Numpy Functions :-

⇒ **Numpy** (**Numerical python**) is a powerful python library for scientific computing.

It supports

1) multidimensional arrays (ndarray)
2) High speed mathematical operations
3) Broadcasting and Vectorization
4) Lineal algebra, statistics, Fourier transforms, etc.

→ **Why Numpy?**

1) Fast computations (implemented by c)
2) memory efficient compared to python lists
3) supports vectorized operations (no explicit loops)
4) widely used in machine learning, AI, and datascience.

→ import numpy as np.

Creating Arrays :-

1) np. array ()

creates a array from a list

eg:- np. array([1,2,3])

2) np. arange (start, stop, step)

Generates evenly spaced values.

eg:- np. arange (1,10,2)

3) np. linespace (start, stop, num)

Linearly spaced numbers.

eg:- np. linespace (0,10,5)

4) np. ones ()

Creating an array with ones.

eg:- np. ones(5)

5) np.zeros()

Creating an array with zeros.

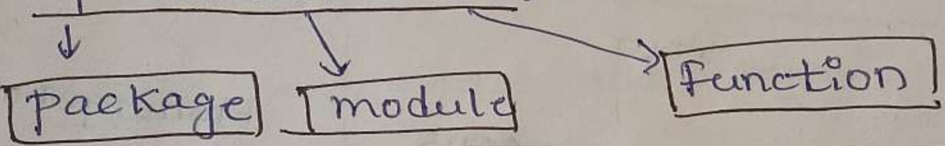
eg:- np.zeros(3)

6) np.identity(n)

identity matrix

eg:- np.identity(3)

7) np.random.rand()



=> generate random numbers (float)

eg:- np.random.rand(3)

8) np.random.randint()

=> generate random numbers (integers)

eg:- np.random.randint(5).

⇒ **dtype parameter**

specifies the data type of array elements.

ex dtype = float
dtype = int ...

Python list (vs) Numpy arrays.

Feature	Pythonlist	Numpy array
speed	slow	Fast (C backend)
Type	Heterogenous	Homogeneous
memory usuage	High	low
Vectorization	No	Yes
Broadcasting	No	Yes.

Array Attributes :

1) ndim ---> gives the number of dimensions

print(a.ndim)

2) size --> Gives total number of elements

eg:- a.size

3) shape --> Returs the tuple of array dimensions.

a.shape

4) dtype --> shows data type

5) itemsize --> Memory in bytes pee item

Type conversion

.astype() ---> converts the data to another data type

Arithmetic operations.

1) addition --> a+b

2) subtraction → a-b

3) Multiplication → a*2

4) Division → a/2

5) power → a**2

6) modulus → a%2

Relational operations.

compare arrays element-wise, returns boolean values.

1) <=

2) >=

3) <

4) >

5) !=

6) ==

Matrix operations.

1) np.dot()

matrix multiplication
dot product.

2) np.exp()

e^x → exponentials (powers)

3) np.log()

Natural log → Logarithm

Aggregation functions

1) np.max() → gives max value

2) np.min() → gives min value

3) np.sum() → Return sum value

4) np.prod() → Return product

5) np.mean() → given mean value

6) np.median() → median value.

7) np.std() → standard deviation.

8) np.var() → Variance

Trigonometric Functions

1) np.sin()

eg: np.sin(np.array([0, np.pi/2]))

⇒ [0,1]

2) np.cos()

eg: np.cos(np.array([0, np.pi/2]))

⇒ [1,0]

3) np.tan()

eg: np.tan(np.array([0, np.pi/4]))

→ [0,1]

Rounding Functions

1) np.round()

Round the nearest values.

eg: np.round([1.2, 2.7]) ⇒ [1,3]

2) np.floor()

Round the values down

eg: np.floor([2.9]) ⇒ [2]

3) np.ceil()

Round the value upwards.

eg: np.ceil([2.1]) ⟶ [3]

Iteration

np.nditer()

Reshaping arrays :-

1) reshape() ⟶ chage the dimensions

2) transpose() ⟶ swap rows to columns & columns to rows.

3) ravel() ⟶ convert any dimension array to 1D array.

Stacking & spliting

Horizontal & Vertical stacking

1) np.hstack()

2) np.vstack()

Horizontal & Vertical spliting

1) np.hsplit()

2) np.vsplit()

working with missing values.

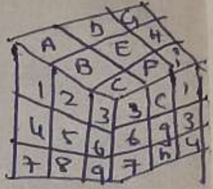
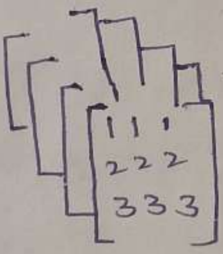
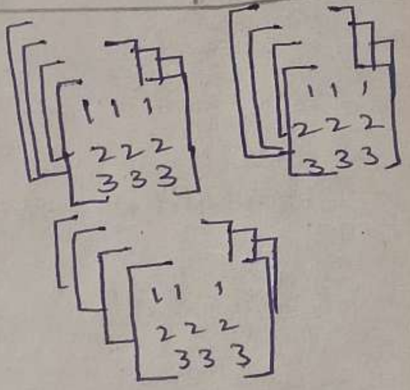
```
>> s = np.array([1, 2, 3, 4, np.nap, 6]
>> np.isnan(s)
>> [False, False, False, False, True, False]

>> S[np.isnan(S)]
>> array([nan])                      # nan values.

>> S[~np.isnan(S)]                   # Not nan values.
>> array([1., 2., 3., 4., 6.])         using '~'
```

=> Numpy (vs) PyTorch Tensor :-

Feature	Numpy array (nd array)	Pytorch Tensor (torch.Tensor)
defination	core data structure in Numpy for Numerical computation	core data structure in PyTorch for numerical computation and deep Learning
Library	comes from numpy	comes from torch
purpose.	general-purpose scientific computing (cpu only)	machine learning, automatic, differentiation, and GPU acceleration. GPU (using .cude() (or) .to('cuda')]

(11) scalar	$\begin{bmatrix}5 & 3 & 7\end{bmatrix}$ Row vector (shape 1x3)	$\begin{bmatrix}5 \\ 7 \\ 2\end{bmatrix}$ column vector (shape 3x1)	$\begin{bmatrix}4 & 19 & 18 \\ 16 & 3 & 5\end{bmatrix}$ matrix	 A B C D E F G H 1 2 3 4 5 6 7 8 9 Tensor.
4 Scalar (0D Tensor)	$\begin{bmatrix}1 \\ 2 \\ 5 \\ 9\end{bmatrix}$ vector (1D Tensor)	$\begin{bmatrix}1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 9 & 9 & 9\end{bmatrix}$ Matrix (2D Tensor)	 Tensor (3D Tensor)	 Tensor (ND tensor)

Numpy workshop

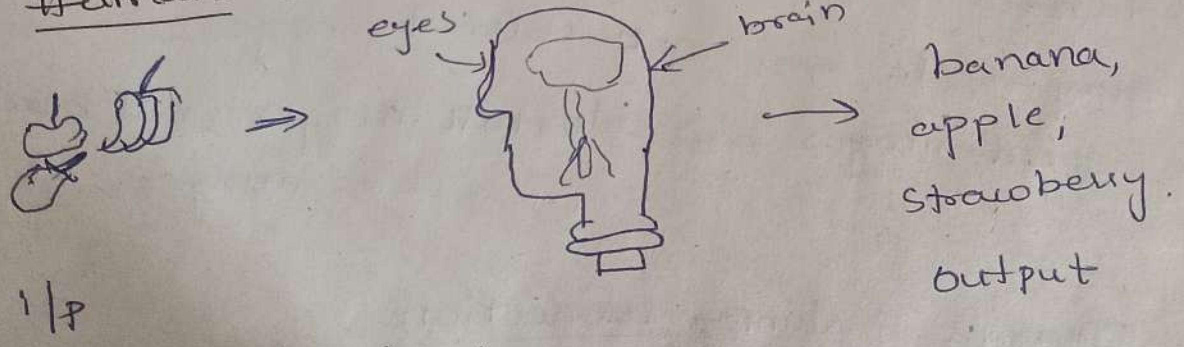
→ Exploring genrative Ai through computer vision

Agenda :-

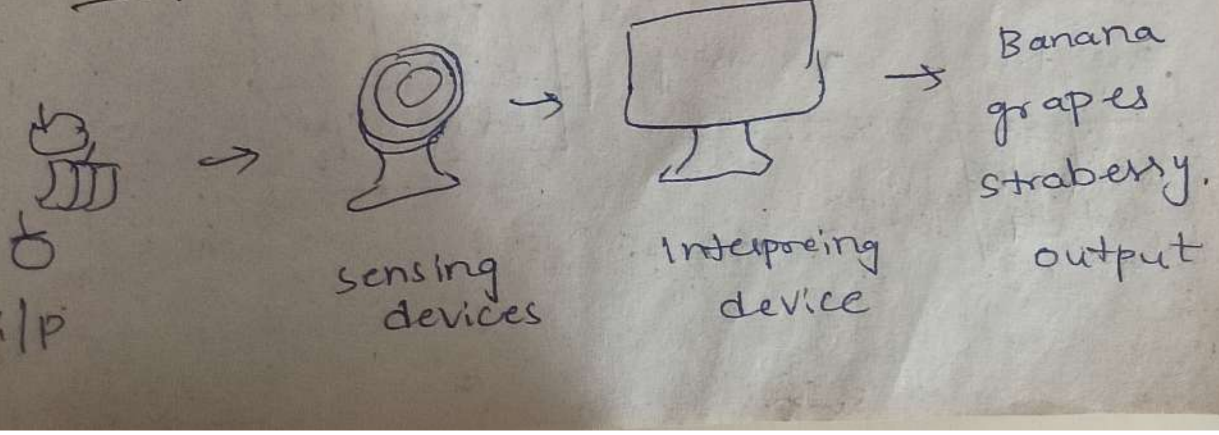
1) introduction to computer vision
2) what is image
3) Numpy & image connection
   - Image reading with Numpy & matloplib
4) introducte to opencv
   - image processing
   - opening image file with opencv
   - video detection using opencv

1) computer vision

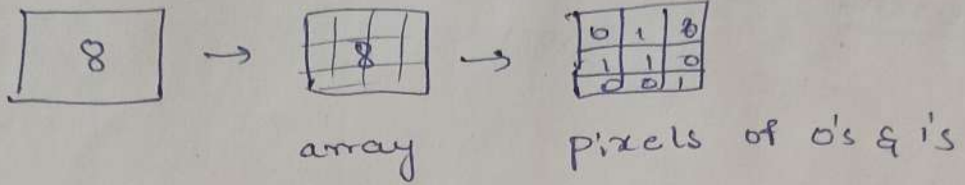
Human vision :-



computer vision :-



2) what is image :-



→ image ⇒ array ⇒ pixels.

→ pixels range between (0-255)

0 → completly black

255 → bright color.

- dark red color → (200-255)
- light red color → (0-10, 0,20)

→ 2d channel - black & white

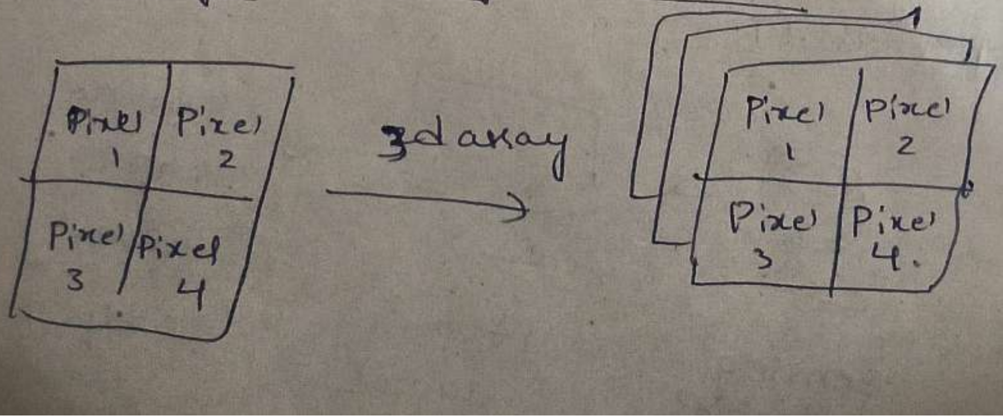
→ 3d channel - R, g, b (red, green, blue)

→ rainbow nature create this color

→ gray-scale ⇒ not colored images.

→ colored images ⇒ colorfull images. (not black & white images.

3) Image & Numpy connection



→ matplotlib → Visualization

→ %matplotlib inline (all the picture keep inside)

→ PIL (python imaging library)

→ ~~google →~~

→ upload the image

→ text prompt to image generation.

→ image to array

→ shape of an image

→ values of image we found in array

→ matplotlib colormap_reference.

→ cmap = "gray"