**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.

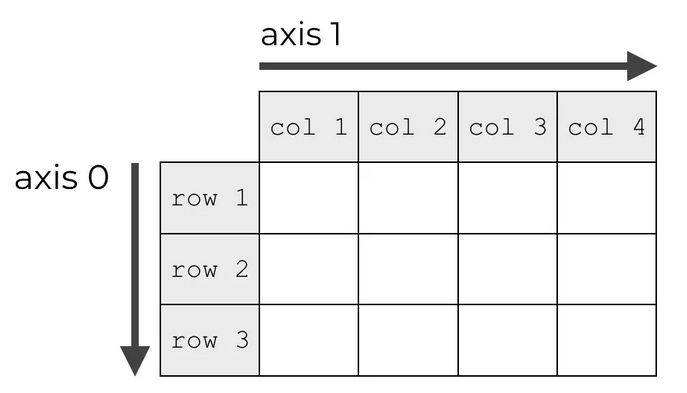
**List vs Numpy Array**

Python lists store pointers to memory locations. On the other hand, numpy arrays are typed, where the default type is floating point. Because of this, the system knows how much memory to allocate, and if you ask for an array of size 100, it will allocate one hundred contiguous spots in memory, where the size of each spot is based on the type. This makes access extremely fast.

**Axis convention in numpy:**

Axis =0: operation will be row wise for each column

Axis=1: operation will be column wise for each row



**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- ones, 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 standard 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. (Start, Stop and Step)

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, stop, num. of data point)

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.

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

**# Indexing, Slicing in numpy array #**

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

Concept of slicing in numpy is same as python slicing as for each diamension. We can generalize it as-

arr[ diam\_1\_start : diam\_1\_end : setp, diam\_2\_start : diam\_2\_end : end , . . . ]

**Parameter:**

We have below three parameter **and all three are optional.**

* start --- default to 0
* end --- default to end of array
* step – default to 1

**#slicing 1 -d array #**

* arr[ n1 : : ] ---- slice the all array value starting from n1
* arr[ n1 : n1 : ] --- slice the array value starting from n1 to n2-1 index with step 1

Let say we have below array –

arr = np.array([3, 5, 7, 9, 11, 15, 18, 22])

# Example 1: Use slicing a 1D arrays

arr2 = arr[1:6]

# Example 2: Slice Starting from 3rd value to end

arr2 = arr[3:]

# Example 3: Slice 0 to 4 index

arr2 = arr[:5]

# Example 4: Use negative slicing

arr2 = arr[-4:-2]

# Example 5: Use step value

arr2 = arr[::3]

**# Slicing 2-D array #**

arr[ diam\_1\_start: diam\_2\_end : step2 , diam\_2\_start : diam\_2\_end : step2 ]

**Example:**

arr = np.array([[3, 5, 7, 9, 11], [2, 4, 6, 8, 10]])

arr2 = arr[1:,1:3]

print(arr2) # [[4 6]]

**#Slicing 3-D Array #**

Slicing of 3-D array is same as 2-D or 1-D just we will have to take care of third dimension.

**Example:**

arr = np.array([[[3, 5, 7, 9, 11],

[2, 4, 6, 8, 10]],

[[5, 7, 8, 9, 2],

[7, 2, 3, 6, 7]]])

arr2 = arr[0,1,0:2]

print(arr2) #[2 4]

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

**# Getting particular Row or Column data ##** --- 2-D Array

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

For getting row or column data from a array we need to use indexers in numpy-

arr[ [row1, row2, . . . rown] , [col1, col2, . . . coln] ] **---- getting given row and column data**

arr [ [row1, row2, . . . rown], : ] **---- getting given rows and all columns data**

arr[ : , [ col1, col2, . . . . coln] ] **---- getting all columns and given rows data**

Example:1 --- Get second row and second column data

Solution : print(arr[[1],[1]])

Example 2: Get the first column data

Solution: print(arr[:,[0]])

Example 3: get the second-row data

Solution: print( arr[ [1],: ]

**Question:**

Swap the first column with second column data.

**Solution:**

arr[ :,[0] ] = arr[:, [1] ]

Question:

Swap the first and second column data

Solution:

arr[ :,[0] ],arr[:, [1] ] = arr[:, [1] ],arr[ :,[0] ]

**Note:**

One thing we should keep in mind that shape of result also depends on the shape of input indexing.

Let say we have below 1-D array-

arr= [51 92 14 71 60 20 82 86 74 74]

when we pass indexes as list of list.

ind = np.array([[3, 7],

[4, 5]])

x[ind]

output:

array([[71, 86],

[60, 20]])

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

**# Filtering / conditional filtering of data #**

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

Applying any condition to any array is same as applying condition to dataframe.

In dataframe we have column name to we can apply condition on one/more specific columns but in array condition will be applied on all data.

& ---- AND operation while applying multiple condition

| ---- OR operation while applying multiple condition

~ ---- NOT operation while applying multiple condition

**arr condition\_operation** ----- this will always give a Boolean array of same shape of arr

Let say we have below array-

[[5 0 3 3]

[7 9 3 5]

[2 4 7 6]]

arr>6 will give Boolean array where array values are greater than 6 of same shape of arr

|  |  |
| --- | --- |
| arr is:-  [[5 0 3 3]  [7 9 3 5]  [2 4 7 6]]  print([arr>6]) | #print( arr>6 )  [array([[False, False, False, False],  [ **True**, **True**, False, False],  [False, False, **True**, False]])] |

**arr[ Boolean\_array ]** ----- this will given all value of arrary for which Boolean array values are TRUE.

Shape of Boolean array must be same as shape or array

**Example:**

Find all elements from array which are greater than 10.

**Solution:**

arr[ ar>10 ]

**Example 2:** Find all values which are greater than 4 and less than 100.

**Solution:**

arr[ ar>10 & arr<100 ]

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

# 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.diag(v, k=0)

**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.

**Note:**

If V is any array, then it will extract the diagonal from that array and create a matrix. Where diagonal element will be diagonal element of V and 0 else where.

**Example:**

|  |  |
| --- | --- |
| a = np.arange(12).reshape((4,3))  print(np.diag(np.diag(a)))  diag() have extracted and from a and assigned them as diagonal to new matrix/array | array([[0, 0, 0],  [0, 4, 0],  [0, 0, 8]]) |

**Example 2:**

np.diag([ [4, 5], [6, 8] ] ) # Error because input is not 1-d. If input if not 1-d then use diagflat()

numpy.diagflat(v, k=0)

**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.

**Note:**

If V is not a 1-d list or array then it flattens those value into 1-d and creates matrix. Values of flattened array will be diagonal value of new matrix created and 0 else where

|  |  |
| --- | --- |
| x = np.diagflat([ [4, 5], [6, 8] ] )  Input is not 1-D, in output it’s is flattened and created as diagonal | [[4 0 0 0]  [0 5 0 0]  [0 0 6 0]  [0 0 0 8]] |

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

# 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: Index, Means along row or for each column

Axis=1: Column, Means along column or for each row

**Note:**

We can achieve concatenate function behaviour using append function also. Both works same.

**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]]) |

**np.append(arr, values, axis=None)**

Append values to the end of an array.

Parameters are:

**arr** : array\_like

**obj** : slice, int or array of ints

0 -- means first row or column (v=based on axis)

1-- means second row or column (based on axis )

2 -- means third row or column (based on axis ) and so on

**axis** = **{0 means delete the row, 1 means delete the column }**

**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.

**Note:**

For if we are trying to append array along any dimension then **in that dimension shape of another array must be same.** i.e-

* If adding along array row then column of new array must be same
* If adding along the column then row of new array must be same.

Example:

Let say we have below array-

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

**Question:**

Add a row into this array.

**Solution:**

If want to add row then array/values to be added must have same number of column.

arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

print(np.append( arr,[[100, 200, 300, 400]],axis=0 ) )

|  |  |
| --- | --- |
| Output:  [[ 1 2 3 4]  [ 5 6 7 8]  [ 9 10 11 12]  [100 200 300 400]] | New row is added at end and it’s end at end of array. |

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=1**)==**np.hstack()**

np.concatenate((arr,brr),**axis=0**)==**np.vstack()**

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])  arr=np.array([[ 1, 2, 3], [ -1, -2, -3]] )  arr1=np.array([[ 4, 5, 6], [ -4, -5, -6]] )  print(np.hstack((arr,arr1)))  [[ 1 2 3 4 5 6]  [-1 -2 -3 -4 -5 -6]] | np.vstack((a,b))  array([[1, 2, 3],  [2, 3, 4]])  arr=np.array([[ 1, 2, 3], [ -1, -2, -3]] )  arr1=np.array([[ 4, 5, 6], [ -4, -5, -6]] )  print(np.vstack((arr,arr1)))  [[ 1 2 3]  [-1 -2 -3]  [ 4 5 6]  [-4 -5 -6]] |

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

# 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. ( split range goes upto num-1 )

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.

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.

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. |

**np.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 are:

**arr** : array\_like

**obj** : slice, int or array of ints

0 -- means first row or column (v=based on axis)

1-- means second row or column (based on axis )

2 -- means third row or column (based on axis ) and so on

**axis** = {0 means delete the row, 1 means delete the column }

**Note:**

In delete **axis = 0 points to deleting the column and 1 deleting rows**.

**np.insert(arr, obj, values, 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 are:

**arr** : array\_like

**obj** : slice, int or array of ints

0 -- means first row or column (v=based on axis)

1-- means second row or column (based on axis )

2 -- means third row or column (based on axis ) and so on

**axis** = {0 means delete the row, 1 means delete the column }

**values** : Values to be inserted into array

**Note:**

In delete **axis = 0 points to deleting the column and 1 deleting rows**.

Example:

Let say we have below array.

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

**Question:**

Add 100 as values to second row of this array.

**Solution:**

arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

print(np.insert(arr,1,100,axis=0))

[[ 1 2 3 4]

[100 100 100 100]

[ 5 6 7 8]

[ 9 10 11 12]]

**np.append(arr, values, axis=None)**

Append values to the end of an array.

Parameters are:

**arr** : array\_like

**obj** : slice, int or array of ints

0 -- means first row or column (v=based on axis)

1-- means second row or column (based on axis )

2 -- means third row or column (based on axis ) and so on

**axis** = **{0 means delete the row, 1 means delete the column }**

**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.

**Note:**

For if we are trying to append array along any dimension then **in that dimension shape of another array must be same.** i.e-

* If adding along array row then column of new array must be same
* If adding along the column then row of new array must be same.

Example:

Let say we have below array-

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]]

**Question:**

Add a row into this array.

**Solution:**

If want to add row then array/values to be added must have same number of column.

arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

print(np.append( arr,[[100, 200, 300, 400]],axis=0 ) )

|  |  |
| --- | --- |
| Output:  [[ 1 2 3 4]  [ 5 6 7 8]  [ 9 10 11 12]  [100 200 300 400]] | New row is added at end and it’s end at end of array. |

**Note:**

We can achieve the append function behaviour by using concatenate(). Both work exactly same except way of using two array in both functions.

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: Index, Means along row or for each column

Axis=1: Column, Means along column or for each row

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

# 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 #**

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

numpy.sort(a, axis=-1, kind=None, order=None)

Return a sorted copy of an array

**Order:** When a is an array with fields defined, this argument specifies which fields to compare first, second, etc

**axis** : int. None, default = -1

Axis along which sorting will be performed. -1 meas last axis

If None then it will flatten into 1-D and then will sort.

{ 0 – means each column data, 1 mean each row data }

**Question: ---------- Impt.**

Sort the below structured array based on height.

data\_type = [('name', 'S15'), ('class', int), ('height', float)]

students\_details = [('James', 5, 48.5), ('Nail', 6, 52.5),('Paul', 5, 42.10), ('Pit', 5, 40.11)]

# create a structured array

students = np.array(students\_details, dtype=data\_type)

**Answer:**

students.sort(order='height')

print(students)

let say we have below array-

[[5 0 3 3]

[7 9 3 5]

[2 4 7 6]]

**Example**

|  |  |
| --- | --- |
| # [[5,0,3,3],[7,9,3,5],[2,4,7,6]]  sort( arr, **axis = 0** )  [[2 0 3 3]  [5 4 3 5]  [7 9 7 6]]  Data is sorted for each column | # [[5,0,3,3],[7,9,3,5],[2,4,7,6]]  sort( arr, **axis = 1** )  [[0 3 3 5]  [3 5 7 9]  [2 4 6 7]]  Data is sorted for each row |

numpy.lexsort(keys, axis=-1)

Perform an indirect stable sort using a sequence of keys.

Returns array of indices that sort the keys along the specified axis.

**keys :** (k, N) array or tuple containing k (N,)-shaped sequences. First sort by N then by K

axis=int, optional.

**Example:**

Do the lexsort for below two arrays.

a = [1,5,1,4,3,4,4] # First column

b = [9,4,0,4,0,2,1] # Second column

**Answer:**

print(np.lexsort((b,a))) #[2 0 4 6 5 3 1]

numpy.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.

Example:

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

x.argsort() #array(**[1, 2, 0]**, dtype=int64)

If we fetch the data from above indices/output array then data would come in sorted array.

ndarray.sort(axis=-1, kind=None, order=None)

Sort an array in-place. It sorts the array and returns none, result of sorting array are stored in same variable.

|  |  |
| --- | --- |
| [msort](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.msort.html#numpy.msort)(a) | Return a copy of an array sorted along the first axis. |
| [sort\_complex](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.sort_complex.html#numpy.sort_complex)(a) | Sort a complex array using the real part first, then the imaginary part. |
| [partition](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.partition.html#numpy.partition)(a, kth[, axis, kind, order]) | Return a partitioned copy of an array. |
| [argpartition](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.argpartition.html#numpy.argpartition)(a, kth[, axis, kind, order]) | Perform an indirect partition along the given axis using the algorithm specified by the *kind* keyword. |

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

**# Searching #**

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

|  |  |
| --- | --- |
| [argmax](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.argmax.html#numpy.argmax)(a[, axis, out]) | Returns the indices of the maximum values along an axis. |
| [nanargmax](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.nanargmax.html#numpy.nanargmax)(a[, axis]) | Return the indices of the maximum values in the specified axis ignoring NaNs. |
| [argmin](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.argmin.html#numpy.argmin)(a[, axis, out]) | Returns the indices of the minimum values along an axis. |
| [nanargmin](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.nanargmin.html#numpy.nanargmin)(a[, axis]) | Return the indices of the minimum values in the specified axis ignoring NaNs. |
| [argwhere](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.argwhere.html#numpy.argwhere)(a) | Find the indices of array elements that are non-zero, grouped by element. |
| [nonzero](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.nonzero.html#numpy.nonzero)(a) | Return the indices of the elements that are non-zero. |
| [flatnonzero](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.flatnonzero.html#numpy.flatnonzero)(a) | Return indices that are non-zero in the flattened version of a. |
| [where](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.where.html#numpy.where)(condition, [x, y]) | Return elements chosen from *x* or *y* depending on *condition*. |
| [searchsorted](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.searchsorted.html#numpy.searchsorted)(a, v[, side, sorter]) | Find indices where elements should be inserted to maintain order. |
| [extract](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.extract.html#numpy.extract)(condition, arr) | Return the elements of an array that satisfy some condition. |

###########

# Counting #

###########

|  |  |
| --- | --- |
| [count\_nonzero](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.count_nonzero.html#numpy.count_nonzero)(a[, axis]) | Counts the number of non-zero values in the array a |

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

# Linear algebra #

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

We have below methods for linear algebra, all these methods are – 'numpy.linalg' module.

numpy.linalg.det(a)

Compute the determinant of an array.

a: (...,M,M) Square array

Example:

>>> np.linalg.det(np.array([[1, 2], [3, 4]]))

-2.0000000000000004

numpy.linalg.eig(a)

Compute the eigenvalues and right eigenvectors of a square array.

a : (…, M, M) array

Matrices for which the eigenvalues and right eigenvectors will be computed.

numpy.linalg.inv(a)

Compute the (multiplicative) inverse of a matrix.

Given a square matrix a, return the matrix ainv satisfying

dot(a, ainv) = dot(ainv, a) = eye(a.shape[0]).

a:(…, M, M) array\_like, Matrix to be inverted

|  |  |
| --- | --- |
| [norm](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.norm.html#numpy.linalg.norm)(x[, ord, axis, keepdims]) | Matrix or vector norm. |
| [pinv](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.pinv.html#numpy.linalg.pinv)(a[, rcond, hermitian]) | Compute the (Moore-Penrose) pseudo-inverse of a matrix. |
| [solve](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve)(a, b) | Solve a linear matrix equation, or system of linear scalar equations. |
| [svd](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.svd.html#numpy.linalg.svd)(a[, full\_matrices, compute\_uv, hermitian]) | Singular Value Decomposition. |
| [eig](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eig.html#numpy.linalg.eig)(a) | Compute the eigenvalues and right eigenvectors of a square array. |
| [eigh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh)(a[, UPLO]) | Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix. |
| [eigvals](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigvals.html#numpy.linalg.eigvals)(a) | Compute the eigenvalues of a general matrix. |
| [eigvalsh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigvalsh.html#numpy.linalg.eigvalsh)(a[, UPLO]) | Compute the eigenvalues of a complex Hermitian or real symmetric matrix. |

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

# Eigen Values #

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

|  |  |
| --- | --- |
| [linalg.eig](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eig.html#numpy.linalg.eig)(a) | Compute the eigenvalues and right eigenvectors of a square array. |
| [linalg.eigh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh)(a[, UPLO]) | Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix. |
| [linalg.eigvals](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigvals.html#numpy.linalg.eigvals)(a) | Compute the eigenvalues of a general matrix. |
| [linalg.eigvalsh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.eigvalsh.html#numpy.linalg.eigvalsh)(a[, UPLO]) | Compute the eigenvalues of a complex Hermitian or real symmetric matrix. |

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

# Matrix and vector products #

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

|  |  |
| --- | --- |
| [dot](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.dot.html#numpy.dot)(a, b[, out]) | Dot product of two arrays. |
| [linalg.multi\_dot](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.multi_dot.html#numpy.linalg.multi_dot)(arrays) | Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order. |
| [vdot](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.vdot.html#numpy.vdot)(a, b) | Return the dot product of two vectors. |
| [inner](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.inner.html#numpy.inner)(a, b) | Inner product of two arrays. |
| [outer](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.outer.html#numpy.outer)(a, b[, out]) | Compute the outer product of two vectors. |
| [matmul](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.matmul.html#numpy.matmul)(x1, x2, /[, out, casting, order, …]) | Matrix product of two arrays. |
| [tensordot](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tensordot.html#numpy.tensordot)(a, b[, axes]) | Compute tensor dot product along specified axes. |
| [einsum](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.einsum.html#numpy.einsum)(subscripts, \*operands[, out, dtype, …]) | Evaluates the Einstein summation convention on the operands. |
| [einsum\_path](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.einsum_path.html#numpy.einsum_path)(subscripts, \*operands[, optimize]) | Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays. |
| [linalg.matrix\_power](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.linalg.matrix_power.html#numpy.linalg.matrix_power)(a, n) | Raise a square matrix to the (integer) power *n*. |
| [kron](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.kron.html#numpy.kron)(a, b) | Kronecker product of two arrays. |

**dot() vs vdot()**

If input arrays are complex then vdot() use complex conjugate of first array(first input array) then calculates the dot(). Dot() doesn’t use complex conjugate of any array though they are complex.

**Example:**

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

b = np.array([5+6j,7+8j])

c=np.array([1-2j,3-4j]) #com. Conj(a)

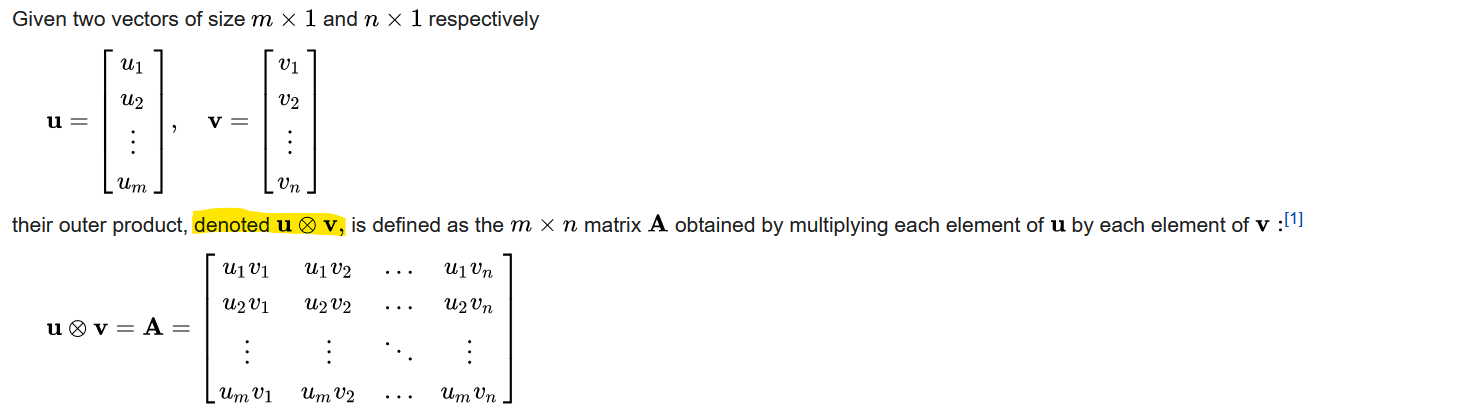
print(np.dot(a,b)) #(-18+68j)

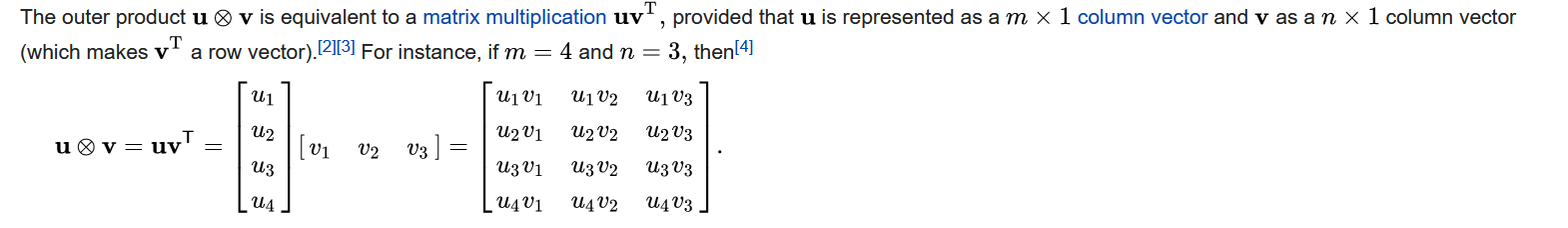
print(np.dot(c,b)) #(70-8j)

print(np.vdot(a,b)) #(70-8j)

**np.vdot(a,b)== np.dot(c,b)** ---- because it c is complex of a and vdot will use complex of a

**Outer product**





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

**# Discrete FFT Computation #**

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

We have below methods in numpy for discrete Fourier Transform calucation.

numpy.fft.fft(a, n=None, axis=-1, norm=None)

Compute the one-dimensional discrete Fourier Transform.

This function computes the one-dimensional n-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].

**a:** array\_like

**n :** int, optional, Length of the transformed axis of the output.

numpy.fft.fft2(a, s=None, axes=(-2, -1), norm=None)

Compute the 2-dimensional discrete Fourier Transform

numpy.fft.fftn(a, s=None, axes=None, norm=None)

Compute the N-dimensional discrete Fourier Transform.

This function computes the N-dimensional discrete Fourier Transform over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT).

|  |  |
| --- | --- |
| [ifft](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.ifft.html#numpy.fft.ifft)(a[, n, axis, norm]) | Compute the one-dimensional inverse discrete Fourier Transform. |
| [ifft2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.ifft2.html#numpy.fft.ifft2)(a[, s, axes, norm]) | Compute the 2-dimensional inverse discrete Fourier Transform. |
| [ifftn](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.ifftn.html#numpy.fft.ifftn)(a[, s, axes, norm]) | Compute the N-dimensional inverse discrete Fourier Transform. |

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

**# Real FFT computation #**

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

|  |  |
| --- | --- |
| [rfft](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.rfft.html#numpy.fft.rfft)(a[, n, axis, norm]) | Compute the one-dimensional discrete Fourier Transform for real input. |
| [irfft](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.irfft.html#numpy.fft.irfft)(a[, n, axis, norm]) | Compute the inverse of the n-point DFT for real input. |
| [rfft2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.rfft2.html#numpy.fft.rfft2)(a[, s, axes, norm]) | Compute the 2-dimensional FFT of a real array. |
| [irfft2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.irfft2.html#numpy.fft.irfft2)(a[, s, axes, norm]) | Compute the 2-dimensional inverse FFT of a real array. |
| [rfftn](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.rfftn.html#numpy.fft.rfftn)(a[, s, axes, norm]) | Compute the N-dimensional discrete Fourier Transform for real input. |
| [irfftn](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.irfftn.html#numpy.fft.irfftn)(a[, s, axes, norm]) | Compute the inverse of the N-dimensional FFT of real input. |

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

# Hermitian FFT calculation #

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

|  |  |
| --- | --- |
| [hfft](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.hfft.html#numpy.fft.hfft)(a[, n, axis, norm]) | Compute the FFT of a signal that has Hermitian symmetry, i.e., a real spectrum. |
| [ihfft](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.ihfft.html#numpy.fft.ihfft)(a[, n, axis, norm]) | Compute the inverse FFT of a signal that has Hermitian symmetry. |

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

# Helper Routines #

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

|  |  |
| --- | --- |
| [fftfreq](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.fftfreq.html#numpy.fft.fftfreq)(n[, d]) | Return the Discrete Fourier Transform sample frequencies. |
| [rfftfreq](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.rfftfreq.html#numpy.fft.rfftfreq)(n[, d]) | Return the Discrete Fourier Transform sample frequencies (for usage with rfft, irfft). |
| [fftshift](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.fftshift.html#numpy.fft.fftshift)(x[, axes]) | Shift the zero-frequency component to the center of the spectrum. |
| [ifftshift](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.ifftshift.html#numpy.fft.ifftshift)(x[, axes]) | The inverse of [fftshift](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fft.fftshift.html#numpy.fft.fftshift). |

**################################################################################# Trigonometric function #**

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

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

**# Rounding #**

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

Numpy have below methods for precision handling/rounding the decimal numbers.

|  |  |
| --- | --- |
| [around](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.around.html#numpy.around)(a[, decimals, out]) | Evenly round to the given number of decimals. |
| [round\_](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.round_.html#numpy.round_)(a[, decimals, out]) | Round an array to the given number of decimals. |
| [rint](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.rint.html#numpy.rint)(x, /[, out, where, casting, order, …]) | Round elements of the array to the nearest integer. |
| [fix](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fix.html#numpy.fix)(x[, out]) | Round to nearest integer towards zero. |
| [floor](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.floor.html#numpy.floor)(x, /[, out, where, casting, order, …]) | Return the floor of the input, element-wise. |
| [ceil](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.ceil.html#numpy.ceil)(x, /[, out, where, casting, order, …]) | Return the ceiling of the input, element-wise. |
| [trunc](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.trunc.html#numpy.trunc)(x, /[, out, where, casting, order, …]) | Return the truncated value of the input, element-wise. |

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

# Trigonometric function #

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

|  |  |
| --- | --- |
| [sin](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.sin.html#numpy.sin)(x, /[, out, where, casting, order, …]) | Trigonometric sine, element-wise. |
| [cos](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.cos.html#numpy.cos)(x, /[, out, where, casting, order, …]) | Cosine element-wise. |
| [tan](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tan.html#numpy.tan)(x, /[, out, where, casting, order, …]) | Compute tangent element-wise. |
| [arcsin](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arcsin.html#numpy.arcsin)(x, /[, out, where, casting, order, …]) | Inverse sine, element-wise. |
| [arccos](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arccos.html#numpy.arccos)(x, /[, out, where, casting, order, …]) | Trigonometric inverse cosine, element-wise. |
| [arctan](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arctan.html#numpy.arctan)(x, /[, out, where, casting, order, …]) | Trigonometric inverse tangent, element-wise. |
| [hypot](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.hypot.html#numpy.hypot)(x1, x2, /[, out, where, casting, …]) | Given the “legs” of a right triangle, return its hypotenuse. |
| [arctan2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arctan2.html#numpy.arctan2)(x1, x2, /[, out, where, casting, …]) | Element-wise arc tangent of x1/x2 choosing the quadrant correctly. |
| [degrees](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.degrees.html#numpy.degrees)(x, /[, out, where, casting, order, …]) | Convert angles from radians to degrees. |
| [radians](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.radians.html#numpy.radians)(x, /[, out, where, casting, order, …]) | Convert angles from degrees to radians. |
| [unwrap](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.unwrap.html#numpy.unwrap)(p[, discont, axis]) | Unwrap by changing deltas between values to 2\*pi complement. |
| [deg2rad](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.deg2rad.html#numpy.deg2rad)(x, /[, out, where, casting, order, …]) | Convert angles from degrees to radians. |
| [rad2deg](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.rad2deg.html#numpy.rad2deg)(x, /[, out, where, casting, order, …]) | Convert angles from radians to degrees. |

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

# Hyperbolic functions #

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

|  |  |
| --- | --- |
| [sinh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.sinh.html#numpy.sinh)(x, /[, out, where, casting, order, …]) | Hyperbolic sine, element-wise. |
| [cosh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.cosh.html#numpy.cosh)(x, /[, out, where, casting, order, …]) | Hyperbolic cosine, element-wise. |
| [tanh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.tanh.html#numpy.tanh)(x, /[, out, where, casting, order, …]) | Compute hyperbolic tangent element-wise. |
| [arcsinh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arcsinh.html#numpy.arcsinh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic sine element-wise. |
| [arccosh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arccosh.html#numpy.arccosh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic cosine, element-wise. |
| [arctanh](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.arctanh.html#numpy.arctanh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic tangent element-wise. |

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

# Exponents and logarithms #

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

|  |  |
| --- | --- |
| [exp](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.exp.html#numpy.exp)(x, /[, out, where, casting, order, …]) | Calculate the exponential of all elements in the input array. |
| [expm1](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.expm1.html#numpy.expm1)(x, /[, out, where, casting, order, …]) | Calculate exp(x) - 1 for all elements in the array. |
| [exp2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.exp2.html#numpy.exp2)(x, /[, out, where, casting, order, …]) | Calculate *2\*\*p* for all *p* in the input array. |
| [log](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.log.html#numpy.log)(x, /[, out, where, casting, order, …]) | Natural logarithm, element-wise. |
| [log10](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.log10.html#numpy.log10)(x, /[, out, where, casting, order, …]) | Return the base 10 logarithm of the input array, element-wise. |
| [log2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.log2.html#numpy.log2)(x, /[, out, where, casting, order, …]) | Base-2 logarithm of *x*. |
| [log1p](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.log1p.html#numpy.log1p)(x, /[, out, where, casting, order, …]) | Return the natural logarithm of one plus the input array, element-wise. |
| [logaddexp](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logaddexp.html#numpy.logaddexp)(x1, x2, /[, out, where, casting, …]) | Logarithm of the sum of exponentiations() of the inputs.  log(exp(x1) + exp(x2)) |
| [logaddexp2](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logaddexp2.html#numpy.logaddexp2)(x1, x2, /[, out, where, casting, …]) | Logarithm of the sum of exponentiations of the inputs in base-2.  log(exp(x1) + exp(x2)) |

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

# Numpy Set Routine #

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

numpy.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 --- return\_index
* the indices of the unique array that reconstruct the input array --- return\_inverse
* the number of times each unique value comes up in the input array – return\_count

**ar : array\_like**

**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.

|  |  |
| --- | --- |
|  | **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.  New in version 1.9.0. |

|  |  |
| --- | --- |
| [in1d](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.in1d.html#numpy.in1d)(ar1, ar2[, assume\_unique, invert]) | Test whether each element of a 1-D array is also present in a second array. |
| [intersect1d](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.intersect1d.html#numpy.intersect1d)(ar1, ar2[, assume\_unique, …]) | Find the intersection/common element of two arrays. |
| [isin](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isin.html#numpy.isin)(element, test\_elements[, …]) | Calculates *element in test\_elements*, broadcasting over *element* only. |
| [setdiff1d](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.setdiff1d.html#numpy.setdiff1d)(ar1, ar2[, assume\_unique]) | Find the set difference of two arrays. |
| [setxor1d](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.setxor1d.html#numpy.setxor1d)(ar1, ar2[, assume\_unique]) | Find the set exclusive-or of two arrays. |
| [union1d](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.union1d.html#numpy.union1d)(ar1, ar2) | Find the union of two arrays. |

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

**# Logic functions #**

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

**# Truth value testing#**

|  |  |
| --- | --- |
| [all](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.all.html#numpy.all)(a[, axis, out, keepdims]) | Test whether all array elements along a given axis evaluate to True. |
| [any](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.any.html#numpy.any)(a[, axis, out, keepdims]) | Test whether any array element along a given axis evaluates to True. |

**# Array contents #**

|  |  |
| --- | --- |
| [isfinite](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isfinite.html#numpy.isfinite)(x, /[, out, where, casting, order, …]) | Test element-wise for finiteness (not infinity or not Not a Number). |
| [isinf](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isinf.html#numpy.isinf)(x, /[, out, where, casting, order, …]) | Test element-wise for positive or negative infinity. |
| [isnan](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isnan.html#numpy.isnan)(x, /[, out, where, casting, order, …]) | Test element-wise for NaN and return result as a boolean array. |
| [isnat](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isnat.html#numpy.isnat)(x, /[, out, where, casting, order, …]) | Test element-wise for NaT (not a time) and return result as a boolean array. |
| [isneginf](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isneginf.html#numpy.isneginf)(x[, out]) | Test element-wise for negative infinity, return result as bool array. |
| [isposinf](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isposinf.html#numpy.isposinf)(x[, out]) | Test element-wise for positive infinity, return result as bool array. |

**np.isnan(x, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj])**

this function checks if each element of array x is nan/null values and return Boolean array.

**Example:**

Check if element or array are nan/missing in array and print them.

**Solution:**

arr[ np.isnan(arr) ]

**# Array type testing #**

|  |  |
| --- | --- |
| [iscomplex](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.iscomplex.html#numpy.iscomplex)(x) | Returns a bool array, where True if input element is complex. |
| [iscomplexobj](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.iscomplexobj.html#numpy.iscomplexobj)(x) | Check for a complex type or an array of complex numbers. |
| [isfortran](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isfortran.html#numpy.isfortran)(a) | Check if the array is Fortran contiguous but *not* C contiguous. |
| [isreal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isreal.html#numpy.isreal)(x) | Returns a bool array, where True if input element is real. |
| [isrealobj](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isrealobj.html#numpy.isrealobj)(x) | Return True if x is a not complex type or an array of complex numbers. |
| [isscalar](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isscalar.html#numpy.isscalar)(num) | Returns True if the type of *num* is a scalar type. |

**# Logical operations #**

|  |  |
| --- | --- |
| [logical\_and](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logical_and.html#numpy.logical_and)(x1, x2, /[, out, where, …]) | Compute the truth value of x1 AND x2 element-wise. |
| [logical\_or](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logical_or.html#numpy.logical_or)(x1, x2, /[, out, where, casting, …]) | Compute the truth value of x1 OR x2 element-wise. |
| [logical\_not](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logical_not.html#numpy.logical_not)(x, /[, out, where, casting, …]) | Compute the truth value of NOT x element-wise. |
| [logical\_xor](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.logical_xor.html#numpy.logical_xor)(x1, x2, /[, out, where, …]) | Compute the truth value of x1 XOR x2, element-wise. |

# Comparison #

|  |  |
| --- | --- |
| [allclose](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.allclose.html#numpy.allclose)(a, b[, rtol, atol, equal\_nan]) | Returns True if two arrays are element-wise equal within a tolerance. |
| [isclose](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.isclose.html#numpy.isclose)(a, b[, rtol, atol, equal\_nan]) | Returns a boolean array where two arrays are element-wise equal within a tolerance. |
| [array\_equal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.array_equal.html#numpy.array_equal)(a1, a2) | True if two arrays have the same shape and elements, False otherwise. |
| [array\_equiv](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.array_equiv.html#numpy.array_equiv)(a1, a2) | Returns True if input arrays are shape consistent and all elements equal. |

|  |  |
| --- | --- |
| [greater](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.greater.html#numpy.greater)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 > x2) element-wise. |
| [greater\_equal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.greater_equal.html#numpy.greater_equal)(x1, x2, /[, out, where, …]) | Return the truth value of (x1 >= x2) element-wise. |
| [less](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.less.html#numpy.less)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 < x2) element-wise. |
| [less\_equal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.less_equal.html#numpy.less_equal)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 =< x2) element-wise. |
| [equal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.equal.html#numpy.equal)(x1, x2, /[, out, where, casting, …]) | Return (x1 == x2) element-wise. |
| [not\_equal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.not_equal.html#numpy.not_equal)(x1, x2, /[, out, where, casting, …]) | Return (x1 != x2) element-wise. |

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

# Arithmetic function #

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

|  |  |
| --- | --- |
| [add](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.add.html#numpy.add)(x1, x2, /[, out, where, casting, order, …]) | Add arguments element-wise. |
| [reciprocal](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.reciprocal.html#numpy.reciprocal)(x, /[, out, where, casting, …]) | Return the reciprocal of the argument, element-wise. |
| [positive](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.positive.html#numpy.positive)(x, /[, out, where, casting, order, …]) | Numerical positive, element-wise. |
| [negative](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.negative.html#numpy.negative)(x, /[, out, where, casting, order, …]) | Numerical negative, element-wise. |
| [multiply](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.multiply.html#numpy.multiply)(x1, x2, /[, out, where, casting, …]) | Multiply arguments element-wise. |
| [divide](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.divide.html#numpy.divide)(x1, x2, /[, out, where, casting, …]) | Returns a true division of the inputs, element-wise. |
| [power](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.power.html#numpy.power)(x1, x2, /[, out, where, casting, …]) | First array elements raised to powers from second array, element-wise. |
| [subtract](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.subtract.html#numpy.subtract)(x1, x2, /[, out, where, casting, …]) | Subtract arguments, element-wise. |
| [true\_divide](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.true_divide.html#numpy.true_divide)(x1, x2, /[, out, where, …]) | Returns a true division of the inputs, element-wise. |
| [floor\_divide](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.floor_divide.html#numpy.floor_divide)(x1, x2, /[, out, where, …]) | Return the largest integer smaller or equal to the division of the inputs. |
| [float\_power](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.float_power.html#numpy.float_power)(x1, x2, /[, out, where, …]) | First array elements raised to powers from second array, element-wise. |
| [fmod](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.fmod.html#numpy.fmod)(x1, x2, /[, out, where, casting, …]) | Return the element-wise remainder of division. |
| [mod](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.mod.html#numpy.mod)(x1, x2, /[, out, where, casting, order, …]) | Return element-wise remainder of division. |
| [modf](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.modf.html#numpy.modf)(x[, out1, out2], / [[, out, where, …]) | Return the fractional and integral parts of an array, element-wise. |
| [remainder](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.remainder.html#numpy.remainder)(x1, x2, /[, out, where, casting, …]) | Return element-wise remainder of division. |
| [divmod](https://docs.scipy.org/doc/numpy-1.17.0/reference/generated/numpy.divmod.html#numpy.divmod)(x1, x2[, out1, out2], / [[, out, …]) | Return element-wise quotient and remainder simultaneously. |

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

# Interview Qeustion #

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

**Why numpy is preferred over python list?**

1. Python lists support storing heterogeneous data types whereas NumPy arrays can store homogeneous data so it assign the required amount of memory for each element leading to less memory uses.
2. Numpy supports vectorization, leading to fast operation.

**Numpy supported data type and how to identify?**

* **i** - integer
* **S** - string
* **b** - boolean
* **f** - float
* **u** - unsigned integer
* **c** - complex float
* **m** - timedelta
* **M** - datetime
* **O** - object
* **U** - unicode string
* **V** - fixed memory chunk for types such as void

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

**#Numpy random module #**

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

Numpy random module is used to create the array by randomly picking value based on parameters.

**random.randint(start, end, size=(m,n))**

This function is used to create a random array by randomly picking value from range [start, end) whose size will be specified by size(m,n)

**Start** : minimum value , default 0

**np.random.random([size])**

This function of random module is used to generate random floats number in the half-open interval [0.0, 1.0)

**np.random.random\_sample([size])**

This function of random module is used to generate random floats number in the half-open interval [0.0, 1.0).

**np.random.random\_integers(low[, high, size])**

This function of random module is used to generate random integers number of type np.int between low and high.

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

**# Input and output #**

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

**#Methods related with text files processing#**

We can use text file data to convert into numpy array or can save array data into text file and other operation.

We have below method for this-

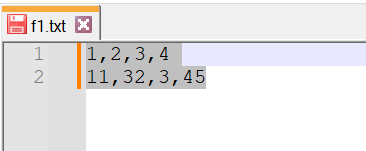
np.loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes', max\_rows=None, \*, quotechar=None, like=None)

This function is used to read data from test file and convert into array.

Number of column or column data must be same in each row.

**Example:**

Let say we have below data into f1.txt file. Read the data and convert into array.



**Solution:**

arr=np.loadtxt('f1.txt', delimiter=',', dtype='i')

print(arr)

**Output**:

[[ 1 2 3 4]

[11 32 3 45]]

**np.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)**

Save an array to a text file.

**Parameter:**

**fname** : file name, if file name is .gz then it will be uncompressed by itself.

**delimiter** : delimiter for column data

|  |  |
| --- | --- |
| [genfromtxt](https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html#numpy.genfromtxt)(fname[, dtype, comments, ...]) | Load data from a text file, with missing values handled as specified. |
| [fromregex](https://numpy.org/doc/stable/reference/generated/numpy.fromregex.html#numpy.fromregex)(file, regexp, dtype[, encoding]) | Construct an array from a text file, using regular expression parsing. |
| [fromstring](https://numpy.org/doc/stable/reference/generated/numpy.fromstring.html#numpy.fromstring)(string[, dtype, count, like]) | A new 1-D array initialized from text data in a string. |
| [ndarray.tofile](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.tofile.html#numpy.ndarray.tofile)(fid[, sep, format]) | Write array to a file as text or binary (default). |
| [ndarray.tolist](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.tolist.html#numpy.ndarray.tolist)() | Return the array as an a.ndim-levels deep nested list of Python scalars. |

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

**# String operations #**

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

In numpy we have methods to operate on value of numpy arrays, in case arrays are string values.

|  |  |
| --- | --- |
| np.cahr.add(x1, x2) | Return element-wise string concatenation for two arrays of str or unicode. |
| np.cahr.multiply(a, i) | Return (a \* i), that is string multiple concatenation, element-wise. |
| np.cahr.mod(a, values) | Return (a % i), that is pre-Python 2.6 string formatting (interpolation), element-wise for a pair of array\_likes of str or unicode. |
| np.cahr.capitalize(a) | Return a copy of *a* with only the first character of each element capitalized. |
| np.cahr.center(a, width[, fillchar]) | Return a copy of *a* with its elements centered in a string of length *width*. |
| np.cahr.decode(a[, encoding, errors]) | Calls bytes.decode element-wise. |
| np.cahr.encode(a[, encoding, errors]) | Calls *str.encode* element-wise. |
| np.cahr.expandtabs(a[, tabsize]) | Return a copy of each string element where all tab characters are replaced by one or more spaces. |
| np.cahr.join(sep, seq) | Return a string which is the concatenation of the strings in the sequence *seq*. |
| np.cahr.ljust(a, width[, fillchar]) | Return an array with the elements of *a* left-justified in a string of length *width*. |
| np.cahr.lower(a) | Return an array with the elements converted to lowercase. |
| np.cahr.lstrip(a[, chars]) | For each element in *a*, return a copy with the leading characters removed. |
| np.cahr.partition(a, sep) | Partition each element in *a* around *sep*. |
| np.cahr.replace(a, old, new[, count]) | For each element in *a*, return a copy of the string with all occurrences of substring *old* replaced by *new*. |
| np.cahr.rjust(a, width[, fillchar]) | Return an array with the elements of *a* right-justified in a string of length *width*. |
| np.cahr.rpartition(a, sep) | Partition (split) each element around the right-most separator. |
| np.cahr.rsplit(a[, sep, maxsplit]) | For each element in *a*, return a list of the words in the string, using *sep* as the delimiter string. |
| np.cahr.rstrip(a[, chars]) | For each element in *a*, return a copy with the trailing characters removed. |
| np.cahr.split(a[, sep, maxsplit]) | For each element in *a*, return a list of the words in the string, using *sep* as the delimiter string. |
| np.cahr.splitlines(a[, keepends]) | For each element in *a*, return a list of the lines in the element, breaking at line boundaries. |
| np.cahr.strip(a[, chars]) | For each element in *a*, return a copy with the leading and trailing characters removed. |
| np.cahr.swapcase(a) | Return element-wise a copy of the string with uppercase characters converted to lowercase and vice versa. |
| np.cahr.title(a) | Return element-wise title cased version of string or unicode. |
| np.cahr.translate(a, table[, deletechars]) | For each element in *a*, 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. |
| np.cahr.upper(a) | Return an array with the elements converted to uppercase. |
| np.cahr.zfill(a, width) | Return the numeric string left-filled with zeros |

**Example:**

From below two list, create array and then concatenate element wise data and create a new array.

l1=[['hello','hii'],['Ram','Rakesh']]

l2=[['GM','GE'],['p1','P2']]

**Solution:**

arr1=np.array(l1)

arr2=np.array(l2)

x=np.char.add(arr1,arr2)

|  |  |
| --- | --- |
| Arr1:  [['hello' 'hii']  ['Ram' 'Rakesh']]  Arr2:  [['hello' 'hii']  ['Ram' 'Rakesh']] | np.char( arr1, arr2 )  [['hello' 'hii']  ['Ram' 'Rakesh']] |

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

**# String information #**

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

We have below methods –

|  |  |
| --- | --- |
| np.cahar.count(a, sub[, start, end]) | Returns an array with the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. |
| np.cahar.endswith(a, suffix[, start, end]) | Returns a boolean array which is *True* where the string element in *a* ends with *suffix*, otherwise *False*. |
| np.cahar.find(a, sub[, start, end]) | For each element, return the lowest index in the string where substring *sub* is found. |
| np.cahar.index(a, sub[, start, end]) | [Like find, but raises ValueError when the substring is not found.](https://numpy.org/doc/stable/reference/generated/numpy.char.find.html#numpy.char.find) |
| np.cahar.isalpha(a) | Returns true for each element if all characters in the string are alphabetic and there is at least one character, false otherwise. |
| np.cahar.isalnum(a) | Returns true for each element if all characters in the string are alphanumeric and there is at least one character, false otherwise. |
| np.cahar.isdecimal(a) | For each element, return True if there are only decimal characters in the element. |
| np.cahar.isdigit(a) | Returns true for each element if all characters in the string are digits and there is at least one character, false otherwise. |
| np.cahar.islower(a) | Returns true for each element if all cased characters in the string are lowercase and there is at least one cased character, false otherwise. |
| np.cahar.isnumeric(a) | For each element, return True if there are only numeric characters in the element. |
| np.cahar.isspace(a) | Returns true for each element if there are only whitespace characters in the string and there is at least one character, false otherwise. |
| np.cahar.istitle(a) | Returns true for each element if the element is a titlecased string and there is at least one character, false otherwise. |
| np.cahar.isupper(a) | Return true for each element if all cased characters in the string are uppercase and there is at least one character, false otherwise. |
| np.cahar.rfind(a, sub[, start, end]) | For each element in *a*, return the highest index in the string where substring *sub* is found, such that *sub* is contained within [*start*, *end*]. |
| np.cahar.rindex(a, sub[, start, end]) | [Like rfind, but raises ValueError when the substring sub is not found.](https://numpy.org/doc/stable/reference/generated/numpy.char.rfind.html#numpy.char.rfind) |
| np.cahar.startswith(a, prefix[, start, end]) | Returns a boolean array which is *True* where the string element in *a* starts with *prefix*, otherwise *False*. |
| np.cahar.str\_len(a) | Return len(a) element-wise. |

**Question**:

Write a NumPy program to extract all the rows to compute the student weight from a given array (student information) where a third column starts with a given character 'E'. Array is:-

[['01' 'V' 'Debby Pramod' '30.21']

['02' 'V' 'Artemiy Ellie' '29.32']

['03' 'V' 'Baptist Kamal' '31.0']

['04' 'V' 'Lavanya Davide' '30.22']

['05' 'V' 'Fulton Antwan' '30.21']

['06' 'V' 'Euanthe Sandeep' '31.0']

['07' 'V' 'Endzela Sanda' '32.0']

['08' 'V' 'Victoire Waman' '29.21']

['09' 'V' 'Briar Nur' '30.0']

['10' 'V' 'Rose Lykos' '32.0']]

**Solution:**

#first get those rows which start with 'E'

x = student[np.char.startswith(student[:,2], 'E')]

print(x)

#Now convert into flat type

y=x[:,3].astype(float)

#Now get the sum

print(y.sum())

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

**# Mathematical Function #**

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

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

**# Arithmetic Operation #**

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

We have below methods for arithmetic related operation-

|  |  |
| --- | --- |
| np.add(x1, x2, /[, out, where, casting, order, ...]) | Add arguments element-wise. |
| np.reciprocal(x, /[, out, where, casting, ...]) | Return the reciprocal of the argument, element-wise. |
| np.positive(x, /[, out, where, casting, order, ...]) | Numerical positive, element-wise. |
| np.negative(x, /[, out, where, casting, order, ...]) | Numerical negative, element-wise. |
| np.multiply(x1, x2, /[, out, where, casting, ...]) | Multiply arguments element-wise. |
| np.divide(x1, x2, /[, out, where, casting, ...]) | Divide arguments element-wise. |
| np.power(x1, x2, /[, out, where, casting, ...]) | First array elements raised to powers from second array, element-wise. |
| np.subtract(x1, x2, /[, out, where, casting, ...]) | Subtract arguments, element-wise. |
| np.true\_divide(x1, x2, /[, out, where, ...]) | Divide arguments element-wise. |
| np.floor\_divide(x1, x2, /[, out, where, ...]) | Return the largest integer smaller or equal to the division of the inputs. |
| np.float\_power(x1, x2, /[, out, where, ...]) | First array elements raised to powers from second array, element-wise. |
| np.fmod(x1, x2, /[, out, where, casting, ...]) | Returns the element-wise remainder of division. |
| np.mod(x1, x2, /[, out, where, casting, order, ...]) | Returns the element-wise remainder of division. |
| np.modf(x[, out1, out2], / [[, out, where, ...]) | Return the fractional and integral parts of an array, element-wise. |
| np.remainder(x1, x2, /[, out, where, casting, ...]) | Returns the element-wise remainder of division. |
| np.divmod(x1, x2[, out1, out2], / [[, out, ...]) | Return element-wise quotient and remainder simultaneously. |

**Question:**

Write a NumPy program to calculate the arithmetic means of corresponding elements of two given arrays of same size

**Solution:**

print(np.divide(np.add(nums1, nums2), 2))

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

**# Sums, products, differences #**

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

|  |  |
| --- | --- |
| np.prod(a[, axis, dtype, out, keepdims, ...]) | Return the product of array elements over a given axis. |
| np.sum(a[, axis, dtype, out, keepdims, ...]) | Sum of array elements over a given axis. |
| np.nanprod(a[, axis, dtype, out, keepdims, ...]) | Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones. |
| np.nansum(a[, axis, dtype, out, keepdims, ...]) | Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero. |
| np.cumprod(a[, axis, dtype, out]) | Return the cumulative product of elements along a given axis. |
| np.cumsum(a[, axis, dtype, out]) | Return the cumulative sum of the elements along a given axis. |
| np.nancumprod(a[, axis, dtype, out]) | Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one. |
| np.nancumsum(a[, axis, dtype, out]) | Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero. |
| np.diff(a[, n, axis, prepend, append]) | Calculate the n-th discrete difference along the given axis. |
| np.ediff1d(ary[, to\_end, to\_begin]) | The differences between consecutive elements of an array. |
| np.gradient(f, \*varargs[, axis, edge\_order]) | Return the gradient of an N-dimensional array. |
| np.cross(a, b[, axisa, axisb, axisc, axis]) | Return the cross product of two (arrays of) vectors. |
| np.trapz(y[, x, dx, axis]) | Integrate along the given axis using the composite trapezoidal rule. |

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

**# Trigonometric functions #**

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

|  |  |
| --- | --- |
| [sin(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.sin.html#numpy.sin) | Trigonometric sine, element-wise. |
| [cos(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.cos.html#numpy.cos) | Cosine element-wise. |
| [tan(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.tan.html#numpy.tan) | Compute tangent element-wise. |
| [arcsin(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.arcsin.html#numpy.arcsin) | Inverse sine, element-wise. |
| [arccos(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.arccos.html#numpy.arccos) | Trigonometric inverse cosine, element-wise. |
| [arctan(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.arctan.html#numpy.arctan) | Trigonometric inverse tangent, element-wise. |
| [hypot(x1, x2, /[, out, where, casting, ...])](https://numpy.org/doc/stable/reference/generated/numpy.hypot.html#numpy.hypot) | Given the "legs" of a right triangle, return its hypotenuse. |
| [arctan2(x1, x2, /[, out, where, casting, ...])](https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html#numpy.arctan2) | Element-wise arc tangent of x1/x2 choosing the quadrant correctly. |
| [degrees(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.degrees.html#numpy.degrees) | Convert angles from radians to degrees. |
| [radians(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.radians.html#numpy.radians) | Convert angles from degrees to radians. |
| [unwrap(p[, discont, axis, period])](https://numpy.org/doc/stable/reference/generated/numpy.unwrap.html#numpy.unwrap) | Unwrap by taking the complement of large deltas with respect to the period. |
| [deg2rad(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.deg2rad.html#numpy.deg2rad) | Convert angles from degrees to radians. |
| [rad2deg(x, /[, out, where, casting, order, ...])](https://numpy.org/doc/stable/reference/generated/numpy.rad2deg.html#numpy.rad2deg) | Convert angles from radians to degrees. |

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

**# Rounding #**

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

|  |  |
| --- | --- |
| np.around(a[, decimals, out]) | Evenly round to the given number of decimals. |
| np.rint(x, /[, out, where, casting, order, ...]) | Round elements of the array to the nearest integer. |
| np.fix(x[, out]) | Round to nearest integer towards zero. |
| np.floor(x, /[, out, where, casting, order, ...]) | Return the floor of the input, element-wise. |
| np.ceil(x, /[, out, where, casting, order, ...]) | Return the ceiling of the input, element-wise. |
| np.trunc(x, /[, out, where, casting, order, ...]) | Return the truncated value of the input, element-wise. |