Introduction to NumPy & Numerical Operations

What is NumPy?

NumPy stands for **Numerical Python**.
It is a powerful Python library used for working with **numbers**, especially **arrays** (lists of numbers) and **matrices** (2D arrays).

It is fast and efficient for **mathematical operations** like:

- Addition, subtraction
- Mean, standard deviation
- Matrix multiplication
- Reshaping arrays

Why Use NumPy?

- Faster than Python lists
- Uses less memory
- Has built-in functions for math operations
- Makes it easy to work with large data sets

Installing NumPy

You can install NumPy using pip:

pip install numpy

Importing NumPy

We generally import NumPy like this:

import numpy as np

From a Python list:

import numpy as np

print(np._version_)

Direct use of numpy

```
import numpy
a=numpy.array([1,2,3,4,5])
print(a)
print(type(a))
```

Numpy use as np

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

Numpy => numerical python

```
import numpy
a = numpy.array([1, 2, 3, 4, 5])
print(a)
print(type(a))
```

Create a 0-D array with value 42

```
import numpy as np
arr = np.array(42)
print(arr)
```

1-D array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D array

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

Create a 3-D array with two 2-D arrays

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
```

array filled with zeros

```
import numpy as np
arr_zero=np.zeros(4)
print(arr_zero)


arr_zero1=np.zeros((3,4))
print(arr_zero1)
```

array filled with ones 1's

```
import numpy as np
arr_ones=np.ones(4)
print(arr_ones)
```

array diagonal element filled with 1's

```
import numpy as np
ar_dia=np.eye(4)
print(ar_dia)
```

create an array with values that are spaced linearly a specified interval

```
import numpy as np

ar_lin=np.linspace(0,10,num=11)

print(ar_lin)
```

NumPy Random Numbers in Arrays

NumPy provides several functions to generate **random numbers**, which are useful in simulations, testing, machine learning, etc.

random numbers in array

rand(): Random Numbers from 0 to 1

```
import numpy as np

var=np.random.rand(4)

print(var)
```

randn() returns result in positive and negative[Random Numbers from Normal Distribution (can be positive or negative)]

```
import numpy as np

var=np.random.randn(4)

print(var)
```

randint() result in random numbers in given range

```
var=np.random.randint(min,max,total_value)

import numpy as np

var=np.random.randint(3,30,10)

print(var)
```

Explanation:

- randint(3, 30, 10) means:

- **Minimum value = 3**
- **Maximum value (exclusive) = 30**
- **Total numbers = 10**

- It returns an array of **10 random integers** from **3 to 29**.

Function	Description	Values Type	Range	Example
rand(n)	Random floats	0 to 1 (positive only)	Float	np.random.rand(4)
randn(n)	Random floats (normal distribution)	Both positive & negative	Float	np.random.randn(4)
randint(a,b,n)	Random integers in range [a, b)	Integer only	From a to b-1	np.random.randint(3,30,10)

NumPy shuffle() – Shuffle Elements in an Array

The np.random.shuffle() function is used to **randomly change the order** of elements in an array. It's useful when you want to **randomize your dataset** (for example, in machine learning).

```
import numpy as np

var = np.array([1, 2, 8, 3, 9, 2, 78, 45, 23, 79])

np.random.shuffle(var)

print(var)
```

NumPy Arithmetic Operations – 1D and 2D Arrays

NumPy makes it super easy to perform arithmetic like +, -, *, / on arrays—either with:

- A **single number** (called a scalar)
- **Two arrays** (element-wise operations)

1D Array Arithmetic Operations

Add Scalar to Array

```
import numpy as np
var = np.array([1, 2, 3, 4])
varadd = var + 3
print(varadd)
```

Explanation: Adds 3 to each element → [4, 5, 6, 7]

Add Two Arrays

```
var1 = np.array([1, 2, 3, 4])
var2 = np.array([5, 6, 7, 8])
varadd = var1 + var2
print(varadd)
```

Explanation: Adds each corresponding element → [6, 8, 10, 12]

Subtract Scalar from Array

```
var = np.array([1, 2, 3, 4])
varsub = var - 3
print(varsub)
```

Result: [-2, -1, 0, 1]

Subtract Two Arrays

```
varsub = var1 - var2
print(varsub)
```

Result: [-4, -4, -4, -4]

Multiply Array by Scalar

```
var = np.array([1, 2, 3, 4])
varmul = var * 3
```

print(varmul)

Result: [3, 6, 9, 12]

Multiply Two Arrays

varmul = var1 * var2

print(varmul)

Result: [5, 12, 21, 32]

Divide Array by Scalar

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

vardiv = var / 3

print(vardiv)

Result: [0.33, 0.66, 1.0, 1.33] (approx)

Divide Two Arrays

vardiv = var1 / var2

print(vardiv)

Result: Element-wise division → [0.2, 0.33, 0.43, 0.5] (approx)

2D Array Arithmetic Operations

2D arrays are like **tables (matrices)**.

Add Scalar

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

varadd = var + 3

print(varadd)

Result:

[[4 5 6 7]

 [8 9 10 11]]

Add Two Arrays

var1 = np.array([[1,2,3,4],[5,6,7,8]])

var2 = np.array([[5,6,7,8],[5,6,7,8]])

varadd = var1 + var2

print(varadd)

Result:

[[6 8 10 12]

 [10 12 14 16]]

Subtract Scalar

varsub = var - 3

print(varsub)

Result:

[[-2 -1 0 1]

 [2 3 4 5]]

Subtract Two Arrays

varsub = var1 - var2

print(varsub)

Result:

[[-4 -4 -4 -4]

 [0 0 0 0]]

Multiply by Scalar

varmul = var * 3

print(varmul)

Result:

[[3 6 9 12]

 [15 18 21 24]]

Multiply Two Arrays

varmul = var1 * var2

print(varmul)

Result:

[[5 12 21 32]

 [25 36 49 64]]

Divide by Scalar

vardiv = var / 3

print(vardiv)

Result:

[[0.33 0.66 1.0 1.33]

 [1.66 2.0 2.33 2.66]]

Divide Two Arrays

vardiv = var1 / var2

print(vardiv)

NumPy shape and reshape() – Mastering Array Dimensions

The .shape attribute tells you the **structure of the array** — how many **rows and columns** it has.

Example: 2D Array

import numpy as np

```
var = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(var)
print()
print(var.shape)
```

Example: High-Dimensional Array

```
var = np.array([1, 2, 3, 4], ndmin=4)
print(var)
print(var.ndim)
print(var.shape)
```

What is reshape()?

The reshape() function is used to **change the shape** of an array **without changing its data**.

Convert 1D → 2D

```
var = np.array([1, 2, 3, 4, 5, 6])
print(var)
print(var.ndim)
x = var.reshape(3, 2)
print(x)
print(x.ndim)
```

Explanation:

- 1D → 2D with 3 rows and 2 columns
- Shape changes from (6,) to (3, 2)

Convert 1D → 3D

```
var = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
print(var)
```

```
print(var.ndim)
x = var.reshape(1, 4, 3)
print(x)
print(x.ndim)
```

Explanation:

- 1D → 3D with shape (1, 4, 3)
    - 1 block
    - 4 rows
    - 3 values in each row

Convert 3D → 1D using reshape(-1)

```
var = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
x = var.reshape(3, 2, 2)
print(x)
print(x.ndim)
one = x.reshape(-1)
print(one)
print(one.ndim)
```

Explanation:

- reshape(-1) **flattens** the array into 1D.
- Automatically adjusts dimensions based on total elements.

Convert 2D → 1D using reshape(-1)

```
var = np.array([1,2,3,4,5,6])
x = var.reshape(3, 2)
print(x)
print(x.ndim)
```

```
one = x.reshape(-1)

print(one)

print(one.ndim)
```

Result:

- 2D to 1D: reshaped to [1 2 3 4 5 6]

Action	**Code Example**	**Output Shape**
Check shape	array.shape	(rows, columns)
Reshape 1D → 2D	arr.reshape(3,2)	(3, 2)
Reshape 1D → 3D	arr.reshape(1,4,3)	(1, 4, 3)
Reshape any → 1D	arr.reshape(-1)	Flat 1D
Create high dim (ndmin)	np.array([1,2], ndmin=4)	(1,1,1,2)

NumPy Indexing – Accessing Elements in Arrays

1D Array Indexing

```
import numpy as np

var = np.array([9, 8, 7, 6])

print(var[1])     # Positive Indexing

print(var[-3])    # Negative Indexing
```

Explanation:

- var[1] → Indexing starts from 0, so index 1 = **8**
- var[-3] → Negative indexing starts from right, so -1 = 6, -2 = 7, -3 = **8**

2D Array Indexing

```
import numpy as np

var = np.array([[9, 8, 7, 6], [1, 2, 3, 4]])
```

```
print(var[0, 1])  # Row 0, Column 1
```

Explanation:

Visual of the array:

```
[
 [9, 8, 7, 6],    # Row 0
 [1, 2, 3, 4]     # Row 1
]
```

- var[0, 1] → First row, second element = **8**

Syntax: array[row, column]

3D Array Indexing

```
import numpy as np
var = np.array([[[9, 8, 7, 6], [1, 2, 3, 4], [11, 12, 13, 14]]])
print(var[0, 1, 1])  # Block 0, Row 1, Column 1
```

Explanation:

Visual structure:

```
[
 [
  [9, 8, 7, 6],     # Row 0
  [1, 2, 3, 4],     # Row 1
  [11, 12, 13, 14]  # Row 2
 ]
]
```

- var[0, 1, 1] → Block 0 → Row 1 → Second element = **2**

Syntax: array[block, row, column]

Type	Syntax	Example	Meaning

1D	arr[index]	arr[2]	3rd item from array
2D	arr[row, col]	arr[1,2]	2nd row, 3rd column
3D	arr[b,r,c]	arr[0,1,3]	Block 0, row 1, column 3

NumPy Slicing – Extracting Parts of an Array

Slicing means selecting **a range of elements** from an array.
The basic syntax is:

array[start : end : step]

It includes start, excludes end, and uses the step to jump through elements.

1D Array Slicing

```
import numpy as np

var = np.array([10, 20, 30, 40, 50, 60, 70])

print(var[2:6:1])  # from index 2 to 5

print(var[2:])     # from index 2 to end

print(var[:6])     # from start to index 5
```

Explanation:

Index	**Value**
0	10
1	20
2	30
3	40
4	50
5	60
6	70 (excluded in [2:6])

2D Array Slicing

```
var = np.array([[9, 8, 7, 6], [1, 2, 3, 4]])
print(var[0, 0:3])  # Row 0, columns 0 to 2
```

Explanation:

We're accessing:

- Row 0 → [9, 8, 7, 6]
- From column 0 to 2 → 9, 8, 7

3D Array Slicing

```
var = np.array([[[9,8,7,6],[1,2,3,4],[11,12,13,14]]])
print(var[0, 2, 1:3])  # Block 0, row 2, columns 1 and 2
```

Explanation:

Visual breakdown:

```
Block 0:
[
 [9, 8, 7, 6],      # Row 0
 [1, 2, 3, 4],      # Row 1
 [11, 12, 13, 14]   # Row 2
]
```

- var[0, 2, 1:3] → Row 2 → Slice columns 1 & 2 → **12 and 13**

Type	Example	Meaning
1D	arr[1:4]	Index 1 to 3
1D	arr[::2]	Every second element
2D	arr[row, col1:col2]	From one row, multiple columns
3D	arr[block, row, start:end]	Slice columns from 3D data

NumPy Array Functions – where() and sort()

np.where() – Find Index Locations Based on a Condition

np.where() helps you **find positions (indexes)** of elements that match a condition in the array.

Example 1: Where values are equal to 2

```
import numpy as np

var = np.array([1,2,4,6,3,2,78,90,56,45,34,2,3,4,5,8])

x = np.where(var == 2)

print(x)
```

Explanation:

- It found **value 2** at **positions (indexes)**: 1, 5, 11.

Example 2: Where values are even

```
x = np.where(var % 2 == 0)

print(x)
```

Explanation:

- %2 == 0 means even numbers.
- It lists the indexes of all even values.

np.sort() – Sort Array Values

np.sort() arranges elements in **ascending order**.

Example 3: Sort a 1D Array (Numbers)

```
x = np.sort(var)

print(x)
```

Example 4: Sort a 1D Array (Strings)

```
var = np.array(["a", "z", "d", "e"])

x = np.sort(var)

print(x)
```

Explanation:

- Alphabetical sorting happens like a dictionary.

Example 5: Sort a 2D Array

```
var = np.array([[1,2,4,6,3,2],[78,90,56,45,34,2],[3,4,5,8,98,56]])

x = np.sort(var)

print(x)
```

Explanation:

- Sorts **each row independently** in ascending order.

Example 6: Sort a 3D Array

```
var = np.array([[[1,2,4,6,3,2],[78,90,56,45,34,2],[3,4,5,8,98,56]]])

print(var.ndim)  # Output: 3

x = np.sort(var)

print(x)
```

Explanation:

- Even 3D arrays can be sorted, and **each sub-array (row) is sorted independently**.

Function	Purpose	Returns
np.where()	Finds index locations matching condition	Tuple of indexes
np.sort()	Sorts elements (1D, 2D, 3D)	New sorted array (original unchanged)

NumPy Insert and Delete Functions

These functions help you **modify arrays** by **adding** or **removing** elements using **index positions**.

np.insert() – Insert Values into an Array

1D Array Insert

```
import numpy as np

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

x = np.insert(var, 3, 20)

print(x)
```

Explanation:

- 3 is the index where 20 will be inserted.
- Original: [1, 2, 3, 4, 5, 6]

- After: [1, 2, 3, **20**, 4, 5, 6]

2D Array Insert – Insert Column

```
import numpy as np

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

x = np.insert(var, 1, 90, axis=1)

print(x)
```

Explanation:

- axis=1 → means we are inserting **column-wise**.
- At **index 1**, 90 is added to every row.

2D Array Insert – Insert Multiple Values as Column

```
import numpy as np

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

x = np.insert(var, 1, [9,11], axis=1)

print(x)
```

Explanation:

- Inserting [9, 11] at column index 1.
- 9 goes to first row, 11 goes to second row.

np.delete() – Delete Values from an Array

1D Array Delete

```
import numpy as np

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

x = np.delete(var, 3)

print(x)
```

Explanation:

- Removes element at index 3 → which is 4.
- Result: [1 2 3 5 6]

Function	Usage	Example

np.insert()	Add value at specific position	np.insert(arr, 2, 100)
np.insert(..., axis=1)	Insert column in 2D array	np.insert(arr, 1, val, axis=1)
np.delete()	Delete by index	np.delete(arr, 3)

Basic Statistical Operations in NumPy

These operations help us **understand data better**. Let's go step-by-step.

1. Mean (Average)

Formula:

Mean = (Sum of all values) / (Number of values)

```
import numpy as np

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

print(np.mean(a))
```

Explanation:

(1 + 2 + 3 + 4 + 5) / 5 = 15 / 5 = **3.0**

2. Median (Middle Value)

Rule:

- Sort the numbers.
- If odd count → pick the middle.
- If even count → average of the two middle numbers.

```
import numpy as np

a = [1, 2, 3, 4, 5, 6]

print(np.median(a))
```

Explanation:

Middle two values: 3 & 4
(3 + 4) / 2 = **3.5**

3. Mode (Most Frequent Value)

NumPy doesn't provide mode directly. Use scipy.stats.mode instead.

```
from scipy import stats
```

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

print(stats.mode(a, keepdims=True))
```

Explanation:

1 appears twice → **most frequent value**

4. Standard Deviation (Spread of Data)

Formula:

1. Find **mean**.
2. Subtract mean from each value and **square** the result.
3. Take **mean** of squared values.
4. Take **square root** of the result.

```
import numpy as np

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

print(np.std(a))
```

Step-by-Step Breakdown:

- Mean = 3
- Subtract & square:
  $(1-3)^2 = 4$, $(2-3)^2 = 1$, $(3-3)^2 = 0$, $(4-3)^2 = 1$, $(5-3)^2 = 4$
- Average of squared: (4+1+0+1+4) / 5 = 2
- Square root: √2 ≈ **1.414**

Function	**Meaning**	**NumPy Function**
Mean	Average value	np.mean(array)
Median	Middle value	np.median(array)
Mode	Most repeated value	stats.mode(array)
Std. Deviation	Spread from mean	np.std(array)

Coefficient of Variation (CV)

The **Coefficient of Variation** tells us **how much variability** (spread) there is **in relation to the mean** of the data.

Formula:

CV = Standard Deviation / Mean

It's **useful to compare the spread of data across datasets**, especially when their **means are different**.

```
import numpy as np

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

print(np.std(a) / np.mean(a))
```

PRACTICE QUESTIONS

Short Answer

1. What is NumPy used for in Python?
2. How do you create a NumPy array from a Python list?
3. What is the difference between np.array() and np.arange()?
4. What does the function np.zeros((2,3)) return?
5. What is the shape of the array np.array([[1,2,3],[4,5,6]])?

Coding

6. Create a NumPy array with values from 1 to 10.
7. Create a 3x3 array filled with ones.
8. Create an array of 5 random integers between 1 and 50.
9. Reverse a 1D array using slicing.
10. Create a 4x4 identity matrix.

Intermediate Level Questions

11. Reshape a 1D array of 12 elements into a 3D array of shape (2, 2, 3).
12. Flatten a 2D array to 1D.
13. Given an array, replace all values greater than 10 with 10.
14. Find the maximum, minimum, mean, and standard deviation of an array.
15. Sort a 2D array row-wise.

Indexing, Slicing, and Operations

16. Extract elements from index 2 to 5 in a 1D array.
17. From a 2D array, extract the second row.
18. Add 5 to every element of an array using broadcasting.

19. Perform element-wise multiplication of two arrays.
20. Write code to count how many values in an array are divisible by 3.

Statistical Functions

21. Write a program to calculate:

- Mean
- Median
- Mode
- Standard Deviation
- Coefficient of Variation

22. From a list of students' marks, find who scored above the average.

Random, Insert, Delete, Where

23. Generate a 1D array of 5 random float numbers between 0 and 1.
24. Insert a value at index 3 in an array.
25. Delete the 2nd row from a 2D array.
26. Use np.where() to find indices of even numbers in an array.