1.Array Creation Functions

In [1]:
```
import numpy as np
```

Create an array from list

In [2]:
```
a=np.array([1,2,3,4])
print('Array a:',a)
```

```
Array a: [1 2 3 4]
```

Create an array with evenly spaced values

arange()

In [3]:
```
b=np.arange(0,20,2)  # values from 0 to 20 with step 2
print('Array b:',b)
```

```
Array b: [ 0  2  4  6  8 10 12 14 16 18]
```

create an array filled with zeros

zeros()

In [5]:
```
c=np.zeros(3)
print(c)
```

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

In [8]:
```
d=np.zeros((2,3),dtype=int) # 2x3 array of zeros
print(d)
```

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

create an array filled with ones

ones()

In [9]:
```
np.ones(6)
```

Out[9]:
```
array([1., 1., 1., 1., 1., 1.])
```

In [10]:
```
np.ones(6,dtype=int)
```

Out[10]:
```
array([1, 1, 1, 1, 1, 1])
```

In [11]:
```
np.ones((3,4),dtype=int) # 3x4 matrix of ones
```

Out[11]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

In [13]:
```
np.ones((5,5))
```

Out[13]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

Create an identity matrix

identity()--The identity array is a square array with ones on the main diagonal.

In [16]:
```
f=np.identity(4)
print('Identity matric f:\n',f)
```

```
Identity matric f:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [17]:
```
np.identity(5, dtype=int)
```

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

2.Array Manipulation Functions

Reshape an array

reshape()

In [19]:
```
a1=np.array([1,2,3,4,5,6])
reshaped=np.reshape(a1,(2,3))
print('Reshaped Array:\n',reshaped)
```

```
Reshaped Array:
 [[1 2 3]
 [4 5 6]]
```

Flatten an array

ravel()

In [20]:
```
f1=np.array([[1,2],[3,4]])
flattened=np.ravel(f1) # flatten to 1D Array
print('Flattened array:\n',flattened)
```

```
Flattened array:
 [1 2 3 4]
```

Transpose an array

transpose()

In [22]:
```
e1=np.array([[1,2],[3,4],[5,6]])
transposed=np.transpose(e1)
print('Transposed array:\n',transposed)
```

```
Transposed array:
 [[1 3 5]
 [2 4 6]]
```

Stack arrays vertically and horizantally

vstack() & hstack()

In [24]:
```
a2 = np.array([1, 2])
b2 = np.array([3, 4])
v_stacked=np.vstack([a2,b2])
print('Stacked arrays vertically\n',v_stacked)
h_stacked=np.hstack([a2,b2])
print('Stacked arrays horizantally\n',h_stacked)
```

```
Stacked arrays vertically
 [[1 2]
 [3 4]]
Stacked arrays horizantally
 [1 2 3 4]
```

3. Mathematical Functions

add()

In [26]:
```
# Add 2 to each elemt in array
g = np.array([1,2,3,4])
added=np.add(g,2)
print('added 2 to array g:', added)
```

```
added 2 to array g: [3 4 5 6]
```

power()

In [27]:
```
# square each element
squared=np.power(g,2)
print('squared array g:',squared)
```

```
squared array g: [ 1  4  9 16]
```

sqrt()

In [28]:
```
sqrt_val=np.sqrt(g) # square root of each element
print('square root of g:\n',sqrt_val)
```

```
square root of g:
 [1.         1.41421356 1.73205081 2.        ]
```

dot() --> product of 2 arrays

In [30]:
```
print(a1)
print(g)
```

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

In [31]:
```
np.dot(a1,g)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[31], line 1
----> 1 np.dot(a1,g)

ValueError: shapes (6,) and (4,) not aligned: 6 (dim 0) != 4 (dim 0)
```

In [32]:
```
g1=np.array([1,2,3,4])
procted_array=np.dot(g,g1)
print('Dot product of g and g1',procted_array)
```

```
Dot product of g and g1 30
```

4. Statistical Functions

mean() ---> average of an array elements

In [33]:
```
s=np.array([1,2,3,4,5,6])
print('mean of an array is:',np.mean(s))
```

```
mean of an array is: 3.5
```

std() - standard Deviation of an array

In [35]:
```
print('standard deviation of an array is:',np.std(s))
```

```
standard deviation of an array is: 1.707825127659933
```

min()

In [36]:
```
print('minimum number in an array is:',np.min(s))
```

```
minimum number in an array is: 1
```

max()

In [37]:
```
print('maximum element of an array is:',np.max(s))
```

```
maximum element of an array is: 6
```

5.Linear Algebra Functions

In [39]:
```
# create a matrix
matrix=np.array([[1,2],[3,4]])
matrix
```

Out[39]:
```
array([[1, 2],
       [3, 4]])
```

6. Random Sampling Functions

rand() --> Generate random values between 0 and 1

In [40]:
```
print('random values:',np.random.rand(3))
```

```
random values: [0.23485653 0.07984467 0.71277074]
```

seed() ---> sets the seed for NumPy's random generator, so that you get same random resultsevery time you run the code

In [72]:
```
np.random.seed(1)

# Generate random values between 0 and 1
random_vals = np.random.rand(3)  # Array of 3 random values between 0 and 1
print("Random values:", random_vals)
```

```
Random values: [4.17022005e-01 7.20324493e-01 1.14374817e-04]
```

In [73]:
```
np.random.seed(1)
np.random.rand(3)
```

Out[73]:
```
array([4.17022005e-01, 7.20324493e-01, 1.14374817e-04])
```

randint() ----> generate random integers

In [70]:
```
np.random.seed(23)
np.random.randint(0,10,2)
```

Out[70]:
```
array([3, 6])
```

In [71]:
```
np.random.seed(23)
np.random.randint(0,10,2)
```

Out[71]:
```
array([3, 6])
```

In [75]:
```
np.random.randint(0,10,2) ## generate random numbers
```

Out[75]:
```
array([0, 1])
```

7.Boolean and Logical Functions

all() ---> returns true if all elementsare true

logical_test = np.array([True, False, True]) np.all(logical_test)

In [78]:
```
logical_test1 = np.array([True, 2, True])
np.all(logical_test1)
```

Out[78]:
```
True
```

any() ---> returns true if any one elememt is True

In [81]:
```
logical_test2 = np.array([True, False, 3])
np.any(logical_test2)
```

Out[81]:
```
True
```

In [82]:
```
logical_test3 = np.array([False, False, 0])
np.any(logical_test3)
```

Out[82]:
```
False
```

8. Set operations

In [83]:
```
# Intersection of two arrays
a = np.array([1, 2, 3, 4])
b = np.array([3, 4, 5, 6])
intersection = np.intersect1d(a, b)
print("Intersection of a and b:", intersection)
```

```
Intersection of a and b: [3 4]
```

In [84]:
```
# Union of two arrays
union=np.union1d(a,b)
print('Union of a and b:',union)
```

```
Union of a and b: [1 2 3 4 5 6]
```

9. Array Attribute Functions

In [85]:
```
# Array Attributes
a=np.array([1,2,3])
```

a.shape ---> shape of the array

In [87]:
```
a.shape
```

Out[87]: (3,)

a.size ---> Number of elements

In [88]:
```
a.size
```

Out[88]: 3

a.ndim---> number of dimentions

In [89]:
```
a.ndim
```

Out[89]: 1

a.dtype ----> data type of the array

In [90]:
```
a.dtype
```

Out[90]: dtype('int32')

10. Other Functions

In [91]:
```
a=np.array([1,2,3,4,5])
```

copy() ---->create a copy of an array

In [92]:
```
copied_array=np.copy(a)
print('copied array is:',copied_array)
```

```
copied array is: [1 2 3 4 5]
```

nbytes ----> size of an array in bytes

In [93]:
```
a.nbytes
```

Out[93]:
```
20
```

shared_memory(arr1,arr2) ---> check if two arrays shares same memory

In [94]:
```
np.shares_memory(a, copied_array)
```

Out[94]:
```
False
```