**2.Numpy:**

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

* **Import:** We import third-party Python packages into Python files in the same way as we import modules from the Python Standard Library into Python files. we could make use of the numpy package to round numbers up and down.

**Ex:** import numpy as np

* “pip install package\_name” is the standard tool for installing python packages.

**Ex:** !pip install np

**2.1.Arrays:**

An array is a special variable, which can hold more than one value at a time.

**2.1.1.**

**#creating 1D Array(rows)**

a=np.array([4,5,6,7])

print(type(a))

**Output:**

<class 'numpy.ndarray'>

**#creating 2D Array (rows and columns)**

b=np.array([[1,2,3],[2,3,4]])

print(b)

**Output:**

[ [ 1 2 3 ]

[ 2 3 4 ] ]

**#creating 3D Array (rows,columns,groups)**

c=np.array([[[2,3,4],[3,4,5]],[[4,5,6],[5,6,7]]])

print(c)

**Output:**

[ [ 2 3 4 ]

[ 3 4 5 ] ]

[ [ 4 5 6 ]

[ 5 6 7 ] ]

**#checking dimensions**

print(a.ndim)

print(b.ndim)

print(c.ndim)

**Output:**

1

2

3

**2.1.2.Creating Arrays using zeros() and ones()**

* **zeros():**Return a new array of given shape and type, filled with zeros.

**Ex:**

f=np.zeros((3,2)) **#2D**

print(f)

**Output:**

[[0. 0.]

[0. 0.]

[0. 0.]]

* **ones():** function returns a new array of given shape and data type, where the element's value is set to 1.

**Ex-1:**

e=np.ones((3,2)) **#rows,columns,2D**

print(e)

output:

[[1. 1.]

[1. 1.]

[1. 1.]]

**Ex-2:**

d=np.ones((2,3,3)) **#groups,rows,colums #3D**

print(d)

**Output:**

[[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]]

**2.1.3.eye():**

It is used to return array with ones (1) on the diagonal and zeros (0) elsewhere.

**Ex-1: Ex-2:**

**#identity matrix** h=np.eye(4,3) **#4rows & 3columns**

j=np.eye(4) print(h)

print(j) **Output:**

**Output:**  [[1. 0. 0.

[[1. 0. 0. 0.] [0. 1. 0.]

[0. 1. 0. 0.] [0. 0. 1.]

[0. 0. 1. 0.] [0. 0. 0.]]

[0. 0. 0. 1**.**]]

**2.1.4.Arange()**:

This function generates a sequence of numbers within the specified range.

**Ex-1:**

i=np.arange(3,31,3) #start,stop,step

print(i)

**Output:**

[ 3 6 9 12 15 18 21 24 27 30]

**Ex-2:**

**#arange with reshape (we can change into rows and columns with our specification)**

i=np.arange(4,41,4).reshape(5,2)

print(i)

**Output:**

[[ 4 8]

[12 16]

[20 24]

[28 32]

[36 40]]

**2.1.5.linspace():**

used to create an array of evenly spaced numbers within a specified range.

**Ex-1:**

j=np.linspace(1,2,6)

print(j)

**Output:**

[1. 1.2 1.4 1.6 1.8 2. ]

**2.1.6.Operations:**

**Ex-1: Ex-2:**

k=np.arange(2,8).reshape(3,2) a=np.array([[1,1],[0,1]])

print(k) b=np.array([[2,0],[3,4]])

m=np.arange(13,19).reshape(3,2) print(a\*b)

print(m) print(a@b)

print(m+k) **Output:**

print(m-k) [[2 0]

print(m\*k) [0 4]]

**Output:** [[5 4]

[[2 3] [3 4]]

[4 5]

[6 7]] =>k

[[13 14]

[15 16]

[17 18]] =>m

[[15 17]

[19 21]

[23 25]] =>m+k

[[11 11]

[11 11]

[11 11]] =>m-k

[[ 26 42]

[ 60 80]

[102 126]] =>m\*k

**Ex-3:**

a=np.array([14,15,16,17,18])

b=np.array([25,26,27,28,29])

m=np.multiply(a,b)

print(m)

n=np.divide(a,b)

print(n)

o=np.mod(a,b)

print(o)

p=np.divmod(a,b)

print(p)

**Output:**

[350 390 432 476 522]

[0.56 0.57692308 0.59259259 0.60714286 0.62068966]

[14 15 16 17 18]

(array([0, 0, 0, 0, 0]), array([14, 15, 16, 17, 18]))

**2.1.7.axis():**

An axis is similar to a dimension. For a 2-dimensional array, there are 2 axes: vertical and horizontal.

* axis=0 => columns
* axis=1 => rows
* axis=2 => groups

**Ex-1:**

k=np.arange(2,8).reshape(3,2)

print(k)

m=np.arange(13,19).reshape(3,2)

print(m)

o=np.sum((k,m),axis=0)

print(o)

n=np.sum((k,m),axis=1)

print(n)

l=np.sum((k,m),axis=2)

print(l)

**Output:**

[[2 3]

[4 5]

[6 7]] =>k

[[13 14]

[15 16]

[17 18]] =>m

[[15 17]

[19 21]

[23 25]] #axis=0

[[12 15]

[45 48]] #axis=1

[[ 5 9 13]

[27 31 35]] #axis=2

**Problem:**

array[25,289,361,81] #find square roots and iterate through result values output=5 square is 25…..

**Code:**

b=np.array([25,289,361,81])

for i in b:

print((np.sqrt(i)),"square is",i)

**Output:**

5.0 square is 25

17.0 square is 289

19.0 square is 361

9.0 square is 81

**2.1.8.Array\_joins:**

we can join the arrays using hstack,vstack,dstack

* **hstack():** Stack arrays in sequence horizontally **(column).**
* **vstack():** Stack arrays in sequence vertically **(row).**
* **dstack():** Stack arrays in sequence depth wise **(along third axis).**

**Ex:**

a=np.array([23,24,25,26,27,28])

a.resize(2,3)

b=np.array([4,5,6,7,8,9])

b.resize(2,3)

print(np.vstack((a,b))) **#columns**

print(np.hstack((a,b))) **#rows**

**Output:**

[[23 24 25]

[26 27 28]

[ 4 5 6]

[ 7 8 9]]

[[23 24 25 4 5 6]

[26 27 28 7 8 9]]

**Ex-dstack():**

a=np.arange(10).reshape(2,3,5) **#groups,rows,columns**

print(a)

print(np.dstack(a))

**#no. of groups=2 => 2 columns in each group**

**#no. of rows=3 => 3 groups formed**

**Output:**

[[[ 0 1 2 3 4]

[ 5 6 7 8 9]

[10 11 12 13 14]]

[[15 16 17 18 19]

[20 21 22 23 24]

[25 26 27 28 29]]]

[[[ 0 15] [ 8 23]

[ 1 16] [ 9 24]]

[ 2 17] [[10 25]

[ 3 18] [11 26]

[ 4 19]] [12 27]

[[ 5 20] [13 28]

[ 6 21] [14 29]]]

[ 7 22]

**2.1.9.Random:**

* **Inside numpy random is subpackage**
* **rand is subpackage of random**
* **when we have to access subpackage we have to assign package name.**
* **subpackage range of rand function is 0-1.**

**Ex-1:**

a=np.random.rand(1)

print(a)

**Output:**

[0.61416485]

**Ex-2:**

a=np.random.rand(8,4) **#to return 8 rows 4 columns**

print(a)

**Output:**

[[0.93433303 0.53391377 0.21465164 0.86594388]

[0.92321908 0.55346386 0.67062962 0.43202116]

[0.15505824 0.19369611 0.62810361 0.83582954]

[0.99448691 0.25201506 0.33671444 0.97520452]

[0.67265109 0.62490786 0.08307684 0.89804854]

[0.12199127 0.86296616 0.83391205 0.59343497]

[0.31961844 0.89563866 0.99061707 0.01919016]

[0.33937943 0.17872777 0.71091489 0.67995342]]

**Ex-3:**

a=10\*np.random.rand(8,4) **#returns values btw 0-10**

print(a)

**Output:**

[[9.40697016 1.13512218 2.31711389 5.23188308]

[3.68216483 3.74073377 0.06615358 2.81885886]

[3.53912624 8.16258123 1.1067624 1.33469387]

[4.51775803 2.51472768 9.07335807 8.33590053]

[8.55439897 7.49695794 3.7650845 3.92876223]

[1.49263432 0.98197754 2.36428093 2.14099368]

[2.75462491 9.03773102 4.62109381 7.85427448]

[1.59905541 6.99528344 6.29385853 4.3080128 ]]

**Ex-4:**

a=np.ceil(10\*np.random.rand(8,4))  **#floor=min value,ceil=max value**

print(a)

**Output:**

[[ 9. 8. 10. 10.]

[ 8. 4. 6. 9.]

[10. 4. 2. 1.]

[ 3. 7. 7. 7.]

[ 8. 10. 6. 2.]

[ 7. 8. 3. 5.]

[ 8. 9. 8. 7.]

[ 5. 2. 2. 9.]]

**2.1.10.Splitting:**

It is 2 types:

* hsplit: Split an array into multiple sub-arrays horizontally (side-by-side)
* vsplit: Split an array into multiple sub-arrays vertically (top to bottom)

**Ex:(vsplit)**

A=np.arange(1,33).reshape(8,4) **#for vertical split**

print(A)

**Output:**

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16]

[17 18 19 20]

[21 22 23 24]

[25 26 27 28]

[29 30 31 32]]

**Code:**

np.vsplit(A,4) **#4 will divide 8 so we are getting 2 rows each for 4 arrays**

**Output:**

[array([[1, 2, 3, 4],

[5, 6, 7, 8]]),

array([[ 9, 10, 11, 12],

[13, 14, 15, 16]]),

array([[17, 18, 19, 20],

[21, 22, 23, 24]]),

array([[25, 26, 27, 28],

[29, 30, 31, 32]])]

**Code:**

np.vsplit(A,(3,4)) **#split it after 3rd row and 4th row**

**Output:**

[array([[ 1, 2, 3, 4],

[ 5, 6, 7, 8],

[ 9, 10, 11, 12]]),

array([[13, 14, 15, 16]]),

array([[17, 18, 19, 20],

[21, 22, 23, 24],

[25, 26, 27, 28],

[29, 30, 31, 32]])]

**Ex:(hsplit)**

B=np.arange(1,33).reshape(4,8) #for horizontal split

print(B)

**Output:**

[[ 1 2 3 4 5 6 7 8]

[ 9 10 11 12 13 14 15 16]

[17 18 19 20 21 22 23 24]

[25 26 27 28 29 30 31 32]]

**Code:**

np.hsplit(B,4) **#4 should divide 8 and we have 2 columns for 4 arrays**

**Output:**

[array([[ 1, 2],

[ 9, 10],

[17, 18],

[25, 26]]),

array([[ 3, 4],

[11, 12],

[19, 20],

[27, 28]]),

array([[ 5, 6],

[13, 14],

[21, 22],

[29, 30]]),

array([[ 7, 8],

[15, 16],

[23, 24],

[31, 32]])]

**Code:**

np.hsplit(B,(3,4,5)) **# split after 3rd column and 4th column and 5th column**

**Output:**

[array([[ 1, 2, 3],

[ 9, 10, 11],

[17, 18, 19],

[25, 26, 27]]),

array([[ 4],

[12],

[20],

[28]]),

array([[ 5],

[13],

[21],

[29]]),

array([[ 6, 7, 8],

[14, 15, 16],

[22, 23, 24],

[30, 31, 32]])]

**2.2.Trigonometry**

**Ex-1:**

np.pi

**Output:**

3.141592653589793

**Ex-2:**

a=[np.pi/4,np.pi/3,np.pi/2,np.pi]

print(a)

**Output:**

[0.7853981633974483, 1.0471975511965976, 1.5707963267948966, 3.141592653589793]

**Ex-3:**

b=np.rad2deg(a) **#convert radians to degree**

print(b)

**Output:**

[ 45. 60. 90. 180.]

**Ex-4:**

c=[ 45., 60., 90., 180.] **#converts deg to radians**

print(np.deg2rad(c))

**Output:**

[0.78539816 1.04719755 1.57079633 3.14159265]

**Ex-5:**

a=np.sin(90)

print(a)

b=np.cos(0)

print(b)

c=np.tan(1)

print(c)

**Output:**

0.8939966636005579

1.0

1.5574077246549023

**2.3.Statistics**

**Statistics:**

- mean: sum of observ/no. of observ

- median: after sorting the array middle value is median

- standard deviation: difference between n values and mean and sum them and divide the sum with n-1

- variance: sd\*\*2

**Ex:**

ar=np.array([23,13,9,6,28,26])

a=np.mean(ar)

print(a)

b=np.median(ar)

print(b)

c=np.var(ar)

print(c)

d=np.std(ar)

print(d)

**Output:**

17.5

18.0

72.91666666666667

8.539125638299666

#**linalg is also subpackage in linalg we have inverse subpackage**

c=np.arange(1,5).reshape(2,2)

print(c)

**Output:**

[[1 2]

[3 4]]

**Code:**

np.linalg.inv(c) **#inverse of a matrix**

**Output:**

array([[-2. , 1. ],

[ 1.5, -0.5]])

**2.4.argmax(),argmin():**

argmax=returns index position of max value in the array

argmin=returns index position of min value in the array

**Ex-1:(argmax())**

c=np.arange(1,25).reshape(6,4)

print(c)

**Output:**

[[ 1 2 3 4]

[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16]

[17 18 19 20]

[21 22 23 24]]

**Code:**

print(np.argmax(c)) **#returns index position of max value in the array**

**Output:**

23

**Ex-2:**

d=np.floor(10\*np.random.rand(24)).reshape(6,4)

print(d)

**Output:**

[[1. 0. 4. 7.]

[6. 7. 7. 5.]

[4. 7. 2. 9.]

[7. 0. 9. 4.]

[7. 7. 3. 7.]

[1. 3. 2. 4.]]

**Code:**

print(np.argmax(d))

**Output:**

11

**Ex-3:**

d=10\*np.random.rand(24).reshape(6,4)

print(d)

**Output:**

[[2.77018345 8.49919484 2.11128424 3.55558174]

[9.08639892 8.77191963 0.31168096 5.33982317]

[4.11690054 6.34883831 2.51194345 8.0748807 ]

[9.64406709 1.49927088 9.16440803 0.88408508]

[6.7473124 3.36862755 3.23085361 7.77078445]

[3.26114597 4.9444712 5.65359818 2.11846695]]

**Code:**

print(np.argmax(d,axis=0)) **#returns each column index position of max value**

**Output:**

[3 1 3 2]

**Code:**

print(np.argmax(d,axis=1)) **#returns each row index position of max value**

**Output:**

[1 0 3 0 3 2]

**Ex:(argmin())**

d=10\*np.random.rand(24).reshape(6,4)

print(d)

**Output:**

[[2.77018345 8.49919484 2.11128424 3.55558174]

[9.08639892 8.77191963 0.31168096 5.33982317]

[4.11690054 6.34883831 2.51194345 8.0748807 ]

[9.64406709 1.49927088 9.16440803 0.88408508]

[6.7473124 3.36862755 3.23085361 7.77078445]

[3.26114597 4.9444712 5.65359818 2.11846695]]

**Code:**

print(np.argmin(d)) **#returns min value in the array**

**Output:**

6

**Code:**

print(np.argmin(d,axis=0)) **#returns each column index position of min value**

**Output:**

[0 3 1 3]

**Code:**

print(np.argmin(d,axis=1)) **#returns each row index position of minvalue**

**Output:**

[2 2 2 3 2 3 ]

**2.5.Searching**

* You can search an array for a certain value, and return the indexes that get a match.
* To search an array, use the **where()** method.

**Ex-1:**

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

print(np.where(a%2==0))

**Output:**

(array([], dtype=int64),)

**Ex-2:**

a=np.array([24,16,7,17,54,60])

print(np.where(a%6==0))

**Output:**

(array([0, 4, 5]),)

* To search a sorted array, use the searchsorted() method.
* It is not applicable for unsorted array.

**Ex-1:**

a=np.array([1,2,3,7,8,9])

x=np.searchsorted(a,2)

print(x)

**Output:**

1

**Ex-2:**

a=np.array([24,16,7,17,54,60]) **#not applicable for unsorted**

x=np.searchsorted(a,17)

print(x)

**Output:**

3

**2.6.Sorting:**

* Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.
* The NumPy ndarray object has a function called sort(), that will sort a specified array.

**Ex:**

a=np.array([24,16,7,17,54,60]) **#numerical,1D**

d=np.array([[24,16,7],[17,54,60]]) **#2D**

b=np.array(['apple','cherry','banana']) **#string**

c=np.array([True,True,False]) **#boolean**

print(np.sort(a))

print(np.sort(b))

print(np.sort(c))

print(np.sort(d))

**Output:**

[ 7 16 17 24 54 60]

['apple' 'banana' 'cherry']

[False True True]

[[ 7 16 24]

[17 54 60]]

**2.7.Filter:**

* Getting some elements out of an existing array and creating a new array out of them is called filtering.
* In NumPy, you filter an array using a boolean indexed list.
* If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

**Ex:**

a=np.array([40,43,40,44,67,78])

filt=np.where(a%2==0)

print(filt)

**Output:**

(array([0, 2, 3, 5]),)

**Code:**

a[filt]

**Output:**

array([40, 40, 44, 78])

**Ex-2:**

**#zip is used to join two arrays,lists**

names=np.array(["jwalitha","vasavi","pavan","harsha"])

initials=np.array(["k","k","k","k"])

for i,j in zip(initials,names):

print(i,".",j)

**Output:**

k . jwalitha

k . vasavi

k . pavan

k . harsha

**2.8.Logarithms:**

a=np.arange(1,10)

print(a)

print(np.log2(a)) #log a base2

print(np.log10(a)) #log a base10

print(np.log(a)) #natural log a

**Output:**

[1 2 3 4 5 6 7 8 9]

[0. 1. 1.5849625 2. 2.32192809 2.5849625

2.80735492 3. 3.169925 ]

[0. 0.30103 0.47712125 0.60205999 0.69897 0.77815125

0.84509804 0.90308999 0.95424251]

[0. 0.69314718 1.09861229 1.38629436 1.60943791 1.79175947

1.94591015 2.07944154 2.19722458]

**2.9. CUMMULATIVE FUNCTIONS**

* cummulative sum
* cummulative product

**Ex:**

a=np.array([13,14,15,16,17])

x=np.cumsum(a)

print(x)

y=np.cumprod(a)

print(y)

**Output:**

[13 27 42 58 75]

[ 13 182 2730 43680 742560]

**Ex:**

**#difference**

a=np.array([20,13,45,60]) #e2-e1

x=np.diff(a)

print(x)

**Output:**

[-7 32 15]

**Ex-1:**

**#lcm**

a=455

b=665

x=np.lcm(a,b)

print(x)

**Output:**

8645

**Ex-2:**

a=np.array([455,665])

x=np.lcm.reduce(a)

**#reduce is used for taking multiple inputs and gives single output**

print(x)

**Output:**

8645

**Ex-1:**

#gcd

a=455

b=665

x=np.gcd(a,b)

print(x)

**Output:**

35

**Ex-2:**

a=np.array([455,665])

x=np.gcd.reduce(a)

**#reduce is used for taking multiple inputs and gives single output**

print(x)

**Output:**

35