# creating numpy arrays

In [1]:

```
#1D array
import numpy as np
arr1=np.array([1,2,3])
arr1
```

Out[1]:

```
array([1, 2, 3])
```

In [4]:

```
#2D array
arr2=np.array([[1,2,3],[4,5,6]])
arr2
```

Out[4]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [8]:

```
#dtype
arr3=np.array([1,2,3],dtype=float)
arr3
#arr4=np.array([2,3,4,5],dtype=bool)
#arr4
```

Out[8]:

```
array([1., 2., 3.])
```

# Using numpy.zeros and numpy.ones

In [9]:

```
#1D array
zeros_array=np.zeros(4)
zeros_array
```

Out[9]:

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

In [10]:

```
#2D array
ones_array=np.ones((5,3))
ones_array
```

Out[10]:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

# using numpy.arange

In [11]:

```
#from 0-9
arr=np.arange(10)
arr
```

Out[11]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [12]: #from 2-10 with step 2
         arr=np.arange(2,11,2)
         arr
```

```
Out[12]: array([ 2,  4,  6,  8, 10])
```

```
In [13]: #reshape
         arr=np.arange(16).reshape(2,2,2,2)
         arr
```

```
Out[13]: array([[[[ 0,  1],
                  [ 2,  3]],

                 [[ 4,  5],
                  [ 6,  7]]],


                [[[ 8,  9],
                  [10, 11]],

                 [[12, 13],
                  [14, 15]]]])
```

# using numpy.linspace

```
In [15]: arr=np.linspace(1,6,5)
         arr
```

```
Out[15]: array([1.  , 2.25, 3.5 , 4.75, 6.  ])
```

```
In [16]: arr=np.linspace(1,23,23)
         arr
```

```
Out[16]: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
                14., 15., 16., 17., 18., 19., 20., 21., 22., 23.])
```

```
In [17]: arr=np.linspace(0,5,5,dtype=int)
         arr
```

```
Out[17]: array([0, 1, 2, 3, 5])
```

# using numpy.random

```
In [18]: rand_arr=np.random.rand(5)
         rand_arr
```

```
Out[18]: array([0.32517218, 0.30128424, 0.54270378, 0.93610065, 0.59094601])
```

```
In [19]: arr=np.random.randint(1,11,(3,4))
         arr
```

```
Out[19]: array([[ 2,  7,  7,  9],
                [10,  9,  6,  7],
                [ 3, 10,  1, 10]], dtype=int32)
```

```
In [21]: arr=np.random.randint(1,13,(2,4))
         arr
```

```
Out[21]: array([[ 1,  3,  2,  1],
                [10, 11,  8, 12]], dtype=int32)
```

# using np.identity

```
In [22]: arr=np.identity(3)
         arr
```

```
Out[22]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

```
In [23]: arr=np.identity(4)
         arr
```

```
Out[23]: array([[1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.]])
```

# using np.eye()

```
In [24]: arr=np.eye(3)
         arr
```

```
Out[24]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

# ndim

```
In [25]: arr = np.array([[1, 2, 3], [4, 5, 6]])
         print(arr.ndim)
```

```
2
```

```
In [26]: arr=np.array([1,2,3,4,5])
         arr.ndim
```

```
Out[26]: 1
```

# shape of array

In [27]:

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

```
(2, 3)
```

In [30]:

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

Out[30]:

```
(2, 5)
```

# size

In [31]:

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

Out[31]:

```
10
```

In [32]:

```
arr=np.array([1,2,3,4,5])
arr.size
```

Out[32]:

```
5
```

# dtype

In [33]:

```
arr=np.array([1,2,3,4,5])
arr.dtype
```

Out[33]:

```
dtype('int64')
```

In [34]:

```
arr=np.array([1,2,3,4,5],dtype=float)
arr.dtype
```

Out[34]:

```
dtype('float64')
```

# itemsize

In [35]:

```
arr=np.array([1,2,3,4,5])
arr.itemsize
```

Out[35]:

```
8
```

In [36]:

```
arr=np.array([1,2,3,4,5],dtype=np.int32)
arr.itemsize
```

Out[36]:

```
4
```

# changing datatype of array

In [37]:

```
arr=np.array([1,2,3,4,5])
arr.astype(np.int32)
```

Out[37]:

```
array([1, 2, 3, 4, 5], dtype=int32)
```

In [38]:

```
arr=np.array([1,2,3,4,5])
arr.astype(np.float32)
```

Out[38]:

```
array([1., 2., 3., 4., 5.], dtype=float32)
```

# element wise operations(add,subtract,multiply,divide)

In [40]:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
add=arr1+arr2
add
```

Out[40]:

```
array([5, 7, 9])
```

In [41]:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
sub=arr1-arr2
sub
```

Out[41]:

```
array([-3, -3, -3])
```

# universal functions(sqrt,exp,trig)

In [42]:

```
np.sqrt(arr1)
```

Out[42]:

```
array([1.        , 1.41421356, 1.73205081])
```

In [43]:

```
np.exp(arr1)
```

Out[43]:

```
array([ 2.71828183,  7.3890561 , 20.08553692])
```

# Matrix operation (dot product)

In [46]:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr3=np.dot(arr1,arr2)
print(arr3)
```

```
32
```

# Sum/mean/min/max

In [49]:

```
arr1 = np.array([1, 2, 3])
arr=np.sum(arr1)
print(arr)
```

```
6
```

In [50]:

```
arr1 = np.array([1, 2, 3])
arr=np.mean(arr1)
print(arr)
```

```
2.0
```

In [51]:

```
arr1 = np.array([1, 2, 3])
arr=np.min(arr1)
print(arr)
```

```
1
```

In [52]:

```
arr1 = np.array([1, 2, 3])
arr=np.max(arr1)
print(arr)
```

```
3
```

# Statistical functions

In [54]:

```
arr = np.array([1, 2, 3, 4, 5])
arr1=np.std(arr)
print(arr1)
```

```
1.4142135623730951
```

In [56]:

```
arr2=np.percentile(arr,50)
print(arr2)
```

```
3.0
```

# logical operations

In [57]:

```
arr = np.array([1, 2, 3, 4, 5])
res=arr>3
print(res)
```

```
[False False False  True  True]
```

In [58]:

```
res=np.logical_and(arr > 1, arr < 4)
print(res)
```

```
[False  True  True False False]
```

In [59]:

```
res=np.logical_or(arr > 1, arr < 4)
print(res)
```

```
[ True  True  True  True  True]
```

# numpy.unique

In [60]:

```
arr = np.array([1,2,3,4,5,2,3,4])
arr1=np.unique(arr)
print(arr1)
```

```
[1 2 3 4 5]
```

# numpy.round

In [61]:

```
arr = np.array([1.2, 2.5, 3.7, 4.0, 5.9])
rounded_arr = np.round(arr)
print(rounded_arr)
```

```
[1. 2. 4. 4. 6.]
```

# numpy.floor

In [62]:

```
floor_arr = np.floor(arr)
print(floor_arr)
```

```
[1. 2. 3. 4. 5.]
```

# numpy.ceil

In [63]:

```
ceil_arr = np.ceil(arr)
print(ceil_arr)
```

```
[2. 3. 4. 4. 6.]
```

# concatenate

In [65]:

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6]])
arr = np.concatenate((arr1, arr2))
print(arr)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

# transpose

In [67]:

```
arr = np.array([[1, 2], [3, 4]])
arr1 = np.transpose(arr)
print(arr1)
```

```
[[1 3]
 [2 4]]
```

# Indexing

In [68]:

```
arr = np.array([1, 2, 3, 4, 5])
arr2=arr[3]
print(arr2)
```

```
4
```

In [69]:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
arr1=arr[1,2]
print(arr1)
```

```
6
```

# Slicing

In [72]:

```
arr = np.array([1, 2, 3, 4, 5])
arr1=arr[1:3]
print(arr1)
```

```
[2 3]
```

In [73]:

```
arr = np.array([[1, 2, 3], [4, 5, 6],[7, 8, 9]])
arr1=arr[1:3,0:2]
print(arr1)
```

```
[[4 5]
 [7 8]]
```

In [74]:

```
#conditional indexing
arr = np.array([1, 2, 3, 4, 5])
ele=arr[arr>3]
print(ele)
```

```
[4 5]
```

In [75]:

```
res=np.array([1,3])
print(res)
```

```
[1 3]
```

# Flatten

In [77]:

```
arr = np.array([[1, 2],[6,7]])
arr1=arr.flatten()
print(arr1)
```

```
[1 2 6 7]
```

### Stacking arrays-Horizontal stack(side by side)

In [4]:

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

Out[4]:

```
array([1, 2, 3, 4, 5, 6])
```

# Vertical Stack (top-to-bottom)

In [5]:

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

Out[5]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

## Column Stack (combine 1D arrays as columns)

In [7]:

```
np.column_stack((a,b))
```

Out[7]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

## Splitting Arrays

Split arrays into equal parts.

```
In [2]: arr = np.arange(10)
        print(np.split(arr, 2)) # split into 2 equal parts
```

```
[array([0, 1, 2, 3, 4]), array([5, 6, 7, 8, 9])]
```

# Inserting, Appending, Deleting

## Append

```
In [3]: arr = np.array([1,2,3])
        np.append(arr, [4,5])
```

```
Out[3]: array([1, 2, 3, 4, 5])
```

## Insert

```
In [4]: np.insert(arr, 1, 100)
```

```
Out[4]: array([  1, 100,   2,   3])
```

## Delete

```
In [5]: np.delete(arr, 1)
```

```
Out[5]: array([1, 3])
```

# Copy as view

copy- Any changes made to copy array willn't affect original array

view- Any changes made to view will affect original array

```
In [6]: a = np.arange(6)
        b = a.view() # view (linked)
        c = a.copy() # independent copy
        b[0] = 99
        print(a) # [99 1 2 3 4 5]
        print(c) # [0 1 2 3 4 5]
```

```
[99  1  2  3  4  5]
[0 1 2 3 4 5]
```

# What is Broadcasting?

Broadcasting allows NumPy to perform operations between arrays of different shapes automatically, without explicit looping.

In [7]:

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

```
[5 7 9]
```

## ⚠️ Broadcasting Rules

To make two arrays compatible

Dimensions are compared from right to left.

Dimensions are compatible if:

they are equal, or one of them is 1.

If not compatible → raises ValueError.

# What is Vectorization?

Vectorization means performing operations on entire arrays instead of using loops. It’s the key to NumPy’s speed.

In [9]:

```
nums = [1,2,3,4,5]
result = []
for n in nums:
    result.append(n * 2)
print(result)
```

```
[2, 4, 6, 8, 10]
```

In [10]:

```
a = np.array([1,2,3])
b = np.array([4,5,6])
print(a + b) # [5 7 9]
print(a * b) # [ 4 10 18]
print(a ** 2) # [1 4 9]
print(np.sin(a)) # apply sin() to all elements
```

```
[5 7 9]
[ 4 10 18]
[1 4 9]
[0.84147098 0.90929743 0.14112001]
```

# Combining Broadcasting +Vectorization

You can mix both concepts for complex operations

In [11]:
```
a = np.array([[1,2,3],
[4,5,6],
[7,8,9]])
b = np.array([1,2,3])
# Broadcasting b across all rows
print(a * b)
```

```
[[ 1  4  9]
 [ 4 10 18]
 [ 7 16 27]]
```

# Handling Missing Values (NaN, None,Infs)

Missing or invalid data is very common in datasets — before any analysis or ML, you must handle them properly.

## 1. What are Missing Values?

type-------meaning---------------------------example

np.nan---- Not a Number (missing or undefined)---- np.nan

None -----Python’s null object ----None

np.inf / -np.inf ---------Infinity / negative infinity----- np.inf

In [12]:
```
# Creating Arrays with Missing Values
import numpy as np
arr = np.array([1, 2, np.nan, 4, 5])
print(arr)
```

```
[ 1.  2. nan  4.  5.]
```

## Detecting Missing Values

```
Function       Description                    Example           Output
np.isnan(x)    True where elementsare NaN     np.isnan(arr)       [False
False True False False]
np.isfinite(x) True where finite              np.isfinite(arr)      [True
True False True True]
np.isinf(x)     True for infinite values      np.isinf(arr)         [False
False False False False]
```

## Removing Missing Values

You can filter out missing entries using boolean indexing:

In [14]:

```
arr = np.array([1, 2, np.nan, 4, 5])
clean = arr[~np.isnan(arr)]
print(clean)
```

```
[1. 2. 4. 5.]
```

### Replacing Missing Values

In [15]:

```
arr = np.array([1, np.nan, 3, np.nan, 5])
arr[np.isnan(arr)] = 0
print(arr)
```

```
[1. 0. 3. 0. 5.]
```

In [16]:

```
arr = np.array([1, 2, np.nan, 4, 5])
mean_val = np.nanmean(arr)
arr[np.isnan(arr)] = mean_val
print(arr)
```

```
[1. 2. 3. 4. 5.]
```

```
Function                  Description
np.nanmean(arr)           Mean ignoring NaNs
np.nanstd(arr)            Standard deviation ignoring NaNs
np.nanmin(arr)             Minimum ignoring NaNs
np.nanmax(arr)             Maximum ignoring NaNs
np.nansum(arr)             Sum ignoring NaNs
```

In [17]:

```
import numpy as np
data = np.array([10, np.nan, 20, np.inf, 30, np.nan])
print(np.isnan(data))
# Step 1: Replace inf with nan
data[np.isinf(data)] = np.nan
# Step 2: Replace missing with mean
data[np.isnan(data)] = np.nanmean(data)
print(data)
```

```
[False  True False False False  True]
[10. 20. 20. 20. 30. 20.]
```

In [ ]: