Numpy Syntax, Options

You can assume that any variable given here like - a, b, c etc.   => refers to a Numpy Array
If the variable is of any other Data Type then it will be explicitly specified

Below formulae would work for Arrays of all dimensions =>1d, 2d, 3d, nd-Arrays
If a formula works on only Arrays of particular dimension size, then it will be specified explicitly.

Description	Code
Numpy Basics	
Libraries, Array Basics	

Libraries Imported	import numpy as np
	import matplotlib.pyplot as plt
	
	import timeit
	import copy as cp

Data Type of all Items of Numpy Array	b.dtype
Number of Dimension in Numpy Array	b.ndim or np.ndim(b)
Shape of Numpy Array	b.shape or np.shape(b)
Creating a Numpy Array	
Create Numpy array from a List	b = np.array(a)
Create Numpy array from a List/Array + Change Data Type of all Items of List	b = np.array(a, dtype = "float")
Create Numpy array from a List defined by RANGE	b = np.array(range(1,5))
Create Numpy array using ARANGE	b = np.arange(1, 5, 0.5)

Create Array having only 1. values	b = np.ones(12)
	b = np.ones((5,5),dtype='int')

Change Array Shape/Dimensions	
Reshape, Add Dimension, Remove Dimension and Transpose	
Reshape Numpy Array, Create a 2d/nd-Array	b = a.reshape(2, 4,-211) #A has 16 items #Auto reshape by giving negative number
Add a new dimension of shape/size => 1 to LEFT of an array	b = a[np.newaxis, np.newaxis] #Adds 2d
Add a new dimension of shape/size => 1 to RIGHT of an array	b = a[: , np.newaxis]
Add a new dimension of shape/size => 1 in MIDDLE of an array	b = a [: , :, np.newaxis, :] Shape from (2, 2, 2) => to (2, 2, 1, 2)
Use None like np.newaxis to add a new dimension to Left/Right/Middle Like np.newaxis, you can use None also to add new dimension to Left/Right/Middle	b = a[: , None]
Adds a new Dimension of Shape =1 at position given by Axis parameter	b = np.expand_dims(a, axis=N)

To remove one or more Dimensions having shape of 1	b = np.squeeze(a, axis=N)
	  = np.squeeze(a)

Flattening a Nd-array to 1d-array	b = a.flatten()
Transpose of a 2d Numpy Array	b = a.T

Access an Element or a part (Subset) of a Numpy Array	
Indexing, Multi-Indexing	
Access element in 1d Array	b = a[2]
Create Subset of 1d Numpy Array by Multi Indexing We can use either List or Array as Indices for Multi-Indexing Multi Indexing has 2 Levels of Square Bracket	b = a[[2, 6, 4]] #Using List = a[x] # x is a 1d array, e.g. [2 6 4]
Access elements in a 2d-Array	A [i] [j] ; A [i, j]
Access elements in a 3d/nd-Numpy Array	A [i] [j] [k] ; A [i, j, k]
Multi Indexing in 3d/nd-Array # Gets a[0][2][1], a[1][3][1]	b = a[[0,1], [2,3], [1,1]]
Slicing, Slicing + Indexing/ Multi-Indexing	
Slicing in 1d Numpy Array (same as in List)	b[-5:-1:1]
Slicing in 3d/nd-Numpy Array	a[0:3, 0:1, 0:2]
Use both Slicing and Indexing (O/p Array has Lesser Dimensions)	
Slicing + Accessing on a 2d Array(fixing 1 dimensions) gives a 1d array	b = a[0:2, 1]
Slicing + Accessing on a 3d Array(fixing 2 dimensions) gives a 1d array	b = a[2:3, 1, 1]
Difference between a[:, 1] and a[:, 1:2]	a[:, 1] => 1d-Array; a[:, 1:2] => 2d-Array
Slicing + Multi-Indexing => **Reorders the rows** # a.shape = (3, 3)	b = a[[2, 1, 0], 0:2]
Unpacking	
Unpacking a 2d Array (each row/inner list) goes to a variable	a, b, c, d = M (Shape of M = 4 x n) *a, b, c, d are 1d array of shape (n,)*
Unpacking a 3d Array	a, b = M (Shape of M = 2 x n1 x n2) *a, b are 2d arrays of shape (n1, n2)*
Use Unpacking to put rows/columns of an Array into variables	x, y = a.shape *#a is 2d Array*
Masking/Comparison	
For 1d array	
Create Subset of Array by **Masking** using same Array	b = a[a < 6]
Create Subset of Numpy Array by **Masking** using **another** Array	b = a[c < d] (a, c & d have same shape)
Create Subset of Numpy Array by **Masking** using a **Boolean List**	b = a[c] (c is a Boolean array with same shape as a)
For nd-Array	
Get nd-Boolean array	
Create nd-Boolean Numpy Array (Mask on nd-array & a number)	b = a < 6
Create nd-Boolean Numpy Array (Masking condition on 2 nd-arrays)	b = a < b
ANY, ALL, WHERE (For nd-Array)	Apply Condition on an Array OR Compare 2 Array of same shape
ANY (Get Result as a Single Boolean value => True/ False)	b = np.any(a < b)
ALL (Get Result as a Single Boolean value => True/ False)	b = np.all(a < b)
WHERE (Get Result as an Array of same Shape as input Array)	b = np.where(a < 0, ‘neg’, ‘pos’)

Getting info about elements in a Numpy Array or in one of its Axis	Axis values 2d Array = 0, 1 or -2, -1 Axis values 3d Array = 0, 1, 2 or -3, -2, -1

Sum	b = a.sum() or b = np.sum(a)
	b = np.sum (a, axis = 1)

Mean/Average	b = a.mean() or b = np.mean(a)
	b = np.mean (a, axis = 1)

Minimum	b = a.min() or b =np.min(a)
	b = np.min (a, axis = 1)

Maximum	b = a.max() or b = np.max(a)
	b = np.min (a, axis = 0)

argmax (gives Index of max value)	b = np.argmax(a) ; b=np.argmax(a, axis =0)
argmin (gives Index of min value)	b = np.argmin(a) ; b=np.argmin(a, axis =1)
Using Length of Array	
Number of Elements in a 1d Numpy Array	len(a) #a is 1d array
Number of Rows in a 2d Numpy Array	len(b) #b is 2d array
Number of Columns in a 2d Numpy Array	len(b [0]) #b is 2d array
Modify Element Level Values	
Element wise Operations	
By using mathematical symbols between 2 arrays of same shape Or between an array and a number(scalar)	a + b a - b a * b a**b a/b OR a + 2 a – 2 a * 2 a**2 a/2
Or by using Numpy functions between 2 arrays of same shape Or between an array and a number(scalar)	np.add(a,b) or np.add(a,3) Other functions like add are subtract, multiply, divide, power
Slicing + Assigning a Value to ALL elements "in the slice" a.shape = (2, 2, 3)	`a [ :, :, 0:2 ] = 5` OR a [:, :, 0:-1] = 5
Slicing + Multi-Indexing + Assigning Value to some fields "in the slice"	a[: , :, [0, 1]] = 5
To repeat a Numpy array rows m times and columns n times This is like Manual **Broadcasting**.	b = np.tile(a,(m,n))
Matrix Multiplication	a @ b OR np.matmul(a, b)
dot product	np.dot(a, b)
Miscellaneous Points	
Upload a Text file to Jupyter Notebook	a = np.loadtxt('survey.txt', dtype = 'int')
List Comprehension (Create new List from a List)	L = [i**2 for i in a]
%timeit	%timeit [i**2 for i in a] *(a is a List)* %timeit a**2 *(a is an Array)*
Vectorization of a function	b = np.vectorize(function_name)
Shallow Copy/Deep Copy	
Shallow Copy	a.view( ), reshape, assignment, slicing
Deep Copy	a.copy(), Any mathematical operation (Element wise operation), Masking/Comparison
How to check if Deep or Shallow Copy happened	np.shares_memory(a, b) (True/False)

Array Manipulations	
Sort an array based on its current Data Type	a = np.sort(a, axis = 2)
argsort returns an array having ‘Sequence of Indices’; which can be used to get the sorted array	b = np.argsort(a, axis = 1)
Split an array into Sub-Arrays	
split => number of sub-arrays to be created is given as the 2nd parameter	b1, b2 = np.split(a, 2, axis=1) *# e.g. a.shape = (3, 4)*
split => Index values at which Split should be done is given in Square brackets as the 2nd parameter.	b1, b2, b3 = np.split(a, [2, 3], axis = 1) *# e.g a.shape = (3, 4)*
vsplit always axis = 1	b1, b2 = np.vsplit(a, 2)
hsplit always axis = 0	b1, b2 = np.hsplit(a, 2)
dsplit always axis = 2	b1, b2 = np.dsplit(a, 2)
Stack/Merge multiple arrays into a larger array	
stack *(It adds a NEW Dimension)*	b = np.stack((a,a), axis=0)
vstack always axis = 1	b = np.vstack((a,a))
hstack always axis = 0	b = np.hstack((a,a))
dstack always axis = 2	b = np.dstack((a,a))
concatenate	b = np.concatenate((a,a), axis = 0)
Image Processing	
Basics	
Read a jpg file and make a 3d-Numpy array of Image type	a = np.array(plt.imread('puppy.jpg'))
Display a 3d Numpy Image Array as a picture	plt.imshow(a)
Saving a 3d Array storing Image data as a jpg/jpeg file	plt.imsave(‘abc.jpeg’, b) or abc.jpg
Image Manipulation	
1. Invert image **upside-down** by inverting the ROW dimension	plt.imshow (a[: : -1, : , :])
2. Reverse left-right of an image by inverting the Column dimension	plt.imshow (a[: , : :-1 , :])
3. Cropping the image, by slicing row/column shape	plt.imshow (a[100:460 , 100:460 , :])
4. Blurring image, by increasing gap while slicing rows, columns	plt.imshow (a[::10 , ::10, :])
5. Contrast the image by assigning pixels extreme values 0 & 255	plt.imshow (np.where(a >150, 255, 0))
6. Negative of the image by subtracting all pixel values by 255	plt.imshow (255 - a)
7. Display image with a single RGB colour by making values of other 2 colours zero in 3d Array of image	b[: , :, [0, 2]] = 0 ; plt.imshow(b)

8. Using Transpose to Rotate the Image by 90 degrees (swapping row and column)	b = np.transpose(a, (1, 0, 2)) ;
	plt.imshow(b)

8. Usual Transpose of 3d Numpy Array	b = a.T .... Then b => a(2, 1, 0) *(1st and 3rd dimensions get swapped)*