NUMPY ARRAYS

BASIC OPERATIONS IN NUMPY

What is NumPy?

NumPy is a **Python** library used for **numerical** and **matrix computations**.

Main Features

I. Fast Array Processing

NumPy arrays are **more efficient** than **Python lists** for large data.

II. Vectorized Operations

Perform **element-wise** operations **without** writing **loops**.

III. Multidimensional Arrays

Support for **arrays** of **any dimension**.

IV. Integration

Works with other **libraries** like **Pandas**, **Matplotlib**, and **Scikit-learn**.

```
import numpy as np
```

Creating Arrays

There are **several ways** to create arrays in **NumPy**:

1. `np.array()`

Purpose:

Converts **input data** (lists, tuples) into a **NumPy array**.

```
import numpy as np

ages = [16, 22, 39, 86]

array = np.array(ages)

[16 22 39 86]
```

Creating Arrays

2. `np.zeros()`

Purpose:

Creates an **array** filled with **zeros**.

Parameters:

`**shape**`: A **tuple** defining the **shape** of the **array**.

```
import numpy as np

zeros_array = np.zeros((3,4))

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Creating Arrays

3. np.ones()

Purpose:

Creates an **array** filled with **ones**.

Parameters:

`**shape**`: A **tuple** defining the **shape** of the **array**.

```
import numpy as np

ones_array = np.ones((3,4))

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Creating Arrays

4. np.arange()

Purpose:

Generates an **array** with **values** from **start** to **stop** (exclusive) with a **step**.

Parameters:

`**start**` : Starting value.

`**stop**` : Stopping value (excluded).

`**step**` : Step size (default is 1).

```
import numpy as np

arange_array = np.arange(5, 25, 5)

[5 10 15 20]
```

Creating Arrays

5. np.linspace()

Purpose:

Creates an **array** of **evenly spaced values** between **start** and **stop**.

Parameters:

`**start**` : Starting value.

`**stop**` : Stopping value (included).

`**num**` : Number of values to generate.

```
import numpy as np

linspace_array = np.linspace(5, 25, 10)

[ 5. 7.22222222 9.44444444 11.66666667 13.88888889
16.11111111 18.33333333 20.55555556 22.77777778 25. ]
```

Creating Arrays

Summary

`np.array()`	Converts lists/tuples to arrays.
`np.zeros()`	Creates zero-filled arrays.
`np.ones()`	Creates one-filled arrays.
`np.arange()`	Creates arrays with a range of values.
`np.linspace()`	Creates arrays with evenly spaced values.

Array Attributes

1. shape

Purpose:

Returns the **dimensions** (size along each axis) of the **array**.

```
import numpy as np

array = np.array([[9, 8, 7, -3], [-1, 2, 0, 5]])

print(array.shape)
(2, 4)
```

Array Attributes

2. dtype

Purpose:

Returns the **data type** of the array **elements**.

```
array = np.array([[9, 8, 7], [-1, 2, 0]])

print(array.dtype)
int32
```

```
array = np.linspace(-8, 8, 16)

[-8. -6.93333333 -5.86666667 -4.8 -3.73333333 -
2.66666667 -1.6 -0.53333333 0.53333333 1.6
2.66666667 3.73333333 4.8 5.86666667 6.93333333 8. ]

print(array.dtype)
float64
```

Array Attributes

3. `size`

Purpose:

Returns the **total** number of **elements** in the array.

```
array = np.array([[9, 8, 7], [-1, 2, 0]])

print(array.size)
6
```

```
array = np.arange(1,9,3)
[1 4 7]

print(array.size)
3
```

Array Attributes

4. ndim

Purpose:

Returns the **number** of **dimensions** (**axes**) of the array.

```
array = np.array([[9, 8, 7], [-1, 2, 0]])

print(array.ndim)
2
```

Array Attributes

Summary

`shape`	Dimensions of the array.
`dtype`	Data type of array elements.
`size`	Total number of elements.
`ndim`	Number of dimensions.

Basic Operations

Element-wise

1. Addition (+)

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

c = a + b

print(c)
[5 7 9]
```

```
d = np.ones((2,3))
e = np.array([[1,-1,1], [0,-2,-1]])

f = d + e

print(f)
[[ 2. 0. 2.]
 [ 1. -1. 0.]]
```

Basic Operations

Element-wise

2. Subtraction (-)

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

c = a - b

print(c)
[-3 -3 -3]
```

```
d = np.ones((2,3))
e = np.array([[1,-1,1], [0,-2,-1]])

f = d - e

print(f)
[[0. 2. 0.]
 [1. 3. 2.]]
```

Basic Operations

Element-wise

3. Multiplication (*)

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

c = a * b

print(c)
[ 4 10 18]
```

```
d = np.ones((2,3))
e = np.array([[1,-1,1], [0,-2,-1]])

f = d * e

print(f)
[[ 1. -1. 1.]
 [ 0. -2. -1.]]
```

Basic Operations

Element-wise

4. Division (/)

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

c = a / b

print(c)
[0.25 0.4 0.5 ]
```

```
d = np.ones((2,3))
e = np.array([[1,-1,1], [0,-2,-1]])

f = d / e

print(f)
[[ 1. -1. 1. ]
 [ inf -0.5 -1. ]]
```

Basic Operations

Element-wise

5. Exponentiation (**)

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

c = a ** b

print(c)
[ 1 32 729]
```

```
d = np.ones((2,3))
e = np.array([[1,-1,1], [0,-2,-1]])

f = d ** e

print(f)
[[1. 1. 1.]
 [1. 1. 1.]]
```

Basic Operations

Element-wise

Summary

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Basic Operations

Unary

1. Sum: np.sum()

```
v = np.array([-2, 3, 9])

np.sum(v)
10
```

```
x = np.array([[-2, 3, 9], np.ones(3)])

[[-2. 3. 9.]
 [ 1. 1. 1.]]

np.sum(x)
13.0
```

Basic Operations

Unary

2. Minimum: np.min()

```
v = np.array([-2, 3, 9])

np.min(v)
-2
```

```
x = np.array([[-2, 3, 9], np.ones(3)])

[[-2. 3. 9.]
 [ 1. 1. 1.]]

np.min(x)
-2.0
```

Basic Operations

Unary

3. Maximum: np.max()

```
v = np.array([-2, 3, 9])

np.max(v)
9
```

```
x = np.array([[-2, 3, 9], np.ones(3)])

[[-2. 3. 9.]
 [ 1. 1. 1.]]

np.max(x)
-9.0
```

Basic Operations

Unary

4. Mean: np.mean()

```
v = np.array([-2, 3, 9])

np.mean(v)
3.3333333333333335
```

```
x = np.array([[-2, 3, 9], np.ones(3)])

[[-2. 3. 9.]
 [ 1. 1. 1.]]

np.mean(x)
2.1666666666666665
```

Basic Operations

Unary

5. Standard Deviation: np.std()

```
v = np.array([-2, 3, 9])

np.std(v)
4.496912521077347 5
```

```
x = np.array([[-2, 3, 9], np.ones(3)])

[[-2. 3. 9.]
 [ 1. 1. 1.]]

np.std(x)
3.387066905483596
```

Basic Operations

Unary

Summary

Sum	np.sum()
Minimum	np.min()
Maximum	np.max()
Mean	np.mean()
Standard Deviation	np.std()

Advanced Manipulations

1. Reshaping Arrays: reshape()

Purpose:

Change the **shape** of an array **without** changing its data.

```
import random

array = np.array([random.randint(1,10) for _ in range(3)])
[2 1 4]

reshaped = array.reshape((3,1))
[[2]
 [1]
 [4]]
```

Advanced Manipulations

2. Transposing Arrays: T

Purpose:

Flip the **dimensions** of an array.

```
import random

array = np.array([[random.randint(1,10) for _ in range(3)],
                  [random.randint(11,20) for _ in range(3)]])
[[ 5 3 8]
 [13 20 12]]

transposed = array.T
[[ 5 13]
 [ 3 20]
 [ 8 12]]
```

Advanced Manipulations

3. Concatenation: np.concatenate()

Purpose:

Combine multiple arrays into **one**.

```
a = np.arange(1,12,4)
[1 5 9]

b = np.linspace(5,9,6)
[5. 5.8 6.6 7.4 8.2 9. ]

combined = np.concatenate((a,b))
[1. 5. 9. 5. 5.8 6.6 7.4 8.2 9. ]
```

Advanced Manipulations

4. Splitting: np.split()

Purpose:

Split an **array** into **multiple sub-arrays**.

```
array = np.arange(8.0)
[0. 1. 2. 3. 4. 5. 6. 7.]

splitted = np.split(array, 2)
[array([0., 1., 2., 3.]), array([4., 5., 6., 7.])]
```

Advanced Manipulations

Summary

`reshape()`	Reshapes an array
`T`	Transposes an array
`np.concatenate()`	Merges arrays
`np.split()`	Divides an array