**Numpy Notes:**

**Basics:**

**ndarray.ndim**

the number of axes (dimensions) of the array.

**ndarray.shape**

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n* rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

**ndarray.size**

the total number of elements of the array. This is equal to the product of the elements of shape.

**ndarray.dtype**

an object describing the type of the elements in the array. One can create or specify dtype’s using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

**ndarray.itemsize**

the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

**ndarray.data**

the buffer containing the actual elements of the array. Normally, we won’t need to use this attribute because we will access the elements in an array using indexing facilities.

**Array Creation:**

**np.array([(2,4,5),(2,1,6)])**

creates an array with these values

**np.zeroes((rows,# of values))**

creates arrays filled 0s with the number of rows and columns.

**np.empty()**

creates arrays filled with random numbers depending on the state of memory

**np.arange(start,end,step)**

creates an array from the range by step

Example: np.arrange(15,30,5) = [15,20,25]

**np.linspace(start,end,# of values)**

creates and array but instead gets the number of values you want from a range

Example: np.linspace(2,6,5) = [2,3,4,5,6] #5 numbers from 2 to 6

**Creating multi-dimensional arrays:**

**2d array**

np.zeros((3, 4))

array([[0., 0., 0., 0.], 3 Rows, 4 values

[0., 0., 0., 0.],

[0., 0., 0., 0.]])

**3d array**

np.ones( (2,3,4), )

array([[[1, 1, 1, 1], 2 sets, 3 rows in each, 4 values

[1, 1, 1, 1],

[1, 1, 1, 1]],

[[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]]]

**4d array and so on…**

np.ones( (2,2,3,4), )

array([[[[1, 1, 1, 1],

[1, 1, 1, 1], 2 large sets, 2 small sets, 3 rows, 4 values

[1, 1, 1, 1]],

[[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]]]

---------------------------------

[[[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]],

[[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]]]]

**Adding, removing, and sorting elements**

arr=np.array([2,5,6,1])

**np.sort(arr)**

Sorts the array into ascending order # [1,2,5,6]

**-np.sort(-arr)**

Sorts in descending order

**np.concatenate()**

combines arrays into one

**Reshaping an array**

a=np.arange(6)

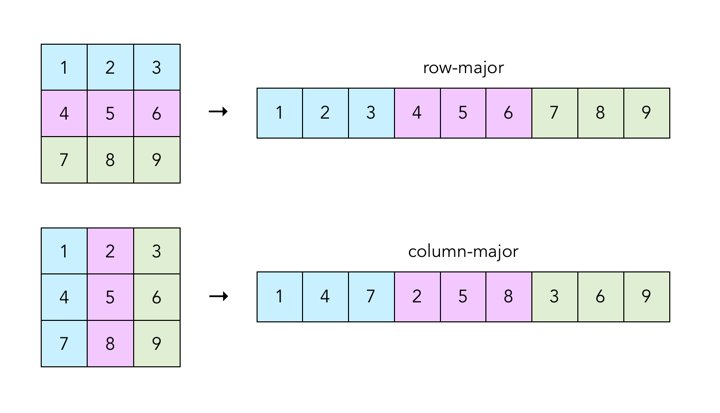
**a.reshape(3,2)**

reshape can change the shape of the original array. This for example, will create a 3 by 2 array.

**You can also add parameters np.reshape(a, newshape=(3,2), order=’C’)**

**Newshape is the new shape**

Order is the way you want to order, C or F. F is Fortran ordering which goes from columns, column major. C is row major.



**Matrix Multiplication :** a **@** b

**[1,3] \* [3,4] =[1\*3+3\*3, 1\*4+3\*1] =[12,7]**

**[2,4] \* [3,1 ] [2\*3+4\*3, 2\*4+4\*1] [18,12]**

Multiply each element in the row of M1 with each element in the column of M2

## How to create an array from existing data

**[Start:End] Doesn’t include end value**

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

arr2=arr[3:8] = [4,5,6,7,8]

arr3=arr[:4] = [1,2,3,4]

arr4=[4:]= [5,6,7,8,9,10]

a=[[1,2],[3,4]] b=[[5,6],[7,8]

**np.vstack(a,b)**  [1,2]

vertically stacks the 2 arrays [3,4]

[5,6]

[7,8]

**np.hstack(a,b) :** [[1,2,5,6],[3,4,7,8]]

horizontally stacks the 2 arrays

**b2=a.copy():**

creates a copy of the array

## Indexing and slicing[¶](https://numpy.org/doc/1.19/user/absolute_beginners.html#indexing-and-slicing)

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

**np.nonzero(a<5):**

Prints out 2 arrays, the row which it is loacated and the column.

Example: b=(array( [0,0,0,0], array( [0,1,2,3]))

At row 0, the first 4 columns have values less than 5

**list(zip(b[0],b[1])):**

Zip groups each value with the same index. Here, it takes the first array and groups it’s values to the second array, respective to their indexes.

list\_of\_coordinates= list(zip(b[0], b[1]))

**>>> for** coord **in** list\_of\_coordinates:

**...**  print(coord)

(0, 0)

(0, 1)

(0, 2)

(0, 3)

## Basic array operations[¶](https://numpy.org/doc/1.19/user/absolute_beginners.html#basic-array-operations)

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

b=np.array([2,3])

a+b = [3,5]

a-b = [-1,-1]

a\*b = [2,6]

a/b = [1/2,2/3]

You can sum the rows with:

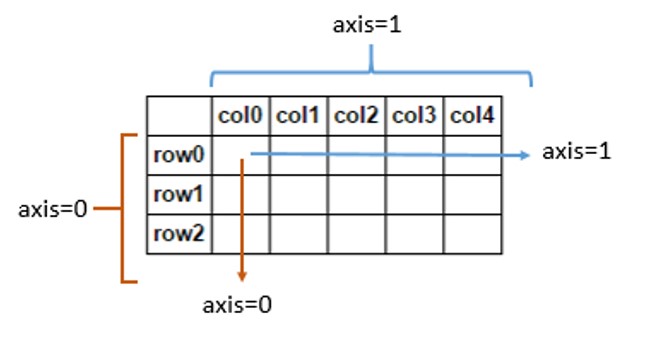
**>>>** b.sum(axis=0)

array([3, 3])

You can sum the columns with:

**>>>** b.sum(axis=1)

array([2, 4])



## More useful array operations

**data.max()**

Gets maximum value of data set

**data.min()**

Gets minimum value of data set

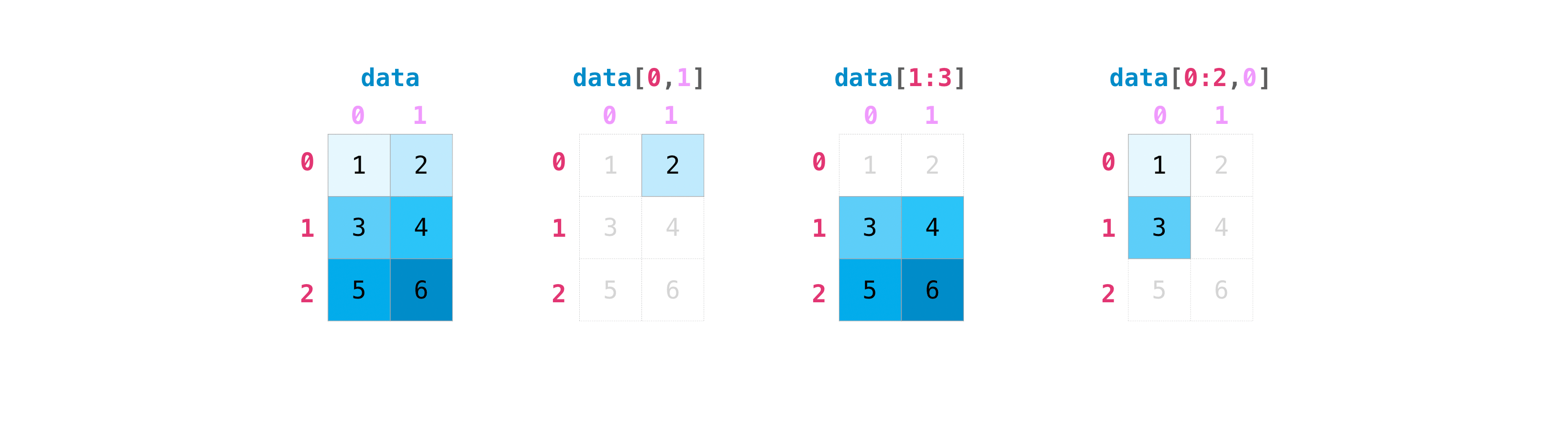
**data.std()**

Gets standard deviation of a data set

## Creating matrices

data=np.array([[1,2],[3,4],[5,6]])

**data[0,1]=1 data[1:3] = [[3,4],[5,6]] data[0:2,0] = [1,3]**



## Random Generator

**np.random.randint(low=1, high=10, size=3):**

Random 3 numbers from 1(inclusive) to 10(exclusive).

**np.random.seed(x):**

To store the value produced by the random generator. For example,

np.random.seed(123)

np.random.randint(1,7,3) = [6,3,5]

If I want this same value again, I can use the seed

np.random.seed(123)

np.random.randint(1,7,3) = [6,3,5]

**np.random.choice:**

np.random.choice(

a = np.arange(1, 7),

size = 3,

replace = False,

p = np.array([0.1, 0.1, 0.1, 0.1, 0.3, 0.3]))

array, # of values, replacement, probability of each value

**np.random.uniform(low,high,size):**

Samples values from the uniform distribution.

**np.random.normal( loc, scale, size) (mean, standard deviation, size)**

Samples from a standard normal distribution.

**np.random.binomial( n, p, size): np.random.binomial(10,0.5)**

Samples from binomial distribution.

Ex: [6] This would mean it succeeds 6 times out of the 10.

generator = np.random.default\_rng(seed=123)

Creating a generator can allow faster implementations. For example,

generator.integers(1,7) instead of np.random.randint(1,7)

## How to get unique items and counts

**np.unique(a):**

prints array of all unique values.

**np.unique(a, return\_index=True):**

prints array of all indices of unique values.

**np.unique(a, return\_counts=True):**

prints an array of the frequency of the unique value.

**np.unique(a, axis=0):**

Finds unique values from 2d array by rows (1st column, 2nd column…)

Axis=1 will go by columns (1st row, 2nd row…)

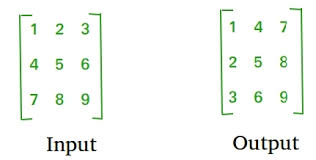
## Transposing a matrix

a=np.array([[1,2],[3,4],[5,6]])

**a.transpose() or a.T**

Takes the columns of matrix and puts them into an array

Ex: a.transpose() = [1,3,5] , [2,4,6]



## How to reverse an array

**np.flip()**

reverses an array based on the axis. If no axis is specified, it will reverse among all axes of the input array.

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

reversed\_a\_rows = np.flip(a, axis=0)

**>>>** print(reversed\_a\_rows)

[[ 9 10 11 12] Reverses the column of each row

[ 5 6 7 8] Column 1, 3 rows get reversed…

[ 1 2 3 4]]

reversed\_a\_columns = np.flip(a, axis=1)

**>>>** print(reversed\_a\_columns)

[[ 4 3 2 1] Reverses the row of each column

[ 8 7 6 5] Row 1, 4 columns get reversd

[12 11 10 9]]

**Views vs Copies:**

Views will change the original array. Copies will not.

## Reshaping and flattening multidimensional arrays

**.flatten() and .ravel() both shape multidimensional arrays to 1d**

**.**flatten() is a copy. This means any changes to the flattened array will NOT change the original array.

.ravel() is a view. This means that changes to the flattened array WILL change the original array

a1 = x.flatten()

**>>>** a1[0] = 99

**>>>** print(x) *# Original array*

[[ 1 2 3 4] Did not change the first number to 99

[ 5 6 7 8]

[ 9 10 11 12]]

**>>>** print(a1) *# New array*

[99 2 3 4 5 6 7 8 9 10 11 12] But updated in the new array

**.ravel()**

**>>>** a2 = x.ravel()

**>>>** a2[0] = 98

**>>>** print(x) *# Original array*

[[98 2 3 4] Changes the 1st number to 98

[ 5 6 7 8]

[ 9 10 11 12]]

**>>>** print(a2) *# New array*

[98 2 3 4 5 6 7 8 9 10 11 12] Also updated in the new array

## How to save and load NumPy objects[¶](https://numpy.org/doc/1.19/user/absolute_beginners.html#how-to-save-and-load-numpy-objects)

**np.save(‘Hello’,a)**

saves the array in the file name ‘Hello’

**np.load(‘Hello.npy)**

loads the array saves with that file name

**np.savetxt(‘Hello.csv’,a)**

saves the array in the file name ‘Hello’ as text or csv files

**np.loadtxt(‘Hello.csv’)**

loads the array with text/csv file type and name