**Numpy Commands**

1. **Import numpy as np**
2. **1-D Array -** A = np.array( [1,2,3,4,5] ) # To create a One-dimensional array.
3. **2-D Array -** A = np.array( [[1,2,3],[4,5,6]] ) # To create a Two-dimensional array.
4. **3-D Array -**  A = np.array( [[[1,2,3],[4,5,6],[7,8,9]]] ) # To create a Three-dimensional array.
5. **Array From Tuple -** A = np.array( (1,2,3,4,5) ) # To create an array from tuple.
6. **np.array( [1,2,3,4] , ndmin = 2 , dtype = complex )**
7. **np.arange() -** A = np.arange( 1,20,3 ) # To create sequences of numbers.
8. **Reshape () -** A = A.reshape ( 3,4 ) # To reshape an array.
9. **Ndim -** A**.**ndim # To show the number of axis (dimensions/rank) of the array.
10. **shape -** A.shape # Shape of the array i.e., matrix, rows, columns.

1. **Size -** A.size # It shows the total no. of elements of the array.
2. **dtype**  - A.dtype # It shows the data type of elements of the array.
3. **itemsize**  - A.itemsize # It shows the size in bytes of each element of the array.
4. **type() -** type(A) # It shows the type of the array.
5. .**data -** A.data # It indicates the memory address of the first byte in the array.
6. **strides -** A.strides # It is the no. of bytes that should be skipped in memory to go to the next element.
7. **A = np.array( [[1,2,3], [4,5,6]] , dtype = float )** # Creating an array from lists with type float.
8. **Zeros Array -** A = np.zeros( (3,4) ) # Creating an array with all zeros values.
9. **Full Value Array -** A = np.full ( (3,4), 7 ) # Creating an array with one constant value everywhere.
10. **np.random() -** A = np.random.random() # Create an array with random values.

A = np.random.random( (2,3) )

1. **np.linspace () -** A = np.linspace (1,100,12) # It returns evenly(linearly) spaced values within a given interval.

np.linspace(start, stop , num=50, endpoint=True, retstep=True, dtype=None)

1. **Flatten Array -** A.flatten() # It is used to get a copy of array collapsed into 1-D.
2. **np.empty() -** A = np.empty( 4, dtype=int ) # It returns a new array of given shape & type, with random values.
3. **We can define the data types of rows & columns**

A = np.full( (2,3), 3, dtype = [ (‘x’,float) , (‘y’,int) ])

1. **np.eye() -**  A = np.eye(4,3) # It returns a 2-D array with ones on diagonal and zeros elsewhere. No. of rows = No. of columns.
2. **np.identity() -** A = np.identity(3, dtype=int) # It returns an identity matrix i.e., a square matrix with 1 on the main diagonal. No.of rows and no. of columns may be different.
3. **np.ones() -** A = np.ones((2,4)) # Creating an array with all Ones values.
4. **np.ones\_like() -** A = np.ones\_like(a) # It returns an array of Ones with the same shape & type as a given array.
5. **np.zeros\_like() -** A = np.zeros\_like(a) # It returns an array of Zeros with the same shape & type as a given array.
6. **np.full\_like() -** A = np.full\_like(a , 3) # It returns an array of Constant Values with same shape & type as a given array.
7. **.copy() -**  A = a.copy() # It returns a copy of the array.
8. **.diag() -**  A = np.diag(a) # It extracts the diagonal elements as a 1-D array.
9. **Operators - +, - , \* , / -** A = np.array([1,2,3]) ; B = A + 1 🡪 B = [2,3,4] ; C = A \* 2 🡪 C = [2,4,6]
10. **Transpose -**  a.T # Coverts the rows into columns and columns into rows.
11. **Unary Operators -** a.max() , a. max(axis=1), a.max(axis=0) , a.sum()

a.min() , a.min(axis=1) , a.min(axis=0) , np.sum(a, axis=1)

# These functions can be applied row-wise or column-wise by setting an axis parameter.

1. **stack -** c = np.stack( (a,b) ) # It creates a matrix using the arrays as rows.
2. **column\_stack -** c = np.column\_stack( (a,b) ) # It creates a matrix using the arrays as columns.
3. **vstack -**  c = np.vstack( (a,b) ) # It appends the data vertically. It creates 2-D array.
4. **hstack -** c = np.hstack( (a,b) ) # It appends the data horizontally.
5. **Array Indexing -** a[1:2,1:2,1:2] # Since arrays may be multidimensional, we must specify a slice for each dimension of the array.
6. **Mix-Integer Indexing -** a[1,1:2,1:2] # Mix integer indexing with Slice Indexing yields an array of lower rank. While, using only slices, it yields an array of same rank as the original array.
7. **Integer Array Indexing -** a[[0,1,2],[0,1,0]] # It allows us to construct arbitrary (random choice) array using the data from another array.
8. **Boolean Array Indexing -** a[a>2] # It is used to select the elements of an array that satisfy some condition.
9. **.dot() -** v.dot(w) = np.dot(v,w) , x.dot(v) = np.dot(x,v) , x.dot(y) = np.dot(x,y) # It is used to compute inner product of the vectors, to multiply a vector by matrix, & to multiply matrixes.
10. **random() -**

np.random.rand(10) # It creates an array of 10 random numbers between 0 and 1.

np.random.random(5) # It takes only one number x(5 here) & displays values equal to number quantity.

np.random.randint(5,20,4) # It displays given no. of values(4 here) between given input numbers 5 & 20.

np.random.randn(2,3,4) # It displays values (+/-) in the form of arrays.

np.random.uniform(1,5,50) # It displays given no. of unique values between given input numbers.

np.random.choice([‘x’,’y’,’z’], size=20 , replace=True/False) # It returns a random no. array.

np.random.normal( loc=100, scale=5 , size=10 ) # It draws a random sample form normal distribution.

1. **np.any(x > 0.9) #** It checks if any value is greater than 0.9 in x. ( x = np.random.random(10))
2. **np.all(x >= 0.9) #** It checks if all values are greater than or equal to 0.1 in x. ( x = np.random.random(10))
3. **array\_A[array\_A == x] = y #** Replacing all x in the given array\_A with y.
4. **a[[2,4]] or a[(1,3),:] #** Getting the values from 2nd and 4th row of the matrix.
5. **To get the results from the matrix :** a.sum(), a.std(), a.var(), a.mean(), a.max(), a.min()