Array creation routines

See also

Array creation

From shape or value

`empty`(shape[, dtype, order, device, like])	Return a new array of given shape and type, without initializing entries.
`empty_like`(prototype[, dtype, order, subok, ...])	Return a new array with the same shape and type as a given array.
`eye`(N[, M, k, dtype, order, device, like])	Return a 2-D array with ones on the diagonal and zeros elsewhere.
`identity`(n[, dtype, like])	Return the identity array.
`ones`(shape[, dtype, order, device, like])	Return a new array of given shape and type, filled with ones.
`ones_like`(a[, dtype, order, subok, shape, ...])	Return an array of ones with the same shape and type as a given array.
`zeros`(shape[, dtype, order, like])	Return a new array of given shape and type, filled with zeros.
`zeros_like`(a[, dtype, order, subok, shape, ...])	Return an array of zeros with the same shape and type as a given array.
`full`(shape, fill_value[, dtype, order, ...])	Return a new array of given shape and type, filled with *fill_value*.
`full_like`(a, fill_value[, dtype, order, ...])	Return a full array with the same shape and type as a given array.

From existing data

`array`(object[, dtype, copy, order, subok, ...])	Create an array.
`asarray`(a[, dtype, order, device, copy, like])	Convert the input to an array.
`asanyarray`(a[, dtype, order, device, copy, like])	Convert the input to an ndarray, but pass ndarray subclasses through.
`ascontiguousarray`(a[, dtype, like])	Return a contiguous array (ndim >= 1) in memory (C order).
`asmatrix`(data[, dtype])	Interpret the input as a matrix.
`astype`(x, dtype, /, *[, copy, device])	Copies an array to a specified data type.
`copy`(a[, order, subok])	Return an array copy of the given object.
`frombuffer`(buffer[, dtype, count, offset, like])	Interpret a buffer as a 1-dimensional array.
`from_dlpack`(x, /, *[, device, copy])	Create a NumPy array from an object implementing the `__dlpack__` protocol.
`fromfile`(file[, dtype, count, sep, offset, like])	Construct an array from data in a text or binary file.
`fromfunction`(function, shape, *[, dtype, like])	Construct an array by executing a function over each coordinate.
`fromiter`(iter, dtype[, count, like])	Create a new 1-dimensional array from an iterable object.
`fromstring`(string[, dtype, count, like])	A new 1-D array initialized from text data in a string.
`loadtxt`(fname[, dtype, comments, delimiter, ...])	Load data from a text file.

Creating record arrays

> **Note**
>
> Please refer to Record arrays for record arrays.

`rec.array`(obj[, dtype, shape, offset, ...])	Construct a record array from a wide-variety of objects.
`rec.fromarrays`(arrayList[, dtype, shape, ...])	Create a record array from a (flat) list of arrays
`rec.fromrecords`(recList[, dtype, shape, ...])	Create a recarray from a list of records in text form.
`rec.fromstring`(datastring[, dtype, shape, ...])	Create a record array from binary data
`rec.fromfile`(fd[, dtype, shape, offset, ...])	Create an array from binary file data

Creating character arrays (`numpy.char`)

> **Note**
>
> `numpy.char` is used to create character arrays.

`char.array`(obj[, itemsize, copy, unicode, order])	Create a `chararray`.
`char.asarray`(obj[, itemsize, unicode, order])	Convert the input to a `chararray`, copying the data only if necessary.

Numerical ranges

`arange`([start,] stop[, step,][, dtype, ...])	Return evenly spaced values within a given interval.

`linspace`(start, stop[, num, endpoint, ...])	Return evenly spaced numbers over a specified interval.
`logspace`(start, stop[, num, endpoint, base, ...])	Return numbers spaced evenly on a log scale.
`geomspace`(start, stop[, num, endpoint, ...])	Return numbers spaced evenly on a log scale (a geometric progression).
`meshgrid`(*xi[, copy, sparse, indexing])	Return a tuple of coordinate matrices from coordinate vectors.
`mgrid`	An instance which returns a dense multi-dimensional "meshgrid".
`ogrid`	An instance which returns an open multi-dimensional "meshgrid".

Building matrices

`diag`(v[, k])	Extract a diagonal or construct a diagonal array.
`diagflat`(v[, k])	Create a two-dimensional array with the flattened input as a diagonal.
`tri`(N[, M, k, dtype, like])	An array with ones at and below the given diagonal and zeros elsewhere.
`tril`(m[, k])	Lower triangle of an array.
`triu`(m[, k])	Upper triangle of an array.
`vander`(x[, N, increasing])	Generate a Vandermonde matrix.

The matrix class

`bmat`(obj[, ldict, gdict])	Build a matrix object from a string, nested sequence, or array.