Numpy

In [70]:
```
# now we will create a numpy ndarry object
```

In [71]:
```
import numpy as np
```

In [72]:
```
x=np.array([1,2,3,4,5,])
```

In [73]:
```
print(x)
print(type(x))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

we can also pass a list, tuple or any array like object with array() . and it will be converted to ndarray

In [74]:
```
y=np.array((1,2,3,4,5))
```

In [75]:
```
print(y)
print(type(y))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

dimensions in array a dimensions in array is one level of array depth (nested array)

0-D arraya-- scalars,are the element in an array,each value in an array is 0-D array.

In [76]:
```
# now will create 0-D array with value 42
```

In [77]:
```
x=np.array(42)
```

In [78]:
```
print(x)
```

```
42
```

In [79]:
```
# now will create 1-D array
```

In [80]:
```
x1=np.array([1,2,3,4,5])
```

In [81]:
```
print(x1)
```

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

In [82]:
```
# now will create 2-D array, with certain values
```

In [83]:
```
x2=np.array([[1,2,3,4,5],[6,7,8,9,10]])
```

In [84]:
```
print(x2)
```

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

In [85]:
```
# now will create 3-D array, with certain values
```

In [86]:
```
x3=np.array([[[1,2,3,],[4,5,6,]], [[1,2,3],[4,5,6]]])
```

In [87]:
```
print(x3)
```

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

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

In [88]:
```
# how many dimensions the arryay have : ndim attribute values
```

In [89]:
```
x=np.array(42)
x1=np.array([1,2,3,4,5])
x2=np.array([[1,2,3,4,5],[6,7,8,9,10]])
x3=np.array([[[1,2,3,],[4,5,6,]], [[1,2,3],[4,5,6]]])
```

In [90]:
```
print(x.ndim)
print(x1.ndim)
print(x2.ndim)
print(x3.ndim)
```

```
0
1
2
3
```

In [91]:
```
# now will create 5-D array, and verify that it has 5 dimensions
```

In [92]:
```
x5=np.array([1,2,3,4,5], ndmin=5)
```

In [93]:
```
print(x5)
```

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

In [94]:
```
print(x5.ndim)
```

```
5
```

Array indexing

In [95]:
```
#start with 0, second 1
```

In [96]:
```
x1=np.array([1,2,3,4,5,6,7])
```

In [97]:
```
x1[0]
```

Out[97]: 1

In [98]:
```
x1[4]
```

Out[98]: 5

In [99]:
```
x1[6]
```

Out[99]: 7

In [100]:
```
# we can third and fourth element from adding them.
```

In [101]:
```
x1=np.array([1,2,3,4,5,6,7])
```

In [102]:
```
x1[2]+ x1[4]
```

Out[102]: 8

In [103]:
```
#Accessing the 2-d it is like a row and col.
```

In [104]:
```
x1=np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
```

In [105]:
```
print("2nd element in the  1st rows",x1[0,1])
```

```
2nd element in the  1st rows 2
```

In [106]:
```
print("5nd element in the  2st rows",x1[1,4])
```

```
5nd element in the  2st rows 12
```

In [107]:
```
#Accessing the 3-d same as accessing
```

In [108]:
```
x1=np.array([[[1,2,3,],[4,5,6,]], [[1,2,3],[4,5,6]]])
```

In [109]:
```
print(x1[0,1,2])
```

```
6
```

In [110]:
```
print(x1[1,0,1])
```

```
2
```

numpy slicing array

slicing in python means taking elements from one give index to another index

In [111]:
```
#[start:end],[start:end:step]
```

In [112]:
```
# now we will slice an element from 1 to 5
```

In [113]:
```
x1=np.array([1,2,3,4,5,6,7,8,9,10])
```

In [114]:
```
x1[1:5]
```

Out[114]:
```
array([2, 3, 4, 5])
```

In [115]:
```
x1[4:]
```

Out[115]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [116]:
```
x1[:4]
```

Out[116]:
```
array([1, 2, 3, 4])
```

In [117]:
```
x1[:-1]
```

Out[117]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [118]:
```
x1[-1:]
```

Out[118]:
```
array([10])
```

In [119]:
```
x1[-3:-1]
```

Out[119]:
```
array([8, 9])
```

In [120]:
```
x1[2:-3]
```

Out[120]:
```
array([3, 4, 5, 6, 7])
```

In [121]:
```
x1[::2]
```

Out[121]:
```
array([1, 3, 5, 7, 9])
```

In [122]:
```
x1[2:7:2]
```

Out[122]:
```
array([3, 5, 7])
```

In [123]:
```
# slicing 2-d array  #print 8,9,10
```

In [124]:
```
x1=np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
```

In [125]:
```
x1[1,0:3]
```

Out[125]:
```
array([ 8,  9, 10])
```

numpy array datatype

In [126]:
```
#data type in np
```

bool, int , uint , float , complex , timedelta , unsigned , datetime , object , string , unicode

In [127]:
```
#checking the data type of numpy array
```

In [128]:
```
x1=np.array([1,2,3,4,5,6,7,8,9,10])
```

In [129]:
```
x1.dtype
```

Out[129]:
```
dtype('int32')
```

In [130]:
```
#checking the data type of numpy array --string
```

In [131]:
```
x1=np.array(['a','s','j','d'])
```

In [132]:
```
x1.dtype
```

Out[132]:
```
dtype('<U1')
```

In [133]:
```
#checking  array with a defined data type
```

In [134]:
```
x1=np.array([1,2,3,4,5,6,7,8,9,10], dtype='S')
```

In [135]:
```
x1
```

Out[135]:
```
array([b'1', b'2', b'3', b'4', b'5', b'6', b'7', b'8', b'9', b'10'],
      dtype='|S2')
```

In [136]:
```
x1=np.array(['a','s','j','4','5','6'], dtype='i')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[136], line 1
----> 1 x1=np.array(['a','s','j','4','5','6'], dtype='i')

ValueError: invalid literal for int() with base 10: 'a'
```

numpy [copy, view]

In [137]:
```
#difference between np copy and view
```

In [138]:
```
# we will make a copy first
```

In [139]:
```
x1=np.array([1,2,3,4,5])
```

In [140]:
```
x2=x1.copy()
```

In [141]:
```
print(x1)
print(x2)
```

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

In [142]:
```
#now we will make a view, change original, display both
```

In [143]:
```
x1=np.array([1,2,3,4,5])
```

In [144]:
```
x2=x1.view()
```

In [145]:
```
x1[0]=6
```

In [146]:
```
print(x1)
print(x2)
```

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

numpy array shape

In [147]:
```
#shape of an array
```

In [148]:
```
#now we will try to get the shape of an array
```

In [149]:
```
x1=np.array([[1,2,3,4],[6,7,8,9]])
```

In [150]:
```
x1.shape #2 dimension and 4 element
```

Out[150]:
```
(2, 4)
```

In [151]:
```
#now we will create a 5-d array using .ndmin
```

In [152]:
```
x1=np.array([1,2,3,4],ndmin=5)
```

In [153]:
```
x1
```

Out[153]:
```
array([[[[[1, 2, 3, 4]]]]])
```

In [154]:
```
x1.shape
```

Out[154]:
```
(1, 1, 1, 1, 4)
```

In [155]:
```
x3=np.array([[[1,2,3,],[4,5,6,]], [[1,2,3],[4,5,6]]])
```

In [156]:
```
x3.shape
```

Out[156]:
```
(2, 2, 3)
```

numpy array reshape

In [157]:
```
# means changing the shape of an array, like adding or removing the element
```

In [158]:
```
#reshaping from 1-d to 2-d
```

In [159]:
```
x1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
```

In [160]:
```
x2=x1.reshape(2,6)
```

In [161]:
```
x2
```

Out[161]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

In [162]:
```
#reshaping from 1-d to 3-d
```

In [163]:
```
x3=x1.reshape(2,3,2)
```

In [164]:
```
x3
```

Out[164]:
```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6]],

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

In [165]:
```
x4=x1.reshape(4,3)
```

In [166]:
```
x4
```

Out[166]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [167]:
```
x4=x1.reshape(4,4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[167], line 1
----> 1 x4=x1.reshape(4,4)

ValueError: cannot reshape array of size 12 into shape (4,4)
```

In [168]:
```
# return copy or view
```

In [169]:
```
x1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
```

In [170]:
```
x1.reshape(3,4)
```

Out[170]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [171]:
```
x1.reshape(3,4).base
```

Out[171]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [172]:
```
#unknown dimensions
```

In [173]:
```
x1=np.array([1,2,3,4,5,6,7,8])
```

In [174]:
```
x1.reshape(2,2,-1)
```

Out[174]:
```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

In [175]:
```
#flattening the array by converting multidimensional array in 1-D
```

In [176]:
```
x1=np.array([[1,2,3],[4,5,6]])
```

In [177]:
```
x1
```

Out[177]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [178]:
```
x1.reshape(-1)
```

Out[178]:
```
array([1, 2, 3, 4, 5, 6])
```

numpy Iterating

In [179]:
```
#going to element step by step like loop
```

In [180]:
```
#Iterating the element of 1-d
```

In [181]:
```
x1=np.array([1,2,3,4])
```

In [182]:
```
for i in x1:
    print(i)
```

```
1
2
3
4
```

In [183]:
```
#Iterating the element of 2-d
```

In [184]:
```
x1=np.array([[1,2,3],[4,5,6]])
```

In [185]:
```
for i in x1:
    print(i)
```

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

In [186]:
```
#Iterating on each scalar element of the 2-d
```

In [187]:
```
x1=np.array([[1,2,3],[4,5,6]])
for i in x1:
    for a in i:
        print(a)
```

```
1
2
3
4
5
6
```

In [188]:
```
#Iterating the element of 3-d
```

In [189]:
```
x1=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[9,10,11]]])
```

In [190]:
```
print(x1.ndim)
```

```
3
```

In [191]:
```
for i in x1:
    for a in i:
        for b in a:
            print(b)
```

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

In [192]:
```
for i in x1:
    for a in i:
        for b in a:
            print(b,end=" ")
```

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

nditer()

In [193]:
```
#Iterating arrays using the nditer() function.
```

In [194]:
```
# now we will iterate on each scalar problem
```

In [195]:
```
x1=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[9,10,11]]])
```

In [196]:
```
for i in np.nditer(x1):
    print(i,end=" ")
```

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

numpy array joining

In [197]:
```
#heer for this we will pass concatenate.
```

concatenate()

In [198]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
```

In [199]:
```
np.concatenate((x,y))
```

Out[199]:
```
array([1, 2, 3, 4, 5, 6])
```

In [200]:
```
#join of 2-d along with axis 1
```

In [201]:
```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
```

In [202]:
```
np.concatenate((x,y),axis=1)
```

Out[202]:
```
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

In [203]:
```
# joining array using the stack function.
```

In [204]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
```

In [205]:
```
np.stack((x,y),axis=1)
```

Out[205]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [206]:
```
# stacking along with rows : hstack()
```

In [207]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
```

In [208]:
```
np.hstack((x,y))
```

Out[208]:
```
array([1, 2, 3, 4, 5, 6])
```

In [209]:
```
# stacking along with col.
```

In [210]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
```

In [211]:
```
np.vstack((x,y))
```

Out[211]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [212]:
```
# stacking along with depth.
```

In [213]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
```

In [214]:
```
np.dstack((x,y))
```

Out[214]:
```
array([[[1, 4],
        [2, 5],
        [3, 6]]])
```

numpy splitting

In [215]:
```
#it is reverse to joining, breacking the array
```

In [216]:
```
# arr_split()
```

In [217]:
```
# splite the array in the tree parts.
```

In [218]:
```
x1=np.array([1, 2, 3, 4, 5, 6])
```

In [219]:
```
np.split(x1,2)
```

Out[219]:
```
[array([1, 2, 3]), array([4, 5, 6])]
```

In [220]:
```
np.split(x1,3)
```

Out[220]:
```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

In [221]:
```
# now we will splite in 4 parts.
```

In [222]:
```
np.split(x1,4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[222], line 1
----> 1 np.split(x1,4)

File <__array_function__ internals>:180, in split(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\numpy\lib\shape_base.py:872, in split(ary,
indices_or_sections, axis)
    870     N = ary.shape[axis]
    871     if N % sections:
--> 872         raise ValueError(
    873             'array split does not result in an equal division') from
None
    874 return array_split(ary, indices_or_sections, axis)

ValueError: array split does not result in an equal division
```

In [223]:
```
np.array_split(x1,4)
```

Out[223]:
```
[array([1, 2]), array([3, 4]), array([5]), array([6])]
```

In [224]:
```
#split into array with index
```

In [225]:
```
x1=np.array([1, 2, 3, 4, 5, 6])
```

In [226]:
```
x2=np.array_split(x1,3)
```

In [227]:
```
x2
```

Out[227]:
```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

In [228]:
```
print(x2[0])
print(x2[1])
print(x2[2])
```

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

In [229]:
```
# splitting the 2-d array
```

In [230]:
```
x1=np.array([[1,2,3],[4,5,6],[7,8,9],[11,12,13],[14,15,16]])
```

In [231]:
```
x2=np.array_split(x1,3)
```

In [232]:
```
x2
```

Out[232]:
```
[array([[1, 2, 3],
        [4, 5, 6]]),
 array([[ 7,  8,  9],
        [11, 12, 13]]),
 array([[14, 15, 16]])]
```

In [233]:
```
x2[1]
```

Out[233]:
```
array([[ 7,  8,  9],
       [11, 12, 13]])
```

In [234]:
```
# split the 2-D into three 2-d arrays.
```

In [235]:
```
x1=np.array([[1,2,3],[4,5,6],[7,8,9],[11,12,13],[14,15,16]])
```

In [236]:
```
np.array_split(x1,2)
```

Out[236]:
```
[array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]),
 array([[11, 12, 13],
        [14, 15, 16]])]
```

In [237]:
```
#spliting the 2-d into three 2-d along with rows.
```

In [238]:
```
x1=np.array([[1,2,3],[4,5,6],[7,8,9],[11,12,13],[14,15,16]])
```

In [239]:
```
np.array_split(x1,3)
```

Out[239]:
```
[array([[1, 2, 3],
        [4, 5, 6]]),
 array([[ 7,  8,  9],
        [11, 12, 13]]),
 array([[14, 15, 16]])]
```

In [240]:
```
np.array_split(x1,3,axis=1)
```

Out[240]:
```
[array([[ 1],
        [ 4],
        [ 7],
        [11],
        [14]]),
 array([[ 2],
        [ 5],
        [ 8],
        [12],
        [15]]),
 array([[ 3],
        [ 6],
        [ 9],
        [13],
        [16]])]
```

In [241]:
```
# atternate sol using hsplit, opposite hstack ()
```

In [242]:
```
x1=np.array([[1,2,3],[4,5,6],[7,8,9],[11,12,13],[14,15,16]])
```

In [243]:
```
np.hsplit(x1,3)
```

Out[243]:
```
[array([[ 1],
        [ 4],
        [ 7],
        [11],
        [14]]),
 array([[ 2],
        [ 5],
        [ 8],
        [12],
        [15]]),
 array([[ 3],
        [ 6],
        [ 9],
        [13],
        [16]])]
```

numpy searching array

you can search an array for a certain value and return the indexes that get the match

In [244]:
```
#using where()
```

In [245]:
```
x1=np.array([1,2,3,4,5,6,3,5,2])
```

In [246]:
```
np.where(x1==2)
```

Out[246]:
```
(array([1, 8], dtype=int64),)
```

In [247]:
```
#now we will find the indexes where the values are even.
```

In [248]:
```
x1=np.array([1,2,3,4,5,6,3,5,2])
```

In [249]:
```
np.where(x1%2==0)
```

Out[249]:
```
(array([1, 3, 5, 8], dtype=int64),)
```

In [250]:
```
#now we will find the indexes where the values are odd.
```

In [251]:
```
np.where(x1%2==1)
```

Out[251]:
```
(array([0, 2, 4, 6, 7], dtype=int64),)
```

In [252]:
```
# searchsorted()-perform binary search in array
```

In [253]:
```
#we willnow find the index where the value 7 should be insterted
```

In [254]:
```
x1=np.array([1,7,3,4,2])
```

In [255]:
```
np.searchsorted(x1,7)
```

Out[255]:
```
5
```

In [256]:
```
#now we search right side.
```

In [257]:
```
np.searchsorted(x1,7,side="left")
```

Out[257]:
```
5
```

In [258]:
```
# how to insert multiple values.
```

In [259]:
```
x1=np.array([1,7,3,4,2])
```

In [260]:
```
np.searchsorted(x1,[2,4,6])
```

Out[260]:
```
array([1, 3, 5], dtype=int64)
```

numpy sorting array

In [261]:
```
#sort()- numpy n-d array object has a fun  which is called sort ()
# and this is sort a specified array
```

In [262]:
```
x1=np.array([1,7,3,4,2])
```

In [263]:
```
np.sort(x1)# ths method like a copy
```

Out[263]:
```
array([1, 2, 3, 4, 7])
```

In [264]:
```
# sort the array alphabetically.
```

In [265]:
```
x1=np.array(['a','e','c','w','g','s','q'])
```

In [266]:
```
x1.sort()
```

In [267]:
```
x1
```

Out[267]:
```
array(['a', 'c', 'e', 'g', 'q', 's', 'w'], dtype='<U1')
```

In [268]:
```
x1=np.array(['a','e','c','w','g','s','q'])
```

In [269]:
```
np.sort(x1)
```

Out[269]:
```
array(['a', 'c', 'e', 'g', 'q', 's', 'w'], dtype='<U1')
```

In [270]:
```
x1
```

Out[270]:
```
array(['a', 'e', 'c', 'w', 'g', 's', 'q'], dtype='<U1')
```

In [271]:
```

# sort the boolen value
```

In [272]:
```
x1=np.array(["False","True","True"])
```

In [273]:
```
np.sort(x1)
```

Out[273]:
```
array(['False', 'True', 'True'], dtype='<U5')
```

In [274]:
```
# sortthe 2-d array
```

In [275]:
```
x1=np.array([[1,5,3],[4,3,6]])
```

In [276]:
```
np.sort(x1)
```

Out[276]:
```
array([[1, 3, 5],
       [3, 4, 6]])
```

numpy filter array

gtting some element out of an existing array and creating a new array is called filtering

In [277]:
```
# create an array from the element on index 0to 2:
```

In [278]:
```
x1=np.array([1,2,3,4])
```

In [279]:
```
x2=[True,False,True,False]
```

In [280]:
```
x1[x2]
```

Out[280]:
```
array([1, 3])
```

In [281]:
```
# now will creating a fillter array
```

In [282]:
```
# that will return only values higher than 42.
```

In [283]:
```
x1=np.array([41,42,43,44])
```

In [284]:
```
x2=[]
```

In [285]:
```
for i in x1:
    if i>42:
        x2.append(True)

    else :
        x2.append(False)
```

In [286]:
```
x2
```

Out[286]:
```
[False, False, True, True]
```

In [287]:
```
x1[x2]
```

Out[287]:
```
array([43, 44])
```

In [288]:
```
# you can also filtter directly  array.
```

In [289]:
```
#that will return only values higher than 42.
```

In [290]:
```
x1=np.array([41,42,43,44])
```

In [291]:
```
x1>42
```

Out[291]:
```
array([False, False,  True,  True])
```

In [292]:
```
x1%2==0
```

Out[292]:
```
array([False,  True, False,  True])
```

numpy random no..

In [293]:
```
# that can not be predict  logically.
```

In [294]:
```
# we will generate a random no.. from 0 to 100
```

In [295]:
```
from numpy import random
```

In [296]:
```
x1=random(100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[296], line 1
----> 1 x1=random(100)

TypeError: 'module' object is not callable
```

In [297]:
```
x1=random.randint(100)
```

In [298]:
```
x1
```

Out[298]:
```
51
```

In [299]:
```
#you can also enetrate float() via rand () 0 to 1
```

In [300]:
```
x1=random.rand()
```

In [301]:
```
x1
```

Out[301]:
```
0.4030845633982648
```

In [302]:
```
#you can also generate random array.
```

In [303]:
```
#we will generate 1-D cntaining 5 random int from 0 to 100
```

In [304]:
```
 x1=random.randint(100,size=(5))
```

In [305]:
```
x1
```

Out[305]:
```
array([13, 59, 91,  0, 32])
```

we will generate 2-D with tree 3 row is as cntaining 5 random int from 0 to 100

In [306]:
```
 x1=random.randint(100,size=(3,5))
```

In [307]:
```
x1
```

Out[307]:
```
array([[ 0, 65, 40, 75, 22],
       [ 5, 31, 47, 97, 69],
       [94, 38, 32, 42, 92]])
```

In [308]:
```
#we will generate 1-D cntaining 5 random float
```

In [309]:
```
x1=random.rand(5)
```

In [310]:
```
x1
```

Out[310]:
```
array([0.11060362, 0.73933935, 0.40229079, 0.54873645, 0.99017427])
```

we will generate 2-D with tree 3 row is as cntaining 5 random int from 0 to 100

In [311]:
```
x1=random.rand(3,5)
```

In [312]:
```
x1
```

Out[312]:
```
array([[0.24479687, 0.08292316, 0.62108097, 0.77535091, 0.13119865],
       [0.31778486, 0.7882957 , 0.46009349, 0.21399335, 0.45736771],
       [0.32654685, 0.96756688, 0.73864213, 0.12319227, 0.26034713]])
```

In [313]:
```
  #you can also generate randm no from an array
```

In [314]:
```
#choice()
```

In [315]:
```
x1=random.choice([3,4,5,6,2,8,6,1])
```

In [316]:
```
x1
```

Out[316]:
```
8
```

In [317]:
```
  #you can also generate randm no from an 2-D array
```

In [318]:
```
x1=random.choice([3,4,5,6,2,8,6,1],size=(3,5))
```

In [319]:
```
x1
```

Out[319]:
```
array([[6, 5, 3, 3, 2],
       [4, 5, 5, 6, 8],
       [6, 4, 4, 4, 3]])
```

In [320]:
```
x1
```

Out[320]:
```
array([[6, 5, 3, 3, 2],
       [4, 5, 5, 6, 8],
       [6, 4, 4, 4, 3]])
```

numpy datadistribution

In [321]:
```
#list of all possible value and how to often each value occor
```

In [322]:
```
#such lists are important when working with static and datascience
```

In [323]:
```
#random distribu...-- probability function.
```

In [324]:
```
#now w will generate 1-d 100 values where values has to be 3,5,7,9
```

the probability for the value 3 is set to be 0.1 like that other value

In [325]:
```
#the some of all proba.. no should be "1"
```

In [326]:
```
x1=random.choice([3,5,7,9],p=[0.1,0.3,0.6,0.0],size=(100))
```

In [327]:
```
x1
```

Out[327]:
```
array([5, 5, 7, 7, 3, 7, 7, 7, 7, 5, 7, 7, 7, 7, 7, 7, 5, 7, 7, 3, 5, 7,
       5, 7, 7, 3, 7, 3, 7, 5, 7, 5, 7, 7, 3, 7, 7, 5, 7, 5, 3, 7, 5, 7,
       3, 7, 3, 3, 7, 7, 7, 7, 5, 5, 3, 7, 7, 7, 7, 7, 5, 7, 7, 7, 7, 5,
       5, 7, 7, 3, 7, 7, 7, 5, 7, 7, 5, 7, 5, 7, 7, 7, 7, 7, 5, 5, 5, 5,
       5, 7, 5, 5, 3, 7, 7, 3, 7, 7, 7, 3])
```

In [328]:
```
# now we will return 2-D with 3 rows each conta.. 5 values
```

In [329]:
```
x1=random.choice([3,5,7,9],p=[0.1,0.3,0.6,0.0],size=(3,5))
```

In [330]:
```
x1
```

Out[330]:
```
array([[5, 7, 7, 7, 5],
       [7, 7, 5, 5, 7],
       [7, 5, 5, 7, 3]])
```

numpy random permutation

refers to an arrangement of element like[3,2,1] is permutation of [1,2,3] and vice versa

In [331]:
```
# 1] shuffle 2]permutation.
```

In [332]:
```
#now we will randomly suffle elements  for the below array:
```

In [333]:
```
x1=np.array([1,2,3,4,5])
```

In [334]:
```
random.shuffle(x1) # shuffle make changes to the original array.
```

In [335]:
```
x1
```

Out[335]:
```
array([3, 4, 1, 5, 2])
```

In [336]:
```
x1
```

Out[336]:
```
array([3, 4, 1, 5, 2])
```

In [337]:
```
#now we will generate a permutation of elements for the below array.
```

In [338]:
```
x1=np.array([1,2,3,4,5])
```

In [339]:
```
random.permutation(x1) #the per.. method leaves the original array.
```

Out[339]:
```
array([2, 5, 3, 4, 1])
```

numpy normal distribution

In [340]:
```
#is same as gaussian distribution- very important.
```

In [341]:
```
#random.normal method loc(mean), scale(sd),size()
```

In [342]:
```

#we are generate a (r n d) of size 2,3
```

In [343]:
```
x1=random.normal(size=(2,3))
```

In [344]:
```
x1
```

Out[344]:
```
array([[ 0.27639054, -0.47776171, -0.23014079],
       [-0.39670203,  0.59518891,  0.41809229]])
```

In [345]:
```
#here we will generate (r n d) of size 2,3 with mean and sd of 2
```

In [346]:
```
x1=random.normal(loc=1,scale=2,size=(2,3))
```

In [347]:
```
x1
```

Out[347]:
```
array([[ 0.01341061, -0.0058592 ,  0.71608327],
       [ 5.82726498,  1.7545631 ,  4.33422112]])
```

numpy binomial

In [348]:
```
#is same as discreate distribution.
```

In [349]:
```
# param:-n(no of triails), p(proba..), size(shape-return)
```

In [350]:
```
# give 10 trial a coin which will generate 10 data points
```

In [351]:
```
x1=random.binomial(n=10,p=0.5,size=10)
```

In [352]:
```
x1
```

Out[352]:
```
array([6, 6, 4, 2, 7, 8, 3, 5, 7, 7])
```

In [353]:
```
# difference between normal and binomial.
```

In [354]:
```
# normal(continuous)       binomial(discrete)
```

numpy poisson distribution

In [355]:
```
# it estimate how many time an event can happen
```

In [356]:
```
#param:-  lam(no of occuranc of rate) size()
```

In [357]:
```
# generate a random 1*10 dist for the occurance 2
```

In [358]:
```
random.poisson(size=10,lam=2)
```

Out[358]:
```
array([4, 2, 4, 0, 2, 4, 4, 0, 1, 3])
```

In [359]:
```
random.poisson(size=10,lam=2)
```

Out[359]:
```
array([2, 0, 1, 5, 3, 2, 3, 2, 1, 0])
```

numpy U function

stands for universal fun and they are acually numpy fun that operates on the ndarray object

U fun also additional arguments like,where ,dtype and out

vectorization - converting the iterative statement into a vector based statement

In [360]:
```
# ex without u fun.
```

In [361]:
```
x=[1,2,3,4]
y=[4,5,6,7]
z=[]
for i ,j in zip(x,y):
    z.append(i+j)
print(z)
```

```
[5, 7, 9, 11]
```

In [362]:
```
# with u fun.
```

In [363]:
```
x=[1,2,3,4]
y=[4,5,6,7]
z=np.add(x,y)
```

In [364]:
```
z
```

Out[364]:
```
array([ 5,  7,  9, 11])
```

numpy create U fun...

In [365]:
```
# create your own ufan..
```

In [366]:
```
# arguments of frmpyfun(): fun,input,output
```

In [367]:
```
# create your own ufan.. for add
```

In [368]:
```
def myadd(x,y):
    return x+y

myadd =np.frompyfunc(myadd,2,1)
print(myadd([1,2,3,4],[5,6,7,8]))
```

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

In [369]:
```
# checking if tis function in ufun or not.
```

In [370]:
```
print(type(np.add))
```

```
<class 'numpy.ufunc'>
```

In [371]:
```
# concatenate ()
```

In [372]:
```
print(type(np.concatenate))
```

```
<class 'function'>
```

numpy simple arithematic operators

+,-,*,/

by using ufan additional arguments like ,where,dtype and out

here now we will use add fun..

In [373]:
```
x=np.array([10,11,12,13,14,15])
x1=np.array([20,21,22,23,24,25])
```

In [374]:
```
np.add(x,x1)#add
```

Out[374]:
```
array([30, 32, 34, 36, 38, 40])
```

In [375]:
```
np.subtract(x,x1)#sub
```

Out[375]:
```
array([-10, -10, -10, -10, -10, -10])
```

In [376]:
```
np.multiply(x,x1)#mul
```

Out[376]:
```
array([200, 231, 264, 299, 336, 375])
```

In [377]:
```
np.divide(x,x1)#div
```

Out[377]:
```
array([0.5       , 0.52380952, 0.54545455, 0.56521739, 0.58333333,
       0.6       ])
```

#power() function raises the value from the 1 st array to the power pf the values of the 2 nd aray and return the new array

In [378]:
```
x=np.array([10,22,37,40,50,60])
x1=np.array([3,4,5,6,3,2])
```

In [379]:
```
np.power(x,x1)#pow
```

Out[379]:
```
array([      1000,     234256,   69343957, -198967296,     125000,
            3600])
```

reminder mode() reminder() functction return the reminder of the 1st array corresponding to the 2 nd array and result in the array

In [380]:
```
np.mod(x,x1)#mod
```

Out[380]:
```
array([1, 2, 2, 4, 2, 0])
```

In [381]:
```
np.remainder(x,x1)#remainder
```

Out[381]:
```
array([1, 2, 2, 4, 2, 0])
```

quotient and mod(rem) the divmod () function return both quotiebt and mod

In [382]:
```
np.divmod(x,x1)#divmod
```

Out[382]:
```
(array([ 3,  5,  7,  6, 16, 30]), array([1, 2, 2, 4, 2, 0]))
```

numpy rounding

#rounding decimals-: 1]truction 2]fix 3]rounding 4]floor 5]ceil

In [383]:
```
x=np.trunc([-3.1666,3.6667])#remove the decimal.
```

In [384]:
```
x
```

Out[384]:
```
array([-3.,  3.])
```

In [385]:
```
np.fix([-3.1666,3.6667])# as trunc...
```

Out[385]:
```
array([-3.,  3.])
```

rounding : the around() fuction preceding digit or decimal by nearest to 1: if n>5 or n<5 =0

In [386]:
```
np.around([3.1144,2])
```

Out[386]:
```
array([3., 2.])
```

In [387]:
```
np.around([3.766,2])
```

Out[387]:
```
array([4., 2.])
```

floor() - round off decimal to the lower integer

In [388]:
```
x=np.floor([-3.1666,3.6667])
```

In [389]:
```
x
```

Out[389]:
```
array([-4.,  3.])
```

ceil() -upper integer

In [390]:
```
x=np.ceil([-3.1666,3.6667])
```

In [391]:
```
x
```

Out[391]:
```
array([-3.,  4.])
```

numpy summation

add is done between two argument where as summation happens over n element.

In [392]:
```
#add the 2 array
```

In [393]:
```
x1=np.array([1,2,3,4])
x2=np.array([1,2,3,4])
np.add(x1,x2)
```

Out[393]:
```
array([2, 4, 6, 8])
```

In [394]:
```
#sum the values in two array..
```

In [395]:
```
x1=np.array([1,2,3,4])
x2=np.array([1,2,3,4])
np.sum([x1,x2])
```

Out[395]:
```
20
```

In [396]:
```
# summation over as axis..
```

In [397]:
```
x1=np.array([1,2,3,4])
x2=np.array([1,2,3,4])
np.sum([x1,x2],axis=1)
```

Out[397]:
```
array([10, 10])
```

In [398]:
```
# cumulative sum: means partially adding the array..
```

In [399]:
```
x1=np.array([1,2,3,4])
np.cumsum(x1)
```

Out[399]:
```
array([ 1,  3,  6, 10])
```

numpy product

products: prod() function

In [400]:
```
#here we will find the product of the array element of the below array
```

In [401]:
```
x1=np.array([1,2,3,4])
np.prod(x1)
```

Out[401]:
```
24
```

In [402]:
```
#here we will find the product of element in 2 differet array:
```

In [403]:
```
x1=np.array([1,2,3,4])
x2=np.array([7,6,5,4])
```

In [404]:
```
np.prod([x1,x2])
```

Out[404]:
```
20160
```

In [405]:
```
x1=np.array([1,2,3,4])
x2=np.array([4,5,6,7])
```

In [406]:
```
np.prod([x1,x2])
```

Out[406]:
```
20160
```

In [407]:
```
#product over axis:
```

In [408]:
```
np.prod([x1,x2],axis=1)
```

Out[408]:
```
array([ 24, 840])
```

In [409]:
```
#cumulative product:
```

In [410]:
```
np.cumsum([x1,x2])
```

Out[410]:
```
array([ 1,  3,  6, 10, 14, 19, 25, 32])
```

In [411]:
```
np.cumsum([x1,x2],axis=1)
```

Out[411]:
```
array([[ 1,  3,  6, 10],
       [ 4,  9, 15, 22]])
```

numpy differences

In [412]:
```
#use-: deff() function
```

In [413]:
```
x1=np.array([1,2,3,4])
np.diff(x1)
```

Out[413]:
```
array([1, 1, 1])
```

In [414]:
```
x1=np.array([1,5,3,8])
np.diff(x1)
```

Out[414]:
```
array([ 4, -2,  5])
```

In [415]:
```
x1=np.array([1,2,3,4])
x2=np.array([4,5,6,7])
np.diff([x1,x2])
```

Out[415]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

In [416]:
```
x1=np.array([1,2,3,4])
x2=np.array([4,5,6,7])
np.diff([x1,x2],axis=1)
```

Out[416]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

numpy GCD_HCF

GCD(GRETEST COMMON DNOMINATOR),ALSO KNOWN AS

HCF(HIGHEST COMMON FACTOR)

In [417]:
```
X1=5
X2=7
np.gcd(X1,X2)
```

Out[417]:
```
1
```

Ans will be 1 because that is the highest nimber and both number can divided by....

In [418]:
```
x1=6
x2=9
np.gcd(x1,x2)
```

Out[418]:
```
3
```

In [419]:
```
#finding the gcd in an array..
```

In [420]:
```
x1=np.array([20,8,16,36,44])
```

In [421]:
```
np.gcd.reduce(x1)
```

Out[421]:
```
4
```

it will return 4 because 4 in the higest number of all values that can be divided in bet array..

numpy Trigonometric

numpy provides the ufuncs line sin() cos() and tan() that values in radians and produce the corresponding sin, cos, tan values

In [422]:
```
x1=np.sin(np.pi/2)
x1
```

Out[422]:
```
1.0
```

In [423]:
```
#we will now find sin value of an array..
```

In [424]:
```
x1=np.array([np.pi/2,np.pi/3,np.pi/4,np.pi/5])
x1
```

Out[424]:
```
array([1.57079633, 1.04719755, 0.78539816, 0.62831853])
```

In [425]:
```
np.sin(x1)
```

Out[425]:
```
array([1.        , 0.8660254 , 0.70710678, 0.58778525])
```

In [426]:
```
# convert  degree into radians
```

by defsult all of the function tri.. function taies as parameter's.

In [427]:
```
#radians value are pi/180 degree value
```

In [428]:
```
x1=np.array([90,180,270,360])
np.deg2rad(x1)
```

Out[428]:
```
array([1.57079633, 3.14159265, 4.71238898, 6.28318531])
```

In [429]:
```
#her we will convert rad to deg...
```

In [430]:
```
x1=np.sin(np.pi/2)
x1
```

Out[430]:
```
1.0
```

In [431]:
```
x1=np.sin([np.pi/2,np.pi,1.5*np.pi,2*np.pi])
```

In [432]:
```
np.rad2deg(x1)
```

Out[432]:
```
array([ 5.72957795e+01,  7.01670930e-15, -5.72957795e+01, -1.40334186e-14])
```

numpy setoperation

collections of unique elements

In [433]:
```
#here we will convert the array with reoeted element to a set
```

In [434]:
```
x1=np.array([11,1,1,3,2,4,5,3,2,6,4,7,5])
```

In [435]:
```
np.unique(x1)
```

Out[435]:
```
array([ 1,  2,  3,  4,  5,  6,  7, 11])
```

In [436]:
```
# to find unique value of 2 1D array we will use unuon1d() method
```

In [437]:
```
x1=([1,2,3,4])
x2=([3,4,5,6])
```

In [438]:
```
np.union1d(x1,x2)
```

Out[438]:
```
array([1, 2, 3, 4, 5, 6])
```

to find the only value tat are present in both array we will use interset1d() method

In [439]:
```
np.intersect1d(x1,x2)
```

Out[439]:
```
array([3, 4])
```

to find only values that are in 1st set and not present in the 2nd set: use setfdiffid()

In [440]:
```
np.setdiff1d(x1,x2)
```

Out[440]:
```
array([1, 2])
```

to find the only values that are not present in both the sets use setxorld()

In [441]:
```
np.setxor1d(x1,x2)
```

Out[441]:
```
array([1, 2, 5, 6])
```

In [ ]:
```

```