Numpy

In [2]:

```
!pip install Numpy
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: Numpy in c:\users\vaishnavi\lib\site-packages (1.24.3)
```

What is an array?

An array is a collection of elements of same data type stored in contiguous memory locations. And these memory locations are pointed or indexed as 0 to n. NumPy array is a powerful N-dimensional array object and its use in linear algebra, Fourier transform, and random number capabilities. It provides an array object much faster than traditional Python lists.

In [2]:

```
import numpy as np
```

In [3]:

```
#creating an array
a = np.array([1,2,3,4])
print (a)
```

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

In [7]:

```
# If there is any float element in the tuple, we have to mention dtype = float. If not
b = np.array([[1,2,3],[2,3,4.1]], dtype = float)
print(b)
```

```
[[1.  2.  3. ]
 [2.  3.  4.1]]
```

In [8]:

```
b = np.array([1,2,3],[2,3,4.1])
print(b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 b = np.array([1,2,3],[2,3,4.1])
      2 print(b)

TypeError: Field elements must be 2- or 3-tuples, got '2'
```

Initial place holders

These are used when we want to assign the values randomly, either all zeros or ones.

In [9]:

```
#array of zeros
c = np.zeros((2,3))
print(c)
```

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

In [11]:

```
d = np.ones((2,3,4))
print(d)
```

```
[[[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 array of evenly spaced values with step value

In [12]:

```
e = np.arange(5,19,3)
print(e)
```

```
[ 5  8 11 14 17]
```

It starts counting from 5 and end at n-1, i.e, 18. it gives the values with space of 3.

create an array of evenly spaced values when no of samples is given

In [14]:

```
f = np.linspace(0,15,8)
print(f)
```

```
[ 0.          2.14285714  4.28571429  6.42857143  8.57142857 10.71428571
 12.85714286 15.        ]
```

It starts at 0, ends at 7. creating totally 8 samples with even space. space = (end value - intial value)/(no.of samples - 1)

create a constant array

In [5]:

```
!pip install numpy
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\vaishnavi\lib\site-packages (1.24.3)
```

In [7]:

```
import numpy as np
```

In [11]:

```
# create 1-D array
g = np.full(3,11)
print(g)
print(g.ndim)
```

```
[11 11 11]
1
```

In [4]:

```
import numpy as np
```

In [5]:

```
g1 = np.full(11,3)
print(g1)
```

```
[3 3 3 3 3 3 3 3 3 3 3]
```

In [16]:

```
#create a 2-D array
h = np.full((2,2),11)
```

```
print(h)
print(h.ndim)
```

```
[[11 11]
 [11 11]]
2
```

In [14]:

```
#create a 3-D array
i = np.full((3,2,3),11)
print(i)
print(i.ndim)
```

```
[[[11 11 11]
  [11 11 11]]

 [[11 11 11]
  [11 11 11]]

 [[11 11 11]
  [11 11 11]]]
3
```

In [17]:

```
#create an identity matrix
j = np.eye(4)
print(j)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

create an array with random values

In [20]:

```
random_array = np.random.random((3,2))
print(random_array)
```

```
[[0.52662187 0.05854796]
 [0.45921952 0.14257465]
 [0.70582035 0.73237464]]
```

Find a shape of an array

In [25]:

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

```
(2, 3, 3)
```

In [26]:

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

print(L.shape)
```

```
(3, 3)
```

In [27]:

```
#To find no.of elements in an array

print(random_array.size)
```

```
6
```

In [29]:

```
#To find the length of an array i.e, no.of rows in an array

len(random_array)
```

Out[29]:

```
3
```

Access array elements

The indices in numpy starts from zero. That means the first element is marked as zero.

In [37]:

```
#index of 1-D array

M = np.array([12,4,34,68])
print(M[3])
print(M.ndim)
```

```
68
1
```

array here is [12,4,34,68] marked as [0,1,2,3].

In [31]:

```
#Adding of two index positioned elemnts

print(M[0]+M[2])
```

```
46
```

In [42]:

```
#index of 2-D array

N = np.array([[23,23,54,33],[56,23,87,34]])
print(N[1,3])
```

```
34
```

i.e, index 1 in rows and 3 column. row and column start from 0. so index 1 in row is [56,23,87,34] and 3 in column is 34.

In [44]:

```
#index of 3-D array

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

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

 [[4 5]
  [6 7]]]
```

In [45]:

```
print(O[1,0,1])
```

```
5
```

from shape index 1 -> row index 0 -> column index 1 = 5

In [47]:

```
print(O[0,0,1])
```

```
2
```

Data types in array

In [52]:

```
#integer

Arr = np.array([1,2,3,4])
print(Arr)
print(Arr.dtype)
```

```
[1 2 3 4]
int32
```

In [51]:

```
#string

Arr = np.array([1,2,3,4], dtype = 'S')
print(Arr)
print(Arr.dtype)
```

```
[b'1' b'2' b'3' b'4']
|S1
```

In [57]:

```
#change float to integer by using 'i' as parameter

A1 = np.array([1.2, 2.3, 3.4, 4.5])
A_1 = A1.astype('i')
print(A_1)
print(A1.dtype)
print(A_1.dtype)
```

```
[1 2 3 4]
float64
int32
```

In [8]:

```
# change data from int to boolean

B_arr = np.array([1,0,4,2,5])
B_arr2 = B_arr.astype(bool)
print(B_arr2)
print(B_arr2.dtype)
```

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

In [14]:

```
# copy method

P = np.array([1,5,3,7,93])
P1 = P.copy()
print(P1)
```

```
[ 1  5  3  7 93]
```

In [15]:

```
P1[1] = 67
print(P1)
```

```
[ 1 67  3  7 93]
```

Reshaping of array

In [17]:

```
# convert 1-D array to 2-D array

Q = np.array([2,4,5,2,67,34,7,8,23,5])
Q1 = Q.reshape(2,5)
print(Q1)
```

```
[[ 2  4  5  2 67]
 [34  7  8 23  5]]
```

In [19]:

```
# convert 1-D array to 3-D array

R = np.array([3,5,2,6,7,9,2,8,2,1,9,0,3,6,2,67])
R1 = R.reshape(2,4,2)
print(R1)
```

```
[[[ 3  5]
  [ 2  6]
  [ 7  9]
  [ 2  8]]

 [[ 2  1]
  [ 9  0]
  [ 3  6]
  [ 2 67]]]
```

python operations

In [21]:

```
S = np.array([[1,2],[3,4]])
T = np.array([[3,4],[4,5]])
print(S + T)
print(np.add(S,T))
```

```
[[4 6]
 [7 9]]
[[4 6]
 [7 9]]
```

In [22]:

```
print(S-T)
print(np.subtract(S,T))
```

```
[[-2 -2]
 [-1 -1]]
[[-2 -2]
 [-1 -1]]
```

In [25]:

```
print(S*T)
print(np.multiply(S,T))
print(S/T)
print(np.divide(S,T))
print(np.sqrt(S))
```

```
[[ 3  8]
 [12 20]]
[[ 3  8]
 [12 20]]
[[0.33333333 0.5       ]
 [0.75       0.8       ]]
[[0.33333333 0.5       ]
 [0.75       0.8       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

Sort

In [27]:

```
U = np.array([5,3,8,4,3])
U1 = np.sort(U)
print(U1)
```

```
[3 3 4 5 8]
```

Slicing and indexing

In [29]:

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

```
[3 4 5 6 7]
```

In [31]:

```
print(V[7:])
print(V[:4])
```

```
[ 8  9 10]
[1 2 3 4]
```

In [33]:

```
# to give elements with step wise

print(V[1:8:2]) # start from index 1 to 8, with step of 2
```

```
[2 4 6 8]
```

In [34]:

```
print(V[::3]) # start to end, with step 3
```

```
[ 1  4  7 10]
```

In [35]:

```
# slicing of 2- D array

W = np.array([[1,2,3,4],[5,6,7,8]])
print(W[0,0:2]) # from 0 index shape -> elements index 0 to n-1
```

```
[1 2]
```

In [38]:

```
print(W[0:2, 3]) # from shape index 0 to n-1, element index 3
```

```
[4 8]
```

In [ ]: