numpy.ndarray

class numpy.**ndarray**(*shape, dtype=float, buffer=None, offset=0, strides=None, order=None*) [source]

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.

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.

numpy.typing.NDArray
An ndarray alias generic w.r.t. its **dtype.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])
```

Attributes: **T** : *ndarray*

View of the transposed array.

data : *buffer*

Python buffer object pointing to the start of the array's data.

dtype : *dtype object*

Data-type of the array's elements.

flags : *dict*

Information about the memory layout of the array.

flat : *numpy.flatiter object*

A 1-D iterator over the array.

imag : *ndarray*

The imaginary part of the array.

real : *ndarray*

The real part of the array.

size : *int*

Number of elements in the array.

itemsize : *int*

Length of one array element in bytes.

nbytes : *int*

Total bytes consumed by the elements of the array.

ndim : *int*

Number of array dimensions.

shape : *tuple of ints*

Tuple of array dimensions.

strides : *tuple of ints*

Tuple of bytes to step in each dimension when traversing an array.

ctypes : *ctypes object*

An object to simplify the interaction of the array with the ctypes module.

base : *ndarray*

Base object if memory is from some other object.

Methods

all([axis, out, keepdims, where])	Returns True if all elements evaluate to True.

`any`([axis, out, keepdims, where])	Returns True if any of the elements of *a* evaluate to True.
`argmax`([axis, out, keepdims])	Return indices of the maximum values along the given axis.
`argmin`([axis, out, keepdims])	Return indices of the minimum values along the given axis.
`argpartition`(kth[, axis, kind, order])	Returns the indices that would partition this array.
`argsort`([axis, kind, order])	Returns the indices that would sort this array.
`astype`(dtype[, order, casting, subok, copy])	Copy of the array, cast to a specified type.
`byteswap`([inplace])	Swap the bytes of the array elements
`choose`(choices[, out, mode])	Use an index array to construct a new array from a set of choices.
`clip`([min, max, out])	Return an array whose values are limited to `[min, max]`.
`compress`(condition[, axis, out])	Return selected slices of this array along given axis.
`conj`()	Complex-conjugate all elements.
`conjugate`()	Return the complex conjugate, element-wise.
`copy`([order])	Return a copy of the array.
`cumprod`([axis, dtype, out])	Return the cumulative product of the elements along the given axis.
`cumsum`([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
`diagonal`([offset, axis1, axis2])	Return specified diagonals.
`dump`(file)	Dump a pickle of the array to the specified file.
`dumps`()	Returns the pickle of the array as a string.
`fill`(value)	Fill the array with a scalar value.
`flatten`([order])	Return a copy of the array collapsed into one dimension.
`getfield`(dtype[, offset])	Returns a field of the given array as a certain type.
`item`(*args)	Copy an element of an array to a standard Python scalar and return it.

`itemset`(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
`max`([axis, out, keepdims, initial, where])	Return the maximum along a given axis.
`mean`([axis, dtype, out, keepdims, where])	Returns the average of the array elements along given axis.
`min`([axis, out, keepdims, initial, where])	Return the minimum along a given axis.
`newbyteorder`([new_order])	Return the array with the same data viewed with a different byte order.
`nonzero`()	Return the indices of the elements that are non-zero.
`partition`(kth[, axis, kind, order])	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.
`prod`([axis, dtype, out, keepdims, initial, ...])	Return the product of the array elements over the given axis
`ptp`([axis, out, keepdims])	Peak to peak (maximum - minimum) value along a given axis.
`put`(indices, values[, mode])	Set `a.flat[n] = values[n]` for all *n* in indices.
`ravel`([order])	Return a flattened array.
`repeat`(repeats[, axis])	Repeat elements of an array.
`reshape`(shape[, order])	Returns an array containing the same data with a new shape.
`resize`(new_shape[, refcheck])	Change shape and size of array in-place.
`round`([decimals, out])	Return *a* with each element rounded to the given number of decimals.
`searchsorted`(v[, side, sorter])	Find indices where elements of v should be inserted in a to maintain order.
`setfield`(val, dtype[, offset])	Put a value into a specified place in a field defined by a data-type.
`setflags`([write, align, uic])	Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY, respectively.
`sort`([axis, kind, order])	Sort an array in-place.
`squeeze`([axis])	Remove axes of length one from *a*.
`std`([axis, dtype, out, ddof, keepdims, where])	Returns the standard deviation of the array elements along given axis.
`sum`([axis, dtype, out, keepdims, initial, where])	Return the sum of the array elements over the given axis.

swapaxes(axis1, axis2)	Return a view of the array with *axis1* and *axis2* interchanged.
take(indices[, axis, out, mode])	Return an array formed from the elements of *a* at the given indices.
tobytes([order])	Construct Python bytes containing the raw data bytes in the array.
tofile(fid[, sep, format])	Write array to a file as text or binary (default).
tolist()	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
tostring([order])	A compatibility alias for **tobytes**, with exactly the same behavior.
trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
transpose(*axes)	Returns a view of the array with axes transposed.
var([axis, dtype, out, ddof, keepdims, where])	Returns the variance of the array elements, along given axis.
view([dtype][, type])	New view of array with the same data.

dot

Previous
The N-dimensional array (ndarray)

Next
numpy.ndarray.all