importing NumPy

To access NumPy and its functions import it in your Python code like this:

In [3]:
```
import numpy as np
```

In [13]:
```
#Creating an list
a = [1,3,5,7,23,2]
type(a)  #Checking the type of an list
```

Out[13]:
```
list
```

In [15]:
```
print(a) #Printing an list
```

```
[1, 3, 5, 7, 23, 2]
```

In [16]:
```
#Creating an list with different datatype
#every elements maintains its identity
b = [1,3,5, True, (2,5), "nashit", {3,5}]
print(b)
```

```
[1, 3, 5, True, (2, 5), 'nashit', {3, 5}]
```

In [17]:
```
type(b) #Checking datatype of a list
```

Out[17]:
```
list
```

In [9]:
```
# Checking version of numpy
numpy.__version__
```

Out[9]:
```
'1.23.5'
```

In [25]:
```
#0-D Arrays Construction:- 0-D arrays, or Scalars, are the elements in an array. Ea
# np.array -> array constructor
arr0 = np.array(5)
```

In [20]:
```
print(arr0) #Printing array
```

```
5
```

In [21]:
```
type(arr0)
```

Out[21]:
```
numpy.ndarray
```

In [23]:
```
#ndim attribute use to returns an integer that tells us how many dimensions the arr
#ndim -> no of dimension
arr0.ndim
```

Out[23]:
```
0
```

In [24]:
```
# 1-D Array Creation
arr1 = np.array([5, 5, 2, 3.4])
```

In [18]:
```
type(arr1)
```

Out[18]:
```
numpy.ndarray
```

In [17]:
```
arr1.ndim
```

Out[17]:
```
1
```

In [19]:
```
print(arr1)
```

```
[5.  5.  2.  3.4]
```

In [60]:
```
a1 = np.array([3, 6, 4, "vinod", 7, 4, 2])
print(a1)
print(type(a1))
print(a1.ndim)
a1.dtype
```

```
['3' '6' '4' 'vinod' '7' '4' '2']
<class 'numpy.ndarray'>
1
```

Out[60]:
```
dtype('<U11')
```

In [26]:
```
arr4 = np.array([5+9j,2,3])
print(arr4)
```

```
[5.+9.j 2.+0.j 3.+0.j]
```

In [29]:
```
a2 = np.array(['c', 'd', 't', 8])
print(a2)
```

```
['c' 'd' 't' '8']
```

In [30]:
```
a3 = np.array([9, 4, 0, 6.3, 9, 700])
print(a3)
```

```
[  9.    4.    0.    6.3   9.  700. ]
```

In [26]:
```
# 2-D Array Creation used to represent matrix
arr2 = np.array([
    [8, 2, 8, 5, 1, 0, 5],
    [0, 9, 0, 8, 1, 0, 9]
])
arr2.ndim #Checking dimentions of an array
```

Out[26]:
```
2
```

In [35]:
```
arr3 = np.array(
    [
    [9, 8]
]
)
arr3.ndim
```

Out[35]:
```
2
```

In [36]:
```
#Creating 3-D Array
arr3 = np.array (
    [

            [[6,3,4], [4,2,7]],[[3,2,5],['b', 'r', 3]]
    ]
)
arr3   #Printing an array
```

Out[36]:
```
array([[['6', '3', '4'],
        ['4', '2', '7']],

       [['3', '2', '5'],
        ['b', 'r', '3']]], dtype='<U11')
```

In [37]:
```
#Checking the dimentions of an array
arr3.ndim
```

Out[37]:
```
3
```

In [39]:
```
#Creating Higher Dimensional Arrays
arr = np.array([1, 2, 3, 4], ndmin=5)
arr
```

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

In [40]:
```
arr.ndim
```

Out[40]:
```
5
```

In [5]:
```
my_list = []
c = np.array(my_list)
print(type(my_list))
print(type(c))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

In [6]:
```
for i in range(1,5):
    x = int(input('elements: '))
    my_list.append(x)
d = np.array(my_list)
print(d)
```

```
[7 6 9 3]
```

In [7]:
```
type(d)
```

Out[7]:
```
numpy.ndarray
```

In [10]:
```
e = np.zeros(4)
print(e)
```

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

In [11]:
```
f = np.ones(5)
print(f)
```

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

In [13]:
```
g = np.arange(8)
print(g)
```

```
[0 1 2 3 4 5 6 7]
```

In [16]:
```
#np.arange(start, stop, steps, dtype)
h = np.arange(15,25)
h
```

Out[16]:
```
array([15, 16, 17, 18, 19, 20, 21, 22, 23, 24])
```

In [4]:
```
i = np.arange(0,10,4)
i
```

Out[4]:
```
array([0, 4, 8])
```

In [5]:
```
#The NumPy linspace function creates sequences of evenly spaced values within a def
#Spacial
#linespace
#IQR (25%), 75%

j = np.linspace(0,10, num= 6)
j
```

Out[5]:
```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [59]:
```
#(- -> +)
#array elements
l = np.random.randn(6)
l.dtype
```

Out[59]:
```
dtype('float64')
```

In [8]:
```
# A Numpy array of random floats between 0 (inclusive) and 1 (exclusive). If size i
m = np.random.ranf(6)
print(m)
print(m.dtype)
```

```
[0.99259845 0.48961439 0.7376175  0.2781377  0.08994099 0.59816059]
float64
```

In [9]:
```
# Return random integers from low to high
#(start, stop, count)
# random.randint(low, high=None, size=None, dtype=int)
n = np.random.randint(4, 20, 5)
print(n)
print(n.dtype)
```

```
[16 14  4 17  9]
int32
```

In [53]:
```
#(min, max, count)
o = np.linspace(100, 1000, num=5)
o
```

Out[53]:
```
array([ 100.,  400.,  700., 1000.])
```

In [54]:
```
o.dtype
```

Out[54]:
```
dtype('float64')
```

In [50]:
```
h = np.zeros((6,4))
h
```

Out[50]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [51]:
```
i = np.ones((3,3))
i
```

Out[51]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [54]:
```
j = np.eye(5)
j
```

Out[54]:
```
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.]])
```

In [56]:
```
k = np.eye(4, 6)
print(k)
k.ndim
```

Out[56]:
```
2
```

In [7]:
```
c = numpy.array([[[5,2,6,8,9,0]]])
c.shape
```

Out[7]:
```
(1, 1, 6)
```

In [86]:
```
a = np.array([3,5,7,1,6,4,1,2,2])
print(a.ndim)
# dimensions 2d(x*y), 3d(x*y*z)
b = a.reshape(3,3)
print(b)
print(b.ndim)
```

```
1
[[3 5 7]
 [1 6 4]
 [1 2 2]]
2
```

In [88]:
```
a = np.array([3,5,7,1,6,4,1,2,2,4,2,8])
print("original dimensions:" ,a.ndim)
b = a.reshape(2,3,2)
print(b)
print("reshaped dimensions: ", b.ndim)
```

```
original dimensions: 1
[[[3 5]
  [7 1]
  [6 4]]

 [[1 2]
  [2 4]
  [2 8]]]
reshaped dimensions:  3
```

In [21]:
```
d = numpy.array([[3,7,7],[7,9,7]])
d.shape
#d.ndim
```

Out[21]:
```
(2, 3)
```

In [22]:
```
e = numpy.array([
    [[2,5,9],[5,8,9],[7,0,2],[5,4,9]]
])
```

In [23]:
```
e.shape
```

Out[23]:
```
(1, 4, 3)
```

In [24]:
```
e.ndim
```

Out[24]:
```
3
```

In [25]:
```
f = numpy.array([3,7,9,2,4])
#                0,1,2,3,4
#               -5, -4, -3, -2, -1
f[1]
```

Out[25]:
```
7
```

In [26]:
```
f[-2]
```

Out[26]:
```
2
```

In [30]:
```
g = numpy.array([
    [7, 3, 8],
    [6, 2, 9]
])

g[1,2]
```

Out[30]:
```
9
```

In [31]:
```
g[-2,-3]
```

Out[31]:
```
7
```

In [34]:
```
h = numpy.array([
    [
        [8,3,5],
        [7,2,4]]
])
h.shape
```

Out[34]:
```
(1, 2, 3)
```

In [35]:
```
h[0]
```

Out[35]:
```
array([[8, 3, 5],
       [7, 2, 4]])
```

In [40]:
```
h[0, 1, 2]
```

Out[40]:
```
4
```

In [42]:
```
a = numpy.array([4, 2, 6,3])
b = numpy.array([8,3,4, 6])
#1/4 =0.25 =0; 2/3 = 0.66 = 0; 6/4= 1;
# 4/8 = .5, 2/3 = .66, 6/4 = 1.5, 3/6 = ,.5
# -4, -1, 2, -3
a+b
```

Out[42]:
```
array([12,  5, 10,  9])
```

In [43]:
```
numpy.add(a,b)
```

Out[43]:
```
array([12,  5, 10,  9])
```

In [45]:
```
c = numpy.array([[4, 3, 6, 2], [4, 2, 5, 7]])
d = numpy.array([[1, 5, 2, 9], [3, 0, 3, 2]])
c+d
```

Out[45]:
```
array([[ 5,  8,  8, 11],
       [ 7,  2,  8,  9]])
```

In [46]:
```
a-b
```

Out[46]:
```
array([-4, -1,  2, -3])
```

In [47]:
```
numpy.subtract(a,b)
```

Out[47]:
```
array([-4, -1,  2, -3])
```

In [48]:
```
b-a
```

Out[48]:
```
array([ 4,  1, -2,  3])
```

In [49]:
```
numpy.subtract(b,a)
```

Out[49]:
```
array([ 4,  1, -2,  3])
```

In [50]:
```
a*b
```

Out[50]:
```
array([32,  6, 24, 18])
```

In [51]:
```
c*d
```

Out[51]:
```
array([[ 4, 15, 12, 18],
       [12,  0, 15, 14]])
```

In [52]:
```
numpy.multiply(a,b)
```

Out[52]:
```
array([32,  6, 24, 18])
```

In [53]:
```
a/b
```

Out[53]:
```
array([0.5       , 0.66666667, 1.5       , 0.5       ])
```

In [54]:
```
numpy.divide(a,b)
```

Out[54]:
```
array([0.5       , 0.66666667, 1.5       , 0.5       ])
```

In [55]:
```
numpy.mod(a,b)
```

Out[55]:
```
array([4, 2, 2, 3])
```

In [56]:
```
a%b
```

Out[56]:
```
array([4, 2, 2, 3])
```

In [58]:
```
numpy.reciprocal(a)
```

Out[58]:
```
array([0, 0, 0, 0])
```

In [59]:
```
#statistical methods
numpy.min(a)
```

Out[59]:
```
2
```

In [60]:
```
d = numpy.array([1,2,3,4])
# 1/1 = 1; 1/2 = 0.5= 0; 1/3 = 0; 0.24 n= 0
numpy.reciprocal(d)
```

Out[60]:
```
array([1, 0, 0, 0])
```

In [68]:
```
a
```

Out[68]:
```
array([4, 2, 6, 3])
```

In [74]:
```
print(numpy.max(a))
print(numpy.argmax(a)) #index position
```

```
6
2
```

In [70]:
```
numpy.sqrt(a)
```

Out[70]:
```
array([2.        , 1.41421356, 2.44948974, 1.73205081])
```

In [71]:
```
numpy.sin(a)
```

Out[71]:
```
array([-0.7568025 ,  0.90929743, -0.2794155 ,  0.14112001])
```

In [72]:
```
numpy.cos(a)
```

Out[72]:
```
array([-0.65364362, -0.41614684,  0.96017029, -0.9899925 ])
```

In [75]:
```
print(numpy.min(a))
print(numpy.argmin(a))
```

```
2
1
```

In [41]:
```
#Broadcasting

a = numpy.array([4, 2, 5, 1]) # shape(4,)
b = numpy.array([5]) # shape(1,)
a+b
```

Out[41]:
```
array([ 9,  7, 10,  6])
```

In [42]:
```
# Broadcast error -> dimension same + similar shape/shape
```

In [49]:
```
a = numpy.array([[1,2], [3,2]])
print(a.shape)
b = numpy.array([[1,4,3], [3,2]])
print(b.shape)
#a+b
```

```
(2, 2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[49], line 3
      1 a = numpy.array([[1,2], [3,2]])
      2 print(a.shape)
----> 3 b = numpy.array([[1,4,3], [3,2]])
      4 print(b.shape)
      5 #a+b

ValueError: setting an array element with a sequence. The requested array has an in
homogeneous shape after 1 dimensions. The detected shape was (2,) + inhomogeneous p
art.
```

In [62]:
```
a = numpy.array([[2,3], [4,2]])
b = numpy.array([[1, 3, 4], [3,2,5]])
print(b.shape)
a.shape
```

```
(2, 3)
```
Out[62]:
```
(2, 2)
```

In [63]:
```
b = numpy.array([[1,3,4], [3,2,5]])
print(b.shape)
```

```
(2, 3)
```

In [64]:
```
a*b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[64], line 1
----> 1 a*b

ValueError: operands could not be broadcast together with shapes (2,2) (2,3)
```

Adding and sorting elements

In [5]:
```
# Creating an array
arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
```

In [6]:
```
np.sort(arr) #sorting an array
```

Out[6]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

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

In [8]:
```
#array concatination
np.concatenate((a,b))
```

Out[8]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Creating an array from existing data

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

In [10]:
```
arr1 = a[3:8]
arr1
```

Out[10]:
```
array([4, 5, 6, 7, 8])
```

Creating matrices

You can pass Python lists of lists to create a 2-D array (or “matrix”) to represent them in NumPy.

In [11]:
```
data = np.array([[1, 2], [3, 4], [5, 6]])
```

In [12]:
```
data
```

Out[12]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [13]:
```
# Indexing and slicing operations are useful when you’re manipulating matrices:
data[0, 1]
```

Out[13]:
```
2
```

In [14]:
```
data[1:3]
```

Out[14]:
```
array([[3, 4],
       [5, 6]])
```

In [15]:
```
data[0:2, 0]
```

Out[15]:
```
array([1, 3])
```

In [17]:
```
#You can aggregate matrices the same way you aggregated vectors:
data.max()
```

Out[17]:
```
6
```

In [18]:
```
data.min()
```

Out[18]:
```
1
```

In [19]:
```
data.sum()
```

Out[19]:
```
21
```

In [ ]:
```

```