NumPy Notes:

array\_name = np.array(list\_name)

All elements of np array have to have the same type.

Elements of a list are type casted to most significant type if needed.

array\_name.dtype : type of the array

array\_name.shape : gives (# of elements in 1st dimension, # of elements in 2nd dimension, ….)

array\_name.ndim : gives number of rows

An array made out of list of lists where the internal lists have different lengths: An array of lists having object type.

Array initialized to zeros:

array\_name = np.zeros(shape)

shape must be a tuple except for 1d.

Array initialized to ones:

array\_name = np.ones(shape, dtype = int)

shape must be a tuple except for 1d.

dtype = int: optional. Default: float

Array initialized to empty:

np.emty(dim)

Not safe to assume that this will get you ones, or zeros, or anything specific

np.eye(n) : gives the square identity matrix of dimension n\*n

np.arange(same parameters given to range): makes a 1d np array following parameter specification.

array\_name.reshape(shape)

Does not change the array ‘array\_name’.

Shape does not have to be exactly in type form.

Shape = 3,5 or shape = (3,5)

Error is thrown if number of elements in ‘array\_name’ cannot be reshaped into ‘shape’

You can make a zeros, ones, empty array out of another array’s shape.

Does not change the array ‘array\_name’.

np.ones\_like(array\_name)

np.empty\_like(array\_name)

np.zeros\_like(array\_name)

Shape: 2\*3\*2

Array reshaped into this should have 2\*3\*2 = 12 elements. Otherwise, error.

2 planes of (3\*2)

Two ways of indexing:

arr3d[0,1,0]

arr3d[0][1][0]

**You can CAST an array from one dtype to another using astype method. Using astype ALWAYS CREATES A NEW ARRAY, leaving the original array untouched**

array\_name.astype(type)

[1, 2.5, 3] to int type: [1, 2, 3]

[‘1.5’, ‘3.6’, ‘-2.9’] to float type: [1.5, 3.6, -2.9]

[‘1.5’. ‘2.5’, ‘x.y’] to float type: error

arr\*arr -> each element is squared -> arr\*\*2->np.power(arr, 2)

1/arr -> each element in inverted

Slicing an array gives a new array (view). Works similarly to slicing a list.

More on indexing:

Say, ‘myarr’ is a 3\*3 matrix. ‘myarr’ is itself changed here

myarr[1] = [-1, -2, -3]

(updates the second row)

myarr[:,2] = -1

(updates the second column)

narr[[2,4,0,7],[1,2,0,3]] # selects [2,1],[4,2],[0,0],[7,3] and creates new 1d array of those values

narr[[2,4,0,7]] # rows as specified

narr[[2,4,0,7]][:,[1,2,3,0]] # shuffle columns

An array slice is a “view” (not copy) on original array. If you modify a slice, the original array is modified, and so are the other overlapping slices.

Copy of a slice can be made.

array.copy()

arr2d[:,1] # all rows, 2nd column

prints out the first column as a 1d array

rowslc = arr2d[1:] # 2nd and 3rd rows

arr2d:

[[1 2 3]

[4 5 6]

[7 8 9]]

colslc = arr2d[:, [0,2]] # 1st and 3rd columns

array([[1, 3],

[4, 6],

[7, 9]])

Slicing by column gives a copy, not a view.

Slicing by row gives a view, not a copy.

Filtering by values

Slicing with Boolean filtering gets a copy, not a view.

Applying a Boolean mask from one array to another.

Global filtering with any and all

array.any()

array.all()

Universal Functions (unary/binary) [Don’t change the original array]

Unary

np.exp(array)

np.square(array)

np.sqrt(array)

np.power(array, num)

np.abs(array)

np.fabs(array) -> faster and gives real numbers

np.ceil(array)

np.floor(array)

np.rint(array)-> element 5.5 is rounded to 5.

np.isnan(array)

np.max(array, axis)

axis=1: maximum for each row

axis=0: maximum for each column

np.nan gives nan

Binary

np.power(arr1, arr2) #raising elements of arr1 to those of arr2

np.maximum(arr1,arr2)

np.fmax(arr1,arr2) -> ignores nan

max is not a binary ufunc

np.greater(arr1, arr2)

np.random

np.random.randint(a,b): random integer between a (inclusive) and b (exclusive)

Python’s random.randint is same but b is inclusive there.

You can fill an array randomly by choosing random integers this way.

Python’s random.random() returns single random number in the range [0, 1).

np.random.random(a) -> gives a 1d array of 5 random [0,1) elements

np.random.random((3,2)) -> gives a 3\*2 array of random [0,1) elements

np.random.randn() -> creates an array of specified shape and fills it with **random** values as per standard normal distribution

np.random.choice(array, a, replace = bool) “a random selections from array”

bool = false: no duplicates (The same element is not chosen twice)

random.choice(array): gives a single item

np.random.shuffle(array) -> shuffles the original array

If array is 2d, only the rows are shuffled.

np.random.permutation(array) -> same as shuffle but creates a new array. Does not change original.

Math and stats functions (Reductions)

arr.max() -> same as np.max(array). Same for below

arr.min()

arr.mean()

arr.sum()

arr.std()

arr.cumsum()

2 ways to get index of minimum value:

Array.argmin()

Np.argmin(array)

Axis parameters can be added within the parenthesis for both cases or even for above functions. Adding axis for 2d arrays, specifies whether the operation should take place along the row or column.

Unique

Np.unique(array): gives a 1d array of sorted distinct values in array

Sorting

Np.sort(array) -> does not change array

Array.sort() -> changes array

For a 2d array, each row is sorted

Np.argsort(array)

Array.argsort()

These are exactly the same.

Give an array of the same size with the indexes that will sort the array.

Happens row-wise for 2d array, unless axis is explicitly specified as 0, in which case it occurs column wise.

Zip() function

np.where(cond, xarr, yarr) to conditionally manipulate items of an array.

Array.sum() returns number of True values when array is of Boolean type.

Any() and all() work for non Boolean types too.

Linear Algebra

2 ways of transpose (both don’t change original):

Arr.T

Arr.transpose()

Matrix multiplication:

Np.dot(mat1, mat2).

If mat1’s column # != mat2’s row#, error.

You can also use this to find dot product of 2 1d arrays with same number of elements.