1-D array

In [15]:
```
import numpy as np
```

In [16]:
```
a = np.array([3,2, 5,6])
type(a)
```

Out[16]:
```
numpy.ndarray
```

In [17]:
```
# it will convert into float value
np.array([3, 6, 8, 4.2, 3, 2.3, 2.3, 8.09, ])
```

Out[17]:
```
array([3.  , 6.  , 8.  , 4.2 , 3.  , 2.3 , 2.3 , 8.09])
```

In [18]:
```
print(len(b))
print(type(b)) # both vector and matrix belong to ndarray class
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_18712/2913355245.py in <module>
----> 1 print(len(b))
      2 print(type(b)) # both vector and matrix belong to ndarray class

NameError: name 'b' is not defined
```

In [19]:
```
np.zeros(3)
```

Out[19]:
```
array([0., 0., 0.])
```

In [20]:
```
np.ones(5)
```

Out[20]:
```
array([1., 1., 1., 1., 1.])
```

In [21]:
```
np.empty(3)
```

Out[21]:
```
array([0., 0., 0.])
```

In [22]:
```
#with range of elements
e = np.arange(6) # 6 is exclusive
e
```

Out[22]:
```
array([0, 1, 2, 3, 4, 5])
```

In [23]:
```
# with specific range of elements
e = np.arange(2 , 10)
e
```

Out[23]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [24]:
```
e
```

Out[24]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [25]:
```
# continue with interval
e = np.arange(3, 21, 4)
print(e)
```

```
[ 3  7 11 15 19]
```

In [26]:
```
# linearly space arrays
e = np.linspace(2, 10, num=6)
print(e)
```

```
[ 2.   3.6  5.2  6.8  8.4 10. ]
```

- **specific data types in array**

In [27]:
```
# specific data types in array
i = np.ones(5, dtype =np.int8)
i
```

Out[27]:
```
array([1, 1, 1, 1, 1], dtype=int8)
```

In [28]:
```
np.ones(5, dtype = np.float16)
```

Out[28]:
```
array([1., 1., 1., 1., 1.], dtype=float16)
```

2-D array

In [29]:
```
a = np.zeros((3, 5))
a
```

Out[29]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [30]:
```
np.ones((2,7))
```

Out[30]:
```
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

In [31]:
```
np.ones((6, 7))
```

Out[31]:
```
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., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

In [32]:
```
np.empty((3, 5))
```

Out[32]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
```

```
       [0., 0., 0., 0., 0.]])
```

In [33]:
```
a1 = np.array([[2, 4, 8, 9], [2, 3, 4, 4], [10, 11, 33, 6]])
a1
```

Out[33]:
```
array([[ 2,  4,  8,  9],
       [ 2,  3,  4,  4],
       [10, 11, 33,  6]])
```

In [34]:
```
a1.ndim # ndim is used for finding the dimension of an array
```

Out[34]:
```
2
```

In [35]:
```
b1 = np.array([[3, 2, 2, 2], [30, 22, 19, 90], [97, 2, 3, 2]])
b1
```

Out[35]:
```
array([[ 3,  2,  2,  2],
       [30, 22, 19, 90],
       [97,  2,  3,  2]])
```

In [36]:
```
# Concatenate two arrays
print(np.concatenate((a1, b1), axis = 0))
print('\n\n')
print(np.concatenate((a1, b1), axis = 1))
```

```
[[ 2  4  8  9]
 [ 2  3  4  4]
 [10 11 33  6]
 [ 3  2  2  2]
 [30 22 19 90]
 [97  2  3  2]]



[[ 2  4  8  9  3  2  2  2]
 [ 2  3  4  4 30 22 19 90]
 [10 11 33  6 97  2  3  2]]
```

3-D array

In [37]:
```
d = np.arange(24).reshape((2, 3, 4))
d
```

Out[37]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

In [38]:
```
d.ndim
```

Out[38]:
```
3
```

Array functions

In [39]:
```
a = np.array([[2, 3, 2, 89, 90], [21, 3, 55, 43, 90],
              [33, 7, 89, 4, 90], [9, 3, 7, 5, 90]])
a
```

Out[39]:
```
array([[ 2,  3,  2, 89, 90],
       [21,  3, 55, 43, 90],
       [33,  7, 89,  4, 90],
       [ 9,  3,  7,  5, 90]])
```

In [40]:
```
type(a)
```

Out[40]:
```
numpy.ndarray
```

In [41]:
```
len(a) # find length of array
```

Out[41]:
```
4
```

In [70]:
```
np.array(20, ndmin= 5)
```

Out[70]:
```
array([[[[[20]]]]])
```

In [43]:
```
a.any()
a
```

Out[43]:
```
array([[ 2,  3,  2, 89, 90],
       [21,  3, 55, 43, 90],
       [33,  7, 89,  4, 90],
       [ 9,  3,  7,  5, 90]])
```

In [44]:
```
t = a.transpose()
t
```

Out[44]:
```
array([[ 2, 21, 33,  9],
       [ 3,  3,  7,  3],
       [ 2, 55, 89,  7],
       [89, 43,  4,  5],
       [90, 90, 90, 90]])
```

In [45]:
```
s = np.arange(8)
s
```

Out[45]:
```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [46]:
```
# split array into sub array
np.split(s, 4)
```

Out[46]:
```
[array([0, 1]), array([2, 3]), array([4, 5]), array([6, 7])]
```

In [47]:
```
# insert element into specific point
np.insert(s, 0, 90, axis=0)
```

Out[47]:
```
array([90,  0,  1,  2,  3,  4,  5,  6,  7])
```

In [48]:

```
#to find the size of an array
a.size
```

Out[48]:
```
20
```

In [49]:
```
a.shape
```

Out[49]:
```
(4, 5)
```

In [50]:
```
s.shape
```

Out[50]:
```
(8,)
```

In [51]:
```
# reshape array
a = np.arange(9)  # 3*3
a
```

Out[51]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [52]:
```
a.reshape(3, 3)  # 3*3 = 9
```

Out[52]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [53]:
```
# reshape are into new shape
np.reshape(a, newshape=(3, 3))
```

Out[53]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [54]:
```
# Conversion into d/f dimensions

# row wise dimension
b = a[np.newaxis, : ]
b
```

Out[54]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

In [55]:
```
b.shape  # now b became 2-D array
```

Out[55]:
```
(1, 9)
```

In [56]:
```
# column wise dimension
c = a[:, np.newaxis ]
c
```

Out[56]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
```

```
       [6],
       [7],
       [8]])
```

In [57]:
```
a
```

Out[57]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

Array indexing and slicing

In [58]:
```
a[2]
```

Out[58]:
```
2
```

In [59]:
```
a[8]
```

Out[59]:
```
8
```

In [60]:
```
# slicing
a[::]
```

Out[60]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [61]:
```
a[2 : ]
```

Out[61]:
```
array([2, 3, 4, 5, 6, 7, 8])
```

In [62]:
```
a[ : 5]
```

Out[62]:
```
array([0, 1, 2, 3, 4])
```

In [63]:
```
a[-4 : -1]
```

Out[63]:
```
array([5, 6, 7])
```

In [64]:
```
a.size
```

Out[64]:
```
9
```

In [65]:
```
a[:,np.newaxis]
```

Out[65]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8]])
```

In [ ]: