**What is numpy ?**

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

**Advantage of numpy over python list or array**

Numpy has below three major advantages-

**Size** - Numpy data structures take up less space

**Performance** – Numpy operation are faster b/c of less space consumption and vectorization.

**Functionality** - SciPy and NumPy have optimized functions such as linear algebra operations built in.

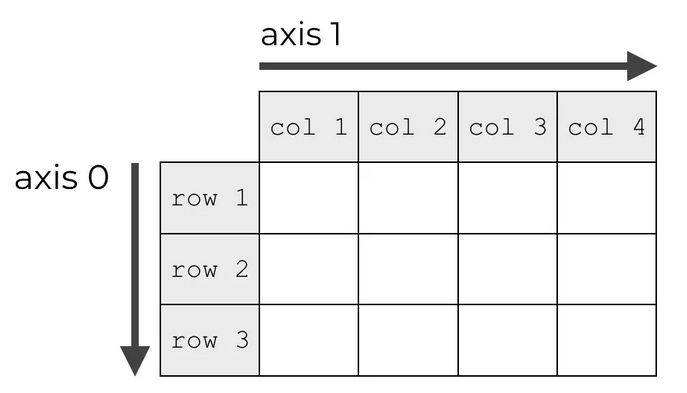
**Why numpy is faster**

Numpy is faster compare to list/array because numpy consumes less space and implements vectorization.

Axis convention in numpy:

Axis =0: operation will be done column wise

Axis=1: operation will be done row wise



**Example**:

Concatenate two array row wise.

>>> a = np.array([[1, 2], [3, 4]])

>>> a

array([[1, 2],

[3, 4]])

>>> b = np.array([[5, 6]])

>>> b

array([[5, 6]])

>>> np.concatenate((a, b), axis=0)

array([[1, 2],

[3, 4],

[5, 6]])

>>>

**Attributes of numpy**

|  |  |
| --- | --- |
| [ndarray.T](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.T.html#numpy.ndarray.T) | The transposed array. |
| [ndarray.real](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.real.html#numpy.ndarray.real) | The real part of the array. |
| [ndarray.imag](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.imag.html#numpy.ndarray.imag) | The imaginary part of the array. |
| [ndarray.flat](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.flat.html#numpy.ndarray.flat) | A 1-D iterator over the array. |
| [ndarray.ctypes](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.ctypes.html#numpy.ndarray.ctypes) | An object to simplify the interaction of the array with the ctypes module. |
| [ndarray.dtype](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.dtype.html#numpy.ndarray.dtype) | Data-type of the array’s elements. |

|  |  |
| --- | --- |
| [ndarray.flags](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.flags.html#numpy.ndarray.flags) | Information about the memory layout of the array. |
| [ndarray.shape](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.shape.html#numpy.ndarray.shape) | Tuple of array dimensions. |
| [ndarray.strides](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.strides.html#numpy.ndarray.strides) | Tuple of bytes to step in each dimension when traversing an array. |
| [ndarray.ndim](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.ndim.html#numpy.ndarray.ndim) | Number of array dimensions. |
| [ndarray.data](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.data.html#numpy.ndarray.data) | Python buffer object pointing to the start of the array’s data. |
| [ndarray.size](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.size.html#numpy.ndarray.size) | Number of elements in the array. |
| [ndarray.itemsize](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.itemsize.html#numpy.ndarray.itemsize) | Length of one array element in bytes. |
| [ndarray.nbytes](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.nbytes.html#numpy.ndarray.nbytes) | Total bytes consumed by the elements of the array. |
| [ndarray.base](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.base.html#numpy.ndarray.base) | Base object if memory is from some other object. |

#####################

# Numpy Array creation #

#####################

We can use generally below 5 ways to create array –

1. conversion from python iterable data types (list, tuples, etc)

np.array(pyth\_iterable\_data)

1. Intrinsic numpy array creation object- onez, zeros, arrange etc.

np.zeros(shape\_of\_array))

np.ones(shape\_of\_array))

|  |  |
| --- | --- |
| np.zeros((2,3)) | [[0. 0. 0.]  [0. 0. 0.]] |

1. Reading array from disk either from standar or custom.
2. Creating array from raw bytes through use of string and buffer.
3. Use of special library function.

#######################

# Array Creation Routine #

#######################

We have below methods for numpy array creation.

####################

# From zeros and ones #

####################

numpy.empty(shape, dtype=float, order='C')

Return a new array of given shape and type, without initializing entries.

Shape of the empty array, e.g., (2, 3) or 2, dtype and order are optional.

numpy.empty\_like(prototype\_array, dtype=None, order='K', subok=True, shape=None)

Return a new array with the same shape and type as a given array.

Except prototype\_array remaining all parameter are optional.

numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

Return a 2-D array with ones on the diagonal and zeros elsewhere.

N , M ---- Number or rows and columns. Except N all are optional.

>>> np.eye(2, dtype=int)

array([[1, 0],

[0, 1]])

numpy.identity(n, dtype=None)

Return the identity array.The identity array is a square array with ones on the main diagonal

n: --- Number or rows and column. Out array will be square array of n x n.

numpy.ones(shape, dtype=None, order='C')

Return a new array of given shape and type, filled with ones.

shape ----- It represents the shape of array, except shape all parameter are optional.

numpy.ones\_like(prototype\_array, dtype=None, order='K', subok=True, shape=None)

Return an array of ones with the same shape and type as a given prototype array.

Except prototype\_array all parameter are optional.

numpy.zeros(shape, dtype=float, order='C')

Return a new array of given shape and type, filled with zeros.

numpy.full(shape, fill\_value, dtype=None, order='C')

Return a new array of given shape and type, filled with fill\_value.

numpy.full\_like(prototype\_array, fill\_value, dtype=None, order='K', subok=True, shape=None)

Return a full array with the same shape and type as a given array.

##################

# From existing data #

##################

numpy.array(data, dtype=None, copy=True, order='K', subok=False, ndmin=0)

Creating an array from given data.

numpy.asarray(data, dtype=None, order=None)

Convert the input to an array.

data : 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.

Except data all are optional.

numpy.asmatrix(data, dtype=None)

Interpret the input as a matrix.

|  |  |
| --- | --- |
| [copy](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.copy.html#numpy.copy)(a[, order]) | Return an array copy of the given object. |
| [frombuffer](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.frombuffer.html#numpy.frombuffer)(buffer[, dtype, count, offset]) | Interpret a buffer as a 1-dimensional array. |
| [fromfile](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fromfile.html#numpy.fromfile)(file[, dtype, count, sep, offset]) | Construct an array from data in a text or binary file. |
| [fromfunction](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fromfunction.html#numpy.fromfunction)(function, shape, \\*\\*kwargs) | Construct an array by executing a function over each coordinate. |
| [fromiter](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fromiter.html#numpy.fromiter)(iterable, dtype[, count]) | Create a new 1-dimensional array from an iterable object. |
| [fromstring](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fromstring.html#numpy.fromstring)(string[, dtype, count, sep]) | A new 1-D array initialized from text data in a string. |
| [loadtxt](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.loadtxt.html#numpy.loadtxt)(fname[, dtype, comments, delimiter, …]) | Load data from a text file. |

#################

# Numerical ranges #

#################

numpy.arange([start, ]stop, [step, ]dtype=None)

Return evenly spaced values within a given interval.

Interval is half open interval [start,stop) i.e end value is not included in interval.

Stop ------ End value of interval.

Start ----- start value of interval, default =0

Step ----- Step size of interval, default=1.

Except stop all values are optional.

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

Return specified number of data points from give data.

start : Start value of the sequence, this is array like

stop : End value of the sequence

num : Number of data points, default=50

Except start and stop all are optional.

numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

Same as linspace() but here data are even spaced on log scale

Except start and stop all are optional.

numpy.geomspace(start, stop, num=50, endpoint=True, dtype=None, axis=0)

Return numbers spaced evenly on a log scale (a geometric progression).

Except start and stop all are optional.

#################

# Building matrices #

#################

|  |  |
| --- | --- |
| [diag](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.diag.html#numpy.diag)(v[, k]) | Extract a diagonal or construct a diagonal array. |
| [diagflat](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.diagflat.html#numpy.diagflat)(v[, k]) | Create a two-dimensional array with the flattened input as a diagonal. |
| [tri](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tri.html#numpy.tri)(N[, M, k, dtype]) | An array with ones at and below the given diagonal and zeros elsewhere. |
| [tril](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tril.html#numpy.tril)(m[, k]) | Lower triangle of an array. |
| [triu](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.triu.html#numpy.triu)(m[, k]) | Upper triangle of an array. |
| [vander](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.vander.html#numpy.vander)(x[, N, increasing]) | Generate a Vandermonde matrix. |

#########################

# Numpy Array Broadcasting #

#########################

Broadcasting is an operation of matching the dimensions of differently shaped arrays in order to be able to perform further operations on those arrays (eg per-element arithmetic).

**Rules of broadcasting**

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side
2. If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

**Question 1:**

Check if two array if shape (2,3) and (3) are broadcastable.

Shape(m)=(2,3)

Shape(n)=3

By rule 1: ----- pad 1 on the left with ones:

Shape(n)=(1,3)

By rule 2: ----- first dimension disagrees, so we stretch this dimension to match

Shape(n)=(2,3)

Finally shape(n)=(2,3) hence shape of both m and n are same so both are broadcastable.

**Question 2:**

Check if array m with shape (3,1) is brodcastable with n with shape(3)

By rule 1: shape(n)=(1,3) ------ padding

By rule 2: shape(n)=(3,3) ------ Stretching

By rule 2 on m: shape(m)=(3,3) ---- stretching on m

Now shape of n and m are (3,3) hence operation on both are bordcastable.

**Question 3:**

Check if below two array are broadcastable:

M = np.ones((3, 2))

a = np.arange(3)

Apply rule 1:

M.shape -> (3, 2)

a.shape -> (1, 3)

Apply rule 2:

M.shape -> (3, 2)

a.shape -> (3, 3)

shape is not matching after applying the rules so arrays are not broadcastable.

##########################

# Array manipulation routines #

##########################

numpy.reshape(arr, newshape, order='C')

Gives a new shape to an array without changing its data.

arr -------- Array to be reshaped

newshape ---- New shape of array. Total number of element must be matching with old shape.

order ----- Optional.

numpy.ravel(arr, order='C')

Return a contiguous flattened array. A 1-D array, containing the elements of the input (arr), is returned. A copy is made only if needed.

Order ------ this is optional.

ndarray.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

ndarray.flat[num] ---- Convert the ndarray into 1-d and return value from index num

array([[1, 0],

[0, 1]])

>>>

>>> x = np.arange(1, 7).reshape(2, 3)

>>> **x.flat[3]** #Convert into 1-d array and give the value at index 4.

4

>>> **type(x.flat)**

<class 'numpy.flatiter'>

>>>

ndarray.flatten(order='C')

Return a copy of the array collapsed into one dimension.

Order ---- this is optional.

**########################**

**# Transpose like operation #**

**########################**

|  |  |
| --- | --- |
| [moveaxis](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.moveaxis.html#numpy.moveaxis)(a, source, destination) | Move axes of an array to new positions. |
| [rollaxis](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.rollaxis.html#numpy.rollaxis)(a, axis[, start]) | Roll the specified axis backwards, until it lies in a given position. |
| [swapaxes](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.swapaxes.html#numpy.swapaxes)(a, axis1, axis2) | Interchange two axes of an array. |
| [ndarray.T](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ndarray.T.html#numpy.ndarray.T) | The transposed array. |
| [transpose](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.transpose.html#numpy.transpose)(a[, axes]) | Permute the dimensions of an array. |

#############################

# Changing number of dimensions #

#############################

numpy.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.

numpy.atleast\_2d(\*arys)

View inputs as arrays with at least two dimensions.

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 an array, or list of arrays, each with a.ndim >= 2.

x = np.arange(3.0)

np.atleast\_2d(x).base is x # True

numpy.atleast\_3d(\*arys)

View inputs as arrays with at least three dimensions.

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.

|  |  |
| --- | --- |
| [broadcast\_to](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.broadcast_to.html#numpy.broadcast_to)(array, shape[, subok]) | Broadcast an array to a new shape. |
| [broadcast\_arrays](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.broadcast_arrays.html#numpy.broadcast_arrays)(\\*args, \\*\\*kwargs) | Broadcast any number of arrays against each other. |
| [expand\_dims](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.expand_dims.html#numpy.expand_dims)(a, axis) | Expand the shape of an array. |
| [squeeze](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.squeeze.html#numpy.squeeze)(a[, axis]) | Remove single-dimensional entries from the shape of an array. |

#####################

# Changing kind of array #

#####################

numpy.asarray(arr, dtype=None, order=None)

Convert the input to an array.

arr: 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.

numpy.asanyarray(a, dtype=None, order=None)

Convert the input to an ndarray, but pass ndarray subclasses through.

arr: 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.

|  |  |
| --- | --- |
| [asmatrix](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.asmatrix.html#numpy.asmatrix)(data[, dtype]) | Interpret the input as a matrix. |
| [asfarray](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.asfarray.html#numpy.asfarray)(a[, dtype]) | Return an array converted to a float type. |
| [asfortranarray](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.asfortranarray.html#numpy.asfortranarray)(a[, dtype]) | Return an array (ndim >= 1) laid out in Fortran order in memory. |
| [ascontiguousarray](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ascontiguousarray.html#numpy.ascontiguousarray)(a[, dtype]) | Return a contiguous array (ndim >= 1) in memory (C order). |
| [asarray\_chkfinite](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.asarray_chkfinite.html#numpy.asarray_chkfinite)(a[, dtype, order]) | Convert the input to an array, checking for NaNs or Infs. |
| [asscalar](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.asscalar.html#numpy.asscalar)(a) | Convert an array of size 1 to its scalar equivalent. |
| [require](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.require.html#numpy.require)(a[, dtype, requirements]) | Return an ndarray of the provided type that satisfies requirements. |

##############

# Joining arrays #

##############

numpy.concatenate((a1, a2, ...), axis=0, out=None)

Join a sequence of arrays along an existing axis.

**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.

Axis =0: Means along row or for each column

Axis=1: Means along column or for each row

**Question1:**

Concatenate array a and b along the row.

a = np.array([[1, 2], [3, 4]])

b = np.array([[5, 6]])

**Answer:**

|  |  |
| --- | --- |
| 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]]) |

numpy.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.

**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.

It’s like concatenate but conceptually not clear on this method.

numpy.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.

**tup** : sequence of 1-D or 2-D arrays

Arrays to stack. All of them must have the same first dimension.

**stacked** : 2-D array

The array formed by stacking the given arrays.

**Note:**

It converts rows on both array into column and concatenate columns.

Question 1:

Stack array a and array b. a = np.array((1,2,3)), b = np.array((2,3,4)).

|  |  |
| --- | --- |
| a = np.array((1,2,3))  array([1, 2, 3])  b = np.array((2,3,4)) | np.column\_stack((a,b))  array([[1, 2],  [2, 3],  [3, 4]]) |

numpy.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).

|  |  |
| --- | --- |
| [dstack](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.dstack.html#numpy.dstack)(tup) | Stack arrays in sequence depth wise (along third axis). |
| [hstack](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.hstack.html#numpy.hstack)(tup) | Stack arrays in sequence horizontally (column wise). |
| [vstack](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.vstack.html#numpy.vstack)(tup) | Stack arrays in sequence vertically (row wise). |
| [block](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.block.html#numpy.block)(arrays) | Assemble an nd-array from nested lists of blocks. |

numpy.hstack(tup)

Stack arrays in sequence horizontally (column wise).

np.concatenate(arr,brr,axis=0)==np.hstack()

For hstack() shape and number of elements must be same.

**Example:** --------- Good

Show find the vstack and hstack for below arrays.

a = np.array((1,2,3))

b = np.array((2,3,4))

Answer:

|  |  |
| --- | --- |
| np.hstack((a,b))  array([1, 2, 3, 2, 3, 4]) | np.vstack((a,b))  array([[1, 2, 3],  [2, 3, 4]]) |

###############

# Splitting arrays #

###############

numpy.split(ary, indices\_or\_sections, axis=0)

Split an array into multiple sub-arrays and returns it.

**ary : ndarray**

Array to be divided into sub-arrays.

**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

If an index exceeds the dimension of the array along axis, an empty sub-array is returned correspondingly.

**Example 1: if indices is integer.**

x = np.arange(9.0)

np.split(x, 3) # Split in 3 equal parts--- [array([0., 1., 2.]), array([3., 4., 5.]), array([6., 7., 8.])]

**Example 2: If indices is list of number.**

x = np.arange(8.0)

np.split(x, [3, 5, 6, 10]) #Split at that indixes

[array([0., 1., 2.]), array([3., 4.]), array([5.]), array([6., 7.]), array([], dtype=float64)]

numpy.array\_split(ary, indices\_or\_sections, axis=0)

Split an array into multiple sub-arrays.

he 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.

Example:

Let take array – mp.array(8) and split it using split() and array\_split()

|  |  |
| --- | --- |
| x = np.arange(8.0)  np.split(x,3)  #this will give error b/c array will have total 8 elements and that can’t be divided in equal length of size 3. | x = np.arange(7.0)  np.array\_split(x, 3)  [array([0., 1., 2.]), array([3., 4.]), array([5., 6.])] |

|  |  |
| --- | --- |
| [dsplit](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.dsplit.html#numpy.dsplit)(ary, indices\_or\_sections) | Split array into multiple sub-arrays along the 3rd axis (depth). |
| [hsplit](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.hsplit.html#numpy.hsplit)(ary, indices\_or\_sections) | Split an array into multiple sub-arrays horizontally (column-wise). |
| [vsplit](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.vsplit.html#numpy.vsplit)(ary, indices\_or\_sections) | Split an array into multiple sub-arrays vertically (row-wise). |

**######################**

**# Tiling/Repeating arrays #**

**######################**

numpy.repeat(a, repeats, axis=None)

Repeat elements of an array.

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.

Example:

>>> 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]])

|  |  |
| --- | --- |
| [tile](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tile.html#numpy.tile)(A, reps) | Construct an array by repeating A the number of times given by reps. |

**#############################**

**# Adding and removing elements #**

**#############################**

|  |  |
| --- | --- |
| [delete](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.delete.html#numpy.delete)(arr, obj[, axis]) | Return a new array with sub-arrays along an axis deleted. |
| [insert](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.insert.html#numpy.insert)(arr, obj, values[, axis]) | Insert values along the given axis before the given indices. |
| [append](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.append.html#numpy.append)(arr, values[, axis]) | Append values to the end of an array. |
| [resize](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.resize.html#numpy.resize)(a, new\_shape) | Return a new array with the specified shape. |
| [trim\_zeros](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.trim_zeros.html#numpy.trim_zeros)(filt[, trim]) | Trim the leading and/or trailing zeros from a 1-D array or sequence. |
| [unique](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.unique.html#numpy.unique)(ar[, return\_index, return\_inverse, …]) | Find the unique elements of an array. |

###########################

# Reshaping/Rotating the array #

###########################

|  |  |
| --- | --- |
| [rot90](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.rot90.html#numpy.rot90)(m[, k, axes]) | Rotate an array by 90 degrees in the plane specified by axes. |
| [reshape](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.reshape.html#numpy.reshape)(a, newshape[, order]) | Gives a new shape to an array without changing its data. |

>>> m = np.array([[1,2],[3,4]], int)

>>> m

array([[1, 2],

[3, 4]])

>>> np.rot90(m)

array([[2, 4],

[1, 3]])

**##############################**

**# Sorting, searching, and counting #**

**##############################**