List - Mutable Row x Colu...
Tuple - Immutable.

It store from list & Tuple

(i) Array()

⇒ import numpy as np

a = np.array([1,2,3,4]) ⇒ One-dimen list.
print(a)
array([1,2,3,4])

⇒ np.array([[1,2],[3,4]]) 2-d - array
print array([[1,2],
[3,4]])

⇒ np.array([1,2,3], dtype="complex")
print array([[1.+0.j, 2.+0.j, 3.+6.j]])

⇒ np.array((1,2,3)) tuple - 1-d array
print array([1,2,3])

(ii) Arrange() ⇒ Creates Array of evenly spaced value
like - a[range()] in a loop.
Arrange([start,], stop[, step,], dtype=None)
opt/optional compulsory opt opt.

⇒ import numpy as np.
np.arrange(1,10)
print array([1,2,3,4,5,6,7,8,9])

— stop value

=> np.arrange (3.0)
array ([0.0, 1.0, 2.0])

=> np.array (20, dtype = "complex")
array ([0.+0.j + 1.+0.j --- 19.+0.j])

=> np.arrange (1, 10, 2)
start stop gap
array ([1, 3, 5, 7, 9])

=> np.arrange (1, 10, 2, dtype = "float")

Zeros ()

=> Create Array filled as zeros

=> import numpy as np
help (np.zeros)

zeros (shape, dtype = "float", order = 'c')
Compl. opt.

=> np.zeros (5)
array ([0., 0., 0., 0., 0.])

=> np.zeros (3, dtype = "int") array ([0, 0, 0]

=> np.zeros ((2,3)) — ~~tuple~~ tuple.
array ([[0., 0., 0.],
[0., 0., 0.]])

⇒ np.zeros([3,4]) - list.

array([[0., 0., 0., 0.],
[0., 0., 0., 0.],
[0., 0., 0., 0.]])

Ones()

⇒ Creates Array filled with ones.

Same as - zeros().

Linspace()

⇒ "Create Array filled evenly spaced values." linspace [start, stop], num=50,
comp. include

⇒ import numpy as np

np.linspace(1, 100, num=5)

op: array([1., 25.75, 50.5, 75.25, 100.])

np.linspace(1, 100, num=5, endpoint=False)

op: array([1., 20.8, 40.6, 60.4, 80.2])

np.linspace(1, 100, num=4, retstep=True)

(array([1., 34., 67., 100.]), 33.0)

np.linspace(1, 10, num=4, dtype=int)

num default = 50

+ Numpy Random Module's -

(i) rand()
(ii) randn()
(iii) randf()
(iv) randint()

=> import numpy as np
help(np.random)

=> np.random.rand - Create an array of the given shape and populate it with random samples from a uniform distribution.

all positive value
np.random.rand(~~prob~~ shape)
np.random.rand(5)
array([5 0.--, 0.---, 0.95--, 0.---, 0.--])

np.random(3,8)
array([[----------]])

np.random.randn - contain positive and negative value. means a standard value.
array([1.4, 0.5, -1.14, 0.83, -1.51])

np.random.randf - return float value.

np.random.randint - return int value.
np.random.randint(5, size=(2,2))
array([[4,3], [2,0],]) random range - 0-4
size = array size.

+ Attribute of array's

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

⇒ a.ndim = 2
tell the dimension of array

⇒ Shape :-
Tell the ~~Array~~ number of rows, column and dimension

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

a.shape
⇒ (2,2,2)

⇒ Size :- Tell the ~~no of~~ total no' of elements

a = ~~arr~~ np.zeros([2,3,4])
a.size
= 2*3*4 = 20

⇒ Dtype :- a.dtype
= dtype('float')

itemsize ⇒ contain the memory
a.itemsize = 8
because the size of each element float is = 8 byte

Numerical Data types:-

(i) Boolean — bool value
(ii) Integers — Int16, int32, int64
(iii) float — float16, float32, float64
(iv) Unsigned integer :-
(v) complex

int = default 64 or 32 bit integer.
int8 = 8 bit integer - -128 to 127
int16 = -32768 to 32767
int32 = -2147483648 to 2147483647
int64 = -9223372036 854775808 to 9223372036 854775807

+ b = np.zeros(6, dtype = np.int8)

+ Use as a function to change data type

* x = np.float32(1)
print(x) = 1.0

+ If array is created then we change a datatype -
a.astype(int)
np.int(a)

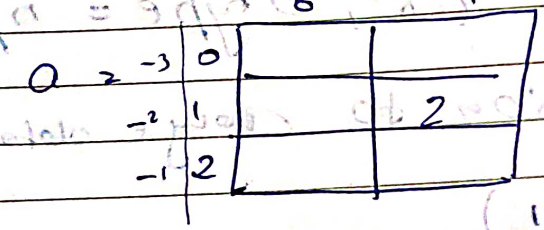
Indexing :-

(i) One - d array

a[3] = 4
a[-2] = 4

⇒ import numpy as np
a = np.array([1,2,3,4,5])

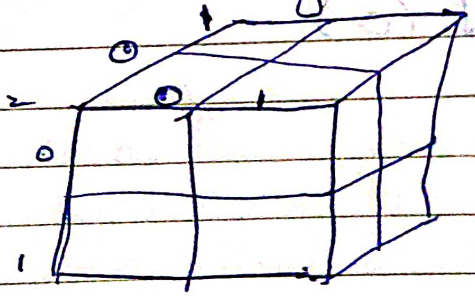
(ii) 2 - d array :-



a[1][1] = 2
a[-2][-1] = 2

a = np.array([[2,2],[2,1]])
a[0][0] = 2

(iii) 3 - d array



a[ ][ ][ ]

C = [[[i,2,3],[4,5,6]],
[[7,8,9],[10,11,12]]]

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

a[0][0][1] = 2
a[-2][-2][-2] = 2

Operation in Numpy :-

slicing :-

(i) 1-d array :

a[:] = full array.
a[1:] = starting index.
a[:a1] = ending index
a[1:3] = starts at 1 and at 2 because 3 is exclude.

```
a = np.array([1, 2, 3, 4, 5])
a[1::2] = start at 1 index = 2
          skip at 2 index - means ->
print(2, 4)
```

(ii) 2-d array gap / step

a[start:end:step, start:end:step]
row column

a[1:, 1:]

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

	0	1
0	1	2
1	3	4
2	5	6

i => 0 :
j => 1 :

x[:, :] = entire array.
x[::, ::2]
step - skip index 2nd position.

3-d array:

a[start : end : step , st : end : step , st : end : step]

a = [: , : , : 1]

+ Advanced Indexing:

Basic Index - slicing or simple call element
x[index] — integer number

Advance index - 2 type -
- Integer Index
- boolean Indexing

```
a = np.array([1,2,3,4,5,6])
b = np.array([1,3,5])
print(a[b]) = array([2,4,6])   pass index as
a[[1,3,5]] = array([2,4,6])    list in the form of list
c = array([[[1,2,3],
            [4,5,6],
            [7,8,9]]])
c[[0,2],[2,0]] = array([3,7])
                 0-2 1=3
                 2-0 = 7
Also c[[0,1,2],[0,1,2]] = array([1,5,9])
Also repeadly = a[[1,1,1,1]] = ([2,5,2,2])
```

Boolean a = np.array([[1,-2,3],
[4,-2,3]])

a[a<0] = array([-2,-2])

a[a>0]

a[a<0]*2 = array([-4, -4]) ... (1)

Arthematic operation on array :-
a = np.arrange(1, 5) using scalers
several operation
a-1, a*2, a/2, a+2, a**2 (square)
a%2, same operation as 2-d array.

b = np.arrange(6, 10)
Then add same shape of array.
= a+b ≠ a*b, a-b, a/b etc.
Same as 2-d array.

When shape is different of 2 arrays then we use Broadcasting :-

Broadcasting rules -(i) size of each dimension be same.
(ii) size of one of the dimension is 1

Rule-1 shape :
a = (3,2) , c = (1,2) ~~right~~ one dimension ~~left side of~~ will be same.
dimension. one of dimension is 1

Rule 2 - a=(3, 2) , b = (1, 3)
3≠2 so we can't add this array
1-dimension is one satisfy - so this shows
Rule-3 a = (3,2) b = (2,3) show error.
because of one-dim.

~~So Solve this problems :~~
If rule satisfy :-

a = (3, 1) b = (1, 3)
So, we stretch the array.
a = (3, 3) b = (3, 3) copy element

+ Array Manipulation
(i) Reshaping, spliting, sorting, shifting.
Add/Remove, joining

+ Reshape :- 1-d array to 2-d array
numpy.reshape(array, shape, order)
return ndarray
* (Sometime it copying the ~~as~~ element) *
a = np.arrange(10)
a.shape = (10,)
b = np.reshape(a, (5, 2)) default order = 'C' row wise
Also a.reshape((5, 2)) If column wise
size of element be same in both array otherwise Operation'
It cannot change data of array only change a shape otherwise it show error

Resize :- It will be change the data
numpy.resize(arrayname, Shape)
=> a = np.array(5) — contain - selement
np.resize(a, (2, 3)) — contain - 6 - elem
array([[0, 1, 2]
[3, 4, 0]]) In this start with initial.
=> a.resize((2, 3)) then it change the version of array copy the elem at every position

+ flattening :- flatten method - ~~copy~~
collapse to the one-dimension
2-d or 3-d to 1-d array
np.flatten (order = 'c')
K - order occurred in memory. C - Row major
F - column major

a = ([[[2, 3], [3, 4]]])
a.flatten() = ([2, 3, 3, 4])

=) ravel :- If it is necessary then copy it.
np.ravel (a, order)
a.ravel (order)
difference - If it need copy it is function
flatten is a object

+ Transpose :- numpy.transpose (array, axes = None)
It reverse the dimensions.

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

If b = b.shape (2, 3, 4)
after transpose (b).shape = (4, 3, 2)
only use in 2-d, 3-d, 4-d arr.

for 2-d axis = (0, 1) reverse
= (1, 0) default same order.

Swap axis = ~~multi~~ numpy.swapaxes (arr, axis1, axis2)
" " (a, 0, 1)

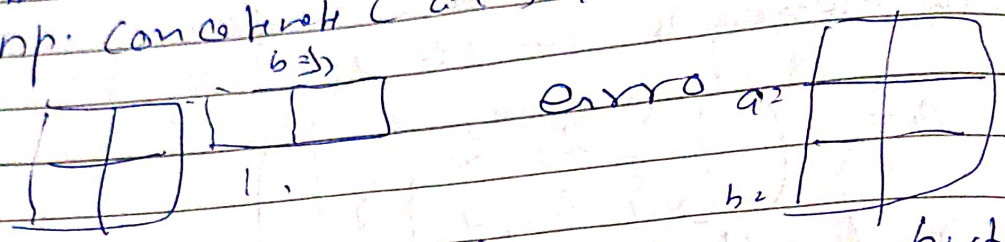
+ Joining or Concatenate of array :—

Op. ~~dump~~ np. Conatenate (array, axis = 0, out = None)

a = [1, 2, 3, 4, 5]

b = [1, 2, 3, 4, 5) c = np.zeros (10)

np. concatenate (a+b, out = c)



+ np.vstack (tuple) Same as but not default axis = 0 => change the array only concate the column.

Ex. shape (5, 1) shape (5, 1) = (5, 2)
(5, 1) (6, 1) = error

+ np.hstack = horizontally.
1-d array = axis = 0 shape 1-d.
2-d, 3-d - axis = 1

+ split () np.split (array, section, axis)
a = np.arange (10)
np.split (a, 2) = (1 — 5) (6 —
, 3 = error multiple

vsplit, hsplit :

Insert :- np.insert (array, obj, value, index axis = None)

np.insert (a, i, 50)
Multiple (1, 3)

For 2-d initially flatten then insert if ✓

append() ⇒ np.append(array, value, axis = None)

insert in last position
at = (0, [[2,3]], axis=0)

delete() ⇒ np.delete(array, obj, axis = None)
index.

Matrix :- Rectangular array of data.

Matrix Addition - simple addition 2 array of 2-d. ⇒ a+b

Matrix Multiplication are different, we use 'dot' method.

1	2
3	4

10	20
30	40

⇒

70	100
150	220

a*b = simple Multiplication
a.dot(b) = matrix Multiplication

transpose :- which flip the matrix.
Same as - x.T

np.matrix(data, dtype=None, copy=True)
a = np.matrix([[1,2],[3,4]]) - astuple also
matrix([[1,2],[3,4]])

b = matrix ⇒ a*b.
⇒ a*b used dot
if we use array
if we use matrix class then use *

Transpose = a.T = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ = $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$.

+ Inverse :- $A \cdot A^{-1} = I$ Identity Matrix

⇒ numpy.linalg.inv(a)
inverted a matrix
a([[1, 2], [3, 4]]) = ([[-2, 1], [1.5, -0.5]])

+ power = np.linalg.matrix_power(a, n) magic func
a**b = a.a n = power
For neg. first inverse then square of it.

+ Linear Equation :- x+y = 10, 2x+2y = 20
find out the x & y value
np.linalg.solve(a, b) a - array coefficient
b - dependent array

eg - 3x + y = 9 $a = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$ b = [9, 8]
x + 2y = 8 x = [x, x]
np.linalg.solve(a, b) = array([2., 3.])
x = 2, y = 3

also solve big equations.

+ Determinant of a Matrix :- It is calculated the diagonal of a matrix.
Square Matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ = |A| = ad - bc
np.linalg.det(a)
round() = round of the value.

$\begin{Bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{Bmatrix}$ = a{ei - fh} - b(di - fg) + c(dh - eg).