```
import numpy as np
a = np.arange(6)
b = np.arange(5)
c = np.arange(5)
np.hstack((a,b)) → o/p
np.hstack((a,c)) → o/p

import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([[5,6]])
np.concatenate((a,b.T), axis=1)
np.hstack((a,b.T))
```

Same.

Vstack → 1d → same shape ⇒ 2d as o/p
→ 2d,3d → same as concatenation, axis=0

hstack → 1d → anyshape → it will join
→ 2d,3d → same as concatenation, axis=1

splitting:-

split()

np.split(array, section, axis=0) ← subarray.

```
import numpy as np
a = np.arange(1,10)
np.split(a,3) → 3 sub arrays
np.split(a,4) → error
```

```
import numpy as np
a = np.arange(1,13).reshape(6,2)
a →
np.split(a,2) → o/p
```

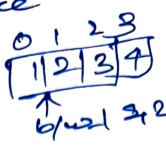
→ np.vsplit(a,2)
np.hsplit(a,3)
(or)

numpy.hsplit(array, section)
numpy.vsplit(array, section)

Array manipulation

- insert
- delete

insert :- → copy of the array original. not the



insert()

np.insert(array, object, values, axis = None)
=1 → new value

```
import numpy as np
a = np.arange(1,11)
a
np.insert(a, 1, 50) → o/p
```

np.insert(a, 1, 50.5) → convert to int then it will insert

np.insert(a, (1,3), 50)
↳ 1st index → 3rd index

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

np.insert(a, 1, 23) → whatever the dimension it will flatten then adding → (2D, 3D...)

np.insert(a, 1, 23, axis=0) → 2D correct

np.insert(a, 1, 23, axis=1) → 2D correct

np.insert(a, [1,2], axis=0) → o/p

np.insert(a, 1, [1,2,3], axis=0) → error
↳ dimension change

append → if I need to add some no at the end of the array.

append()

np.append(array, values, axis=None)

```
import numpy as np
a = np.arange(1,11)
a
```

np.append(a, 3452) → o/p

a → is just copied

import numpy as np

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

np.append(a,[[4,5]], axis=0) = (4,5)

np.append(a, [[4,5]], axis=1) → error

1	2	4	5
3	4		

X

np.append(a, [[4],[5]]) → 1d array as no axis

np.append(a, [[4],[5]], axis=1) → o/p

1	2	4
3	4	5

Delete:

delete()

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

import numpy as np

a = np.arange(1,11)

a

np.delete(a,2) → index position

a → orginal not deleted.

import numpy as np

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

a

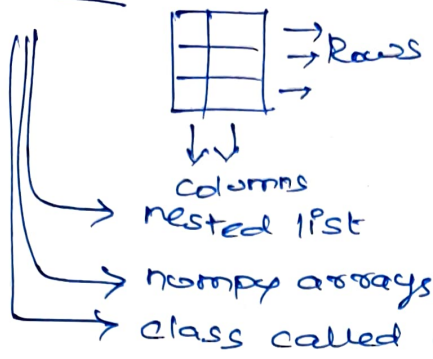
np.delete(a,2) → 124 (no axis)

np.delete(a,2,axis=0) → error

np.delete(a,1,axis=0) → o/p

np.delete(a,1,axis=1) → o/p.

MATRIX → Rectangular arrangement of data/no.



1	2
3	4

(2×2)

1	2	3
4	5	6

(2×3)

① numpy array:-

matrix addition a+b

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([[10,20],[30,40]])
a+b (element by element)
```

matrix multiplication a*b (array)

a dot b (dot)

a	
1	2
3	4

b	
10	20
30	40

→ o/p

10	40
90	160

element by element

1×10+2×30 1×20+2×40

70	100
150	220

3×10+4×30 3×20+4×40

```
import numpy as np
a = np.array([[10,20],[30,40]])
b = np.array([[1,2],[3,4]])
a*b → element by element
a.dot(b) → matrix multiplication
```

help(np.dot)

transpose

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

② Matrix

```
np.matrix(data, dtype = None, copy = True)
import numpy as np
a = np.matrix("1 2; 3 4")
a
b = np.matrix([[10,20],[30,40]])
b
```

addition

```
a+b
```

multiplication

```
a*b
```

transpose

```
a.T
b.T
```

Difference with numpy array,

1. matrix can be created using string notation.
2. matrix objects are always 2D
3. can use * for matrix multiplication

```
help(np.matrix)
```

Linear algebra:-

numpy.linalg
- Invert
- Power
- linear equation
- determinant of matrix
- eigen values & many more.

Inverse:-

np.linalg.inv( ) → inverse the given matrix

```
import numpy as np
a = np.array([[1,2],[3,4]])
a
b = np.linalg.inv(a)
```

$A \cdot A^{-1} = I$

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \frac{1}{4-6} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$$

$$= \frac{-1}{2} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -1 \\ 1.5 & -0.5 \end{bmatrix}$$

np.dot(a,b)

```
import numpy as np
a = np.array([1,2,3,4])
np.linalg.inv(a) → error
```

help(np.linalg.inv)

Power of matrix:- Used to find the power of matrix.

np.linalg.matrix_power(a,n)

- n → 0 → identity matrix
- n > 0 →
- n < 0 → inverse matrix then power

```
import numpy as np
a = np.array([[1,2],[3,4]])
np.linalg.matrix_power(a,2)   ← same → np.dot(a,a)
np.linalg.matrix_power(a,3)
np.linalg.matrix_power(a,0) → identity
np.linalg.matrix_power(a,-2)  ← same
```

⇓

```
b = np.linalg.inv(a)
b
np.linalg.matrix_power(b,2)
```

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a
np.linalg.matrix_power(a,2) → error
```

Linear equation:

↳ equation of a straight line

$$x + y = 10$$
$$2x + 2y = 20$$

np.linalg.solve(a,b)

$Ax = B$ → constant
↓ ↓
values variables

$$3x + y = 9 \longrightarrow 3x + y = 9$$
$$x + 2y = 8 \qquad y = 9 - 3x$$

$$a = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \quad b = [9\ 8]$$

$x + 2y = 8$
$x + 2(9 - 3x) = 8$
$x = 2$

```
import numpy as np
a = np.array([[3,1],[1,2]])
b = np.array([9,8])
np.linalg.solve(a,b) → (x,y)
```

$$6x + 2y - 5z = 13$$
$$3x + 3y - 2z = 13$$
$$7x + 5y - 3z = 26$$

$$a = \begin{bmatrix} 6 & 2 & -5 \\ 3 & 3 & -2 \\ 7 & 5 & -3 \end{bmatrix} \quad b = [13\ 13\ 26]$$

```
x = np.array([[6,2,-5],[3,3,-2],[7,5,-3]])
y = np.array([13,13,26])
np.linalg.solve(x,y) → o/p
```

Determinant:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 1 \times 4 - 2 \times 3 \rightarrow 4 - 6 \rightarrow -2$$

np.linalg.det(a) → square matrix

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

2D

```
a
np.linalg.det(a)
round(np.linalg.det(a))
b = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

3D

```
b
np.linalg.det(b)
round(np.linalg.det(b))
```

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

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = 1(45-48) - 2(36-42) + 3(32-35)$$

$$= -3 - 2(-6) + 3(-3)$$

$$= -3 + 12 - 9 = 0$$

images,

```
from matplotlib.image import imread
picture = imread('a.png')
print(type(picture))
picture
picture.size
picture.shape
picture.ndim
```