0., 2., 2., 4., 4.])

>>> np.around([1,2,3,11], decimals=1) # ndarray of ints is returned

array([ 1, 2, 3, 11])

>>> np.around([1,2,3,11], decimals=-1)

array([ 0, 0, 0, 10])

array(...)

array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

Create an array.

Parameters

----------

object : array\_like

An array, any object exposing the array interface, an object whose

\_\_array\_\_ method returns an array, or any (nested) sequence.

dtype : data-type, optional

The desired data-type for the array. If not given, then the type will

be determined as the minimum type required to hold the objects in the

sequence. This argument can only be used to 'upcast' the array. For

downcasting, use the .astype(t) method.

copy : bool, optional

If true (default), then the object is copied. Otherwise, a copy will

only be made if \_\_array\_\_ returns a copy, if obj is a nested sequence,

or if a copy is needed to satisfy any of the other requirements

(`dtype`, `order`, etc.).

order : {'K', 'A', 'C', 'F'}, optional

Specify the memory layout of the array. If object is not an array, the

newly created array will be in C order (row major) unless 'F' is

specified, in which case it will be in Fortran order (column major).

If object is an array the following holds.

===== ========= ===================================================

order no copy copy=True

===== ========= ===================================================

'K' unchanged F & C order preserved, otherwise most similar order

'A' unchanged F order if input is F and not C, otherwise C order

'C' C order C order

'F' F order F order

===== ========= ===================================================

When ``copy=False`` and a copy is made for other reasons, the result is

the same as if ``copy=True``, with some exceptions for `A`, see the

Notes section. The default order is 'K'.

subok : bool, optional

If True, then sub-classes will be passed-through, otherwise

the returned array will be forced to be a base-class array (default).

ndmin : int, optional

Specifies the minimum number of dimensions that the resulting

array should have. Ones will be pre-pended to the shape as

needed to meet this requirement.

Returns

-------

out : ndarray

An array object satisfying the specified requirements.

See Also

--------

empty\_like : Return an empty array with shape and type of input.

ones\_like : Return an array of ones with shape and type of input.

zeros\_like : Return an array of zeros with shape and type of input.

full\_like : Return a new array with shape of input filled with value.

empty : Return a new uninitialized array.

ones : Return a new array setting values to one.

zeros : Return a new array setting values to zero.

full : Return a new array of given shape filled with value.

Notes

-----

When order is 'A' and `object` is an array in neither 'C' nor 'F' order,

and a copy is forced by a change in dtype, then the order of the result is

not necessarily 'C' as expected. This is likely a bug.

Examples

--------

>>> np.array([1, 2, 3])

array([1, 2, 3])

Upcasting:

>>> np.array([1, 2, 3.0])

array([ 1., 2., 3.])

More than one dimension:

>>> np.array([[1, 2], [3, 4]])

array([[1, 2],

[3, 4]])

Minimum dimensions 2:

>>> np.array([1, 2, 3], ndmin=2)

array([[1, 2, 3]])

Type provided:

>>> np.array([1, 2, 3], dtype=complex)

array([ 1.+0.j, 2.+0.j, 3.+0.j])

Data-type consisting of more than one element:

>>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')])

>>> x['a']

array([1, 3])

Creating an array from sub-classes:

>>> np.array(np.mat('1 2; 3 4'))

array([[1, 2],

[3, 4]])

>>> np.array(np.mat('1 2; 3 4'), subok=True)

matrix([[1, 2],

[3, 4]])

array2string(a, max\_line\_width=None, precision=None, suppress\_small=None, separator=' ', prefix='', style=<no value>, formatter=None, threshold=None, edgeitems=None, sign=None, floatmode=None, suffix='', \*\*kwarg)

Return a string representation of an array.

Parameters

----------

a : array\_like

Input array.

max\_line\_width : int, optional

Inserts newlines if text is longer than `max\_line\_width`.

Defaults to ``numpy.get\_printoptions()['linewidth']``.

precision : int or None, optional

Floating point precision.

Defaults to ``numpy.get\_printoptions()['precision']``.

suppress\_small : bool, optional

Represent numbers "very close" to zero as zero; default is False.

Very close is defined by precision: if the precision is 8, e.g.,

numbers smaller (in absolute value) than 5e-9 are represented as

zero.

Defaults to ``numpy.get\_printoptions()['suppress']``.

separator : str, optional

Inserted between elements.

prefix : str, optional

suffix: str, optional

The length of the prefix and suffix strings are used to respectively

align and wrap the output. An array is typically printed as::

prefix + array2string(a) + suffix

The output is left-padded by the length of the prefix string, and

wrapping is forced at the column ``max\_line\_width - len(suffix)``.

It should be noted that the content of prefix and suffix strings are

not included in the output.

style : \_NoValue, optional

Has no effect, do not use.

.. deprecated:: 1.14.0

formatter : dict of callables, optional

If not None, the keys should indicate the type(s) that the respective

formatting function applies to. Callables should return a string.

Types that are not specified (by their corresponding keys) are handled

by the default formatters. Individual types for which a formatter

can be set are:

- 'bool'

- 'int'

- 'timedelta' : a `numpy.timedelta64`

- 'datetime' : a `numpy.datetime64`

- 'float'

- 'longfloat' : 128-bit floats

- 'complexfloat'

- 'longcomplexfloat' : composed of two 128-bit floats

- 'void' : type `numpy.void`

- 'numpystr' : types `numpy.string\_` and `numpy.unicode\_`

- 'str' : all other strings

Other keys that can be used to set a group of types at once are:

- 'all' : sets all types

- 'int\_kind' : sets 'int'

- 'float\_kind' : sets 'float' and 'longfloat'

- 'complex\_kind' : sets 'complexfloat' and 'longcomplexfloat'

- 'str\_kind' : sets 'str' and 'numpystr'

threshold : int, optional

Total number of array elements which trigger summarization

rather than full repr.

Defaults to ``numpy.get\_printoptions()['threshold']``.

edgeitems : int, optional

Number of array items in summary at beginning and end of

each dimension.

Defaults to ``numpy.get\_printoptions()['edgeitems']``.

sign : string, either '-', '+', or ' ', optional

Controls printing of the sign of floating-point types. If '+', always

print the sign of positive values. If ' ', always prints a space

(whitespace character) in the sign position of positive values. If

'-', omit the sign character of positive values.

Defaults to ``numpy.get\_printoptions()['sign']``.

floatmode : str, optional

Controls the interpretation of the `precision` option for

floating-point types.

Defaults to ``numpy.get\_printoptions()['floatmode']``.

Can take the following values:

- 'fixed': Always print exactly `precision` fractional digits,

even if this would print more or fewer digits than

necessary to specify the value uniquely.

- 'unique': Print the minimum number of fractional digits necessary

to represent each value uniquely. Different elements may

have a different number of digits. The value of the

`precision` option is ignored.

- 'maxprec': Print at most `precision` fractional digits, but if

an element can be uniquely represented with fewer digits

only print it with that many.

- 'maxprec\_equal': Print at most `precision` fractional digits,

but if every element in the array can be uniquely

represented with an equal number of fewer digits, use that

many digits for all elements.

legacy : string or `False`, optional

If set to the string `'1.13'` enables 1.13 legacy printing mode. This

approximates numpy 1.13 print output by including a space in the sign

position of floats and different behavior for 0d arrays. If set to

`False`, disables legacy mode. Unrecognized strings will be ignored

with a warning for forward compatibility.

.. versionadded:: 1.14.0

Returns

-------

array\_str : str

String representation of the array.

Raises

------

TypeError

if a callable in `formatter` does not return a string.

See Also

--------

array\_str, array\_repr, set\_printoptions, get\_printoptions

Notes

-----

If a formatter is specified for a certain type, the `precision` keyword is

ignored for that type.

This is a very flexible function; `array\_repr` and `array\_str` are using

`array2string` internally so keywords with the same name should work

identically in all three functions.

Examples

--------

>>> x = np.array([1e-16,1,2,3])

>>> np.array2string(x, precision=2, separator=',',

... suppress\_small=True)

'[0.,1.,2.,3.]'

>>> x = np.arange(3.)

>>> np.array2string(x, formatter={'float\_kind':lambda x: "%.2f" % x})

'[0.00 1.00 2.00]'

>>> x = np.arange(3)

>>> np.array2string(x, formatter={'int':lambda x: hex(x)})

'[0x0 0x1 0x2]'

array\_equal(a1, a2)

True if two arrays have the same shape and elements, False otherwise.

Parameters

----------

a1, a2 : array\_like

Input arrays.

Returns

-------

b : bool

Returns True if the arrays are equal.

See Also

--------

allclose: Returns True if two arrays are element-wise equal within a

tolerance.

array\_equiv: Returns True if input arrays are shape consistent and all

elements equal.

Examples

--------

>>> np.array\_equal([1, 2], [1, 2])

True

>>> np.array\_equal(np.array([1, 2]), np.array([1, 2]))

True

>>> np.array\_equal([1, 2], [1, 2, 3])

False

>>> np.array\_equal([1, 2], [1, 4])

False

array\_equiv(a1, a2)

Returns True if input arrays are shape consistent and all elements equal.

Shape consistent means they are either the same shape, or one input array

can be broadcasted to create the same shape as the other one.

Parameters

----------

a1, a2 : array\_like

Input arrays.

Returns

-------

out : bool

True if equivalent, False otherwise.

Examples

--------

>>> np.array\_equiv([1, 2], [1, 2])

True

>>> np.array\_equiv([1, 2], [1, 3])

False

Showing the shape equivalence:

>>> np.array\_equiv([1, 2], [[1, 2], [1, 2]])

True

>>> np.array\_equiv([1, 2], [[1, 2, 1, 2], [1, 2, 1, 2]])

False

>>> np.array\_equiv([1, 2], [[1, 2], [1, 3]])

False

array\_repr(arr, max\_line\_width=None, precision=None, suppress\_small=None)

Return the string representation of an array.

Parameters

----------

arr : ndarray

Input array.

max\_line\_width : int, optional

Inserts newlines if text is longer than `max\_line\_width`.

Defaults to ``numpy.get\_printoptions()['linewidth']``.

precision : int, optional

Floating point precision.

Defaults to ``numpy.get\_printoptions()['precision']``.

suppress\_small : bool, optional

Represent numbers "very close" to zero as zero; default is False.

Very close is defined by precision: if the precision is 8, e.g.,

numbers smaller (in absolute value) than 5e-9 are represented as

zero.

Defaults to ``numpy.get\_printoptions()['suppress']``.

Returns

-------

string : str

The string representation of an array.

See Also

--------

array\_str, array2string, set\_printoptions

Examples

--------

>>> np.array\_repr(np.array([1,2]))

'array([1, 2])'

>>> np.array\_repr(np.ma.array([0.]))

'MaskedArray([0.])'

>>> np.array\_repr(np.array([], np.int32))

'array([], dtype=int32)'

>>> x = np.array([1e-6, 4e-7, 2, 3])

>>> np.array\_repr(x, precision=6, suppress\_small=True)

'array([0.000001, 0. , 2. , 3. ])'

array\_split(ary, indices\_or\_sections, axis=0)

Split an array into multiple sub-arrays.

Please refer to the ``split`` documentation. The only difference

between these functions is that ``array\_split`` allows

`indices\_or\_sections` to be an integer that does \*not\* equally

divide the axis. For an array of length l that should be split

into n sections, it returns l % n sub-arrays of size l//n + 1

and the rest of size l//n.

See Also

--------

split : Split array into multiple sub-arrays of equal size.

Examples

--------

>>> x = np.arange(8.0)

>>> np.array\_split(x, 3)

[array([0., 1., 2.]), array([3., 4., 5.]), array([6., 7.])]

>>> x = np.arange(7.0)

>>> np.array\_split(x, 3)

[array([0., 1., 2.]), array([3., 4.]), array([5., 6.])]

array\_str(a, max\_line\_width=None, precision=None, suppress\_small=None)

Return a string representation of the data in an array.

The data in the array is returned as a single string. This function is

similar to `array\_repr`, the difference being that `array\_repr` also

returns information on the kind of array and its data type.

Parameters

----------

a : ndarray

Input array.

max\_line\_width : int, optional

Inserts newlines if text is longer than `max\_line\_width`.

Defaults to ``numpy.get\_printoptions()['linewidth']``.

precision : int, optional

Floating point precision.

Defaults to ``numpy.get\_printoptions()['precision']``.

suppress\_small : bool, optional

Represent numbers "very close" to zero as zero; default is False.

Very close is defined by precision: if the precision is 8, e.g.,

numbers smaller (in absolute value) than 5e-9 are represented as

zero.

Defaults to ``numpy.get\_printoptions()['suppress']``.

See Also

--------

array2string, array\_repr, set\_printoptions

Examples

--------

>>> np.array\_str(np.arange(3))

'[0 1 2]'

asanyarray(a, dtype=None, order=None)

Convert the input to an ndarray, but pass ndarray subclasses through.

Parameters

----------

a : array\_like

Input data, in any form that can be converted to an array. This

includes scalars, lists, lists of tuples, tuples, tuples of tuples,

tuples of lists, and ndarrays.

dtype : data-type, optional

By default, the data-type is inferred from the input data.

order : {'C', 'F'}, optional

Whether to use row-major (C-style) or column-major

(Fortran-style) memory representation. Defaults to 'C'.

Returns

-------

out : ndarray or an ndarray subclass

Array interpretation of `a`. If `a` is an ndarray or a subclass

of ndarray, it is returned as-is and no copy is performed.

See Also

--------

asarray : Similar function which always returns ndarrays.

ascontiguousarray : Convert input to a contiguous array.

asfarray : Convert input to a floating point ndarray.

asfortranarray : Convert input to an ndarray with column-major

memory order.

asarray\_chkfinite : Similar function which checks input for NaNs and

Infs.

fromiter : Create an array from an iterator.

fromfunction : Construct an array by executing a function on grid

positions.

Examples

--------

Convert a list into an array:

>>> a = [1, 2]

>>> np.asanyarray(a)

array([1, 2])

Instances of `ndarray` subclasses are passed through as-is:

>>> a = np.array([(1.0, 2), (3.0, 4)], dtype='f4,i4').view(np.recarray)

>>> np.asanyarray(a) is a

True

asarray(a, dtype=None, order=None)

Convert the input to an array.

Parameters

----------

a : array\_like

Input data, in any form that can be converted to an array. This

includes lists, lists of tuples, tuples, tuples of tuples, tuples

of lists and ndarrays.

dtype : data-type, optional

By default, the data-type is inferred from the input data.

order : {'C', 'F'}, optional

Whether to use row-major (C-style) or

column-major (Fortran-style) memory representation.

Defaults to 'C'.

Returns

-------

out : ndarray

Array interpretation of `a`. No copy is performed if the input

is already an ndarray with matching dtype and order. If `a` is a

subclass of ndarray, a base class ndarray is returned.

See Also

--------

asanyarray : Similar function which passes through subclasses.

ascontiguousarray : Convert input to a contiguous array.

asfarray : Convert input to a floating point ndarray.

asfortranarray : Convert input to an ndarray with column-major

memory order.

asarray\_chkfinite : Similar function which checks input for NaNs and Infs.

fromiter : Create an array from an iterator.

fromfunction : Construct an array by executing a function on grid

positions.

Examples

--------

Convert a list into an array:

>>> a = [1, 2]

>>> np.asarray(a)

array([1, 2])

Existing arrays are not copied:

>>> a = np.array([1, 2])

>>> np.asarray(a) is a

True

If `dtype` is set, array is copied only if dtype does not match:

>>> a = np.array([1, 2], dtype=np.float32)

>>> np.asarray(a, dtype=np.float32) is a

True

>>> np.asarray(a, dtype=np.float64) is a

False

Contrary to `asanyarray`, ndarray subclasses are not passed through:

>>> issubclass(np.recarray, np.ndarray)

True

>>> a = np.array([(1.0, 2), (3.0, 4)], dtype='f4,i4').view(np.recarray)

>>> np.asarray(a) is a

False

>>> np.asanyarray(a) is a

True

asarray\_chkfinite(a, dtype=None, order=None)

Convert the input to an array, checking for NaNs or Infs.

Parameters

----------

a : array\_like

Input data, in any form that can be converted to an array. This

includes lists, lists of tuples, tuples, tuples of tuples, tuples

of lists and ndarrays. Success requires no NaNs or Infs.

dtype : data-type, optional

By default, the data-type is inferred from the input data.

order : {'C', 'F'}, optional

Whether to use row-major (C-style) or

column-major (Fortran-style) memory representation.

Defaults to 'C'.

Returns

-------

out : ndarray

Array interpretation of `a`. No copy is performed if the input

is already an ndarray. If `a` is a subclass of ndarray, a base

class ndarray is returned.

Raises

------

ValueError

Raises ValueError if `a` contains NaN (Not a Number) or Inf (Infinity).

See Also

--------

asarray : Create and array.

asanyarray : Similar function which passes through subclasses.

ascontiguousarray : Convert input to a contiguous array.

asfarray : Convert input to a floating point ndarray.

asfortranarray : Convert input to an ndarray with column-major

memory order.

fromiter : Create an array from an iterator.

fromfunction : Construct an array by executing a function on grid

positions.

Examples

--------

Convert a list into an array. If all elements are finite

``asarray\_chkfinite`` is identical to ``asarray``.

>>> a = [1, 2]

>>> np.asarray\_chkfinite(a, dtype=float)

array([1., 2.])

Raises ValueError if array\_like contains Nans or Infs.

>>> a = [1, 2, np.inf]

>>> try:

... np.asarray\_chkfinite(a)

... except ValueError:

... print('ValueError')

...

ValueError

ascontiguousarray(a, dtype=None)

Return a contiguous array (ndim >= 1) in memory (C order).

Parameters

----------

a : array\_like

Input array.

dtype : str or dtype object, optional

Data-type of returned array.

Returns

-------

out : ndarray

Contiguous array of same shape and content as `a`, with type `dtype`

if specified.

See Also

--------

asfortranarray : Convert input to an ndarray with column-major

memory order.

require : Return an ndarray that satisfies requirements.

ndarray.flags : Information about the memory layout of the array.

Examples

--------

>>> x = np.arange(6).reshape(2,3)

>>> np.ascontiguousarray(x, dtype=np.float32)

array([[0., 1., 2.],

[3., 4., 5.]], dtype=float32)

>>> x.flags['C\_CONTIGUOUS']

True

Note: This function returns an array with at least one-dimension (1-d)

so it will not preserve 0-d arrays.

asfarray(a, dtype=<class 'numpy.float64'>)

Return an array converted to a float type.

Parameters

----------

a : array\_like

The input array.

dtype : str or dtype object, optional

Float type code to coerce input array `a`. If `dtype` is one of the

'int' dtypes, it is replaced with float64.

Returns

-------

out : ndarray

The input `a` as a float ndarray.

Examples

--------

>>> np.asfarray([2, 3])

array([2., 3.])

>>> np.asfarray([2, 3], dtype='float')

array([2., 3.])

>>> np.asfarray([2, 3], dtype='int8')

array([2., 3.])

asfortranarray(a, dtype=None)

Return an array (ndim >= 1) laid out in Fortran order in memory.

Parameters

----------

a : array\_like

Input array.

dtype : str or dtype object, optional

By default, the data-type is inferred from the input data.

Returns

-------

out : ndarray

The input `a` in Fortran, or column-major, order.

See Also

--------

ascontiguousarray : Convert input to a contiguous (C order) array.

asanyarray : Convert input to an ndarray with either row or

column-major memory order.

require : Return an ndarray that satisfies requirements.

ndarray.flags : Information about the memory layout of the array.

Examples

--------

>>> x = np.arange(6).reshape(2,3)

>>> y = np.asfortranarray(x)

>>> x.flags['F\_CONTIGUOUS']

False

>>> y.flags['F\_CONTIGUOUS']

True

Note: This function returns an array with at least one-dimension (1-d)

so it will not preserve 0-d arrays.

asmatrix(data, dtype=None)

Interpret the input as a matrix.

Unlike `matrix`, `asmatrix` does not make a copy if the input is already

a matrix or an ndarray. Equivalent to ``matrix(data, copy=False)``.

Parameters

----------

data : array\_like

Input data.

dtype : data-type

Data-type of the output matrix.

Returns

-------

mat : matrix

`data` interpreted as a matrix.

Examples

--------

>>> x = np.array([[1, 2], [3, 4]])

>>> m = np.asmatrix(x)

>>> x[0,0] = 5

>>> m

matrix([[5, 2],

[3, 4]])

asscalar(a)

Convert an array of size 1 to its scalar equivalent.

.. deprecated:: 1.16

Deprecated, use `numpy.ndarray.item()` instead.

Parameters

----------

a : ndarray

Input array of size 1.

Returns

-------

out : scalar

Scalar representation of `a`. The output data type is the same type

returned by the input's `item` method.

Examples

--------

>>> np.asscalar(np.array([24]))

24

atleast\_1d(\*arys)

Convert inputs to arrays with at least one dimension.

Scalar inputs are converted to 1-dimensional arrays, whilst

higher-dimensional inputs are preserved.

Parameters

----------

arys1, arys2, ... : array\_like

One or more input arrays.

Returns

-------

ret : ndarray

An array, or list of arrays, each with ``a.ndim >= 1``.

Copies are made only if necessary.

See Also

--------

atleast\_2d, atleast\_3d

Examples

--------

>>> np.atleast\_1d(1.0)

array([1.])

>>> x = np.arange(9.0).reshape(3,3)

>>> np.atleast\_1d(x)

array([[0., 1., 2.],

[3., 4., 5.],

[6., 7., 8.]])

>>> np.atleast\_1d(x) is x

True

>>> np.atleast\_1d(1, [3, 4])

[array([1]), array([3, 4])]

atleast\_2d(\*arys)

View inputs as arrays with at least two dimensions.

Parameters

----------

arys1, arys2, ... : array\_like

One or more array-like sequences. Non-array inputs are converted

to arrays. Arrays that already have two or more dimensions are

preserved.

Returns

-------

res, res2, ... : ndarray

An array, or list of arrays, each with ``a.ndim >= 2``.

Copies are avoided where possible, and views with two or more

dimensions are returned.

See Also

--------

atleast\_1d, atleast\_3d

Examples

--------

>>> np.atleast\_2d(3.0)

array([[3.]])

>>> x = np.arange(3.0)

>>> np.atleast\_2d(x)

array([[0., 1., 2.]])

>>> np.atleast\_2d(x).base is x

True

>>> np.atleast\_2d(1, [1, 2], [[1, 2]])

[array([[1]]), array([[1, 2]]), array([[1, 2]])]

atleast\_3d(\*arys)

View inputs as arrays with at least three dimensions.

Parameters

----------

arys1, arys2, ... : array\_like

One or more array-like sequences. Non-array inputs are converted to

arrays. Arrays that already have three or more dimensions are

preserved.

Returns

-------

res1, res2, ... : ndarray

An array, or list of arrays, each with ``a.ndim >= 3``. Copies are

avoided where possible, and views with three or more dimensions are

returned. For example, a 1-D array of shape ``(N,)`` becomes a view

of shape ``(1, N, 1)``, and a 2-D array of shape ``(M, N)`` becomes a

view of shape ``(M, N, 1)``.

See Also

--------

atleast\_1d, atleast\_2d

Examples

--------

>>> np.atleast\_3d(3.0)

array([[[3.]]])

>>> x = np.arange(3.0)

>>> np.atleast\_3d(x).shape

(1, 3, 1)

>>> x = np.arange(12.0).reshape(4,3)

>>> np.atleast\_3d(x).shape

(4, 3, 1)

>>> np.atleast\_3d(x).base is x.base # x is a reshape, so not base itself

True

>>> for arr in np.atleast\_3d([1, 2], [[1, 2]], [[[1, 2]]]):

... print(arr, arr.shape) # doctest: +SKIP

...

[[[1]

[2]]] (1, 2, 1)

[[[1]

[2]]] (1, 2, 1)

[[[1 2]]] (1, 1, 2)

average(a, axis=None, weights=None, returned=False)

Compute the weighted average along the specified axis.

Parameters

----------

a : array\_like

Array containing data to be averaged. If `a` is not an array, a

conversion is attempted.

axis : None or int or tuple of ints, optional

Axis or axes along which to average `a`. The default,

axis=None, will average over all of the elements of the input array.

If axis is negative it counts from the last to the first axis.

.. versionadded:: 1.7.0

If axis is a tuple of ints, averaging is performed on all of the axes

specified in the tuple instead of a single axis or all the axes as

before.

weights : array\_like, optional

An array of weights associated with the values in `a`. Each value in

`a` contributes to the average according to its associated weight.

The weights array can either be 1-D (in which case its length must be

the size of `a` along the given axis) or of the same shape as `a`.

If `weights=None`, then all data in `a` are assumed to have a

weight equal to one.

returned : bool, optional

Default is `False`. If `True`, the tuple (`average`, `sum\_of\_weights`)

is returned, otherwise only the average is returned.

If `weights=None`, `sum\_of\_weights` is equivalent to the number of

elements over which the average is taken.

Returns

-------

retval, [sum\_of\_weights] : array\_type or double

Return the average along the specified axis. When `returned` is `True`,

return a tuple with the average as the first element and the sum

of the weights as the second element. `sum\_of\_weights` is of the

same type as `retval`. The result dtype follows a genereal pattern.

If `weights` is None, the result dtype will be that of `a` , or ``float64``

if `a` is integral. Otherwise, if `weights` is not None and `a` is non-

integral, the result type will be the type of lowest precision capable of

representing values of both `a` and `weights`. If `a` happens to be

integral, the previous rules still applies but the result dtype will

at least be ``float64``.

Raises

------

ZeroDivisionError

When all weights along axis are zero. See `numpy.ma.average` for a

version robust to this type of error.

TypeError

When the length of 1D `weights` is not the same as the shape of `a`

along axis.

See Also

--------

mean

ma.average : average for masked arrays -- useful if your data contains

"missing" values

numpy.result\_type : Returns the type that results from applying the

numpy type promotion rules to the arguments.

Examples

--------

>>> data = list(range(1,5))

>>> data

[1, 2, 3, 4]

>>> np.average(data)

2.5

>>> np.average(range(1,11), weights=range(10,0,-1))

4.0

>>> data = np.arange(6).reshape((3,2))

>>> data

array([[0, 1],

[2, 3],

[4, 5]])

>>> np.average(data, axis=1, weights=[1./4, 3./4])

array([0.75, 2.75, 4.75])

>>> np.average(data, weights=[1./4, 3./4])

Traceback (most recent call last):

...

TypeError: Axis must be specified when shapes of a and weights differ.

>>> a = np.ones(5, dtype=np.float128)

>>> w = np.ones(5, dtype=np.complex64)

>>> avg = np.average(a, weights=w)

>>> print(avg.dtype)

complex256

bartlett(M)

Return the Bartlett window.

The Bartlett window is very similar to a triangular window, except

that the end points are at zero. It is often used in signal

processing for tapering a signal, without generating too much

ripple in the frequency domain.

Parameters

----------

M : int

Number of points in the output window. If zero or less, an

empty array is returned.

Returns

-------

out : array

The triangular window, with the maximum value normalized to one

(the value one appears only if the number of samples is odd), with

the first and last samples equal to zero.

See Also

--------

blackman, hamming, hanning, kaiser

Notes

-----

The Bartlett window is defined as

.. math:: w(n) = \frac{2}{M-1} \left(

\frac{M-1}{2} - \left|n - \frac{M-1}{2}\right|

\right)

Most references to the Bartlett window come from the signal

processing literature, where it is used as one of many windowing

functions for smoothing values. Note that convolution with this

window produces linear interpolation. It is also known as an

apodization (which means"removing the foot", i.e. smoothing

discontinuities at the beginning and end of the sampled signal) or

tapering function. The fourier transform of the Bartlett is the product

of two sinc functions.

Note the excellent discussion in Kanasewich.

References

----------

.. [1] M.S. Bartlett, "Periodogram Analysis and Continuous Spectra",

Biometrika 37, 1-16, 1950.

.. [2] E.R. Kanasewich, "Time Sequence Analysis in Geophysics",

The University of Alberta Press, 1975, pp. 109-110.

.. [3] A.V. Oppenheim and R.W. Schafer, "Discrete-Time Signal

Processing", Prentice-Hall, 1999, pp. 468-471.

.. [4] Wikipedia, "Window function",

https://en.wikipedia.org/wiki/Window\_function

.. [5] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling,

"Numerical Recipes", Cambridge University Press, 1986, page 429.

Examples

--------

>>> import matplotlib.pyplot as plt

>>> np.bartlett(12)

array([ 0. , 0.18181818, 0.36363636, 0.54545455, 0.72727273, # may vary

0.90909091, 0.90909091, 0.72727273, 0.54545455, 0.36363636,

0.18181818, 0. ])

Plot the window and its frequency response (requires SciPy and matplotlib):

>>> from numpy.fft import fft, fftshift

>>> window = np.bartlett(51)

>>> plt.plot(window)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Bartlett window")

Text(0.5, 1.0, 'Bartlett window')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("Sample")

Text(0.5, 0, 'Sample')

>>> plt.show()

>>> plt.figure()

<Figure size 640x480 with 0 Axes>

>>> A = fft(window, 2048) / 25.5

>>> mag = np.abs(fftshift(A))

>>> freq = np.linspace(-0.5, 0.5, len(A))

>>> with np.errstate(divide='ignore', invalid='ignore'):

... response = 20 \* np.log10(mag)

...

>>> response = np.clip(response, -100, 100)

>>> plt.plot(freq, response)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Frequency response of Bartlett window")

Text(0.5, 1.0, 'Frequency response of Bartlett window')

>>> plt.ylabel("Magnitude [dB]")

Text(0, 0.5, 'Magnitude [dB]')

>>> plt.xlabel("Normalized frequency [cycles per sample]")

Text(0.5, 0, 'Normalized frequency [cycles per sample]')

>>> \_ = plt.axis('tight')

>>> plt.show()

base\_repr(number, base=2, padding=0)

Return a string representation of a number in the given base system.

Parameters

----------

number : int

The value to convert. Positive and negative values are handled.

base : int, optional

Convert `number` to the `base` number system. The valid range is 2-36,

the default value is 2.

padding : int, optional

Number of zeros padded on the left. Default is 0 (no padding).

Returns

-------

out : str

String representation of `number` in `base` system.

See Also

--------

binary\_repr : Faster version of `base\_repr` for base 2.

Examples

--------

>>> np.base\_repr(5)

'101'

>>> np.base\_repr(6, 5)

'11'

>>> np.base\_repr(7, base=5, padding=3)

'00012'

>>> np.base\_repr(10, base=16)

'A'

>>> np.base\_repr(32, base=16)

'20'

binary\_repr(num, width=None)

Return the binary representation of the input number as a string.

For negative numbers, if width is not given, a minus sign is added to the

front. If width is given, the two's complement of the number is

returned, with respect to that width.

In a two's-complement system negative numbers are represented by the two's

complement of the absolute value. This is the most common method of

representing signed integers on computers [1]\_. A N-bit two's-complement

system can represent every integer in the range

:math:`-2^{N-1}` to :math:`+2^{N-1}-1`.

Parameters

----------

num : int

Only an integer decimal number can be used.

width : int, optional

The length of the returned string if `num` is positive, or the length

of the two's complement if `num` is negative, provided that `width` is

at least a sufficient number of bits for `num` to be represented in the

designated form.

If the `width` value is insufficient, it will be ignored, and `num` will

be returned in binary (`num` > 0) or two's complement (`num` < 0) form

with its width equal to the minimum number of bits needed to represent

the number in the designated form. This behavior is deprecated and will

later raise an error.

.. deprecated:: 1.12.0

Returns

-------

bin : str

Binary representation of `num` or two's complement of `num`.

See Also

--------

base\_repr: Return a string representation of a number in the given base

system.

bin: Python's built-in binary representation generator of an integer.

Notes

-----

`binary\_repr` is equivalent to using `base\_repr` with base 2, but about 25x

faster.

References

----------

.. [1] Wikipedia, "Two's complement",

https://en.wikipedia.org/wiki/Two's\_complement

Examples

--------

>>> np.binary\_repr(3)

'11'

>>> np.binary\_repr(-3)

'-11'

>>> np.binary\_repr(3, width=4)

'0011'

The two's complement is returned when the input number is negative and

width is specified:

>>> np.binary\_repr(-3, width=3)

'101'

>>> np.binary\_repr(-3, width=5)

'11101'

bincount(...)

bincount(x, weights=None, minlength=0)

Count number of occurrences of each value in array of non-negative ints.

The number of bins (of size 1) is one larger than the largest value in

`x`. If `minlength` is specified, there will be at least this number

of bins in the output array (though it will be longer if necessary,

depending on the contents of `x`).

Each bin gives the number of occurrences of its index value in `x`.

If `weights` is specified the input array is weighted by it, i.e. if a

value ``n`` is found at position ``i``, ``out[n] += weight[i]`` instead

of ``out[n] += 1``.

Parameters

----------

x : array\_like, 1 dimension, nonnegative ints

Input array.

weights : array\_like, optional

Weights, array of the same shape as `x`.

minlength : int, optional

A minimum number of bins for the output array.

.. versionadded:: 1.6.0

Returns

-------

out : ndarray of ints

The result of binning the input array.

The length of `out` is equal to ``np.amax(x)+1``.

Raises

------

ValueError

If the input is not 1-dimensional, or contains elements with negative

values, or if `minlength` is negative.

TypeError

If the type of the input is float or complex.

See Also

--------

histogram, digitize, unique

Examples

--------

>>> np.bincount(np.arange(5))

array([1, 1, 1, 1, 1])

>>> np.bincount(np.array([0, 1, 1, 3, 2, 1, 7]))

array([1, 3, 1, 1, 0, 0, 0, 1])

>>> x = np.array([0, 1, 1, 3, 2, 1, 7, 23])

>>> np.bincount(x).size == np.amax(x)+1

True

The input array needs to be of integer dtype, otherwise a

TypeError is raised:

>>> np.bincount(np.arange(5, dtype=float))

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: array cannot be safely cast to required type

A possible use of ``bincount`` is to perform sums over

variable-size chunks of an array, using the ``weights`` keyword.

>>> w = np.array([0.3, 0.5, 0.2, 0.7, 1., -0.6]) # weights

>>> x = np.array([0, 1, 1, 2, 2, 2])

>>> np.bincount(x, weights=w)

array([ 0.3, 0.7, 1.1])

blackman(M)

Return the Blackman window.

The Blackman window is a taper formed by using the first three

terms of a summation of cosines. It was designed to have close to the

minimal leakage possible. It is close to optimal, only slightly worse

than a Kaiser window.

Parameters

----------

M : int

Number of points in the output window. If zero or less, an empty

array is returned.

Returns

-------

out : ndarray

The window, with the maximum value normalized to one (the value one

appears only if the number of samples is odd).

See Also

--------

bartlett, hamming, hanning, kaiser

Notes

-----

The Blackman window is defined as

.. math:: w(n) = 0.42 - 0.5 \cos(2\pi n/M) + 0.08 \cos(4\pi n/M)

Most references to the Blackman window come from the signal processing

literature, where it is used as one of many windowing functions for

smoothing values. It is also known as an apodization (which means

"removing the foot", i.e. smoothing discontinuities at the beginning

and end of the sampled signal) or tapering function. It is known as a

"near optimal" tapering function, almost as good (by some measures)

as the kaiser window.

References

----------

Blackman, R.B. and Tukey, J.W., (1958) The measurement of power spectra,

Dover Publications, New York.

Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing.

Upper Saddle River, NJ: Prentice-Hall, 1999, pp. 468-471.

Examples

--------

>>> import matplotlib.pyplot as plt

>>> np.blackman(12)

array([-1.38777878e-17, 3.26064346e-02, 1.59903635e-01, # may vary

4.14397981e-01, 7.36045180e-01, 9.67046769e-01,

9.67046769e-01, 7.36045180e-01, 4.14397981e-01,

1.59903635e-01, 3.26064346e-02, -1.38777878e-17])

Plot the window and the frequency response:

>>> from numpy.fft import fft, fftshift

>>> window = np.blackman(51)

>>> plt.plot(window)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Blackman window")

Text(0.5, 1.0, 'Blackman window')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("Sample")

Text(0.5, 0, 'Sample')

>>> plt.show()

>>> plt.figure()

<Figure size 640x480 with 0 Axes>

>>> A = fft(window, 2048) / 25.5

>>> mag = np.abs(fftshift(A))

>>> freq = np.linspace(-0.5, 0.5, len(A))

>>> with np.errstate(divide='ignore', invalid='ignore'):

... response = 20 \* np.log10(mag)

...

>>> response = np.clip(response, -100, 100)

>>> plt.plot(freq, response)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Frequency response of Blackman window")

Text(0.5, 1.0, 'Frequency response of Blackman window')

>>> plt.ylabel("Magnitude [dB]")

Text(0, 0.5, 'Magnitude [dB]')

>>> plt.xlabel("Normalized frequency [cycles per sample]")

Text(0.5, 0, 'Normalized frequency [cycles per sample]')

>>> \_ = plt.axis('tight')

>>> plt.show()

block(arrays)

Assemble an nd-array from nested lists of blocks.

Blocks in the innermost lists are concatenated (see `concatenate`) along

the last dimension (-1), then these are concatenated along the

second-last dimension (-2), and so on until the outermost list is reached.

Blocks can be of any dimension, but will not be broadcasted using the normal

rules. Instead, leading axes of size 1 are inserted, to make ``block.ndim``

the same for all blocks. This is primarily useful for working with scalars,

and means that code like ``np.block([v, 1])`` is valid, where

``v.ndim == 1``.

When the nested list is two levels deep, this allows block matrices to be

constructed from their components.

.. versionadded:: 1.13.0

Parameters

----------

arrays : nested list of array\_like or scalars (but not tuples)

If passed a single ndarray or scalar (a nested list of depth 0), this

is returned unmodified (and not copied).

Elements shapes must match along the appropriate axes (without

broadcasting), but leading 1s will be prepended to the shape as

necessary to make the dimensions match.

Returns

-------

block\_array : ndarray

The array assembled from the given blocks.

The dimensionality of the output is equal to the greatest of:

\* the dimensionality of all the inputs

\* the depth to which the input list is nested

Raises

------

ValueError

\* If list depths are mismatched - for instance, ``[[a, b], c]`` is

illegal, and should be spelt ``[[a, b], [c]]``

\* If lists are empty - for instance, ``[[a, b], []]``

See Also

--------

concatenate : Join a sequence of arrays together.

stack : Stack arrays in sequence along a new dimension.

hstack : Stack arrays in sequence horizontally (column wise).

vstack : Stack arrays in sequence vertically (row wise).

dstack : Stack arrays in sequence depth wise (along third dimension).

vsplit : Split array into a list of multiple sub-arrays vertically.

Notes

-----

When called with only scalars, ``np.block`` is equivalent to an ndarray

call. So ``np.block([[1, 2], [3, 4]])`` is equivalent to

``np.array([[1, 2], [3, 4]])``.

This function does not enforce that the blocks lie on a fixed grid.

``np.block([[a, b], [c, d]])`` is not restricted to arrays of the form::

AAAbb

AAAbb

cccDD

But is also allowed to produce, for some ``a, b, c, d``::

AAAbb

AAAbb

cDDDD

Since concatenation happens along the last axis first, `block` is \_not\_

capable of producing the following directly::

AAAbb

cccbb

cccDD

Matlab's "square bracket stacking", ``[A, B, ...; p, q, ...]``, is

equivalent to ``np.block([[A, B, ...], [p, q, ...]])``.

Examples

--------

The most common use of this function is to build a block matrix

>>> A = np.eye(2) \* 2

>>> B = np.eye(3) \* 3

>>> np.block([

... [A, np.zeros((2, 3))],

... [np.ones((3, 2)), B ]

... ])

array([[2., 0., 0., 0., 0.],

[0., 2., 0., 0., 0.],

[1., 1., 3., 0., 0.],

[1., 1., 0., 3., 0.],

[1., 1., 0., 0., 3.]])

With a list of depth 1, `block` can be used as `hstack`

>>> np.block([1, 2, 3]) # hstack([1, 2, 3])

array([1, 2, 3])

>>> a = np.array([1, 2, 3])

>>> b = np.array([2, 3, 4])

>>> np.block([a, b, 10]) # hstack([a, b, 10])

array([ 1, 2, 3, 2, 3, 4, 10])

>>> A = np.ones((2, 2), int)

>>> B = 2 \* A

>>> np.block([A, B]) # hstack([A, B])

array([[1, 1, 2, 2],

[1, 1, 2, 2]])

With a list of depth 2, `block` can be used in place of `vstack`:

>>> a = np.array([1, 2, 3])

>>> b = np.array([2, 3, 4])

>>> np.block([[a], [b]]) # vstack([a, b])

array([[1, 2, 3],

[2, 3, 4]])

>>> A = np.ones((2, 2), int)

>>> B = 2 \* A

>>> np.block([[A], [B]]) # vstack([A, B])

array([[1, 1],

[1, 1],

[2, 2],

[2, 2]])

It can also be used in places of `atleast\_1d` and `atleast\_2d`

>>> a = np.array(0)

>>> b = np.array([1])

>>> np.block([a]) # atleast\_1d(a)

array([0])

>>> np.block([b]) # atleast\_1d(b)

array([1])

>>> np.block([[a]]) # atleast\_2d(a)

array([[0]])

>>> np.block([[b]]) # atleast\_2d(b)

array([[1]])

bmat(obj, ldict=None, gdict=None)

Build a matrix object from a string, nested sequence, or array.

Parameters

----------

obj : str or array\_like

Input data. If a string, variables in the current scope may be

referenced by name.

ldict : dict, optional

A dictionary that replaces local operands in current frame.

Ignored if `obj` is not a string or `gdict` is `None`.

gdict : dict, optional

A dictionary that replaces global operands in current frame.

Ignored if `obj` is not a string.

Returns

-------

out : matrix

Returns a matrix object, which is a specialized 2-D array.

See Also

--------

block :

A generalization of this function for N-d arrays, that returns normal

ndarrays.

Examples

--------

>>> A = np.mat('1 1; 1 1')

>>> B = np.mat('2 2; 2 2')

>>> C = np.mat('3 4; 5 6')

>>> D = np.mat('7 8; 9 0')

All the following expressions construct the same block matrix:

>>> np.bmat([[A, B], [C, D]])

matrix([[1, 1, 2, 2],

[1, 1, 2, 2],

[3, 4, 7, 8],

[5, 6, 9, 0]])

>>> np.bmat(np.r\_[np.c\_[A, B], np.c\_[C, D]])

matrix([[1, 1, 2, 2],

[1, 1, 2, 2],

[3, 4, 7, 8],

[5, 6, 9, 0]])

>>> np.bmat('A,B; C,D')

matrix([[1, 1, 2, 2],

[1, 1, 2, 2],

[3, 4, 7, 8],

[5, 6, 9, 0]])

broadcast\_arrays(\*args, \*\*kwargs)

Broadcast any number of arrays against each other.

Parameters

----------

`\*args` : array\_likes

The arrays to broadcast.

subok : bool, optional

If True, then sub-classes will be passed-through, otherwise

the returned arrays will be forced to be a base-class array (default).

Returns

-------

broadcasted : list of arrays

These arrays are views on the original arrays. They are typically

not contiguous. Furthermore, more than one element of a

broadcasted array may refer to a single memory location. If you need

to write to the arrays, make copies first. While you can set the

``writable`` flag True, writing to a single output value may end up

changing more than one location in the output array.

.. deprecated:: 1.17

The output is currently marked so that if written to, a deprecation

warning will be emitted. A future version will set the

``writable`` flag False so writing to it will raise an error.

Examples

--------

>>> x = np.array([[1,2,3]])

>>> y = np.array([[4],[5]])

>>> np.broadcast\_arrays(x, y)

[array([[1, 2, 3],

[1, 2, 3]]), array([[4, 4, 4],

[5, 5, 5]])]

Here is a useful idiom for getting contiguous copies instead of

non-contiguous views.

>>> [np.array(a) for a in np.broadcast\_arrays(x, y)]

[array([[1, 2, 3],

[1, 2, 3]]), array([[4, 4, 4],

[5, 5, 5]])]

broadcast\_to(array, shape, subok=False)

Broadcast an array to a new shape.

Parameters

----------

array : array\_like

The array to broadcast.

shape : tuple

The shape of the desired array.

subok : bool, optional

If True, then sub-classes will be passed-through, otherwise

the returned array will be forced to be a base-class array (default).

Returns

-------

broadcast : array

A readonly view on the original array with the given shape. It is

typically not contiguous. Furthermore, more than one element of a

broadcasted array may refer to a single memory location.

Raises

------

ValueError

If the array is not compatible with the new shape according to NumPy's

broadcasting rules.

Notes

-----

.. versionadded:: 1.10.0

Examples

--------

>>> x = np.array([1, 2, 3])

>>> np.broadcast\_to(x, (3, 3))

array([[1, 2, 3],

[1, 2, 3],

[1, 2, 3]])

busday\_count(...)

busday\_count(begindates, enddates, weekmask='1111100', holidays=[], busdaycal=None, out=None)

Counts the number of valid days between `begindates` and

`enddates`, not including the day of `enddates`.

If ``enddates`` specifies a date value that is earlier than the

corresponding ``begindates`` date value, the count will be negative.

.. versionadded:: 1.7.0

Parameters

----------

begindates : array\_like of datetime64[D]

The array of the first dates for counting.

enddates : array\_like of datetime64[D]

The array of the end dates for counting, which are excluded

from the count themselves.

weekmask : str or array\_like of bool, optional

A seven-element array indicating which of Monday through Sunday are

valid days. May be specified as a length-seven list or array, like

[1,1,1,1,1,0,0]; a length-seven string, like '1111100'; or a string

like "Mon Tue Wed Thu Fri", made up of 3-character abbreviations for

weekdays, optionally separated by white space. Valid abbreviations

are: Mon Tue Wed Thu Fri Sat Sun

holidays : array\_like of datetime64[D], optional

An array of dates to consider as invalid dates. They may be

specified in any order, and NaT (not-a-time) dates are ignored.

This list is saved in a normalized form that is suited for

fast calculations of valid days.

busdaycal : busdaycalendar, optional

A `busdaycalendar` object which specifies the valid days. If this

parameter is provided, neither weekmask nor holidays may be

provided.

out : array of int, optional

If provided, this array is filled with the result.

Returns

-------

out : array of int

An array with a shape from broadcasting ``begindates`` and ``enddates``

together, containing the number of valid days between

the begin and end dates.

See Also

--------

busdaycalendar: An object that specifies a custom set of valid days.

is\_busday : Returns a boolean array indicating valid days.

busday\_offset : Applies an offset counted in valid days.

Examples

--------

>>> # Number of weekdays in January 2011

... np.busday\_count('2011-01', '2011-02')

21

>>> # Number of weekdays in 2011

>>> np.busday\_count('2011', '2012')

260

>>> # Number of Saturdays in 2011

... np.busday\_count('2011', '2012', weekmask='Sat')

53

busday\_offset(...)

busday\_offset(dates, offsets, roll='raise', weekmask='1111100', holidays=None, busdaycal=None, out=None)

First adjusts the date to fall on a valid day according to

the ``roll`` rule, then applies offsets to the given dates

counted in valid days.

.. versionadded:: 1.7.0

Parameters

----------

dates : array\_like of datetime64[D]

The array of dates to process.

offsets : array\_like of int

The array of offsets, which is broadcast with ``dates``.

roll : {'raise', 'nat', 'forward', 'following', 'backward', 'preceding', 'modifiedfollowing', 'modifiedpreceding'}, optional

How to treat dates that do not fall on a valid day. The default

is 'raise'.

\* 'raise' means to raise an exception for an invalid day.

\* 'nat' means to return a NaT (not-a-time) for an invalid day.

\* 'forward' and 'following' mean to take the first valid day

later in time.

\* 'backward' and 'preceding' mean to take the first valid day

earlier in time.

\* 'modifiedfollowing' means to take the first valid day

later in time unless it is across a Month boundary, in which

case to take the first valid day earlier in time.

\* 'modifiedpreceding' means to take the first valid day

earlier in time unless it is across a Month boundary, in which

case to take the first valid day later in time.

weekmask : str or array\_like of bool, optional

A seven-element array indicating which of Monday through Sunday are

valid days. May be specified as a length-seven list or array, like

[1,1,1,1,1,0,0]; a length-seven string, like '1111100'; or a string

like "Mon Tue Wed Thu Fri", made up of 3-character abbreviations for

weekdays, optionally separated by white space. Valid abbreviations

are: Mon Tue Wed Thu Fri Sat Sun

holidays : array\_like of datetime64[D], optional

An array of dates to consider as invalid dates. They may be

specified in any order, and NaT (not-a-time) dates are ignored.

This list is saved in a normalized form that is suited for

fast calculations of valid days.

busdaycal : busdaycalendar, optional

A `busdaycalendar` object which specifies the valid days. If this

parameter is provided, neither weekmask nor holidays may be

provided.

out : array of datetime64[D], optional

If provided, this array is filled with the result.

Returns

-------

out : array of datetime64[D]

An array with a shape from broadcasting ``dates`` and ``offsets``

together, containing the dates with offsets applied.

See Also

--------

busdaycalendar: An object that specifies a custom set of valid days.

is\_busday : Returns a boolean array indicating valid days.

busday\_count : Counts how many valid days are in a half-open date range.

Examples

--------

>>> # First business day in October 2011 (not accounting for holidays)

... np.busday\_offset('2011-10', 0, roll='forward')

numpy.datetime64('2011-10-03')

>>> # Last business day in February 2012 (not accounting for holidays)

... np.busday\_offset('2012-03', -1, roll='forward')

numpy.datetime64('2012-02-29')

>>> # Third Wednesday in January 2011

... np.busday\_offset('2011-01', 2, roll='forward', weekmask='Wed')

numpy.datetime64('2011-01-19')

>>> # 2012 Mother's Day in Canada and the U.S.

... np.busday\_offset('2012-05', 1, roll='forward', weekmask='Sun')

numpy.datetime64('2012-05-13')

>>> # First business day on or after a date

... np.busday\_offset('2011-03-20', 0, roll='forward')

numpy.datetime64('2011-03-21')

>>> np.busday\_offset('2011-03-22', 0, roll='forward')

numpy.datetime64('2011-03-22')

>>> # First business day after a date

... np.busday\_offset('2011-03-20', 1, roll='backward')

numpy.datetime64('2011-03-21')

>>> np.busday\_offset('2011-03-22', 1, roll='backward')

numpy.datetime64('2011-03-23')

byte\_bounds(a)

Returns pointers to the end-points of an array.

Parameters

----------

a : ndarray

Input array. It must conform to the Python-side of the array

interface.

Returns

-------

(low, high) : tuple of 2 integers

The first integer is the first byte of the array, the second

integer is just past the last byte of the array. If `a` is not

contiguous it will not use every byte between the (`low`, `high`)

values.

Examples

--------

>>> I = np.eye(2, dtype='f'); I.dtype

dtype('float32')

>>> low, high = np.byte\_bounds(I)

>>> high - low == I.size\*I.itemsize

True

>>> I = np.eye(2); I.dtype

dtype('float64')

>>> low, high = np.byte\_bounds(I)

>>> high - low == I.size\*I.itemsize

True

can\_cast(...)

can\_cast(from\_, to, casting='safe')

Returns True if cast between data types can occur according to the

casting rule. If from is a scalar or array scalar, also returns

True if the scalar value can be cast without overflow or truncation

to an integer.

Parameters

----------

from\_ : dtype, dtype specifier, scalar, or array

Data type, scalar, or array to cast from.

to : dtype or dtype specifier

Data type to cast to.

casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

Controls what kind of data casting may occur.

\* 'no' means the data types should not be cast at all.

\* 'equiv' means only byte-order changes are allowed.

\* 'safe' means only casts which can preserve values are allowed.

\* 'same\_kind' means only safe casts or casts within a kind,

like float64 to float32, are allowed.

\* 'unsafe' means any data conversions may be done.

Returns

-------

out : bool

True if cast can occur according to the casting rule.

Notes

-----

.. versionchanged:: 1.17.0

Casting between a simple data type and a structured one is possible only

for "unsafe" casting. Casting to multiple fields is allowed, but

casting from multiple fields is not.

.. versionchanged:: 1.9.0

Casting from numeric to string types in 'safe' casting mode requires

that the string dtype length is long enough to store the maximum

integer/float value converted.

See also

--------

dtype, result\_type

Examples

--------

Basic examples

>>> np.can\_cast(np.int32, np.int64)

True

>>> np.can\_cast(np.float64, complex)

True

>>> np.can\_cast(complex, float)

False

>>> np.can\_cast('i8', 'f8')

True

>>> np.can\_cast('i8', 'f4')

False

>>> np.can\_cast('i4', 'S4')

False

Casting scalars

>>> np.can\_cast(100, 'i1')

True

>>> np.can\_cast(150, 'i1')

False

>>> np.can\_cast(150, 'u1')

True

>>> np.can\_cast(3.5e100, np.float32)

False

>>> np.can\_cast(1000.0, np.float32)

True

Array scalar checks the value, array does not

>>> np.can\_cast(np.array(1000.0), np.float32)

True

>>> np.can\_cast(np.array([1000.0]), np.float32)

False

Using the casting rules

>>> np.can\_cast('i8', 'i8', 'no')

True

>>> np.can\_cast('<i8', '>i8', 'no')

False

>>> np.can\_cast('<i8', '>i8', 'equiv')

True

>>> np.can\_cast('<i4', '>i8', 'equiv')

False

>>> np.can\_cast('<i4', '>i8', 'safe')

True

>>> np.can\_cast('<i8', '>i4', 'safe')

False

>>> np.can\_cast('<i8', '>i4', 'same\_kind')

True

>>> np.can\_cast('<i8', '>u4', 'same\_kind')

False

>>> np.can\_cast('<i8', '>u4', 'unsafe')

True

choose(a, choices, out=None, mode='raise')

Construct an array from an index array and a set of arrays to choose from.

First of all, if confused or uncertain, definitely look at the Examples -

in its full generality, this function is less simple than it might

seem from the following code description (below ndi =

`numpy.lib.index\_tricks`):

``np.choose(a,c) == np.array([c[a[I]][I] for I in ndi.ndindex(a.shape)])``.

But this omits some subtleties. Here is a fully general summary:

Given an "index" array (`a`) of integers and a sequence of `n` arrays

(`choices`), `a` and each choice array are first broadcast, as necessary,

to arrays of a common shape; calling these \*Ba\* and \*Bchoices[i], i =

0,...,n-1\* we have that, necessarily, ``Ba.shape == Bchoices[i].shape``

for each `i`. Then, a new array with shape ``Ba.shape`` is created as

follows:

\* if ``mode=raise`` (the default), then, first of all, each element of

`a` (and thus `Ba`) must be in the range `[0, n-1]`; now, suppose that

`i` (in that range) is the value at the `(j0, j1, ..., jm)` position

in `Ba` - then the value at the same position in the new array is the

value in `Bchoices[i]` at that same position;

\* if ``mode=wrap``, values in `a` (and thus `Ba`) may be any (signed)

integer; modular arithmetic is used to map integers outside the range

`[0, n-1]` back into that range; and then the new array is constructed

as above;

\* if ``mode=clip``, values in `a` (and thus `Ba`) may be any (signed)

integer; negative integers are mapped to 0; values greater than `n-1`

are mapped to `n-1`; and then the new array is constructed as above.

Parameters

----------

a : int array

This array must contain integers in `[0, n-1]`, where `n` is the number

of choices, unless ``mode=wrap`` or ``mode=clip``, in which cases any

integers are permissible.

choices : sequence of arrays

Choice arrays. `a` and all of the choices must be broadcastable to the

same shape. If `choices` is itself an array (not recommended), then

its outermost dimension (i.e., the one corresponding to

``choices.shape[0]``) is taken as defining the "sequence".

out : array, optional

If provided, the result will be inserted into this array. It should

be of the appropriate shape and dtype. Note that `out` is always

buffered if `mode='raise'`; use other modes for better performance.

mode : {'raise' (default), 'wrap', 'clip'}, optional

Specifies how indices outside `[0, n-1]` will be treated:

\* 'raise' : an exception is raised

\* 'wrap' : value becomes value mod `n`

\* 'clip' : values < 0 are mapped to 0, values > n-1 are mapped to n-1

Returns

-------

merged\_array : array

The merged result.

Raises

------

ValueError: shape mismatch

If `a` and each choice array are not all broadcastable to the same

shape.

See Also

--------

ndarray.choose : equivalent method

Notes

-----

To reduce the chance of misinterpretation, even though the following

"abuse" is nominally supported, `choices` should neither be, nor be

thought of as, a single array, i.e., the outermost sequence-like container

should be either a list or a tuple.

Examples

--------

>>> choices = [[0, 1, 2, 3], [10, 11, 12, 13],

... [20, 21, 22, 23], [30, 31, 32, 33]]

>>> np.choose([2, 3, 1, 0], choices

... # the first element of the result will be the first element of the

... # third (2+1) "array" in choices, namely, 20; the second element

... # will be the second element of the fourth (3+1) choice array, i.e.,

... # 31, etc.

... )

array([20, 31, 12, 3])

>>> np.choose([2, 4, 1, 0], choices, mode='clip') # 4 goes to 3 (4-1)

array([20, 31, 12, 3])

>>> # because there are 4 choice arrays

>>> np.choose([2, 4, 1, 0], choices, mode='wrap') # 4 goes to (4 mod 4)

array([20, 1, 12, 3])

>>> # i.e., 0

A couple examples illustrating how choose broadcasts:

>>> a = [[1, 0, 1], [0, 1, 0], [1, 0, 1]]

>>> choices = [-10, 10]

>>> np.choose(a, choices)

array([[ 10, -10, 10],

[-10, 10, -10],

[ 10, -10, 10]])

>>> # With thanks to Anne Archibald

>>> a = np.array([0, 1]).reshape((2,1,1))

>>> c1 = np.array([1, 2, 3]).reshape((1,3,1))

>>> c2 = np.array([-1, -2, -3, -4, -5]).reshape((1,1,5))

>>> np.choose(a, (c1, c2)) # result is 2x3x5, res[0,:,:]=c1, res[1,:,:]=c2

array([[[ 1, 1, 1, 1, 1],

[ 2, 2, 2, 2, 2],

[ 3, 3, 3, 3, 3]],

[[-1, -2, -3, -4, -5],

[-1, -2, -3, -4, -5],

[-1, -2, -3, -4, -5]]])

clip(a, a\_min, a\_max, out=None, \*\*kwargs)

Clip (limit) the values in an array.

Given an interval, values outside the interval are clipped to

the interval edges. For example, if an interval of ``[0, 1]``

is specified, values smaller than 0 become 0, and values larger

than 1 become 1.

Equivalent to but faster than ``np.maximum(a\_min, np.minimum(a, a\_max))``.

No check is performed to ensure ``a\_min < a\_max``.

Parameters

----------

a : array\_like

Array containing elements to clip.

a\_min : scalar or array\_like or `None`

Minimum value. If `None`, clipping is not performed on lower

interval edge. Not more than one of `a\_min` and `a\_max` may be

`None`.

a\_max : scalar or array\_like or `None`

Maximum value. If `None`, clipping is not performed on upper

interval edge. Not more than one of `a\_min` and `a\_max` may be

`None`. If `a\_min` or `a\_max` are array\_like, then the three

arrays will be broadcasted to match their shapes.

out : ndarray, optional

The results will be placed in this array. It may be the input

array for in-place clipping. `out` must be of the right shape

to hold the output. Its type is preserved.

\*\*kwargs

For other keyword-only arguments, see the

:ref:`ufunc docs <ufuncs.kwargs>`.

.. versionadded:: 1.17.0

Returns

-------

clipped\_array : ndarray

An array with the elements of `a`, but where values

< `a\_min` are replaced with `a\_min`, and those > `a\_max`

with `a\_max`.

See Also

--------

numpy.doc.ufuncs : Section "Output arguments"

Examples

--------

>>> a = np.arange(10)

>>> np.clip(a, 1, 8)

array([1, 1, 2, 3, 4, 5, 6, 7, 8, 8])

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.clip(a, 3, 6, out=a)

array([3, 3, 3, 3, 4, 5, 6, 6, 6, 6])

>>> a = np.arange(10)

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.clip(a, [3, 4, 1, 1, 1, 4, 4, 4, 4, 4], 8)

array([3, 4, 2, 3, 4, 5, 6, 7, 8, 8])

column\_stack(tup)

Stack 1-D arrays as columns into a 2-D array.

Take a sequence of 1-D arrays and stack them as columns

to make a single 2-D array. 2-D arrays are stacked as-is,

just like with `hstack`. 1-D arrays are turned into 2-D columns

first.

Parameters

----------

tup : sequence of 1-D or 2-D arrays.

Arrays to stack. All of them must have the same first dimension.

Returns

-------

stacked : 2-D array

The array formed by stacking the given arrays.

See Also

--------

stack, hstack, vstack, concatenate

Examples

--------

>>> a = np.array((1,2,3))

>>> b = np.array((2,3,4))

>>> np.column\_stack((a,b))

array([[1, 2],

[2, 3],

[3, 4]])

common\_type(\*arrays)

Return a scalar type which is common to the input arrays.

The return type will always be an inexact (i.e. floating point) scalar

type, even if all the arrays are integer arrays. If one of the inputs is

an integer array, the minimum precision type that is returned is a

64-bit floating point dtype.

All input arrays except int64 and uint64 can be safely cast to the

returned dtype without loss of information.

Parameters

----------

array1, array2, ... : ndarrays

Input arrays.

Returns

-------

out : data type code

Data type code.

See Also

--------

dtype, mintypecode

Examples

--------

>>> np.common\_type(np.arange(2, dtype=np.float32))

<class 'numpy.float32'>

>>> np.common\_type(np.arange(2, dtype=np.float32), np.arange(2))

<class 'numpy.float64'>

>>> np.common\_type(np.arange(4), np.array([45, 6.j]), np.array([45.0]))

<class 'numpy.complex128'>

compare\_chararrays(...)

compare\_chararrays(a, b, cmp\_op, rstrip)

Performs element-wise comparison of two string arrays using the

comparison operator specified by `cmp\_op`.

Parameters

----------

a, b : array\_like

Arrays to be compared.

cmp\_op : {"<", "<=", "==", ">=", ">", "!="}

Type of comparison.

rstrip : Boolean

If True, the spaces at the end of Strings are removed before the comparison.

Returns

-------

out : ndarray

The output array of type Boolean with the same shape as a and b.

Raises

------

ValueError

If `cmp\_op` is not valid.

TypeError

If at least one of `a` or `b` is a non-string array

Examples

--------

>>> a = np.array(["a", "b", "cde"])

>>> b = np.array(["a", "a", "dec"])

>>> np.compare\_chararrays(a, b, ">", True)

array([False, True, False])

compress(condition, a, axis=None, out=None)

Return selected slices of an array along given axis.

When working along a given axis, a slice along that axis is returned in

`output` for each index where `condition` evaluates to True. When

working on a 1-D array, `compress` is equivalent to `extract`.

Parameters

----------

condition : 1-D array of bools

Array that selects which entries to return. If len(condition)

is less than the size of `a` along the given axis, then output is

truncated to the length of the condition array.

a : array\_like

Array from which to extract a part.

axis : int, optional

Axis along which to take slices. If None (default), work on the

flattened array.

out : ndarray, optional

Output array. Its type is preserved and it must be of the right

shape to hold the output.

Returns

-------

compressed\_array : ndarray

A copy of `a` without the slices along axis for which `condition`

is false.

See Also

--------

take, choose, diag, diagonal, select

ndarray.compress : Equivalent method in ndarray

np.extract: Equivalent method when working on 1-D arrays

numpy.doc.ufuncs : Section "Output arguments"

Examples

--------

>>> a = np.array([[1, 2], [3, 4], [5, 6]])

>>> a

array([[1, 2],

[3, 4],

[5, 6]])

>>> np.compress([0, 1], a, axis=0)

array([[3, 4]])

>>> np.compress([False, True, True], a, axis=0)

array([[3, 4],

[5, 6]])

>>> np.compress([False, True], a, axis=1)

array([[2],

[4],

[6]])

Working on the flattened array does not return slices along an axis but

selects elements.

>>> np.compress([False, True], a)

array([2])

concatenate(...)

concatenate((a1, a2, ...), axis=0, out=None)

Join a sequence of arrays along an existing axis.

Parameters

----------

a1, a2, ... : sequence of array\_like

The arrays must have the same shape, except in the dimension

corresponding to `axis` (the first, by default).

axis : int, optional

The axis along which the arrays will be joined. If axis is None,

arrays are flattened before use. Default is 0.

out : ndarray, optional

If provided, the destination to place the result. The shape must be

correct, matching that of what concatenate would have returned if no

out argument were specified.

Returns

-------

res : ndarray

The concatenated array.

See Also

--------

ma.concatenate : Concatenate function that preserves input masks.

array\_split : Split an array into multiple sub-arrays of equal or

near-equal size.

split : Split array into a list of multiple sub-arrays of equal size.

hsplit : Split array into multiple sub-arrays horizontally (column wise)

vsplit : Split array into multiple sub-arrays vertically (row wise)

dsplit : Split array into multiple sub-arrays along the 3rd axis (depth).

stack : Stack a sequence of arrays along a new axis.

hstack : Stack arrays in sequence horizontally (column wise)

vstack : Stack arrays in sequence vertically (row wise)

dstack : Stack arrays in sequence depth wise (along third dimension)

block : Assemble arrays from blocks.

Notes

-----

When one or more of the arrays to be concatenated is a MaskedArray,

this function will return a MaskedArray object instead of an ndarray,

but the input masks are \*not\* preserved. In cases where a MaskedArray

is expected as input, use the ma.concatenate function from the masked

array module instead.

Examples

--------

>>> a = np.array([[1, 2], [3, 4]])

>>> b = np.array([[5, 6]])

>>> np.concatenate((a, b), axis=0)

array([[1, 2],

[3, 4],

[5, 6]])

>>> np.concatenate((a, b.T), axis=1)

array([[1, 2, 5],

[3, 4, 6]])

>>> np.concatenate((a, b), axis=None)

array([1, 2, 3, 4, 5, 6])

This function will not preserve masking of MaskedArray inputs.

>>> a = np.ma.arange(3)

>>> a[1] = np.ma.masked

>>> b = np.arange(2, 5)

>>> a

masked\_array(data=[0, --, 2],

mask=[False, True, False],

fill\_value=999999)

>>> b

array([2, 3, 4])

>>> np.concatenate([a, b])

masked\_array(data=[0, 1, 2, 2, 3, 4],

mask=False,

fill\_value=999999)

>>> np.ma.concatenate([a, b])

masked\_array(data=[0, --, 2, 2, 3, 4],

mask=[False, True, False, False, False, False],

fill\_value=999999)

convolve(a, v, mode='full')

Returns the discrete, linear convolution of two one-dimensional sequences.

The convolution operator is often seen in signal processing, where it

models the effect of a linear time-invariant system on a signal [1]\_. In

probability theory, the sum of two independent random variables is

distributed according to the convolution of their individual

distributions.

If `v` is longer than `a`, the arrays are swapped before computation.

Parameters

----------

a : (N,) array\_like

First one-dimensional input array.

v : (M,) array\_like

Second one-dimensional input array.

mode : {'full', 'valid', 'same'}, optional

'full':

By default, mode is 'full'. This returns the convolution

at each point of overlap, with an output shape of (N+M-1,). At

the end-points of the convolution, the signals do not overlap

completely, and boundary effects may be seen.

'same':

Mode 'same' returns output of length ``max(M, N)``. Boundary

effects are still visible.

'valid':

Mode 'valid' returns output of length

``max(M, N) - min(M, N) + 1``. The convolution product is only given

for points where the signals overlap completely. Values outside

the signal boundary have no effect.

Returns

-------

out : ndarray

Discrete, linear convolution of `a` and `v`.

See Also

--------

scipy.signal.fftconvolve : Convolve two arrays using the Fast Fourier

Transform.

scipy.linalg.toeplitz : Used to construct the convolution operator.

polymul : Polynomial multiplication. Same output as convolve, but also

accepts poly1d objects as input.

Notes

-----

The discrete convolution operation is defined as

.. math:: (a \* v)[n] = \sum\_{m = -\infty}^{\infty} a[m] v[n - m]

It can be shown that a convolution :math:`x(t) \* y(t)` in time/space

is equivalent to the multiplication :math:`X(f) Y(f)` in the Fourier

domain, after appropriate padding (padding is necessary to prevent

circular convolution). Since multiplication is more efficient (faster)

than convolution, the function `scipy.signal.fftconvolve` exploits the

FFT to calculate the convolution of large data-sets.

References

----------

.. [1] Wikipedia, "Convolution",

https://en.wikipedia.org/wiki/Convolution

Examples

--------

Note how the convolution operator flips the second array

before "sliding" the two across one another:

>>> np.convolve([1, 2, 3], [0, 1, 0.5])

array([0. , 1. , 2.5, 4. , 1.5])

Only return the middle values of the convolution.

Contains boundary effects, where zeros are taken

into account:

>>> np.convolve([1,2,3],[0,1,0.5], 'same')

array([1. , 2.5, 4. ])

The two arrays are of the same length, so there

is only one position where they completely overlap:

>>> np.convolve([1,2,3],[0,1,0.5], 'valid')

array([2.5])

copy(a, order='K')

Return an array copy of the given object.

Parameters

----------

a : array\_like

Input data.

order : {'C', 'F', 'A', 'K'}, optional

Controls the memory layout of the copy. 'C' means C-order,

'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

'C' otherwise. 'K' means match the layout of `a` as closely

as possible. (Note that this function and :meth:`ndarray.copy` are very

similar, but have different default values for their order=

arguments.)

Returns

-------

arr : ndarray

Array interpretation of `a`.

Notes

-----

This is equivalent to:

>>> np.array(a, copy=True) #doctest: +SKIP

Examples

--------

Create an array x, with a reference y and a copy z:

>>> x = np.array([1, 2, 3])

>>> y = x

>>> z = np.copy(x)

Note that, when we modify x, y changes, but not z:

>>> x[0] = 10

>>> x[0] == y[0]

True

>>> x[0] == z[0]

False

copyto(...)

copyto(dst, src, casting='same\_kind', where=True)

Copies values from one array to another, broadcasting as necessary.

Raises a TypeError if the `casting` rule is violated, and if

`where` is provided, it selects which elements to copy.

.. versionadded:: 1.7.0

Parameters

----------

dst : ndarray

The array into which values are copied.

src : array\_like

The array from which values are copied.

casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

Controls what kind of data casting may occur when copying.

\* 'no' means the data types should not be cast at all.

\* 'equiv' means only byte-order changes are allowed.

\* 'safe' means only casts which can preserve values are allowed.

\* 'same\_kind' means only safe casts or casts within a kind,

like float64 to float32, are allowed.

\* 'unsafe' means any data conversions may be done.

where : array\_like of bool, optional

A boolean array which is broadcasted to match the dimensions

of `dst`, and selects elements to copy from `src` to `dst`

wherever it contains the value True.

corrcoef(x, y=None, rowvar=True, bias=<no value>, ddof=<no value>)

Return Pearson product-moment correlation coefficients.

Please refer to the documentation for `cov` for more detail. The

relationship between the correlation coefficient matrix, `R`, and the

covariance matrix, `C`, is

.. math:: R\_{ij} = \frac{ C\_{ij} } { \sqrt{ C\_{ii} \* C\_{jj} } }

The values of `R` are between -1 and 1, inclusive.

Parameters

----------

x : array\_like

A 1-D or 2-D array containing multiple variables and observations.

Each row of `x` represents a variable, and each column a single

observation of all those variables. Also see `rowvar` below.

y : array\_like, optional

An additional set of variables and observations. `y` has the same

shape as `x`.

rowvar : bool, optional

If `rowvar` is True (default), then each row represents a

variable, with observations in the columns. Otherwise, the relationship

is transposed: each column represents a variable, while the rows

contain observations.

bias : \_NoValue, optional

Has no effect, do not use.

.. deprecated:: 1.10.0

ddof : \_NoValue, optional

Has no effect, do not use.

.. deprecated:: 1.10.0

Returns

-------

R : ndarray

The correlation coefficient matrix of the variables.

See Also

--------

cov : Covariance matrix

Notes

-----

Due to floating point rounding the resulting array may not be Hermitian,

the diagonal elements may not be 1, and the elements may not satisfy the

inequality abs(a) <= 1. The real and imaginary parts are clipped to the

interval [-1, 1] in an attempt to improve on that situation but is not

much help in the complex case.

This function accepts but discards arguments `bias` and `ddof`. This is

for backwards compatibility with previous versions of this function. These

arguments had no effect on the return values of the function and can be

safely ignored in this and previous versions of numpy.

correlate(a, v, mode='valid')

Cross-correlation of two 1-dimensional sequences.

This function computes the correlation as generally defined in signal

processing texts::

c\_{av}[k] = sum\_n a[n+k] \* conj(v[n])

with a and v sequences being zero-padded where necessary and conj being

the conjugate.

Parameters

----------

a, v : array\_like

Input sequences.

mode : {'valid', 'same', 'full'}, optional

Refer to the `convolve` docstring. Note that the default

is 'valid', unlike `convolve`, which uses 'full'.

old\_behavior : bool

`old\_behavior` was removed in NumPy 1.10. If you need the old

behavior, use `multiarray.correlate`.

Returns

-------

out : ndarray

Discrete cross-correlation of `a` and `v`.

See Also

--------

convolve : Discrete, linear convolution of two one-dimensional sequences.

multiarray.correlate : Old, no conjugate, version of correlate.

Notes

-----

The definition of correlation above is not unique and sometimes correlation

may be defined differently. Another common definition is::

c'\_{av}[k] = sum\_n a[n] conj(v[n+k])

which is related to ``c\_{av}[k]`` by ``c'\_{av}[k] = c\_{av}[-k]``.

Examples

--------

>>> np.correlate([1, 2, 3], [0, 1, 0.5])

array([3.5])

>>> np.correlate([1, 2, 3], [0, 1, 0.5], "same")

array([2. , 3.5, 3. ])

>>> np.correlate([1, 2, 3], [0, 1, 0.5], "full")

array([0.5, 2. , 3.5, 3. , 0. ])

Using complex sequences:

>>> np.correlate([1+1j, 2, 3-1j], [0, 1, 0.5j], 'full')

array([ 0.5-0.5j, 1.0+0.j , 1.5-1.5j, 3.0-1.j , 0.0+0.j ])

Note that you get the time reversed, complex conjugated result

when the two input sequences change places, i.e.,

``c\_{va}[k] = c^{\*}\_{av}[-k]``:

>>> np.correlate([0, 1, 0.5j], [1+1j, 2, 3-1j], 'full')

array([ 0.0+0.j , 3.0+1.j , 1.5+1.5j, 1.0+0.j , 0.5+0.5j])

count\_nonzero(a, axis=None)

Counts the number of non-zero values in the array ``a``.

The word "non-zero" is in reference to the Python 2.x

built-in method ``\_\_nonzero\_\_()`` (renamed ``\_\_bool\_\_()``

in Python 3.x) of Python objects that tests an object's

"truthfulness". For example, any number is considered

truthful if it is nonzero, whereas any string is considered

truthful if it is not the empty string. Thus, this function

(recursively) counts how many elements in ``a`` (and in

sub-arrays thereof) have their ``\_\_nonzero\_\_()`` or ``\_\_bool\_\_()``

method evaluated to ``True``.

Parameters

----------

a : array\_like

The array for which to count non-zeros.

axis : int or tuple, optional

Axis or tuple of axes along which to count non-zeros.

Default is None, meaning that non-zeros will be counted

along a flattened version of ``a``.

.. versionadded:: 1.12.0

Returns

-------

count : int or array of int

Number of non-zero values in the array along a given axis.

Otherwise, the total number of non-zero values in the array

is returned.

See Also

--------

nonzero : Return the coordinates of all the non-zero values.

Examples

--------

>>> np.count\_nonzero(np.eye(4))

4

>>> np.count\_nonzero([[0,1,7,0,0],[3,0,0,2,19]])

5

>>> np.count\_nonzero([[0,1,7,0,0],[3,0,0,2,19]], axis=0)

array([1, 1, 1, 1, 1])

>>> np.count\_nonzero([[0,1,7,0,0],[3,0,0,2,19]], axis=1)

array([2, 3])

cov(m, y=None, rowvar=True, bias=False, ddof=None, fweights=None, aweights=None)

Estimate a covariance matrix, given data and weights.

Covariance indicates the level to which two variables vary together.

If we examine N-dimensional samples, :math:`X = [x\_1, x\_2, ... x\_N]^T`,

then the covariance matrix element :math:`C\_{ij}` is the covariance of

:math:`x\_i` and :math:`x\_j`. The element :math:`C\_{ii}` is the variance

of :math:`x\_i`.

See the notes for an outline of the algorithm.

Parameters

----------

m : array\_like

A 1-D or 2-D array containing multiple variables and observations.

Each row of `m` represents a variable, and each column a single

observation of all those variables. Also see `rowvar` below.

y : array\_like, optional

An additional set of variables and observations. `y` has the same form

as that of `m`.

rowvar : bool, optional

If `rowvar` is True (default), then each row represents a

variable, with observations in the columns. Otherwise, the relationship

is transposed: each column represents a variable, while the rows

contain observations.

bias : bool, optional

Default normalization (False) is by ``(N - 1)``, where ``N`` is the

number of observations given (unbiased estimate). If `bias` is True,

then normalization is by ``N``. These values can be overridden by using

the keyword ``ddof`` in numpy versions >= 1.5.

ddof : int, optional

If not ``None`` the default value implied by `bias` is overridden.

Note that ``ddof=1`` will return the unbiased estimate, even if both

`fweights` and `aweights` are specified, and ``ddof=0`` will return

the simple average. See the notes for the details. The default value

is ``None``.

.. versionadded:: 1.5

fweights : array\_like, int, optional

1-D array of integer frequency weights; the number of times each

observation vector should be repeated.

.. versionadded:: 1.10

aweights : array\_like, optional

1-D array of observation vector weights. These relative weights are

typically large for observations considered "important" and smaller for

observations considered less "important". If ``ddof=0`` the array of

weights can be used to assign probabilities to observation vectors.

.. versionadded:: 1.10

Returns

-------

out : ndarray

The covariance matrix of the variables.

See Also

--------

corrcoef : Normalized covariance matrix

Notes

-----

Assume that the observations are in the columns of the observation

array `m` and let ``f = fweights`` and ``a = aweights`` for brevity. The

steps to compute the weighted covariance are as follows::

>>> m = np.arange(10, dtype=np.float64)

>>> f = np.arange(10) \* 2

>>> a = np.arange(10) \*\* 2.

>>> ddof = 9 # N - 1

>>> w = f \* a

>>> v1 = np.sum(w)

>>> v2 = np.sum(w \* a)

>>> m -= np.sum(m \* w, axis=None, keepdims=True) / v1

>>> cov = np.dot(m \* w, m.T) \* v1 / (v1\*\*2 - ddof \* v2)

Note that when ``a == 1``, the normalization factor

``v1 / (v1\*\*2 - ddof \* v2)`` goes over to ``1 / (np.sum(f) - ddof)``

as it should.

Examples

--------

Consider two variables, :math:`x\_0` and :math:`x\_1`, which

correlate perfectly, but in opposite directions:

>>> x = np.array([[0, 2], [1, 1], [2, 0]]).T

>>> x

array([[0, 1, 2],

[2, 1, 0]])

Note how :math:`x\_0` increases while :math:`x\_1` decreases. The covariance

matrix shows this clearly:

>>> np.cov(x)

array([[ 1., -1.],

[-1., 1.]])

Note that element :math:`C\_{0,1}`, which shows the correlation between

:math:`x\_0` and :math:`x\_1`, is negative.

Further, note how `x` and `y` are combined:

>>> x = [-2.1, -1, 4.3]

>>> y = [3, 1.1, 0.12]

>>> X = np.stack((x, y), axis=0)

>>> np.cov(X)

array([[11.71 , -4.286 ], # may vary

[-4.286 , 2.144133]])

>>> np.cov(x, y)

array([[11.71 , -4.286 ], # may vary

[-4.286 , 2.144133]])

>>> np.cov(x)

array(11.71)

cross(a, b, axisa=-1, axisb=-1, axisc=-1, axis=None)

Return the cross product of two (arrays of) vectors.

The cross product of `a` and `b` in :math:`R^3` is a vector perpendicular

to both `a` and `b`. If `a` and `b` are arrays of vectors, the vectors

are defined by the last axis of `a` and `b` by default, and these axes

can have dimensions 2 or 3. Where the dimension of either `a` or `b` is

2, the third component of the input vector is assumed to be zero and the

cross product calculated accordingly. In cases where both input vectors

have dimension 2, the z-component of the cross product is returned.

Parameters

----------

a : array\_like

Components of the first vector(s).

b : array\_like

Components of the second vector(s).

axisa : int, optional

Axis of `a` that defines the vector(s). By default, the last axis.

axisb : int, optional

Axis of `b` that defines the vector(s). By default, the last axis.

axisc : int, optional

Axis of `c` containing the cross product vector(s). Ignored if

both input vectors have dimension 2, as the return is scalar.

By default, the last axis.

axis : int, optional

If defined, the axis of `a`, `b` and `c` that defines the vector(s)

and cross product(s). Overrides `axisa`, `axisb` and `axisc`.

Returns

-------

c : ndarray

Vector cross product(s).

Raises

------

ValueError

When the dimension of the vector(s) in `a` and/or `b` does not

equal 2 or 3.

See Also

--------

inner : Inner product

outer : Outer product.

ix\_ : Construct index arrays.

Notes

-----

.. versionadded:: 1.9.0

Supports full broadcasting of the inputs.

Examples

--------

Vector cross-product.

>>> x = [1, 2, 3]

>>> y = [4, 5, 6]

>>> np.cross(x, y)

array([-3, 6, -3])

One vector with dimension 2.

>>> x = [1, 2]

>>> y = [4, 5, 6]

>>> np.cross(x, y)

array([12, -6, -3])

Equivalently:

>>> x = [1, 2, 0]

>>> y = [4, 5, 6]

>>> np.cross(x, y)

array([12, -6, -3])

Both vectors with dimension 2.

>>> x = [1,2]

>>> y = [4,5]

>>> np.cross(x, y)

array(-3)

Multiple vector cross-products. Note that the direction of the cross

product vector is defined by the `right-hand rule`.

>>> x = np.array([[1,2,3], [4,5,6]])

>>> y = np.array([[4,5,6], [1,2,3]])

>>> np.cross(x, y)

array([[-3, 6, -3],

[ 3, -6, 3]])

The orientation of `c` can be changed using the `axisc` keyword.

>>> np.cross(x, y, axisc=0)

array([[-3, 3],

[ 6, -6],

[-3, 3]])

Change the vector definition of `x` and `y` using `axisa` and `axisb`.

>>> x = np.array([[1,2,3], [4,5,6], [7, 8, 9]])

>>> y = np.array([[7, 8, 9], [4,5,6], [1,2,3]])

>>> np.cross(x, y)

array([[ -6, 12, -6],

[ 0, 0, 0],

[ 6, -12, 6]])

>>> np.cross(x, y, axisa=0, axisb=0)

array([[-24, 48, -24],

[-30, 60, -30],

[-36, 72, -36]])

cumprod(a, axis=None, dtype=None, out=None)

Return the cumulative product of elements along a given axis.

Parameters

----------

a : array\_like

Input array.

axis : int, optional

Axis along which the cumulative product is computed. By default

the input is flattened.

dtype : dtype, optional

Type of the returned array, as well as of the accumulator in which

the elements are multiplied. If \*dtype\* is not specified, it

defaults to the dtype of `a`, unless `a` has an integer dtype with

a precision less than that of the default platform integer. In

that case, the default platform integer is used instead.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output

but the type of the resulting values will be cast if necessary.

Returns

-------

cumprod : ndarray

A new array holding the result is returned unless `out` is

specified, in which case a reference to out is returned.

See Also

--------

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

Arithmetic is modular when using integer types, and no error is

raised on overflow.

Examples

--------

>>> a = np.array([1,2,3])

>>> np.cumprod(a) # intermediate results 1, 1\*2

... # total product 1\*2\*3 = 6

array([1, 2, 6])

>>> a = np.array([[1, 2, 3], [4, 5, 6]])

>>> np.cumprod(a, dtype=float) # specify type of output

array([ 1., 2., 6., 24., 120., 720.])

The cumulative product for each column (i.e., over the rows) of `a`:

>>> np.cumprod(a, axis=0)

array([[ 1, 2, 3],

[ 4, 10, 18]])

The cumulative product for each row (i.e. over the columns) of `a`:

>>> np.cumprod(a,axis=1)

array([[ 1, 2, 6],

[ 4, 20, 120]])

cumproduct(\*args, \*\*kwargs)

Return the cumulative product over the given axis.

See Also

--------

cumprod : equivalent function; see for details.

cumsum(a, axis=None, dtype=None, out=None)

Return the cumulative sum of the elements along a given axis.

Parameters

----------

a : array\_like

Input array.

axis : int, optional

Axis along which the cumulative sum is computed. The default

(None) is to compute the cumsum over the flattened array.

dtype : dtype, optional

Type of the returned array and of the accumulator in which the

elements are summed. If `dtype` is not specified, it defaults

to the dtype of `a`, unless `a` has an integer dtype with a

precision less than that of the default platform integer. In

that case, the default platform integer is used.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output

but the type will be cast if necessary. See `doc.ufuncs`

(Section "Output arguments") for more details.

Returns

-------

cumsum\_along\_axis : ndarray.

A new array holding the result is returned unless `out` is

specified, in which case a reference to `out` is returned. The

result has the same size as `a`, and the same shape as `a` if

`axis` is not None or `a` is a 1-d array.

See Also

--------

sum : Sum array elements.

trapz : Integration of array values using the composite trapezoidal rule.

diff : Calculate the n-th discrete difference along given axis.

Notes

-----

Arithmetic is modular when using integer types, and no error is

raised on overflow.

Examples

--------

>>> a = np.array([[1,2,3], [4,5,6]])

>>> a

array([[1, 2, 3],

[4, 5, 6]])

>>> np.cumsum(a)

array([ 1, 3, 6, 10, 15, 21])

>>> np.cumsum(a, dtype=float) # specifies type of output value(s)

array([ 1., 3., 6., 10., 15., 21.])

>>> np.cumsum(a,axis=0) # sum over rows for each of the 3 columns

array([[1, 2, 3],

[5, 7, 9]])

>>> np.cumsum(a,axis=1) # sum over columns for each of the 2 rows

array([[ 1, 3, 6],

[ 4, 9, 15]])

datetime\_as\_string(...)

datetime\_as\_string(arr, unit=None, timezone='naive', casting='same\_kind')

Convert an array of datetimes into an array of strings.

Parameters

----------

arr : array\_like of datetime64

The array of UTC timestamps to format.

unit : str

One of None, 'auto', or a :ref:`datetime unit <arrays.dtypes.dateunits>`.

timezone : {'naive', 'UTC', 'local'} or tzinfo

Timezone information to use when displaying the datetime. If 'UTC', end

with a Z to indicate UTC time. If 'local', convert to the local timezone

first, and suffix with a +-#### timezone offset. If a tzinfo object,

then do as with 'local', but use the specified timezone.

casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}

Casting to allow when changing between datetime units.

Returns

-------

str\_arr : ndarray

An array of strings the same shape as `arr`.

Examples

--------

>>> import pytz

>>> d = np.arange('2002-10-27T04:30', 4\*60, 60, dtype='M8[m]')

>>> d

array(['2002-10-27T04:30', '2002-10-27T05:30', '2002-10-27T06:30',

'2002-10-27T07:30'], dtype='datetime64[m]')

Setting the timezone to UTC shows the same information, but with a Z suffix

>>> np.datetime\_as\_string(d, timezone='UTC')

array(['2002-10-27T04:30Z', '2002-10-27T05:30Z', '2002-10-27T06:30Z',

'2002-10-27T07:30Z'], dtype='<U35')

Note that we picked datetimes that cross a DST boundary. Passing in a

``pytz`` timezone object will print the appropriate offset

>>> np.datetime\_as\_string(d, timezone=pytz.timezone('US/Eastern'))

array(['2002-10-27T00:30-0400', '2002-10-27T01:30-0400',

'2002-10-27T01:30-0500', '2002-10-27T02:30-0500'], dtype='<U39')

Passing in a unit will change the precision

>>> np.datetime\_as\_string(d, unit='h')

array(['2002-10-27T04', '2002-10-27T05', '2002-10-27T06', '2002-10-27T07'],

dtype='<U32')

>>> np.datetime\_as\_string(d, unit='s')

array(['2002-10-27T04:30:00', '2002-10-27T05:30:00', '2002-10-27T06:30:00',

'2002-10-27T07:30:00'], dtype='<U38')

'casting' can be used to specify whether precision can be changed

>>> np.datetime\_as\_string(d, unit='h', casting='safe')

Traceback (most recent call last):

...

TypeError: Cannot create a datetime string as units 'h' from a NumPy

datetime with units 'm' according to the rule 'safe'

datetime\_data(...)

datetime\_data(dtype, /)

Get information about the step size of a date or time type.

The returned tuple can be passed as the second argument of `numpy.datetime64` and

`numpy.timedelta64`.

Parameters

----------

dtype : dtype

The dtype object, which must be a `datetime64` or `timedelta64` type.

Returns

-------

unit : str

The :ref:`datetime unit <arrays.dtypes.dateunits>` on which this dtype

is based.

count : int

The number of base units in a step.

Examples

--------

>>> dt\_25s = np.dtype('timedelta64[25s]')

>>> np.datetime\_data(dt\_25s)

('s', 25)

>>> np.array(10, dt\_25s).astype('timedelta64[s]')

array(250, dtype='timedelta64[s]')

The result can be used to construct a datetime that uses the same units

as a timedelta

>>> np.datetime64('2010', np.datetime\_data(dt\_25s))

numpy.datetime64('2010-01-01T00:00:00','25s')

delete(arr, obj, axis=None)

Return a new array with sub-arrays along an axis deleted. For a one

dimensional array, this returns those entries not returned by

`arr[obj]`.

Parameters

----------

arr : array\_like

Input array.

obj : slice, int or array of ints

Indicate indices of sub-arrays to remove along the specified axis.

axis : int, optional

The axis along which to delete the subarray defined by `obj`.

If `axis` is None, `obj` is applied to the flattened array.

Returns

-------

out : ndarray

A copy of `arr` with the elements specified by `obj` removed. Note

that `delete` does not occur in-place. If `axis` is None, `out` is

a flattened array.

See Also

--------

insert : Insert elements into an array.

append : Append elements at the end of an array.

Notes

-----

Often it is preferable to use a boolean mask. For example:

>>> arr = np.arange(12) + 1

>>> mask = np.ones(len(arr), dtype=bool)

>>> mask[[0,2,4]] = False

>>> result = arr[mask,...]

Is equivalent to `np.delete(arr, [0,2,4], axis=0)`, but allows further

use of `mask`.

Examples

--------

>>> arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

>>> arr

array([[ 1, 2, 3, 4],

[ 5, 6, 7, 8],

[ 9, 10, 11, 12]])

>>> np.delete(arr, 1, 0)

array([[ 1, 2, 3, 4],

[ 9, 10, 11, 12]])

>>> np.delete(arr, np.s\_[::2], 1)

array([[ 2, 4],

[ 6, 8],

[10, 12]])

>>> np.delete(arr, [1,3,5], None)

array([ 1, 3, 5, 7, 8, 9, 10, 11, 12])

deprecate(\*args, \*\*kwargs)

Issues a DeprecationWarning, adds warning to `old\_name`'s

docstring, rebinds ``old\_name.\_\_name\_\_`` and returns the new

function object.

This function may also be used as a decorator.

Parameters

----------

func : function

The function to be deprecated.

old\_name : str, optional

The name of the function to be deprecated. Default is None, in

which case the name of `func` is used.

new\_name : str, optional

The new name for the function. Default is None, in which case the

deprecation message is that `old\_name` is deprecated. If given, the

deprecation message is that `old\_name` is deprecated and `new\_name`

should be used instead.

message : str, optional

Additional explanation of the deprecation. Displayed in the

docstring after the warning.

Returns

-------

old\_func : function

The deprecated function.

Examples

--------

Note that ``olduint`` returns a value after printing Deprecation

Warning:

>>> olduint = np.deprecate(np.uint)

DeprecationWarning: `uint64` is deprecated! # may vary

>>> olduint(6)

6

deprecate\_with\_doc lambda msg

diag(v, k=0)

Extract a diagonal or construct a diagonal array.

See the more detailed documentation for ``numpy.diagonal`` if you use this

function to extract a diagonal and wish to write to the resulting array;

whether it returns a copy or a view depends on what version of numpy you

are using.

Parameters

----------

v : array\_like

If `v` is a 2-D array, return a copy of its `k`-th diagonal.

If `v` is a 1-D array, return a 2-D array with `v` on the `k`-th

diagonal.

k : int, optional

Diagonal in question. The default is 0. Use `k>0` for diagonals

above the main diagonal, and `k<0` for diagonals below the main

diagonal.

Returns

-------

out : ndarray

The extracted diagonal or constructed diagonal array.

See Also

--------

diagonal : Return specified diagonals.

diagflat : Create a 2-D array with the flattened input as a diagonal.

trace : Sum along diagonals.

triu : Upper triangle of an array.

tril : Lower triangle of an array.

Examples

--------

>>> x = np.arange(9).reshape((3,3))

>>> x

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

>>> np.diag(x)

array([0, 4, 8])

>>> np.diag(x, k=1)

array([1, 5])

>>> np.diag(x, k=-1)

array([3, 7])

>>> np.diag(np.diag(x))

array([[0, 0, 0],

[0, 4, 0],

[0, 0, 8]])

diag\_indices(n, ndim=2)

Return the indices to access the main diagonal of an array.

This returns a tuple of indices that can be used to access the main

diagonal of an array `a` with ``a.ndim >= 2`` dimensions and shape

(n, n, ..., n). For ``a.ndim = 2`` this is the usual diagonal, for

``a.ndim > 2`` this is the set of indices to access ``a[i, i, ..., i]``

for ``i = [0..n-1]``.

Parameters

----------

n : int

The size, along each dimension, of the arrays for which the returned

indices can be used.

ndim : int, optional

The number of dimensions.

See also

--------

diag\_indices\_from

Notes

-----

.. versionadded:: 1.4.0

Examples

--------

Create a set of indices to access the diagonal of a (4, 4) array:

>>> di = np.diag\_indices(4)

>>> di

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

>>> a = np.arange(16).reshape(4, 4)

>>> a

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

>>> a[di] = 100

>>> a

array([[100, 1, 2, 3],

[ 4, 100, 6, 7],

[ 8, 9, 100, 11],

[ 12, 13, 14, 100]])

Now, we create indices to manipulate a 3-D array:

>>> d3 = np.diag\_indices(2, 3)

>>> d3

(array([0, 1]), array([0, 1]), array([0, 1]))

And use it to set the diagonal of an array of zeros to 1:

>>> a = np.zeros((2, 2, 2), dtype=int)

>>> a[d3] = 1

>>> a

array([[[1, 0],

[0, 0]],

[[0, 0],

[0, 1]]])

diag\_indices\_from(arr)

Return the indices to access the main diagonal of an n-dimensional array.

See `diag\_indices` for full details.

Parameters

----------

arr : array, at least 2-D

See Also

--------

diag\_indices

Notes

-----

.. versionadded:: 1.4.0

diagflat(v, k=0)

Create a two-dimensional array with the flattened input as a diagonal.

Parameters

----------

v : array\_like

Input data, which is flattened and set as the `k`-th

diagonal of the output.

k : int, optional

Diagonal to set; 0, the default, corresponds to the "main" diagonal,

a positive (negative) `k` giving the number of the diagonal above

(below) the main.

Returns

-------

out : ndarray

The 2-D output array.

See Also

--------

diag : MATLAB work-alike for 1-D and 2-D arrays.

diagonal : Return specified diagonals.

trace : Sum along diagonals.

Examples

--------

>>> np.diagflat([[1,2], [3,4]])

array([[1, 0, 0, 0],

[0, 2, 0, 0],

[0, 0, 3, 0],

[0, 0, 0, 4]])

>>> np.diagflat([1,2], 1)

array([[0, 1, 0],

[0, 0, 2],

[0, 0, 0]])

diagonal(a, offset=0, axis1=0, axis2=1)

Return specified diagonals.

If `a` is 2-D, returns the diagonal of `a` with the given offset,

i.e., the collection of elements of the form ``a[i, i+offset]``. If

`a` has more than two dimensions, then the axes specified by `axis1`

and `axis2` are used to determine the 2-D sub-array whose diagonal is

returned. The shape of the resulting array can be determined by

removing `axis1` and `axis2` and appending an index to the right equal

to the size of the resulting diagonals.

In versions of NumPy prior to 1.7, this function always returned a new,

independent array containing a copy of the values in the diagonal.

In NumPy 1.7 and 1.8, it continues to return a copy of the diagonal,

but depending on this fact is deprecated. Writing to the resulting

array continues to work as it used to, but a FutureWarning is issued.

Starting in NumPy 1.9 it returns a read-only view on the original array.

Attempting to write to the resulting array will produce an error.

In some future release, it will return a read/write view and writing to

the returned array will alter your original array. The returned array

will have the same type as the input array.

If you don't write to the array returned by this function, then you can

just ignore all of the above.

If you depend on the current behavior, then we suggest copying the

returned array explicitly, i.e., use ``np.diagonal(a).copy()`` instead

of just ``np.diagonal(a)``. This will work with both past and future

versions of NumPy.

Parameters

----------

a : array\_like

Array from which the diagonals are taken.

offset : int, optional

Offset of the diagonal from the main diagonal. Can be positive or

negative. Defaults to main diagonal (0).

axis1 : int, optional

Axis to be used as the first axis of the 2-D sub-arrays from which

the diagonals should be taken. Defaults to first axis (0).

axis2 : int, optional

Axis to be used as the second axis of the 2-D sub-arrays from

which the diagonals should be taken. Defaults to second axis (1).

Returns

-------

array\_of\_diagonals : ndarray

If `a` is 2-D, then a 1-D array containing the diagonal and of the

same type as `a` is returned unless `a` is a `matrix`, in which case

a 1-D array rather than a (2-D) `matrix` is returned in order to

maintain backward compatibility.

If ``a.ndim > 2``, then the dimensions specified by `axis1` and `axis2`

are removed, and a new axis inserted at the end corresponding to the

diagonal.

Raises

------

ValueError

If the dimension of `a` is less than 2.

See Also

--------

diag : MATLAB work-a-like for 1-D and 2-D arrays.

diagflat : Create diagonal arrays.

trace : Sum along diagonals.

Examples

--------

>>> a = np.arange(4).reshape(2,2)

>>> a

array([[0, 1],

[2, 3]])

>>> a.diagonal()

array([0, 3])

>>> a.diagonal(1)

array([1])

A 3-D example:

>>> a = np.arange(8).reshape(2,2,2); a

array([[[0, 1],

[2, 3]],

[[4, 5],

[6, 7]]])

>>> a.diagonal(0, # Main diagonals of two arrays created by skipping

... 0, # across the outer(left)-most axis last and

... 1) # the "middle" (row) axis first.

array([[0, 6],

[1, 7]])

The sub-arrays whose main diagonals we just obtained; note that each

corresponds to fixing the right-most (column) axis, and that the

diagonals are "packed" in rows.

>>> a[:,:,0] # main diagonal is [0 6]

array([[0, 2],

[4, 6]])

>>> a[:,:,1] # main diagonal is [1 7]

array([[1, 3],

[5, 7]])

The anti-diagonal can be obtained by reversing the order of elements

using either `numpy.flipud` or `numpy.fliplr`.

>>> a = np.arange(9).reshape(3, 3)

>>> a

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

>>> np.fliplr(a).diagonal() # Horizontal flip

array([2, 4, 6])

>>> np.flipud(a).diagonal() # Vertical flip

array([6, 4, 2])

Note that the order in which the diagonal is retrieved varies depending

on the flip function.

diff(a, n=1, axis=-1, prepend=<no value>, append=<no value>)

Calculate the n-th discrete difference along the given axis.

The first difference is given by ``out[i] = a[i+1] - a[i]`` along

the given axis, higher differences are calculated by using `diff`

recursively.

Parameters

----------

a : array\_like

Input array

n : int, optional

The number of times values are differenced. If zero, the input

is returned as-is.

axis : int, optional

The axis along which the difference is taken, default is the

last axis.

prepend, append : array\_like, optional

Values to prepend or append to "a" along axis prior to

performing the difference. Scalar values are expanded to

arrays with length 1 in the direction of axis and the shape

of the input array in along all other axes. Otherwise the

dimension and shape must match "a" except along axis.

Returns

-------

diff : ndarray

The n-th differences. The shape of the output is the same as `a`

except along `axis` where the dimension is smaller by `n`. The

type of the output is the same as the type of the difference

between any two elements of `a`. This is the same as the type of

`a` in most cases. A notable exception is `datetime64`, which

results in a `timedelta64` output array.

See Also

--------

gradient, ediff1d, cumsum

Notes

-----

Type is preserved for boolean arrays, so the result will contain

`False` when consecutive elements are the same and `True` when they

differ.

For unsigned integer arrays, the results will also be unsigned. This

should not be surprising, as the result is consistent with

calculating the difference directly:

>>> u8\_arr = np.array([1, 0], dtype=np.uint8)

>>> np.diff(u8\_arr)

array([255], dtype=uint8)

>>> u8\_arr[1,...] - u8\_arr[0,...]

255

If this is not desirable, then the array should be cast to a larger

integer type first:

>>> i16\_arr = u8\_arr.astype(np.int16)

>>> np.diff(i16\_arr)

array([-1], dtype=int16)

Examples

--------

>>> x = np.array([1, 2, 4, 7, 0])

>>> np.diff(x)

array([ 1, 2, 3, -7])

>>> np.diff(x, n=2)

array([ 1, 1, -10])

>>> x = np.array([[1, 3, 6, 10], [0, 5, 6, 8]])

>>> np.diff(x)

array([[2, 3, 4],

[5, 1, 2]])

>>> np.diff(x, axis=0)

array([[-1, 2, 0, -2]])

>>> x = np.arange('1066-10-13', '1066-10-16', dtype=np.datetime64)

>>> np.diff(x)

array([1, 1], dtype='timedelta64[D]')

digitize(x, bins, right=False)

Return the indices of the bins to which each value in input array belongs.

========= ============= ============================

`right` order of bins returned index `i` satisfies

========= ============= ============================

``False`` increasing ``bins[i-1] <= x < bins[i]``

``True`` increasing ``bins[i-1] < x <= bins[i]``

``False`` decreasing ``bins[i-1] > x >= bins[i]``

``True`` decreasing ``bins[i-1] >= x > bins[i]``

========= ============= ============================

If values in `x` are beyond the bounds of `bins`, 0 or ``len(bins)`` is

returned as appropriate.

Parameters

----------

x : array\_like

Input array to be binned. Prior to NumPy 1.10.0, this array had to

be 1-dimensional, but can now have any shape.

bins : array\_like

Array of bins. It has to be 1-dimensional and monotonic.

right : bool, optional

Indicating whether the intervals include the right or the left bin

edge. Default behavior is (right==False) indicating that the interval

does not include the right edge. The left bin end is open in this

case, i.e., bins[i-1] <= x < bins[i] is the default behavior for

monotonically increasing bins.

Returns

-------

indices : ndarray of ints

Output array of indices, of same shape as `x`.

Raises

------

ValueError

If `bins` is not monotonic.

TypeError

If the type of the input is complex.

See Also

--------

bincount, histogram, unique, searchsorted

Notes

-----

If values in `x` are such that they fall outside the bin range,

attempting to index `bins` with the indices that `digitize` returns

will result in an IndexError.

.. versionadded:: 1.10.0

`np.digitize` is implemented in terms of `np.searchsorted`. This means

that a binary search is used to bin the values, which scales much better

for larger number of bins than the previous linear search. It also removes

the requirement for the input array to be 1-dimensional.

For monotonically \_increasing\_ `bins`, the following are equivalent::

np.digitize(x, bins, right=True)

np.searchsorted(bins, x, side='left')

Note that as the order of the arguments are reversed, the side must be too.

The `searchsorted` call is marginally faster, as it does not do any

monotonicity checks. Perhaps more importantly, it supports all dtypes.

Examples

--------

>>> x = np.array([0.2, 6.4, 3.0, 1.6])

>>> bins = np.array([0.0, 1.0, 2.5, 4.0, 10.0])

>>> inds = np.digitize(x, bins)

>>> inds

array([1, 4, 3, 2])

>>> for n in range(x.size):

... print(bins[inds[n]-1], "<=", x[n], "<", bins[inds[n]])

...

0.0 <= 0.2 < 1.0

4.0 <= 6.4 < 10.0

2.5 <= 3.0 < 4.0

1.0 <= 1.6 < 2.5

>>> x = np.array([1.2, 10.0, 12.4, 15.5, 20.])

>>> bins = np.array([0, 5, 10, 15, 20])

>>> np.digitize(x,bins,right=True)

array([1, 2, 3, 4, 4])

>>> np.digitize(x,bins,right=False)

array([1, 3, 3, 4, 5])

disp(mesg, device=None, linefeed=True)

Display a message on a device.

Parameters

----------

mesg : str

Message to display.

device : object

Device to write message. If None, defaults to ``sys.stdout`` which is

very similar to ``print``. `device` needs to have ``write()`` and

``flush()`` methods.

linefeed : bool, optional

Option whether to print a line feed or not. Defaults to True.

Raises

------

AttributeError

If `device` does not have a ``write()`` or ``flush()`` method.

Examples

--------

Besides ``sys.stdout``, a file-like object can also be used as it has

both required methods:

>>> from io import StringIO

>>> buf = StringIO()

>>> np.disp(u'"Display" in a file', device=buf)

>>> buf.getvalue()

'"Display" in a file\n'

dot(...)

dot(a, b, out=None)

Dot product of two arrays. Specifically,

- If both `a` and `b` are 1-D arrays, it is inner product of vectors

(without complex conjugation).

- If both `a` and `b` are 2-D arrays, it is matrix multiplication,

but using :func:`matmul` or ``a @ b`` is preferred.

- If either `a` or `b` is 0-D (scalar), it is equivalent to :func:`multiply`

and using ``numpy.multiply(a, b)`` or ``a \* b`` is preferred.

- If `a` is an N-D array and `b` is a 1-D array, it is a sum product over

the last axis of `a` and `b`.

- If `a` is an N-D array and `b` is an M-D array (where ``M>=2``), it is a

sum product over the last axis of `a` and the second-to-last axis of `b`::

dot(a, b)[i,j,k,m] = sum(a[i,j,:] \* b[k,:,m])

Parameters

----------

a : array\_like

First argument.

b : array\_like

Second argument.

out : ndarray, optional

Output argument. This must have the exact kind that would be returned

if it was not used. In particular, it must have the right type, must be

C-contiguous, and its dtype must be the dtype that would be returned

for `dot(a,b)`. This is a performance feature. Therefore, if these

conditions are not met, an exception is raised, instead of attempting

to be flexible.

Returns

-------

output : ndarray

Returns the dot product of `a` and `b`. If `a` and `b` are both

scalars or both 1-D arrays then a scalar is returned; otherwise

an array is returned.

If `out` is given, then it is returned.

Raises

------

ValueError

If the last dimension of `a` is not the same size as

the second-to-last dimension of `b`.

See Also

--------

vdot : Complex-conjugating dot product.

tensordot : Sum products over arbitrary axes.

einsum : Einstein summation convention.

matmul : '@' operator as method with out parameter.

Examples

--------

>>> np.dot(3, 4)

12

Neither argument is complex-conjugated:

>>> np.dot([2j, 3j], [2j, 3j])

(-13+0j)

For 2-D arrays it is the matrix product:

>>> a = [[1, 0], [0, 1]]

>>> b = [[4, 1], [2, 2]]

>>> np.dot(a, b)

array([[4, 1],

[2, 2]])

>>> a = np.arange(3\*4\*5\*6).reshape((3,4,5,6))

>>> b = np.arange(3\*4\*5\*6)[::-1].reshape((5,4,6,3))

>>> np.dot(a, b)[2,3,2,1,2,2]

499128

>>> sum(a[2,3,2,:] \* b[1,2,:,2])

499128

dsplit(ary, indices\_or\_sections)

Split array into multiple sub-arrays along the 3rd axis (depth).

Please refer to the `split` documentation. `dsplit` is equivalent

to `split` with ``axis=2``, the array is always split along the third

axis provided the array dimension is greater than or equal to 3.

See Also

--------

split : Split an array into multiple sub-arrays of equal size.

Examples

--------

>>> x = np.arange(16.0).reshape(2, 2, 4)

>>> x

array([[[ 0., 1., 2., 3.],

[ 4., 5., 6., 7.]],

[[ 8., 9., 10., 11.],

[12., 13., 14., 15.]]])

>>> np.dsplit(x, 2)

[array([[[ 0., 1.],

[ 4., 5.]],

[[ 8., 9.],

[12., 13.]]]), array([[[ 2., 3.],

[ 6., 7.]],

[[10., 11.],

[14., 15.]]])]

>>> np.dsplit(x, np.array([3, 6]))

[array([[[ 0., 1., 2.],

[ 4., 5., 6.]],

[[ 8., 9., 10.],

[12., 13., 14.]]]),

array([[[ 3.],

[ 7.]],

[[11.],

[15.]]]),

array([], shape=(2, 2, 0), dtype=float64)]

dstack(tup)

Stack arrays in sequence depth wise (along third axis).

This is equivalent to concatenation along the third axis after 2-D arrays

of shape `(M,N)` have been reshaped to `(M,N,1)` and 1-D arrays of shape

`(N,)` have been reshaped to `(1,N,1)`. Rebuilds arrays divided by

`dsplit`.

This function makes most sense for arrays with up to 3 dimensions. For

instance, for pixel-data with a height (first axis), width (second axis),

and r/g/b channels (third axis). The functions `concatenate`, `stack` and

`block` provide more general stacking and concatenation operations.

Parameters

----------

tup : sequence of arrays

The arrays must have the same shape along all but the third axis.

1-D or 2-D arrays must have the same shape.

Returns

-------

stacked : ndarray

The array formed by stacking the given arrays, will be at least 3-D.

See Also

--------

stack : Join a sequence of arrays along a new axis.

vstack : Stack along first axis.

hstack : Stack along second axis.

concatenate : Join a sequence of arrays along an existing axis.

dsplit : Split array along third axis.

Examples

--------

>>> a = np.array((1,2,3))

>>> b = np.array((2,3,4))

>>> np.dstack((a,b))

array([[[1, 2],

[2, 3],

[3, 4]]])

>>> a = np.array([[1],[2],[3]])

>>> b = np.array([[2],[3],[4]])

>>> np.dstack((a,b))

array([[[1, 2]],

[[2, 3]],

[[3, 4]]])

ediff1d(ary, to\_end=None, to\_begin=None)

The differences between consecutive elements of an array.

Parameters

----------

ary : array\_like

If necessary, will be flattened before the differences are taken.

to\_end : array\_like, optional

Number(s) to append at the end of the returned differences.

to\_begin : array\_like, optional

Number(s) to prepend at the beginning of the returned differences.

Returns

-------

ediff1d : ndarray

The differences. Loosely, this is ``ary.flat[1:] - ary.flat[:-1]``.

See Also

--------

diff, gradient

Notes

-----

When applied to masked arrays, this function drops the mask information

if the `to\_begin` and/or `to\_end` parameters are used.

Examples

--------

>>> x = np.array([1, 2, 4, 7, 0])

>>> np.ediff1d(x)

array([ 1, 2, 3, -7])

>>> np.ediff1d(x, to\_begin=-99, to\_end=np.array([88, 99]))

array([-99, 1, 2, ..., -7, 88, 99])

The returned array is always 1D.

>>> y = [[1, 2, 4], [1, 6, 24]]

>>> np.ediff1d(y)

array([ 1, 2, -3, 5, 18])

einsum(\*operands, \*\*kwargs)

einsum(subscripts, \*operands, out=None, dtype=None, order='K',

casting='safe', optimize=False)

Evaluates the Einstein summation convention on the operands.

Using the Einstein summation convention, many common multi-dimensional,

linear algebraic array operations can be represented in a simple fashion.

In \*implicit\* mode `einsum` computes these values.

In \*explicit\* mode, `einsum` provides further flexibility to compute

other array operations that might not be considered classical Einstein

summation operations, by disabling, or forcing summation over specified

subscript labels.

See the notes and examples for clarification.

Parameters

----------

subscripts : str

Specifies the subscripts for summation as comma separated list of

subscript labels. An implicit (classical Einstein summation)

calculation is performed unless the explicit indicator '->' is

included as well as subscript labels of the precise output form.

operands : list of array\_like

These are the arrays for the operation.

out : ndarray, optional

If provided, the calculation is done into this array.

dtype : {data-type, None}, optional

If provided, forces the calculation to use the data type specified.

Note that you may have to also give a more liberal `casting`

parameter to allow the conversions. Default is None.

order : {'C', 'F', 'A', 'K'}, optional

Controls the memory layout of the output. 'C' means it should

be C contiguous. 'F' means it should be Fortran contiguous,

'A' means it should be 'F' if the inputs are all 'F', 'C' otherwise.

'K' means it should be as close to the layout as the inputs as

is possible, including arbitrarily permuted axes.

Default is 'K'.

casting : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, optional

Controls what kind of data casting may occur. Setting this to

'unsafe' is not recommended, as it can adversely affect accumulations.

\* 'no' means the data types should not be cast at all.

\* 'equiv' means only byte-order changes are allowed.

\* 'safe' means only casts which can preserve values are allowed.

\* 'same\_kind' means only safe casts or casts within a kind,

like float64 to float32, are allowed.

\* 'unsafe' means any data conversions may be done.

Default is 'safe'.

optimize : {False, True, 'greedy', 'optimal'}, optional

Controls if intermediate optimization should occur. No optimization

will occur if False and True will default to the 'greedy' algorithm.

Also accepts an explicit contraction list from the ``np.einsum\_path``

function. See ``np.einsum\_path`` for more details. Defaults to False.

Returns

-------

output : ndarray

The calculation based on the Einstein summation convention.

See Also

--------

einsum\_path, dot, inner, outer, tensordot, linalg.multi\_dot

Notes

-----

.. versionadded:: 1.6.0

The Einstein summation convention can be used to compute

many multi-dimensional, linear algebraic array operations. `einsum`

provides a succinct way of representing these.

A non-exhaustive list of these operations,

which can be computed by `einsum`, is shown below along with examples:

\* Trace of an array, :py:func:`numpy.trace`.

\* Return a diagonal, :py:func:`numpy.diag`.

\* Array axis summations, :py:func:`numpy.sum`.

\* Transpositions and permutations, :py:func:`numpy.transpose`.

\* Matrix multiplication and dot product, :py:func:`numpy.matmul` :py:func:`numpy.dot`.

\* Vector inner and outer products, :py:func:`numpy.inner` :py:func:`numpy.outer`.

\* Broadcasting, element-wise and scalar multiplication, :py:func:`numpy.multiply`.

\* Tensor contractions, :py:func:`numpy.tensordot`.

\* Chained array operations, in efficient calculation order, :py:func:`numpy.einsum\_path`.

The subscripts string is a comma-separated list of subscript labels,

where each label refers to a dimension of the corresponding operand.

Whenever a label is repeated it is summed, so ``np.einsum('i,i', a, b)``

is equivalent to :py:func:`np.inner(a,b) <numpy.inner>`. If a label

appears only once, it is not summed, so ``np.einsum('i', a)`` produces a

view of ``a`` with no changes. A further example ``np.einsum('ij,jk', a, b)``

describes traditional matrix multiplication and is equivalent to

:py:func:`np.matmul(a,b) <numpy.matmul>`. Repeated subscript labels in one

operand take the diagonal. For example, ``np.einsum('ii', a)`` is equivalent

to :py:func:`np.trace(a) <numpy.trace>`.

In \*implicit mode\*, the chosen subscripts are important

since the axes of the output are reordered alphabetically. This

means that ``np.einsum('ij', a)`` doesn't affect a 2D array, while

``np.einsum('ji', a)`` takes its transpose. Additionally,

``np.einsum('ij,jk', a, b)`` returns a matrix multiplication, while,

``np.einsum('ij,jh', a, b)`` returns the transpose of the

multiplication since subscript 'h' precedes subscript 'i'.

In \*explicit mode\* the output can be directly controlled by

specifying output subscript labels. This requires the

identifier '->' as well as the list of output subscript labels.

This feature increases the flexibility of the function since

summing can be disabled or forced when required. The call

``np.einsum('i->', a)`` is like :py:func:`np.sum(a, axis=-1) <numpy.sum>`,

and ``np.einsum('ii->i', a)`` is like :py:func:`np.diag(a) <numpy.diag>`.

The difference is that `einsum` does not allow broadcasting by default.

Additionally ``np.einsum('ij,jh->ih', a, b)`` directly specifies the

order of the output subscript labels and therefore returns matrix

multiplication, unlike the example above in implicit mode.

To enable and control broadcasting, use an ellipsis. Default

NumPy-style broadcasting is done by adding an ellipsis

to the left of each term, like ``np.einsum('...ii->...i', a)``.

To take the trace along the first and last axes,

you can do ``np.einsum('i...i', a)``, or to do a matrix-matrix

product with the left-most indices instead of rightmost, one can do

``np.einsum('ij...,jk...->ik...', a, b)``.

When there is only one operand, no axes are summed, and no output

parameter is provided, a view into the operand is returned instead

of a new array. Thus, taking the diagonal as ``np.einsum('ii->i', a)``

produces a view (changed in version 1.10.0).

`einsum` also provides an alternative way to provide the subscripts

and operands as ``einsum(op0, sublist0, op1, sublist1, ..., [sublistout])``.

If the output shape is not provided in this format `einsum` will be

calculated in implicit mode, otherwise it will be performed explicitly.

The examples below have corresponding `einsum` calls with the two

parameter methods.

.. versionadded:: 1.10.0

Views returned from einsum are now writeable whenever the input array

is writeable. For example, ``np.einsum('ijk...->kji...', a)`` will now

have the same effect as :py:func:`np.swapaxes(a, 0, 2) <numpy.swapaxes>`

and ``np.einsum('ii->i', a)`` will return a writeable view of the diagonal

of a 2D array.

.. versionadded:: 1.12.0

Added the ``optimize`` argument which will optimize the contraction order

of an einsum expression. For a contraction with three or more operands this

can greatly increase the computational efficiency at the cost of a larger

memory footprint during computation.

Typically a 'greedy' algorithm is applied which empirical tests have shown

returns the optimal path in the majority of cases. In some cases 'optimal'

will return the superlative path through a more expensive, exhaustive search.

For iterative calculations it may be advisable to calculate the optimal path

once and reuse that path by supplying it as an argument. An example is given

below.

See :py:func:`numpy.einsum\_path` for more details.

Examples

--------

>>> a = np.arange(25).reshape(5,5)

>>> b = np.arange(5)

>>> c = np.arange(6).reshape(2,3)

Trace of a matrix:

>>> np.einsum('ii', a)

60

>>> np.einsum(a, [0,0])

60

>>> np.trace(a)

60

Extract the diagonal (requires explicit form):

>>> np.einsum('ii->i', a)

array([ 0, 6, 12, 18, 24])

>>> np.einsum(a, [0,0], [0])

array([ 0, 6, 12, 18, 24])

>>> np.diag(a)

array([ 0, 6, 12, 18, 24])

Sum over an axis (requires explicit form):

>>> np.einsum('ij->i', a)

array([ 10, 35, 60, 85, 110])

>>> np.einsum(a, [0,1], [0])

array([ 10, 35, 60, 85, 110])

>>> np.sum(a, axis=1)

array([ 10, 35, 60, 85, 110])

For higher dimensional arrays summing a single axis can be done with ellipsis:

>>> np.einsum('...j->...', a)

array([ 10, 35, 60, 85, 110])

>>> np.einsum(a, [Ellipsis,1], [Ellipsis])

array([ 10, 35, 60, 85, 110])

Compute a matrix transpose, or reorder any number of axes:

>>> np.einsum('ji', c)

array([[0, 3],

[1, 4],

[2, 5]])

>>> np.einsum('ij->ji', c)

array([[0, 3],

[1, 4],

[2, 5]])

>>> np.einsum(c, [1,0])

array([[0, 3],

[1, 4],

[2, 5]])

>>> np.transpose(c)

array([[0, 3],

[1, 4],

[2, 5]])

Vector inner products:

>>> np.einsum('i,i', b, b)

30

>>> np.einsum(b, [0], b, [0])

30

>>> np.inner(b,b)

30

Matrix vector multiplication:

>>> np.einsum('ij,j', a, b)

array([ 30, 80, 130, 180, 230])

>>> np.einsum(a, [0,1], b, [1])

array([ 30, 80, 130, 180, 230])

>>> np.dot(a, b)

array([ 30, 80, 130, 180, 230])

>>> np.einsum('...j,j', a, b)

array([ 30, 80, 130, 180, 230])

Broadcasting and scalar multiplication:

>>> np.einsum('..., ...', 3, c)

array([[ 0, 3, 6],

[ 9, 12, 15]])

>>> np.einsum(',ij', 3, c)

array([[ 0, 3, 6],

[ 9, 12, 15]])

>>> np.einsum(3, [Ellipsis], c, [Ellipsis])

array([[ 0, 3, 6],

[ 9, 12, 15]])

>>> np.multiply(3, c)

array([[ 0, 3, 6],

[ 9, 12, 15]])

Vector outer product:

>>> np.einsum('i,j', np.arange(2)+1, b)

array([[0, 1, 2, 3, 4],

[0, 2, 4, 6, 8]])

>>> np.einsum(np.arange(2)+1, [0], b, [1])

array([[0, 1, 2, 3, 4],

[0, 2, 4, 6, 8]])

>>> np.outer(np.arange(2)+1, b)

array([[0, 1, 2, 3, 4],

[0, 2, 4, 6, 8]])

Tensor contraction:

>>> a = np.arange(60.).reshape(3,4,5)

>>> b = np.arange(24.).reshape(4,3,2)

>>> np.einsum('ijk,jil->kl', a, b)

array([[4400., 4730.],

[4532., 4874.],

[4664., 5018.],

[4796., 5162.],

[4928., 5306.]])

>>> np.einsum(a, [0,1,2], b, [1,0,3], [2,3])

array([[4400., 4730.],

[4532., 4874.],

[4664., 5018.],

[4796., 5162.],

[4928., 5306.]])

>>> np.tensordot(a,b, axes=([1,0],[0,1]))

array([[4400., 4730.],

[4532., 4874.],

[4664., 5018.],

[4796., 5162.],

[4928., 5306.]])

Writeable returned arrays (since version 1.10.0):

>>> a = np.zeros((3, 3))

>>> np.einsum('ii->i', a)[:] = 1

>>> a

array([[1., 0., 0.],

[0., 1., 0.],

[0., 0., 1.]])

Example of ellipsis use:

>>> a = np.arange(6).reshape((3,2))

>>> b = np.arange(12).reshape((4,3))

>>> np.einsum('ki,jk->ij', a, b)

array([[10, 28, 46, 64],

[13, 40, 67, 94]])

>>> np.einsum('ki,...k->i...', a, b)

array([[10, 28, 46, 64],

[13, 40, 67, 94]])

>>> np.einsum('k...,jk', a, b)

array([[10, 28, 46, 64],

[13, 40, 67, 94]])

Chained array operations. For more complicated contractions, speed ups

might be achieved by repeatedly computing a 'greedy' path or pre-computing the

'optimal' path and repeatedly applying it, using an

`einsum\_path` insertion (since version 1.12.0). Performance improvements can be

particularly significant with larger arrays:

>>> a = np.ones(64).reshape(2,4,8)

Basic `einsum`: ~1520ms (benchmarked on 3.1GHz Intel i5.)

>>> for iteration in range(500):

... \_ = np.einsum('ijk,ilm,njm,nlk,abc->',a,a,a,a,a)

Sub-optimal `einsum` (due to repeated path calculation time): ~330ms

>>> for iteration in range(500):

... \_ = np.einsum('ijk,ilm,njm,nlk,abc->',a,a,a,a,a, optimize='optimal')

Greedy `einsum` (faster optimal path approximation): ~160ms

>>> for iteration in range(500):

... \_ = np.einsum('ijk,ilm,njm,nlk,abc->',a,a,a,a,a, optimize='greedy')

Optimal `einsum` (best usage pattern in some use cases): ~110ms

>>> path = np.einsum\_path('ijk,ilm,njm,nlk,abc->',a,a,a,a,a, optimize='optimal')[0]

>>> for iteration in range(500):

... \_ = np.einsum('ijk,ilm,njm,nlk,abc->',a,a,a,a,a, optimize=path)

einsum\_path(\*operands, \*\*kwargs)

einsum\_path(subscripts, \*operands, optimize='greedy')

Evaluates the lowest cost contraction order for an einsum expression by

considering the creation of intermediate arrays.

Parameters

----------

subscripts : str

Specifies the subscripts for summation.

\*operands : list of array\_like

These are the arrays for the operation.

optimize : {bool, list, tuple, 'greedy', 'optimal'}

Choose the type of path. If a tuple is provided, the second argument is

assumed to be the maximum intermediate size created. If only a single

argument is provided the largest input or output array size is used

as a maximum intermediate size.

\* if a list is given that starts with ``einsum\_path``, uses this as the

contraction path

\* if False no optimization is taken

\* if True defaults to the 'greedy' algorithm

\* 'optimal' An algorithm that combinatorially explores all possible

ways of contracting the listed tensors and choosest the least costly

path. Scales exponentially with the number of terms in the

contraction.

\* 'greedy' An algorithm that chooses the best pair contraction

at each step. Effectively, this algorithm searches the largest inner,

Hadamard, and then outer products at each step. Scales cubically with

the number of terms in the contraction. Equivalent to the 'optimal'

path for most contractions.

Default is 'greedy'.

Returns

-------

path : list of tuples

A list representation of the einsum path.

string\_repr : str

A printable representation of the einsum path.

Notes

-----

The resulting path indicates which terms of the input contraction should be

contracted first, the result of this contraction is then appended to the

end of the contraction list. This list can then be iterated over until all

intermediate contractions are complete.

See Also

--------

einsum, linalg.multi\_dot

Examples

--------

We can begin with a chain dot example. In this case, it is optimal to

contract the ``b`` and ``c`` tensors first as represented by the first

element of the path ``(1, 2)``. The resulting tensor is added to the end

of the contraction and the remaining contraction ``(0, 1)`` is then

completed.

>>> np.random.seed(123)

>>> a = np.random.rand(2, 2)

>>> b = np.random.rand(2, 5)

>>> c = np.random.rand(5, 2)

>>> path\_info = np.einsum\_path('ij,jk,kl->il', a, b, c, optimize='greedy')

>>> print(path\_info[0])

['einsum\_path', (1, 2), (0, 1)]

>>> print(path\_info[1])

Complete contraction: ij,jk,kl->il # may vary

Naive scaling: 4

Optimized scaling: 3

Naive FLOP count: 1.600e+02

Optimized FLOP count: 5.600e+01

Theoretical speedup: 2.857

Largest intermediate: 4.000e+00 elements

-------------------------------------------------------------------------

scaling current remaining

-------------------------------------------------------------------------

3 kl,jk->jl ij,jl->il

3 jl,ij->il il->il

A more complex index transformation example.

>>> I = np.random.rand(10, 10, 10, 10)

>>> C = np.random.rand(10, 10)

>>> path\_info = np.einsum\_path('ea,fb,abcd,gc,hd->efgh', C, C, I, C, C,

... optimize='greedy')

>>> print(path\_info[0])

['einsum\_path', (0, 2), (0, 3), (0, 2), (0, 1)]

>>> print(path\_info[1])

Complete contraction: ea,fb,abcd,gc,hd->efgh # may vary

Naive scaling: 8

Optimized scaling: 5

Naive FLOP count: 8.000e+08

Optimized FLOP count: 8.000e+05

Theoretical speedup: 1000.000

Largest intermediate: 1.000e+04 elements

--------------------------------------------------------------------------

scaling current remaining

--------------------------------------------------------------------------

5 abcd,ea->bcde fb,gc,hd,bcde->efgh

5 bcde,fb->cdef gc,hd,cdef->efgh

5 cdef,gc->defg hd,defg->efgh

5 defg,hd->efgh efgh->efgh

empty(...)

empty(shape, dtype=float, order='C')

Return a new array of given shape and type, without initializing entries.

Parameters

----------

shape : int or tuple of int

Shape of the empty array, e.g., ``(2, 3)`` or ``2``.

dtype : data-type, optional

Desired output data-type for the array, e.g, `numpy.int8`. Default is

`numpy.float64`.

order : {'C', 'F'}, optional, default: 'C'

Whether to store multi-dimensional data in row-major

(C-style) or column-major (Fortran-style) order in

memory.

Returns

-------

out : ndarray

Array of uninitialized (arbitrary) data of the given shape, dtype, and

order. Object arrays will be initialized to None.

See Also

--------

empty\_like : Return an empty array with shape and type of input.

ones : Return a new array setting values to one.

zeros : Return a new array setting values to zero.

full : Return a new array of given shape filled with value.

Notes

-----

`empty`, unlike `zeros`, does not set the array values to zero,

and may therefore be marginally faster. On the other hand, it requires

the user to manually set all the values in the array, and should be

used with caution.

Examples

--------

>>> np.empty([2, 2])

array([[ -9.74499359e+001, 6.69583040e-309],

[ 2.13182611e-314, 3.06959433e-309]]) #uninitialized

>>> np.empty([2, 2], dtype=int)

array([[-1073741821, -1067949133],

[ 496041986, 19249760]]) #uninitialized

empty\_like(...)

empty\_like(prototype, dtype=None, order='K', subok=True, shape=None)

Return a new array with the same shape and type as a given array.

Parameters

----------

prototype : array\_like

The shape and data-type of `prototype` define these same attributes

of the returned array.

dtype : data-type, optional

Overrides the data type of the result.

.. versionadded:: 1.6.0

order : {'C', 'F', 'A', or 'K'}, optional

Overrides the memory layout of the result. 'C' means C-order,

'F' means F-order, 'A' means 'F' if ``prototype`` is Fortran

contiguous, 'C' otherwise. 'K' means match the layout of ``prototype``

as closely as possible.

.. versionadded:: 1.6.0

subok : bool, optional.

If True, then the newly created array will use the sub-class

type of 'a', otherwise it will be a base-class array. Defaults

to True.

shape : int or sequence of ints, optional.

Overrides the shape of the result. If order='K' and the number of

dimensions is unchanged, will try to keep order, otherwise,

order='C' is implied.

.. versionadded:: 1.17.0

Returns

-------

out : ndarray

Array of uninitialized (arbitrary) data with the same

shape and type as `prototype`.

See Also

--------

ones\_like : Return an array of ones with shape and type of input.

zeros\_like : Return an array of zeros with shape and type of input.

full\_like : Return a new array with shape of input filled with value.

empty : Return a new uninitialized array.

Notes

-----

This function does \*not\* initialize the returned array; to do that use

`zeros\_like` or `ones\_like` instead. It may be marginally faster than

the functions that do set the array values.

Examples

--------

>>> a = ([1,2,3], [4,5,6]) # a is array-like

>>> np.empty\_like(a)

array([[-1073741821, -1073741821, 3], # uninitialized

[ 0, 0, -1073741821]])

>>> a = np.array([[1., 2., 3.],[4.,5.,6.]])

>>> np.empty\_like(a)

array([[ -2.00000715e+000, 1.48219694e-323, -2.00000572e+000], # uninitialized

[ 4.38791518e-305, -2.00000715e+000, 4.17269252e-309]])

expand\_dims(a, axis)

Expand the shape of an array.

Insert a new axis that will appear at the `axis` position in the expanded

array shape.

.. note:: Previous to NumPy 1.13.0, neither ``axis < -a.ndim - 1`` nor

``axis > a.ndim`` raised errors or put the new axis where documented.

Those axis values are now deprecated and will raise an AxisError in the

future.

Parameters

----------

a : array\_like

Input array.

axis : int

Position in the expanded axes where the new axis is placed.

Returns

-------

res : ndarray

View of `a` with the number of dimensions increased by one.

See Also

--------

squeeze : The inverse operation, removing singleton dimensions

reshape : Insert, remove, and combine dimensions, and resize existing ones

doc.indexing, atleast\_1d, atleast\_2d, atleast\_3d

Examples

--------

>>> x = np.array([1,2])

>>> x.shape

(2,)

The following is equivalent to ``x[np.newaxis,:]`` or ``x[np.newaxis]``:

>>> y = np.expand\_dims(x, axis=0)

>>> y

array([[1, 2]])

>>> y.shape

(1, 2)

>>> y = np.expand\_dims(x, axis=1) # Equivalent to x[:,np.newaxis]

>>> y

array([[1],

[2]])

>>> y.shape

(2, 1)

Note that some examples may use ``None`` instead of ``np.newaxis``. These

are the same objects:

>>> np.newaxis is None

True

extract(condition, arr)

Return the elements of an array that satisfy some condition.

This is equivalent to ``np.compress(ravel(condition), ravel(arr))``. If

`condition` is boolean ``np.extract`` is equivalent to ``arr[condition]``.

Note that `place` does the exact opposite of `extract`.

Parameters

----------

condition : array\_like

An array whose nonzero or True entries indicate the elements of `arr`

to extract.

arr : array\_like

Input array of the same size as `condition`.

Returns

-------

extract : ndarray

Rank 1 array of values from `arr` where `condition` is True.

See Also

--------

take, put, copyto, compress, place

Examples

--------

>>> arr = np.arange(12).reshape((3, 4))

>>> arr

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

>>> condition = np.mod(arr, 3)==0

>>> condition

array([[ True, False, False, True],

[False, False, True, False],

[False, True, False, False]])

>>> np.extract(condition, arr)

array([0, 3, 6, 9])

If `condition` is boolean:

>>> arr[condition]

array([0, 3, 6, 9])

eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

Return a 2-D array with ones on the diagonal and zeros elsewhere.

Parameters

----------

N : int

Number of rows in the output.

M : int, optional

Number of columns in the output. If None, defaults to `N`.

k : int, optional

Index of the diagonal: 0 (the default) refers to the main diagonal,

a positive value refers to an upper diagonal, and a negative value

to a lower diagonal.

dtype : data-type, optional

Data-type of the returned array.

order : {'C', 'F'}, optional

Whether the output should be stored in row-major (C-style) or

column-major (Fortran-style) order in memory.

.. versionadded:: 1.14.0

Returns

-------

I : ndarray of shape (N,M)

An array where all elements are equal to zero, except for the `k`-th

diagonal, whose values are equal to one.

See Also

--------

identity : (almost) equivalent function

diag : diagonal 2-D array from a 1-D array specified by the user.

Examples

--------

>>> np.eye(2, dtype=int)

array([[1, 0],

[0, 1]])

>>> np.eye(3, k=1)

array([[0., 1., 0.],

[0., 0., 1.],

[0., 0., 0.]])

fastCopyAndTranspose = \_fastCopyAndTranspose(...)

\_fastCopyAndTranspose(a)

fill\_diagonal(a, val, wrap=False)

Fill the main diagonal of the given array of any dimensionality.

For an array `a` with ``a.ndim >= 2``, the diagonal is the list of

locations with indices ``a[i, ..., i]`` all identical. This function

modifies the input array in-place, it does not return a value.

Parameters

----------

a : array, at least 2-D.

Array whose diagonal is to be filled, it gets modified in-place.

val : scalar

Value to be written on the diagonal, its type must be compatible with

that of the array a.

wrap : bool

For tall matrices in NumPy version up to 1.6.2, the

diagonal "wrapped" after N columns. You can have this behavior

with this option. This affects only tall matrices.

See also

--------

diag\_indices, diag\_indices\_from

Notes

-----

.. versionadded:: 1.4.0

This functionality can be obtained via `diag\_indices`, but internally

this version uses a much faster implementation that never constructs the

indices and uses simple slicing.

Examples

--------

>>> a = np.zeros((3, 3), int)

>>> np.fill\_diagonal(a, 5)

>>> a

array([[5, 0, 0],

[0, 5, 0],

[0, 0, 5]])

The same function can operate on a 4-D array:

>>> a = np.zeros((3, 3, 3, 3), int)

>>> np.fill\_diagonal(a, 4)

We only show a few blocks for clarity:

>>> a[0, 0]

array([[4, 0, 0],

[0, 0, 0],

[0, 0, 0]])

>>> a[1, 1]

array([[0, 0, 0],

[0, 4, 0],

[0, 0, 0]])

>>> a[2, 2]

array([[0, 0, 0],

[0, 0, 0],

[0, 0, 4]])

The wrap option affects only tall matrices:

>>> # tall matrices no wrap

>>> a = np.zeros((5, 3), int)

>>> np.fill\_diagonal(a, 4)

>>> a

array([[4, 0, 0],

[0, 4, 0],

[0, 0, 4],

[0, 0, 0],

[0, 0, 0]])

>>> # tall matrices wrap

>>> a = np.zeros((5, 3), int)

>>> np.fill\_diagonal(a, 4, wrap=True)

>>> a

array([[4, 0, 0],

[0, 4, 0],

[0, 0, 4],

[0, 0, 0],

[4, 0, 0]])

>>> # wide matrices

>>> a = np.zeros((3, 5), int)

>>> np.fill\_diagonal(a, 4, wrap=True)

>>> a

array([[4, 0, 0, 0, 0],

[0, 4, 0, 0, 0],

[0, 0, 4, 0, 0]])

The anti-diagonal can be filled by reversing the order of elements

using either `numpy.flipud` or `numpy.fliplr`.

>>> a = np.zeros((3, 3), int);

>>> np.fill\_diagonal(np.fliplr(a), [1,2,3]) # Horizontal flip

>>> a

array([[0, 0, 1],

[0, 2, 0],

[3, 0, 0]])

>>> np.fill\_diagonal(np.flipud(a), [1,2,3]) # Vertical flip

>>> a

array([[0, 0, 3],

[0, 2, 0],

[1, 0, 0]])

Note that the order in which the diagonal is filled varies depending

on the flip function.

find\_common\_type(array\_types, scalar\_types)

Determine common type following standard coercion rules.

Parameters

----------

array\_types : sequence

A list of dtypes or dtype convertible objects representing arrays.

scalar\_types : sequence

A list of dtypes or dtype convertible objects representing scalars.

Returns

-------

datatype : dtype

The common data type, which is the maximum of `array\_types` ignoring

`scalar\_types`, unless the maximum of `scalar\_types` is of a

different kind (`dtype.kind`). If the kind is not understood, then

None is returned.

See Also

--------

dtype, common\_type, can\_cast, mintypecode

Examples

--------

>>> np.find\_common\_type([], [np.int64, np.float32, complex])

dtype('complex128')

>>> np.find\_common\_type([np.int64, np.float32], [])

dtype('float64')

The standard casting rules ensure that a scalar cannot up-cast an

array unless the scalar is of a fundamentally different kind of data

(i.e. under a different hierarchy in the data type hierarchy) then

the array:

>>> np.find\_common\_type([np.float32], [np.int64, np.float64])

dtype('float32')

Complex is of a different type, so it up-casts the float in the

`array\_types` argument:

>>> np.find\_common\_type([np.float32], [complex])

dtype('complex128')

Type specifier strings are convertible to dtypes and can therefore

be used instead of dtypes:

>>> np.find\_common\_type(['f4', 'f4', 'i4'], ['c8'])

dtype('complex128')

fix(x, out=None)

Round to nearest integer towards zero.

Round an array of floats element-wise to nearest integer towards zero.

The rounded values are returned as floats.

Parameters

----------

x : array\_like

An array of floats to be rounded

y : ndarray, optional

Output array

Returns

-------

out : ndarray of floats

The array of rounded numbers

See Also

--------

trunc, floor, ceil

around : Round to given number of decimals

Examples

--------

>>> np.fix(3.14)

3.0

>>> np.fix(3)

3.0

>>> np.fix([2.1, 2.9, -2.1, -2.9])

array([ 2., 2., -2., -2.])

flatnonzero(a)

Return indices that are non-zero in the flattened version of a.

This is equivalent to np.nonzero(np.ravel(a))[0].

Parameters

----------

a : array\_like

Input data.

Returns

-------

res : ndarray

Output array, containing the indices of the elements of `a.ravel()`

that are non-zero.

See Also

--------

nonzero : Return the indices of the non-zero elements of the input array.

ravel : Return a 1-D array containing the elements of the input array.

Examples

--------

>>> x = np.arange(-2, 3)

>>> x

array([-2, -1, 0, 1, 2])

>>> np.flatnonzero(x)

array([0, 1, 3, 4])

Use the indices of the non-zero elements as an index array to extract

these elements:

>>> x.ravel()[np.flatnonzero(x)]

array([-2, -1, 1, 2])

flip(m, axis=None)

Reverse the order of elements in an array along the given axis.

The shape of the array is preserved, but the elements are reordered.

.. versionadded:: 1.12.0

Parameters

----------

m : array\_like

Input array.

axis : None or int or tuple of ints, optional

Axis or axes along which to flip over. The default,

axis=None, will flip over all of the axes of the input array.

If axis is negative it counts from the last to the first axis.

If axis is a tuple of ints, flipping is performed on all of the axes

specified in the tuple.

.. versionchanged:: 1.15.0

None and tuples of axes are supported

Returns

-------

out : array\_like

A view of `m` with the entries of axis reversed. Since a view is

returned, this operation is done in constant time.

See Also

--------

flipud : Flip an array vertically (axis=0).

fliplr : Flip an array horizontally (axis=1).

Notes

-----

flip(m, 0) is equivalent to flipud(m).

flip(m, 1) is equivalent to fliplr(m).

flip(m, n) corresponds to ``m[...,::-1,...]`` with ``::-1`` at position n.

flip(m) corresponds to ``m[::-1,::-1,...,::-1]`` with ``::-1`` at all

positions.

flip(m, (0, 1)) corresponds to ``m[::-1,::-1,...]`` with ``::-1`` at

position 0 and position 1.

Examples

--------

>>> A = np.arange(8).reshape((2,2,2))

>>> A

array([[[0, 1],

[2, 3]],

[[4, 5],

[6, 7]]])

>>> np.flip(A, 0)

array([[[4, 5],

[6, 7]],

[[0, 1],

[2, 3]]])

>>> np.flip(A, 1)

array([[[2, 3],

[0, 1]],

[[6, 7],

[4, 5]]])

>>> np.flip(A)

array([[[7, 6],

[5, 4]],

[[3, 2],

[1, 0]]])

>>> np.flip(A, (0, 2))

array([[[5, 4],

[7, 6]],

[[1, 0],

[3, 2]]])

>>> A = np.random.randn(3,4,5)

>>> np.all(np.flip(A,2) == A[:,:,::-1,...])

True

fliplr(m)

Flip array in the left/right direction.

Flip the entries in each row in the left/right direction.

Columns are preserved, but appear in a different order than before.

Parameters

----------

m : array\_like

Input array, must be at least 2-D.

Returns

-------

f : ndarray

A view of `m` with the columns reversed. Since a view

is returned, this operation is :math:`\mathcal O(1)`.

See Also

--------

flipud : Flip array in the up/down direction.

rot90 : Rotate array counterclockwise.

Notes

-----

Equivalent to m[:,::-1]. Requires the array to be at least 2-D.

Examples

--------

>>> A = np.diag([1.,2.,3.])

>>> A

array([[1., 0., 0.],

[0., 2., 0.],

[0., 0., 3.]])

>>> np.fliplr(A)

array([[0., 0., 1.],

[0., 2., 0.],

[3., 0., 0.]])

>>> A = np.random.randn(2,3,5)

>>> np.all(np.fliplr(A) == A[:,::-1,...])

True

flipud(m)

Flip array in the up/down direction.

Flip the entries in each column in the up/down direction.

Rows are preserved, but appear in a different order than before.

Parameters

----------

m : array\_like

Input array.

Returns

-------

out : array\_like

A view of `m` with the rows reversed. Since a view is

returned, this operation is :math:`\mathcal O(1)`.

See Also

--------

fliplr : Flip array in the left/right direction.

rot90 : Rotate array counterclockwise.

Notes

-----

Equivalent to ``m[::-1,...]``.

Does not require the array to be two-dimensional.

Examples

--------

>>> A = np.diag([1.0, 2, 3])

>>> A

array([[1., 0., 0.],

[0., 2., 0.],

[0., 0., 3.]])

>>> np.flipud(A)

array([[0., 0., 3.],

[0., 2., 0.],

[1., 0., 0.]])

>>> A = np.random.randn(2,3,5)

>>> np.all(np.flipud(A) == A[::-1,...])

True

>>> np.flipud([1,2])

array([2, 1])

format\_float\_positional(x, precision=None, unique=True, fractional=True, trim='k', sign=False, pad\_left=None, pad\_right=None)

Format a floating-point scalar as a decimal string in positional notation.

Provides control over rounding, trimming and padding. Uses and assumes

IEEE unbiased rounding. Uses the "Dragon4" algorithm.

Parameters

----------

x : python float or numpy floating scalar

Value to format.

precision : non-negative integer or None, optional

Maximum number of digits to print. May be None if `unique` is

`True`, but must be an integer if unique is `False`.

unique : boolean, optional

If `True`, use a digit-generation strategy which gives the shortest

representation which uniquely identifies the floating-point number from

other values of the same type, by judicious rounding. If `precision`

was omitted, print out all necessary digits, otherwise digit generation

is cut off after `precision` digits and the remaining value is rounded.

If `False`, digits are generated as if printing an infinite-precision

value and stopping after `precision` digits, rounding the remaining

value.

fractional : boolean, optional

If `True`, the cutoff of `precision` digits refers to the total number

of digits after the decimal point, including leading zeros.

If `False`, `precision` refers to the total number of significant

digits, before or after the decimal point, ignoring leading zeros.

trim : one of 'k', '.', '0', '-', optional

Controls post-processing trimming of trailing digits, as follows:

\* 'k' : keep trailing zeros, keep decimal point (no trimming)

\* '.' : trim all trailing zeros, leave decimal point

\* '0' : trim all but the zero before the decimal point. Insert the

zero if it is missing.

\* '-' : trim trailing zeros and any trailing decimal point

sign : boolean, optional

Whether to show the sign for positive values.

pad\_left : non-negative integer, optional

Pad the left side of the string with whitespace until at least that

many characters are to the left of the decimal point.

pad\_right : non-negative integer, optional

Pad the right side of the string with whitespace until at least that

many characters are to the right of the decimal point.

Returns

-------

rep : string

The string representation of the floating point value

See Also

--------

format\_float\_scientific

Examples

--------

>>> np.format\_float\_positional(np.float32(np.pi))

'3.1415927'

>>> np.format\_float\_positional(np.float16(np.pi))

'3.14'

>>> np.format\_float\_positional(np.float16(0.3))

'0.3'

>>> np.format\_float\_positional(np.float16(0.3), unique=False, precision=10)

'0.3000488281'

format\_float\_scientific(x, precision=None, unique=True, trim='k', sign=False, pad\_left=None, exp\_digits=None)

Format a floating-point scalar as a decimal string in scientific notation.

Provides control over rounding, trimming and padding. Uses and assumes

IEEE unbiased rounding. Uses the "Dragon4" algorithm.

Parameters

----------

x : python float or numpy floating scalar

Value to format.

precision : non-negative integer or None, optional

Maximum number of digits to print. May be None if `unique` is

`True`, but must be an integer if unique is `False`.

unique : boolean, optional

If `True`, use a digit-generation strategy which gives the shortest

representation which uniquely identifies the floating-point number from

other values of the same type, by judicious rounding. If `precision`

was omitted, print all necessary digits, otherwise digit generation is

cut off after `precision` digits and the remaining value is rounded.

If `False`, digits are generated as if printing an infinite-precision

value and stopping after `precision` digits, rounding the remaining

value.

trim : one of 'k', '.', '0', '-', optional

Controls post-processing trimming of trailing digits, as follows:

\* 'k' : keep trailing zeros, keep decimal point (no trimming)

\* '.' : trim all trailing zeros, leave decimal point

\* '0' : trim all but the zero before the decimal point. Insert the

zero if it is missing.

\* '-' : trim trailing zeros and any trailing decimal point

sign : boolean, optional

Whether to show the sign for positive values.

pad\_left : non-negative integer, optional

Pad the left side of the string with whitespace until at least that

many characters are to the left of the decimal point.

exp\_digits : non-negative integer, optional

Pad the exponent with zeros until it contains at least this many digits.

If omitted, the exponent will be at least 2 digits.

Returns

-------

rep : string

The string representation of the floating point value

See Also

--------

format\_float\_positional

Examples

--------

>>> np.format\_float\_scientific(np.float32(np.pi))

'3.1415927e+00'

>>> s = np.float32(1.23e24)

>>> np.format\_float\_scientific(s, unique=False, precision=15)

'1.230000071797338e+24'

>>> np.format\_float\_scientific(s, exp\_digits=4)

'1.23e+0024'

frombuffer(...)

frombuffer(buffer, dtype=float, count=-1, offset=0)

Interpret a buffer as a 1-dimensional array.

Parameters

----------

buffer : buffer\_like

An object that exposes the buffer interface.

dtype : data-type, optional

Data-type of the returned array; default: float.

count : int, optional

Number of items to read. ``-1`` means all data in the buffer.

offset : int, optional

Start reading the buffer from this offset (in bytes); default: 0.

Notes

-----

If the buffer has data that is not in machine byte-order, this should

be specified as part of the data-type, e.g.::

>>> dt = np.dtype(int)

>>> dt = dt.newbyteorder('>')

>>> np.frombuffer(buf, dtype=dt) # doctest: +SKIP

The data of the resulting array will not be byteswapped, but will be

interpreted correctly.

Examples

--------

>>> s = b'hello world'

>>> np.frombuffer(s, dtype='S1', count=5, offset=6)

array([b'w', b'o', b'r', b'l', b'd'], dtype='|S1')

>>> np.frombuffer(b'\x01\x02', dtype=np.uint8)

array([1, 2], dtype=uint8)

>>> np.frombuffer(b'\x01\x02\x03\x04\x05', dtype=np.uint8, count=3)

array([1, 2, 3], dtype=uint8)

fromfile(...)

fromfile(file, dtype=float, count=-1, sep='', offset=0)

Construct an array from data in a text or binary file.

A highly efficient way of reading binary data with a known data-type,

as well as parsing simply formatted text files. Data written using the

`tofile` method can be read using this function.

Parameters

----------

file : file or str or Path

Open file object or filename.

.. versionchanged:: 1.17.0

`pathlib.Path` objects are now accepted.

dtype : data-type

Data type of the returned array.

For binary files, it is used to determine the size and byte-order

of the items in the file.

count : int

Number of items to read. ``-1`` means all items (i.e., the complete

file).

sep : str

Separator between items if file is a text file.

Empty ("") separator means the file should be treated as binary.

Spaces (" ") in the separator match zero or more whitespace characters.

A separator consisting only of spaces must match at least one

whitespace.

offset : int

The offset (in bytes) from the file's current position. Defaults to 0.

Only permitted for binary files.

.. versionadded:: 1.17.0

See also

--------

load, save

ndarray.tofile

loadtxt : More flexible way of loading data from a text file.

Notes

-----

Do not rely on the combination of `tofile` and `fromfile` for

data storage, as the binary files generated are are not platform

independent. In particular, no byte-order or data-type information is

saved. Data can be stored in the platform independent ``.npy`` format

using `save` and `load` instead.

Examples

--------

Construct an ndarray:

>>> dt = np.dtype([('time', [('min', np.int64), ('sec', np.int64)]),

... ('temp', float)])

>>> x = np.zeros((1,), dtype=dt)

>>> x['time']['min'] = 10; x['temp'] = 98.25

>>> x

array([((10, 0), 98.25)],

dtype=[('time', [('min', '<i8'), ('sec', '<i8')]), ('temp', '<f8')])

Save the raw data to disk:

>>> import tempfile

>>> fname = tempfile.mkstemp()[1]

>>> x.tofile(fname)

Read the raw data from disk:

>>> np.fromfile(fname, dtype=dt)

array([((10, 0), 98.25)],

dtype=[('time', [('min', '<i8'), ('sec', '<i8')]), ('temp', '<f8')])

The recommended way to store and load data:

>>> np.save(fname, x)

>>> np.load(fname + '.npy')

array([((10, 0), 98.25)],

dtype=[('time', [('min', '<i8'), ('sec', '<i8')]), ('temp', '<f8')])

fromfunction(function, shape, \*\*kwargs)

Construct an array by executing a function over each coordinate.

The resulting array therefore has a value ``fn(x, y, z)`` at

coordinate ``(x, y, z)``.

Parameters

----------

function : callable

The function is called with N parameters, where N is the rank of

`shape`. Each parameter represents the coordinates of the array

varying along a specific axis. For example, if `shape`

were ``(2, 2)``, then the parameters would be

``array([[0, 0], [1, 1]])`` and ``array([[0, 1], [0, 1]])``

shape : (N,) tuple of ints

Shape of the output array, which also determines the shape of

the coordinate arrays passed to `function`.

dtype : data-type, optional

Data-type of the coordinate arrays passed to `function`.

By default, `dtype` is float.

Returns

-------

fromfunction : any

The result of the call to `function` is passed back directly.

Therefore the shape of `fromfunction` is completely determined by

`function`. If `function` returns a scalar value, the shape of

`fromfunction` would not match the `shape` parameter.

See Also

--------

indices, meshgrid

Notes

-----

Keywords other than `dtype` are passed to `function`.

Examples

--------

>>> np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)

array([[ True, False, False],

[False, True, False],

[False, False, True]])

>>> np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)

array([[0, 1, 2],

[1, 2, 3],

[2, 3, 4]])

fromiter(...)

fromiter(iterable, dtype, count=-1)

Create a new 1-dimensional array from an iterable object.

Parameters

----------

iterable : iterable object

An iterable object providing data for the array.

dtype : data-type

The data-type of the returned array.

count : int, optional

The number of items to read from \*iterable\*. The default is -1,

which means all data is read.

Returns

-------

out : ndarray

The output array.

Notes

-----

Specify `count` to improve performance. It allows ``fromiter`` to

pre-allocate the output array, instead of resizing it on demand.

Examples

--------

>>> iterable = (x\*x for x in range(5))

>>> np.fromiter(iterable, float)

array([ 0., 1., 4., 9., 16.])

frompyfunc(...)

frompyfunc(func, nin, nout)

Takes an arbitrary Python function and returns a NumPy ufunc.

Can be used, for example, to add broadcasting to a built-in Python

function (see Examples section).

Parameters

----------

func : Python function object

An arbitrary Python function.

nin : int

The number of input arguments.

nout : int

The number of objects returned by `func`.

Returns

-------

out : ufunc

Returns a NumPy universal function (``ufunc``) object.

See Also

--------

vectorize : evaluates pyfunc over input arrays using broadcasting rules of numpy

Notes

-----

The returned ufunc always returns PyObject arrays.

Examples

--------

Use frompyfunc to add broadcasting to the Python function ``oct``:

>>> oct\_array = np.frompyfunc(oct, 1, 1)

>>> oct\_array(np.array((10, 30, 100)))

array(['0o12', '0o36', '0o144'], dtype=object)

>>> np.array((oct(10), oct(30), oct(100))) # for comparison

array(['0o12', '0o36', '0o144'], dtype='<U5')

fromregex(file, regexp, dtype, encoding=None)

Construct an array from a text file, using regular expression parsing.

The returned array is always a structured array, and is constructed from

all matches of the regular expression in the file. Groups in the regular

expression are converted to fields of the structured array.

Parameters

----------

file : str or file

File name or file object to read.

regexp : str or regexp

Regular expression used to parse the file.

Groups in the regular expression correspond to fields in the dtype.

dtype : dtype or list of dtypes

Dtype for the structured array.

encoding : str, optional

Encoding used to decode the inputfile. Does not apply to input streams.

.. versionadded:: 1.14.0

Returns

-------

output : ndarray

The output array, containing the part of the content of `file` that

was matched by `regexp`. `output` is always a structured array.

Raises

------

TypeError

When `dtype` is not a valid dtype for a structured array.

See Also

--------

fromstring, loadtxt

Notes

-----

Dtypes for structured arrays can be specified in several forms, but all

forms specify at least the data type and field name. For details see

`doc.structured\_arrays`.

Examples

--------

>>> f = open('test.dat', 'w')

>>> \_ = f.write("1312 foo\n1534 bar\n444 qux")

>>> f.close()

>>> regexp = r"(\d+)\s+(...)" # match [digits, whitespace, anything]

>>> output = np.fromregex('test.dat', regexp,

... [('num', np.int64), ('key', 'S3')])

>>> output

array([(1312, b'foo'), (1534, b'bar'), ( 444, b'qux')],

dtype=[('num', '<i8'), ('key', 'S3')])

>>> output['num']

array([1312, 1534, 444])

fromstring(...)

fromstring(string, dtype=float, count=-1, sep='')

A new 1-D array initialized from text data in a string.

Parameters

----------

string : str

A string containing the data.

dtype : data-type, optional

The data type of the array; default: float. For binary input data,

the data must be in exactly this format.

count : int, optional

Read this number of `dtype` elements from the data. If this is

negative (the default), the count will be determined from the

length of the data.

sep : str, optional

The string separating numbers in the data; extra whitespace between

elements is also ignored.

.. deprecated:: 1.14

Passing ``sep=''``, the default, is deprecated since it will

trigger the deprecated binary mode of this function. This mode

interprets `string` as binary bytes, rather than ASCII text with

decimal numbers, an operation which is better spelt

``frombuffer(string, dtype, count)``. If `string` contains unicode

text, the binary mode of `fromstring` will first encode it into

bytes using either utf-8 (python 3) or the default encoding

(python 2), neither of which produce sane results.

Returns

-------

arr : ndarray

The constructed array.

Raises

------

ValueError

If the string is not the correct size to satisfy the requested

`dtype` and `count`.

See Also

--------

frombuffer, fromfile, fromiter

Examples

--------

>>> np.fromstring('1 2', dtype=int, sep=' ')

array([1, 2])

>>> np.fromstring('1, 2', dtype=int, sep=',')

array([1, 2])

full(shape, fill\_value, dtype=None, order='C')

Return a new array of given shape and type, filled with `fill\_value`.

Parameters

----------

shape : int or sequence of ints

Shape of the new array, e.g., ``(2, 3)`` or ``2``.

fill\_value : scalar

Fill value.

dtype : data-type, optional

The desired data-type for the array The default, `None`, means

`np.array(fill\_value).dtype`.

order : {'C', 'F'}, optional

Whether to store multidimensional data in C- or Fortran-contiguous

(row- or column-wise) order in memory.

Returns

-------

out : ndarray

Array of `fill\_value` with the given shape, dtype, and order.

See Also

--------

full\_like : Return a new array with shape of input filled with value.

empty : Return a new uninitialized array.

ones : Return a new array setting values to one.

zeros : Return a new array setting values to zero.

Examples

--------

>>> np.full((2, 2), np.inf)

array([[inf, inf],

[inf, inf]])

>>> np.full((2, 2), 10)

array([[10, 10],

[10, 10]])

full\_like(a, fill\_value, dtype=None, order='K', subok=True, shape=None)

Return a full array with the same shape and type as a given array.

Parameters

----------

a : array\_like

The shape and data-type of `a` define these same attributes of

the returned array.

fill\_value : scalar

Fill value.

dtype : data-type, optional

Overrides the data type of the result.

order : {'C', 'F', 'A', or 'K'}, optional

Overrides the memory layout of the result. 'C' means C-order,

'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

'C' otherwise. 'K' means match the layout of `a` as closely

as possible.

subok : bool, optional.

If True, then the newly created array will use the sub-class

type of 'a', otherwise it will be a base-class array. Defaults

to True.

shape : int or sequence of ints, optional.

Overrides the shape of the result. If order='K' and the number of

dimensions is unchanged, will try to keep order, otherwise,

order='C' is implied.

.. versionadded:: 1.17.0

Returns

-------

out : ndarray

Array of `fill\_value` with the same shape and type as `a`.

See Also

--------

empty\_like : Return an empty array with shape and type of input.

ones\_like : Return an array of ones with shape and type of input.

zeros\_like : Return an array of zeros with shape and type of input.

full : Return a new array of given shape filled with value.

Examples

--------

>>> x = np.arange(6, dtype=int)

>>> np.full\_like(x, 1)

array([1, 1, 1, 1, 1, 1])

>>> np.full\_like(x, 0.1)

array([0, 0, 0, 0, 0, 0])

>>> np.full\_like(x, 0.1, dtype=np.double)

array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1])

>>> np.full\_like(x, np.nan, dtype=np.double)

array([nan, nan, nan, nan, nan, nan])

>>> y = np.arange(6, dtype=np.double)

>>> np.full\_like(y, 0.1)

array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1])

fv(rate, nper, pmt, pv, when='end')

Compute the future value.

Given:

\* a present value, `pv`

\* an interest `rate` compounded once per period, of which

there are

\* `nper` total

\* a (fixed) payment, `pmt`, paid either

\* at the beginning (`when` = {'begin', 1}) or the end

(`when` = {'end', 0}) of each period

Return:

the value at the end of the `nper` periods

Parameters

----------

rate : scalar or array\_like of shape(M, )

Rate of interest as decimal (not per cent) per period

nper : scalar or array\_like of shape(M, )

Number of compounding periods

pmt : scalar or array\_like of shape(M, )

Payment

pv : scalar or array\_like of shape(M, )

Present value

when : {{'begin', 1}, {'end', 0}}, {string, int}, optional

When payments are due ('begin' (1) or 'end' (0)).

Defaults to {'end', 0}.

Returns

-------

out : ndarray

Future values. If all input is scalar, returns a scalar float. If

any input is array\_like, returns future values for each input element.

If multiple inputs are array\_like, they all must have the same shape.

Notes

-----

The future value is computed by solving the equation::

fv +

pv\*(1+rate)\*\*nper +

pmt\*(1 + rate\*when)/rate\*((1 + rate)\*\*nper - 1) == 0

or, when ``rate == 0``::

fv + pv + pmt \* nper == 0

References

----------

.. [WRW] Wheeler, D. A., E. Rathke, and R. Weir (Eds.) (2009, May).

Open Document Format for Office Applications (OpenDocument)v1.2,

Part 2: Recalculated Formula (OpenFormula) Format - Annotated Version,

Pre-Draft 12. Organization for the Advancement of Structured Information

Standards (OASIS). Billerica, MA, USA. [ODT Document].

Available:

http://www.oasis-open.org/committees/documents.php?wg\_abbrev=office-formula

OpenDocument-formula-20090508.odt

Examples

--------

What is the future value after 10 years of saving $100 now, with

an additional monthly savings of $100. Assume the interest rate is

5% (annually) compounded monthly?

>>> np.fv(0.05/12, 10\*12, -100, -100)

15692.928894335748

By convention, the negative sign represents cash flow out (i.e. money not

available today). Thus, saving $100 a month at 5% annual interest leads

to $15,692.93 available to spend in 10 years.

If any input is array\_like, returns an array of equal shape. Let's

compare different interest rates from the example above.

>>> a = np.array((0.05, 0.06, 0.07))/12

>>> np.fv(a, 10\*12, -100, -100)

array([ 15692.92889434, 16569.87435405, 17509.44688102]) # may vary

genfromtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, skip\_header=0, skip\_footer=0, converters=None, missing\_values=None, filling\_values=None, usecols=None, names=None, excludelist=None, deletechars=" !#$%&'()\*+,-./:;<=>?@[\\]^{|}~", replace\_space='\_', autostrip=False, case\_sensitive=True, defaultfmt='f%i', unpack=None, usemask=False, loose=True, invalid\_raise=True, max\_rows=None, encoding='bytes')

Load data from a text file, with missing values handled as specified.

Each line past the first `skip\_header` lines is split at the `delimiter`

character, and characters following the `comments` character are discarded.

Parameters

----------

fname : file, str, pathlib.Path, list of str, generator

File, filename, list, or generator to read. If the filename

extension is `.gz` or `.bz2`, the file is first decompressed. Note

that generators must return byte strings in Python 3k. The strings

in a list or produced by a generator are treated as lines.

dtype : dtype, optional

Data type of the resulting array.

If None, the dtypes will be determined by the contents of each

column, individually.

comments : str, optional

The character used to indicate the start of a comment.

All the characters occurring on a line after a comment are discarded

delimiter : str, int, or sequence, optional

The string used to separate values. By default, any consecutive

whitespaces act as delimiter. An integer or sequence of integers

can also be provided as width(s) of each field.

skiprows : int, optional

`skiprows` was removed in numpy 1.10. Please use `skip\_header` instead.

skip\_header : int, optional

The number of lines to skip at the beginning of the file.

skip\_footer : int, optional

The number of lines to skip at the end of the file.

converters : variable, optional

The set of functions that convert the data of a column to a value.

The converters can also be used to provide a default value

for missing data: ``converters = {3: lambda s: float(s or 0)}``.

missing : variable, optional

`missing` was removed in numpy 1.10. Please use `missing\_values`

instead.

missing\_values : variable, optional

The set of strings corresponding to missing data.

filling\_values : variable, optional

The set of values to be used as default when the data are missing.

usecols : sequence, optional

Which columns to read, with 0 being the first. For example,

``usecols = (1, 4, 5)`` will extract the 2nd, 5th and 6th columns.

names : {None, True, str, sequence}, optional

If `names` is True, the field names are read from the first line after

the first `skip\_header` lines. This line can optionally be proceeded

by a comment delimiter. If `names` is a sequence or a single-string of

comma-separated names, the names will be used to define the field names

in a structured dtype. If `names` is None, the names of the dtype

fields will be used, if any.

excludelist : sequence, optional

A list of names to exclude. This list is appended to the default list

['return','file','print']. Excluded names are appended an underscore:

for example, `file` would become `file\_`.

deletechars : str, optional

A string combining invalid characters that must be deleted from the

names.

defaultfmt : str, optional

A format used to define default field names, such as "f%i" or "f\_%02i".

autostrip : bool, optional

Whether to automatically strip white spaces from the variables.

replace\_space : char, optional

Character(s) used in replacement of white spaces in the variables

names. By default, use a '\_'.

case\_sensitive : {True, False, 'upper', 'lower'}, optional

If True, field names are case sensitive.

If False or 'upper', field names are converted to upper case.

If 'lower', field names are converted to lower case.

unpack : bool, optional

If True, the returned array is transposed, so that arguments may be

unpacked using ``x, y, z = loadtxt(...)``

usemask : bool, optional

If True, return a masked array.

If False, return a regular array.

loose : bool, optional

If True, do not raise errors for invalid values.

invalid\_raise : bool, optional

If True, an exception is raised if an inconsistency is detected in the

number of columns.

If False, a warning is emitted and the offending lines are skipped.

max\_rows : int, optional

The maximum number of rows to read. Must not be used with skip\_footer

at the same time. If given, the value must be at least 1. Default is

to read the entire file.

.. versionadded:: 1.10.0

encoding : str, optional

Encoding used to decode the inputfile. Does not apply when `fname` is

a file object. The special value 'bytes' enables backward compatibility

workarounds that ensure that you receive byte arrays when possible

and passes latin1 encoded strings to converters. Override this value to

receive unicode arrays and pass strings as input to converters. If set

to None the system default is used. The default value is 'bytes'.

.. versionadded:: 1.14.0

Returns

-------

out : ndarray

Data read from the text file. If `usemask` is True, this is a

masked array.

See Also

--------

numpy.loadtxt : equivalent function when no data is missing.

Notes

-----

\* When spaces are used as delimiters, or when no delimiter has been given

as input, there should not be any missing data between two fields.

\* When the variables are named (either by a flexible dtype or with `names`,

there must not be any header in the file (else a ValueError

exception is raised).

\* Individual values are not stripped of spaces by default.

When using a custom converter, make sure the function does remove spaces.

References

----------

.. [1] NumPy User Guide, section `I/O with NumPy

<https://docs.scipy.org/doc/numpy/user/basics.io.genfromtxt.html>`\_.

Examples

---------

>>> from io import StringIO

>>> import numpy as np

Comma delimited file with mixed dtype

>>> s = StringIO(u"1,1.3,abcde")

>>> data = np.genfromtxt(s, dtype=[('myint','i8'),('myfloat','f8'),

... ('mystring','S5')], delimiter=",")

>>> data

array((1, 1.3, b'abcde'),

dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])

Using dtype = None

>>> \_ = s.seek(0) # needed for StringIO example only

>>> data = np.genfromtxt(s, dtype=None,

... names = ['myint','myfloat','mystring'], delimiter=",")

>>> data

array((1, 1.3, b'abcde'),

dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])

Specifying dtype and names

>>> \_ = s.seek(0)

>>> data = np.genfromtxt(s, dtype="i8,f8,S5",

... names=['myint','myfloat','mystring'], delimiter=",")

>>> data

array((1, 1.3, b'abcde'),

dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])

An example with fixed-width columns

>>> s = StringIO(u"11.3abcde")

>>> data = np.genfromtxt(s, dtype=None, names=['intvar','fltvar','strvar'],

... delimiter=[1,3,5])

>>> data

array((1, 1.3, b'abcde'),

dtype=[('intvar', '<i8'), ('fltvar', '<f8'), ('strvar', 'S5')])

An example to show comments

>>> f = StringIO('''

... text,# of chars

... hello world,11

... numpy,5''')

>>> np.genfromtxt(f, dtype='S12,S12', delimiter=',')

array([(b'text', b''), (b'hello world', b'11'), (b'numpy', b'5')],

dtype=[('f0', 'S12'), ('f1', 'S12')])

geomspace(start, stop, num=50, endpoint=True, dtype=None, axis=0)

Return numbers spaced evenly on a log scale (a geometric progression).

This is similar to `logspace`, but with endpoints specified directly.

Each output sample is a constant multiple of the previous.

.. versionchanged:: 1.16.0

Non-scalar `start` and `stop` are now supported.

Parameters

----------

start : array\_like

The starting value of the sequence.

stop : array\_like

The final value of the sequence, unless `endpoint` is False.

In that case, ``num + 1`` values are spaced over the

interval in log-space, of which all but the last (a sequence of

length `num`) are returned.

num : integer, optional

Number of samples to generate. Default is 50.

endpoint : boolean, optional

If true, `stop` is the last sample. Otherwise, it is not included.

Default is True.

dtype : dtype

The type of the output array. If `dtype` is not given, infer the data

type from the other input arguments.

axis : int, optional

The axis in the result to store the samples. Relevant only if start

or stop are array-like. By default (0), the samples will be along a

new axis inserted at the beginning. Use -1 to get an axis at the end.

.. versionadded:: 1.16.0

Returns

-------

samples : ndarray

`num` samples, equally spaced on a log scale.

See Also

--------

logspace : Similar to geomspace, but with endpoints specified using log

and base.

linspace : Similar to geomspace, but with arithmetic instead of geometric

progression.

arange : Similar to linspace, with the step size specified instead of the

number of samples.

Notes

-----

If the inputs or dtype are complex, the output will follow a logarithmic

spiral in the complex plane. (There are an infinite number of spirals

passing through two points; the output will follow the shortest such path.)

Examples

--------

>>> np.geomspace(1, 1000, num=4)

array([ 1., 10., 100., 1000.])

>>> np.geomspace(1, 1000, num=3, endpoint=False)

array([ 1., 10., 100.])

>>> np.geomspace(1, 1000, num=4, endpoint=False)

array([ 1. , 5.62341325, 31.6227766 , 177.827941 ])

>>> np.geomspace(1, 256, num=9)

array([ 1., 2., 4., 8., 16., 32., 64., 128., 256.])

Note that the above may not produce exact integers:

>>> np.geomspace(1, 256, num=9, dtype=int)

array([ 1, 2, 4, 7, 16, 32, 63, 127, 256])

>>> np.around(np.geomspace(1, 256, num=9)).astype(int)

array([ 1, 2, 4, 8, 16, 32, 64, 128, 256])

Negative, decreasing, and complex inputs are allowed:

>>> np.geomspace(1000, 1, num=4)

array([1000., 100., 10., 1.])

>>> np.geomspace(-1000, -1, num=4)

array([-1000., -100., -10., -1.])

>>> np.geomspace(1j, 1000j, num=4) # Straight line

array([0. +1.j, 0. +10.j, 0. +100.j, 0.+1000.j])

>>> np.geomspace(-1+0j, 1+0j, num=5) # Circle

array([-1.00000000e+00+1.22464680e-16j, -7.07106781e-01+7.07106781e-01j,

6.12323400e-17+1.00000000e+00j, 7.07106781e-01+7.07106781e-01j,

1.00000000e+00+0.00000000e+00j])

Graphical illustration of ``endpoint`` parameter:

>>> import matplotlib.pyplot as plt

>>> N = 10

>>> y = np.zeros(N)

>>> plt.semilogx(np.geomspace(1, 1000, N, endpoint=True), y + 1, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.semilogx(np.geomspace(1, 1000, N, endpoint=False), y + 2, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.axis([0.5, 2000, 0, 3])

[0.5, 2000, 0, 3]

>>> plt.grid(True, color='0.7', linestyle='-', which='both', axis='both')

>>> plt.show()

get\_array\_wrap(\*args)

Find the wrapper for the array with the highest priority.

In case of ties, leftmost wins. If no wrapper is found, return None

get\_include()

Return the directory that contains the NumPy \\*.h header files.

Extension modules that need to compile against NumPy should use this

function to locate the appropriate include directory.

Notes

-----

When using ``distutils``, for example in ``setup.py``.

::

import numpy as np

...

Extension('extension\_name', ...

include\_dirs=[np.get\_include()])

...

get\_printoptions()

Return the current print options.

Returns

-------

print\_opts : dict

Dictionary of current print options with keys

- precision : int

- threshold : int

- edgeitems : int

- linewidth : int

- suppress : bool

- nanstr : str

- infstr : str

- formatter : dict of callables

- sign : str

For a full description of these options, see `set\_printoptions`.

See Also

--------

set\_printoptions, set\_string\_function

getbufsize()

Return the size of the buffer used in ufuncs.

Returns

-------

getbufsize : int

Size of ufunc buffer in bytes.

geterr()

Get the current way of handling floating-point errors.

Returns

-------

res : dict

A dictionary with keys "divide", "over", "under", and "invalid",

whose values are from the strings "ignore", "print", "log", "warn",

"raise", and "call". The keys represent possible floating-point

exceptions, and the values define how these exceptions are handled.

See Also

--------

geterrcall, seterr, seterrcall

Notes

-----

For complete documentation of the types of floating-point exceptions and

treatment options, see `seterr`.

Examples

--------

>>> from collections import OrderedDict

>>> sorted(np.geterr().items())

[('divide', 'warn'), ('invalid', 'warn'), ('over', 'warn'), ('under', 'ignore')]

>>> np.arange(3.) / np.arange(3.)

array([nan, 1., 1.])

>>> oldsettings = np.seterr(all='warn', over='raise')

>>> OrderedDict(sorted(np.geterr().items()))

OrderedDict([('divide', 'warn'), ('invalid', 'warn'), ('over', 'raise'), ('under', 'warn')])

>>> np.arange(3.) / np.arange(3.)

array([nan, 1., 1.])

geterrcall()

Return the current callback function used on floating-point errors.

When the error handling for a floating-point error (one of "divide",

"over", "under", or "invalid") is set to 'call' or 'log', the function

that is called or the log instance that is written to is returned by

`geterrcall`. This function or log instance has been set with

`seterrcall`.

Returns

-------

errobj : callable, log instance or None

The current error handler. If no handler was set through `seterrcall`,

``None`` is returned.

See Also

--------

seterrcall, seterr, geterr

Notes

-----

For complete documentation of the types of floating-point exceptions and

treatment options, see `seterr`.

Examples

--------

>>> np.geterrcall() # we did not yet set a handler, returns None

>>> oldsettings = np.seterr(all='call')

>>> def err\_handler(type, flag):

... print("Floating point error (%s), with flag %s" % (type, flag))

>>> oldhandler = np.seterrcall(err\_handler)

>>> np.array([1, 2, 3]) / 0.0

Floating point error (divide by zero), with flag 1

array([inf, inf, inf])

>>> cur\_handler = np.geterrcall()

>>> cur\_handler is err\_handler

True

geterrobj(...)

geterrobj()

Return the current object that defines floating-point error handling.

The error object contains all information that defines the error handling

behavior in NumPy. `geterrobj` is used internally by the other

functions that get and set error handling behavior (`geterr`, `seterr`,

`geterrcall`, `seterrcall`).

Returns

-------

errobj : list

The error object, a list containing three elements:

[internal numpy buffer size, error mask, error callback function].

The error mask is a single integer that holds the treatment information

on all four floating point errors. The information for each error type

is contained in three bits of the integer. If we print it in base 8, we

can see what treatment is set for "invalid", "under", "over", and

"divide" (in that order). The printed string can be interpreted with

\* 0 : 'ignore'

\* 1 : 'warn'

\* 2 : 'raise'

\* 3 : 'call'

\* 4 : 'print'

\* 5 : 'log'

See Also

--------

seterrobj, seterr, geterr, seterrcall, geterrcall

getbufsize, setbufsize

Notes

-----

For complete documentation of the types of floating-point exceptions and

treatment options, see `seterr`.

Examples

--------

>>> np.geterrobj() # first get the defaults

[8192, 521, None]

>>> def err\_handler(type, flag):

... print("Floating point error (%s), with flag %s" % (type, flag))

...

>>> old\_bufsize = np.setbufsize(20000)

>>> old\_err = np.seterr(divide='raise')

>>> old\_handler = np.seterrcall(err\_handler)

>>> np.geterrobj()

[8192, 521, <function err\_handler at 0x91dcaac>]

>>> old\_err = np.seterr(all='ignore')

>>> np.base\_repr(np.geterrobj()[1], 8)

'0'

>>> old\_err = np.seterr(divide='warn', over='log', under='call',

... invalid='print')

>>> np.base\_repr(np.geterrobj()[1], 8)

'4351'

gradient(f, \*varargs, \*\*kwargs)

Return the gradient of an N-dimensional array.

The gradient is computed using second order accurate central differences

in the interior points and either first or second order accurate one-sides

(forward or backwards) differences at the boundaries.

The returned gradient hence has the same shape as the input array.

Parameters

----------

f : array\_like

An N-dimensional array containing samples of a scalar function.

varargs : list of scalar or array, optional

Spacing between f values. Default unitary spacing for all dimensions.

Spacing can be specified using:

1. single scalar to specify a sample distance for all dimensions.

2. N scalars to specify a constant sample distance for each dimension.

i.e. `dx`, `dy`, `dz`, ...

3. N arrays to specify the coordinates of the values along each

dimension of F. The length of the array must match the size of

the corresponding dimension

4. Any combination of N scalars/arrays with the meaning of 2. and 3.

If `axis` is given, the number of varargs must equal the number of axes.

Default: 1.

edge\_order : {1, 2}, optional

Gradient is calculated using N-th order accurate differences

at the boundaries. Default: 1.

.. versionadded:: 1.9.1

axis : None or int or tuple of ints, optional

Gradient is calculated only along the given axis or axes

The default (axis = None) is to calculate the gradient for all the axes

of the input array. axis may be negative, in which case it counts from

the last to the first axis.

.. versionadded:: 1.11.0

Returns

-------

gradient : ndarray or list of ndarray

A set of ndarrays (or a single ndarray if there is only one dimension)

corresponding to the derivatives of f with respect to each dimension.

Each derivative has the same shape as f.

Examples

--------

>>> f = np.array([1, 2, 4, 7, 11, 16], dtype=float)

>>> np.gradient(f)

array([1. , 1.5, 2.5, 3.5, 4.5, 5. ])

>>> np.gradient(f, 2)

array([0.5 , 0.75, 1.25, 1.75, 2.25, 2.5 ])

Spacing can be also specified with an array that represents the coordinates

of the values F along the dimensions.

For instance a uniform spacing:

>>> x = np.arange(f.size)

>>> np.gradient(f, x)

array([1. , 1.5, 2.5, 3.5, 4.5, 5. ])

Or a non uniform one:

>>> x = np.array([0., 1., 1.5, 3.5, 4., 6.], dtype=float)

>>> np.gradient(f, x)

array([1. , 3. , 3.5, 6.7, 6.9, 2.5])

For two dimensional arrays, the return will be two arrays ordered by

axis. In this example the first array stands for the gradient in

rows and the second one in columns direction:

>>> np.gradient(np.array([[1, 2, 6], [3, 4, 5]], dtype=float))

[array([[ 2., 2., -1.],

[ 2., 2., -1.]]), array([[1. , 2.5, 4. ],

[1. , 1. , 1. ]])]

In this example the spacing is also specified:

uniform for axis=0 and non uniform for axis=1

>>> dx = 2.

>>> y = [1., 1.5, 3.5]

>>> np.gradient(np.array([[1, 2, 6], [3, 4, 5]], dtype=float), dx, y)

[array([[ 1. , 1. , -0.5],

[ 1. , 1. , -0.5]]), array([[2. , 2. , 2. ],

[2. , 1.7, 0.5]])]

It is possible to specify how boundaries are treated using `edge\_order`

>>> x = np.array([0, 1, 2, 3, 4])

>>> f = x\*\*2

>>> np.gradient(f, edge\_order=1)

array([1., 2., 4., 6., 7.])

>>> np.gradient(f, edge\_order=2)

array([0., 2., 4., 6., 8.])

The `axis` keyword can be used to specify a subset of axes of which the

gradient is calculated

>>> np.gradient(np.array([[1, 2, 6], [3, 4, 5]], dtype=float), axis=0)

array([[ 2., 2., -1.],

[ 2., 2., -1.]])

Notes

-----

Assuming that :math:`f\in C^{3}` (i.e., :math:`f` has at least 3 continuous

derivatives) and let :math:`h\_{\*}` be a non-homogeneous stepsize, we

minimize the "consistency error" :math:`\eta\_{i}` between the true gradient

and its estimate from a linear combination of the neighboring grid-points:

.. math::

\eta\_{i} = f\_{i}^{\left(1\right)} -

\left[ \alpha f\left(x\_{i}\right) +

\beta f\left(x\_{i} + h\_{d}\right) +

\gamma f\left(x\_{i}-h\_{s}\right)

\right]

By substituting :math:`f(x\_{i} + h\_{d})` and :math:`f(x\_{i} - h\_{s})`

with their Taylor series expansion, this translates into solving

the following the linear system:

.. math::

\left\{

\begin{array}{r}

\alpha+\beta+\gamma=0 \\

\beta h\_{d}-\gamma h\_{s}=1 \\

\beta h\_{d}^{2}+\gamma h\_{s}^{2}=0

\end{array}

\right.

The resulting approximation of :math:`f\_{i}^{(1)}` is the following:

.. math::

\hat f\_{i}^{(1)} =

\frac{

h\_{s}^{2}f\left(x\_{i} + h\_{d}\right)

+ \left(h\_{d}^{2} - h\_{s}^{2}\right)f\left(x\_{i}\right)

- h\_{d}^{2}f\left(x\_{i}-h\_{s}\right)}

{ h\_{s}h\_{d}\left(h\_{d} + h\_{s}\right)}

+ \mathcal{O}\left(\frac{h\_{d}h\_{s}^{2}

+ h\_{s}h\_{d}^{2}}{h\_{d}

+ h\_{s}}\right)

It is worth noting that if :math:`h\_{s}=h\_{d}`

(i.e., data are evenly spaced)

we find the standard second order approximation:

.. math::

\hat f\_{i}^{(1)}=

\frac{f\left(x\_{i+1}\right) - f\left(x\_{i-1}\right)}{2h}

+ \mathcal{O}\left(h^{2}\right)

With a similar procedure the forward/backward approximations used for

boundaries can be derived.

References

----------

.. [1] Quarteroni A., Sacco R., Saleri F. (2007) Numerical Mathematics

(Texts in Applied Mathematics). New York: Springer.

.. [2] Durran D. R. (1999) Numerical Methods for Wave Equations

in Geophysical Fluid Dynamics. New York: Springer.

.. [3] Fornberg B. (1988) Generation of Finite Difference Formulas on

Arbitrarily Spaced Grids,

Mathematics of Computation 51, no. 184 : 699-706.

`PDF <http://www.ams.org/journals/mcom/1988-51-184/

S0025-5718-1988-0935077-0/S0025-5718-1988-0935077-0.pdf>`\_.

hamming(M)

Return the Hamming window.

The Hamming window is a taper formed by using a weighted cosine.

Parameters

----------

M : int

Number of points in the output window. If zero or less, an

empty array is returned.

Returns

-------

out : ndarray

The window, with the maximum value normalized to one (the value

one appears only if the number of samples is odd).

See Also

--------

bartlett, blackman, hanning, kaiser

Notes

-----

The Hamming window is defined as

.. math:: w(n) = 0.54 - 0.46cos\left(\frac{2\pi{n}}{M-1}\right)

\qquad 0 \leq n \leq M-1

The Hamming was named for R. W. Hamming, an associate of J. W. Tukey

and is described in Blackman and Tukey. It was recommended for

smoothing the truncated autocovariance function in the time domain.

Most references to the Hamming window come from the signal processing

literature, where it is used as one of many windowing functions for

smoothing values. It is also known as an apodization (which means

"removing the foot", i.e. smoothing discontinuities at the beginning

and end of the sampled signal) or tapering function.

References

----------

.. [1] Blackman, R.B. and Tukey, J.W., (1958) The measurement of power

spectra, Dover Publications, New York.

.. [2] E.R. Kanasewich, "Time Sequence Analysis in Geophysics", The

University of Alberta Press, 1975, pp. 109-110.

.. [3] Wikipedia, "Window function",

https://en.wikipedia.org/wiki/Window\_function

.. [4] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling,

"Numerical Recipes", Cambridge University Press, 1986, page 425.

Examples

--------

>>> np.hamming(12)

array([ 0.08 , 0.15302337, 0.34890909, 0.60546483, 0.84123594, # may vary

0.98136677, 0.98136677, 0.84123594, 0.60546483, 0.34890909,

0.15302337, 0.08 ])

Plot the window and the frequency response:

>>> import matplotlib.pyplot as plt

>>> from numpy.fft import fft, fftshift

>>> window = np.hamming(51)

>>> plt.plot(window)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Hamming window")

Text(0.5, 1.0, 'Hamming window')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("Sample")

Text(0.5, 0, 'Sample')

>>> plt.show()

>>> plt.figure()

<Figure size 640x480 with 0 Axes>

>>> A = fft(window, 2048) / 25.5

>>> mag = np.abs(fftshift(A))

>>> freq = np.linspace(-0.5, 0.5, len(A))

>>> response = 20 \* np.log10(mag)

>>> response = np.clip(response, -100, 100)

>>> plt.plot(freq, response)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Frequency response of Hamming window")

Text(0.5, 1.0, 'Frequency response of Hamming window')

>>> plt.ylabel("Magnitude [dB]")

Text(0, 0.5, 'Magnitude [dB]')

>>> plt.xlabel("Normalized frequency [cycles per sample]")

Text(0.5, 0, 'Normalized frequency [cycles per sample]')

>>> plt.axis('tight')

...

>>> plt.show()

hanning(M)

Return the Hanning window.

The Hanning window is a taper formed by using a weighted cosine.

Parameters

----------

M : int

Number of points in the output window. If zero or less, an

empty array is returned.

Returns

-------

out : ndarray, shape(M,)

The window, with the maximum value normalized to one (the value

one appears only if `M` is odd).

See Also

--------

bartlett, blackman, hamming, kaiser

Notes

-----

The Hanning window is defined as

.. math:: w(n) = 0.5 - 0.5cos\left(\frac{2\pi{n}}{M-1}\right)

\qquad 0 \leq n \leq M-1

The Hanning was named for Julius von Hann, an Austrian meteorologist.

It is also known as the Cosine Bell. Some authors prefer that it be

called a Hann window, to help avoid confusion with the very similar

Hamming window.

Most references to the Hanning window come from the signal processing

literature, where it is used as one of many windowing functions for

smoothing values. It is also known as an apodization (which means

"removing the foot", i.e. smoothing discontinuities at the beginning

and end of the sampled signal) or tapering function.

References

----------

.. [1] Blackman, R.B. and Tukey, J.W., (1958) The measurement of power

spectra, Dover Publications, New York.

.. [2] E.R. Kanasewich, "Time Sequence Analysis in Geophysics",

The University of Alberta Press, 1975, pp. 106-108.

.. [3] Wikipedia, "Window function",

https://en.wikipedia.org/wiki/Window\_function

.. [4] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling,

"Numerical Recipes", Cambridge University Press, 1986, page 425.

Examples

--------

>>> np.hanning(12)

array([0. , 0.07937323, 0.29229249, 0.57115742, 0.82743037,

0.97974649, 0.97974649, 0.82743037, 0.57115742, 0.29229249,

0.07937323, 0. ])

Plot the window and its frequency response:

>>> import matplotlib.pyplot as plt

>>> from numpy.fft import fft, fftshift

>>> window = np.hanning(51)

>>> plt.plot(window)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Hann window")

Text(0.5, 1.0, 'Hann window')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("Sample")

Text(0.5, 0, 'Sample')

>>> plt.show()

>>> plt.figure()

<Figure size 640x480 with 0 Axes>

>>> A = fft(window, 2048) / 25.5

>>> mag = np.abs(fftshift(A))

>>> freq = np.linspace(-0.5, 0.5, len(A))

>>> with np.errstate(divide='ignore', invalid='ignore'):

... response = 20 \* np.log10(mag)

...

>>> response = np.clip(response, -100, 100)

>>> plt.plot(freq, response)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Frequency response of the Hann window")

Text(0.5, 1.0, 'Frequency response of the Hann window')

>>> plt.ylabel("Magnitude [dB]")

Text(0, 0.5, 'Magnitude [dB]')

>>> plt.xlabel("Normalized frequency [cycles per sample]")

Text(0.5, 0, 'Normalized frequency [cycles per sample]')

>>> plt.axis('tight')

...

>>> plt.show()

histogram(a, bins=10, range=None, normed=None, weights=None, density=None)

Compute the histogram of a set of data.

Parameters

----------

a : array\_like

Input data. The histogram is computed over the flattened array.

bins : int or sequence of scalars or str, optional

If `bins` is an int, it defines the number of equal-width

bins in the given range (10, by default). If `bins` is a

sequence, it defines a monotonically increasing array of bin edges,

including the rightmost edge, allowing for non-uniform bin widths.

.. versionadded:: 1.11.0

If `bins` is a string, it defines the method used to calculate the

optimal bin width, as defined by `histogram\_bin\_edges`.

range : (float, float), optional

The lower and upper range of the bins. If not provided, range

is simply ``(a.min(), a.max())``. Values outside the range are

ignored. The first element of the range must be less than or

equal to the second. `range` affects the automatic bin

computation as well. While bin width is computed to be optimal

based on the actual data within `range`, the bin count will fill

the entire range including portions containing no data.

normed : bool, optional

.. deprecated:: 1.6.0

This is equivalent to the `density` argument, but produces incorrect

results for unequal bin widths. It should not be used.

.. versionchanged:: 1.15.0

DeprecationWarnings are actually emitted.

weights : array\_like, optional

An array of weights, of the same shape as `a`. Each value in

`a` only contributes its associated weight towards the bin count

(instead of 1). If `density` is True, the weights are

normalized, so that the integral of the density over the range

remains 1.

density : bool, optional

If ``False``, the result will contain the number of samples in

each bin. If ``True``, the result is the value of the

probability \*density\* function at the bin, normalized such that

the \*integral\* over the range is 1. Note that the sum of the

histogram values will not be equal to 1 unless bins of unity

width are chosen; it is not a probability \*mass\* function.

Overrides the ``normed`` keyword if given.

Returns

-------

hist : array

The values of the histogram. See `density` and `weights` for a

description of the possible semantics.

bin\_edges : array of dtype float

Return the bin edges ``(length(hist)+1)``.

See Also

--------

histogramdd, bincount, searchsorted, digitize, histogram\_bin\_edges

Notes

-----

All but the last (righthand-most) bin is half-open. In other words,

if `bins` is::

[1, 2, 3, 4]

then the first bin is ``[1, 2)`` (including 1, but excluding 2) and

the second ``[2, 3)``. The last bin, however, is ``[3, 4]``, which

\*includes\* 4.

Examples

--------

>>> np.histogram([1, 2, 1], bins=[0, 1, 2, 3])

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

>>> np.histogram(np.arange(4), bins=np.arange(5), density=True)

(array([0.25, 0.25, 0.25, 0.25]), array([0, 1, 2, 3, 4]))

>>> np.histogram([[1, 2, 1], [1, 0, 1]], bins=[0,1,2,3])

(array([1, 4, 1]), array([0, 1, 2, 3]))

>>> a = np.arange(5)

>>> hist, bin\_edges = np.histogram(a, density=True)

>>> hist

array([0.5, 0. , 0.5, 0. , 0. , 0.5, 0. , 0.5, 0. , 0.5])

>>> hist.sum()

2.4999999999999996

>>> np.sum(hist \* np.diff(bin\_edges))

1.0

.. versionadded:: 1.11.0

Automated Bin Selection Methods example, using 2 peak random data

with 2000 points:

>>> import matplotlib.pyplot as plt

>>> rng = np.random.RandomState(10) # deterministic random data

>>> a = np.hstack((rng.normal(size=1000),

... rng.normal(loc=5, scale=2, size=1000)))

>>> \_ = plt.hist(a, bins='auto') # arguments are passed to np.histogram

>>> plt.title("Histogram with 'auto' bins")

Text(0.5, 1.0, "Histogram with 'auto' bins")

>>> plt.show()

histogram2d(x, y, bins=10, range=None, normed=None, weights=None, density=None)

Compute the bi-dimensional histogram of two data samples.

Parameters

----------

x : array\_like, shape (N,)

An array containing the x coordinates of the points to be

histogrammed.

y : array\_like, shape (N,)

An array containing the y coordinates of the points to be

histogrammed.

bins : int or array\_like or [int, int] or [array, array], optional

The bin specification:

\* If int, the number of bins for the two dimensions (nx=ny=bins).

\* If array\_like, the bin edges for the two dimensions

(x\_edges=y\_edges=bins).

\* If [int, int], the number of bins in each dimension

(nx, ny = bins).

\* If [array, array], the bin edges in each dimension

(x\_edges, y\_edges = bins).

\* A combination [int, array] or [array, int], where int

is the number of bins and array is the bin edges.

range : array\_like, shape(2,2), optional

The leftmost and rightmost edges of the bins along each dimension

(if not specified explicitly in the `bins` parameters):

``[[xmin, xmax], [ymin, ymax]]``. All values outside of this range

will be considered outliers and not tallied in the histogram.

density : bool, optional

If False, the default, returns the number of samples in each bin.

If True, returns the probability \*density\* function at the bin,

``bin\_count / sample\_count / bin\_area``.

normed : bool, optional

An alias for the density argument that behaves identically. To avoid

confusion with the broken normed argument to `histogram`, `density`

should be preferred.

weights : array\_like, shape(N,), optional

An array of values ``w\_i`` weighing each sample ``(x\_i, y\_i)``.

Weights are normalized to 1 if `normed` is True. If `normed` is

False, the values of the returned histogram are equal to the sum of

the weights belonging to the samples falling into each bin.

Returns

-------

H : ndarray, shape(nx, ny)

The bi-dimensional histogram of samples `x` and `y`. Values in `x`

are histogrammed along the first dimension and values in `y` are

histogrammed along the second dimension.

xedges : ndarray, shape(nx+1,)

The bin edges along the first dimension.

yedges : ndarray, shape(ny+1,)

The bin edges along the second dimension.

See Also

--------

histogram : 1D histogram

histogramdd : Multidimensional histogram

Notes

-----

When `normed` is True, then the returned histogram is the sample

density, defined such that the sum over bins of the product

``bin\_value \* bin\_area`` is 1.

Please note that the histogram does not follow the Cartesian convention

where `x` values are on the abscissa and `y` values on the ordinate

axis. Rather, `x` is histogrammed along the first dimension of the

array (vertical), and `y` along the second dimension of the array

(horizontal). This ensures compatibility with `histogramdd`.

Examples

--------

>>> from matplotlib.image import NonUniformImage

>>> import matplotlib.pyplot as plt

Construct a 2-D histogram with variable bin width. First define the bin

edges:

>>> xedges = [0, 1, 3, 5]

>>> yedges = [0, 2, 3, 4, 6]

Next we create a histogram H with random bin content:

>>> x = np.random.normal(2, 1, 100)

>>> y = np.random.normal(1, 1, 100)

>>> H, xedges, yedges = np.histogram2d(x, y, bins=(xedges, yedges))

>>> H = H.T # Let each row list bins with common y range.

:func:`imshow <matplotlib.pyplot.imshow>` can only display square bins:

>>> fig = plt.figure(figsize=(7, 3))

>>> ax = fig.add\_subplot(131, title='imshow: square bins')

>>> plt.imshow(H, interpolation='nearest', origin='low',

... extent=[xedges[0], xedges[-1], yedges[0], yedges[-1]])

<matplotlib.image.AxesImage object at 0x...>

:func:`pcolormesh <matplotlib.pyplot.pcolormesh>` can display actual edges:

>>> ax = fig.add\_subplot(132, title='pcolormesh: actual edges',

... aspect='equal')

>>> X, Y = np.meshgrid(xedges, yedges)

>>> ax.pcolormesh(X, Y, H)

<matplotlib.collections.QuadMesh object at 0x...>

:class:`NonUniformImage <matplotlib.image.NonUniformImage>` can be used to

display actual bin edges with interpolation:

>>> ax = fig.add\_subplot(133, title='NonUniformImage: interpolated',

... aspect='equal', xlim=xedges[[0, -1]], ylim=yedges[[0, -1]])

>>> im = NonUniformImage(ax, interpolation='bilinear')

>>> xcenters = (xedges[:-1] + xedges[1:]) / 2

>>> ycenters = (yedges[:-1] + yedges[1:]) / 2

>>> im.set\_data(xcenters, ycenters, H)

>>> ax.images.append(im)

>>> plt.show()

histogram\_bin\_edges(a, bins=10, range=None, weights=None)

Function to calculate only the edges of the bins used by the `histogram`

function.

Parameters

----------

a : array\_like

Input data. The histogram is computed over the flattened array.

bins : int or sequence of scalars or str, optional

If `bins` is an int, it defines the number of equal-width

bins in the given range (10, by default). If `bins` is a

sequence, it defines the bin edges, including the rightmost

edge, allowing for non-uniform bin widths.

If `bins` is a string from the list below, `histogram\_bin\_edges` will use

the method chosen to calculate the optimal bin width and

consequently the number of bins (see `Notes` for more detail on

the estimators) from the data that falls within the requested

range. While the bin width will be optimal for the actual data

in the range, the number of bins will be computed to fill the

entire range, including the empty portions. For visualisation,

using the 'auto' option is suggested. Weighted data is not

supported for automated bin size selection.

'auto'

Maximum of the 'sturges' and 'fd' estimators. Provides good

all around performance.

'fd' (Freedman Diaconis Estimator)

Robust (resilient to outliers) estimator that takes into

account data variability and data size.

'doane'

An improved version of Sturges' estimator that works better

with non-normal datasets.

'scott'

Less robust estimator that that takes into account data

variability and data size.

'stone'

Estimator based on leave-one-out cross-validation estimate of

the integrated squared error. Can be regarded as a generalization

of Scott's rule.

'rice'

Estimator does not take variability into account, only data

size. Commonly overestimates number of bins required.

'sturges'

R's default method, only accounts for data size. Only

optimal for gaussian data and underestimates number of bins

for large non-gaussian datasets.

'sqrt'

Square root (of data size) estimator, used by Excel and

other programs for its speed and simplicity.

range : (float, float), optional

The lower and upper range of the bins. If not provided, range

is simply ``(a.min(), a.max())``. Values outside the range are

ignored. The first element of the range must be less than or

equal to the second. `range` affects the automatic bin

computation as well. While bin width is computed to be optimal

based on the actual data within `range`, the bin count will fill

the entire range including portions containing no data.

weights : array\_like, optional

An array of weights, of the same shape as `a`. Each value in

`a` only contributes its associated weight towards the bin count

(instead of 1). This is currently not used by any of the bin estimators,

but may be in the future.

Returns

-------

bin\_edges : array of dtype float

The edges to pass into `histogram`

See Also

--------

histogram

Notes

-----

The methods to estimate the optimal number of bins are well founded

in literature, and are inspired by the choices R provides for

histogram visualisation. Note that having the number of bins

proportional to :math:`n^{1/3}` is asymptotically optimal, which is

why it appears in most estimators. These are simply plug-in methods

that give good starting points for number of bins. In the equations

below, :math:`h` is the binwidth and :math:`n\_h` is the number of

bins. All estimators that compute bin counts are recast to bin width

using the `ptp` of the data. The final bin count is obtained from

``np.round(np.ceil(range / h))``.

'auto' (maximum of the 'sturges' and 'fd' estimators)

A compromise to get a good value. For small datasets the Sturges

value will usually be chosen, while larger datasets will usually

default to FD. Avoids the overly conservative behaviour of FD

and Sturges for small and large datasets respectively.

Switchover point is usually :math:`a.size \approx 1000`.

'fd' (Freedman Diaconis Estimator)

.. math:: h = 2 \frac{IQR}{n^{1/3}}

The binwidth is proportional to the interquartile range (IQR)

and inversely proportional to cube root of a.size. Can be too

conservative for small datasets, but is quite good for large

datasets. The IQR is very robust to outliers.

'scott'

.. math:: h = \sigma \sqrt[3]{\frac{24 \* \sqrt{\pi}}{n}}

The binwidth is proportional to the standard deviation of the

data and inversely proportional to cube root of ``x.size``. Can

be too conservative for small datasets, but is quite good for

large datasets. The standard deviation is not very robust to

outliers. Values are very similar to the Freedman-Diaconis

estimator in the absence of outliers.

'rice'

.. math:: n\_h = 2n^{1/3}

The number of bins is only proportional to cube root of

``a.size``. It tends to overestimate the number of bins and it

does not take into account data variability.

'sturges'

.. math:: n\_h = \log \_{2}n+1

The number of bins is the base 2 log of ``a.size``. This

estimator assumes normality of data and is too conservative for

larger, non-normal datasets. This is the default method in R's

``hist`` method.

'doane'

.. math:: n\_h = 1 + \log\_{2}(n) +

\log\_{2}(1 + \frac{|g\_1|}{\sigma\_{g\_1}})

g\_1 = mean[(\frac{x - \mu}{\sigma})^3]

\sigma\_{g\_1} = \sqrt{\frac{6(n - 2)}{(n + 1)(n + 3)}}

An improved version of Sturges' formula that produces better

estimates for non-normal datasets. This estimator attempts to

account for the skew of the data.

'sqrt'

.. math:: n\_h = \sqrt n

The simplest and fastest estimator. Only takes into account the

data size.

Examples

--------

>>> arr = np.array([0, 0, 0, 1, 2, 3, 3, 4, 5])

>>> np.histogram\_bin\_edges(arr, bins='auto', range=(0, 1))

array([0. , 0.25, 0.5 , 0.75, 1. ])

>>> np.histogram\_bin\_edges(arr, bins=2)

array([0. , 2.5, 5. ])

For consistency with histogram, an array of pre-computed bins is

passed through unmodified:

>>> np.histogram\_bin\_edges(arr, [1, 2])

array([1, 2])

This function allows one set of bins to be computed, and reused across

multiple histograms:

>>> shared\_bins = np.histogram\_bin\_edges(arr, bins='auto')

>>> shared\_bins

array([0., 1., 2., 3., 4., 5.])

>>> group\_id = np.array([0, 1, 1, 0, 1, 1, 0, 1, 1])

>>> hist\_0, \_ = np.histogram(arr[group\_id == 0], bins=shared\_bins)

>>> hist\_1, \_ = np.histogram(arr[group\_id == 1], bins=shared\_bins)

>>> hist\_0; hist\_1

array([1, 1, 0, 1, 0])

array([2, 0, 1, 1, 2])

Which gives more easily comparable results than using separate bins for

each histogram:

>>> hist\_0, bins\_0 = np.histogram(arr[group\_id == 0], bins='auto')

>>> hist\_1, bins\_1 = np.histogram(arr[group\_id == 1], bins='auto')

>>> hist\_0; hist\_1

array([1, 1, 1])

array([2, 1, 1, 2])

>>> bins\_0; bins\_1

array([0., 1., 2., 3.])

array([0. , 1.25, 2.5 , 3.75, 5. ])

histogramdd(sample, bins=10, range=None, normed=None, weights=None, density=None)

Compute the multidimensional histogram of some data.

Parameters

----------

sample : (N, D) array, or (D, N) array\_like

The data to be histogrammed.

Note the unusual interpretation of sample when an array\_like:

\* When an array, each row is a coordinate in a D-dimensional space -

such as ``histogramgramdd(np.array([p1, p2, p3]))``.

\* When an array\_like, each element is the list of values for single

coordinate - such as ``histogramgramdd((X, Y, Z))``.

The first form should be preferred.

bins : sequence or int, optional

The bin specification:

\* A sequence of arrays describing the monotonically increasing bin

edges along each dimension.

\* The number of bins for each dimension (nx, ny, ... =bins)

\* The number of bins for all dimensions (nx=ny=...=bins).

range : sequence, optional

A sequence of length D, each an optional (lower, upper) tuple giving

the outer bin edges to be used if the edges are not given explicitly in

`bins`.

An entry of None in the sequence results in the minimum and maximum

values being used for the corresponding dimension.

The default, None, is equivalent to passing a tuple of D None values.

density : bool, optional

If False, the default, returns the number of samples in each bin.

If True, returns the probability \*density\* function at the bin,

``bin\_count / sample\_count / bin\_volume``.

normed : bool, optional

An alias for the density argument that behaves identically. To avoid

confusion with the broken normed argument to `histogram`, `density`

should be preferred.

weights : (N,) array\_like, optional

An array of values `w\_i` weighing each sample `(x\_i, y\_i, z\_i, ...)`.

Weights are normalized to 1 if normed is True. If normed is False,

the values of the returned histogram are equal to the sum of the

weights belonging to the samples falling into each bin.

Returns

-------

H : ndarray

The multidimensional histogram of sample x. See normed and weights

for the different possible semantics.

edges : list

A list of D arrays describing the bin edges for each dimension.

See Also

--------

histogram: 1-D histogram

histogram2d: 2-D histogram

Examples

--------

>>> r = np.random.randn(100,3)

>>> H, edges = np.histogramdd(r, bins = (5, 8, 4))

>>> H.shape, edges[0].size, edges[1].size, edges[2].size

((5, 8, 4), 6, 9, 5)

hsplit(ary, indices\_or\_sections)

Split an array into multiple sub-arrays horizontally (column-wise).

Please refer to the `split` documentation. `hsplit` is equivalent

to `split` with ``axis=1``, the array is always split along the second

axis regardless of the array dimension.

See Also

--------

split : Split an array into multiple sub-arrays of equal size.

Examples

--------

>>> x = np.arange(16.0).reshape(4, 4)

>>> x

array([[ 0., 1., 2., 3.],

[ 4., 5., 6., 7.],

[ 8., 9., 10., 11.],

[12., 13., 14., 15.]])

>>> np.hsplit(x, 2)

[array([[ 0., 1.],

[ 4., 5.],

[ 8., 9.],

[12., 13.]]),

array([[ 2., 3.],

[ 6., 7.],

[10., 11.],

[14., 15.]])]

>>> np.hsplit(x, np.array([3, 6]))

[array([[ 0., 1., 2.],

[ 4., 5., 6.],

[ 8., 9., 10.],

[12., 13., 14.]]),

array([[ 3.],

[ 7.],

[11.],

[15.]]),

array([], shape=(4, 0), dtype=float64)]

With a higher dimensional array the split is still along the second axis.

>>> x = np.arange(8.0).reshape(2, 2, 2)

>>> x

array([[[0., 1.],

[2., 3.]],

[[4., 5.],

[6., 7.]]])

>>> np.hsplit(x, 2)

[array([[[0., 1.]],

[[4., 5.]]]),

array([[[2., 3.]],

[[6., 7.]]])]

hstack(tup)

Stack arrays in sequence horizontally (column wise).

This is equivalent to concatenation along the second axis, except for 1-D

arrays where it concatenates along the first axis. Rebuilds arrays divided

by `hsplit`.

This function makes most sense for arrays with up to 3 dimensions. For

instance, for pixel-data with a height (first axis), width (second axis),

and r/g/b channels (third axis). The functions `concatenate`, `stack` and

`block` provide more general stacking and concatenation operations.

Parameters

----------

tup : sequence of ndarrays

The arrays must have the same shape along all but the second axis,

except 1-D arrays which can be any length.

Returns

-------

stacked : ndarray

The array formed by stacking the given arrays.

See Also

--------

stack : Join a sequence of arrays along a new axis.

vstack : Stack arrays in sequence vertically (row wise).

dstack : Stack arrays in sequence depth wise (along third axis).

concatenate : Join a sequence of arrays along an existing axis.

hsplit : Split array along second axis.

block : Assemble arrays from blocks.

Examples

--------

>>> a = np.array((1,2,3))

>>> b = np.array((2,3,4))

>>> np.hstack((a,b))

array([1, 2, 3, 2, 3, 4])

>>> a = np.array([[1],[2],[3]])

>>> b = np.array([[2],[3],[4]])

>>> np.hstack((a,b))

array([[1, 2],

[2, 3],

[3, 4]])

i0(x)

Modified Bessel function of the first kind, order 0.

Usually denoted :math:`I\_0`. This function does broadcast, but will \*not\*

"up-cast" int dtype arguments unless accompanied by at least one float or

complex dtype argument (see Raises below).

Parameters

----------

x : array\_like, dtype float or complex

Argument of the Bessel function.

Returns

-------

out : ndarray, shape = x.shape, dtype = x.dtype

The modified Bessel function evaluated at each of the elements of `x`.

Raises

------

TypeError: array cannot be safely cast to required type

If argument consists exclusively of int dtypes.

See Also

--------

scipy.special.i0, scipy.special.iv, scipy.special.ive

Notes

-----

The scipy implementation is recommended over this function: it is a

proper ufunc written in C, and more than an order of magnitude faster.

We use the algorithm published by Clenshaw [1]\_ and referenced by

Abramowitz and Stegun [2]\_, for which the function domain is

partitioned into the two intervals [0,8] and (8,inf), and Chebyshev

polynomial expansions are employed in each interval. Relative error on

the domain [0,30] using IEEE arithmetic is documented [3]\_ as having a

peak of 5.8e-16 with an rms of 1.4e-16 (n = 30000).

References

----------

.. [1] C. W. Clenshaw, "Chebyshev series for mathematical functions", in

\*National Physical Laboratory Mathematical Tables\*, vol. 5, London:

Her Majesty's Stationery Office, 1962.

.. [2] M. Abramowitz and I. A. Stegun, \*Handbook of Mathematical

Functions\*, 10th printing, New York: Dover, 1964, pp. 379.

http://www.math.sfu.ca/~cbm/aands/page\_379.htm

.. [3] http://kobesearch.cpan.org/htdocs/Math-Cephes/Math/Cephes.html

Examples

--------

>>> np.i0(0.)

array(1.0) # may vary

>>> np.i0([0., 1. + 2j])

array([ 1.00000000+0.j , 0.18785373+0.64616944j]) # may vary

identity(n, dtype=None)

Return the identity array.

The identity array is a square array with ones on

the main diagonal.

Parameters

----------

n : int

Number of rows (and columns) in `n` x `n` output.

dtype : data-type, optional

Data-type of the output. Defaults to ``float``.

Returns

-------

out : ndarray

`n` x `n` array with its main diagonal set to one,

and all other elements 0.

Examples

--------

>>> np.identity(3)

array([[1., 0., 0.],

[0., 1., 0.],

[0., 0., 1.]])

imag(val)

Return the imaginary part of the complex argument.

Parameters

----------

val : array\_like

Input array.

Returns

-------

out : ndarray or scalar

The imaginary component of the complex argument. If `val` is real,

the type of `val` is used for the output. If `val` has complex

elements, the returned type is float.

See Also

--------

real, angle, real\_if\_close

Examples

--------

>>> a = np.array([1+2j, 3+4j, 5+6j])

>>> a.imag

array([2., 4., 6.])

>>> a.imag = np.array([8, 10, 12])

>>> a

array([1. +8.j, 3.+10.j, 5.+12.j])

>>> np.imag(1 + 1j)

1.0

in1d(ar1, ar2, assume\_unique=False, invert=False)

Test whether each element of a 1-D array is also present in a second array.

Returns a boolean array the same length as `ar1` that is True

where an element of `ar1` is in `ar2` and False otherwise.

We recommend using :func:`isin` instead of `in1d` for new code.

Parameters

----------

ar1 : (M,) array\_like

Input array.

ar2 : array\_like

The values against which to test each value of `ar1`.

assume\_unique : bool, optional

If True, the input arrays are both assumed to be unique, which

can speed up the calculation. Default is False.

invert : bool, optional

If True, the values in the returned array are inverted (that is,

False where an element of `ar1` is in `ar2` and True otherwise).

Default is False. ``np.in1d(a, b, invert=True)`` is equivalent

to (but is faster than) ``np.invert(in1d(a, b))``.

.. versionadded:: 1.8.0

Returns

-------

in1d : (M,) ndarray, bool

The values `ar1[in1d]` are in `ar2`.

See Also

--------

isin : Version of this function that preserves the

shape of ar1.

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Notes

-----

`in1d` can be considered as an element-wise function version of the

python keyword `in`, for 1-D sequences. ``in1d(a, b)`` is roughly

equivalent to ``np.array([item in b for item in a])``.

However, this idea fails if `ar2` is a set, or similar (non-sequence)

container: As ``ar2`` is converted to an array, in those cases

``asarray(ar2)`` is an object array rather than the expected array of

contained values.

.. versionadded:: 1.4.0

Examples

--------

>>> test = np.array([0, 1, 2, 5, 0])

>>> states = [0, 2]

>>> mask = np.in1d(test, states)

>>> mask

array([ True, False, True, False, True])

>>> test[mask]

array([0, 2, 0])

>>> mask = np.in1d(test, states, invert=True)

>>> mask

array([False, True, False, True, False])

>>> test[mask]

array([1, 5])

indices(dimensions, dtype=<class 'int'>, sparse=False)

Return an array representing the indices of a grid.

Compute an array where the subarrays contain index values 0, 1, ...

varying only along the corresponding axis.

Parameters

----------

dimensions : sequence of ints

The shape of the grid.

dtype : dtype, optional

Data type of the result.

sparse : boolean, optional

Return a sparse representation of the grid instead of a dense

representation. Default is False.

.. versionadded:: 1.17

Returns

-------

grid : one ndarray or tuple of ndarrays

If sparse is False:

Returns one array of grid indices,

``grid.shape = (len(dimensions),) + tuple(dimensions)``.

If sparse is True:

Returns a tuple of arrays, with

``grid[i].shape = (1, ..., 1, dimensions[i], 1, ..., 1)`` with

dimensions[i] in the ith place

See Also

--------

mgrid, ogrid, meshgrid

Notes

-----

The output shape in the dense case is obtained by prepending the number

of dimensions in front of the tuple of dimensions, i.e. if `dimensions`

is a tuple ``(r0, ..., rN-1)`` of length ``N``, the output shape is

``(N, r0, ..., rN-1)``.

The subarrays ``grid[k]`` contains the N-D array of indices along the

``k-th`` axis. Explicitly::

grid[k, i0, i1, ..., iN-1] = ik

Examples

--------

>>> grid = np.indices((2, 3))

>>> grid.shape

(2, 2, 3)

>>> grid[0] # row indices

array([[0, 0, 0],

[1, 1, 1]])

>>> grid[1] # column indices

array([[0, 1, 2],

[0, 1, 2]])

The indices can be used as an index into an array.

>>> x = np.arange(20).reshape(5, 4)

>>> row, col = np.indices((2, 3))

>>> x[row, col]

array([[0, 1, 2],

[4, 5, 6]])

Note that it would be more straightforward in the above example to

extract the required elements directly with ``x[:2, :3]``.

If sparse is set to true, the grid will be returned in a sparse

representation.

>>> i, j = np.indices((2, 3), sparse=True)

>>> i.shape

(2, 1)

>>> j.shape

(1, 3)

>>> i # row indices

array([[0],

[1]])

>>> j # column indices

array([[0, 1, 2]])

info(object=None, maxwidth=76, output=<idlelib.run.PseudoOutputFile object at 0x03FD4690>, toplevel='numpy')

Get help information for a function, class, or module.

Parameters

----------

object : object or str, optional

Input object or name to get information about. If `object` is a

numpy object, its docstring is given. If it is a string, available

modules are searched for matching objects. If None, information

about `info` itself is returned.

maxwidth : int, optional

Printing width.

output : file like object, optional

File like object that the output is written to, default is

``stdout``. The object has to be opened in 'w' or 'a' mode.

toplevel : str, optional

Start search at this level.

See Also

--------

source, lookfor

Notes

-----

When used interactively with an object, ``np.info(obj)`` is equivalent

to ``help(obj)`` on the Python prompt or ``obj?`` on the IPython

prompt.

Examples

--------

>>> np.info(np.polyval) # doctest: +SKIP

polyval(p, x)

Evaluate the polynomial p at x.

...

When using a string for `object` it is possible to get multiple results.

>>> np.info('fft') # doctest: +SKIP

\*\*\* Found in numpy \*\*\*

Core FFT routines

...

\*\*\* Found in numpy.fft \*\*\*

fft(a, n=None, axis=-1)

...

\*\*\* Repeat reference found in numpy.fft.fftpack \*\*\*

\*\*\* Total of 3 references found. \*\*\*

inner(...)

inner(a, b)

Inner product of two arrays.

Ordinary inner product of vectors for 1-D arrays (without complex

conjugation), in higher dimensions a sum product over the last axes.

Parameters

----------

a, b : array\_like

If `a` and `b` are nonscalar, their last dimensions must match.

Returns

-------

out : ndarray

`out.shape = a.shape[:-1] + b.shape[:-1]`

Raises

------

ValueError

If the last dimension of `a` and `b` has different size.

See Also

--------

tensordot : Sum products over arbitrary axes.

dot : Generalised matrix product, using second last dimension of `b`.

einsum : Einstein summation convention.

Notes

-----

For vectors (1-D arrays) it computes the ordinary inner-product::

np.inner(a, b) = sum(a[:]\*b[:])

More generally, if `ndim(a) = r > 0` and `ndim(b) = s > 0`::

np.inner(a, b) = np.tensordot(a, b, axes=(-1,-1))

or explicitly::

np.inner(a, b)[i0,...,ir-1,j0,...,js-1]

= sum(a[i0,...,ir-1,:]\*b[j0,...,js-1,:])

In addition `a` or `b` may be scalars, in which case::

np.inner(a,b) = a\*b

Examples

--------

Ordinary inner product for vectors:

>>> a = np.array([1,2,3])

>>> b = np.array([0,1,0])

>>> np.inner(a, b)

2

A multidimensional example:

>>> a = np.arange(24).reshape((2,3,4))

>>> b = np.arange(4)

>>> np.inner(a, b)

array([[ 14, 38, 62],

[ 86, 110, 134]])

An example where `b` is a scalar:

>>> np.inner(np.eye(2), 7)

array([[7., 0.],

[0., 7.]])

insert(arr, obj, values, axis=None)

Insert values along the given axis before the given indices.

Parameters

----------

arr : array\_like

Input array.

obj : int, slice or sequence of ints

Object that defines the index or indices before which `values` is

inserted.

.. versionadded:: 1.8.0

Support for multiple insertions when `obj` is a single scalar or a

sequence with one element (similar to calling insert multiple

times).

values : array\_like

Values to insert into `arr`. If the type of `values` is different

from that of `arr`, `values` is converted to the type of `arr`.

`values` should be shaped so that ``arr[...,obj,...] = values``

is legal.

axis : int, optional

Axis along which to insert `values`. If `axis` is None then `arr`

is flattened first.

Returns

-------

out : ndarray

A copy of `arr` with `values` inserted. Note that `insert`

does not occur in-place: a new array is returned. If

`axis` is None, `out` is a flattened array.

See Also

--------

append : Append elements at the end of an array.

concatenate : Join a sequence of arrays along an existing axis.

delete : Delete elements from an array.

Notes

-----

Note that for higher dimensional inserts `obj=0` behaves very different

from `obj=[0]` just like `arr[:,0,:] = values` is different from

`arr[:,[0],:] = values`.

Examples

--------

>>> a = np.array([[1, 1], [2, 2], [3, 3]])

>>> a

array([[1, 1],

[2, 2],

[3, 3]])

>>> np.insert(a, 1, 5)

array([1, 5, 1, ..., 2, 3, 3])

>>> np.insert(a, 1, 5, axis=1)

array([[1, 5, 1],

[2, 5, 2],

[3, 5, 3]])

Difference between sequence and scalars:

>>> np.insert(a, [1], [[1],[2],[3]], axis=1)

array([[1, 1, 1],

[2, 2, 2],

[3, 3, 3]])

>>> np.array\_equal(np.insert(a, 1, [1, 2, 3], axis=1),

... np.insert(a, [1], [[1],[2],[3]], axis=1))

True

>>> b = a.flatten()

>>> b

array([1, 1, 2, 2, 3, 3])

>>> np.insert(b, [2, 2], [5, 6])

array([1, 1, 5, ..., 2, 3, 3])

>>> np.insert(b, slice(2, 4), [5, 6])

array([1, 1, 5, ..., 2, 3, 3])

>>> np.insert(b, [2, 2], [7.13, False]) # type casting

array([1, 1, 7, ..., 2, 3, 3])

>>> x = np.arange(8).reshape(2, 4)

>>> idx = (1, 3)

>>> np.insert(x, idx, 999, axis=1)

array([[ 0, 999, 1, 2, 999, 3],

[ 4, 999, 5, 6, 999, 7]])

int\_asbuffer(...)

interp(x, xp, fp, left=None, right=None, period=None)

One-dimensional linear interpolation.

Returns the one-dimensional piecewise linear interpolant to a function

with given discrete data points (`xp`, `fp`), evaluated at `x`.

Parameters

----------

x : array\_like

The x-coordinates at which to evaluate the interpolated values.

xp : 1-D sequence of floats

The x-coordinates of the data points, must be increasing if argument

`period` is not specified. Otherwise, `xp` is internally sorted after

normalizing the periodic boundaries with ``xp = xp % period``.

fp : 1-D sequence of float or complex

The y-coordinates of the data points, same length as `xp`.

left : optional float or complex corresponding to fp

Value to return for `x < xp[0]`, default is `fp[0]`.

right : optional float or complex corresponding to fp

Value to return for `x > xp[-1]`, default is `fp[-1]`.

period : None or float, optional

A period for the x-coordinates. This parameter allows the proper

interpolation of angular x-coordinates. Parameters `left` and `right`

are ignored if `period` is specified.

.. versionadded:: 1.10.0

Returns

-------

y : float or complex (corresponding to fp) or ndarray

The interpolated values, same shape as `x`.

Raises

------

ValueError

If `xp` and `fp` have different length

If `xp` or `fp` are not 1-D sequences

If `period == 0`

Notes

-----

Does not check that the x-coordinate sequence `xp` is increasing.

If `xp` is not increasing, the results are nonsense.

A simple check for increasing is::

np.all(np.diff(xp) > 0)

Examples

--------

>>> xp = [1, 2, 3]

>>> fp = [3, 2, 0]

>>> np.interp(2.5, xp, fp)

1.0

>>> np.interp([0, 1, 1.5, 2.72, 3.14], xp, fp)

array([3. , 3. , 2.5 , 0.56, 0. ])

>>> UNDEF = -99.0

>>> np.interp(3.14, xp, fp, right=UNDEF)

-99.0

Plot an interpolant to the sine function:

>>> x = np.linspace(0, 2\*np.pi, 10)

>>> y = np.sin(x)

>>> xvals = np.linspace(0, 2\*np.pi, 50)

>>> yinterp = np.interp(xvals, x, y)

>>> import matplotlib.pyplot as plt

>>> plt.plot(x, y, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.plot(xvals, yinterp, '-x')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.show()

Interpolation with periodic x-coordinates:

>>> x = [-180, -170, -185, 185, -10, -5, 0, 365]

>>> xp = [190, -190, 350, -350]

>>> fp = [5, 10, 3, 4]

>>> np.interp(x, xp, fp, period=360)

array([7.5 , 5. , 8.75, 6.25, 3. , 3.25, 3.5 , 3.75])

Complex interpolation:

>>> x = [1.5, 4.0]

>>> xp = [2,3,5]

>>> fp = [1.0j, 0, 2+3j]

>>> np.interp(x, xp, fp)

array([0.+1.j , 1.+1.5j])

intersect1d(ar1, ar2, assume\_unique=False, return\_indices=False)

Find the intersection of two arrays.

Return the sorted, unique values that are in both of the input arrays.

Parameters

----------

ar1, ar2 : array\_like

Input arrays. Will be flattened if not already 1D.

assume\_unique : bool

If True, the input arrays are both assumed to be unique, which

can speed up the calculation. Default is False.

return\_indices : bool

If True, the indices which correspond to the intersection of the two

arrays are returned. The first instance of a value is used if there are

multiple. Default is False.

.. versionadded:: 1.15.0

Returns

-------

intersect1d : ndarray

Sorted 1D array of common and unique elements.

comm1 : ndarray

The indices of the first occurrences of the common values in `ar1`.

Only provided if `return\_indices` is True.

comm2 : ndarray

The indices of the first occurrences of the common values in `ar2`.

Only provided if `return\_indices` is True.

See Also

--------

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Examples

--------

>>> np.intersect1d([1, 3, 4, 3], [3, 1, 2, 1])

array([1, 3])

To intersect more than two arrays, use functools.reduce:

>>> from functools import reduce

>>> reduce(np.intersect1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))

array([3])

To return the indices of the values common to the input arrays

along with the intersected values:

>>> x = np.array([1, 1, 2, 3, 4])

>>> y = np.array([2, 1, 4, 6])

>>> xy, x\_ind, y\_ind = np.intersect1d(x, y, return\_indices=True)

>>> x\_ind, y\_ind

(array([0, 2, 4]), array([1, 0, 2]))

>>> xy, x[x\_ind], y[y\_ind]

(array([1, 2, 4]), array([1, 2, 4]), array([1, 2, 4]))

ipmt(rate, per, nper, pv, fv=0, when='end')

Compute the interest portion of a payment.

Parameters

----------

rate : scalar or array\_like of shape(M, )

Rate of interest as decimal (not per cent) per period

per : scalar or array\_like of shape(M, )

Interest paid against the loan changes during the life or the loan.

The `per` is the payment period to calculate the interest amount.

nper : scalar or array\_like of shape(M, )

Number of compounding periods

pv : scalar or array\_like of shape(M, )

Present value

fv : scalar or array\_like of shape(M, ), optional

Future value

when : {{'begin', 1}, {'end', 0}}, {string, int}, optional

When payments are due ('begin' (1) or 'end' (0)).

Defaults to {'end', 0}.

Returns

-------

out : ndarray

Interest portion of payment. If all input is scalar, returns a scalar

float. If any input is array\_like, returns interest payment for each

input element. If multiple inputs are array\_like, they all must have

the same shape.

See Also

--------

ppmt, pmt, pv

Notes

-----

The total payment is made up of payment against principal plus interest.

``pmt = ppmt + ipmt``

Examples

--------

What is the amortization schedule for a 1 year loan of $2500 at

8.24% interest per year compounded monthly?

>>> principal = 2500.00

The 'per' variable represents the periods of the loan. Remember that

financial equations start the period count at 1!

>>> per = np.arange(1\*12) + 1

>>> ipmt = np.ipmt(0.0824/12, per, 1\*12, principal)

>>> ppmt = np.ppmt(0.0824/12, per, 1\*12, principal)

Each element of the sum of the 'ipmt' and 'ppmt' arrays should equal

'pmt'.

>>> pmt = np.pmt(0.0824/12, 1\*12, principal)

>>> np.allclose(ipmt + ppmt, pmt)

True

>>> fmt = '{0:2d} {1:8.2f} {2:8.2f} {3:8.2f}'

>>> for payment in per:

... index = payment - 1

... principal = principal + ppmt[index]

... print(fmt.format(payment, ppmt[index], ipmt[index], principal))

1 -200.58 -17.17 2299.42

2 -201.96 -15.79 2097.46

3 -203.35 -14.40 1894.11

4 -204.74 -13.01 1689.37

5 -206.15 -11.60 1483.22

6 -207.56 -10.18 1275.66

7 -208.99 -8.76 1066.67

8 -210.42 -7.32 856.25

9 -211.87 -5.88 644.38

10 -213.32 -4.42 431.05

11 -214.79 -2.96 216.26

12 -216.26 -1.49 -0.00

>>> interestpd = np.sum(ipmt)

>>> np.round(interestpd, 2)

-112.98

irr(values)

Return the Internal Rate of Return (IRR).

This is the "average" periodically compounded rate of return

that gives a net present value of 0.0; for a more complete explanation,

see Notes below.

:class:`decimal.Decimal` type is not supported.

Parameters

----------

values : array\_like, shape(N,)

Input cash flows per time period. By convention, net "deposits"

are negative and net "withdrawals" are positive. Thus, for

example, at least the first element of `values`, which represents

the initial investment, will typically be negative.

Returns

-------

out : float

Internal Rate of Return for periodic input values.

Notes

-----

The IRR is perhaps best understood through an example (illustrated

using np.irr in the Examples section below). Suppose one invests 100

units and then makes the following withdrawals at regular (fixed)

intervals: 39, 59, 55, 20. Assuming the ending value is 0, one's 100

unit investment yields 173 units; however, due to the combination of

compounding and the periodic withdrawals, the "average" rate of return

is neither simply 0.73/4 nor (1.73)^0.25-1. Rather, it is the solution

(for :math:`r`) of the equation:

.. math:: -100 + \frac{39}{1+r} + \frac{59}{(1+r)^2}

+ \frac{55}{(1+r)^3} + \frac{20}{(1+r)^4} = 0

In general, for `values` :math:`= [v\_0, v\_1, ... v\_M]`,

irr is the solution of the equation: [G]\_

.. math:: \sum\_{t=0}^M{\frac{v\_t}{(1+irr)^{t}}} = 0

References

----------

.. [G] L. J. Gitman, "Principles of Managerial Finance, Brief," 3rd ed.,

Addison-Wesley, 2003, pg. 348.

Examples

--------

>>> round(np.irr([-100, 39, 59, 55, 20]), 5)

0.28095

>>> round(np.irr([-100, 0, 0, 74]), 5)

-0.0955

>>> round(np.irr([-100, 100, 0, -7]), 5)

-0.0833

>>> round(np.irr([-100, 100, 0, 7]), 5)

0.06206

>>> round(np.irr([-5, 10.5, 1, -8, 1]), 5)

0.0886

(Compare with the Example given for numpy.lib.financial.npv)

is\_busday(...)

is\_busday(dates, weekmask='1111100', holidays=None, busdaycal=None, out=None)

Calculates which of the given dates are valid days, and which are not.

.. versionadded:: 1.7.0

Parameters

----------

dates : array\_like of datetime64[D]

The array of dates to process.

weekmask : str or array\_like of bool, optional

A seven-element array indicating which of Monday through Sunday are

valid days. May be specified as a length-seven list or array, like

[1,1,1,1,1,0,0]; a length-seven string, like '1111100'; or a string

like "Mon Tue Wed Thu Fri", made up of 3-character abbreviations for

weekdays, optionally separated by white space. Valid abbreviations

are: Mon Tue Wed Thu Fri Sat Sun

holidays : array\_like of datetime64[D], optional

An array of dates to consider as invalid dates. They may be

specified in any order, and NaT (not-a-time) dates are ignored.

This list is saved in a normalized form that is suited for

fast calculations of valid days.

busdaycal : busdaycalendar, optional

A `busdaycalendar` object which specifies the valid days. If this

parameter is provided, neither weekmask nor holidays may be

provided.

out : array of bool, optional

If provided, this array is filled with the result.

Returns

-------

out : array of bool

An array with the same shape as ``dates``, containing True for

each valid day, and False for each invalid day.

See Also

--------

busdaycalendar: An object that specifies a custom set of valid days.

busday\_offset : Applies an offset counted in valid days.

busday\_count : Counts how many valid days are in a half-open date range.

Examples

--------

>>> # The weekdays are Friday, Saturday, and Monday

... np.is\_busday(['2011-07-01', '2011-07-02', '2011-07-18'],

... holidays=['2011-07-01', '2011-07-04', '2011-07-17'])

array([False, False, True])

isclose(a, b, rtol=1e-05, atol=1e-08, equal\_nan=False)

Returns a boolean array where two arrays are element-wise equal within a

tolerance.

The tolerance values are positive, typically very small numbers. The

relative difference (`rtol` \* abs(`b`)) and the absolute difference

`atol` are added together to compare against the absolute difference

between `a` and `b`.

.. warning:: The default `atol` is not appropriate for comparing numbers

that are much smaller than one (see Notes).

Parameters

----------

a, b : array\_like

Input arrays to compare.

rtol : float

The relative tolerance parameter (see Notes).

atol : float

The absolute tolerance parameter (see Notes).

equal\_nan : bool

Whether to compare NaN's as equal. If True, NaN's in `a` will be

considered equal to NaN's in `b` in the output array.

Returns

-------

y : array\_like

Returns a boolean array of where `a` and `b` are equal within the

given tolerance. If both `a` and `b` are scalars, returns a single

boolean value.

See Also

--------

allclose

Notes

-----

.. versionadded:: 1.7.0

For finite values, isclose uses the following equation to test whether

two floating point values are equivalent.

absolute(`a` - `b`) <= (`atol` + `rtol` \* absolute(`b`))

Unlike the built-in `math.isclose`, the above equation is not symmetric

in `a` and `b` -- it assumes `b` is the reference value -- so that

`isclose(a, b)` might be different from `isclose(b, a)`. Furthermore,

the default value of atol is not zero, and is used to determine what

small values should be considered close to zero. The default value is

appropriate for expected values of order unity: if the expected values

are significantly smaller than one, it can result in false positives.

`atol` should be carefully selected for the use case at hand. A zero value

for `atol` will result in `False` if either `a` or `b` is zero.

Examples

--------

>>> np.isclose([1e10,1e-7], [1.00001e10,1e-8])

array([ True, False])

>>> np.isclose([1e10,1e-8], [1.00001e10,1e-9])

array([ True, True])

>>> np.isclose([1e10,1e-8], [1.0001e10,1e-9])

array([False, True])

>>> np.isclose([1.0, np.nan], [1.0, np.nan])

array([ True, False])

>>> np.isclose([1.0, np.nan], [1.0, np.nan], equal\_nan=True)

array([ True, True])

>>> np.isclose([1e-8, 1e-7], [0.0, 0.0])

array([ True, False])

>>> np.isclose([1e-100, 1e-7], [0.0, 0.0], atol=0.0)

array([False, False])

>>> np.isclose([1e-10, 1e-10], [1e-20, 0.0])

array([ True, True])

>>> np.isclose([1e-10, 1e-10], [1e-20, 0.999999e-10], atol=0.0)

array([False, True])

iscomplex(x)

Returns a bool array, where True if input element is complex.

What is tested is whether the input has a non-zero imaginary part, not if

the input type is complex.

Parameters

----------

x : array\_like

Input array.

Returns

-------

out : ndarray of bools

Output array.

See Also

--------

isreal

iscomplexobj : Return True if x is a complex type or an array of complex

numbers.

Examples

--------

>>> np.iscomplex([1+1j, 1+0j, 4.5, 3, 2, 2j])

array([ True, False, False, False, False, True])

iscomplexobj(x)

Check for a complex type or an array of complex numbers.

The type of the input is checked, not the value. Even if the input

has an imaginary part equal to zero, `iscomplexobj` evaluates to True.

Parameters

----------

x : any

The input can be of any type and shape.

Returns

-------

iscomplexobj : bool

The return value, True if `x` is of a complex type or has at least

one complex element.

See Also

--------

isrealobj, iscomplex

Examples

--------

>>> np.iscomplexobj(1)

False

>>> np.iscomplexobj(1+0j)

True

>>> np.iscomplexobj([3, 1+0j, True])

True

isfortran(a)

Check if the array is Fortran contiguous but \*not\* C contiguous.

This function is obsolete and, because of changes due to relaxed stride

checking, its return value for the same array may differ for versions

of NumPy >= 1.10.0 and previous versions. If you only want to check if an

array is Fortran contiguous use ``a.flags.f\_contiguous`` instead.

Parameters

----------

a : ndarray

Input array.

Returns

-------

isfortran : bool

Returns True if the array is Fortran contiguous but \*not\* C contiguous.

Examples

--------

np.array allows to specify whether the array is written in C-contiguous

order (last index varies the fastest), or FORTRAN-contiguous order in

memory (first index varies the fastest).

>>> a = np.array([[1, 2, 3], [4, 5, 6]], order='C')

>>> a

array([[1, 2, 3],

[4, 5, 6]])

>>> np.isfortran(a)

False

>>> b = np.array([[1, 2, 3], [4, 5, 6]], order='F')

>>> b

array([[1, 2, 3],

[4, 5, 6]])

>>> np.isfortran(b)

True

The transpose of a C-ordered array is a FORTRAN-ordered array.

>>> a = np.array([[1, 2, 3], [4, 5, 6]], order='C')

>>> a

array([[1, 2, 3],

[4, 5, 6]])

>>> np.isfortran(a)

False

>>> b = a.T

>>> b

array([[1, 4],

[2, 5],

[3, 6]])

>>> np.isfortran(b)

True

C-ordered arrays evaluate as False even if they are also FORTRAN-ordered.

>>> np.isfortran(np.array([1, 2], order='FORTRAN'))

False

isin(element, test\_elements, assume\_unique=False, invert=False)

Calculates `element in test\_elements`, broadcasting over `element` only.

Returns a boolean array of the same shape as `element` that is True

where an element of `element` is in `test\_elements` and False otherwise.

Parameters

----------

element : array\_like

Input array.

test\_elements : array\_like

The values against which to test each value of `element`.

This argument is flattened if it is an array or array\_like.

See notes for behavior with non-array-like parameters.

assume\_unique : bool, optional

If True, the input arrays are both assumed to be unique, which

can speed up the calculation. Default is False.

invert : bool, optional

If True, the values in the returned array are inverted, as if

calculating `element not in test\_elements`. Default is False.

``np.isin(a, b, invert=True)`` is equivalent to (but faster

than) ``np.invert(np.isin(a, b))``.

Returns

-------

isin : ndarray, bool

Has the same shape as `element`. The values `element[isin]`

are in `test\_elements`.

See Also

--------

in1d : Flattened version of this function.

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Notes

-----

`isin` is an element-wise function version of the python keyword `in`.

``isin(a, b)`` is roughly equivalent to

``np.array([item in b for item in a])`` if `a` and `b` are 1-D sequences.

`element` and `test\_elements` are converted to arrays if they are not

already. If `test\_elements` is a set (or other non-sequence collection)

it will be converted to an object array with one element, rather than an

array of the values contained in `test\_elements`. This is a consequence

of the `array` constructor's way of handling non-sequence collections.

Converting the set to a list usually gives the desired behavior.

.. versionadded:: 1.13.0

Examples

--------

>>> element = 2\*np.arange(4).reshape((2, 2))

>>> element

array([[0, 2],

[4, 6]])

>>> test\_elements = [1, 2, 4, 8]

>>> mask = np.isin(element, test\_elements)

>>> mask

array([[False, True],

[ True, False]])

>>> element[mask]

array([2, 4])

The indices of the matched values can be obtained with `nonzero`:

>>> np.nonzero(mask)

(array([0, 1]), array([1, 0]))

The test can also be inverted:

>>> mask = np.isin(element, test\_elements, invert=True)

>>> mask

array([[ True, False],

[False, True]])

>>> element[mask]

array([0, 6])

Because of how `array` handles sets, the following does not

work as expected:

>>> test\_set = {1, 2, 4, 8}

>>> np.isin(element, test\_set)

array([[False, False],

[False, False]])

Casting the set to a list gives the expected result:

>>> np.isin(element, list(test\_set))

array([[False, True],

[ True, False]])

isneginf(x, out=None)

Test element-wise for negative infinity, return result as bool array.

Parameters

----------

x : array\_like

The input array.

out : array\_like, optional

A boolean array with the same shape and type as `x` to store the

result.

Returns

-------

out : ndarray

A boolean array with the same dimensions as the input.

If second argument is not supplied then a numpy boolean array is

returned with values True where the corresponding element of the

input is negative infinity and values False where the element of

the input is not negative infinity.

If a second argument is supplied the result is stored there. If the

type of that array is a numeric type the result is represented as

zeros and ones, if the type is boolean then as False and True. The

return value `out` is then a reference to that array.

See Also

--------

isinf, isposinf, isnan, isfinite

Notes

-----

NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic

(IEEE 754).

Errors result if the second argument is also supplied when x is a scalar

input, if first and second arguments have different shapes, or if the

first argument has complex values.

Examples

--------

>>> np.isneginf(np.NINF)

True

>>> np.isneginf(np.inf)

False

>>> np.isneginf(np.PINF)

False

>>> np.isneginf([-np.inf, 0., np.inf])

array([ True, False, False])

>>> x = np.array([-np.inf, 0., np.inf])

>>> y = np.array([2, 2, 2])

>>> np.isneginf(x, y)

array([1, 0, 0])

>>> y

array([1, 0, 0])

isposinf(x, out=None)

Test element-wise for positive infinity, return result as bool array.

Parameters

----------

x : array\_like

The input array.

y : array\_like, optional

A boolean array with the same shape as `x` to store the result.

Returns

-------

out : ndarray

A boolean array with the same dimensions as the input.

If second argument is not supplied then a boolean array is returned

with values True where the corresponding element of the input is

positive infinity and values False where the element of the input is

not positive infinity.

If a second argument is supplied the result is stored there. If the

type of that array is a numeric type the result is represented as zeros

and ones, if the type is boolean then as False and True.

The return value `out` is then a reference to that array.

See Also

--------

isinf, isneginf, isfinite, isnan

Notes

-----

NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic

(IEEE 754).

Errors result if the second argument is also supplied when x is a scalar

input, if first and second arguments have different shapes, or if the

first argument has complex values

Examples

--------

>>> np.isposinf(np.PINF)

True

>>> np.isposinf(np.inf)

True

>>> np.isposinf(np.NINF)

False

>>> np.isposinf([-np.inf, 0., np.inf])

array([False, False, True])

>>> x = np.array([-np.inf, 0., np.inf])

>>> y = np.array([2, 2, 2])

>>> np.isposinf(x, y)

array([0, 0, 1])

>>> y

array([0, 0, 1])

isreal(x)

Returns a bool array, where True if input element is real.

If element has complex type with zero complex part, the return value

for that element is True.

Parameters

----------

x : array\_like

Input array.

Returns

-------

out : ndarray, bool

Boolean array of same shape as `x`.

See Also

--------

iscomplex

isrealobj : Return True if x is not a complex type.

Examples

--------

>>> np.isreal([1+1j, 1+0j, 4.5, 3, 2, 2j])

array([False, True, True, True, True, False])

isrealobj(x)

Return True if x is a not complex type or an array of complex numbers.

The type of the input is checked, not the value. So even if the input

has an imaginary part equal to zero, `isrealobj` evaluates to False

if the data type is complex.

Parameters

----------

x : any

The input can be of any type and shape.

Returns

-------

y : bool

The return value, False if `x` is of a complex type.

See Also

--------

iscomplexobj, isreal

Examples

--------

>>> np.isrealobj(1)

True

>>> np.isrealobj(1+0j)

False

>>> np.isrealobj([3, 1+0j, True])

False

isscalar(num)

Returns True if the type of `num` is a scalar type.

Parameters

----------

num : any

Input argument, can be of any type and shape.

Returns

-------

val : bool

True if `num` is a scalar type, False if it is not.

See Also

--------

ndim : Get the number of dimensions of an array

Notes

-----

In almost all cases ``np.ndim(x) == 0`` should be used instead of this

function, as that will also return true for 0d arrays. This is how

numpy overloads functions in the style of the ``dx`` arguments to `gradient`

and the ``bins`` argument to `histogram`. Some key differences:

+--------------------------------------+---------------+-------------------+

| x |``isscalar(x)``|``np.ndim(x) == 0``|

+======================================+===============+===================+

| PEP 3141 numeric objects (including | ``True`` | ``True`` |

| builtins) | | |

+--------------------------------------+---------------+-------------------+

| builtin string and buffer objects | ``True`` | ``True`` |

+--------------------------------------+---------------+-------------------+

| other builtin objects, like | ``False`` | ``True`` |

| `pathlib.Path`, `Exception`, | | |

| the result of `re.compile` | | |

+--------------------------------------+---------------+-------------------+

| third-party objects like | ``False`` | ``True`` |

| `matplotlib.figure.Figure` | | |

+--------------------------------------+---------------+-------------------+

| zero-dimensional numpy arrays | ``False`` | ``True`` |

+--------------------------------------+---------------+-------------------+

| other numpy arrays | ``False`` | ``False`` |

+--------------------------------------+---------------+-------------------+

| `list`, `tuple`, and other sequence | ``False`` | ``False`` |

| objects | | |

+--------------------------------------+---------------+-------------------+

Examples

--------

>>> np.isscalar(3.1)

True

>>> np.isscalar(np.array(3.1))

False

>>> np.isscalar([3.1])

False

>>> np.isscalar(False)

True

>>> np.isscalar('numpy')

True

NumPy supports PEP 3141 numbers:

>>> from fractions import Fraction

>>> np.isscalar(Fraction(5, 17))

True

>>> from numbers import Number

>>> np.isscalar(Number())

True

issctype(rep)

Determines whether the given object represents a scalar data-type.

Parameters

----------

rep : any

If `rep` is an instance of a scalar dtype, True is returned. If not,

False is returned.

Returns

-------

out : bool

Boolean result of check whether `rep` is a scalar dtype.

See Also

--------

issubsctype, issubdtype, obj2sctype, sctype2char

Examples

--------

>>> np.issctype(np.int32)

True

>>> np.issctype(list)

False

>>> np.issctype(1.1)

False

Strings are also a scalar type:

>>> np.issctype(np.dtype('str'))

True

issubclass\_(arg1, arg2)

Determine if a class is a subclass of a second class.

`issubclass\_` is equivalent to the Python built-in ``issubclass``,

except that it returns False instead of raising a TypeError if one

of the arguments is not a class.

Parameters

----------

arg1 : class

Input class. True is returned if `arg1` is a subclass of `arg2`.

arg2 : class or tuple of classes.

Input class. If a tuple of classes, True is returned if `arg1` is a

subclass of any of the tuple elements.

Returns

-------

out : bool

Whether `arg1` is a subclass of `arg2` or not.

See Also

--------

issubsctype, issubdtype, issctype

Examples

--------

>>> np.issubclass\_(np.int32, int)

False # True on Python 2.7

>>> np.issubclass\_(np.int32, float)

False

issubdtype(arg1, arg2)

Returns True if first argument is a typecode lower/equal in type hierarchy.

Parameters

----------

arg1, arg2 : dtype\_like

dtype or string representing a typecode.

Returns

-------

out : bool

See Also

--------

issubsctype, issubclass\_

numpy.core.numerictypes : Overview of numpy type hierarchy.

Examples

--------

>>> np.issubdtype('S1', np.string\_)

True

>>> np.issubdtype(np.float64, np.float32)

False

issubsctype(arg1, arg2)

Determine if the first argument is a subclass of the second argument.

Parameters

----------

arg1, arg2 : dtype or dtype specifier

Data-types.

Returns

-------

out : bool

The result.

See Also

--------

issctype, issubdtype, obj2sctype

Examples

--------

>>> np.issubsctype('S8', str)

False

>>> np.issubsctype(np.array([1]), int)

True

>>> np.issubsctype(np.array([1]), float)

False

iterable(y)

Check whether or not an object can be iterated over.

Parameters

----------

y : object

Input object.

Returns

-------

b : bool

Return ``True`` if the object has an iterator method or is a

sequence and ``False`` otherwise.

Examples

--------

>>> np.iterable([1, 2, 3])

True

>>> np.iterable(2)

False

ix\_(\*args)

Construct an open mesh from multiple sequences.

This function takes N 1-D sequences and returns N outputs with N

dimensions each, such that the shape is 1 in all but one dimension

and the dimension with the non-unit shape value cycles through all

N dimensions.

Using `ix\_` one can quickly construct index arrays that will index

the cross product. ``a[np.ix\_([1,3],[2,5])]`` returns the array

``[[a[1,2] a[1,5]], [a[3,2] a[3,5]]]``.

Parameters

----------

args : 1-D sequences

Each sequence should be of integer or boolean type.

Boolean sequences will be interpreted as boolean masks for the

corresponding dimension (equivalent to passing in

``np.nonzero(boolean\_sequence)``).

Returns

-------

out : tuple of ndarrays

N arrays with N dimensions each, with N the number of input

sequences. Together these arrays form an open mesh.

See Also

--------

ogrid, mgrid, meshgrid

Examples

--------

>>> a = np.arange(10).reshape(2, 5)

>>> a

array([[0, 1, 2, 3, 4],

[5, 6, 7, 8, 9]])

>>> ixgrid = np.ix\_([0, 1], [2, 4])

>>> ixgrid

(array([[0],

[1]]), array([[2, 4]]))

>>> ixgrid[0].shape, ixgrid[1].shape

((2, 1), (1, 2))

>>> a[ixgrid]

array([[2, 4],

[7, 9]])

>>> ixgrid = np.ix\_([True, True], [2, 4])

>>> a[ixgrid]

array([[2, 4],

[7, 9]])

>>> ixgrid = np.ix\_([True, True], [False, False, True, False, True])

>>> a[ixgrid]

array([[2, 4],

[7, 9]])

kaiser(M, beta)

Return the Kaiser window.

The Kaiser window is a taper formed by using a Bessel function.

Parameters

----------

M : int

Number of points in the output window. If zero or less, an

empty array is returned.

beta : float

Shape parameter for window.

Returns

-------

out : array

The window, with the maximum value normalized to one (the value

one appears only if the number of samples is odd).

See Also

--------

bartlett, blackman, hamming, hanning

Notes

-----

The Kaiser window is defined as

.. math:: w(n) = I\_0\left( \beta \sqrt{1-\frac{4n^2}{(M-1)^2}}

\right)/I\_0(\beta)

with

.. math:: \quad -\frac{M-1}{2} \leq n \leq \frac{M-1}{2},

where :math:`I\_0` is the modified zeroth-order Bessel function.

The Kaiser was named for Jim Kaiser, who discovered a simple

approximation to the DPSS window based on Bessel functions. The Kaiser

window is a very good approximation to the Digital Prolate Spheroidal

Sequence, or Slepian window, which is the transform which maximizes the

energy in the main lobe of the window relative to total energy.

The Kaiser can approximate many other windows by varying the beta

parameter.

==== =======================

beta Window shape

==== =======================

0 Rectangular

5 Similar to a Hamming

6 Similar to a Hanning

8.6 Similar to a Blackman

==== =======================

A beta value of 14 is probably a good starting point. Note that as beta

gets large, the window narrows, and so the number of samples needs to be

large enough to sample the increasingly narrow spike, otherwise NaNs will

get returned.

Most references to the Kaiser window come from the signal processing

literature, where it is used as one of many windowing functions for

smoothing values. It is also known as an apodization (which means

"removing the foot", i.e. smoothing discontinuities at the beginning

and end of the sampled signal) or tapering function.

References

----------

.. [1] J. F. Kaiser, "Digital Filters" - Ch 7 in "Systems analysis by

digital computer", Editors: F.F. Kuo and J.F. Kaiser, p 218-285.

John Wiley and Sons, New York, (1966).

.. [2] E.R. Kanasewich, "Time Sequence Analysis in Geophysics", The

University of Alberta Press, 1975, pp. 177-178.

.. [3] Wikipedia, "Window function",

https://en.wikipedia.org/wiki/Window\_function

Examples

--------

>>> import matplotlib.pyplot as plt

>>> np.kaiser(12, 14)

array([7.72686684e-06, 3.46009194e-03, 4.65200189e-02, # may vary

2.29737120e-01, 5.99885316e-01, 9.45674898e-01,

9.45674898e-01, 5.99885316e-01, 2.29737120e-01,

4.65200189e-02, 3.46009194e-03, 7.72686684e-06])

Plot the window and the frequency response:

>>> from numpy.fft import fft, fftshift

>>> window = np.kaiser(51, 14)

>>> plt.plot(window)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Kaiser window")

Text(0.5, 1.0, 'Kaiser window')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("Sample")

Text(0.5, 0, 'Sample')

>>> plt.show()

>>> plt.figure()

<Figure size 640x480 with 0 Axes>

>>> A = fft(window, 2048) / 25.5

>>> mag = np.abs(fftshift(A))

>>> freq = np.linspace(-0.5, 0.5, len(A))

>>> response = 20 \* np.log10(mag)

>>> response = np.clip(response, -100, 100)

>>> plt.plot(freq, response)

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Frequency response of Kaiser window")

Text(0.5, 1.0, 'Frequency response of Kaiser window')

>>> plt.ylabel("Magnitude [dB]")

Text(0, 0.5, 'Magnitude [dB]')

>>> plt.xlabel("Normalized frequency [cycles per sample]")

Text(0.5, 0, 'Normalized frequency [cycles per sample]')

>>> plt.axis('tight')

(-0.5, 0.5, -100.0, ...) # may vary

>>> plt.show()

kron(a, b)

Kronecker product of two arrays.

Computes the Kronecker product, a composite array made of blocks of the

second array scaled by the first.

Parameters

----------

a, b : array\_like

Returns

-------

out : ndarray

See Also

--------

outer : The outer product

Notes

-----

The function assumes that the number of dimensions of `a` and `b`

are the same, if necessary prepending the smallest with ones.

If `a.shape = (r0,r1,..,rN)` and `b.shape = (s0,s1,...,sN)`,

the Kronecker product has shape `(r0\*s0, r1\*s1, ..., rN\*SN)`.

The elements are products of elements from `a` and `b`, organized

explicitly by::

kron(a,b)[k0,k1,...,kN] = a[i0,i1,...,iN] \* b[j0,j1,...,jN]

where::

kt = it \* st + jt, t = 0,...,N

In the common 2-D case (N=1), the block structure can be visualized::

[[ a[0,0]\*b, a[0,1]\*b, ... , a[0,-1]\*b ],

[ ... ... ],

[ a[-1,0]\*b, a[-1,1]\*b, ... , a[-1,-1]\*b ]]

Examples

--------

>>> np.kron([1,10,100], [5,6,7])

array([ 5, 6, 7, ..., 500, 600, 700])

>>> np.kron([5,6,7], [1,10,100])

array([ 5, 50, 500, ..., 7, 70, 700])

>>> np.kron(np.eye(2), np.ones((2,2)))

array([[1., 1., 0., 0.],

[1., 1., 0., 0.],

[0., 0., 1., 1.],

[0., 0., 1., 1.]])

>>> a = np.arange(100).reshape((2,5,2,5))

>>> b = np.arange(24).reshape((2,3,4))

>>> c = np.kron(a,b)

>>> c.shape

(2, 10, 6, 20)

>>> I = (1,3,0,2)

>>> J = (0,2,1)

>>> J1 = (0,) + J # extend to ndim=4

>>> S1 = (1,) + b.shape

>>> K = tuple(np.array(I) \* np.array(S1) + np.array(J1))

>>> c[K] == a[I]\*b[J]

True

lexsort(...)

lexsort(keys, axis=-1)

Perform an indirect stable sort using a sequence of keys.

Given multiple sorting keys, which can be interpreted as columns in a

spreadsheet, lexsort returns an array of integer indices that describes

the sort order by multiple columns. The last key in the sequence is used

for the primary sort order, the second-to-last key for the secondary sort

order, and so on. The keys argument must be a sequence of objects that

can be converted to arrays of the same shape. If a 2D array is provided

for the keys argument, it's rows are interpreted as the sorting keys and

sorting is according to the last row, second last row etc.

Parameters

----------

keys : (k, N) array or tuple containing k (N,)-shaped sequences

The `k` different "columns" to be sorted. The last column (or row if

`keys` is a 2D array) is the primary sort key.

axis : int, optional

Axis to be indirectly sorted. By default, sort over the last axis.

Returns

-------

indices : (N,) ndarray of ints

Array of indices that sort the keys along the specified axis.

See Also

--------

argsort : Indirect sort.

ndarray.sort : In-place sort.

sort : Return a sorted copy of an array.

Examples

--------

Sort names: first by surname, then by name.

>>> surnames = ('Hertz', 'Galilei', 'Hertz')

>>> first\_names = ('Heinrich', 'Galileo', 'Gustav')

>>> ind = np.lexsort((first\_names, surnames))

>>> ind

array([1, 2, 0])

>>> [surnames[i] + ", " + first\_names[i] for i in ind]

['Galilei, Galileo', 'Hertz, Gustav', 'Hertz, Heinrich']

Sort two columns of numbers:

>>> a = [1,5,1,4,3,4,4] # First column

>>> b = [9,4,0,4,0,2,1] # Second column

>>> ind = np.lexsort((b,a)) # Sort by a, then by b

>>> ind

array([2, 0, 4, 6, 5, 3, 1])

>>> [(a[i],b[i]) for i in ind]

[(1, 0), (1, 9), (3, 0), (4, 1), (4, 2), (4, 4), (5, 4)]

Note that sorting is first according to the elements of ``a``.

Secondary sorting is according to the elements of ``b``.

A normal ``argsort`` would have yielded:

>>> [(a[i],b[i]) for i in np.argsort(a)]

[(1, 9), (1, 0), (3, 0), (4, 4), (4, 2), (4, 1), (5, 4)]

Structured arrays are sorted lexically by ``argsort``:

>>> x = np.array([(1,9), (5,4), (1,0), (4,4), (3,0), (4,2), (4,1)],

... dtype=np.dtype([('x', int), ('y', int)]))

>>> np.argsort(x) # or np.argsort(x, order=('x', 'y'))

array([2, 0, 4, 6, 5, 3, 1])

linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

Return evenly spaced numbers over a specified interval.

Returns `num` evenly spaced samples, calculated over the

interval [`start`, `stop`].

The endpoint of the interval can optionally be excluded.

.. versionchanged:: 1.16.0

Non-scalar `start` and `stop` are now supported.

Parameters

----------

start : array\_like

The starting value of the sequence.

stop : array\_like

The end value of the sequence, unless `endpoint` is set to False.

In that case, the sequence consists of all but the last of ``num + 1``

evenly spaced samples, so that `stop` is excluded. Note that the step

size changes when `endpoint` is False.

num : int, optional

Number of samples to generate. Default is 50. Must be non-negative.

endpoint : bool, optional

If True, `stop` is the last sample. Otherwise, it is not included.

Default is True.

retstep : bool, optional

If True, return (`samples`, `step`), where `step` is the spacing

between samples.

dtype : dtype, optional

The type of the output array. If `dtype` is not given, infer the data

type from the other input arguments.

.. versionadded:: 1.9.0

axis : int, optional

The axis in the result to store the samples. Relevant only if start

or stop are array-like. By default (0), the samples will be along a

new axis inserted at the beginning. Use -1 to get an axis at the end.

.. versionadded:: 1.16.0

Returns

-------

samples : ndarray

There are `num` equally spaced samples in the closed interval

``[start, stop]`` or the half-open interval ``[start, stop)``

(depending on whether `endpoint` is True or False).

step : float, optional

Only returned if `retstep` is True

Size of spacing between samples.

See Also

--------

arange : Similar to `linspace`, but uses a step size (instead of the

number of samples).

geomspace : Similar to `linspace`, but with numbers spaced evenly on a log

scale (a geometric progression).

logspace : Similar to `geomspace`, but with the end points specified as

logarithms.

Examples

--------

>>> np.linspace(2.0, 3.0, num=5)

array([2. , 2.25, 2.5 , 2.75, 3. ])

>>> np.linspace(2.0, 3.0, num=5, endpoint=False)

array([2. , 2.2, 2.4, 2.6, 2.8])

>>> np.linspace(2.0, 3.0, num=5, retstep=True)

(array([2. , 2.25, 2.5 , 2.75, 3. ]), 0.25)

Graphical illustration:

>>> import matplotlib.pyplot as plt

>>> N = 8

>>> y = np.zeros(N)

>>> x1 = np.linspace(0, 10, N, endpoint=True)

>>> x2 = np.linspace(0, 10, N, endpoint=False)

>>> plt.plot(x1, y, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.plot(x2, y + 0.5, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.ylim([-0.5, 1])

(-0.5, 1)

>>> plt.show()

load(file, mmap\_mode=None, allow\_pickle=False, fix\_imports=True, encoding='ASCII')

Load arrays or pickled objects from ``.npy``, ``.npz`` or pickled files.

.. warning:: Loading files that contain object arrays uses the ``pickle``

module, which is not secure against erroneous or maliciously

constructed data. Consider passing ``allow\_pickle=False`` to

load data that is known not to contain object arrays for the

safer handling of untrusted sources.

Parameters

----------

file : file-like object, string, or pathlib.Path

The file to read. File-like objects must support the

``seek()`` and ``read()`` methods. Pickled files require that the

file-like object support the ``readline()`` method as well.

mmap\_mode : {None, 'r+', 'r', 'w+', 'c'}, optional

If not None, then memory-map the file, using the given mode (see

`numpy.memmap` for a detailed description of the modes). A

memory-mapped array is kept on disk. However, it can be accessed

and sliced like any ndarray. Memory mapping is especially useful

for accessing small fragments of large files without reading the

entire file into memory.

allow\_pickle : bool, optional

Allow loading pickled object arrays stored in npy files. Reasons for

disallowing pickles include security, as loading pickled data can

execute arbitrary code. If pickles are disallowed, loading object

arrays will fail. Default: False

.. versionchanged:: 1.16.3

Made default False in response to CVE-2019-6446.

fix\_imports : bool, optional

Only useful when loading Python 2 generated pickled files on Python 3,

which includes npy/npz files containing object arrays. If `fix\_imports`

is True, pickle will try to map the old Python 2 names to the new names

used in Python 3.

encoding : str, optional

What encoding to use when reading Python 2 strings. Only useful when

loading Python 2 generated pickled files in Python 3, which includes

npy/npz files containing object arrays. Values other than 'latin1',

'ASCII', and 'bytes' are not allowed, as they can corrupt numerical

data. Default: 'ASCII'

Returns

-------

result : array, tuple, dict, etc.

Data stored in the file. For ``.npz`` files, the returned instance

of NpzFile class must be closed to avoid leaking file descriptors.

Raises

------

IOError

If the input file does not exist or cannot be read.

ValueError

The file contains an object array, but allow\_pickle=False given.

See Also

--------

save, savez, savez\_compressed, loadtxt

memmap : Create a memory-map to an array stored in a file on disk.

lib.format.open\_memmap : Create or load a memory-mapped ``.npy`` file.

Notes

-----

- If the file contains pickle data, then whatever object is stored

in the pickle is returned.

- If the file is a ``.npy`` file, then a single array is returned.

- If the file is a ``.npz`` file, then a dictionary-like object is

returned, containing ``{filename: array}`` key-value pairs, one for

each file in the archive.

- If the file is a ``.npz`` file, the returned value supports the

context manager protocol in a similar fashion to the open function::

with load('foo.npz') as data:

a = data['a']

The underlying file descriptor is closed when exiting the 'with'

block.

Examples

--------

Store data to disk, and load it again:

>>> np.save('/tmp/123', np.array([[1, 2, 3], [4, 5, 6]]))

>>> np.load('/tmp/123.npy')

array([[1, 2, 3],

[4, 5, 6]])

Store compressed data to disk, and load it again:

>>> a=np.array([[1, 2, 3], [4, 5, 6]])

>>> b=np.array([1, 2])

>>> np.savez('/tmp/123.npz', a=a, b=b)

>>> data = np.load('/tmp/123.npz')

>>> data['a']

array([[1, 2, 3],

[4, 5, 6]])

>>> data['b']

array([1, 2])

>>> data.close()

Mem-map the stored array, and then access the second row

directly from disk:

>>> X = np.load('/tmp/123.npy', mmap\_mode='r')

>>> X[1, :]

memmap([4, 5, 6])

loads(\*args, \*\*kwargs)

loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes', max\_rows=None)

Load data from a text file.

Each row in the text file must have the same number of values.

Parameters

----------

fname : file, str, or pathlib.Path

File, filename, or generator to read. If the filename extension is

``.gz`` or ``.bz2``, the file is first decompressed. Note that

generators should return byte strings for Python 3k.

dtype : data-type, optional

Data-type of the resulting array; default: float. If this is a

structured data-type, the resulting array will be 1-dimensional, and

each row will be interpreted as an element of the array. In this

case, the number of columns used must match the number of fields in

the data-type.

comments : str or sequence of str, optional

The characters or list of characters used to indicate the start of a

comment. None implies no comments. For backwards compatibility, byte

strings will be decoded as 'latin1'. The default is '#'.

delimiter : str, optional

The string used to separate values. For backwards compatibility, byte

strings will be decoded as 'latin1'. The default is whitespace.

converters : dict, optional

A dictionary mapping column number to a function that will parse the

column string into the desired value. E.g., if column 0 is a date

string: ``converters = {0: datestr2num}``. Converters can also be

used to provide a default value for missing data (but see also

`genfromtxt`): ``converters = {3: lambda s: float(s.strip() or 0)}``.

Default: None.

skiprows : int, optional

Skip the first `skiprows` lines, including comments; default: 0.

usecols : int or sequence, optional

Which columns to read, with 0 being the first. For example,

``usecols = (1,4,5)`` will extract the 2nd, 5th and 6th columns.

The default, None, results in all columns being read.

.. versionchanged:: 1.11.0

When a single column has to be read it is possible to use

an integer instead of a tuple. E.g ``usecols = 3`` reads the

fourth column the same way as ``usecols = (3,)`` would.

unpack : bool, optional

If True, the returned array is transposed, so that arguments may be

unpacked using ``x, y, z = loadtxt(...)``. When used with a structured

data-type, arrays are returned for each field. Default is False.

ndmin : int, optional

The returned array will have at least `ndmin` dimensions.

Otherwise mono-dimensional axes will be squeezed.

Legal values: 0 (default), 1 or 2.

.. versionadded:: 1.6.0

encoding : str, optional

Encoding used to decode the inputfile. Does not apply to input streams.

The special value 'bytes' enables backward compatibility workarounds

that ensures you receive byte arrays as results if possible and passes

'latin1' encoded strings to converters. Override this value to receive

unicode arrays and pass strings as input to converters. If set to None

the system default is used. The default value is 'bytes'.

.. versionadded:: 1.14.0

max\_rows : int, optional

Read `max\_rows` lines of content after `skiprows` lines. The default

is to read all the lines.

.. versionadded:: 1.16.0

Returns

-------

out : ndarray

Data read from the text file.

See Also

--------

load, fromstring, fromregex

genfromtxt : Load data with missing values handled as specified.

scipy.io.loadmat : reads MATLAB data files

Notes

-----

This function aims to be a fast reader for simply formatted files. The

`genfromtxt` function provides more sophisticated handling of, e.g.,

lines with missing values.

.. versionadded:: 1.10.0

The strings produced by the Python float.hex method can be used as

input for floats.

Examples

--------

>>> from io import StringIO # StringIO behaves like a file object

>>> c = StringIO(u"0 1\n2 3")

>>> np.loadtxt(c)

array([[0., 1.],

[2., 3.]])

>>> d = StringIO(u"M 21 72\nF 35 58")

>>> np.loadtxt(d, dtype={'names': ('gender', 'age', 'weight'),

... 'formats': ('S1', 'i4', 'f4')})

array([(b'M', 21, 72.), (b'F', 35, 58.)],

dtype=[('gender', 'S1'), ('age', '<i4'), ('weight', '<f4')])

>>> c = StringIO(u"1,0,2\n3,0,4")

>>> x, y = np.loadtxt(c, delimiter=',', usecols=(0, 2), unpack=True)

>>> x

array([1., 3.])

>>> y

array([2., 4.])

logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

Return numbers spaced evenly on a log scale.

In linear space, the sequence starts at ``base \*\* start``

(`base` to the power of `start`) and ends with ``base \*\* stop``

(see `endpoint` below).

.. versionchanged:: 1.16.0

Non-scalar `start` and `stop` are now supported.

Parameters

----------

start : array\_like

``base \*\* start`` is the starting value of the sequence.

stop : array\_like

``base \*\* stop`` is the final value of the sequence, unless `endpoint`

is False. In that case, ``num + 1`` values are spaced over the

interval in log-space, of which all but the last (a sequence of

length `num`) are returned.

num : integer, optional

Number of samples to generate. Default is 50.

endpoint : boolean, optional

If true, `stop` is the last sample. Otherwise, it is not included.

Default is True.

base : float, optional

The base of the log space. The step size between the elements in

``ln(samples) / ln(base)`` (or ``log\_base(samples)``) is uniform.

Default is 10.0.

dtype : dtype

The type of the output array. If `dtype` is not given, infer the data

type from the other input arguments.

axis : int, optional

The axis in the result to store the samples. Relevant only if start

or stop are array-like. By default (0), the samples will be along a

new axis inserted at the beginning. Use -1 to get an axis at the end.

.. versionadded:: 1.16.0

Returns

-------

samples : ndarray

`num` samples, equally spaced on a log scale.

See Also

--------

arange : Similar to linspace, with the step size specified instead of the

number of samples. Note that, when used with a float endpoint, the

endpoint may or may not be included.

linspace : Similar to logspace, but with the samples uniformly distributed

in linear space, instead of log space.

geomspace : Similar to logspace, but with endpoints specified directly.

Notes

-----

Logspace is equivalent to the code

>>> y = np.linspace(start, stop, num=num, endpoint=endpoint)

... # doctest: +SKIP

>>> power(base, y).astype(dtype)

... # doctest: +SKIP

Examples

--------

>>> np.logspace(2.0, 3.0, num=4)

array([ 100. , 215.443469 , 464.15888336, 1000. ])

>>> np.logspace(2.0, 3.0, num=4, endpoint=False)

array([100. , 177.827941 , 316.22776602, 562.34132519])

>>> np.logspace(2.0, 3.0, num=4, base=2.0)

array([4. , 5.0396842 , 6.34960421, 8. ])

Graphical illustration:

>>> import matplotlib.pyplot as plt

>>> N = 10

>>> x1 = np.logspace(0.1, 1, N, endpoint=True)

>>> x2 = np.logspace(0.1, 1, N, endpoint=False)

>>> y = np.zeros(N)

>>> plt.plot(x1, y, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.plot(x2, y + 0.5, 'o')

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.ylim([-0.5, 1])

(-0.5, 1)

>>> plt.show()

lookfor(what, module=None, import\_modules=True, regenerate=False, output=None)

Do a keyword search on docstrings.

A list of objects that matched the search is displayed,

sorted by relevance. All given keywords need to be found in the

docstring for it to be returned as a result, but the order does

not matter.

Parameters

----------

what : str

String containing words to look for.

module : str or list, optional

Name of module(s) whose docstrings to go through.

import\_modules : bool, optional

Whether to import sub-modules in packages. Default is True.

regenerate : bool, optional

Whether to re-generate the docstring cache. Default is False.

output : file-like, optional

File-like object to write the output to. If omitted, use a pager.

See Also

--------

source, info

Notes

-----

Relevance is determined only roughly, by checking if the keywords occur

in the function name, at the start of a docstring, etc.

Examples

--------

>>> np.lookfor('binary representation') # doctest: +SKIP

Search results for 'binary representation'

------------------------------------------

numpy.binary\_repr

Return the binary representation of the input number as a string.

numpy.core.setup\_common.long\_double\_representation

Given a binary dump as given by GNU od -b, look for long double

numpy.base\_repr

Return a string representation of a number in the given base system.

...

mafromtxt(fname, \*\*kwargs)

Load ASCII data stored in a text file and return a masked array.

.. deprecated:: 1.17

np.mafromtxt is a deprecated alias of `genfromtxt` which

overwrites the ``usemask`` argument with `True` even when

explicitly called as ``mafromtxt(..., usemask=False)``.

Use `genfromtxt` instead.

Parameters

----------

fname, kwargs : For a description of input parameters, see `genfromtxt`.

See Also

--------

numpy.genfromtxt : generic function to load ASCII data.

mask\_indices(n, mask\_func, k=0)

Return the indices to access (n, n) arrays, given a masking function.

Assume `mask\_func` is a function that, for a square array a of size

``(n, n)`` with a possible offset argument `k`, when called as

``mask\_func(a, k)`` returns a new array with zeros in certain locations

(functions like `triu` or `tril` do precisely this). Then this function

returns the indices where the non-zero values would be located.

Parameters

----------

n : int

The returned indices will be valid to access arrays of shape (n, n).

mask\_func : callable

A function whose call signature is similar to that of `triu`, `tril`.

That is, ``mask\_func(x, k)`` returns a boolean array, shaped like `x`.

`k` is an optional argument to the function.

k : scalar

An optional argument which is passed through to `mask\_func`. Functions

like `triu`, `tril` take a second argument that is interpreted as an

offset.

Returns

-------

indices : tuple of arrays.

The `n` arrays of indices corresponding to the locations where

``mask\_func(np.ones((n, n)), k)`` is True.

See Also

--------

triu, tril, triu\_indices, tril\_indices

Notes

-----

.. versionadded:: 1.4.0

Examples

--------

These are the indices that would allow you to access the upper triangular

part of any 3x3 array:

>>> iu = np.mask\_indices(3, np.triu)

For example, if `a` is a 3x3 array:

>>> a = np.arange(9).reshape(3, 3)

>>> a

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

>>> a[iu]

array([0, 1, 2, 4, 5, 8])

An offset can be passed also to the masking function. This gets us the

indices starting on the first diagonal right of the main one:

>>> iu1 = np.mask\_indices(3, np.triu, 1)

with which we now extract only three elements:

>>> a[iu1]

array([1, 2, 5])

mat = asmatrix(data, dtype=None)

Interpret the input as a matrix.

Unlike `matrix`, `asmatrix` does not make a copy if the input is already

a matrix or an ndarray. Equivalent to ``matrix(data, copy=False)``.

Parameters

----------

data : array\_like

Input data.

dtype : data-type

Data-type of the output matrix.

Returns

-------

mat : matrix

`data` interpreted as a matrix.

Examples

--------

>>> x = np.array([[1, 2], [3, 4]])

>>> m = np.asmatrix(x)

>>> x[0,0] = 5

>>> m

matrix([[5, 2],

[3, 4]])

maximum\_sctype(t)

Return the scalar type of highest precision of the same kind as the input.

Parameters

----------

t : dtype or dtype specifier

The input data type. This can be a `dtype` object or an object that

is convertible to a `dtype`.

Returns

-------

out : dtype

The highest precision data type of the same kind (`dtype.kind`) as `t`.

See Also

--------

obj2sctype, mintypecode, sctype2char

dtype

Examples

--------

>>> np.maximum\_sctype(int)

<class 'numpy.int64'>

>>> np.maximum\_sctype(np.uint8)

<class 'numpy.uint64'>

>>> np.maximum\_sctype(complex)

<class 'numpy.complex256'> # may vary

>>> np.maximum\_sctype(str)

<class 'numpy.str\_'>

>>> np.maximum\_sctype('i2')

<class 'numpy.int64'>

>>> np.maximum\_sctype('f4')

<class 'numpy.float128'> # may vary

may\_share\_memory(...)

may\_share\_memory(a, b, max\_work=None)

Determine if two arrays might share memory

A return of True does not necessarily mean that the two arrays

share any element. It just means that they \*might\*.

Only the memory bounds of a and b are checked by default.

Parameters

----------

a, b : ndarray

Input arrays

max\_work : int, optional

Effort to spend on solving the overlap problem. See

`shares\_memory` for details. Default for ``may\_share\_memory``

is to do a bounds check.

Returns

-------

out : bool

See Also

--------

shares\_memory

Examples

--------

>>> np.may\_share\_memory(np.array([1,2]), np.array([5,8,9]))

False

>>> x = np.zeros([3, 4])

>>> np.may\_share\_memory(x[:,0], x[:,1])

True

mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over

the flattened array by default, otherwise over the specified axis.

`float64` intermediate and return values are used for integer inputs.

Parameters

----------

a : array\_like

Array containing numbers whose mean is desired. If `a` is not an

array, a conversion is attempted.

axis : None or int or tuple of ints, optional

Axis or axes along which the means are computed. The default is to

compute the mean of the flattened array.

.. versionadded:: 1.7.0

If this is a tuple of ints, a mean is performed over multiple axes,

instead of a single axis or all the axes as before.

dtype : data-type, optional

Type to use in computing the mean. For integer inputs, the default

is `float64`; for floating point inputs, it is the same as the

input dtype.

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``; if provided, it must have the same shape as the

expected output, but the type will be cast if necessary.

See `doc.ufuncs` for details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `mean` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

m : ndarray, see dtype parameter above

If `out=None`, returns a new array containing the mean values,

otherwise a reference to the output array is returned.

See Also

--------

average : Weighted average

std, var, nanmean, nanstd, nanvar

Notes

-----

The arithmetic mean is the sum of the elements along the axis divided

by the number of elements.

Note that for floating-point input, the mean is computed using the

same precision the input has. Depending on the input data, this can

cause the results to be inaccurate, especially for `float32` (see

example below). Specifying a higher-precision accumulator using the

`dtype` keyword can alleviate this issue.

By default, `float16` results are computed using `float32` intermediates

for extra precision.

Examples

--------

>>> a = np.array([[1, 2], [3, 4]])

>>> np.mean(a)

2.5

>>> np.mean(a, axis=0)

array([2., 3.])

>>> np.mean(a, axis=1)

array([1.5, 3.5])

In single precision, `mean` can be inaccurate:

>>> a = np.zeros((2, 512\*512), dtype=np.float32)

>>> a[0, :] = 1.0

>>> a[1, :] = 0.1

>>> np.mean(a)

0.54999924

Computing the mean in float64 is more accurate:

>>> np.mean(a, dtype=np.float64)

0.55000000074505806 # may vary

median(a, axis=None, out=None, overwrite\_input=False, keepdims=False)

Compute the median along the specified axis.

Returns the median of the array elements.

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

axis : {int, sequence of int, None}, optional

Axis or axes along which the medians are computed. The default

is to compute the median along a flattened version of the array.

A sequence of axes is supported since version 1.9.0.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow use of memory of input array `a` for

calculations. The input array will be modified by the call to

`median`. This will save memory when you do not need to preserve

the contents of the input array. Treat the input as undefined,

but it will probably be fully or partially sorted. Default is

False. If `overwrite\_input` is ``True`` and `a` is not already an

`ndarray`, an error will be raised.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `arr`.

.. versionadded:: 1.9.0

Returns

-------

median : ndarray

A new array holding the result. If the input contains integers

or floats smaller than ``float64``, then the output data-type is

``np.float64``. Otherwise, the data-type of the output is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

mean, percentile

Notes

-----

Given a vector ``V`` of length ``N``, the median of ``V`` is the

middle value of a sorted copy of ``V``, ``V\_sorted`` - i

e., ``V\_sorted[(N-1)/2]``, when ``N`` is odd, and the average of the

two middle values of ``V\_sorted`` when ``N`` is even.

Examples

--------

>>> a = np.array([[10, 7, 4], [3, 2, 1]])

>>> a

array([[10, 7, 4],

[ 3, 2, 1]])

>>> np.median(a)

3.5

>>> np.median(a, axis=0)

array([6.5, 4.5, 2.5])

>>> np.median(a, axis=1)

array([7., 2.])

>>> m = np.median(a, axis=0)

>>> out = np.zeros\_like(m)

>>> np.median(a, axis=0, out=m)

array([6.5, 4.5, 2.5])

>>> m

array([6.5, 4.5, 2.5])

>>> b = a.copy()

>>> np.median(b, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a==b)

>>> b = a.copy()

>>> np.median(b, axis=None, overwrite\_input=True)

3.5

>>> assert not np.all(a==b)

meshgrid(\*xi, \*\*kwargs)

Return coordinate matrices from coordinate vectors.

Make N-D coordinate arrays for vectorized evaluations of

N-D scalar/vector fields over N-D grids, given

one-dimensional coordinate arrays x1, x2,..., xn.

.. versionchanged:: 1.9

1-D and 0-D cases are allowed.

Parameters

----------

x1, x2,..., xn : array\_like

1-D arrays representing the coordinates of a grid.

indexing : {'xy', 'ij'}, optional

Cartesian ('xy', default) or matrix ('ij') indexing of output.

See Notes for more details.

.. versionadded:: 1.7.0

sparse : bool, optional

If True a sparse grid is returned in order to conserve memory.

Default is False.

.. versionadded:: 1.7.0

copy : bool, optional

If False, a view into the original arrays are returned in order to

conserve memory. Default is True. Please note that

``sparse=False, copy=False`` will likely return non-contiguous

arrays. Furthermore, more than one element of a broadcast array

may refer to a single memory location. If you need to write to the

arrays, make copies first.

.. versionadded:: 1.7.0

Returns

-------

X1, X2,..., XN : ndarray

For vectors `x1`, `x2`,..., 'xn' with lengths ``Ni=len(xi)`` ,

return ``(N1, N2, N3,...Nn)`` shaped arrays if indexing='ij'

or ``(N2, N1, N3,...Nn)`` shaped arrays if indexing='xy'

with the elements of `xi` repeated to fill the matrix along

the first dimension for `x1`, the second for `x2` and so on.

Notes

-----

This function supports both indexing conventions through the indexing

keyword argument. Giving the string 'ij' returns a meshgrid with

matrix indexing, while 'xy' returns a meshgrid with Cartesian indexing.

In the 2-D case with inputs of length M and N, the outputs are of shape

(N, M) for 'xy' indexing and (M, N) for 'ij' indexing. In the 3-D case

with inputs of length M, N and P, outputs are of shape (N, M, P) for

'xy' indexing and (M, N, P) for 'ij' indexing. The difference is

illustrated by the following code snippet::

xv, yv = np.meshgrid(x, y, sparse=False, indexing='ij')

for i in range(nx):

for j in range(ny):

# treat xv[i,j], yv[i,j]

xv, yv = np.meshgrid(x, y, sparse=False, indexing='xy')

for i in range(nx):

for j in range(ny):

# treat xv[j,i], yv[j,i]

In the 1-D and 0-D case, the indexing and sparse keywords have no effect.

See Also

--------

index\_tricks.mgrid : Construct a multi-dimensional "meshgrid"

using indexing notation.

index\_tricks.ogrid : Construct an open multi-dimensional "meshgrid"

using indexing notation.

Examples

--------

>>> nx, ny = (3, 2)

>>> x = np.linspace(0, 1, nx)

>>> y = np.linspace(0, 1, ny)

>>> xv, yv = np.meshgrid(x, y)

>>> xv

array([[0. , 0.5, 1. ],

[0. , 0.5, 1. ]])

>>> yv

array([[0., 0., 0.],

[1., 1., 1.]])

>>> xv, yv = np.meshgrid(x, y, sparse=True) # make sparse output arrays

>>> xv

array([[0. , 0.5, 1. ]])

>>> yv

array([[0.],

[1.]])

`meshgrid` is very useful to evaluate functions on a grid.

>>> import matplotlib.pyplot as plt

>>> x = np.arange(-5, 5, 0.1)

>>> y = np.arange(-5, 5, 0.1)

>>> xx, yy = np.meshgrid(x, y, sparse=True)

>>> z = np.sin(xx\*\*2 + yy\*\*2) / (xx\*\*2 + yy\*\*2)

>>> h = plt.contourf(x,y,z)

>>> plt.show()

min\_scalar\_type(...)

min\_scalar\_type(a)

For scalar ``a``, returns the data type with the smallest size

and smallest scalar kind which can hold its value. For non-scalar

array ``a``, returns the vector's dtype unmodified.

Floating point values are not demoted to integers,

and complex values are not demoted to floats.

Parameters

----------

a : scalar or array\_like

The value whose minimal data type is to be found.

Returns

-------

out : dtype

The minimal data type.

Notes

-----

.. versionadded:: 1.6.0

See Also

--------

result\_type, promote\_types, dtype, can\_cast

Examples

--------

>>> np.min\_scalar\_type(10)

dtype('uint8')

>>> np.min\_scalar\_type(-260)

dtype('int16')

>>> np.min\_scalar\_type(3.1)

dtype('float16')

>>> np.min\_scalar\_type(1e50)

dtype('float64')

>>> np.min\_scalar\_type(np.arange(4,dtype='f8'))

dtype('float64')

mintypecode(typechars, typeset='GDFgdf', default='d')

Return the character for the minimum-size type to which given types can

be safely cast.

The returned type character must represent the smallest size dtype such

that an array of the returned type can handle the data from an array of

all types in `typechars` (or if `typechars` is an array, then its

dtype.char).

Parameters

----------

typechars : list of str or array\_like

If a list of strings, each string should represent a dtype.

If array\_like, the character representation of the array dtype is used.

typeset : str or list of str, optional

The set of characters that the returned character is chosen from.

The default set is 'GDFgdf'.

default : str, optional

The default character, this is returned if none of the characters in

`typechars` matches a character in `typeset`.

Returns

-------

typechar : str

The character representing the minimum-size type that was found.

See Also

--------

dtype, sctype2char, maximum\_sctype

Examples

--------

>>> np.mintypecode(['d', 'f', 'S'])

'd'

>>> x = np.array([1.1, 2-3.j])

>>> np.mintypecode(x)

'D'

>>> np.mintypecode('abceh', default='G')

'G'

mirr(values, finance\_rate, reinvest\_rate)

Modified internal rate of return.

Parameters

----------

values : array\_like

Cash flows (must contain at least one positive and one negative

value) or nan is returned. The first value is considered a sunk

cost at time zero.

finance\_rate : scalar

Interest rate paid on the cash flows

reinvest\_rate : scalar

Interest rate received on the cash flows upon reinvestment

Returns

-------

out : float

Modified internal rate of return

moveaxis(a, source, destination)

Move axes of an array to new positions.

Other axes remain in their original order.

.. versionadded:: 1.11.0

Parameters

----------

a : np.ndarray

The array whose axes should be reordered.

source : int or sequence of int

Original positions of the axes to move. These must be unique.

destination : int or sequence of int

Destination positions for each of the original axes. These must also be

unique.

Returns

-------

result : np.ndarray

Array with moved axes. This array is a view of the input array.

See Also

--------

transpose: Permute the dimensions of an array.

swapaxes: Interchange two axes of an array.

Examples

--------

>>> x = np.zeros((3, 4, 5))

>>> np.moveaxis(x, 0, -1).shape

(4, 5, 3)

>>> np.moveaxis(x, -1, 0).shape

(5, 3, 4)

These all achieve the same result:

>>> np.transpose(x).shape

(5, 4, 3)

>>> np.swapaxes(x, 0, -1).shape

(5, 4, 3)

>>> np.moveaxis(x, [0, 1], [-1, -2]).shape

(5, 4, 3)

>>> np.moveaxis(x, [0, 1, 2], [-1, -2, -3]).shape

(5, 4, 3)

msort(a)

Return a copy of an array sorted along the first axis.

Parameters

----------

a : array\_like

Array to be sorted.

Returns

-------

sorted\_array : ndarray

Array of the same type and shape as `a`.

See Also

--------

sort

Notes

-----

``np.msort(a)`` is equivalent to ``np.sort(a, axis=0)``.

nan\_to\_num(x, copy=True, nan=0.0, posinf=None, neginf=None)

Replace NaN with zero and infinity with large finite numbers (default

behaviour) or with the numbers defined by the user using the `nan`,

`posinf` and/or `neginf` keywords.

If `x` is inexact, NaN is replaced by zero or by the user defined value in

`nan` keyword, infinity is replaced by the largest finite floating point

values representable by ``x.dtype`` or by the user defined value in

`posinf` keyword and -infinity is replaced by the most negative finite

floating point values representable by ``x.dtype`` or by the user defined

value in `neginf` keyword.

For complex dtypes, the above is applied to each of the real and

imaginary components of `x` separately.

If `x` is not inexact, then no replacements are made.

Parameters

----------

x : scalar or array\_like

Input data.

copy : bool, optional

Whether to create a copy of `x` (True) or to replace values

in-place (False). The in-place operation only occurs if

casting to an array does not require a copy.

Default is True.

nan : int, float, optional

Value to be used to fill NaN values. If no value is passed

then NaN values will be replaced with 0.0.

posinf : int, float, optional

Value to be used to fill positive infinity values. If no value is

passed then positive infinity values will be replaced with a very

large number.

neginf : int, float, optional

Value to be used to fill negative infinity values. If no value is

passed then negative infinity values will be replaced with a very

small (or negative) number.

.. versionadded:: 1.13

Returns

-------

out : ndarray

`x`, with the non-finite values replaced. If `copy` is False, this may

be `x` itself.

See Also

--------

isinf : Shows which elements are positive or negative infinity.

isneginf : Shows which elements are negative infinity.

isposinf : Shows which elements are positive infinity.

isnan : Shows which elements are Not a Number (NaN).

isfinite : Shows which elements are finite (not NaN, not infinity)

Notes

-----

NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic

(IEEE 754). This means that Not a Number is not equivalent to infinity.

Examples

--------

>>> np.nan\_to\_num(np.inf)

1.7976931348623157e+308

>>> np.nan\_to\_num(-np.inf)

-1.7976931348623157e+308

>>> np.nan\_to\_num(np.nan)

0.0

>>> x = np.array([np.inf, -np.inf, np.nan, -128, 128])

>>> np.nan\_to\_num(x)

array([ 1.79769313e+308, -1.79769313e+308, 0.00000000e+000, # may vary

-1.28000000e+002, 1.28000000e+002])

>>> np.nan\_to\_num(x, nan=-9999, posinf=33333333, neginf=33333333)

array([ 3.3333333e+07, 3.3333333e+07, -9.9990000e+03,

-1.2800000e+02, 1.2800000e+02])

>>> y = np.array([complex(np.inf, np.nan), np.nan, complex(np.nan, np.inf)])

array([ 1.79769313e+308, -1.79769313e+308, 0.00000000e+000, # may vary

-1.28000000e+002, 1.28000000e+002])

>>> np.nan\_to\_num(y)

array([ 1.79769313e+308 +0.00000000e+000j, # may vary

0.00000000e+000 +0.00000000e+000j,

0.00000000e+000 +1.79769313e+308j])

>>> np.nan\_to\_num(y, nan=111111, posinf=222222)

array([222222.+111111.j, 111111. +0.j, 111111.+222222.j])

nanargmax(a, axis=None)

Return the indices of the maximum values in the specified axis ignoring

NaNs. For all-NaN slices ``ValueError`` is raised. Warning: the

results cannot be trusted if a slice contains only NaNs and -Infs.

Parameters

----------

a : array\_like

Input data.

axis : int, optional

Axis along which to operate. By default flattened input is used.

Returns

-------

index\_array : ndarray

An array of indices or a single index value.

See Also

--------

argmax, nanargmin

Examples

--------

>>> a = np.array([[np.nan, 4], [2, 3]])

>>> np.argmax(a)

0

>>> np.nanargmax(a)

1

>>> np.nanargmax(a, axis=0)

array([1, 0])

>>> np.nanargmax(a, axis=1)

array([1, 1])

nanargmin(a, axis=None)

Return the indices of the minimum values in the specified axis ignoring

NaNs. For all-NaN slices ``ValueError`` is raised. Warning: the results

cannot be trusted if a slice contains only NaNs and Infs.

Parameters

----------

a : array\_like

Input data.

axis : int, optional

Axis along which to operate. By default flattened input is used.

Returns

-------

index\_array : ndarray

An array of indices or a single index value.

See Also

--------

argmin, nanargmax

Examples

--------

>>> a = np.array([[np.nan, 4], [2, 3]])

>>> np.argmin(a)

0

>>> np.nanargmin(a)

2

>>> np.nanargmin(a, axis=0)

array([1, 1])

>>> np.nanargmin(a, axis=1)

array([1, 0])

nancumprod(a, axis=None, dtype=None, out=None)

Return the cumulative product of array elements over a given axis treating Not a

Numbers (NaNs) as one. The cumulative product does not change when NaNs are

encountered and leading NaNs are replaced by ones.

Ones are returned for slices that are all-NaN or empty.

.. versionadded:: 1.12.0

Parameters

----------

a : array\_like

Input array.

axis : int, optional

Axis along which the cumulative product is computed. By default

the input is flattened.

dtype : dtype, optional

Type of the returned array, as well as of the accumulator in which

the elements are multiplied. If \*dtype\* is not specified, it

defaults to the dtype of `a`, unless `a` has an integer dtype with

a precision less than that of the default platform integer. In

that case, the default platform integer is used instead.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output

but the type of the resulting values will be cast if necessary.

Returns

-------

nancumprod : ndarray

A new array holding the result is returned unless `out` is

specified, in which case it is returned.

See Also

--------

numpy.cumprod : Cumulative product across array propagating NaNs.

isnan : Show which elements are NaN.

Examples

--------

>>> np.nancumprod(1)

array([1])

>>> np.nancumprod([1])

array([1])

>>> np.nancumprod([1, np.nan])

array([1., 1.])

>>> a = np.array([[1, 2], [3, np.nan]])

>>> np.nancumprod(a)

array([1., 2., 6., 6.])

>>> np.nancumprod(a, axis=0)

array([[1., 2.],

[3., 2.]])

>>> np.nancumprod(a, axis=1)

array([[1., 2.],

[3., 3.]])

nancumsum(a, axis=None, dtype=None, out=None)

Return the cumulative sum of array elements over a given axis treating Not a

Numbers (NaNs) as zero. The cumulative sum does not change when NaNs are

encountered and leading NaNs are replaced by zeros.

Zeros are returned for slices that are all-NaN or empty.

.. versionadded:: 1.12.0

Parameters

----------

a : array\_like

Input array.

axis : int, optional

Axis along which the cumulative sum is computed. The default

(None) is to compute the cumsum over the flattened array.

dtype : dtype, optional

Type of the returned array and of the accumulator in which the

elements are summed. If `dtype` is not specified, it defaults

to the dtype of `a`, unless `a` has an integer dtype with a

precision less than that of the default platform integer. In

that case, the default platform integer is used.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output

but the type will be cast if necessary. See `doc.ufuncs`

(Section "Output arguments") for more details.

Returns

-------

nancumsum : ndarray.

A new array holding the result is returned unless `out` is

specified, in which it is returned. The result has the same

size as `a`, and the same shape as `a` if `axis` is not None

or `a` is a 1-d array.

See Also

--------

numpy.cumsum : Cumulative sum across array propagating NaNs.

isnan : Show which elements are NaN.

Examples

--------

>>> np.nancumsum(1)

array([1])

>>> np.nancumsum([1])

array([1])

>>> np.nancumsum([1, np.nan])

array([1., 1.])

>>> a = np.array([[1, 2], [3, np.nan]])

>>> np.nancumsum(a)

array([1., 3., 6., 6.])

>>> np.nancumsum(a, axis=0)

array([[1., 2.],

[4., 2.]])

>>> np.nancumsum(a, axis=1)

array([[1., 3.],

[3., 3.]])

nanmax(a, axis=None, out=None, keepdims=<no value>)

Return the maximum of an array or maximum along an axis, ignoring any

NaNs. When all-NaN slices are encountered a ``RuntimeWarning`` is

raised and NaN is returned for that slice.

Parameters

----------

a : array\_like

Array containing numbers whose maximum is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the maximum is computed. The default is to compute

the maximum of the flattened array.

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``; if provided, it must have the same shape as the

expected output, but the type will be cast if necessary. See

`doc.ufuncs` for details.

.. versionadded:: 1.8.0

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If the value is anything but the default, then

`keepdims` will be passed through to the `max` method

of sub-classes of `ndarray`. If the sub-classes methods

does not implement `keepdims` any exceptions will be raised.

.. versionadded:: 1.8.0

Returns

-------

nanmax : ndarray

An array with the same shape as `a`, with the specified axis removed.

If `a` is a 0-d array, or if axis is None, an ndarray scalar is

returned. The same dtype as `a` is returned.

See Also

--------

nanmin :

The minimum value of an array along a given axis, ignoring any NaNs.

amax :

The maximum value of an array along a given axis, propagating any NaNs.

fmax :

Element-wise maximum of two arrays, ignoring any NaNs.

maximum :

Element-wise maximum of two arrays, propagating any NaNs.

isnan :

Shows which elements are Not a Number (NaN).

isfinite:

Shows which elements are neither NaN nor infinity.

amin, fmin, minimum

Notes

-----

NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic

(IEEE 754). This means that Not a Number is not equivalent to infinity.

Positive infinity is treated as a very large number and negative

infinity is treated as a very small (i.e. negative) number.

If the input has a integer type the function is equivalent to np.max.

Examples

--------

>>> a = np.array([[1, 2], [3, np.nan]])

>>> np.nanmax(a)

3.0

>>> np.nanmax(a, axis=0)

array([3., 2.])

>>> np.nanmax(a, axis=1)

array([2., 3.])

When positive infinity and negative infinity are present:

>>> np.nanmax([1, 2, np.nan, np.NINF])

2.0

>>> np.nanmax([1, 2, np.nan, np.inf])

inf

nanmean(a, axis=None, dtype=None, out=None, keepdims=<no value>)

Compute the arithmetic mean along the specified axis, ignoring NaNs.

Returns the average of the array elements. The average is taken over

the flattened array by default, otherwise over the specified axis.

`float64` intermediate and return values are used for integer inputs.

For all-NaN slices, NaN is returned and a `RuntimeWarning` is raised.

.. versionadded:: 1.8.0

Parameters

----------

a : array\_like

Array containing numbers whose mean is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the means are computed. The default is to compute

the mean of the flattened array.

dtype : data-type, optional

Type to use in computing the mean. For integer inputs, the default

is `float64`; for inexact inputs, it is the same as the input

dtype.

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``; if provided, it must have the same shape as the

expected output, but the type will be cast if necessary. See

`doc.ufuncs` for details.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If the value is anything but the default, then

`keepdims` will be passed through to the `mean` or `sum` methods

of sub-classes of `ndarray`. If the sub-classes methods

does not implement `keepdims` any exceptions will be raised.

Returns

-------

m : ndarray, see dtype parameter above

If `out=None`, returns a new array containing the mean values,

otherwise a reference to the output array is returned. Nan is

returned for slices that contain only NaNs.

See Also

--------

average : Weighted average

mean : Arithmetic mean taken while not ignoring NaNs

var, nanvar

Notes

-----

The arithmetic mean is the sum of the non-NaN elements along the axis

divided by the number of non-NaN elements.

Note that for floating-point input, the mean is computed using the same

precision the input has. Depending on the input data, this can cause

the results to be inaccurate, especially for `float32`. Specifying a

higher-precision accumulator using the `dtype` keyword can alleviate

this issue.

Examples

--------

>>> a = np.array([[1, np.nan], [3, 4]])

>>> np.nanmean(a)

2.6666666666666665

>>> np.nanmean(a, axis=0)

array([2., 4.])

>>> np.nanmean(a, axis=1)

array([1., 3.5]) # may vary

nanmedian(a, axis=None, out=None, overwrite\_input=False, keepdims=<no value>)

Compute the median along the specified axis, while ignoring NaNs.

Returns the median of the array elements.

.. versionadded:: 1.9.0

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

axis : {int, sequence of int, None}, optional

Axis or axes along which the medians are computed. The default

is to compute the median along a flattened version of the array.

A sequence of axes is supported since version 1.9.0.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow use of memory of input array `a` for

calculations. The input array will be modified by the call to

`median`. This will save memory when you do not need to preserve

the contents of the input array. Treat the input as undefined,

but it will probably be fully or partially sorted. Default is

False. If `overwrite\_input` is ``True`` and `a` is not already an

`ndarray`, an error will be raised.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If this is anything but the default value it will be passed

through (in the special case of an empty array) to the

`mean` function of the underlying array. If the array is

a sub-class and `mean` does not have the kwarg `keepdims` this

will raise a RuntimeError.

Returns

-------

median : ndarray

A new array holding the result. If the input contains integers

or floats smaller than ``float64``, then the output data-type is

``np.float64``. Otherwise, the data-type of the output is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

mean, median, percentile

Notes

-----

Given a vector ``V`` of length ``N``, the median of ``V`` is the

middle value of a sorted copy of ``V``, ``V\_sorted`` - i.e.,

``V\_sorted[(N-1)/2]``, when ``N`` is odd and the average of the two

middle values of ``V\_sorted`` when ``N`` is even.

Examples

--------

>>> a = np.array([[10.0, 7, 4], [3, 2, 1]])

>>> a[0, 1] = np.nan

>>> a

array([[10., nan, 4.],

[ 3., 2., 1.]])

>>> np.median(a)

nan

>>> np.nanmedian(a)

3.0

>>> np.nanmedian(a, axis=0)

array([6.5, 2. , 2.5])

>>> np.median(a, axis=1)

array([nan, 2.])

>>> b = a.copy()

>>> np.nanmedian(b, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a==b)

>>> b = a.copy()

>>> np.nanmedian(b, axis=None, overwrite\_input=True)

3.0

>>> assert not np.all(a==b)

nanmin(a, axis=None, out=None, keepdims=<no value>)

Return minimum of an array or minimum along an axis, ignoring any NaNs.

When all-NaN slices are encountered a ``RuntimeWarning`` is raised and

Nan is returned for that slice.

Parameters

----------

a : array\_like

Array containing numbers whose minimum is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the minimum is computed. The default is to compute

the minimum of the flattened array.

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``; if provided, it must have the same shape as the

expected output, but the type will be cast if necessary. See

`doc.ufuncs` for details.

.. versionadded:: 1.8.0

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If the value is anything but the default, then

`keepdims` will be passed through to the `min` method

of sub-classes of `ndarray`. If the sub-classes methods

does not implement `keepdims` any exceptions will be raised.

.. versionadded:: 1.8.0

Returns

-------

nanmin : ndarray

An array with the same shape as `a`, with the specified axis

removed. If `a` is a 0-d array, or if axis is None, an ndarray

scalar is returned. The same dtype as `a` is returned.

See Also

--------

nanmax :

The maximum value of an array along a given axis, ignoring any NaNs.

amin :

The minimum value of an array along a given axis, propagating any NaNs.

fmin :

Element-wise minimum of two arrays, ignoring any NaNs.

minimum :

Element-wise minimum of two arrays, propagating any NaNs.

isnan :

Shows which elements are Not a Number (NaN).

isfinite:

Shows which elements are neither NaN nor infinity.

amax, fmax, maximum

Notes

-----

NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic

(IEEE 754). This means that Not a Number is not equivalent to infinity.

Positive infinity is treated as a very large number and negative

infinity is treated as a very small (i.e. negative) number.

If the input has a integer type the function is equivalent to np.min.

Examples

--------

>>> a = np.array([[1, 2], [3, np.nan]])

>>> np.nanmin(a)

1.0

>>> np.nanmin(a, axis=0)

array([1., 2.])

>>> np.nanmin(a, axis=1)

array([1., 3.])

When positive infinity and negative infinity are present:

>>> np.nanmin([1, 2, np.nan, np.inf])

1.0

>>> np.nanmin([1, 2, np.nan, np.NINF])

-inf

nanpercentile(a, q, axis=None, out=None, overwrite\_input=False, interpolation='linear', keepdims=<no value>)

Compute the qth percentile of the data along the specified axis,

while ignoring nan values.

Returns the qth percentile(s) of the array elements.

.. versionadded:: 1.9.0

Parameters

----------

a : array\_like

Input array or object that can be converted to an array, containing

nan values to be ignored.

q : array\_like of float

Percentile or sequence of percentiles to compute, which must be between

0 and 100 inclusive.

axis : {int, tuple of int, None}, optional

Axis or axes along which the percentiles are computed. The

default is to compute the percentile(s) along a flattened

version of the array.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow the input array `a` to be modified by intermediate

calculations, to save memory. In this case, the contents of the input

`a` after this function completes is undefined.

interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}

This optional parameter specifies the interpolation method to

use when the desired percentile lies between two data points

``i < j``:

\* 'linear': ``i + (j - i) \* fraction``, where ``fraction``

is the fractional part of the index surrounded by ``i``

and ``j``.

\* 'lower': ``i``.

\* 'higher': ``j``.

\* 'nearest': ``i`` or ``j``, whichever is nearest.

\* 'midpoint': ``(i + j) / 2``.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left in

the result as dimensions with size one. With this option, the

result will broadcast correctly against the original array `a`.

If this is anything but the default value it will be passed

through (in the special case of an empty array) to the

`mean` function of the underlying array. If the array is

a sub-class and `mean` does not have the kwarg `keepdims` this

will raise a RuntimeError.

Returns

-------

percentile : scalar or ndarray

If `q` is a single percentile and `axis=None`, then the result

is a scalar. If multiple percentiles are given, first axis of

the result corresponds to the percentiles. The other axes are

the axes that remain after the reduction of `a`. If the input

contains integers or floats smaller than ``float64``, the output

data-type is ``float64``. Otherwise, the output data-type is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

nanmean

nanmedian : equivalent to ``nanpercentile(..., 50)``

percentile, median, mean

nanquantile : equivalent to nanpercentile, but with q in the range [0, 1].

Notes

-----

Given a vector ``V`` of length ``N``, the ``q``-th percentile of

``V`` is the value ``q/100`` of the way from the minimum to the

maximum in a sorted copy of ``V``. The values and distances of

the two nearest neighbors as well as the `interpolation` parameter

will determine the percentile if the normalized ranking does not

match the location of ``q`` exactly. This function is the same as

the median if ``q=50``, the same as the minimum if ``q=0`` and the

same as the maximum if ``q=100``.

Examples

--------

>>> a = np.array([[10., 7., 4.], [3., 2., 1.]])

>>> a[0][1] = np.nan

>>> a

array([[10., nan, 4.],

[ 3., 2., 1.]])

>>> np.percentile(a, 50)

nan

>>> np.nanpercentile(a, 50)

3.0

>>> np.nanpercentile(a, 50, axis=0)

array([6.5, 2. , 2.5])

>>> np.nanpercentile(a, 50, axis=1, keepdims=True)

array([[7.],

[2.]])

>>> m = np.nanpercentile(a, 50, axis=0)

>>> out = np.zeros\_like(m)

>>> np.nanpercentile(a, 50, axis=0, out=out)

array([6.5, 2. , 2.5])

>>> m

array([6.5, 2. , 2.5])

>>> b = a.copy()

>>> np.nanpercentile(b, 50, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a==b)

nanprod(a, axis=None, dtype=None, out=None, keepdims=<no value>)

Return the product of array elements over a given axis treating Not a

Numbers (NaNs) as ones.

One is returned for slices that are all-NaN or empty.

.. versionadded:: 1.10.0

Parameters

----------

a : array\_like

Array containing numbers whose product is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the product is computed. The default is to compute

the product of the flattened array.

dtype : data-type, optional

The type of the returned array and of the accumulator in which the

elements are summed. By default, the dtype of `a` is used. An

exception is when `a` has an integer type with less precision than

the platform (u)intp. In that case, the default will be either

(u)int32 or (u)int64 depending on whether the platform is 32 or 64

bits. For inexact inputs, dtype must be inexact.

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``. If provided, it must have the same shape as the

expected output, but the type will be cast if necessary. See

`doc.ufuncs` for details. The casting of NaN to integer can yield

unexpected results.

keepdims : bool, optional

If True, the axes which are reduced are left in the result as

dimensions with size one. With this option, the result will

broadcast correctly against the original `arr`.

Returns

-------

nanprod : ndarray

A new array holding the result is returned unless `out` is

specified, in which case it is returned.

See Also

--------

numpy.prod : Product across array propagating NaNs.

isnan : Show which elements are NaN.

Examples

--------

>>> np.nanprod(1)

1

>>> np.nanprod([1])

1

>>> np.nanprod([1, np.nan])

1.0

>>> a = np.array([[1, 2], [3, np.nan]])

>>> np.nanprod(a)

6.0

>>> np.nanprod(a, axis=0)

array([3., 2.])

nanquantile(a, q, axis=None, out=None, overwrite\_input=False, interpolation='linear', keepdims=<no value>)

Compute the qth quantile of the data along the specified axis,

while ignoring nan values.

Returns the qth quantile(s) of the array elements.

.. versionadded:: 1.15.0

Parameters

----------

a : array\_like

Input array or object that can be converted to an array, containing

nan values to be ignored

q : array\_like of float

Quantile or sequence of quantiles to compute, which must be between

0 and 1 inclusive.

axis : {int, tuple of int, None}, optional

Axis or axes along which the quantiles are computed. The

default is to compute the quantile(s) along a flattened

version of the array.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow the input array `a` to be modified by intermediate

calculations, to save memory. In this case, the contents of the input

`a` after this function completes is undefined.

interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}

This optional parameter specifies the interpolation method to

use when the desired quantile lies between two data points

``i < j``:

\* linear: ``i + (j - i) \* fraction``, where ``fraction``

is the fractional part of the index surrounded by ``i``

and ``j``.

\* lower: ``i``.

\* higher: ``j``.

\* nearest: ``i`` or ``j``, whichever is nearest.

\* midpoint: ``(i + j) / 2``.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left in

the result as dimensions with size one. With this option, the

result will broadcast correctly against the original array `a`.

If this is anything but the default value it will be passed

through (in the special case of an empty array) to the

`mean` function of the underlying array. If the array is

a sub-class and `mean` does not have the kwarg `keepdims` this

will raise a RuntimeError.

Returns

-------

quantile : scalar or ndarray

If `q` is a single percentile and `axis=None`, then the result

is a scalar. If multiple quantiles are given, first axis of

the result corresponds to the quantiles. The other axes are

the axes that remain after the reduction of `a`. If the input

contains integers or floats smaller than ``float64``, the output

data-type is ``float64``. Otherwise, the output data-type is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

quantile

nanmean, nanmedian

nanmedian : equivalent to ``nanquantile(..., 0.5)``

nanpercentile : same as nanquantile, but with q in the range [0, 100].

Examples

--------

>>> a = np.array([[10., 7., 4.], [3., 2., 1.]])

>>> a[0][1] = np.nan

>>> a

array([[10., nan, 4.],

[ 3., 2., 1.]])

>>> np.quantile(a, 0.5)

nan

>>> np.nanquantile(a, 0.5)

3.0

>>> np.nanquantile(a, 0.5, axis=0)

array([6.5, 2. , 2.5])

>>> np.nanquantile(a, 0.5, axis=1, keepdims=True)

array([[7.],

[2.]])

>>> m = np.nanquantile(a, 0.5, axis=0)

>>> out = np.zeros\_like(m)

>>> np.nanquantile(a, 0.5, axis=0, out=out)

array([6.5, 2. , 2.5])

>>> m

array([6.5, 2. , 2.5])

>>> b = a.copy()

>>> np.nanquantile(b, 0.5, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a==b)

nanstd(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Compute the standard deviation along the specified axis, while

ignoring NaNs.

Returns the standard deviation, a measure of the spread of a

distribution, of the non-NaN array elements. The standard deviation is

computed for the flattened array by default, otherwise over the

specified axis.

For all-NaN slices or slices with zero degrees of freedom, NaN is

returned and a `RuntimeWarning` is raised.

.. versionadded:: 1.8.0

Parameters

----------

a : array\_like

Calculate the standard deviation of the non-NaN values.

axis : {int, tuple of int, None}, optional

Axis or axes along which the standard deviation is computed. The default is

to compute the standard deviation of the flattened array.

dtype : dtype, optional

Type to use in computing the standard deviation. For arrays of

integer type the default is float64, for arrays of float types it

is the same as the array type.

out : ndarray, optional

Alternative output array in which to place the result. It must have

the same shape as the expected output but the type (of the

calculated values) will be cast if necessary.

ddof : int, optional

Means Delta Degrees of Freedom. The divisor used in calculations

is ``N - ddof``, where ``N`` represents the number of non-NaN

elements. By default `ddof` is zero.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If this value is anything but the default it is passed through

as-is to the relevant functions of the sub-classes. If these

functions do not have a `keepdims` kwarg, a RuntimeError will

be raised.

Returns

-------

standard\_deviation : ndarray, see dtype parameter above.

If `out` is None, return a new array containing the standard

deviation, otherwise return a reference to the output array. If

ddof is >= the number of non-NaN elements in a slice or the slice

contains only NaNs, then the result for that slice is NaN.

See Also

--------

var, mean, std

nanvar, nanmean

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

The standard deviation is the square root of the average of the squared

deviations from the mean: ``std = sqrt(mean(abs(x - x.mean())\*\*2))``.

The average squared deviation is normally calculated as

``x.sum() / N``, where ``N = len(x)``. If, however, `ddof` is

specified, the divisor ``N - ddof`` is used instead. In standard

statistical practice, ``ddof=1`` provides an unbiased estimator of the

variance of the infinite population. ``ddof=0`` provides a maximum

likelihood estimate of the variance for normally distributed variables.

The standard deviation computed in this function is the square root of

the estimated variance, so even with ``ddof=1``, it will not be an

unbiased estimate of the standard deviation per se.

Note that, for complex numbers, `std` takes the absolute value before

squaring, so that the result is always real and nonnegative.

For floating-point input, the \*std\* is computed using the same

precision the input has. Depending on the input data, this can cause

the results to be inaccurate, especially for float32 (see example

below). Specifying a higher-accuracy accumulator using the `dtype`

keyword can alleviate this issue.

Examples

--------

>>> a = np.array([[1, np.nan], [3, 4]])

>>> np.nanstd(a)

1.247219128924647

>>> np.nanstd(a, axis=0)

array([1., 0.])

>>> np.nanstd(a, axis=1)

array([0., 0.5]) # may vary

nansum(a, axis=None, dtype=None, out=None, keepdims=<no value>)

Return the sum of array elements over a given axis treating Not a

Numbers (NaNs) as zero.

In NumPy versions <= 1.9.0 Nan is returned for slices that are all-NaN or

empty. In later versions zero is returned.

Parameters

----------

a : array\_like

Array containing numbers whose sum is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the sum is computed. The default is to compute the

sum of the flattened array.

dtype : data-type, optional

The type of the returned array and of the accumulator in which the

elements are summed. By default, the dtype of `a` is used. An

exception is when `a` has an integer type with less precision than

the platform (u)intp. In that case, the default will be either

(u)int32 or (u)int64 depending on whether the platform is 32 or 64

bits. For inexact inputs, dtype must be inexact.

.. versionadded:: 1.8.0

out : ndarray, optional

Alternate output array in which to place the result. The default

is ``None``. If provided, it must have the same shape as the

expected output, but the type will be cast if necessary. See

`doc.ufuncs` for details. The casting of NaN to integer can yield

unexpected results.

.. versionadded:: 1.8.0

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

If the value is anything but the default, then

`keepdims` will be passed through to the `mean` or `sum` methods

of sub-classes of `ndarray`. If the sub-classes methods

does not implement `keepdims` any exceptions will be raised.

.. versionadded:: 1.8.0

Returns

-------

nansum : ndarray.

A new array holding the result is returned unless `out` is

specified, in which it is returned. The result has the same

size as `a`, and the same shape as `a` if `axis` is not None

or `a` is a 1-d array.

See Also

--------

numpy.sum : Sum across array propagating NaNs.

isnan : Show which elements are NaN.

isfinite: Show which elements are not NaN or +/-inf.

Notes

-----

If both positive and negative infinity are present, the sum will be Not

A Number (NaN).

Examples

--------

>>> np.nansum(1)

1

>>> np.nansum([1])

1

>>> np.nansum([1, np.nan])

1.0

>>> a = np.array([[1, 1], [1, np.nan]])

>>> np.nansum(a)

3.0

>>> np.nansum(a, axis=0)

array([2., 1.])

>>> np.nansum([1, np.nan, np.inf])

inf

>>> np.nansum([1, np.nan, np.NINF])

-inf

>>> from numpy.testing import suppress\_warnings

>>> with suppress\_warnings() as sup:

... sup.filter(RuntimeWarning)

... np.nansum([1, np.nan, np.inf, -np.inf]) # both +/- infinity present

nan

nanvar(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Compute the variance along the specified axis, while ignoring NaNs.

Returns the variance of the array elements, a measure of the spread of

a distribution. The variance is computed for the flattened array by

default, otherwise over the specified axis.

For all-NaN slices or slices with zero degrees of freedom, NaN is

returned and a `RuntimeWarning` is raised.

.. versionadded:: 1.8.0

Parameters

----------

a : array\_like

Array containing numbers whose variance is desired. If `a` is not an

array, a conversion is attempted.

axis : {int, tuple of int, None}, optional

Axis or axes along which the variance is computed. The default is to compute

the variance of the flattened array.

dtype : data-type, optional

Type to use in computing the variance. For arrays of integer type

the default is `float32`; for arrays of float types it is the same as

the array type.

out : ndarray, optional

Alternate output array in which to place the result. It must have

the same shape as the expected output, but the type is cast if

necessary.

ddof : int, optional

"Delta Degrees of Freedom": the divisor used in the calculation is

``N - ddof``, where ``N`` represents the number of non-NaN

elements. By default `ddof` is zero.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the original `a`.

Returns

-------

variance : ndarray, see dtype parameter above

If `out` is None, return a new array containing the variance,

otherwise return a reference to the output array. If ddof is >= the

number of non-NaN elements in a slice or the slice contains only

NaNs, then the result for that slice is NaN.

See Also

--------

std : Standard deviation

mean : Average

var : Variance while not ignoring NaNs

nanstd, nanmean

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

The variance is the average of the squared deviations from the mean,

i.e., ``var = mean(abs(x - x.mean())\*\*2)``.

The mean is normally calculated as ``x.sum() / N``, where ``N = len(x)``.

If, however, `ddof` is specified, the divisor ``N - ddof`` is used

instead. In standard statistical practice, ``ddof=1`` provides an

unbiased estimator of the variance of a hypothetical infinite

population. ``ddof=0`` provides a maximum likelihood estimate of the

variance for normally distributed variables.

Note that for complex numbers, the absolute value is taken before

squaring, so that the result is always real and nonnegative.

For floating-point input, the variance is computed using the same

precision the input has. Depending on the input data, this can cause

the results to be inaccurate, especially for `float32` (see example

below). Specifying a higher-accuracy accumulator using the ``dtype``

keyword can alleviate this issue.

For this function to work on sub-classes of ndarray, they must define

`sum` with the kwarg `keepdims`

Examples

--------

>>> a = np.array([[1, np.nan], [3, 4]])

>>> np.nanvar(a)

1.5555555555555554

>>> np.nanvar(a, axis=0)

array([1., 0.])

>>> np.nanvar(a, axis=1)

array([0., 0.25]) # may vary

ndfromtxt(fname, \*\*kwargs)

Load ASCII data stored in a file and return it as a single array.

.. deprecated:: 1.17

ndfromtxt` is a deprecated alias of `genfromtxt` which

overwrites the ``usemask`` argument with `False` even when

explicitly called as ``ndfromtxt(..., usemask=True)``.

Use `genfromtxt` instead.

Parameters

----------

fname, kwargs : For a description of input parameters, see `genfromtxt`.

See Also

--------

numpy.genfromtxt : generic function.

ndim(a)

Return the number of dimensions of an array.

Parameters

----------

a : array\_like

Input array. If it is not already an ndarray, a conversion is

attempted.

Returns

-------

number\_of\_dimensions : int

The number of dimensions in `a`. Scalars are zero-dimensional.

See Also

--------

ndarray.ndim : equivalent method

shape : dimensions of array

ndarray.shape : dimensions of array

Examples

--------

>>> np.ndim([[1,2,3],[4,5,6]])

2

>>> np.ndim(np.array([[1,2,3],[4,5,6]]))

2

>>> np.ndim(1)

0

nested\_iters(...)

Create nditers for use in nested loops

Create a tuple of `nditer` objects which iterate in nested loops over

different axes of the op argument. The first iterator is used in the

outermost loop, the last in the innermost loop. Advancing one will change

the subsequent iterators to point at its new element.

Parameters

----------

op : ndarray or sequence of array\_like

The array(s) to iterate over.

axes : list of list of int

Each item is used as an "op\_axes" argument to an nditer

flags, op\_flags, op\_dtypes, order, casting, buffersize (optional)

See `nditer` parameters of the same name

Returns

-------

iters : tuple of nditer

An nditer for each item in `axes`, outermost first

See Also

--------

nditer

Examples

--------

Basic usage. Note how y is the "flattened" version of

[a[:, 0, :], a[:, 1, 0], a[:, 2, :]] since we specified

the first iter's axes as [1]

>>> a = np.arange(12).reshape(2, 3, 2)

>>> i, j = np.nested\_iters(a, [[1], [0, 2]], flags=["multi\_index"])

>>> for x in i:

... print(i.multi\_index)

... for y in j:

... print('', j.multi\_index, y)

(0,)

(0, 0) 0

(0, 1) 1

(1, 0) 6

(1, 1) 7

(1,)

(0, 0) 2

(0, 1) 3

(1, 0) 8

(1, 1) 9

(2,)

(0, 0) 4

(0, 1) 5

(1, 0) 10

(1, 1) 11

nonzero(a)

Return the indices of the elements that are non-zero.

Returns a tuple of arrays, one for each dimension of `a`,

containing the indices of the non-zero elements in that

dimension. The values in `a` are always tested and returned in

row-major, C-style order.

To group the indices by element, rather than dimension, use `argwhere`,

which returns a row for each non-zero element.

.. note::

When called on a zero-d array or scalar, ``nonzero(a)`` is treated

as ``nonzero(atleast1d(a))``.

..deprecated:: 1.17.0

Use `atleast1d` explicitly if this behavior is deliberate.

Parameters

----------

a : array\_like

Input array.

Returns

-------

tuple\_of\_arrays : tuple

Indices of elements that are non-zero.

See Also

--------

flatnonzero :

Return indices that are non-zero in the flattened version of the input

array.

ndarray.nonzero :

Equivalent ndarray method.

count\_nonzero :

Counts the number of non-zero elements in the input array.

Notes

-----

While the nonzero values can be obtained with ``a[nonzero(a)]``, it is

recommended to use ``x[x.astype(bool)]`` or ``x[x != 0]`` instead, which

will correctly handle 0-d arrays.

Examples

--------

>>> x = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])

>>> x

array([[3, 0, 0],

[0, 4, 0],

[5, 6, 0]])

>>> np.nonzero(x)

(array([0, 1, 2, 2]), array([0, 1, 0, 1]))

>>> x[np.nonzero(x)]

array([3, 4, 5, 6])

>>> np.transpose(np.nonzero(x))

array([[0, 0],

[1, 1],

[2, 0],

[2, 1]])

A common use for ``nonzero`` is to find the indices of an array, where

a condition is True. Given an array `a`, the condition `a` > 3 is a

boolean array and since False is interpreted as 0, np.nonzero(a > 3)

yields the indices of the `a` where the condition is true.

>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

>>> a > 3

array([[False, False, False],

[ True, True, True],

[ True, True, True]])

>>> np.nonzero(a > 3)

(array([1, 1, 1, 2, 2, 2]), array([0, 1, 2, 0, 1, 2]))

Using this result to index `a` is equivalent to using the mask directly:

>>> a[np.nonzero(a > 3)]

array([4, 5, 6, 7, 8, 9])

>>> a[a > 3] # prefer this spelling

array([4, 5, 6, 7, 8, 9])

``nonzero`` can also be called as a method of the array.

>>> (a > 3).nonzero()

(array([1, 1, 1, 2, 2, 2]), array([0, 1, 2, 0, 1, 2]))

nper(rate, pmt, pv, fv=0, when='end')

Compute the number of periodic payments.

:class:`decimal.Decimal` type is not supported.

Parameters

----------

rate : array\_like

Rate of interest (per period)

pmt : array\_like

Payment

pv : array\_like

Present value

fv : array\_like, optional

Future value

when : {{'begin', 1}, {'end', 0}}, {string, int}, optional

When payments are due ('begin' (1) or 'end' (0))

Notes

-----

The number of periods ``nper`` is computed by solving the equation::

fv + pv\*(1+rate)\*\*nper + pmt\*(1+rate\*when)/rate\*((1+rate)\*\*nper-1) = 0

but if ``rate = 0`` then::

fv + pv + pmt\*nper = 0

Examples

--------

If you only had $150/month to pay towards the loan, how long would it take

to pay-off a loan of $8,000 at 7% annual interest?

>>> print(np.round(np.nper(0.07/12, -150, 8000), 5))

64.07335

So, over 64 months would be required to pay off the loan.

The same analysis could be done with several different interest rates

and/or payments and/or total amounts to produce an entire table.

>>> np.nper(\*(np.ogrid[0.07/12: 0.08/12: 0.01/12,

... -150 : -99 : 50 ,

... 8000 : 9001 : 1000]))

array([[[ 64.07334877, 74.06368256],

[108.07548412, 127.99022654]],

[[ 66.12443902, 76.87897353],

[114.70165583, 137.90124779]]])

npv(rate, values)

Returns the NPV (Net Present Value) of a cash flow series.

Parameters

----------

rate : scalar

The discount rate.

values : array\_like, shape(M, )

The values of the time series of cash flows. The (fixed) time

interval between cash flow "events" must be the same as that for

which `rate` is given (i.e., if `rate` is per year, then precisely

a year is understood to elapse between each cash flow event). By

convention, investments or "deposits" are negative, income or

"withdrawals" are positive; `values` must begin with the initial

investment, thus `values[0]` will typically be negative.

Returns

-------

out : float

The NPV of the input cash flow series `values` at the discount

`rate`.

Notes

-----

Returns the result of: [G]\_

.. math :: \sum\_{t=0}^{M-1}{\frac{values\_t}{(1+rate)^{t}}}

References

----------

.. [G] L. J. Gitman, "Principles of Managerial Finance, Brief," 3rd ed.,

Addison-Wesley, 2003, pg. 346.

Examples

--------

>>> np.npv(0.281,[-100, 39, 59, 55, 20])

-0.0084785916384548798 # may vary

(Compare with the Example given for numpy.lib.financial.irr)

obj2sctype(rep, default=None)

Return the scalar dtype or NumPy equivalent of Python type of an object.

Parameters

----------

rep : any

The object of which the type is returned.

default : any, optional

If given, this is returned for objects whose types can not be

determined. If not given, None is returned for those objects.

Returns

-------

dtype : dtype or Python type

The data type of `rep`.

See Also

--------

sctype2char, issctype, issubsctype, issubdtype, maximum\_sctype

Examples

--------

>>> np.obj2sctype(np.int32)

<class 'numpy.int32'>

>>> np.obj2sctype(np.array([1., 2.]))

<class 'numpy.float64'>

>>> np.obj2sctype(np.array([1.j]))

<class 'numpy.complex128'>

>>> np.obj2sctype(dict)

<class 'numpy.object\_'>

>>> np.obj2sctype('string')

>>> np.obj2sctype(1, default=list)

<class 'list'>

ones(shape, dtype=None, order='C')

Return a new array of given shape and type, filled with ones.

Parameters

----------

shape : int or sequence of ints

Shape of the new array, e.g., ``(2, 3)`` or ``2``.

dtype : data-type, optional

The desired data-type for the array, e.g., `numpy.int8`. Default is

`numpy.float64`.

order : {'C', 'F'}, optional, default: C

Whether to store multi-dimensional data in row-major

(C-style) or column-major (Fortran-style) order in

memory.

Returns

-------

out : ndarray

Array of ones with the given shape, dtype, and order.

See Also

--------

ones\_like : Return an array of ones with shape and type of input.

empty : Return a new uninitialized array.

zeros : Return a new array setting values to zero.

full : Return a new array of given shape filled with value.

Examples

--------

>>> np.ones(5)

array([1., 1., 1., 1., 1.])

>>> np.ones((5,), dtype=int)

array([1, 1, 1, 1, 1])

>>> np.ones((2, 1))

array([[1.],

[1.]])

>>> s = (2,2)

>>> np.ones(s)

array([[1., 1.],

[1., 1.]])

ones\_like(a, dtype=None, order='K', subok=True, shape=None)

Return an array of ones with the same shape and type as a given array.

Parameters

----------

a : array\_like

The shape and data-type of `a` define these same attributes of

the returned array.

dtype : data-type, optional

Overrides the data type of the result.

.. versionadded:: 1.6.0

order : {'C', 'F', 'A', or 'K'}, optional

Overrides the memory layout of the result. 'C' means C-order,

'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

'C' otherwise. 'K' means match the layout of `a` as closely

as possible.

.. versionadded:: 1.6.0

subok : bool, optional.

If True, then the newly created array will use the sub-class

type of 'a', otherwise it will be a base-class array. Defaults

to True.

shape : int or sequence of ints, optional.

Overrides the shape of the result. If order='K' and the number of

dimensions is unchanged, will try to keep order, otherwise,

order='C' is implied.

.. versionadded:: 1.17.0

Returns

-------

out : ndarray

Array of ones with the same shape and type as `a`.

See Also

--------

empty\_like : Return an empty array with shape and type of input.

zeros\_like : Return an array of zeros with shape and type of input.

full\_like : Return a new array with shape of input filled with value.

ones : Return a new array setting values to one.

Examples

--------

>>> x = np.arange(6)

>>> x = x.reshape((2, 3))

>>> x

array([[0, 1, 2],

[3, 4, 5]])

>>> np.ones\_like(x)

array([[1, 1, 1],

[1, 1, 1]])

>>> y = np.arange(3, dtype=float)

>>> y

array([0., 1., 2.])

>>> np.ones\_like(y)

array([1., 1., 1.])

outer(a, b, out=None)

Compute the outer product of two vectors.

Given two vectors, ``a = [a0, a1, ..., aM]`` and

``b = [b0, b1, ..., bN]``,

the outer product [1]\_ is::

[[a0\*b0 a0\*b1 ... a0\*bN ]

[a1\*b0 .

[ ... .

[aM\*b0 aM\*bN ]]

Parameters

----------

a : (M,) array\_like

First input vector. Input is flattened if

not already 1-dimensional.

b : (N,) array\_like

Second input vector. Input is flattened if

not already 1-dimensional.

out : (M, N) ndarray, optional

A location where the result is stored

.. versionadded:: 1.9.0

Returns

-------

out : (M, N) ndarray

``out[i, j] = a[i] \* b[j]``

See also

--------

inner

einsum : ``einsum('i,j->ij', a.ravel(), b.ravel())`` is the equivalent.

ufunc.outer : A generalization to N dimensions and other operations.

``np.multiply.outer(a.ravel(), b.ravel())`` is the equivalent.

References

----------

.. [1] : G. H. Golub and C. F. Van Loan, \*Matrix Computations\*, 3rd

ed., Baltimore, MD, Johns Hopkins University Press, 1996,

pg. 8.

Examples

--------

Make a (\*very\* coarse) grid for computing a Mandelbrot set:

>>> rl = np.outer(np.ones((5,)), np.linspace(-2, 2, 5))

>>> rl

array([[-2., -1., 0., 1., 2.],

[-2., -1., 0., 1., 2.],

[-2., -1., 0., 1., 2.],

[-2., -1., 0., 1., 2.],

[-2., -1., 0., 1., 2.]])

>>> im = np.outer(1j\*np.linspace(2, -2, 5), np.ones((5,)))

>>> im

array([[0.+2.j, 0.+2.j, 0.+2.j, 0.+2.j, 0.+2.j],

[0.+1.j, 0.+1.j, 0.+1.j, 0.+1.j, 0.+1.j],

[0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],

[0.-1.j, 0.-1.j, 0.-1.j, 0.-1.j, 0.-1.j],

[0.-2.j, 0.-2.j, 0.-2.j, 0.-2.j, 0.-2.j]])

>>> grid = rl + im

>>> grid

array([[-2.+2.j, -1.+2.j, 0.+2.j, 1.+2.j, 2.+2.j],

[-2.+1.j, -1.+1.j, 0.+1.j, 1.+1.j, 2.+1.j],

[-2.+0.j, -1.+0.j, 0.+0.j, 1.+0.j, 2.+0.j],

[-2.-1.j, -1.-1.j, 0.-1.j, 1.-1.j, 2.-1.j],

[-2.-2.j, -1.-2.j, 0.-2.j, 1.-2.j, 2.-2.j]])

An example using a "vector" of letters:

>>> x = np.array(['a', 'b', 'c'], dtype=object)

>>> np.outer(x, [1, 2, 3])

array([['a', 'aa', 'aaa'],

['b', 'bb', 'bbb'],

['c', 'cc', 'ccc']], dtype=object)

packbits(...)

packbits(a, axis=None, bitorder='big')

Packs the elements of a binary-valued array into bits in a uint8 array.

The result is padded to full bytes by inserting zero bits at the end.

Parameters

----------

a : array\_like

An array of integers or booleans whose elements should be packed to

bits.

axis : int, optional

The dimension over which bit-packing is done.

``None`` implies packing the flattened array.

bitorder : {'big', 'little'}, optional

The order of the input bits. 'big' will mimic bin(val),

``[0, 0, 0, 0, 0, 0, 1, 1] => 3 = 0b00000011 => ``, 'little' will

reverse the order so ``[1, 1, 0, 0, 0, 0, 0, 0] => 3``.

Defaults to 'big'.

.. versionadded:: 1.17.0

Returns

-------

packed : ndarray

Array of type uint8 whose elements represent bits corresponding to the

logical (0 or nonzero) value of the input elements. The shape of

`packed` has the same number of dimensions as the input (unless `axis`

is None, in which case the output is 1-D).

See Also

--------

unpackbits: Unpacks elements of a uint8 array into a binary-valued output

array.

Examples

--------

>>> a = np.array([[[1,0,1],

... [0,1,0]],

... [[1,1,0],

... [0,0,1]]])

>>> b = np.packbits(a, axis=-1)

>>> b

array([[[160],

[ 64]],

[[192],

[ 32]]], dtype=uint8)

Note that in binary 160 = 1010 0000, 64 = 0100 0000, 192 = 1100 0000,

and 32 = 0010 0000.

pad(array, pad\_width, mode='constant', \*\*kwargs)

Pad an array.

Parameters

----------

array : array\_like of rank N

The array to pad.

pad\_width : {sequence, array\_like, int}

Number of values padded to the edges of each axis.

((before\_1, after\_1), ... (before\_N, after\_N)) unique pad widths

for each axis.

((before, after),) yields same before and after pad for each axis.

(pad,) or int is a shortcut for before = after = pad width for all

axes.

mode : str or function, optional

One of the following string values or a user supplied function.

'constant' (default)

Pads with a constant value.

'edge'

Pads with the edge values of array.

'linear\_ramp'

Pads with the linear ramp between end\_value and the

array edge value.

'maximum'

Pads with the maximum value of all or part of the

vector along each axis.

'mean'

Pads with the mean value of all or part of the

vector along each axis.

'median'

Pads with the median value of all or part of the

vector along each axis.

'minimum'

Pads with the minimum value of all or part of the

vector along each axis.

'reflect'

Pads with the reflection of the vector mirrored on

the first and last values of the vector along each

axis.

'symmetric'

Pads with the reflection of the vector mirrored

along the edge of the array.

'wrap'

Pads with the wrap of the vector along the axis.

The first values are used to pad the end and the

end values are used to pad the beginning.

'empty'

Pads with undefined values.

.. versionadded:: 1.17

<function>

Padding function, see Notes.

stat\_length : sequence or int, optional

Used in 'maximum', 'mean', 'median', and 'minimum'. Number of

values at edge of each axis used to calculate the statistic value.

((before\_1, after\_1), ... (before\_N, after\_N)) unique statistic

lengths for each axis.

((before, after),) yields same before and after statistic lengths

for each axis.

(stat\_length,) or int is a shortcut for before = after = statistic

length for all axes.

Default is ``None``, to use the entire axis.

constant\_values : sequence or scalar, optional

Used in 'constant'. The values to set the padded values for each

axis.

``((before\_1, after\_1), ... (before\_N, after\_N))`` unique pad constants

for each axis.

``((before, after),)`` yields same before and after constants for each

axis.

``(constant,)`` or ``constant`` is a shortcut for ``before = after = constant`` for

all axes.

Default is 0.

end\_values : sequence or scalar, optional

Used in 'linear\_ramp'. The values used for the ending value of the

linear\_ramp and that will form the edge of the padded array.

``((before\_1, after\_1), ... (before\_N, after\_N))`` unique end values

for each axis.

``((before, after),)`` yields same before and after end values for each

axis.

``(constant,)`` or ``constant`` is a shortcut for ``before = after = constant`` for

all axes.

Default is 0.

reflect\_type : {'even', 'odd'}, optional

Used in 'reflect', and 'symmetric'. The 'even' style is the

default with an unaltered reflection around the edge value. For

the 'odd' style, the extended part of the array is created by

subtracting the reflected values from two times the edge value.

Returns

-------

pad : ndarray

Padded array of rank equal to `array` with shape increased

according to `pad\_width`.

Notes

-----

.. versionadded:: 1.7.0

For an array with rank greater than 1, some of the padding of later

axes is calculated from padding of previous axes. This is easiest to

think about with a rank 2 array where the corners of the padded array

are calculated by using padded values from the first axis.

The padding function, if used, should modify a rank 1 array in-place. It

has the following signature::

padding\_func(vector, iaxis\_pad\_width, iaxis, kwargs)

where

vector : ndarray

A rank 1 array already padded with zeros. Padded values are

vector[:iaxis\_pad\_width[0]] and vector[-iaxis\_pad\_width[1]:].

iaxis\_pad\_width : tuple

A 2-tuple of ints, iaxis\_pad\_width[0] represents the number of

values padded at the beginning of vector where

iaxis\_pad\_width[1] represents the number of values padded at

the end of vector.

iaxis : int

The axis currently being calculated.

kwargs : dict

Any keyword arguments the function requires.

Examples

--------

>>> a = [1, 2, 3, 4, 5]

>>> np.pad(a, (2, 3), 'constant', constant\_values=(4, 6))

array([4, 4, 1, ..., 6, 6, 6])

>>> np.pad(a, (2, 3), 'edge')

array([1, 1, 1, ..., 5, 5, 5])

>>> np.pad(a, (2, 3), 'linear\_ramp', end\_values=(5, -4))

array([ 5, 3, 1, 2, 3, 4, 5, 2, -1, -4])

>>> np.pad(a, (2,), 'maximum')

array([5, 5, 1, 2, 3, 4, 5, 5, 5])

>>> np.pad(a, (2,), 'mean')

array([3, 3, 1, 2, 3, 4, 5, 3, 3])

>>> np.pad(a, (2,), 'median')

array([3, 3, 1, 2, 3, 4, 5, 3, 3])

>>> a = [[1, 2], [3, 4]]

>>> np.pad(a, ((3, 2), (2, 3)), 'minimum')

array([[1, 1, 1, 2, 1, 1, 1],

[1, 1, 1, 2, 1, 1, 1],

[1, 1, 1, 2, 1, 1, 1],

[1, 1, 1, 2, 1, 1, 1],

[3, 3, 3, 4, 3, 3, 3],

[1, 1, 1, 2, 1, 1, 1],

[1, 1, 1, 2, 1, 1, 1]])

>>> a = [1, 2, 3, 4, 5]

>>> np.pad(a, (2, 3), 'reflect')

array([3, 2, 1, 2, 3, 4, 5, 4, 3, 2])

>>> np.pad(a, (2, 3), 'reflect', reflect\_type='odd')

array([-1, 0, 1, 2, 3, 4, 5, 6, 7, 8])

>>> np.pad(a, (2, 3), 'symmetric')

array([2, 1, 1, 2, 3, 4, 5, 5, 4, 3])

>>> np.pad(a, (2, 3), 'symmetric', reflect\_type='odd')

array([0, 1, 1, 2, 3, 4, 5, 5, 6, 7])

>>> np.pad(a, (2, 3), 'wrap')

array([4, 5, 1, 2, 3, 4, 5, 1, 2, 3])

>>> def pad\_with(vector, pad\_width, iaxis, kwargs):

... pad\_value = kwargs.get('padder', 10)

... vector[:pad\_width[0]] = pad\_value

... vector[-pad\_width[1]:] = pad\_value

>>> a = np.arange(6)

>>> a = a.reshape((2, 3))

>>> np.pad(a, 2, pad\_with)

array([[10, 10, 10, 10, 10, 10, 10],

[10, 10, 10, 10, 10, 10, 10],

[10, 10, 0, 1, 2, 10, 10],

[10, 10, 3, 4, 5, 10, 10],

[10, 10, 10, 10, 10, 10, 10],

[10, 10, 10, 10, 10, 10, 10]])

>>> np.pad(a, 2, pad\_with, padder=100)

array([[100, 100, 100, 100, 100, 100, 100],

[100, 100, 100, 100, 100, 100, 100],

[100, 100, 0, 1, 2, 100, 100],

[100, 100, 3, 4, 5, 100, 100],

[100, 100, 100, 100, 100, 100, 100],

[100, 100, 100, 100, 100, 100, 100]])

partition(a, kth, axis=-1, kind='introselect', order=None)

Return a partitioned copy of an array.

Creates a copy of the array with its elements rearranged in such a

way that the value of the element in k-th position is in the

position it would be in a sorted array. All elements smaller than

the k-th element are moved before this element and all equal or

greater are moved behind it. The ordering of the elements in the two

partitions is undefined.

.. versionadded:: 1.8.0

Parameters

----------

a : array\_like

Array to be sorted.

kth : int or sequence of ints

Element index to partition by. The k-th value of the element

will be in its final sorted position and all smaller elements

will be moved before it and all equal or greater elements behind

it. The order of all elements in the partitions is undefined. If

provided with a sequence of k-th it will partition all elements

indexed by k-th of them into their sorted position at once.

axis : int or None, optional

Axis along which to sort. If None, the array is flattened before

sorting. The default is -1, which sorts along the last axis.

kind : {'introselect'}, optional

Selection algorithm. Default is 'introselect'.

order : str or list of str, optional

When `a` is an array with fields defined, this argument

specifies which fields to compare first, second, etc. A single

field can be specified as a string. Not all fields need be

specified, but unspecified fields will still be used, in the

order in which they come up in the dtype, to break ties.

Returns

-------

partitioned\_array : ndarray

Array of the same type and shape as `a`.

See Also

--------

ndarray.partition : Method to sort an array in-place.

argpartition : Indirect partition.

sort : Full sorting

Notes

-----

The various selection algorithms are characterized by their average

speed, worst case performance, work space size, and whether they are

stable. A stable sort keeps items with the same key in the same

relative order. The available algorithms have the following

properties:

================= ======= ============= ============ =======

kind speed worst case work space stable

================= ======= ============= ============ =======

'introselect' 1 O(n) 0 no

================= ======= ============= ============ =======

All the partition algorithms make temporary copies of the data when

partitioning along any but the last axis. Consequently,

partitioning along the last axis is faster and uses less space than

partitioning along any other axis.

The sort order for complex numbers is lexicographic. If both the

real and imaginary parts are non-nan then the order is determined by

the real parts except when they are equal, in which case the order

is determined by the imaginary parts.

Examples

--------

>>> a = np.array([3, 4, 2, 1])

>>> np.partition(a, 3)

array([2, 1, 3, 4])

>>> np.partition(a, (1, 3))

array([1, 2, 3, 4])

percentile(a, q, axis=None, out=None, overwrite\_input=False, interpolation='linear', keepdims=False)

Compute the q-th percentile of the data along the specified axis.

Returns the q-th percentile(s) of the array elements.

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

q : array\_like of float

Percentile or sequence of percentiles to compute, which must be between

0 and 100 inclusive.

axis : {int, tuple of int, None}, optional

Axis or axes along which the percentiles are computed. The

default is to compute the percentile(s) along a flattened

version of the array.

.. versionchanged:: 1.9.0

A tuple of axes is supported

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow the input array `a` to be modified by intermediate

calculations, to save memory. In this case, the contents of the input

`a` after this function completes is undefined.

interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}

This optional parameter specifies the interpolation method to

use when the desired percentile lies between two data points

``i < j``:

\* 'linear': ``i + (j - i) \* fraction``, where ``fraction``

is the fractional part of the index surrounded by ``i``

and ``j``.

\* 'lower': ``i``.

\* 'higher': ``j``.

\* 'nearest': ``i`` or ``j``, whichever is nearest.

\* 'midpoint': ``(i + j) / 2``.

.. versionadded:: 1.9.0

keepdims : bool, optional

If this is set to True, the axes which are reduced are left in

the result as dimensions with size one. With this option, the

result will broadcast correctly against the original array `a`.

.. versionadded:: 1.9.0

Returns

-------

percentile : scalar or ndarray

If `q` is a single percentile and `axis=None`, then the result

is a scalar. If multiple percentiles are given, first axis of

the result corresponds to the percentiles. The other axes are

the axes that remain after the reduction of `a`. If the input

contains integers or floats smaller than ``float64``, the output

data-type is ``float64``. Otherwise, the output data-type is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

mean

median : equivalent to ``percentile(..., 50)``

nanpercentile

quantile : equivalent to percentile, except with q in the range [0, 1].

Notes

-----

Given a vector ``V`` of length ``N``, the q-th percentile of

``V`` is the value ``q/100`` of the way from the minimum to the

maximum in a sorted copy of ``V``. The values and distances of

the two nearest neighbors as well as the `interpolation` parameter

will determine the percentile if the normalized ranking does not

match the location of ``q`` exactly. This function is the same as

the median if ``q=50``, the same as the minimum if ``q=0`` and the

same as the maximum if ``q=100``.

Examples

--------

>>> a = np.array([[10, 7, 4], [3, 2, 1]])

>>> a

array([[10, 7, 4],

[ 3, 2, 1]])

>>> np.percentile(a, 50)

3.5

>>> np.percentile(a, 50, axis=0)

array([6.5, 4.5, 2.5])

>>> np.percentile(a, 50, axis=1)

array([7., 2.])

>>> np.percentile(a, 50, axis=1, keepdims=True)

array([[7.],

[2.]])

>>> m = np.percentile(a, 50, axis=0)

>>> out = np.zeros\_like(m)

>>> np.percentile(a, 50, axis=0, out=out)

array([6.5, 4.5, 2.5])

>>> m

array([6.5, 4.5, 2.5])

>>> b = a.copy()

>>> np.percentile(b, 50, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a == b)

The different types of interpolation can be visualized graphically:

.. plot::

import matplotlib.pyplot as plt

a = np.arange(4)

p = np.linspace(0, 100, 6001)

ax = plt.gca()

lines = [

('linear', None),

('higher', '--'),

('lower', '--'),

('nearest', '-.'),

('midpoint', '-.'),

]

for interpolation, style in lines:

ax.plot(

p, np.percentile(a, p, interpolation=interpolation),

label=interpolation, linestyle=style)

ax.set(

title='Interpolation methods for list: ' + str(a),

xlabel='Percentile',

ylabel='List item returned',

yticks=a)

ax.legend()

plt.show()

piecewise(x, condlist, funclist, \*args, \*\*kw)

Evaluate a piecewise-defined function.

Given a set of conditions and corresponding functions, evaluate each

function on the input data wherever its condition is true.

Parameters

----------

x : ndarray or scalar

The input domain.

condlist : list of bool arrays or bool scalars

Each boolean array corresponds to a function in `funclist`. Wherever

`condlist[i]` is True, `funclist[i](x)` is used as the output value.

Each boolean array in `condlist` selects a piece of `x`,

and should therefore be of the same shape as `x`.

The length of `condlist` must correspond to that of `funclist`.

If one extra function is given, i.e. if

``len(funclist) == len(condlist) + 1``, then that extra function

is the default value, used wherever all conditions are false.

funclist : list of callables, f(x,\*args,\*\*kw), or scalars

Each function is evaluated over `x` wherever its corresponding

condition is True. It should take a 1d array as input and give an 1d

array or a scalar value as output. If, instead of a callable,

a scalar is provided then a constant function (``lambda x: scalar``) is

assumed.

args : tuple, optional

Any further arguments given to `piecewise` are passed to the functions

upon execution, i.e., if called ``piecewise(..., ..., 1, 'a')``, then

each function is called as ``f(x, 1, 'a')``.

kw : dict, optional

Keyword arguments used in calling `piecewise` are passed to the

functions upon execution, i.e., if called

``piecewise(..., ..., alpha=1)``, then each function is called as

``f(x, alpha=1)``.

Returns

-------

out : ndarray

The output is the same shape and type as x and is found by

calling the functions in `funclist` on the appropriate portions of `x`,

as defined by the boolean arrays in `condlist`. Portions not covered

by any condition have a default value of 0.

See Also

--------

choose, select, where

Notes

-----

This is similar to choose or select, except that functions are

evaluated on elements of `x` that satisfy the corresponding condition from

`condlist`.

The result is::

|--

|funclist[0](x[condlist[0]])

out = |funclist[1](x[condlist[1]])

|...

|funclist[n2](x[condlist[n2]])

|--

Examples

--------

Define the sigma function, which is -1 for ``x < 0`` and +1 for ``x >= 0``.

>>> x = np.linspace(-2.5, 2.5, 6)

>>> np.piecewise(x, [x < 0, x >= 0], [-1, 1])

array([-1., -1., -1., 1., 1., 1.])

Define the absolute value, which is ``-x`` for ``x <0`` and ``x`` for

``x >= 0``.

>>> np.piecewise(x, [x < 0, x >= 0], [lambda x: -x, lambda x: x])

array([2.5, 1.5, 0.5, 0.5, 1.5, 2.5])

Apply the same function to a scalar value.

>>> y = -2

>>> np.piecewise(y, [y < 0, y >= 0], [lambda x: -x, lambda x: x])

array(2)

place(arr, mask, vals)

Change elements of an array based on conditional and input values.

Similar to ``np.copyto(arr, vals, where=mask)``, the difference is that

`place` uses the first N elements of `vals`, where N is the number of

True values in `mask`, while `copyto` uses the elements where `mask`

is True.

Note that `extract` does the exact opposite of `place`.

Parameters

----------

arr : ndarray

Array to put data into.

mask : array\_like

Boolean mask array. Must have the same size as `a`.

vals : 1-D sequence

Values to put into `a`. Only the first N elements are used, where

N is the number of True values in `mask`. If `vals` is smaller

than N, it will be repeated, and if elements of `a` are to be masked,

this sequence must be non-empty.

See Also

--------

copyto, put, take, extract

Examples

--------

>>> arr = np.arange(6).reshape(2, 3)

>>> np.place(arr, arr>2, [44, 55])

>>> arr

array([[ 0, 1, 2],

[44, 55, 44]])

pmt(rate, nper, pv, fv=0, when='end')

Compute the payment against loan principal plus interest.

Given:

\* a present value, `pv` (e.g., an amount borrowed)

\* a future value, `fv` (e.g., 0)

\* an interest `rate` compounded once per period, of which

there are

\* `nper` total

\* and (optional) specification of whether payment is made

at the beginning (`when` = {'begin', 1}) or the end

(`when` = {'end', 0}) of each period

Return:

the (fixed) periodic payment.

Parameters

----------

rate : array\_like

Rate of interest (per period)

nper : array\_like

Number of compounding periods

pv : array\_like

Present value

fv : array\_like, optional

Future value (default = 0)

when : {{'begin', 1}, {'end', 0}}, {string, int}

When payments are due ('begin' (1) or 'end' (0))

Returns

-------

out : ndarray

Payment against loan plus interest. If all input is scalar, returns a

scalar float. If any input is array\_like, returns payment for each

input element. If multiple inputs are array\_like, they all must have

the same shape.

Notes

-----

The payment is computed by solving the equation::

fv +

pv\*(1 + rate)\*\*nper +

pmt\*(1 + rate\*when)/rate\*((1 + rate)\*\*nper - 1) == 0

or, when ``rate == 0``::

fv + pv + pmt \* nper == 0

for ``pmt``.

Note that computing a monthly mortgage payment is only

one use for this function. For example, pmt returns the

periodic deposit one must make to achieve a specified

future balance given an initial deposit, a fixed,

periodically compounded interest rate, and the total

number of periods.

References

----------

.. [WRW] Wheeler, D. A., E. Rathke, and R. Weir (Eds.) (2009, May).

Open Document Format for Office Applications (OpenDocument)v1.2,

Part 2: Recalculated Formula (OpenFormula) Format - Annotated Version,

Pre-Draft 12. Organization for the Advancement of Structured Information

Standards (OASIS). Billerica, MA, USA. [ODT Document].

Available:

http://www.oasis-open.org/committees/documents.php

?wg\_abbrev=office-formulaOpenDocument-formula-20090508.odt

Examples

--------

What is the monthly payment needed to pay off a $200,000 loan in 15

years at an annual interest rate of 7.5%?

>>> np.pmt(0.075/12, 12\*15, 200000)

-1854.0247200054619

In order to pay-off (i.e., have a future-value of 0) the $200,000 obtained

today, a monthly payment of $1,854.02 would be required. Note that this

example illustrates usage of `fv` having a default value of 0.

poly(seq\_of\_zeros)

Find the coefficients of a polynomial with the given sequence of roots.

Returns the coefficients of the polynomial whose leading coefficient

is one for the given sequence of zeros (multiple roots must be included

in the sequence as many times as their multiplicity; see Examples).

A square matrix (or array, which will be treated as a matrix) can also

be given, in which case the coefficients of the characteristic polynomial

of the matrix are returned.

Parameters

----------

seq\_of\_zeros : array\_like, shape (N,) or (N, N)

A sequence of polynomial roots, or a square array or matrix object.

Returns

-------

c : ndarray

1D array of polynomial coefficients from highest to lowest degree:

``c[0] \* x\*\*(N) + c[1] \* x\*\*(N-1) + ... + c[N-1] \* x + c[N]``

where c[0] always equals 1.

Raises

------

ValueError

If input is the wrong shape (the input must be a 1-D or square

2-D array).

See Also

--------

polyval : Compute polynomial values.

roots : Return the roots of a polynomial.

polyfit : Least squares polynomial fit.

poly1d : A one-dimensional polynomial class.

Notes

-----

Specifying the roots of a polynomial still leaves one degree of

freedom, typically represented by an undetermined leading

coefficient. [1]\_ In the case of this function, that coefficient -

the first one in the returned array - is always taken as one. (If

for some reason you have one other point, the only automatic way

presently to leverage that information is to use ``polyfit``.)

The characteristic polynomial, :math:`p\_a(t)`, of an `n`-by-`n`

matrix \*\*A\*\* is given by

:math:`p\_a(t) = \mathrm{det}(t\, \mathbf{I} - \mathbf{A})`,

where \*\*I\*\* is the `n`-by-`n` identity matrix. [2]\_

References

----------

.. [1] M. Sullivan and M. Sullivan, III, "Algebra and Trignometry,

Enhanced With Graphing Utilities," Prentice-Hall, pg. 318, 1996.

.. [2] G. Strang, "Linear Algebra and Its Applications, 2nd Edition,"

Academic Press, pg. 182, 1980.

Examples

--------

Given a sequence of a polynomial's zeros:

>>> np.poly((0, 0, 0)) # Multiple root example

array([1., 0., 0., 0.])

The line above represents z\*\*3 + 0\*z\*\*2 + 0\*z + 0.

>>> np.poly((-1./2, 0, 1./2))

array([ 1. , 0. , -0.25, 0. ])

The line above represents z\*\*3 - z/4

>>> np.poly((np.random.random(1)[0], 0, np.random.random(1)[0]))

array([ 1. , -0.77086955, 0.08618131, 0. ]) # random

Given a square array object:

>>> P = np.array([[0, 1./3], [-1./2, 0]])

>>> np.poly(P)

array([1. , 0. , 0.16666667])

Note how in all cases the leading coefficient is always 1.

polyadd(a1, a2)

Find the sum of two polynomials.

Returns the polynomial resulting from the sum of two input polynomials.

Each input must be either a poly1d object or a 1D sequence of polynomial

coefficients, from highest to lowest degree.

Parameters

----------

a1, a2 : array\_like or poly1d object

Input polynomials.

Returns

-------

out : ndarray or poly1d object

The sum of the inputs. If either input is a poly1d object, then the

output is also a poly1d object. Otherwise, it is a 1D array of

polynomial coefficients from highest to lowest degree.

See Also

--------

poly1d : A one-dimensional polynomial class.

poly, polyadd, polyder, polydiv, polyfit, polyint, polysub, polyval

Examples

--------

>>> np.polyadd([1, 2], [9, 5, 4])

array([9, 6, 6])

Using poly1d objects:

>>> p1 = np.poly1d([1, 2])

>>> p2 = np.poly1d([9, 5, 4])

>>> print(p1)

1 x + 2

>>> print(p2)

2

9 x + 5 x + 4

>>> print(np.polyadd(p1, p2))

2

9 x + 6 x + 6

polyder(p, m=1)

Return the derivative of the specified order of a polynomial.

Parameters

----------

p : poly1d or sequence

Polynomial to differentiate.

A sequence is interpreted as polynomial coefficients, see `poly1d`.

m : int, optional

Order of differentiation (default: 1)

Returns

-------

der : poly1d

A new polynomial representing the derivative.

See Also

--------

polyint : Anti-derivative of a polynomial.

poly1d : Class for one-dimensional polynomials.

Examples

--------

The derivative of the polynomial :math:`x^3 + x^2 + x^1 + 1` is:

>>> p = np.poly1d([1,1,1,1])

>>> p2 = np.polyder(p)

>>> p2

poly1d([3, 2, 1])

which evaluates to:

>>> p2(2.)

17.0

We can verify this, approximating the derivative with

``(f(x + h) - f(x))/h``:

>>> (p(2. + 0.001) - p(2.)) / 0.001

17.007000999997857

The fourth-order derivative of a 3rd-order polynomial is zero:

>>> np.polyder(p, 2)

poly1d([6, 2])

>>> np.polyder(p, 3)

poly1d([6])

>>> np.polyder(p, 4)

poly1d([0.])

polydiv(u, v)

Returns the quotient and remainder of polynomial division.

The input arrays are the coefficients (including any coefficients

equal to zero) of the "numerator" (dividend) and "denominator"

(divisor) polynomials, respectively.

Parameters

----------

u : array\_like or poly1d

Dividend polynomial's coefficients.

v : array\_like or poly1d

Divisor polynomial's coefficients.

Returns

-------

q : ndarray

Coefficients, including those equal to zero, of the quotient.

r : ndarray

Coefficients, including those equal to zero, of the remainder.

See Also

--------

poly, polyadd, polyder, polydiv, polyfit, polyint, polymul, polysub

polyval

Notes

-----

Both `u` and `v` must be 0-d or 1-d (ndim = 0 or 1), but `u.ndim` need

not equal `v.ndim`. In other words, all four possible combinations -

``u.ndim = v.ndim = 0``, ``u.ndim = v.ndim = 1``,

``u.ndim = 1, v.ndim = 0``, and ``u.ndim = 0, v.ndim = 1`` - work.

Examples

--------

.. math:: \frac{3x^2 + 5x + 2}{2x + 1} = 1.5x + 1.75, remainder 0.25

>>> x = np.array([3.0, 5.0, 2.0])

>>> y = np.array([2.0, 1.0])

>>> np.polydiv(x, y)

(array([1.5 , 1.75]), array([0.25]))

polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)

Least squares polynomial fit.

Fit a polynomial ``p(x) = p[0] \* x\*\*deg + ... + p[deg]`` of degree `deg`

to points `(x, y)`. Returns a vector of coefficients `p` that minimises

the squared error in the order `deg`, `deg-1`, ... `0`.

The `Polynomial.fit <numpy.polynomial.polynomial.Polynomial.fit>` class

method is recommended for new code as it is more stable numerically. See

the documentation of the method for more information.

Parameters

----------

x : array\_like, shape (M,)

x-coordinates of the M sample points ``(x[i], y[i])``.

y : array\_like, shape (M,) or (M, K)

y-coordinates of the sample points. Several data sets of sample

points sharing the same x-coordinates can be fitted at once by

passing in a 2D-array that contains one dataset per column.

deg : int

Degree of the fitting polynomial

rcond : float, optional

Relative condition number of the fit. Singular values smaller than

this relative to the largest singular value will be ignored. The

default value is len(x)\*eps, where eps is the relative precision of

the float type, about 2e-16 in most cases.

full : bool, optional

Switch determining nature of return value. When it is False (the

default) just the coefficients are returned, when True diagnostic

information from the singular value decomposition is also returned.

w : array\_like, shape (M,), optional

Weights to apply to the y-coordinates of the sample points. For

gaussian uncertainties, use 1/sigma (not 1/sigma\*\*2).

cov : bool or str, optional

If given and not `False`, return not just the estimate but also its

covariance matrix. By default, the covariance are scaled by

chi2/sqrt(N-dof), i.e., the weights are presumed to be unreliable

except in a relative sense and everything is scaled such that the

reduced chi2 is unity. This scaling is omitted if ``cov='unscaled'``,

as is relevant for the case that the weights are 1/sigma\*\*2, with

sigma known to be a reliable estimate of the uncertainty.

Returns

-------

p : ndarray, shape (deg + 1,) or (deg + 1, K)

Polynomial coefficients, highest power first. If `y` was 2-D, the

coefficients for `k`-th data set are in ``p[:,k]``.

residuals, rank, singular\_values, rcond

Present only if `full` = True. Residuals of the least-squares fit,

the effective rank of the scaled Vandermonde coefficient matrix,

its singular values, and the specified value of `rcond`. For more

details, see `linalg.lstsq`.

V : ndarray, shape (M,M) or (M,M,K)

Present only if `full` = False and `cov`=True. The covariance

matrix of the polynomial coefficient estimates. The diagonal of

this matrix are the variance estimates for each coefficient. If y

is a 2-D array, then the covariance matrix for the `k`-th data set

are in ``V[:,:,k]``

Warns

-----

RankWarning

The rank of the coefficient matrix in the least-squares fit is

deficient. The warning is only raised if `full` = False.

The warnings can be turned off by

>>> import warnings

>>> warnings.simplefilter('ignore', np.RankWarning)

See Also

--------

polyval : Compute polynomial values.

linalg.lstsq : Computes a least-squares fit.

scipy.interpolate.UnivariateSpline : Computes spline fits.

Notes

-----

The solution minimizes the squared error

.. math ::

E = \sum\_{j=0}^k |p(x\_j) - y\_j|^2

in the equations::

x[0]\*\*n \* p[0] + ... + x[0] \* p[n-1] + p[n] = y[0]

x[1]\*\*n \* p[0] + ... + x[1] \* p[n-1] + p[n] = y[1]

...

x[k]\*\*n \* p[0] + ... + x[k] \* p[n-1] + p[n] = y[k]

The coefficient matrix of the coefficients `p` is a Vandermonde matrix.

`polyfit` issues a `RankWarning` when the least-squares fit is badly

conditioned. This implies that the best fit is not well-defined due

to numerical error. The results may be improved by lowering the polynomial

degree or by replacing `x` by `x` - `x`.mean(). The `rcond` parameter

can also be set to a value smaller than its default, but the resulting

fit may be spurious: including contributions from the small singular

values can add numerical noise to the result.

Note that fitting polynomial coefficients is inherently badly conditioned

when the degree of the polynomial is large or the interval of sample points

is badly centered. The quality of the fit should always be checked in these

cases. When polynomial fits are not satisfactory, splines may be a good

alternative.

References

----------

.. [1] Wikipedia, "Curve fitting",

https://en.wikipedia.org/wiki/Curve\_fitting

.. [2] Wikipedia, "Polynomial interpolation",

https://en.wikipedia.org/wiki/Polynomial\_interpolation

Examples

--------

>>> import warnings

>>> x = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])

>>> y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])

>>> z = np.polyfit(x, y, 3)

>>> z

array([ 0.08703704, -0.81349206, 1.69312169, -0.03968254]) # may vary

It is convenient to use `poly1d` objects for dealing with polynomials:

>>> p = np.poly1d(z)

>>> p(0.5)

0.6143849206349179 # may vary

>>> p(3.5)

-0.34732142857143039 # may vary

>>> p(10)

22.579365079365115 # may vary

High-order polynomials may oscillate wildly:

>>> with warnings.catch\_warnings():

... warnings.simplefilter('ignore', np.RankWarning)

... p30 = np.poly1d(np.polyfit(x, y, 30))

...

>>> p30(4)

-0.80000000000000204 # may vary

>>> p30(5)

-0.99999999999999445 # may vary

>>> p30(4.5)

-0.10547061179440398 # may vary

Illustration:

>>> import matplotlib.pyplot as plt

>>> xp = np.linspace(-2, 6, 100)

>>> \_ = plt.plot(x, y, '.', xp, p(xp), '-', xp, p30(xp), '--')

>>> plt.ylim(-2,2)

(-2, 2)

>>> plt.show()

polyint(p, m=1, k=None)

Return an antiderivative (indefinite integral) of a polynomial.

The returned order `m` antiderivative `P` of polynomial `p` satisfies

:math:`\frac{d^m}{dx^m}P(x) = p(x)` and is defined up to `m - 1`

integration constants `k`. The constants determine the low-order

polynomial part

.. math:: \frac{k\_{m-1}}{0!} x^0 + \ldots + \frac{k\_0}{(m-1)!}x^{m-1}

of `P` so that :math:`P^{(j)}(0) = k\_{m-j-1}`.

Parameters

----------

p : array\_like or poly1d

Polynomial to integrate.

A sequence is interpreted as polynomial coefficients, see `poly1d`.

m : int, optional

Order of the antiderivative. (Default: 1)

k : list of `m` scalars or scalar, optional

Integration constants. They are given in the order of integration:

those corresponding to highest-order terms come first.

If ``None`` (default), all constants are assumed to be zero.

If `m = 1`, a single scalar can be given instead of a list.

See Also

--------

polyder : derivative of a polynomial

poly1d.integ : equivalent method

Examples

--------

The defining property of the antiderivative:

>>> p = np.poly1d([1,1,1])

>>> P = np.polyint(p)

>>> P

poly1d([ 0.33333333, 0.5 , 1. , 0. ]) # may vary

>>> np.polyder(P) == p

True

The integration constants default to zero, but can be specified:

>>> P = np.polyint(p, 3)

>>> P(0)

0.0

>>> np.polyder(P)(0)

0.0

>>> np.polyder(P, 2)(0)

0.0

>>> P = np.polyint(p, 3, k=[6,5,3])

>>> P

poly1d([ 0.01666667, 0.04166667, 0.16666667, 3. , 5. , 3. ]) # may vary

Note that 3 = 6 / 2!, and that the constants are given in the order of

integrations. Constant of the highest-order polynomial term comes first:

>>> np.polyder(P, 2)(0)

6.0

>>> np.polyder(P, 1)(0)

5.0

>>> P(0)

3.0

polymul(a1, a2)

Find the product of two polynomials.

Finds the polynomial resulting from the multiplication of the two input

polynomials. Each input must be either a poly1d object or a 1D sequence

of polynomial coefficients, from highest to lowest degree.

Parameters

----------

a1, a2 : array\_like or poly1d object

Input polynomials.

Returns

-------

out : ndarray or poly1d object

The polynomial resulting from the multiplication of the inputs. If

either inputs is a poly1d object, then the output is also a poly1d

object. Otherwise, it is a 1D array of polynomial coefficients from

highest to lowest degree.

See Also

--------

poly1d : A one-dimensional polynomial class.

poly, polyadd, polyder, polydiv, polyfit, polyint, polysub, polyval

convolve : Array convolution. Same output as polymul, but has parameter

for overlap mode.

Examples

--------

>>> np.polymul([1, 2, 3], [9, 5, 1])

array([ 9, 23, 38, 17, 3])

Using poly1d objects:

>>> p1 = np.poly1d([1, 2, 3])

>>> p2 = np.poly1d([9, 5, 1])

>>> print(p1)

2

1 x + 2 x + 3

>>> print(p2)

2

9 x + 5 x + 1

>>> print(np.polymul(p1, p2))

4 3 2

9 x + 23 x + 38 x + 17 x + 3

polysub(a1, a2)

Difference (subtraction) of two polynomials.

Given two polynomials `a1` and `a2`, returns ``a1 - a2``.

`a1` and `a2` can be either array\_like sequences of the polynomials'

coefficients (including coefficients equal to zero), or `poly1d` objects.

Parameters

----------

a1, a2 : array\_like or poly1d

Minuend and subtrahend polynomials, respectively.

Returns

-------

out : ndarray or poly1d

Array or `poly1d` object of the difference polynomial's coefficients.

See Also

--------

polyval, polydiv, polymul, polyadd

Examples

--------

.. math:: (2 x^2 + 10 x - 2) - (3 x^2 + 10 x -4) = (-x^2 + 2)

>>> np.polysub([2, 10, -2], [3, 10, -4])

array([-1, 0, 2])

polyval(p, x)

Evaluate a polynomial at specific values.

If `p` is of length N, this function returns the value:

``p[0]\*x\*\*(N-1) + p[1]\*x\*\*(N-2) + ... + p[N-2]\*x + p[N-1]``

If `x` is a sequence, then `p(x)` is returned for each element of `x`.

If `x` is another polynomial then the composite polynomial `p(x(t))`

is returned.

Parameters

----------

p : array\_like or poly1d object

1D array of polynomial coefficients (including coefficients equal

to zero) from highest degree to the constant term, or an

instance of poly1d.

x : array\_like or poly1d object

A number, an array of numbers, or an instance of poly1d, at

which to evaluate `p`.

Returns

-------

values : ndarray or poly1d

If `x` is a poly1d instance, the result is the composition of the two

polynomials, i.e., `x` is "substituted" in `p` and the simplified

result is returned. In addition, the type of `x` - array\_like or

poly1d - governs the type of the output: `x` array\_like => `values`

array\_like, `x` a poly1d object => `values` is also.

See Also

--------

poly1d: A polynomial class.

Notes

-----

Horner's scheme [1]\_ is used to evaluate the polynomial. Even so,

for polynomials of high degree the values may be inaccurate due to

rounding errors. Use carefully.

If `x` is a subtype of `ndarray` the return value will be of the same type.

References

----------

.. [1] I. N. Bronshtein, K. A. Semendyayev, and K. A. Hirsch (Eng.

trans. Ed.), \*Handbook of Mathematics\*, New York, Van Nostrand

Reinhold Co., 1985, pg. 720.

Examples

--------

>>> np.polyval([3,0,1], 5) # 3 \* 5\*\*2 + 0 \* 5\*\*1 + 1

76

>>> np.polyval([3,0,1], np.poly1d(5))

poly1d([76.])

>>> np.polyval(np.poly1d([3,0,1]), 5)

76

>>> np.polyval(np.poly1d([3,0,1]), np.poly1d(5))

poly1d([76.])

ppmt(rate, per, nper, pv, fv=0, when='end')

Compute the payment against loan principal.

Parameters

----------

rate : array\_like

Rate of interest (per period)

per : array\_like, int

Amount paid against the loan changes. The `per` is the period of

interest.

nper : array\_like

Number of compounding periods

pv : array\_like

Present value

fv : array\_like, optional

Future value

when : {{'begin', 1}, {'end', 0}}, {string, int}

When payments are due ('begin' (1) or 'end' (0))

See Also

--------

pmt, pv, ipmt

printoptions(\*args, \*\*kwargs)

Context manager for setting print options.

Set print options for the scope of the `with` block, and restore the old

options at the end. See `set\_printoptions` for the full description of

available options.

Examples

--------

>>> from numpy.testing import assert\_equal

>>> with np.printoptions(precision=2):

... np.array([2.0]) / 3

array([0.67])

The `as`-clause of the `with`-statement gives the current print options:

>>> with np.printoptions(precision=2) as opts:

... assert\_equal(opts, np.get\_printoptions())

See Also

--------

set\_printoptions, get\_printoptions

prod(a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)

Return the product of array elements over a given axis.

Parameters

----------

a : array\_like

Input data.

axis : None or int or tuple of ints, optional

Axis or axes along which a product is performed. The default,

axis=None, will calculate the product of all the elements in the

input array. If axis is negative it counts from the last to the

first axis.

.. versionadded:: 1.7.0

If axis is a tuple of ints, a product is performed on all of the

axes specified in the tuple instead of a single axis or all the

axes as before.

dtype : dtype, optional

The type of the returned array, as well as of the accumulator in

which the elements are multiplied. The dtype of `a` is used by

default unless `a` has an integer dtype of less precision than the

default platform integer. In that case, if `a` is signed then the

platform integer is used while if `a` is unsigned then an unsigned

integer of the same precision as the platform integer is used.

out : ndarray, optional

Alternative output array in which to place the result. It must have

the same shape as the expected output, but the type of the output

values will be cast if necessary.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left in the

result as dimensions with size one. With this option, the result

will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `prod` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

initial : scalar, optional

The starting value for this product. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.15.0

where : array\_like of bool, optional

Elements to include in the product. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.17.0

Returns

-------

product\_along\_axis : ndarray, see `dtype` parameter above.

An array shaped as `a` but with the specified axis removed.

Returns a reference to `out` if specified.

See Also

--------

ndarray.prod : equivalent method

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

Arithmetic is modular when using integer types, and no error is

raised on overflow. That means that, on a 32-bit platform:

>>> x = np.array([536870910, 536870910, 536870910, 536870910])

>>> np.prod(x)

16 # may vary

The product of an empty array is the neutral element 1:

>>> np.prod([])

1.0

Examples

--------

By default, calculate the product of all elements:

>>> np.prod([1.,2.])

2.0

Even when the input array is two-dimensional:

>>> np.prod([[1.,2.],[3.,4.]])

24.0

But we can also specify the axis over which to multiply:

>>> np.prod([[1.,2.],[3.,4.]], axis=1)

array([ 2., 12.])

Or select specific elements to include:

>>> np.prod([1., np.nan, 3.], where=[True, False, True])

3.0

If the type of `x` is unsigned, then the output type is

the unsigned platform integer:

>>> x = np.array([1, 2, 3], dtype=np.uint8)

>>> np.prod(x).dtype == np.uint

True

If `x` is of a signed integer type, then the output type

is the default platform integer:

>>> x = np.array([1, 2, 3], dtype=np.int8)

>>> np.prod(x).dtype == int

True

You can also start the product with a value other than one:

>>> np.prod([1, 2], initial=5)

10

product(\*args, \*\*kwargs)

Return the product of array elements over a given axis.

See Also

--------

prod : equivalent function; see for details.

promote\_types(...)

promote\_types(type1, type2)

Returns the data type with the smallest size and smallest scalar

kind to which both ``type1`` and ``type2`` may be safely cast.

The returned data type is always in native byte order.

This function is symmetric, but rarely associative.

Parameters

----------

type1 : dtype or dtype specifier

First data type.

type2 : dtype or dtype specifier

Second data type.

Returns

-------

out : dtype

The promoted data type.

Notes

-----

.. versionadded:: 1.6.0

Starting in NumPy 1.9, promote\_types function now returns a valid string

length when given an integer or float dtype as one argument and a string

dtype as another argument. Previously it always returned the input string

dtype, even if it wasn't long enough to store the max integer/float value

converted to a string.

See Also

--------

result\_type, dtype, can\_cast

Examples

--------

>>> np.promote\_types('f4', 'f8')

dtype('float64')

>>> np.promote\_types('i8', 'f4')

dtype('float64')

>>> np.promote\_types('>i8', '<c8')

dtype('complex128')

>>> np.promote\_types('i4', 'S8')

dtype('S11')

An example of a non-associative case:

>>> p = np.promote\_types

>>> p('S', p('i1', 'u1'))

dtype('S6')

>>> p(p('S', 'i1'), 'u1')

dtype('S4')

ptp(a, axis=None, out=None, keepdims=<no value>)

Range of values (maximum - minimum) along an axis.

The name of the function comes from the acronym for 'peak to peak'.

Parameters

----------

a : array\_like

Input values.

axis : None or int or tuple of ints, optional

Axis along which to find the peaks. By default, flatten the

array. `axis` may be negative, in

which case it counts from the last to the first axis.

.. versionadded:: 1.15.0

If this is a tuple of ints, a reduction is performed on multiple

axes, instead of a single axis or all the axes as before.

out : array\_like

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type of the output values will be cast if necessary.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `ptp` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

ptp : ndarray

A new array holding the result, unless `out` was

specified, in which case a reference to `out` is returned.

Examples

--------

>>> x = np.arange(4).reshape((2,2))

>>> x

array([[0, 1],

[2, 3]])

>>> np.ptp(x, axis=0)

array([2, 2])

>>> np.ptp(x, axis=1)

array([1, 1])

put(a, ind, v, mode='raise')

Replaces specified elements of an array with given values.

The indexing works on the flattened target array. `put` is roughly

equivalent to:

::

a.flat[ind] = v

Parameters

----------

a : ndarray

Target array.

ind : array\_like

Target indices, interpreted as integers.

v : array\_like

Values to place in `a` at target indices. If `v` is shorter than

`ind` it will be repeated as necessary.

mode : {'raise', 'wrap', 'clip'}, optional

Specifies how out-of-bounds indices will behave.

\* 'raise' -- raise an error (default)

\* 'wrap' -- wrap around

\* 'clip' -- clip to the range

'clip' mode means that all indices that are too large are replaced

by the index that addresses the last element along that axis. Note

that this disables indexing with negative numbers. In 'raise' mode,

if an exception occurs the target array may still be modified.

See Also

--------

putmask, place

put\_along\_axis : Put elements by matching the array and the index arrays

Examples

--------

>>> a = np.arange(5)

>>> np.put(a, [0, 2], [-44, -55])

>>> a

array([-44, 1, -55, 3, 4])

>>> a = np.arange(5)

>>> np.put(a, 22, -5, mode='clip')

>>> a

array([ 0, 1, 2, 3, -5])

put\_along\_axis(arr, indices, values, axis)

Put values into the destination array by matching 1d index and data slices.

This iterates over matching 1d slices oriented along the specified axis in

the index and data arrays, and uses the former to place values into the

latter. These slices can be different lengths.

Functions returning an index along an axis, like `argsort` and

`argpartition`, produce suitable indices for this function.

.. versionadded:: 1.15.0

Parameters

----------

arr: ndarray (Ni..., M, Nk...)

Destination array.

indices: ndarray (Ni..., J, Nk...)

Indices to change along each 1d slice of `arr`. This must match the

dimension of arr, but dimensions in Ni and Nj may be 1 to broadcast

against `arr`.

values: array\_like (Ni..., J, Nk...)

values to insert at those indices. Its shape and dimension are

broadcast to match that of `indices`.

axis: int

The axis to take 1d slices along. If axis is None, the destination

array is treated as if a flattened 1d view had been created of it.

Notes

-----

This is equivalent to (but faster than) the following use of `ndindex` and

`s\_`, which sets each of ``ii`` and ``kk`` to a tuple of indices::

Ni, M, Nk = a.shape[:axis], a.shape[axis], a.shape[axis+1:]

J = indices.shape[axis] # Need not equal M

for ii in ndindex(Ni):

for kk in ndindex(Nk):

a\_1d = a [ii + s\_[:,] + kk]

indices\_1d = indices[ii + s\_[:,] + kk]

values\_1d = values [ii + s\_[:,] + kk]

for j in range(J):

a\_1d[indices\_1d[j]] = values\_1d[j]

Equivalently, eliminating the inner loop, the last two lines would be::

a\_1d[indices\_1d] = values\_1d

See Also

--------

take\_along\_axis :

Take values from the input array by matching 1d index and data slices

Examples

--------

For this sample array

>>> a = np.array([[10, 30, 20], [60, 40, 50]])

We can replace the maximum values with:

>>> ai = np.expand\_dims(np.argmax(a, axis=1), axis=1)

>>> ai

array([[1],

[0]])

>>> np.put\_along\_axis(a, ai, 99, axis=1)

>>> a

array([[10, 99, 20],

[99, 40, 50]])

putmask(...)

putmask(a, mask, values)

Changes elements of an array based on conditional and input values.

Sets ``a.flat[n] = values[n]`` for each n where ``mask.flat[n]==True``.

If `values` is not the same size as `a` and `mask` then it will repeat.

This gives behavior different from ``a[mask] = values``.

Parameters

----------

a : array\_like

Target array.

mask : array\_like

Boolean mask array. It has to be the same shape as `a`.

values : array\_like

Values to put into `a` where `mask` is True. If `values` is smaller

than `a` it will be repeated.

See Also

--------

place, put, take, copyto

Examples

--------

>>> x = np.arange(6).reshape(2, 3)

>>> np.putmask(x, x>2, x\*\*2)

>>> x

array([[ 0, 1, 2],

[ 9, 16, 25]])

If `values` is smaller than `a` it is repeated:

>>> x = np.arange(5)

>>> np.putmask(x, x>1, [-33, -44])

>>> x

array([ 0, 1, -33, -44, -33])

pv(rate, nper, pmt, fv=0, when='end')

Compute the present value.

Given:

\* a future value, `fv`

\* an interest `rate` compounded once per period, of which

there are

\* `nper` total

\* a (fixed) payment, `pmt`, paid either

\* at the beginning (`when` = {'begin', 1}) or the end

(`when` = {'end', 0}) of each period

Return:

the value now

Parameters

----------

rate : array\_like

Rate of interest (per period)

nper : array\_like

Number of compounding periods

pmt : array\_like

Payment

fv : array\_like, optional

Future value

when : {{'begin', 1}, {'end', 0}}, {string, int}, optional

When payments are due ('begin' (1) or 'end' (0))

Returns

-------

out : ndarray, float

Present value of a series of payments or investments.

Notes

-----

The present value is computed by solving the equation::

fv +

pv\*(1 + rate)\*\*nper +

pmt\*(1 + rate\*when)/rate\*((1 + rate)\*\*nper - 1) = 0

or, when ``rate = 0``::

fv + pv + pmt \* nper = 0

for `pv`, which is then returned.

References

----------

.. [WRW] Wheeler, D. A., E. Rathke, and R. Weir (Eds.) (2009, May).

Open Document Format for Office Applications (OpenDocument)v1.2,

Part 2: Recalculated Formula (OpenFormula) Format - Annotated Version,

Pre-Draft 12. Organization for the Advancement of Structured Information

Standards (OASIS). Billerica, MA, USA. [ODT Document].

Available:

http://www.oasis-open.org/committees/documents.php?wg\_abbrev=office-formula

OpenDocument-formula-20090508.odt

Examples

--------

What is the present value (e.g., the initial investment)

of an investment that needs to total $15692.93

after 10 years of saving $100 every month? Assume the

interest rate is 5% (annually) compounded monthly.

>>> np.pv(0.05/12, 10\*12, -100, 15692.93)

-100.00067131625819

By convention, the negative sign represents cash flow out

(i.e., money not available today). Thus, to end up with

$15,692.93 in 10 years saving $100 a month at 5% annual

interest, one's initial deposit should also be $100.

If any input is array\_like, ``pv`` returns an array of equal shape.

Let's compare different interest rates in the example above:

>>> a = np.array((0.05, 0.04, 0.03))/12

>>> np.pv(a, 10\*12, -100, 15692.93)

array([ -100.00067132, -649.26771385, -1273.78633713]) # may vary

So, to end up with the same $15692.93 under the same $100 per month

"savings plan," for annual interest rates of 4% and 3%, one would

need initial investments of $649.27 and $1273.79, respectively.

quantile(a, q, axis=None, out=None, overwrite\_input=False, interpolation='linear', keepdims=False)

Compute the q-th quantile of the data along the specified axis.

.. versionadded:: 1.15.0

Parameters

----------

a : array\_like

Input array or object that can be converted to an array.

q : array\_like of float

Quantile or sequence of quantiles to compute, which must be between

0 and 1 inclusive.

axis : {int, tuple of int, None}, optional

Axis or axes along which the quantiles are computed. The

default is to compute the quantile(s) along a flattened

version of the array.

out : ndarray, optional

Alternative output array in which to place the result. It must

have the same shape and buffer length as the expected output,

but the type (of the output) will be cast if necessary.

overwrite\_input : bool, optional

If True, then allow the input array `a` to be modified by intermediate

calculations, to save memory. In this case, the contents of the input

`a` after this function completes is undefined.

interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}

This optional parameter specifies the interpolation method to

use when the desired quantile lies between two data points

``i < j``:

\* linear: ``i + (j - i) \* fraction``, where ``fraction``

is the fractional part of the index surrounded by ``i``

and ``j``.

\* lower: ``i``.

\* higher: ``j``.

\* nearest: ``i`` or ``j``, whichever is nearest.

\* midpoint: ``(i + j) / 2``.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left in

the result as dimensions with size one. With this option, the

result will broadcast correctly against the original array `a`.

Returns

-------

quantile : scalar or ndarray

If `q` is a single quantile and `axis=None`, then the result

is a scalar. If multiple quantiles are given, first axis of

the result corresponds to the quantiles. The other axes are

the axes that remain after the reduction of `a`. If the input

contains integers or floats smaller than ``float64``, the output

data-type is ``float64``. Otherwise, the output data-type is the

same as that of the input. If `out` is specified, that array is

returned instead.

See Also

--------

mean

percentile : equivalent to quantile, but with q in the range [0, 100].

median : equivalent to ``quantile(..., 0.5)``

nanquantile

Notes

-----

Given a vector ``V`` of length ``N``, the q-th quantile of

``V`` is the value ``q`` of the way from the minimum to the

maximum in a sorted copy of ``V``. The values and distances of

the two nearest neighbors as well as the `interpolation` parameter

will determine the quantile if the normalized ranking does not

match the location of ``q`` exactly. This function is the same as

the median if ``q=0.5``, the same as the minimum if ``q=0.0`` and the

same as the maximum if ``q=1.0``.

Examples

--------

>>> a = np.array([[10, 7, 4], [3, 2, 1]])

>>> a

array([[10, 7, 4],

[ 3, 2, 1]])

>>> np.quantile(a, 0.5)

3.5

>>> np.quantile(a, 0.5, axis=0)

array([6.5, 4.5, 2.5])

>>> np.quantile(a, 0.5, axis=1)

array([7., 2.])

>>> np.quantile(a, 0.5, axis=1, keepdims=True)

array([[7.],

[2.]])

>>> m = np.quantile(a, 0.5, axis=0)

>>> out = np.zeros\_like(m)

>>> np.quantile(a, 0.5, axis=0, out=out)

array([6.5, 4.5, 2.5])

>>> m

array([6.5, 4.5, 2.5])

>>> b = a.copy()

>>> np.quantile(b, 0.5, axis=1, overwrite\_input=True)

array([7., 2.])

>>> assert not np.all(a == b)

rank(a)

Return the number of dimensions of an array.

.. note::

This function is deprecated in NumPy 1.9 to avoid confusion with

`numpy.linalg.matrix\_rank`. The ``ndim`` attribute or function

should be used instead.

See Also

--------

ndim : equivalent non-deprecated function

Notes

-----

In the old Numeric package, `rank` was the term used for the number of

dimensions, but in NumPy `ndim` is used instead.

rate(nper, pmt, pv, fv, when='end', guess=None, tol=None, maxiter=100)

Compute the rate of interest per period.

Parameters

----------

nper : array\_like

Number of compounding periods

pmt : array\_like

Payment

pv : array\_like

Present value

fv : array\_like

Future value

when : {{'begin', 1}, {'end', 0}}, {string, int}, optional

When payments are due ('begin' (1) or 'end' (0))

guess : Number, optional

Starting guess for solving the rate of interest, default 0.1

tol : Number, optional

Required tolerance for the solution, default 1e-6

maxiter : int, optional

Maximum iterations in finding the solution

Notes

-----

The rate of interest is computed by iteratively solving the

(non-linear) equation::

fv + pv\*(1+rate)\*\*nper + pmt\*(1+rate\*when)/rate \* ((1+rate)\*\*nper - 1) = 0

for ``rate``.

References

----------

Wheeler, D. A., E. Rathke, and R. Weir (Eds.) (2009, May). Open Document

Format for Office Applications (OpenDocument)v1.2, Part 2: Recalculated

Formula (OpenFormula) Format - Annotated Version, Pre-Draft 12.

Organization for the Advancement of Structured Information Standards

(OASIS). Billerica, MA, USA. [ODT Document]. Available:

http://www.oasis-open.org/committees/documents.php?wg\_abbrev=office-formula

OpenDocument-formula-20090508.odt

ravel(a, order='C')

Return a contiguous flattened array.

A 1-D array, containing the elements of the input, is returned. A copy is

made only if needed.

As of NumPy 1.10, the returned array will have the same type as the input

array. (for example, a masked array will be returned for a masked array

input)

Parameters

----------

a : array\_like

Input array. The elements in `a` are read in the order specified by

`order`, and packed as a 1-D array.

order : {'C','F', 'A', 'K'}, optional

The elements of `a` are read using this index order. 'C' means

to index the elements in row-major, C-style order,

with the last axis index changing fastest, back to the first

axis index changing slowest. 'F' means to index the elements

in column-major, Fortran-style order, with the

first index changing fastest, and the last index changing

slowest. Note that the 'C' and 'F' options take no account of

the memory layout of the underlying array, and only refer to

the order of axis indexing. 'A' means to read the elements in

Fortran-like index order if `a` is Fortran \*contiguous\* in

memory, C-like order otherwise. 'K' means to read the

elements in the order they occur in memory, except for

reversing the data when strides are negative. By default, 'C'

index order is used.

Returns

-------

y : array\_like

y is an array of the same subtype as `a`, with shape ``(a.size,)``.

Note that matrices are special cased for backward compatibility, if `a`

is a matrix, then y is a 1-D ndarray.

See Also

--------

ndarray.flat : 1-D iterator over an array.

ndarray.flatten : 1-D array copy of the elements of an array

in row-major order.

ndarray.reshape : Change the shape of an array without changing its data.

Notes

-----

In row-major, C-style order, in two dimensions, the row index

varies the slowest, and the column index the quickest. This can

be generalized to multiple dimensions, where row-major order

implies that the index along the first axis varies slowest, and

the index along the last quickest. The opposite holds for

column-major, Fortran-style index ordering.

When a view is desired in as many cases as possible, ``arr.reshape(-1)``

may be preferable.

Examples

--------

It is equivalent to ``reshape(-1, order=order)``.

>>> x = np.array([[1, 2, 3], [4, 5, 6]])

>>> np.ravel(x)

array([1, 2, 3, 4, 5, 6])

>>> x.reshape(-1)

array([1, 2, 3, 4, 5, 6])

>>> np.ravel(x, order='F')

array([1, 4, 2, 5, 3, 6])

When ``order`` is 'A', it will preserve the array's 'C' or 'F' ordering:

>>> np.ravel(x.T)

array([1, 4, 2, 5, 3, 6])

>>> np.ravel(x.T, order='A')

array([1, 2, 3, 4, 5, 6])

When ``order`` is 'K', it will preserve orderings that are neither 'C'

nor 'F', but won't reverse axes:

>>> a = np.arange(3)[::-1]; a

array([2, 1, 0])

>>> a.ravel(order='C')

array([2, 1, 0])

>>> a.ravel(order='K')

array([2, 1, 0])

>>> a = np.arange(12).reshape(2,3,2).swapaxes(1,2); a

array([[[ 0, 2, 4],

[ 1, 3, 5]],

[[ 6, 8, 10],

[ 7, 9, 11]]])

>>> a.ravel(order='C')

array([ 0, 2, 4, 1, 3, 5, 6, 8, 10, 7, 9, 11])

>>> a.ravel(order='K')

array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

ravel\_multi\_index(...)

ravel\_multi\_index(multi\_index, dims, mode='raise', order='C')

Converts a tuple of index arrays into an array of flat

indices, applying boundary modes to the multi-index.

Parameters

----------

multi\_index : tuple of array\_like

A tuple of integer arrays, one array for each dimension.

dims : tuple of ints

The shape of array into which the indices from ``multi\_index`` apply.

mode : {'raise', 'wrap', 'clip'}, optional

Specifies how out-of-bounds indices are handled. Can specify

either one mode or a tuple of modes, one mode per index.

\* 'raise' -- raise an error (default)

\* 'wrap' -- wrap around

\* 'clip' -- clip to the range

In 'clip' mode, a negative index which would normally

wrap will clip to 0 instead.

order : {'C', 'F'}, optional

Determines whether the multi-index should be viewed as

indexing in row-major (C-style) or column-major

(Fortran-style) order.

Returns

-------

raveled\_indices : ndarray

An array of indices into the flattened version of an array

of dimensions ``dims``.

See Also

--------

unravel\_index

Notes

-----

.. versionadded:: 1.6.0

Examples

--------

>>> arr = np.array([[3,6,6],[4,5,1]])

>>> np.ravel\_multi\_index(arr, (7,6))

array([22, 41, 37])

>>> np.ravel\_multi\_index(arr, (7,6), order='F')

array([31, 41, 13])

>>> np.ravel\_multi\_index(arr, (4,6), mode='clip')

array([22, 23, 19])

>>> np.ravel\_multi\_index(arr, (4,4), mode=('clip','wrap'))

array([12, 13, 13])

>>> np.ravel\_multi\_index((3,1,4,1), (6,7,8,9))

1621

real(val)

Return the real part of the complex argument.

Parameters

----------

val : array\_like

Input array.

Returns

-------

out : ndarray or scalar

The real component of the complex argument. If `val` is real, the type

of `val` is used for the output. If `val` has complex elements, the

returned type is float.

See Also

--------

real\_if\_close, imag, angle

Examples

--------

>>> a = np.array([1+2j, 3+4j, 5+6j])

>>> a.real

array([1., 3., 5.])

>>> a.real = 9

>>> a

array([9.+2.j, 9.+4.j, 9.+6.j])

>>> a.real = np.array([9, 8, 7])

>>> a

array([9.+2.j, 8.+4.j, 7.+6.j])

>>> np.real(1 + 1j)

1.0

real\_if\_close(a, tol=100)

If complex input returns a real array if complex parts are close to zero.

"Close to zero" is defined as `tol` \* (machine epsilon of the type for

`a`).

Parameters

----------

a : array\_like

Input array.

tol : float

Tolerance in machine epsilons for the complex part of the elements

in the array.

Returns

-------

out : ndarray

If `a` is real, the type of `a` is used for the output. If `a`

has complex elements, the returned type is float.

See Also

--------

real, imag, angle

Notes

-----

Machine epsilon varies from machine to machine and between data types

but Python floats on most platforms have a machine epsilon equal to

2.2204460492503131e-16. You can use 'np.finfo(float).eps' to print

out the machine epsilon for floats.

Examples

--------

>>> np.finfo(float).eps

2.2204460492503131e-16 # may vary

>>> np.real\_if\_close([2.1 + 4e-14j], tol=1000)

array([2.1])

>>> np.real\_if\_close([2.1 + 4e-13j], tol=1000)

array([2.1+4.e-13j])

recfromcsv(fname, \*\*kwargs)

Load ASCII data stored in a comma-separated file.

The returned array is a record array (if ``usemask=False``, see

`recarray`) or a masked record array (if ``usemask=True``,

see `ma.mrecords.MaskedRecords`).

Parameters

----------

fname, kwargs : For a description of input parameters, see `genfromtxt`.

See Also

--------

numpy.genfromtxt : generic function to load ASCII data.

Notes

-----

By default, `dtype` is None, which means that the data-type of the output

array will be determined from the data.

recfromtxt(fname, \*\*kwargs)

Load ASCII data from a file and return it in a record array.

If ``usemask=False`` a standard `recarray` is returned,

if ``usemask=True`` a MaskedRecords array is returned.

Parameters

----------

fname, kwargs : For a description of input parameters, see `genfromtxt`.

See Also

--------

numpy.genfromtxt : generic function

Notes

-----

By default, `dtype` is None, which means that the data-type of the output

array will be determined from the data.

repeat(a, repeats, axis=None)

Repeat elements of an array.

Parameters

----------

a : array\_like

Input array.

repeats : int or array of ints

The number of repetitions for each element. `repeats` is broadcasted

to fit the shape of the given axis.

axis : int, optional

The axis along which to repeat values. By default, use the

flattened input array, and return a flat output array.

Returns

-------

repeated\_array : ndarray

Output array which has the same shape as `a`, except along

the given axis.

See Also

--------

tile : Tile an array.

Examples

--------

>>> np.repeat(3, 4)

array([3, 3, 3, 3])

>>> x = np.array([[1,2],[3,4]])

>>> np.repeat(x, 2)

array([1, 1, 2, 2, 3, 3, 4, 4])

>>> np.repeat(x, 3, axis=1)

array([[1, 1, 1, 2, 2, 2],

[3, 3, 3, 4, 4, 4]])

>>> np.repeat(x, [1, 2], axis=0)

array([[1, 2],

[3, 4],

[3, 4]])

require(a, dtype=None, requirements=None)

Return an ndarray of the provided type that satisfies requirements.

This function is useful to be sure that an array with the correct flags

is returned for passing to compiled code (perhaps through ctypes).

Parameters

----------

a : array\_like

The object to be converted to a type-and-requirement-satisfying array.

dtype : data-type

The required data-type. If None preserve the current dtype. If your

application requires the data to be in native byteorder, include

a byteorder specification as a part of the dtype specification.

requirements : str or list of str

The requirements list can be any of the following

\* 'F\_CONTIGUOUS' ('F') - ensure a Fortran-contiguous array

\* 'C\_CONTIGUOUS' ('C') - ensure a C-contiguous array

\* 'ALIGNED' ('A') - ensure a data-type aligned array

\* 'WRITEABLE' ('W') - ensure a writable array

\* 'OWNDATA' ('O') - ensure an array that owns its own data

\* 'ENSUREARRAY', ('E') - ensure a base array, instead of a subclass

Returns

-------

out : ndarray

Array with specified requirements and type if given.

See Also

--------

asarray : Convert input to an ndarray.

asanyarray : Convert to an ndarray, but pass through ndarray subclasses.

ascontiguousarray : Convert input to a contiguous array.

asfortranarray : Convert input to an ndarray with column-major

memory order.

ndarray.flags : Information about the memory layout of the array.

Notes

-----

The returned array will be guaranteed to have the listed requirements

by making a copy if needed.

Examples

--------

>>> x = np.arange(6).reshape(2,3)

>>> x.flags

C\_CONTIGUOUS : True

F\_CONTIGUOUS : False

OWNDATA : False

WRITEABLE : True

ALIGNED : True

WRITEBACKIFCOPY : False

UPDATEIFCOPY : False

>>> y = np.require(x, dtype=np.float32, requirements=['A', 'O', 'W', 'F'])

>>> y.flags

C\_CONTIGUOUS : False

F\_CONTIGUOUS : True

OWNDATA : True

WRITEABLE : True

ALIGNED : True

WRITEBACKIFCOPY : False

UPDATEIFCOPY : False

reshape(a, newshape, order='C')

Gives a new shape to an array without changing its data.

Parameters

----------

a : array\_like

Array to be reshaped.

newshape : int or tuple of ints

The new shape should be compatible with the original shape. If

an integer, then the result will be a 1-D array of that length.

One shape dimension can be -1. In this case, the value is

inferred from the length of the array and remaining dimensions.

order : {'C', 'F', 'A'}, optional

Read the elements of `a` using this index order, and place the

elements into the reshaped array using this index order. 'C'

means to read / write the elements using C-like index order,

with the last axis index changing fastest, back to the first

axis index changing slowest. 'F' means to read / write the

elements using Fortran-like index order, with the first index

changing fastest, and the last index changing slowest. Note that

the 'C' and 'F' options take no account of the memory layout of

the underlying array, and only refer to the order of indexing.

'A' means to read / write the elements in Fortran-like index

order if `a` is Fortran \*contiguous\* in memory, C-like order

otherwise.

Returns

-------

reshaped\_array : ndarray

This will be a new view object if possible; otherwise, it will

be a copy. Note there is no guarantee of the \*memory layout\* (C- or

Fortran- contiguous) of the returned array.

See Also

--------

ndarray.reshape : Equivalent method.

Notes

-----

It is not always possible to change the shape of an array without

copying the data. If you want an error to be raised when the data is copied,

you should assign the new shape to the shape attribute of the array::

>>> a = np.zeros((10, 2))

# A transpose makes the array non-contiguous

>>> b = a.T

# Taking a view makes it possible to modify the shape without modifying

# the initial object.

>>> c = b.view()

>>> c.shape = (20)

Traceback (most recent call last):

...

AttributeError: incompatible shape for a non-contiguous array

The `order` keyword gives the index ordering both for \*fetching\* the values

from `a`, and then \*placing\* the values into the output array.

For example, let's say you have an array:

>>> a = np.arange(6).reshape((3, 2))

>>> a

array([[0, 1],

[2, 3],

[4, 5]])

You can think of reshaping as first raveling the array (using the given

index order), then inserting the elements from the raveled array into the

new array using the same kind of index ordering as was used for the

raveling.

>>> np.reshape(a, (2, 3)) # C-like index ordering

array([[0, 1, 2],

[3, 4, 5]])

>>> np.reshape(np.ravel(a), (2, 3)) # equivalent to C ravel then C reshape

array([[0, 1, 2],

[3, 4, 5]])

>>> np.reshape(a, (2, 3), order='F') # Fortran-like index ordering

array([[0, 4, 3],

[2, 1, 5]])

>>> np.reshape(np.ravel(a, order='F'), (2, 3), order='F')

array([[0, 4, 3],

[2, 1, 5]])

Examples

--------

>>> a = np.array([[1,2,3], [4,5,6]])

>>> np.reshape(a, 6)

array([1, 2, 3, 4, 5, 6])

>>> np.reshape(a, 6, order='F')

array([1, 4, 2, 5, 3, 6])

>>> np.reshape(a, (3,-1)) # the unspecified value is inferred to be 2

array([[1, 2],

[3, 4],

[5, 6]])

resize(a, new\_shape)

Return a new array with the specified shape.

If the new array is larger than the original array, then the new

array is filled with repeated copies of `a`. Note that this behavior

is different from a.resize(new\_shape) which fills with zeros instead

of repeated copies of `a`.

Parameters

----------

a : array\_like

Array to be resized.

new\_shape : int or tuple of int

Shape of resized array.

Returns

-------

reshaped\_array : ndarray

The new array is formed from the data in the old array, repeated

if necessary to fill out the required number of elements. The

data are repeated in the order that they are stored in memory.

See Also

--------

ndarray.resize : resize an array in-place.

Notes

-----

Warning: This functionality does \*\*not\*\* consider axes separately,

i.e. it does not apply interpolation/extrapolation.

It fills the return array with the required number of elements, taken

from `a` as they are laid out in memory, disregarding strides and axes.

(This is in case the new shape is smaller. For larger, see above.)

This functionality is therefore not suitable to resize images,

or data where each axis represents a separate and distinct entity.

Examples

--------

>>> a=np.array([[0,1],[2,3]])

>>> np.resize(a,(2,3))

array([[0, 1, 2],

[3, 0, 1]])

>>> np.resize(a,(1,4))

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

>>> np.resize(a,(2,4))

array([[0, 1, 2, 3],

[0, 1, 2, 3]])

result\_type(...)

result\_type(\*arrays\_and\_dtypes)

Returns the type that results from applying the NumPy

type promotion rules to the arguments.

Type promotion in NumPy works similarly to the rules in languages

like C++, with some slight differences. When both scalars and

arrays are used, the array's type takes precedence and the actual value

of the scalar is taken into account.

For example, calculating 3\*a, where a is an array of 32-bit floats,

intuitively should result in a 32-bit float output. If the 3 is a

32-bit integer, the NumPy rules indicate it can't convert losslessly

into a 32-bit float, so a 64-bit float should be the result type.

By examining the value of the constant, '3', we see that it fits in

an 8-bit integer, which can be cast losslessly into the 32-bit float.

Parameters

----------

arrays\_and\_dtypes : list of arrays and dtypes

The operands of some operation whose result type is needed.

Returns

-------

out : dtype

The result type.

See also

--------

dtype, promote\_types, min\_scalar\_type, can\_cast

Notes

-----

.. versionadded:: 1.6.0

The specific algorithm used is as follows.

Categories are determined by first checking which of boolean,

integer (int/uint), or floating point (float/complex) the maximum

kind of all the arrays and the scalars are.

If there are only scalars or the maximum category of the scalars

is higher than the maximum category of the arrays,

the data types are combined with :func:`promote\_types`

to produce the return value.

Otherwise, `min\_scalar\_type` is called on each array, and

the resulting data types are all combined with :func:`promote\_types`

to produce the return value.

The set of int values is not a subset of the uint values for types

with the same number of bits, something not reflected in

:func:`min\_scalar\_type`, but handled as a special case in `result\_type`.

Examples

--------

>>> np.result\_type(3, np.arange(7, dtype='i1'))

dtype('int8')

>>> np.result\_type('i4', 'c8')

dtype('complex128')

>>> np.result\_type(3.0, -2)

dtype('float64')

roll(a, shift, axis=None)

Roll array elements along a given axis.

Elements that roll beyond the last position are re-introduced at

the first.

Parameters

----------

a : array\_like

Input array.

shift : int or tuple of ints

The number of places by which elements are shifted. If a tuple,

then `axis` must be a tuple of the same size, and each of the

given axes is shifted by the corresponding number. If an int

while `axis` is a tuple of ints, then the same value is used for

all given axes.

axis : int or tuple of ints, optional

Axis or axes along which elements are shifted. By default, the

array is flattened before shifting, after which the original

shape is restored.

Returns

-------

res : ndarray

Output array, with the same shape as `a`.

See Also

--------

rollaxis : Roll the specified axis backwards, until it lies in a

given position.

Notes

-----

.. versionadded:: 1.12.0

Supports rolling over multiple dimensions simultaneously.

Examples

--------

>>> x = np.arange(10)

>>> np.roll(x, 2)

array([8, 9, 0, 1, 2, 3, 4, 5, 6, 7])

>>> np.roll(x, -2)

array([2, 3, 4, 5, 6, 7, 8, 9, 0, 1])

>>> x2 = np.reshape(x, (2,5))

>>> x2

array([[0, 1, 2, 3, 4],

[5, 6, 7, 8, 9]])

>>> np.roll(x2, 1)

array([[9, 0, 1, 2, 3],

[4, 5, 6, 7, 8]])

>>> np.roll(x2, -1)

array([[1, 2, 3, 4, 5],

[6, 7, 8, 9, 0]])

>>> np.roll(x2, 1, axis=0)

array([[5, 6, 7, 8, 9],

[0, 1, 2, 3, 4]])

>>> np.roll(x2, -1, axis=0)

array([[5, 6, 7, 8, 9],

[0, 1, 2, 3, 4]])

>>> np.roll(x2, 1, axis=1)

array([[4, 0, 1, 2, 3],

[9, 5, 6, 7, 8]])

>>> np.roll(x2, -1, axis=1)

array([[1, 2, 3, 4, 0],

[6, 7, 8, 9, 5]])

rollaxis(a, axis, start=0)

Roll the specified axis backwards, until it lies in a given position.

This function continues to be supported for backward compatibility, but you

should prefer `moveaxis`. The `moveaxis` function was added in NumPy

1.11.

Parameters

----------

a : ndarray

Input array.

axis : int

The axis to roll backwards. The positions of the other axes do not

change relative to one another.

start : int, optional

The axis is rolled until it lies before this position. The default,

0, results in a "complete" roll.

Returns

-------

res : ndarray

For NumPy >= 1.10.0 a view of `a` is always returned. For earlier

NumPy versions a view of `a` is returned only if the order of the

axes is changed, otherwise the input array is returned.

See Also

--------

moveaxis : Move array axes to new positions.

roll : Roll the elements of an array by a number of positions along a

given axis.

Examples

--------

>>> a = np.ones((3,4,5,6))

>>> np.rollaxis(a, 3, 1).shape

(3, 6, 4, 5)

>>> np.rollaxis(a, 2).shape

(5, 3, 4, 6)

>>> np.rollaxis(a, 1, 4).shape

(3, 5, 6, 4)

roots(p)

Return the roots of a polynomial with coefficients given in p.

The values in the rank-1 array `p` are coefficients of a polynomial.

If the length of `p` is n+1 then the polynomial is described by::

p[0] \* x\*\*n + p[1] \* x\*\*(n-1) + ... + p[n-1]\*x + p[n]

Parameters

----------

p : array\_like

Rank-1 array of polynomial coefficients.

Returns

-------

out : ndarray

An array containing the roots of the polynomial.

Raises

------

ValueError

When `p` cannot be converted to a rank-1 array.

See also

--------

poly : Find the coefficients of a polynomial with a given sequence

of roots.

polyval : Compute polynomial values.

polyfit : Least squares polynomial fit.

poly1d : A one-dimensional polynomial class.

Notes

-----

The algorithm relies on computing the eigenvalues of the

companion matrix [1]\_.

References

----------

.. [1] R. A. Horn & C. R. Johnson, \*Matrix Analysis\*. Cambridge, UK:

Cambridge University Press, 1999, pp. 146-7.

Examples

--------

>>> coeff = [3.2, 2, 1]

>>> np.roots(coeff)

array([-0.3125+0.46351241j, -0.3125-0.46351241j])

rot90(m, k=1, axes=(0, 1))

Rotate an array by 90 degrees in the plane specified by axes.

Rotation direction is from the first towards the second axis.

Parameters

----------

m : array\_like

Array of two or more dimensions.

k : integer

Number of times the array is rotated by 90 degrees.

axes: (2,) array\_like

The array is rotated in the plane defined by the axes.

Axes must be different.

.. versionadded:: 1.12.0

Returns

-------

y : ndarray

A rotated view of `m`.

See Also

--------

flip : Reverse the order of elements in an array along the given axis.

fliplr : Flip an array horizontally.

flipud : Flip an array vertically.

Notes

-----

rot90(m, k=1, axes=(1,0)) is the reverse of rot90(m, k=1, axes=(0,1))

rot90(m, k=1, axes=(1,0)) is equivalent to rot90(m, k=-1, axes=(0,1))

Examples

--------

>>> m = np.array([[1,2],[3,4]], int)

>>> m

array([[1, 2],

[3, 4]])

>>> np.rot90(m)

array([[2, 4],

[1, 3]])

>>> np.rot90(m, 2)

array([[4, 3],

[2, 1]])

>>> m = np.arange(8).reshape((2,2,2))

>>> np.rot90(m, 1, (1,2))

array([[[1, 3],

[0, 2]],

[[5, 7],

[4, 6]]])

round\_(a, decimals=0, out=None)

Round an array to the given number of decimals.

See Also

--------

around : equivalent function; see for details.

row\_stack = vstack(tup)

Stack arrays in sequence vertically (row wise).

This is equivalent to concatenation along the first axis after 1-D arrays

of shape `(N,)` have been reshaped to `(1,N)`. Rebuilds arrays divided by

`vsplit`.

This function makes most sense for arrays with up to 3 dimensions. For

instance, for pixel-data with a height (first axis), width (second axis),

and r/g/b channels (third axis). The functions `concatenate`, `stack` and

`block` provide more general stacking and concatenation operations.

Parameters

----------

tup : sequence of ndarrays

The arrays must have the same shape along all but the first axis.

1-D arrays must have the same length.

Returns

-------

stacked : ndarray

The array formed by stacking the given arrays, will be at least 2-D.

See Also

--------

stack : Join a sequence of arrays along a new axis.

hstack : Stack arrays in sequence horizontally (column wise).

dstack : Stack arrays in sequence depth wise (along third dimension).

concatenate : Join a sequence of arrays along an existing axis.

vsplit : Split array into a list of multiple sub-arrays vertically.

block : Assemble arrays from blocks.

Examples

--------

>>> a = np.array([1, 2, 3])

>>> b = np.array([2, 3, 4])

>>> np.vstack((a,b))

array([[1, 2, 3],

[2, 3, 4]])

>>> a = np.array([[1], [2], [3]])

>>> b = np.array([[2], [3], [4]])

>>> np.vstack((a,b))

array([[1],

[2],

[3],

[2],

[3],

[4]])

safe\_eval(source)

Protected string evaluation.

Evaluate a string containing a Python literal expression without

allowing the execution of arbitrary non-literal code.

Parameters

----------

source : str

The string to evaluate.

Returns

-------

obj : object

The result of evaluating `source`.

Raises

------

SyntaxError

If the code has invalid Python syntax, or if it contains

non-literal code.

Examples

--------

>>> np.safe\_eval('1')

1

>>> np.safe\_eval('[1, 2, 3]')

[1, 2, 3]

>>> np.safe\_eval('{"foo": ("bar", 10.0)}')

{'foo': ('bar', 10.0)}

>>> np.safe\_eval('import os')

Traceback (most recent call last):

...

SyntaxError: invalid syntax

>>> np.safe\_eval('open("/home/user/.ssh/id\_dsa").read()')

Traceback (most recent call last):

...

ValueError: malformed node or string: <\_ast.Call object at 0x...>

save(file, arr, allow\_pickle=True, fix\_imports=True)

Save an array to a binary file in NumPy ``.npy`` format.

Parameters

----------

file : file, str, or pathlib.Path

File or filename to which the data is saved. If file is a file-object,

then the filename is unchanged. If file is a string or Path, a ``.npy``

extension will be appended to the file name if it does not already

have one.

arr : array\_like

Array data to be saved.

allow\_pickle : bool, optional

Allow saving object arrays using Python pickles. Reasons for disallowing

pickles include security (loading pickled data can execute arbitrary

code) and portability (pickled objects may not be loadable on different

Python installations, for example if the stored objects require libraries

that are not available, and not all pickled data is compatible between

Python 2 and Python 3).

Default: True

fix\_imports : bool, optional

Only useful in forcing objects in object arrays on Python 3 to be

pickled in a Python 2 compatible way. If `fix\_imports` is True, pickle

will try to map the new Python 3 names to the old module names used in

Python 2, so that the pickle data stream is readable with Python 2.

See Also

--------

savez : Save several arrays into a ``.npz`` archive

savetxt, load

Notes

-----

For a description of the ``.npy`` format, see :py:mod:`numpy.lib.format`.

Examples

--------

>>> from tempfile import TemporaryFile

>>> outfile = TemporaryFile()

>>> x = np.arange(10)

>>> np.save(outfile, x)

>>> \_ = outfile.seek(0) # Only needed here to simulate closing & reopening file

>>> np.load(outfile)

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)

Save an array to a text file.

Parameters

----------

fname : filename or file handle

If the filename ends in ``.gz``, the file is automatically saved in

compressed gzip format. `loadtxt` understands gzipped files

transparently.

X : 1D or 2D array\_like

Data to be saved to a text file.

fmt : str or sequence of strs, optional

A single format (%10.5f), a sequence of formats, or a

multi-format string, e.g. 'Iteration %d -- %10.5f', in which

case `delimiter` is ignored. For complex `X`, the legal options

for `fmt` are:

\* a single specifier, `fmt='%.4e'`, resulting in numbers formatted

like `' (%s+%sj)' % (fmt, fmt)`

\* a full string specifying every real and imaginary part, e.g.

`' %.4e %+.4ej %.4e %+.4ej %.4e %+.4ej'` for 3 columns

\* a list of specifiers, one per column - in this case, the real

and imaginary part must have separate specifiers,

e.g. `['%.3e + %.3ej', '(%.15e%+.15ej)']` for 2 columns

delimiter : str, optional

String or character separating columns.

newline : str, optional

String or character separating lines.

.. versionadded:: 1.5.0

header : str, optional

String that will be written at the beginning of the file.

.. versionadded:: 1.7.0

footer : str, optional

String that will be written at the end of the file.

.. versionadded:: 1.7.0

comments : str, optional

String that will be prepended to the ``header`` and ``footer`` strings,

to mark them as comments. Default: '# ', as expected by e.g.

``numpy.loadtxt``.

.. versionadded:: 1.7.0

encoding : {None, str}, optional

Encoding used to encode the outputfile. Does not apply to output

streams. If the encoding is something other than 'bytes' or 'latin1'

you will not be able to load the file in NumPy versions < 1.14. Default

is 'latin1'.

.. versionadded:: 1.14.0

See Also

--------

save : Save an array to a binary file in NumPy ``.npy`` format

savez : Save several arrays into an uncompressed ``.npz`` archive

savez\_compressed : Save several arrays into a compressed ``.npz`` archive

Notes

-----

Further explanation of the `fmt` parameter

(``%[flag]width[.precision]specifier``):

flags:

``-`` : left justify

``+`` : Forces to precede result with + or -.

``0`` : Left pad the number with zeros instead of space (see width).

width:

Minimum number of characters to be printed. The value is not truncated

if it has more characters.

precision:

- For integer specifiers (eg. ``d,i,o,x``), the minimum number of

digits.

- For ``e, E`` and ``f`` specifiers, the number of digits to print

after the decimal point.

- For ``g`` and ``G``, the maximum number of significant digits.

- For ``s``, the maximum number of characters.

specifiers:

``c`` : character

``d`` or ``i`` : signed decimal integer

``e`` or ``E`` : scientific notation with ``e`` or ``E``.

``f`` : decimal floating point

``g,G`` : use the shorter of ``e,E`` or ``f``

``o`` : signed octal

``s`` : string of characters

``u`` : unsigned decimal integer

``x,X`` : unsigned hexadecimal integer

This explanation of ``fmt`` is not complete, for an exhaustive

specification see [1]\_.

References

----------

.. [1] `Format Specification Mini-Language

<https://docs.python.org/library/string.html#format-specification-mini-language>`\_,

Python Documentation.

Examples

--------

>>> x = y = z = np.arange(0.0,5.0,1.0)

>>> np.savetxt('test.out', x, delimiter=',') # X is an array

>>> np.savetxt('test.out', (x,y,z)) # x,y,z equal sized 1D arrays

>>> np.savetxt('test.out', x, fmt='%1.4e') # use exponential notation

savez(file, \*args, \*\*kwds)

Save several arrays into a single file in uncompressed ``.npz`` format.

If arguments are passed in with no keywords, the corresponding variable

names, in the ``.npz`` file, are 'arr\_0', 'arr\_1', etc. If keyword

arguments are given, the corresponding variable names, in the ``.npz``

file will match the keyword names.

Parameters

----------

file : str or file

Either the file name (string) or an open file (file-like object)

where the data will be saved. If file is a string or a Path, the

``.npz`` extension will be appended to the file name if it is not

already there.

args : Arguments, optional

Arrays to save to the file. Since it is not possible for Python to

know the names of the arrays outside `savez`, the arrays will be saved

with names "arr\_0", "arr\_1", and so on. These arguments can be any

expression.

kwds : Keyword arguments, optional

Arrays to save to the file. Arrays will be saved in the file with the

keyword names.

Returns

-------

None

See Also

--------

save : Save a single array to a binary file in NumPy format.

savetxt : Save an array to a file as plain text.

savez\_compressed : Save several arrays into a compressed ``.npz`` archive

Notes

-----

The ``.npz`` file format is a zipped archive of files named after the

variables they contain. The archive is not compressed and each file

in the archive contains one variable in ``.npy`` format. For a

description of the ``.npy`` format, see :py:mod:`numpy.lib.format`.

When opening the saved ``.npz`` file with `load` a `NpzFile` object is

returned. This is a dictionary-like object which can be queried for

its list of arrays (with the ``.files`` attribute), and for the arrays

themselves.

Examples

--------

>>> from tempfile import TemporaryFile

>>> outfile = TemporaryFile()

>>> x = np.arange(10)

>>> y = np.sin(x)

Using `savez` with \\*args, the arrays are saved with default names.

>>> np.savez(outfile, x, y)

>>> \_ = outfile.seek(0) # Only needed here to simulate closing & reopening file

>>> npzfile = np.load(outfile)

>>> npzfile.files

['arr\_0', 'arr\_1']

>>> npzfile['arr\_0']

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

Using `savez` with \\*\*kwds, the arrays are saved with the keyword names.

>>> outfile = TemporaryFile()

>>> np.savez(outfile, x=x, y=y)

>>> \_ = outfile.seek(0)

>>> npzfile = np.load(outfile)

>>> sorted(npzfile.files)

['x', 'y']

>>> npzfile['x']

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

savez\_compressed(file, \*args, \*\*kwds)

Save several arrays into a single file in compressed ``.npz`` format.

If keyword arguments are given, then filenames are taken from the keywords.

If arguments are passed in with no keywords, then stored file names are

arr\_0, arr\_1, etc.

Parameters

----------

file : str or file

Either the file name (string) or an open file (file-like object)

where the data will be saved. If file is a string or a Path, the

``.npz`` extension will be appended to the file name if it is not

already there.

args : Arguments, optional

Arrays to save to the file. Since it is not possible for Python to

know the names of the arrays outside `savez`, the arrays will be saved

with names "arr\_0", "arr\_1", and so on. These arguments can be any

expression.

kwds : Keyword arguments, optional

Arrays to save to the file. Arrays will be saved in the file with the

keyword names.

Returns

-------

None

See Also

--------

numpy.save : Save a single array to a binary file in NumPy format.

numpy.savetxt : Save an array to a file as plain text.

numpy.savez : Save several arrays into an uncompressed ``.npz`` file format

numpy.load : Load the files created by savez\_compressed.

Notes

-----

The ``.npz`` file format is a zipped archive of files named after the

variables they contain. The archive is compressed with

``zipfile.ZIP\_DEFLATED`` and each file in the archive contains one variable

in ``.npy`` format. For a description of the ``.npy`` format, see

:py:mod:`numpy.lib.format`.

When opening the saved ``.npz`` file with `load` a `NpzFile` object is

returned. This is a dictionary-like object which can be queried for

its list of arrays (with the ``.files`` attribute), and for the arrays

themselves.

Examples

--------

>>> test\_array = np.random.rand(3, 2)

>>> test\_vector = np.random.rand(4)

>>> np.savez\_compressed('/tmp/123', a=test\_array, b=test\_vector)

>>> loaded = np.load('/tmp/123.npz')

>>> print(np.array\_equal(test\_array, loaded['a']))

True

>>> print(np.array\_equal(test\_vector, loaded['b']))

True

sctype2char(sctype)

Return the string representation of a scalar dtype.

Parameters

----------

sctype : scalar dtype or object

If a scalar dtype, the corresponding string character is

returned. If an object, `sctype2char` tries to infer its scalar type

and then return the corresponding string character.

Returns

-------

typechar : str

The string character corresponding to the scalar type.

Raises

------

ValueError

If `sctype` is an object for which the type can not be inferred.

See Also

--------

obj2sctype, issctype, issubsctype, mintypecode

Examples

--------

>>> for sctype in [np.int32, np.double, np.complex, np.string\_, np.ndarray]:

... print(np.sctype2char(sctype))

l # may vary

d

D

S

O

>>> x = np.array([1., 2-1.j])

>>> np.sctype2char(x)

'D'

>>> np.sctype2char(list)

'O'

searchsorted(a, v, side='left', sorter=None)

Find indices where elements should be inserted to maintain order.

Find the indices into a sorted array `a` such that, if the

corresponding elements in `v` were inserted before the indices, the

order of `a` would be preserved.

Assuming that `a` is sorted:

====== ============================

`side` returned index `i` satisfies

====== ============================

left ``a[i-1] < v <= a[i]``

right ``a[i-1] <= v < a[i]``

====== ============================

Parameters

----------

a : 1-D array\_like

Input array. If `sorter` is None, then it must be sorted in

ascending order, otherwise `sorter` must be an array of indices

that sort it.

v : array\_like

Values to insert into `a`.

side : {'left', 'right'}, optional

If 'left', the index of the first suitable location found is given.

If 'right', return the last such index. If there is no suitable

index, return either 0 or N (where N is the length of `a`).

sorter : 1-D array\_like, optional

Optional array of integer indices that sort array a into ascending

order. They are typically the result of argsort.

.. versionadded:: 1.7.0

Returns

-------

indices : array of ints

Array of insertion points with the same shape as `v`.

See Also

--------

sort : Return a sorted copy of an array.

histogram : Produce histogram from 1-D data.

Notes

-----

Binary search is used to find the required insertion points.

As of NumPy 1.4.0 `searchsorted` works with real/complex arrays containing

`nan` values. The enhanced sort order is documented in `sort`.

This function uses the same algorithm as the builtin python `bisect.bisect\_left`

(``side='left'``) and `bisect.bisect\_right` (``side='right'``) functions,

which is also vectorized in the `v` argument.

Examples

--------

>>> np.searchsorted([1,2,3,4,5], 3)

2

>>> np.searchsorted([1,2,3,4,5], 3, side='right')

3

>>> np.searchsorted([1,2,3,4,5], [-10, 10, 2, 3])

array([0, 5, 1, 2])

select(condlist, choicelist, default=0)

Return an array drawn from elements in choicelist, depending on conditions.

Parameters

----------

condlist : list of bool ndarrays

The list of conditions which determine from which array in `choicelist`

the output elements are taken. When multiple conditions are satisfied,

the first one encountered in `condlist` is used.

choicelist : list of ndarrays

The list of arrays from which the output elements are taken. It has

to be of the same length as `condlist`.

default : scalar, optional

The element inserted in `output` when all conditions evaluate to False.

Returns

-------

output : ndarray

The output at position m is the m-th element of the array in

`choicelist` where the m-th element of the corresponding array in

`condlist` is True.

See Also

--------

where : Return elements from one of two arrays depending on condition.

take, choose, compress, diag, diagonal

Examples

--------

>>> x = np.arange(10)

>>> condlist = [x<3, x>5]

>>> choicelist = [x, x\*\*2]

>>> np.select(condlist, choicelist)

array([ 0, 1, 2, ..., 49, 64, 81])

set\_numeric\_ops(...)

set\_numeric\_ops(op1=func1, op2=func2, ...)

Set numerical operators for array objects.

.. deprecated:: 1.16

For the general case, use :c:func:`PyUFunc\_ReplaceLoopBySignature`.

For ndarray subclasses, define the ``\_\_array\_ufunc\_\_`` method and

override the relevant ufunc.

Parameters

----------

op1, op2, ... : callable

Each ``op = func`` pair describes an operator to be replaced.

For example, ``add = lambda x, y: np.add(x, y) % 5`` would replace

addition by modulus 5 addition.

Returns

-------

saved\_ops : list of callables

A list of all operators, stored before making replacements.

Notes

-----

.. WARNING::

Use with care! Incorrect usage may lead to memory errors.

A function replacing an operator cannot make use of that operator.

For example, when replacing add, you may not use ``+``. Instead,

directly call ufuncs.

Examples

--------

>>> def add\_mod5(x, y):

... return np.add(x, y) % 5

...

>>> old\_funcs = np.set\_numeric\_ops(add=add\_mod5)

>>> x = np.arange(12).reshape((3, 4))

>>> x + x

array([[0, 2, 4, 1],

[3, 0, 2, 4],

[1, 3, 0, 2]])

>>> ignore = np.set\_numeric\_ops(\*\*old\_funcs) # restore operators

set\_printoptions(precision=None, threshold=None, edgeitems=None, linewidth=None, suppress=None, nanstr=None, infstr=None, formatter=None, sign=None, floatmode=None, \*\*kwarg)

Set printing options.

These options determine the way floating point numbers, arrays and

other NumPy objects are displayed.

Parameters

----------

precision : int or None, optional

Number of digits of precision for floating point output (default 8).

May be `None` if `floatmode` is not `fixed`, to print as many digits as

necessary to uniquely specify the value.

threshold : int, optional

Total number of array elements which trigger summarization

rather than full repr (default 1000).

edgeitems : int, optional

Number of array items in summary at beginning and end of

each dimension (default 3).

linewidth : int, optional

The number of characters per line for the purpose of inserting

line breaks (default 75).

suppress : bool, optional

If True, always print floating point numbers using fixed point

notation, in which case numbers equal to zero in the current precision

will print as zero. If False, then scientific notation is used when

absolute value of the smallest number is < 1e-4 or the ratio of the

maximum absolute value to the minimum is > 1e3. The default is False.

nanstr : str, optional

String representation of floating point not-a-number (default nan).

infstr : str, optional

String representation of floating point infinity (default inf).

sign : string, either '-', '+', or ' ', optional

Controls printing of the sign of floating-point types. If '+', always

print the sign of positive values. If ' ', always prints a space

(whitespace character) in the sign position of positive values. If

'-', omit the sign character of positive values. (default '-')

formatter : dict of callables, optional

If not None, the keys should indicate the type(s) that the respective

formatting function applies to. Callables should return a string.

Types that are not specified (by their corresponding keys) are handled

by the default formatters. Individual types for which a formatter

can be set are:

- 'bool'

- 'int'

- 'timedelta' : a `numpy.timedelta64`

- 'datetime' : a `numpy.datetime64`

- 'float'

- 'longfloat' : 128-bit floats

- 'complexfloat'

- 'longcomplexfloat' : composed of two 128-bit floats

- 'numpystr' : types `numpy.string\_` and `numpy.unicode\_`

- 'object' : `np.object\_` arrays

- 'str' : all other strings

Other keys that can be used to set a group of types at once are:

- 'all' : sets all types

- 'int\_kind' : sets 'int'

- 'float\_kind' : sets 'float' and 'longfloat'

- 'complex\_kind' : sets 'complexfloat' and 'longcomplexfloat'

- 'str\_kind' : sets 'str' and 'numpystr'

floatmode : str, optional

Controls the interpretation of the `precision` option for

floating-point types. Can take the following values

(default maxprec\_equal):

\* 'fixed': Always print exactly `precision` fractional digits,

even if this would print more or fewer digits than

necessary to specify the value uniquely.

\* 'unique': Print the minimum number of fractional digits necessary

to represent each value uniquely. Different elements may

have a different number of digits. The value of the

`precision` option is ignored.

\* 'maxprec': Print at most `precision` fractional digits, but if

an element can be uniquely represented with fewer digits

only print it with that many.

\* 'maxprec\_equal': Print at most `precision` fractional digits,

but if every element in the array can be uniquely

represented with an equal number of fewer digits, use that

many digits for all elements.

legacy : string or `False`, optional

If set to the string `'1.13'` enables 1.13 legacy printing mode. This

approximates numpy 1.13 print output by including a space in the sign

position of floats and different behavior for 0d arrays. If set to

`False`, disables legacy mode. Unrecognized strings will be ignored

with a warning for forward compatibility.

.. versionadded:: 1.14.0

See Also

--------

get\_printoptions, set\_string\_function, array2string

Notes

-----

`formatter` is always reset with a call to `set\_printoptions`.

Examples

--------

Floating point precision can be set:

>>> np.set\_printoptions(precision=4)

>>> np.array([1.123456789])

[1.1235]

Long arrays can be summarised:

>>> np.set\_printoptions(threshold=5)

>>> np.arange(10)

array([0, 1, 2, ..., 7, 8, 9])

Small results can be suppressed:

>>> eps = np.finfo(float).eps

>>> x = np.arange(4.)

>>> x\*\*2 - (x + eps)\*\*2

array([-4.9304e-32, -4.4409e-16, 0.0000e+00, 0.0000e+00])

>>> np.set\_printoptions(suppress=True)

>>> x\*\*2 - (x + eps)\*\*2

array([-0., -0., 0., 0.])

A custom formatter can be used to display array elements as desired:

>>> np.set\_printoptions(formatter={'all':lambda x: 'int: '+str(-x)})

>>> x = np.arange(3)

>>> x

array([int: 0, int: -1, int: -2])

>>> np.set\_printoptions() # formatter gets reset

>>> x

array([0, 1, 2])

To put back the default options, you can use:

>>> np.set\_printoptions(edgeitems=3,infstr='inf',

... linewidth=75, nanstr='nan', precision=8,

... suppress=False, threshold=1000, formatter=None)

set\_string\_function(f, repr=True)

Set a Python function to be used when pretty printing arrays.

Parameters

----------

f : function or None

Function to be used to pretty print arrays. The function should expect

a single array argument and return a string of the representation of

the array. If None, the function is reset to the default NumPy function

to print arrays.

repr : bool, optional

If True (default), the function for pretty printing (``\_\_repr\_\_``)

is set, if False the function that returns the default string

representation (``\_\_str\_\_``) is set.

See Also

--------

set\_printoptions, get\_printoptions

Examples

--------

>>> def pprint(arr):

... return 'HA! - What are you going to do now?'

...

>>> np.set\_string\_function(pprint)

>>> a = np.arange(10)

>>> a

HA! - What are you going to do now?

>>> \_ = a

>>> # [0 1 2 3 4 5 6 7 8 9]

We can reset the function to the default:

>>> np.set\_string\_function(None)

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

`repr` affects either pretty printing or normal string representation.

Note that ``\_\_repr\_\_`` is still affected by setting ``\_\_str\_\_``

because the width of each array element in the returned string becomes

equal to the length of the result of ``\_\_str\_\_()``.

>>> x = np.arange(4)

>>> np.set\_string\_function(lambda x:'random', repr=False)

>>> x.\_\_str\_\_()

'random'

>>> x.\_\_repr\_\_()

'array([0, 1, 2, 3])'

setbufsize(size)

Set the size of the buffer used in ufuncs.

Parameters

----------

size : int

Size of buffer.

setdiff1d(ar1, ar2, assume\_unique=False)

Find the set difference of two arrays.

Return the unique values in `ar1` that are not in `ar2`.

Parameters

----------

ar1 : array\_like

Input array.

ar2 : array\_like

Input comparison array.

assume\_unique : bool

If True, the input arrays are both assumed to be unique, which

can speed up the calculation. Default is False.

Returns

-------

setdiff1d : ndarray

1D array of values in `ar1` that are not in `ar2`. The result

is sorted when `assume\_unique=False`, but otherwise only sorted

if the input is sorted.

See Also

--------

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Examples

--------

>>> a = np.array([1, 2, 3, 2, 4, 1])

>>> b = np.array([3, 4, 5, 6])

>>> np.setdiff1d(a, b)

array([1, 2])

seterr(all=None, divide=None, over=None, under=None, invalid=None)

Set how floating-point errors are handled.

Note that operations on integer scalar types (such as `int16`) are

handled like floating point, and are affected by these settings.

Parameters

----------

all : {'ignore', 'warn', 'raise', 'call', 'print', 'log'}, optional

Set treatment for all types of floating-point errors at once:

- ignore: Take no action when the exception occurs.

- warn: Print a `RuntimeWarning` (via the Python `warnings` module).

- raise: Raise a `FloatingPointError`.

- call: Call a function specified using the `seterrcall` function.

- print: Print a warning directly to ``stdout``.

- log: Record error in a Log object specified by `seterrcall`.

The default is not to change the current behavior.

divide : {'ignore', 'warn', 'raise', 'call', 'print', 'log'}, optional

Treatment for division by zero.

over : {'ignore', 'warn', 'raise', 'call', 'print', 'log'}, optional

Treatment for floating-point overflow.

under : {'ignore', 'warn', 'raise', 'call', 'print', 'log'}, optional

Treatment for floating-point underflow.

invalid : {'ignore', 'warn', 'raise', 'call', 'print', 'log'}, optional

Treatment for invalid floating-point operation.

Returns

-------

old\_settings : dict

Dictionary containing the old settings.

See also

--------

seterrcall : Set a callback function for the 'call' mode.

geterr, geterrcall, errstate

Notes

-----

The floating-point exceptions are defined in the IEEE 754 standard [1]\_:

- Division by zero: infinite result obtained from finite numbers.

- Overflow: result too large to be expressed.

- Underflow: result so close to zero that some precision

was lost.

- Invalid operation: result is not an expressible number, typically

indicates that a NaN was produced.

.. [1] https://en.wikipedia.org/wiki/IEEE\_754

Examples

--------

>>> old\_settings = np.seterr(all='ignore') #seterr to known value

>>> np.seterr(over='raise')

{'divide': 'ignore', 'over': 'ignore', 'under': 'ignore', 'invalid': 'ignore'}

>>> np.seterr(\*\*old\_settings) # reset to default

{'divide': 'ignore', 'over': 'raise', 'under': 'ignore', 'invalid': 'ignore'}

>>> np.int16(32000) \* np.int16(3)

30464

>>> old\_settings = np.seterr(all='warn', over='raise')

>>> np.int16(32000) \* np.int16(3)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

FloatingPointError: overflow encountered in short\_scalars

>>> from collections import OrderedDict

>>> old\_settings = np.seterr(all='print')

>>> OrderedDict(np.geterr())

OrderedDict([('divide', 'print'), ('over', 'print'), ('under', 'print'), ('invalid', 'print')])

>>> np.int16(32000) \* np.int16(3)

30464

seterrcall(func)

Set the floating-point error callback function or log object.

There are two ways to capture floating-point error messages. The first

is to set the error-handler to 'call', using `seterr`. Then, set

the function to call using this function.

The second is to set the error-handler to 'log', using `seterr`.

Floating-point errors then trigger a call to the 'write' method of

the provided object.

Parameters

----------

func : callable f(err, flag) or object with write method

Function to call upon floating-point errors ('call'-mode) or

object whose 'write' method is used to log such message ('log'-mode).

The call function takes two arguments. The first is a string describing

the type of error (such as "divide by zero", "overflow", "underflow",

or "invalid value"), and the second is the status flag. The flag is a

byte, whose four least-significant bits indicate the type of error, one

of "divide", "over", "under", "invalid"::

[0 0 0 0 divide over under invalid]

In other words, ``flags = divide + 2\*over + 4\*under + 8\*invalid``.

If an object is provided, its write method should take one argument,

a string.

Returns

-------

h : callable, log instance or None

The old error handler.

See Also

--------

seterr, geterr, geterrcall

Examples

--------

Callback upon error:

>>> def err\_handler(type, flag):

... print("Floating point error (%s), with flag %s" % (type, flag))

...

>>> saved\_handler = np.seterrcall(err\_handler)

>>> save\_err = np.seterr(all='call')

>>> from collections import OrderedDict

>>> np.array([1, 2, 3]) / 0.0

Floating point error (divide by zero), with flag 1

array([inf, inf, inf])

>>> np.seterrcall(saved\_handler)

<function err\_handler at 0x...>

>>> OrderedDict(sorted(np.seterr(\*\*save\_err).items()))

OrderedDict([('divide', 'call'), ('invalid', 'call'), ('over', 'call'), ('under', 'call')])

Log error message:

>>> class Log(object):

... def write(self, msg):

... print("LOG: %s" % msg)

...

>>> log = Log()

>>> saved\_handler = np.seterrcall(log)

>>> save\_err = np.seterr(all='log')

>>> np.array([1, 2, 3]) / 0.0

LOG: Warning: divide by zero encountered in true\_divide

array([inf, inf, inf])

>>> np.seterrcall(saved\_handler)

<numpy.core.numeric.Log object at 0x...>

>>> OrderedDict(sorted(np.seterr(\*\*save\_err).items()))

OrderedDict([('divide', 'log'), ('invalid', 'log'), ('over', 'log'), ('under', 'log')])

seterrobj(...)

seterrobj(errobj)

Set the object that defines floating-point error handling.

The error object contains all information that defines the error handling

behavior in NumPy. `seterrobj` is used internally by the other

functions that set error handling behavior (`seterr`, `seterrcall`).

Parameters

----------

errobj : list

The error object, a list containing three elements:

[internal numpy buffer size, error mask, error callback function].

The error mask is a single integer that holds the treatment information

on all four floating point errors. The information for each error type

is contained in three bits of the integer. If we print it in base 8, we

can see what treatment is set for "invalid", "under", "over", and

"divide" (in that order). The printed string can be interpreted with

\* 0 : 'ignore'

\* 1 : 'warn'

\* 2 : 'raise'

\* 3 : 'call'

\* 4 : 'print'

\* 5 : 'log'

See Also

--------

geterrobj, seterr, geterr, seterrcall, geterrcall

getbufsize, setbufsize

Notes

-----

For complete documentation of the types of floating-point exceptions and

treatment options, see `seterr`.

Examples

--------

>>> old\_errobj = np.geterrobj() # first get the defaults

>>> old\_errobj

[8192, 521, None]

>>> def err\_handler(type, flag):

... print("Floating point error (%s), with flag %s" % (type, flag))

...

>>> new\_errobj = [20000, 12, err\_handler]

>>> np.seterrobj(new\_errobj)

>>> np.base\_repr(12, 8) # int for divide=4 ('print') and over=1 ('warn')

'14'

>>> np.geterr()

{'over': 'warn', 'divide': 'print', 'invalid': 'ignore', 'under': 'ignore'}

>>> np.geterrcall() is err\_handler

True

setxor1d(ar1, ar2, assume\_unique=False)

Find the set exclusive-or of two arrays.

Return the sorted, unique values that are in only one (not both) of the

input arrays.

Parameters

----------

ar1, ar2 : array\_like

Input arrays.

assume\_unique : bool

If True, the input arrays are both assumed to be unique, which

can speed up the calculation. Default is False.

Returns

-------

setxor1d : ndarray

Sorted 1D array of unique values that are in only one of the input

arrays.

Examples

--------

>>> a = np.array([1, 2, 3, 2, 4])

>>> b = np.array([2, 3, 5, 7, 5])

>>> np.setxor1d(a,b)

array([1, 4, 5, 7])

shape(a)

Return the shape of an array.

Parameters

----------

a : array\_like

Input array.

Returns

-------

shape : tuple of ints

The elements of the shape tuple give the lengths of the

corresponding array dimensions.

See Also

--------

alen

ndarray.shape : Equivalent array method.

Examples

--------

>>> np.shape(np.eye(3))

(3, 3)

>>> np.shape([[1, 2]])

(1, 2)

>>> np.shape([0])

(1,)

>>> np.shape(0)

()

>>> a = np.array([(1, 2), (3, 4)], dtype=[('x', 'i4'), ('y', 'i4')])

>>> np.shape(a)

(2,)

>>> a.shape

(2,)

shares\_memory(...)

shares\_memory(a, b, max\_work=None)

Determine if two arrays share memory

Parameters

----------

a, b : ndarray

Input arrays

max\_work : int, optional

Effort to spend on solving the overlap problem (maximum number

of candidate solutions to consider). The following special

values are recognized:

max\_work=MAY\_SHARE\_EXACT (default)

The problem is solved exactly. In this case, the function returns

True only if there is an element shared between the arrays.

max\_work=MAY\_SHARE\_BOUNDS

Only the memory bounds of a and b are checked.

Raises

------

numpy.TooHardError

Exceeded max\_work.

Returns

-------

out : bool

See Also

--------

may\_share\_memory

Examples

--------

>>> np.may\_share\_memory(np.array([1,2]), np.array([5,8,9]))

False

show\_config = show()

sinc(x)

Return the sinc function.

The sinc function is :math:`\sin(\pi x)/(\pi x)`.

Parameters

----------

x : ndarray

Array (possibly multi-dimensional) of values for which to to

calculate ``sinc(x)``.

Returns

-------

out : ndarray

``sinc(x)``, which has the same shape as the input.

Notes

-----

``sinc(0)`` is the limit value 1.

The name sinc is short for "sine cardinal" or "sinus cardinalis".

The sinc function is used in various signal processing applications,

including in anti-aliasing, in the construction of a Lanczos resampling

filter, and in interpolation.

For bandlimited interpolation of discrete-time signals, the ideal

interpolation kernel is proportional to the sinc function.

References

----------

.. [1] Weisstein, Eric W. "Sinc Function." From MathWorld--A Wolfram Web

Resource. http://mathworld.wolfram.com/SincFunction.html

.. [2] Wikipedia, "Sinc function",

https://en.wikipedia.org/wiki/Sinc\_function

Examples

--------

>>> import matplotlib.pyplot as plt

>>> x = np.linspace(-4, 4, 41)

>>> np.sinc(x)

array([-3.89804309e-17, -4.92362781e-02, -8.40918587e-02, # may vary

-8.90384387e-02, -5.84680802e-02, 3.89804309e-17,

6.68206631e-02, 1.16434881e-01, 1.26137788e-01,

8.50444803e-02, -3.89804309e-17, -1.03943254e-01,

-1.89206682e-01, -2.16236208e-01, -1.55914881e-01,

3.89804309e-17, 2.33872321e-01, 5.04551152e-01,

7.56826729e-01, 9.35489284e-01, 1.00000000e+00,

9.35489284e-01, 7.56826729e-01, 5.04551152e-01,

2.33872321e-01, 3.89804309e-17, -1.55914881e-01,

-2.16236208e-01, -1.89206682e-01, -1.03943254e-01,

-3.89804309e-17, 8.50444803e-02, 1.26137788e-01,

1.16434881e-01, 6.68206631e-02, 3.89804309e-17,

-5.84680802e-02, -8.90384387e-02, -8.40918587e-02,

-4.92362781e-02, -3.89804309e-17])

>>> plt.plot(x, np.sinc(x))

[<matplotlib.lines.Line2D object at 0x...>]

>>> plt.title("Sinc Function")

Text(0.5, 1.0, 'Sinc Function')

>>> plt.ylabel("Amplitude")

Text(0, 0.5, 'Amplitude')

>>> plt.xlabel("X")

Text(0.5, 0, 'X')

>>> plt.show()

It works in 2-D as well:

>>> x = np.linspace(-4, 4, 401)

>>> xx = np.outer(x, x)

>>> plt.imshow(np.sinc(xx))

<matplotlib.image.AxesImage object at 0x...>

size(a, axis=None)

Return the number of elements along a given axis.

Parameters

----------

a : array\_like

Input data.

axis : int, optional

Axis along which the elements are counted. By default, give

the total number of elements.

Returns

-------

element\_count : int

Number of elements along the specified axis.

See Also

--------

shape : dimensions of array

ndarray.shape : dimensions of array

ndarray.size : number of elements in array

Examples

--------

>>> a = np.array([[1,2,3],[4,5,6]])

>>> np.size(a)

6

>>> np.size(a,1)

3

>>> np.size(a,0)

2

sometrue(\*args, \*\*kwargs)

Check whether some values are true.

Refer to `any` for full documentation.

See Also

--------

any : equivalent function; see for details.

sort(a, axis=-1, kind=None, order=None)

Return a sorted copy of an array.

Parameters

----------

a : array\_like

Array to be sorted.

axis : int or None, optional

Axis along which to sort. If None, the array is flattened before

sorting. The default is -1, which sorts along the last axis.

kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional

Sorting algorithm. The default is 'quicksort'. Note that both 'stable'

and 'mergesort' use timsort or radix sort under the covers and, in general,

the actual implementation will vary with data type. The 'mergesort' option

is retained for backwards compatibility.

.. versionchanged:: 1.15.0.

The 'stable' option was added.

order : str or list of str, optional

When `a` is an array with fields defined, this argument specifies

which fields to compare first, second, etc. A single field can

be specified as a string, and not all fields need be specified,

but unspecified fields will still be used, in the order in which

they come up in the dtype, to break ties.

Returns

-------

sorted\_array : ndarray

Array of the same type and shape as `a`.

See Also

--------

ndarray.sort : Method to sort an array in-place.

argsort : Indirect sort.

lexsort : Indirect stable sort on multiple keys.

searchsorted : Find elements in a sorted array.

partition : Partial sort.

Notes

-----

The various sorting algorithms are characterized by their average speed,

worst case performance, work space size, and whether they are stable. A

stable sort keeps items with the same key in the same relative

order. The four algorithms implemented in NumPy have the following

properties:

=========== ======= ============= ============ ========

kind speed worst case work space stable

=========== ======= ============= ============ ========

'quicksort' 1 O(n^2) 0 no

'heapsort' 3 O(n\*log(n)) 0 no

'mergesort' 2 O(n\*log(n)) ~n/2 yes

'timsort' 2 O(n\*log(n)) ~n/2 yes

=========== ======= ============= ============ ========

.. note:: The datatype determines which of 'mergesort' or 'timsort'

is actually used, even if 'mergesort' is specified. User selection

at a finer scale is not currently available.

All the sort algorithms make temporary copies of the data when

sorting along any but the last axis. Consequently, sorting along

the last axis is faster and uses less space than sorting along

any other axis.

The sort order for complex numbers is lexicographic. If both the real

and imaginary parts are non-nan then the order is determined by the

real parts except when they are equal, in which case the order is

determined by the imaginary parts.

Previous to numpy 1.4.0 sorting real and complex arrays containing nan

values led to undefined behaviour. In numpy versions >= 1.4.0 nan

values are sorted to the end. The extended sort order is:

\* Real: [R, nan]

\* Complex: [R + Rj, R + nanj, nan + Rj, nan + nanj]

where R is a non-nan real value. Complex values with the same nan

placements are sorted according to the non-nan part if it exists.

Non-nan values are sorted as before.

.. versionadded:: 1.12.0

quicksort has been changed to an introsort which will switch

heapsort when it does not make enough progress. This makes its

worst case O(n\*log(n)).

'stable' automatically choses the best stable sorting algorithm

for the data type being sorted. It, along with 'mergesort' is

currently mapped to timsort or radix sort depending on the

data type. API forward compatibility currently limits the

ability to select the implementation and it is hardwired for the different

data types.

.. versionadded:: 1.17.0

Timsort is added for better performance on already or nearly

sorted data. On random data timsort is almost identical to

mergesort. It is now used for stable sort while quicksort is still the

default sort if none is chosen. For details of timsort, refer to

`CPython listsort.txt <https://github.com/python/cpython/blob/3.7/Objects/listsort.txt>`\_.

'mergesort' and 'stable' are mapped to radix sort for integer data types. Radix sort is an

O(n) sort instead of O(n log n).

Examples

--------

>>> a = np.array([[1,4],[3,1]])

>>> np.sort(a) # sort along the last axis

array([[1, 4],

[1, 3]])

>>> np.sort(a, axis=None) # sort the flattened array

array([1, 1, 3, 4])

>>> np.sort(a, axis=0) # sort along the first axis

array([[1, 1],

[3, 4]])

Use the `order` keyword to specify a field to use when sorting a

structured array:

>>> dtype = [('name', 'S10'), ('height', float), ('age', int)]

>>> values = [('Arthur', 1.8, 41), ('Lancelot', 1.9, 38),

... ('Galahad', 1.7, 38)]

>>> a = np.array(values, dtype=dtype) # create a structured array

>>> np.sort(a, order='height') # doctest: +SKIP

array([('Galahad', 1.7, 38), ('Arthur', 1.8, 41),

('Lancelot', 1.8999999999999999, 38)],

dtype=[('name', '|S10'), ('height', '<f8'), ('age', '<i4')])

Sort by age, then height if ages are equal:

>>> np.sort(a, order=['age', 'height']) # doctest: +SKIP

array([('Galahad', 1.7, 38), ('Lancelot', 1.8999999999999999, 38),

('Arthur', 1.8, 41)],

dtype=[('name', '|S10'), ('height', '<f8'), ('age', '<i4')])

sort\_complex(a)

Sort a complex array using the real part first, then the imaginary part.

Parameters

----------

a : array\_like

Input array

Returns

-------

out : complex ndarray

Always returns a sorted complex array.

Examples

--------

>>> np.sort\_complex([5, 3, 6, 2, 1])

array([1.+0.j, 2.+0.j, 3.+0.j, 5.+0.j, 6.+0.j])

>>> np.sort\_complex([1 + 2j, 2 - 1j, 3 - 2j, 3 - 3j, 3 + 5j])

array([1.+2.j, 2.-1.j, 3.-3.j, 3.-2.j, 3.+5.j])

source(object, output=<idlelib.run.PseudoOutputFile object at 0x03FD4690>)

Print or write to a file the source code for a NumPy object.

The source code is only returned for objects written in Python. Many

functions and classes are defined in C and will therefore not return

useful information.

Parameters

----------

object : numpy object

Input object. This can be any object (function, class, module,

...).

output : file object, optional

If `output` not supplied then source code is printed to screen

(sys.stdout). File object must be created with either write 'w' or

append 'a' modes.

See Also

--------

lookfor, info

Examples

--------

>>> np.source(np.interp) #doctest: +SKIP

In file: /usr/lib/python2.6/dist-packages/numpy/lib/function\_base.py

def interp(x, xp, fp, left=None, right=None):

""".... (full docstring printed)"""

if isinstance(x, (float, int, number)):

return compiled\_interp([x], xp, fp, left, right).item()

else:

return compiled\_interp(x, xp, fp, left, right)

The source code is only returned for objects written in Python.

>>> np.source(np.array) #doctest: +SKIP

Not available for this object.

split(ary, indices\_or\_sections, axis=0)

Split an array into multiple sub-arrays.

Parameters

----------

ary : ndarray

Array to be divided into sub-arrays.

indices\_or\_sections : int or 1-D array

If `indices\_or\_sections` is an integer, N, the array will be divided

into N equal arrays along `axis`. If such a split is not possible,

an error is raised.

If `indices\_or\_sections` is a 1-D array of sorted integers, the entries

indicate where along `axis` the array is split. For example,

``[2, 3]`` would, for ``axis=0``, result in

- ary[:2]

- ary[2:3]

- ary[3:]

If an index exceeds the dimension of the array along `axis`,

an empty sub-array is returned correspondingly.

axis : int, optional

The axis along which to split, default is 0.

Returns

-------

sub-arrays : list of ndarrays

A list of sub-arrays.

Raises

------

ValueError

If `indices\_or\_sections` is given as an integer, but

a split does not result in equal division.

See Also

--------

array\_split : Split an array into multiple sub-arrays of equal or

near-equal size. Does not raise an exception if

an equal division cannot be made.

hsplit : Split array into multiple sub-arrays horizontally (column-wise).

vsplit : Split array into multiple sub-arrays vertically (row wise).

dsplit : Split array into multiple sub-arrays along the 3rd axis (depth).

concatenate : Join a sequence of arrays along an existing axis.

stack : Join a sequence of arrays along a new axis.

hstack : Stack arrays in sequence horizontally (column wise).

vstack : Stack arrays in sequence vertically (row wise).

dstack : Stack arrays in sequence depth wise (along third dimension).

Examples

--------

>>> x = np.arange(9.0)

>>> np.split(x, 3)

[array([0., 1., 2.]), array([3., 4., 5.]), array([6., 7., 8.])]

>>> x = np.arange(8.0)

>>> np.split(x, [3, 5, 6, 10])

[array([0., 1., 2.]),

array([3., 4.]),

array([5.]),

array([6., 7.]),

array([], dtype=float64)]

squeeze(a, axis=None)

Remove single-dimensional entries from the shape of an array.

Parameters

----------

a : array\_like

Input data.

axis : None or int or tuple of ints, optional

.. versionadded:: 1.7.0

Selects a subset of the single-dimensional entries in the

shape. If an axis is selected with shape entry greater than

one, an error is raised.

Returns

-------

squeezed : ndarray

The input array, but with all or a subset of the

dimensions of length 1 removed. This is always `a` itself

or a view into `a`.

Raises

------

ValueError

If `axis` is not `None`, and an axis being squeezed is not of length 1

See Also

--------

expand\_dims : The inverse operation, adding singleton dimensions

reshape : Insert, remove, and combine dimensions, and resize existing ones

Examples

--------

>>> x = np.array([[[0], [1], [2]]])

>>> x.shape

(1, 3, 1)

>>> np.squeeze(x).shape

(3,)

>>> np.squeeze(x, axis=0).shape

(3, 1)

>>> np.squeeze(x, axis=1).shape

Traceback (most recent call last):

...

ValueError: cannot select an axis to squeeze out which has size not equal to one

>>> np.squeeze(x, axis=2).shape

(1, 3)

stack(arrays, axis=0, out=None)

Join a sequence of arrays along a new axis.

The ``axis`` parameter specifies the index of the new axis in the

dimensions of the result. For example, if ``axis=0`` it will be the first

dimension and if ``axis=-1`` it will be the last dimension.

.. versionadded:: 1.10.0

Parameters

----------

arrays : sequence of array\_like

Each array must have the same shape.

axis : int, optional

The axis in the result array along which the input arrays are stacked.

out : ndarray, optional

If provided, the destination to place the result. The shape must be

correct, matching that of what stack would have returned if no

out argument were specified.

Returns

-------

stacked : ndarray

The stacked array has one more dimension than the input arrays.

See Also

--------

concatenate : Join a sequence of arrays along an existing axis.

split : Split array into a list of multiple sub-arrays of equal size.

block : Assemble arrays from blocks.

Examples

--------

>>> arrays = [np.random.randn(3, 4) for \_ in range(10)]

>>> np.stack(arrays, axis=0).shape

(10, 3, 4)

>>> np.stack(arrays, axis=1).shape

(3, 10, 4)

>>> np.stack(arrays, axis=2).shape

(3, 4, 10)

>>> a = np.array([1, 2, 3])

>>> b = np.array([2, 3, 4])

>>> np.stack((a, b))

array([[1, 2, 3],

[2, 3, 4]])

>>> np.stack((a, b), axis=-1)

array([[1, 2],

[2, 3],

[3, 4]])

std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Compute the standard deviation along the specified axis.

Returns the standard deviation, a measure of the spread of a distribution,

of the array elements. The standard deviation is computed for the

flattened array by default, otherwise over the specified axis.

Parameters

----------

a : array\_like

Calculate the standard deviation of these values.

axis : None or int or tuple of ints, optional

Axis or axes along which the standard deviation is computed. The

default is to compute the standard deviation of the flattened array.

.. versionadded:: 1.7.0

If this is a tuple of ints, a standard deviation is performed over

multiple axes, instead of a single axis or all the axes as before.

dtype : dtype, optional

Type to use in computing the standard deviation. For arrays of

integer type the default is float64, for arrays of float types it is

the same as the array type.

out : ndarray, optional

Alternative output array in which to place the result. It must have

the same shape as the expected output but the type (of the calculated

values) will be cast if necessary.

ddof : int, optional

Means Delta Degrees of Freedom. The divisor used in calculations

is ``N - ddof``, where ``N`` represents the number of elements.

By default `ddof` is zero.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `std` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

standard\_deviation : ndarray, see dtype parameter above.

If `out` is None, return a new array containing the standard deviation,

otherwise return a reference to the output array.

See Also

--------

var, mean, nanmean, nanstd, nanvar

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

The standard deviation is the square root of the average of the squared

deviations from the mean, i.e., ``std = sqrt(mean(abs(x - x.mean())\*\*2))``.

The average squared deviation is normally calculated as

``x.sum() / N``, where ``N = len(x)``. If, however, `ddof` is specified,

the divisor ``N - ddof`` is used instead. In standard statistical

practice, ``ddof=1`` provides an unbiased estimator of the variance

of the infinite population. ``ddof=0`` provides a maximum likelihood

estimate of the variance for normally distributed variables. The

standard deviation computed in this function is the square root of

the estimated variance, so even with ``ddof=1``, it will not be an

unbiased estimate of the standard deviation per se.

Note that, for complex numbers, `std` takes the absolute

value before squaring, so that the result is always real and nonnegative.

For floating-point input, the \*std\* is computed using the same

precision the input has. Depending on the input data, this can cause

the results to be inaccurate, especially for float32 (see example below).

Specifying a higher-accuracy accumulator using the `dtype` keyword can

alleviate this issue.

Examples

--------

>>> a = np.array([[1, 2], [3, 4]])

>>> np.std(a)

1.1180339887498949 # may vary

>>> np.std(a, axis=0)

array([1., 1.])

>>> np.std(a, axis=1)

array([0.5, 0.5])

In single precision, std() can be inaccurate:

>>> a = np.zeros((2, 512\*512), dtype=np.float32)

>>> a[0, :] = 1.0

>>> a[1, :] = 0.1

>>> np.std(a)

0.45000005

Computing the standard deviation in float64 is more accurate:

>>> np.std(a, dtype=np.float64)

0.44999999925494177 # may vary

sum(a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)

Sum of array elements over a given axis.

Parameters

----------

a : array\_like

Elements to sum.

axis : None or int or tuple of ints, optional

Axis or axes along which a sum is performed. The default,

axis=None, will sum all of the elements of the input array. If

axis is negative it counts from the last to the first axis.

.. versionadded:: 1.7.0

If axis is a tuple of ints, a sum is performed on all of the axes

specified in the tuple instead of a single axis or all the axes as

before.

dtype : dtype, optional

The type of the returned array and of the accumulator in which the

elements are summed. The dtype of `a` is used by default unless `a`

has an integer dtype of less precision than the default platform

integer. In that case, if `a` is signed then the platform integer

is used while if `a` is unsigned then an unsigned integer of the

same precision as the platform integer is used.

out : ndarray, optional

Alternative output array in which to place the result. It must have

the same shape as the expected output, but the type of the output

values will be cast if necessary.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `sum` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

initial : scalar, optional

Starting value for the sum. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.15.0

where : array\_like of bool, optional

Elements to include in the sum. See `~numpy.ufunc.reduce` for details.

.. versionadded:: 1.17.0

Returns

-------

sum\_along\_axis : ndarray

An array with the same shape as `a`, with the specified

axis removed. If `a` is a 0-d array, or if `axis` is None, a scalar

is returned. If an output array is specified, a reference to

`out` is returned.

See Also

--------

ndarray.sum : Equivalent method.

add.reduce : Equivalent functionality of `add`.

cumsum : Cumulative sum of array elements.

trapz : Integration of array values using the composite trapezoidal rule.

mean, average

Notes

-----

Arithmetic is modular when using integer types, and no error is

raised on overflow.

The sum of an empty array is the neutral element 0:

>>> np.sum([])

0.0

For floating point numbers the numerical precision of sum (and

``np.add.reduce``) is in general limited by directly adding each number

individually to the result causing rounding errors in every step.

However, often numpy will use a numerically better approach (partial

pairwise summation) leading to improved precision in many use-cases.

This improved precision is always provided when no ``axis`` is given.

When ``axis`` is given, it will depend on which axis is summed.

Technically, to provide the best speed possible, the improved precision

is only used when the summation is along the fast axis in memory.

Note that the exact precision may vary depending on other parameters.

In contrast to NumPy, Python's ``math.fsum`` function uses a slower but

more precise approach to summation.

Especially when summing a large number of lower precision floating point

numbers, such as ``float32``, numerical errors can become significant.

In such cases it can be advisable to use `dtype="float64"` to use a higher

precision for the output.

Examples

--------

>>> np.sum([0.5, 1.5])

2.0

>>> np.sum([0.5, 0.7, 0.2, 1.5], dtype=np.int32)

1

>>> np.sum([[0, 1], [0, 5]])

6

>>> np.sum([[0, 1], [0, 5]], axis=0)

array([0, 6])

>>> np.sum([[0, 1], [0, 5]], axis=1)

array([1, 5])

>>> np.sum([[0, 1], [np.nan, 5]], where=[False, True], axis=1)

array([1., 5.])

If the accumulator is too small, overflow occurs:

>>> np.ones(128, dtype=np.int8).sum(dtype=np.int8)

-128

You can also start the sum with a value other than zero:

>>> np.sum([10], initial=5)

15

swapaxes(a, axis1, axis2)

Interchange two axes of an array.

Parameters

----------

a : array\_like

Input array.

axis1 : int

First axis.

axis2 : int

Second axis.

Returns

-------

a\_swapped : ndarray

For NumPy >= 1.10.0, if `a` is an ndarray, then a view of `a` is

returned; otherwise a new array is created. For earlier NumPy

versions a view of `a` is returned only if the order of the

axes is changed, otherwise the input array is returned.

Examples

--------

>>> x = np.array([[1,2,3]])

>>> np.swapaxes(x,0,1)

array([[1],

[2],

[3]])

>>> x = np.array([[[0,1],[2,3]],[[4,5],[6,7]]])

>>> x

array([[[0, 1],

[2, 3]],

[[4, 5],

[6, 7]]])

>>> np.swapaxes(x,0,2)

array([[[0, 4],

[2, 6]],

[[1, 5],

[3, 7]]])

take(a, indices, axis=None, out=None, mode='raise')

Take elements from an array along an axis.

When axis is not None, this function does the same thing as "fancy"

indexing (indexing arrays using arrays); however, it can be easier to use

if you need elements along a given axis. A call such as

``np.take(arr, indices, axis=3)`` is equivalent to

``arr[:,:,:,indices,...]``.

Explained without fancy indexing, this is equivalent to the following use

of `ndindex`, which sets each of ``ii``, ``jj``, and ``kk`` to a tuple of

indices::

Ni, Nk = a.shape[:axis], a.shape[axis+1:]

Nj = indices.shape

for ii in ndindex(Ni):

for jj in ndindex(Nj):

for kk in ndindex(Nk):

out[ii + jj + kk] = a[ii + (indices[jj],) + kk]

Parameters

----------

a : array\_like (Ni..., M, Nk...)

The source array.

indices : array\_like (Nj...)

The indices of the values to extract.

.. versionadded:: 1.8.0

Also allow scalars for indices.

axis : int, optional

The axis over which to select values. By default, the flattened

input array is used.

out : ndarray, optional (Ni..., Nj..., Nk...)

If provided, the result will be placed in this array. It should

be of the appropriate shape and dtype. Note that `out` is always

buffered if `mode='raise'`; use other modes for better performance.

mode : {'raise', 'wrap', 'clip'}, optional

Specifies how out-of-bounds indices will behave.

\* 'raise' -- raise an error (default)

\* 'wrap' -- wrap around

\* 'clip' -- clip to the range

'clip' mode means that all indices that are too large are replaced

by the index that addresses the last element along that axis. Note

that this disables indexing with negative numbers.

Returns

-------

out : ndarray (Ni..., Nj..., Nk...)

The returned array has the same type as `a`.

See Also

--------

compress : Take elements using a boolean mask

ndarray.take : equivalent method

take\_along\_axis : Take elements by matching the array and the index arrays

Notes

-----

By eliminating the inner loop in the description above, and using `s\_` to

build simple slice objects, `take` can be expressed in terms of applying

fancy indexing to each 1-d slice::

Ni, Nk = a.shape[:axis], a.shape[axis+1:]

for ii in ndindex(Ni):

for kk in ndindex(Nj):

out[ii + s\_[...,] + kk] = a[ii + s\_[:,] + kk][indices]

For this reason, it is equivalent to (but faster than) the following use

of `apply\_along\_axis`::

out = np.apply\_along\_axis(lambda a\_1d: a\_1d[indices], axis, a)

Examples

--------

>>> a = [4, 3, 5, 7, 6, 8]

>>> indices = [0, 1, 4]

>>> np.take(a, indices)

array([4, 3, 6])

In this example if `a` is an ndarray, "fancy" indexing can be used.

>>> a = np.array(a)

>>> a[indices]

array([4, 3, 6])

If `indices` is not one dimensional, the output also has these dimensions.

>>> np.take(a, [[0, 1], [2, 3]])

array([[4, 3],

[5, 7]])

take\_along\_axis(arr, indices, axis)

Take values from the input array by matching 1d index and data slices.

This iterates over matching 1d slices oriented along the specified axis in

the index and data arrays, and uses the former to look up values in the

latter. These slices can be different lengths.

Functions returning an index along an axis, like `argsort` and

`argpartition`, produce suitable indices for this function.

.. versionadded:: 1.15.0

Parameters

----------

arr: ndarray (Ni..., M, Nk...)

Source array

indices: ndarray (Ni..., J, Nk...)

Indices to take along each 1d slice of `arr`. This must match the

dimension of arr, but dimensions Ni and Nj only need to broadcast

against `arr`.

axis: int

The axis to take 1d slices along. If axis is None, the input array is

treated as if it had first been flattened to 1d, for consistency with

`sort` and `argsort`.

Returns

-------

out: ndarray (Ni..., J, Nk...)

The indexed result.

Notes

-----

This is equivalent to (but faster than) the following use of `ndindex` and

`s\_`, which sets each of ``ii`` and ``kk`` to a tuple of indices::

Ni, M, Nk = a.shape[:axis], a.shape[axis], a.shape[axis+1:]

J = indices.shape[axis] # Need not equal M

out = np.empty(Ni + (J,) + Nk)

for ii in ndindex(Ni):

for kk in ndindex(Nk):

a\_1d = a [ii + s\_[:,] + kk]

indices\_1d = indices[ii + s\_[:,] + kk]

out\_1d = out [ii + s\_[:,] + kk]

for j in range(J):

out\_1d[j] = a\_1d[indices\_1d[j]]

Equivalently, eliminating the inner loop, the last two lines would be::

out\_1d[:] = a\_1d[indices\_1d]

See Also

--------

take : Take along an axis, using the same indices for every 1d slice

put\_along\_axis :

Put values into the destination array by matching 1d index and data slices

Examples

--------

For this sample array

>>> a = np.array([[10, 30, 20], [60, 40, 50]])

We can sort either by using sort directly, or argsort and this function

>>> np.sort(a, axis=1)

array([[10, 20, 30],

[40, 50, 60]])

>>> ai = np.argsort(a, axis=1); ai

array([[0, 2, 1],

[1, 2, 0]])

>>> np.take\_along\_axis(a, ai, axis=1)

array([[10, 20, 30],

[40, 50, 60]])

The same works for max and min, if you expand the dimensions:

>>> np.expand\_dims(np.max(a, axis=1), axis=1)

array([[30],

[60]])

>>> ai = np.expand\_dims(np.argmax(a, axis=1), axis=1)

>>> ai

array([[1],

[0]])

>>> np.take\_along\_axis(a, ai, axis=1)

array([[30],

[60]])

If we want to get the max and min at the same time, we can stack the

indices first

>>> ai\_min = np.expand\_dims(np.argmin(a, axis=1), axis=1)

>>> ai\_max = np.expand\_dims(np.argmax(a, axis=1), axis=1)

>>> ai = np.concatenate([ai\_min, ai\_max], axis=1)

>>> ai

array([[0, 1],

[1, 0]])

>>> np.take\_along\_axis(a, ai, axis=1)

array([[10, 30],

[40, 60]])

tensordot(a, b, axes=2)

Compute tensor dot product along specified axes.

Given two tensors, `a` and `b`, and an array\_like object containing

two array\_like objects, ``(a\_axes, b\_axes)``, sum the products of

`a`'s and `b`'s elements (components) over the axes specified by

``a\_axes`` and ``b\_axes``. The third argument can be a single non-negative

integer\_like scalar, ``N``; if it is such, then the last ``N`` dimensions

of `a` and the first ``N`` dimensions of `b` are summed over.

Parameters

----------

a, b : array\_like

Tensors to "dot".

axes : int or (2,) array\_like

\* integer\_like

If an int N, sum over the last N axes of `a` and the first N axes

of `b` in order. The sizes of the corresponding axes must match.

\* (2,) array\_like

Or, a list of axes to be summed over, first sequence applying to `a`,

second to `b`. Both elements array\_like must be of the same length.

Returns

-------

output : ndarray

The tensor dot product of the input.

See Also

--------

dot, einsum

Notes

-----

Three common use cases are:

\* ``axes = 0`` : tensor product :math:`a\otimes b`

\* ``axes = 1`` : tensor dot product :math:`a\cdot b`

\* ``axes = 2`` : (default) tensor double contraction :math:`a:b`

When `axes` is integer\_like, the sequence for evaluation will be: first

the -Nth axis in `a` and 0th axis in `b`, and the -1th axis in `a` and

Nth axis in `b` last.

When there is more than one axis to sum over - and they are not the last

(first) axes of `a` (`b`) - the argument `axes` should consist of

two sequences of the same length, with the first axis to sum over given

first in both sequences, the second axis second, and so forth.

Examples

--------

A "traditional" example:

>>> a = np.arange(60.).reshape(3,4,5)

>>> b = np.arange(24.).reshape(4,3,2)

>>> c = np.tensordot(a,b, axes=([1,0],[0,1]))

>>> c.shape

(5, 2)

>>> c

array([[4400., 4730.],

[4532., 4874.],

[4664., 5018.],

[4796., 5162.],

[4928., 5306.]])

>>> # A slower but equivalent way of computing the same...

>>> d = np.zeros((5,2))

>>> for i in range(5):

... for j in range(2):

... for k in range(3):

... for n in range(4):

... d[i,j] += a[k,n,i] \* b[n,k,j]

>>> c == d

array([[ True, True],

[ True, True],

[ True, True],

[ True, True],

[ True, True]])

An extended example taking advantage of the overloading of + and \\*:

>>> a = np.array(range(1, 9))

>>> a.shape = (2, 2, 2)

>>> A = np.array(('a', 'b', 'c', 'd'), dtype=object)

>>> A.shape = (2, 2)

>>> a; A

array([[[1, 2],

[3, 4]],

[[5, 6],

[7, 8]]])

array([['a', 'b'],

['c', 'd']], dtype=object)

>>> np.tensordot(a, A) # third argument default is 2 for double-contraction

array(['abbcccdddd', 'aaaaabbbbbbcccccccdddddddd'], dtype=object)

>>> np.tensordot(a, A, 1)

array([[['acc', 'bdd'],

['aaacccc', 'bbbdddd']],

[['aaaaacccccc', 'bbbbbdddddd'],

['aaaaaaacccccccc', 'bbbbbbbdddddddd']]], dtype=object)

>>> np.tensordot(a, A, 0) # tensor product (result too long to incl.)

array([[[[['a', 'b'],

['c', 'd']],

...

>>> np.tensordot(a, A, (0, 1))

array([[['abbbbb', 'cddddd'],

['aabbbbbb', 'ccdddddd']],

[['aaabbbbbbb', 'cccddddddd'],

['aaaabbbbbbbb', 'ccccdddddddd']]], dtype=object)

>>> np.tensordot(a, A, (2, 1))

array([[['abb', 'cdd'],

['aaabbbb', 'cccdddd']],

[['aaaaabbbbbb', 'cccccdddddd'],

['aaaaaaabbbbbbbb', 'cccccccdddddddd']]], dtype=object)

>>> np.tensordot(a, A, ((0, 1), (0, 1)))

array(['abbbcccccddddddd', 'aabbbbccccccdddddddd'], dtype=object)

>>> np.tensordot(a, A, ((2, 1), (1, 0)))

array(['acccbbdddd', 'aaaaacccccccbbbbbbdddddddd'], dtype=object)

tile(A, reps)

Construct an array by repeating A the number of times given by reps.

If `reps` has length ``d``, the result will have dimension of

``max(d, A.ndim)``.

If ``A.ndim < d``, `A` is promoted to be d-dimensional by prepending new

axes. So a shape (3,) array is promoted to (1, 3) for 2-D replication,

or shape (1, 1, 3) for 3-D replication. If this is not the desired

behavior, promote `A` to d-dimensions manually before calling this

function.

If ``A.ndim > d``, `reps` is promoted to `A`.ndim by pre-pending 1's to it.

Thus for an `A` of shape (2, 3, 4, 5), a `reps` of (2, 2) is treated as

(1, 1, 2, 2).

Note : Although tile may be used for broadcasting, it is strongly

recommended to use numpy's broadcasting operations and functions.

Parameters

----------

A : array\_like

The input array.

reps : array\_like

The number of repetitions of `A` along each axis.

Returns

-------

c : ndarray

The tiled output array.

See Also

--------

repeat : Repeat elements of an array.

broadcast\_to : Broadcast an array to a new shape

Examples

--------

>>> a = np.array([0, 1, 2])

>>> np.tile(a, 2)

array([0, 1, 2, 0, 1, 2])

>>> np.tile(a, (2, 2))

array([[0, 1, 2, 0, 1, 2],

[0, 1, 2, 0, 1, 2]])

>>> np.tile(a, (2, 1, 2))

array([[[0, 1, 2, 0, 1, 2]],

[[0, 1, 2, 0, 1, 2]]])

>>> b = np.array([[1, 2], [3, 4]])

>>> np.tile(b, 2)

array([[1, 2, 1, 2],

[3, 4, 3, 4]])

>>> np.tile(b, (2, 1))

array([[1, 2],

[3, 4],

[1, 2],

[3, 4]])

>>> c = np.array([1,2,3,4])

>>> np.tile(c,(4,1))

array([[1, 2, 3, 4],

[1, 2, 3, 4],

[1, 2, 3, 4],

[1, 2, 3, 4]])

trace(a, offset=0, axis1=0, axis2=1, dtype=None, out=None)

Return the sum along diagonals of the array.

If `a` is 2-D, the sum along its diagonal with the given offset

is returned, i.e., the sum of elements ``a[i,i+offset]`` for all i.

If `a` has more than two dimensions, then the axes specified by axis1 and

axis2 are used to determine the 2-D sub-arrays whose traces are returned.

The shape of the resulting array is the same as that of `a` with `axis1`

and `axis2` removed.

Parameters

----------

a : array\_like

Input array, from which the diagonals are taken.

offset : int, optional

Offset of the diagonal from the main diagonal. Can be both positive

and negative. Defaults to 0.

axis1, axis2 : int, optional

Axes to be used as the first and second axis of the 2-D sub-arrays

from which the diagonals should be taken. Defaults are the first two

axes of `a`.

dtype : dtype, optional

Determines the data-type of the returned array and of the accumulator

where the elements are summed. If dtype has the value None and `a` is

of integer type of precision less than the default integer

precision, then the default integer precision is used. Otherwise,

the precision is the same as that of `a`.

out : ndarray, optional

Array into which the output is placed. Its type is preserved and

it must be of the right shape to hold the output.

Returns

-------

sum\_along\_diagonals : ndarray

If `a` is 2-D, the sum along the diagonal is returned. If `a` has

larger dimensions, then an array of sums along diagonals is returned.

See Also

--------

diag, diagonal, diagflat

Examples

--------

>>> np.trace(np.eye(3))

3.0

>>> a = np.arange(8).reshape((2,2,2))

>>> np.trace(a)

array([6, 8])

>>> a = np.arange(24).reshape((2,2,2,3))

>>> np.trace(a).shape

(2, 3)

transpose(a, axes=None)

Permute the dimensions of an array.

Parameters

----------

a : array\_like

Input array.

axes : list of ints, optional

By default, reverse the dimensions, otherwise permute the axes

according to the values given.

Returns

-------

p : ndarray

`a` with its axes permuted. A view is returned whenever

possible.

See Also

--------

moveaxis

argsort

Notes

-----

Use `transpose(a, argsort(axes))` to invert the transposition of tensors

when using the `axes` keyword argument.

Transposing a 1-D array returns an unchanged view of the original array.

Examples

--------

>>> x = np.arange(4).reshape((2,2))

>>> x

array([[0, 1],

[2, 3]])

>>> np.transpose(x)

array([[0, 2],

[1, 3]])

>>> x = np.ones((1, 2, 3))

>>> np.transpose(x, (1, 0, 2)).shape

(2, 1, 3)

trapz(y, x=None, dx=1.0, axis=-1)

Integrate along the given axis using the composite trapezoidal rule.

Integrate `y` (`x`) along given axis.

Parameters

----------

y : array\_like

Input array to integrate.

x : array\_like, optional

The sample points corresponding to the `y` values. If `x` is None,

the sample points are assumed to be evenly spaced `dx` apart. The

default is None.

dx : scalar, optional

The spacing between sample points when `x` is None. The default is 1.

axis : int, optional

The axis along which to integrate.

Returns

-------

trapz : float

Definite integral as approximated by trapezoidal rule.

See Also

--------

sum, cumsum

Notes

-----

Image [2]\_ illustrates trapezoidal rule -- y-axis locations of points

will be taken from `y` array, by default x-axis distances between

points will be 1.0, alternatively they can be provided with `x` array

or with `dx` scalar. Return value will be equal to combined area under

the red lines.

References

----------

.. [1] Wikipedia page: https://en.wikipedia.org/wiki/Trapezoidal\_rule

.. [2] Illustration image:

https://en.wikipedia.org/wiki/File:Composite\_trapezoidal\_rule\_illustration.png

Examples

--------

>>> np.trapz([1,2,3])

4.0

>>> np.trapz([1,2,3], x=[4,6,8])

8.0

>>> np.trapz([1,2,3], dx=2)

8.0

>>> a = np.arange(6).reshape(2, 3)

>>> a

array([[0, 1, 2],

[3, 4, 5]])

>>> np.trapz(a, axis=0)

array([1.5, 2.5, 3.5])

>>> np.trapz(a, axis=1)

array([2., 8.])

tri(N, M=None, k=0, dtype=<class 'float'>)

An array with ones at and below the given diagonal and zeros elsewhere.

Parameters

----------

N : int

Number of rows in the array.

M : int, optional

Number of columns in the array.

By default, `M` is taken equal to `N`.

k : int, optional

The sub-diagonal at and below which the array is filled.

`k` = 0 is the main diagonal, while `k` < 0 is below it,

and `k` > 0 is above. The default is 0.

dtype : dtype, optional

Data type of the returned array. The default is float.

Returns

-------

tri : ndarray of shape (N, M)

Array with its lower triangle filled with ones and zero elsewhere;

in other words ``T[i,j] == 1`` for ``i <= j + k``, 0 otherwise.

Examples

--------

>>> np.tri(3, 5, 2, dtype=int)

array([[1, 1, 1, 0, 0],

[1, 1, 1, 1, 0],

[1, 1, 1, 1, 1]])

>>> np.tri(3, 5, -1)

array([[0., 0., 0., 0., 0.],

[1., 0., 0., 0., 0.],

[1., 1., 0., 0., 0.]])

tril(m, k=0)

Lower triangle of an array.

Return a copy of an array with elements above the `k`-th diagonal zeroed.

Parameters

----------

m : array\_like, shape (M, N)

Input array.

k : int, optional

Diagonal above which to zero elements. `k = 0` (the default) is the

main diagonal, `k < 0` is below it and `k > 0` is above.

Returns

-------

tril : ndarray, shape (M, N)

Lower triangle of `m`, of same shape and data-type as `m`.

See Also

--------

triu : same thing, only for the upper triangle

Examples

--------

>>> np.tril([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)

array([[ 0, 0, 0],

[ 4, 0, 0],

[ 7, 8, 0],

[10, 11, 12]])

tril\_indices(n, k=0, m=None)

Return the indices for the lower-triangle of an (n, m) array.

Parameters

----------

n : int

The row dimension of the arrays for which the returned

indices will be valid.

k : int, optional

Diagonal offset (see `tril` for details).

m : int, optional

.. versionadded:: 1.9.0

The column dimension of the arrays for which the returned

arrays will be valid.

By default `m` is taken equal to `n`.

Returns

-------

inds : tuple of arrays

The indices for the triangle. The returned tuple contains two arrays,

each with the indices along one dimension of the array.

See also

--------

triu\_indices : similar function, for upper-triangular.

mask\_indices : generic function accepting an arbitrary mask function.

tril, triu

Notes

-----

.. versionadded:: 1.4.0

Examples

--------

Compute two different sets of indices to access 4x4 arrays, one for the

lower triangular part starting at the main diagonal, and one starting two

diagonals further right:

>>> il1 = np.tril\_indices(4)

>>> il2 = np.tril\_indices(4, 2)

Here is how they can be used with a sample array:

>>> a = np.arange(16).reshape(4, 4)

>>> a

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

Both for indexing:

>>> a[il1]

array([ 0, 4, 5, ..., 13, 14, 15])

And for assigning values:

>>> a[il1] = -1

>>> a

array([[-1, 1, 2, 3],

[-1, -1, 6, 7],

[-1, -1, -1, 11],

[-1, -1, -1, -1]])

These cover almost the whole array (two diagonals right of the main one):

>>> a[il2] = -10

>>> a

array([[-10, -10, -10, 3],

[-10, -10, -10, -10],

[-10, -10, -10, -10],

[-10, -10, -10, -10]])

tril\_indices\_from(arr, k=0)

Return the indices for the lower-triangle of arr.

See `tril\_indices` for full details.

Parameters

----------

arr : array\_like

The indices will be valid for square arrays whose dimensions are

the same as arr.

k : int, optional

Diagonal offset (see `tril` for details).

See Also

--------

tril\_indices, tril

Notes

-----

.. versionadded:: 1.4.0

trim\_zeros(filt, trim='fb')

Trim the leading and/or trailing zeros from a 1-D array or sequence.

Parameters

----------

filt : 1-D array or sequence

Input array.

trim : str, optional

A string with 'f' representing trim from front and 'b' to trim from

back. Default is 'fb', trim zeros from both front and back of the

array.

Returns

-------

trimmed : 1-D array or sequence

The result of trimming the input. The input data type is preserved.

Examples

--------

>>> a = np.array((0, 0, 0, 1, 2, 3, 0, 2, 1, 0))

>>> np.trim\_zeros(a)

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

>>> np.trim\_zeros(a, 'b')

array([0, 0, 0, ..., 0, 2, 1])

The input data type is preserved, list/tuple in means list/tuple out.

>>> np.trim\_zeros([0, 1, 2, 0])

[1, 2]

triu(m, k=0)

Upper triangle of an array.

Return a copy of a matrix with the elements below the `k`-th diagonal

zeroed.

Please refer to the documentation for `tril` for further details.

See Also

--------

tril : lower triangle of an array

Examples

--------

>>> np.triu([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)

array([[ 1, 2, 3],

[ 4, 5, 6],

[ 0, 8, 9],

[ 0, 0, 12]])

triu\_indices(n, k=0, m=None)

Return the indices for the upper-triangle of an (n, m) array.

Parameters

----------

n : int

The size of the arrays for which the returned indices will

be valid.

k : int, optional

Diagonal offset (see `triu` for details).

m : int, optional

.. versionadded:: 1.9.0

The column dimension of the arrays for which the returned

arrays will be valid.

By default `m` is taken equal to `n`.

Returns

-------

inds : tuple, shape(2) of ndarrays, shape(`n`)

The indices for the triangle. The returned tuple contains two arrays,

each with the indices along one dimension of the array. Can be used

to slice a ndarray of shape(`n`, `n`).

See also

--------

tril\_indices : similar function, for lower-triangular.

mask\_indices : generic function accepting an arbitrary mask function.

triu, tril

Notes

-----

.. versionadded:: 1.4.0

Examples

--------

Compute two different sets of indices to access 4x4 arrays, one for the

upper triangular part starting at the main diagonal, and one starting two

diagonals further right:

>>> iu1 = np.triu\_indices(4)

>>> iu2 = np.triu\_indices(4, 2)

Here is how they can be used with a sample array:

>>> a = np.arange(16).reshape(4, 4)

>>> a

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

Both for indexing:

>>> a[iu1]

array([ 0, 1, 2, ..., 10, 11, 15])

And for assigning values:

>>> a[iu1] = -1

>>> a

array([[-1, -1, -1, -1],

[ 4, -1, -1, -1],

[ 8, 9, -1, -1],

[12, 13, 14, -1]])

These cover only a small part of the whole array (two diagonals right

of the main one):

>>> a[iu2] = -10

>>> a

array([[ -1, -1, -10, -10],

[ 4, -1, -1, -10],

[ 8, 9, -1, -1],

[ 12, 13, 14, -1]])

triu\_indices\_from(arr, k=0)

Return the indices for the upper-triangle of arr.

See `triu\_indices` for full details.

Parameters

----------

arr : ndarray, shape(N, N)

The indices will be valid for square arrays.

k : int, optional

Diagonal offset (see `triu` for details).

Returns

-------

triu\_indices\_from : tuple, shape(2) of ndarray, shape(N)

Indices for the upper-triangle of `arr`.

See Also

--------

triu\_indices, triu

Notes

-----

.. versionadded:: 1.4.0

typename(char)

Return a description for the given data type code.

Parameters

----------

char : str

Data type code.

Returns

-------

out : str

Description of the input data type code.

See Also

--------

dtype, typecodes

Examples

--------

>>> typechars = ['S1', '?', 'B', 'D', 'G', 'F', 'I', 'H', 'L', 'O', 'Q',

... 'S', 'U', 'V', 'b', 'd', 'g', 'f', 'i', 'h', 'l', 'q']

>>> for typechar in typechars:

... print(typechar, ' : ', np.typename(typechar))

...

S1 : character

? : bool

B : unsigned char

D : complex double precision

G : complex long double precision

F : complex single precision

I : unsigned integer

H : unsigned short

L : unsigned long integer

O : object

Q : unsigned long long integer

S : string

U : unicode

V : void

b : signed char

d : double precision

g : long precision

f : single precision

i : integer

h : short

l : long integer

q : long long integer

union1d(ar1, ar2)

Find the union of two arrays.

Return the unique, sorted array of values that are in either of the two

input arrays.

Parameters

----------

ar1, ar2 : array\_like

Input arrays. They are flattened if they are not already 1D.

Returns

-------

union1d : ndarray

Unique, sorted union of the input arrays.

See Also

--------

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Examples

--------

>>> np.union1d([-1, 0, 1], [-2, 0, 2])

array([-2, -1, 0, 1, 2])

To find the union of more than two arrays, use functools.reduce:

>>> from functools import reduce

>>> reduce(np.union1d, ([1, 3, 4, 3], [3, 1, 2, 1], [6, 3, 4, 2]))

array([1, 2, 3, 4, 6])

unique(ar, return\_index=False, return\_inverse=False, return\_counts=False, axis=None)

Find the unique elements of an array.

Returns the sorted unique elements of an array. There are three optional

outputs in addition to the unique elements:

\* the indices of the input array that give the unique values

\* the indices of the unique array that reconstruct the input array

\* the number of times each unique value comes up in the input array

Parameters

----------

ar : array\_like

Input array. Unless `axis` is specified, this will be flattened if it

is not already 1-D.

return\_index : bool, optional

If True, also return the indices of `ar` (along the specified axis,

if provided, or in the flattened array) that result in the unique array.

return\_inverse : bool, optional

If True, also return the indices of the unique array (for the specified

axis, if provided) that can be used to reconstruct `ar`.

return\_counts : bool, optional

If True, also return the number of times each unique item appears

in `ar`.

.. versionadded:: 1.9.0

axis : int or None, optional

The axis to operate on. If None, `ar` will be flattened. If an integer,

the subarrays indexed by the given axis will be flattened and treated

as the elements of a 1-D array with the dimension of the given axis,

see the notes for more details. Object arrays or structured arrays

that contain objects are not supported if the `axis` kwarg is used. The

default is None.

.. versionadded:: 1.13.0

Returns

-------

unique : ndarray

The sorted unique values.

unique\_indices : ndarray, optional

The indices of the first occurrences of the unique values in the

original array. Only provided if `return\_index` is True.

unique\_inverse : ndarray, optional

The indices to reconstruct the original array from the

unique array. Only provided if `return\_inverse` is True.

unique\_counts : ndarray, optional

The number of times each of the unique values comes up in the

original array. Only provided if `return\_counts` is True.

.. versionadded:: 1.9.0

See Also

--------

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

Notes

-----

When an axis is specified the subarrays indexed by the axis are sorted.

This is done by making the specified axis the first dimension of the array

and then flattening the subarrays in C order. The flattened subarrays are

then viewed as a structured type with each element given a label, with the

effect that we end up with a 1-D array of structured types that can be

treated in the same way as any other 1-D array. The result is that the

flattened subarrays are sorted in lexicographic order starting with the

first element.

Examples

--------

>>> np.unique([1, 1, 2, 2, 3, 3])

array([1, 2, 3])

>>> a = np.array([[1, 1], [2, 3]])

>>> np.unique(a)

array([1, 2, 3])

Return the unique rows of a 2D array

>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])

>>> np.unique(a, axis=0)

array([[1, 0, 0], [2, 3, 4]])

Return the indices of the original array that give the unique values:

>>> a = np.array(['a', 'b', 'b', 'c', 'a'])

>>> u, indices = np.unique(a, return\_index=True)

>>> u

array(['a', 'b', 'c'], dtype='<U1')

>>> indices

array([0, 1, 3])

>>> a[indices]

array(['a', 'b', 'c'], dtype='<U1')

Reconstruct the input array from the unique values:

>>> a = np.array([1, 2, 6, 4, 2, 3, 2])

>>> u, indices = np.unique(a, return\_inverse=True)

>>> u

array([1, 2, 3, 4, 6])

>>> indices

array([0, 1, 4, ..., 1, 2, 1])

>>> u[indices]

array([1, 2, 6, ..., 2, 3, 2])

unpackbits(...)

unpackbits(a, axis=None, count=None, bitorder='big')

Unpacks elements of a uint8 array into a binary-valued output array.

Each element of `a` represents a bit-field that should be unpacked

into a binary-valued output array. The shape of the output array is

either 1-D (if `axis` is ``None``) or the same shape as the input

array with unpacking done along the axis specified.

Parameters

----------

a : ndarray, uint8 type

Input array.

axis : int, optional

The dimension over which bit-unpacking is done.

``None`` implies unpacking the flattened array.

count : int or None, optional

The number of elements to unpack along `axis`, provided as a way

of undoing the effect of packing a size that is not a multiple

of eight. A non-negative number means to only unpack `count`

bits. A negative number means to trim off that many bits from

the end. ``None`` means to unpack the entire array (the

default). Counts larger than the available number of bits will

add zero padding to the output. Negative counts must not

exceed the available number of bits.

.. versionadded:: 1.17.0

bitorder : {'big', 'little'}, optional

The order of the returned bits. 'big' will mimic bin(val),

``3 = 0b00000011 => [0, 0, 0, 0, 0, 0, 1, 1]``, 'little' will reverse

the order to ``[1, 1, 0, 0, 0, 0, 0, 0]``.

Defaults to 'big'.

.. versionadded:: 1.17.0

Returns

-------

unpacked : ndarray, uint8 type

The elements are binary-valued (0 or 1).

See Also

--------

packbits : Packs the elements of a binary-valued array into bits in

a uint8 array.

Examples

--------

>>> a = np.array([[2], [7], [23]], dtype=np.uint8)

>>> a

array([[ 2],

[ 7],

[23]], dtype=uint8)

>>> b = np.unpackbits(a, axis=1)

>>> b

array([[0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 0, 1, 1, 1],

[0, 0, 0, 1, 0, 1, 1, 1]], dtype=uint8)

>>> c = np.unpackbits(a, axis=1, count=-3)

>>> c

array([[0, 0, 0, 0, 0],

[0, 0, 0, 0, 0],

[0, 0, 0, 1, 0]], dtype=uint8)

>>> p = np.packbits(b, axis=0)

>>> np.unpackbits(p, axis=0)

array([[0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 0, 1, 1, 1],

[0, 0, 0, 1, 0, 1, 1, 1],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0]], dtype=uint8)

>>> np.array\_equal(b, np.unpackbits(p, axis=0, count=b.shape[0]))

True

unravel\_index(...)

unravel\_index(indices, shape, order='C')

Converts a flat index or array of flat indices into a tuple

of coordinate arrays.

Parameters

----------

indices : array\_like

An integer array whose elements are indices into the flattened

version of an array of dimensions ``shape``. Before version 1.6.0,

this function accepted just one index value.

shape : tuple of ints

The shape of the array to use for unraveling ``indices``.

.. versionchanged:: 1.16.0

Renamed from ``dims`` to ``shape``.

order : {'C', 'F'}, optional

Determines whether the indices should be viewed as indexing in

row-major (C-style) or column-major (Fortran-style) order.

.. versionadded:: 1.6.0

Returns

-------

unraveled\_coords : tuple of ndarray

Each array in the tuple has the same shape as the ``indices``

array.

See Also

--------

ravel\_multi\_index

Examples

--------

>>> np.unravel\_index([22, 41, 37], (7,6))

(array([3, 6, 6]), array([4, 5, 1]))

>>> np.unravel\_index([31, 41, 13], (7,6), order='F')

(array([3, 6, 6]), array([4, 5, 1]))

>>> np.unravel\_index(1621, (6,7,8,9))

(3, 1, 4, 1)

unwrap(p, discont=3.141592653589793, axis=-1)

Unwrap by changing deltas between values to 2\*pi complement.

Unwrap radian phase `p` by changing absolute jumps greater than

`discont` to their 2\*pi complement along the given axis.

Parameters

----------

p : array\_like

Input array.

discont : float, optional

Maximum discontinuity between values, default is ``pi``.

axis : int, optional

Axis along which unwrap will operate, default is the last axis.

Returns

-------

out : ndarray

Output array.

See Also

--------

rad2deg, deg2rad

Notes

-----

If the discontinuity in `p` is smaller than ``pi``, but larger than

`discont`, no unwrapping is done because taking the 2\*pi complement

would only make the discontinuity larger.

Examples

--------

>>> phase = np.linspace(0, np.pi, num=5)

>>> phase[3:] += np.pi

>>> phase

array([ 0. , 0.78539816, 1.57079633, 5.49778714, 6.28318531]) # may vary

>>> np.unwrap(phase)

array([ 0. , 0.78539816, 1.57079633, -0.78539816, 0. ]) # may vary

vander(x, N=None, increasing=False)

Generate a Vandermonde matrix.

The columns of the output matrix are powers of the input vector. The

order of the powers is determined by the `increasing` boolean argument.

Specifically, when `increasing` is False, the `i`-th output column is

the input vector raised element-wise to the power of ``N - i - 1``. Such

a matrix with a geometric progression in each row is named for Alexandre-

Theophile Vandermonde.

Parameters

----------

x : array\_like

1-D input array.

N : int, optional

Number of columns in the output. If `N` is not specified, a square

array is returned (``N = len(x)``).

increasing : bool, optional

Order of the powers of the columns. If True, the powers increase

from left to right, if False (the default) they are reversed.

.. versionadded:: 1.9.0

Returns

-------

out : ndarray

Vandermonde matrix. If `increasing` is False, the first column is

``x^(N-1)``, the second ``x^(N-2)`` and so forth. If `increasing` is

True, the columns are ``x^0, x^1, ..., x^(N-1)``.

See Also

--------

polynomial.polynomial.polyvander

Examples

--------

>>> x = np.array([1, 2, 3, 5])

>>> N = 3

>>> np.vander(x, N)

array([[ 1, 1, 1],

[ 4, 2, 1],

[ 9, 3, 1],

[25, 5, 1]])

>>> np.column\_stack([x\*\*(N-1-i) for i in range(N)])

array([[ 1, 1, 1],

[ 4, 2, 1],

[ 9, 3, 1],

[25, 5, 1]])

>>> x = np.array([1, 2, 3, 5])

>>> np.vander(x)

array([[ 1, 1, 1, 1],

[ 8, 4, 2, 1],

[ 27, 9, 3, 1],

[125, 25, 5, 1]])

>>> np.vander(x, increasing=True)

array([[ 1, 1, 1, 1],

[ 1, 2, 4, 8],

[ 1, 3, 9, 27],

[ 1, 5, 25, 125]])

The determinant of a square Vandermonde matrix is the product

of the differences between the values of the input vector:

>>> np.linalg.det(np.vander(x))

48.000000000000043 # may vary

>>> (5-3)\*(5-2)\*(5-1)\*(3-2)\*(3-1)\*(2-1)

48

var(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Compute the variance along the specified axis.

Returns the variance of the array elements, a measure of the spread of a

distribution. The variance is computed for the flattened array by

default, otherwise over the specified axis.

Parameters

----------

a : array\_like

Array containing numbers whose variance is desired. If `a` is not an

array, a conversion is attempted.

axis : None or int or tuple of ints, optional

Axis or axes along which the variance is computed. The default is to

compute the variance of the flattened array.

.. versionadded:: 1.7.0

If this is a tuple of ints, a variance is performed over multiple axes,

instead of a single axis or all the axes as before.

dtype : data-type, optional

Type to use in computing the variance. For arrays of integer type

the default is `float32`; for arrays of float types it is the same as

the array type.

out : ndarray, optional

Alternate output array in which to place the result. It must have

the same shape as the expected output, but the type is cast if

necessary.

ddof : int, optional

"Delta Degrees of Freedom": the divisor used in the calculation is

``N - ddof``, where ``N`` represents the number of elements. By

default `ddof` is zero.

keepdims : bool, optional

If this is set to True, the axes which are reduced are left

in the result as dimensions with size one. With this option,

the result will broadcast correctly against the input array.

If the default value is passed, then `keepdims` will not be

passed through to the `var` method of sub-classes of

`ndarray`, however any non-default value will be. If the

sub-class' method does not implement `keepdims` any

exceptions will be raised.

Returns

-------

variance : ndarray, see dtype parameter above

If ``out=None``, returns a new array containing the variance;

otherwise, a reference to the output array is returned.

See Also

--------

std, mean, nanmean, nanstd, nanvar

numpy.doc.ufuncs : Section "Output arguments"

Notes

-----

The variance is the average of the squared deviations from the mean,

i.e., ``var = mean(abs(x - x.mean())\*\*2)``.

The mean is normally calculated as ``x.sum() / N``, where ``N = len(x)``.

If, however, `ddof` is specified, the divisor ``N - ddof`` is used

instead. In standard statistical practice, ``ddof=1`` provides an

unbiased estimator of the variance of a hypothetical infinite population.

``ddof=0`` provides a maximum likelihood estimate of the variance for

normally distributed variables.

Note that for complex numbers, the absolute value is taken before

squaring, so that the result is always real and nonnegative.

For floating-point input, the variance is computed using the same

precision the input has. Depending on the input data, this can cause

the results to be inaccurate, especially for `float32` (see example

below). Specifying a higher-accuracy accumulator using the ``dtype``

keyword can alleviate this issue.

Examples

--------

>>> a = np.array([[1, 2], [3, 4]])

>>> np.var(a)

1.25

>>> np.var(a, axis=0)

array([1., 1.])

>>> np.var(a, axis=1)

array([0.25, 0.25])

In single precision, var() can be inaccurate:

>>> a = np.zeros((2, 512\*512), dtype=np.float32)

>>> a[0, :] = 1.0

>>> a[1, :] = 0.1

>>> np.var(a)

0.20250003

Computing the variance in float64 is more accurate:

>>> np.var(a, dtype=np.float64)

0.20249999932944759 # may vary

>>> ((1-0.55)\*\*2 + (0.1-0.55)\*\*2)/2

0.2025

vdot(...)

vdot(a, b)

Return the dot product of two vectors.

The vdot(`a`, `b`) function handles complex numbers differently than

dot(`a`, `b`). If the first argument is complex the complex conjugate

of the first argument is used for the calculation of the dot product.

Note that `vdot` handles multidimensional arrays differently than `dot`:

it does \*not\* perform a matrix product, but flattens input arguments

to 1-D vectors first. Consequently, it should only be used for vectors.

Parameters

----------

a : array\_like

If `a` is complex the complex conjugate is taken before calculation

of the dot product.

b : array\_like

Second argument to the dot product.

Returns

-------

output : ndarray

Dot product of `a` and `b`. Can be an int, float, or

complex depending on the types of `a` and `b`.

See Also

--------

dot : Return the dot product without using the complex conjugate of the

first argument.

Examples

--------

>>> a = np.array([1+2j,3+4j])

>>> b = np.array([5+6j,7+8j])

>>> np.vdot(a, b)

(70-8j)

>>> np.vdot(b, a)

(70+8j)

Note that higher-dimensional arrays are flattened!

>>> a = np.array([[1, 4], [5, 6]])

>>> b = np.array([[4, 1], [2, 2]])

>>> np.vdot(a, b)

30

>>> np.vdot(b, a)

30

>>> 1\*4 + 4\*1 + 5\*2 + 6\*2

30

vsplit(ary, indices\_or\_sections)

Split an array into multiple sub-arrays vertically (row-wise).

Please refer to the ``split`` documentation. ``vsplit`` is equivalent

to ``split`` with `axis=0` (default), the array is always split along the

first axis regardless of the array dimension.

See Also

--------

split : Split an array into multiple sub-arrays of equal size.

Examples

--------

>>> x = np.arange(16.0).reshape(4, 4)

>>> x

array([[ 0., 1., 2., 3.],

[ 4., 5., 6., 7.],

[ 8., 9., 10., 11.],

[12., 13., 14., 15.]])

>>> np.vsplit(x, 2)

[array([[0., 1., 2., 3.],

[4., 5., 6., 7.]]), array([[ 8., 9., 10., 11.],

[12., 13., 14., 15.]])]

>>> np.vsplit(x, np.array([3, 6]))

[array([[ 0., 1., 2., 3.],

[ 4., 5., 6., 7.],

[ 8., 9., 10., 11.]]), array([[12., 13., 14., 15.]]), array([], shape=(0, 4), dtype=float64)]

With a higher dimensional array the split is still along the first axis.

>>> x = np.arange(8.0).reshape(2, 2, 2)

>>> x

array([[[0., 1.],

[2., 3.]],

[[4., 5.],

[6., 7.]]])

>>> np.vsplit(x, 2)

[array([[[0., 1.],

[2., 3.]]]), array([[[4., 5.],

[6., 7.]]])]

vstack(tup)

Stack arrays in sequence vertically (row wise).

This is equivalent to concatenation along the first axis after 1-D arrays

of shape `(N,)` have been reshaped to `(1,N)`. Rebuilds arrays divided by

`vsplit`.

This function makes most sense for arrays with up to 3 dimensions. For

instance, for pixel-data with a height (first axis), width (second axis),

and r/g/b channels (third axis). The functions `concatenate`, `stack` and

`block` provide more general stacking and concatenation operations.

Parameters

----------

tup : sequence of ndarrays

The arrays must have the same shape along all but the first axis.

1-D arrays must have the same length.

Returns

-------

stacked : ndarray

The array formed by stacking the given arrays, will be at least 2-D.

See Also

--------

stack : Join a sequence of arrays along a new axis.

hstack : Stack arrays in sequence horizontally (column wise).

dstack : Stack arrays in sequence depth wise (along third dimension).

concatenate : Join a sequence of arrays along an existing axis.

vsplit : Split array into a list of multiple sub-arrays vertically.

block : Assemble arrays from blocks.

Examples

--------

>>> a = np.array([1, 2, 3])

>>> b = np.array([2, 3, 4])

>>> np.vstack((a,b))

array([[1, 2, 3],

[2, 3, 4]])

>>> a = np.array([[1], [2], [3]])

>>> b = np.array([[2], [3], [4]])

>>> np.vstack((a,b))

array([[1],

[2],

[3],

[2],

[3],

[4]])

where(...)

where(condition, [x, y])

Return elements chosen from `x` or `y` depending on `condition`.

.. note::

When only `condition` is provided, this function is a shorthand for

``np.asarray(condition).nonzero()``. Using `nonzero` directly should be

preferred, as it behaves correctly for subclasses. The rest of this

documentation covers only the case where all three arguments are

provided.

Parameters

----------

condition : array\_like, bool

Where True, yield `x`, otherwise yield `y`.

x, y : array\_like

Values from which to choose. `x`, `y` and `condition` need to be

broadcastable to some shape.

Returns

-------

out : ndarray

An array with elements from `x` where `condition` is True, and elements

from `y` elsewhere.

See Also

--------

choose

nonzero : The function that is called when x and y are omitted

Notes

-----

If all the arrays are 1-D, `where` is equivalent to::

[xv if c else yv

for c, xv, yv in zip(condition, x, y)]

Examples

--------

>>> a = np.arange(10)

>>> a

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.where(a < 5, a, 10\*a)

array([ 0, 1, 2, 3, 4, 50, 60, 70, 80, 90])

This can be used on multidimensional arrays too:

>>> np.where([[True, False], [True, True]],

... [[1, 2], [3, 4]],

... [[9, 8], [7, 6]])

array([[1, 8],

[3, 4]])

The shapes of x, y, and the condition are broadcast together:

>>> x, y = np.ogrid[:3, :4]

>>> np.where(x < y, x, 10 + y) # both x and 10+y are broadcast

array([[10, 0, 0, 0],

[10, 11, 1, 1],

[10, 11, 12, 2]])

>>> a = np.array([[0, 1, 2],

... [0, 2, 4],

... [0, 3, 6]])

>>> np.where(a < 4, a, -1) # -1 is broadcast

array([[ 0, 1, 2],

[ 0, 2, -1],

[ 0, 3, -1]])

who(vardict=None)

Print the NumPy arrays in the given dictionary.

If there is no dictionary passed in or `vardict` is None then returns

NumPy arrays in the globals() dictionary (all NumPy arrays in the

namespace).

Parameters

----------

vardict : dict, optional

A dictionary possibly containing ndarrays. Default is globals().

Returns

-------

out : None

Returns 'None'.

Notes

-----

Prints out the name, shape, bytes and type of all of the ndarrays

present in `vardict`.

Examples

--------

>>> a = np.arange(10)

>>> b = np.ones(20)

>>> np.who()

Name Shape Bytes Type

===========================================================

a 10 80 int64

b 20 160 float64

Upper bound on total bytes = 240

>>> d = {'x': np.arange(2.0), 'y': np.arange(3.0), 'txt': 'Some str',

... 'idx':5}

>>> np.who(d)

Name Shape Bytes Type

===========================================================

x 2 16 float64

y 3 24 float64

Upper bound on total bytes = 40

zeros(...)

zeros(shape, dtype=float, order='C')

Return a new array of given shape and type, filled with zeros.

Parameters

----------

shape : int or tuple of ints

Shape of the new array, e.g., ``(2, 3)`` or ``2``.

dtype : data-type, optional

The desired data-type for the array, e.g., `numpy.int8`. Default is

`numpy.float64`.

order : {'C', 'F'}, optional, default: 'C'

Whether to store multi-dimensional data in row-major

(C-style) or column-major (Fortran-style) order in

memory.

Returns

-------

out : ndarray

Array of zeros with the given shape, dtype, and order.

See Also

--------

zeros\_like : Return an array of zeros with shape and type of input.

empty : Return a new uninitialized array.

ones : Return a new array setting values to one.

full : Return a new array of given shape filled with value.

Examples

--------

>>> np.zeros(5)

array([ 0., 0., 0., 0., 0.])

>>> np.zeros((5,), dtype=int)

array([0, 0, 0, 0, 0])

>>> np.zeros((2, 1))

array([[ 0.],

[ 0.]])

>>> s = (2,2)

>>> np.zeros(s)

array([[ 0., 0.],

[ 0., 0.]])

>>> np.zeros((2,), dtype=[('x', 'i4'), ('y', 'i4')]) # custom dtype

array([(0, 0), (0, 0)],

dtype=[('x', '<i4'), ('y', '<i4')])

zeros\_like(a, dtype=None, order='K', subok=True, shape=None)

Return an array of zeros with the same shape and type as a given array.

Parameters

----------

a : array\_like

The shape and data-type of `a` define these same attributes of

the returned array.

dtype : data-type, optional

Overrides the data type of the result.

.. versionadded:: 1.6.0

order : {'C', 'F', 'A', or 'K'}, optional

Overrides the memory layout of the result. 'C' means C-order,

'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,

'C' otherwise. 'K' means match the layout of `a` as closely

as possible.

.. versionadded:: 1.6.0

subok : bool, optional.

If True, then the newly created array will use the sub-class

type of 'a', otherwise it will be a base-class array. Defaults

to True.

shape : int or sequence of ints, optional.

Overrides the shape of the result. If order='K' and the number of

dimensions is unchanged, will try to keep order, otherwise,

order='C' is implied.

.. versionadded:: 1.17.0

Returns

-------

out : ndarray

Array of zeros with the same shape and type as `a`.

See Also

--------

empty\_like : Return an empty array with shape and type of input.

ones\_like : Return an array of ones with shape and type of input.

full\_like : Return a new array with shape of input filled with value.

zeros : Return a new array setting values to zero.

Examples

--------

>>> x = np.arange(6)

>>> x = x.reshape((2, 3))

>>> x

array([[0, 1, 2],

[3, 4, 5]])

>>> np.zeros\_like(x)

array([[0, 0, 0],

[0, 0, 0]])

>>> y = np.arange(3, dtype=float)

>>> y

array([0., 1., 2.])

>>> np.zeros\_like(y)

array([0., 0., 0.])

DATA

ALLOW\_THREADS = 1

BUFSIZE = 8192

CLIP = 0

ERR\_CALL = 3

ERR\_DEFAULT = 521

ERR\_IGNORE = 0

ERR\_LOG = 5

ERR\_PRINT = 4

ERR\_RAISE = 2

ERR\_WARN = 1

FLOATING\_POINT\_SUPPORT = 1

FPE\_DIVIDEBYZERO = 1

FPE\_INVALID = 8

FPE\_OVERFLOW = 2

FPE\_UNDERFLOW = 4

False\_ = False

Inf = inf

Infinity = inf

MAXDIMS = 32

MAY\_SHARE\_BOUNDS = 0

MAY\_SHARE\_EXACT = -1

NAN = nan

NINF = -inf

NZERO = -0.0

NaN = nan

PINF = inf

PZERO = 0.0

RAISE = 2

SHIFT\_DIVIDEBYZERO = 0

SHIFT\_INVALID = 9

SHIFT\_OVERFLOW = 3

SHIFT\_UNDERFLOW = 6

ScalarType = (<class 'int'>, <class 'float'>, <class 'complex'>, <clas...

True\_ = True

UFUNC\_BUFSIZE\_DEFAULT = 8192

UFUNC\_PYVALS\_NAME = 'UFUNC\_PYVALS'

WRAP = 1

\_UFUNC\_API = <capsule object NULL>

\_\_NUMPY\_SETUP\_\_ = False

\_\_all\_\_ = ['ModuleDeprecationWarning', 'VisibleDeprecationWarning', '\_...

\_\_git\_revision\_\_ = 'fce34f559f17674e7a3301c46b0a9cc991c143d4'

absolute = <ufunc 'absolute'>

add = <ufunc 'add'>

arccos = <ufunc 'arccos'>

arccosh = <ufunc 'arccosh'>

arcsin = <ufunc 'arcsin'>

arcsinh = <ufunc 'arcsinh'>

arctan = <ufunc 'arctan'>

arctan2 = <ufunc 'arctan2'>

arctanh = <ufunc 'arctanh'>

bitwise\_and = <ufunc 'bitwise\_and'>

bitwise\_not = <ufunc 'invert'>

bitwise\_or = <ufunc 'bitwise\_or'>

bitwise\_xor = <ufunc 'bitwise\_xor'>

c\_ = <numpy.lib.index\_tricks.CClass object>

cast = {<class 'numpy.int32'>: <function <lambda> at 0x...y.complex64'...

cbrt = <ufunc 'cbrt'>

ceil = <ufunc 'ceil'>

conj = <ufunc 'conjugate'>

conjugate = <ufunc 'conjugate'>

copysign = <ufunc 'copysign'>

cos = <ufunc 'cos'>

cosh = <ufunc 'cosh'>

deg2rad = <ufunc 'deg2rad'>

degrees = <ufunc 'degrees'>

divide = <ufunc 'true\_divide'>

divmod = <ufunc 'divmod'>

e = 2.718281828459045

equal = <ufunc 'equal'>

euler\_gamma = 0.5772156649015329

exp = <ufunc 'exp'>

exp2 = <ufunc 'exp2'>

expm1 = <ufunc 'expm1'>

fabs = <ufunc 'fabs'>

float\_power = <ufunc 'float\_power'>

floor = <ufunc 'floor'>

floor\_divide = <ufunc 'floor\_divide'>

fmax = <ufunc 'fmax'>

fmin = <ufunc 'fmin'>

fmod = <ufunc 'fmod'>

frexp = <ufunc 'frexp'>

gcd = <ufunc 'gcd'>

greater = <ufunc 'greater'>

greater\_equal = <ufunc 'greater\_equal'>

heaviside = <ufunc 'heaviside'>

hypot = <ufunc 'hypot'>

index\_exp = <numpy.lib.index\_tricks.IndexExpression object>

inf = inf

infty = inf

invert = <ufunc 'invert'>

isfinite = <ufunc 'isfinite'>

isinf = <ufunc 'isinf'>

isnan = <ufunc 'isnan'>

isnat = <ufunc 'isnat'>

lcm = <ufunc 'lcm'>

ldexp = <ufunc 'ldexp'>

left\_shift = <ufunc 'left\_shift'>

less = <ufunc 'less'>

less\_equal = <ufunc 'less\_equal'>

little\_endian = True

log = <ufunc 'log'>

log10 = <ufunc 'log10'>

log1p = <ufunc 'log1p'>

log2 = <ufunc 'log2'>

logaddexp = <ufunc 'logaddexp'>

logaddexp2 = <ufunc 'logaddexp2'>

logical\_and = <ufunc 'logical\_and'>

logical\_not = <ufunc 'logical\_not'>

logical\_or = <ufunc 'logical\_or'>

logical\_xor = <ufunc 'logical\_xor'>

matmul = <ufunc 'matmul'>

maximum = <ufunc 'maximum'>

mgrid = <numpy.lib.index\_tricks.MGridClass object>

minimum = <ufunc 'minimum'>

mod = <ufunc 'remainder'>

modf = <ufunc 'modf'>

multiply = <ufunc 'multiply'>

nan = nan

nbytes = {<class 'numpy.bool\_'>: 1, <class 'numpy.int8'>:....datetime6...

negative = <ufunc 'negative'>

newaxis = None

nextafter = <ufunc 'nextafter'>

not\_equal = <ufunc 'not\_equal'>

ogrid = <numpy.lib.index\_tricks.OGridClass object>

pi = 3.141592653589793

positive = <ufunc 'positive'>

power = <ufunc 'power'>

r\_ = <numpy.lib.index\_tricks.RClass object>

rad2deg = <ufunc 'rad2deg'>

radians = <ufunc 'radians'>

reciprocal = <ufunc 'reciprocal'>

remainder = <ufunc 'remainder'>

right\_shift = <ufunc 'right\_shift'>

rint = <ufunc 'rint'>

s\_ = <numpy.lib.index\_tricks.IndexExpression object>

sctypeDict = {'?': <class 'numpy.bool\_'>, 0: <class 'numpy.bool\_'>, 'b...

sctypeNA = {'Bool': <class 'numpy.bool\_'>, <class 'numpy.bo...'>, <cla...

sctypes = {'complex': [<class 'numpy.complex64'>, <class 'numpy.comple...

sign = <ufunc 'sign'>

signbit = <ufunc 'signbit'>

sin = <ufunc 'sin'>

sinh = <ufunc 'sinh'>

spacing = <ufunc 'spacing'>

sqrt = <ufunc 'sqrt'>

square = <ufunc 'square'>

subtract = <ufunc 'subtract'>

tan = <ufunc 'tan'>

tanh = <ufunc 'tanh'>

tracemalloc\_domain = 389047

true\_divide = <ufunc 'true\_divide'>

trunc = <ufunc 'trunc'>

typeDict = {'?': <class 'numpy.bool\_'>, 0: <class 'numpy.bool\_'>, 'byt...

typeNA = {'Bool': <class 'numpy.bool\_'>, <class 'numpy.bo...'>, <class...

typecodes = {'All': '?bhilqpBHILQPefdgFDGSUVOMm', 'AllFloat': 'efdgFDG...

VERSION

1.17.2

FILE

c:\users\com\appdata\local\programs\python\python37-32\lib\site-packages\numpy\\_\_init\_\_.py