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

In [2]:
```
#creating numpy
a = np.array([1,2,3])
b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
c = np.array([[(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]], dtype = float)
```

Initial Placeholders

In [3]:
```
#Create an array of zeros
np.zeros((3,4)) #creating 2D array
```

Out[3]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [4]:
```
#Create an array of ones
np.ones((2,3,4))
```

Out[4]:
```
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.]]])
```

In [5]:
```
#Create an array of evenly spaced values (step value)
d = np.arange(10,25,2)
print(d)
```

```
[10 12 14 16 18 20 22 24]
```

In [6]:
```
#Create an array of evenlyspaced values (number of samples)
np.linspace(0,8,8)
```

Out[6]:
```
array([0.        , 1.14285714, 2.28571429, 3.42857143, 4.57142857,
       5.71428571, 6.85714286, 8.        ])
```

In [7]:
```
#Create a constant array
#Create a 1D array
e = np.full((2),9)
print(e)
print(e.ndim) #to find no of dimension
```

```
[9 9]
1
```

In [8]:
```
#Create a 2D array
e = np.full((2,2),9)
print(e)
print(e.ndim)
```

```
[[9 9]
 [9 9]]
2
```

In [9]:
```
#Create a 3D array
e = np.full((2,2,2),9)
print(e)
print(e.ndim)
```

```
[[[9 9]
  [9 9]]

 [[9 9]
  [9 9]]]
3
```

In [10]:
```
#Create a 2X2 identity matrix
f = np.eye(3)
print(f)
```

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

In [11]:
```
#Create an array with random values
np.random.random((2,2))
```

Out[11]:
```
array([[0.45885198, 0.67821196],
       [0.93937292, 0.19992625]])
```

In [12]:
```
#to find shape of an Array
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [5, 7, 7, 8]])
print(arr.shape)

#Output explaination
# 3 denotes no of ROWS
# 4 Denotes no of Columns
```

```
(3, 4)
```

In [13]:
```
#To Find NO of elements in array
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [5, 6, 7, 8],
                [5, 7, 7, 8]])

print(arr.size)
```

```
16
```

In [14]:
```
#To Find len of array
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [5, 6, 7, 8],
                [5, 7, 7, 8]])
len(arr)

#output explaintion
# It Will count only the rows
#output: 4
```

Out[14]:
```
4
```

Access Array Elements

In [15]:
```
#You can access an array element by referring to its index number.
#The indexes in NumPy arrays start with 0,
#meaning that the first element has index 0
```

In [16]:
```
arr = np.array([1, 2, 3, 4])

print(arr[3])
```

```
4
```

In [17]:
```
# Adding two Index position
arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

```
7
```

In [18]:
```
#Access 2-D Arrays
#Access the element on the first row, second column:
arr = np.array([[1,2,3,4,5],
                [6,7,8,9,10]]
               )

print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

In [19]:
```
#Access 3-D Arrays

arr = np.array([[[1, 2, 3], [4, 5, 6]],
                [[7, 8, 9], [10, 11, 12]]])

#Access by index
print(arr[1, 1, 1])
```

```
11
```

Data Types

Data Types in NumPy NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc. Below is a list of all data types in NumPy and the characters used to represent them. i - integer b - boolean u - unsigned integer(just like integers (whole numbers) but have the property that they don't have a + or - sign associated with them.) f - float c - complex float M - datetime O - object S - string

In [25]:
```
#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 [26]:
```
#integer
arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```

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

In [28]:
```
#Change data type from float to integer by using 'i' as parameter value:
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

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

In [29]:
```
#Change data type from integer to boolean:
arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)
```

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

In [32]:
```
# copy method
arr = np.array([1, 7, 3, 4, 65])
x = arr.copy()
arr[0] = 642
print(arr)
print(x)
```

```
[642   7   3   4  65]
[ 1  7  3  4 65]
```

In [34]:
```
#view method
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

In [35]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```

Reshaping arrays

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

In [36]:
```
#Convert the following 1-D array with 12 elements into a 2-D array.
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [48]:
```
#Convert the following 1-D array with 12 elements into a 3-D array.
#The outermost dimension will have 2 arrays that contains 3 arrays,
#each with 2 elements:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

In [56]:
```
# Python Program to create
# a data type object
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

In [57]:
```
# Addition of two Arrays
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [58]:
```
# subtract of the elements
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

In [60]:
```
#multiplication or  product
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

In [61]:
```
#division
print(x / y)
print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

In [62]:
```
#square root
print(np.sqrt(x))
print(np.sqrt(y))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]]
[[2.23606798 2.44948974]
 [2.64575131 2.82842712]]
```

Sorting Arrays Sorting means putting elements in an ordered sequence. The NumPy ndarray object has a function called sort(), that will sort a specified array.

In [80]:
```
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

```
[0 1 2 3]
```

Slicing arrays Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: [start:end]. We can also define the step, like this: [start:end:step]. If we don't pass start its considered 0 If we don't pass end its considered length of array in that dimension If we don't pass step its considered 1

In [65]:
```
#Slice elements from index 1 to index 5 from the following array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

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

In [66]:
```
#Slice elements from index 4 to the end of the array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

```
[5 6 7]
```

In [67]:
```
#Slice elements from the beginning to index 4 (not included):
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

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

Negative Slicing

In [68]:
```
#Slice from the index 3 from the end to index 1 from the end:
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

```
[5 6]
```

In [71]:
```
#Use the step value to determine the step of the slicing:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 4, 1])

print(arr[1:8:2]) #step is apply like step-1
```

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

In [72]:
```
#Return every other element from the entire array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```

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

In [73]:
```
#Slicing 2-D Arrays
#From the second element, slice elements from index 1 to index 4 (not included):
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

```
[7 8 9]
```

In [79]:
```
#return both element index 3
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 3])
```

```
[4 9]
```

In [ ]: