In [4]:
```
#IMPORTING THE numpy array and make them numpy array
import numpy as np
a = np.array([12,23,45])
b = np.array([11,56,78])
print(a)
print(b)
```
```
[12 23 45]
[11 56 78]
```

In [5]:
```
#call the b array
b
```
Out[5]:
```
array([11, 56, 78])
```

In [8]:
```
#changing vlaue of the value of b array at the idex position 1
b[1]=19
```

In [9]:
```
#then print the b array
print(b)
```
```
[11 19 78]
```

In [12]:
```
#creat the new array with the with help of 1d array
c = np.array([[a],[b]])
print(c)
```
```
[[[12 23 45]]

 [[11 19 78]]]
```

In [14]:
```
#change the index value of a 1-d array to the 23 at position 2
a[2]=23
print(a)
```
```
[12 23 23]
```

In [15]:
```
print(c)
```
```
[[[12 23 45]]

 [[11 19 78]]]
```

In [16]:
```
d = np.array([22,76,45])
e = np.array([59,21,98])
print(d)
print(e)
```
```
[22 76 45]
[59 21 98]
```

In [17]:
```
d[2]=23
print(d)
```
```
[22 76 23]
```

In [35]:
```
f = np.array([[d],[e]])
print(f)
```
```
[[[22 76 23]]

 [[59 21 98]]]
```

In [28]:
```
#ccalling the 2nd row of the f array
f[1]
```
Out[28]:
```
array([[59, 21, 98]])
```

In [29]:
```
#calling the 1st row of the f array
f[0]
```
Out[29]:
```
array([[22, 76, 23]])
```

In [31]:
```
#calling the element of a 1-d array from the elemenet 1st position to 2nd and here 2nd excuded
a[1:2]
```
Out[31]:
```
array([23])
```

In [32]:
```
#calling the element of a 1-d array from the elemenet 0th position to 2nd and here 2nd excuded
a[0:2]
```
Out[32]:
```
array([12, 23])
```

In [40]:
```
#calling the element of a 1-d array from the elemenet 0th position to last position of the a array
a[0:]
a[0:3]
```
Out[40]:
```
array([12, 23, 23])
```

In [47]:
```
#this is all about the masking in numpy arrays it will gives us by default arrays.
print('numpy array initialization')
```
```
numpy array initialization
```

In [49]:
```
#here we paassed the tuple in which we have to spceify the row and column(3,4)
j = np.zeros((3,4))
print(j)
```
```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [51]:
```
#here we have to use the arange in that we have to pass the (start,end,interval)
k = np.arange(20,25,1)
print(k)
```
```
[20 21 22 23 24]
```

In [55]:
```
#here we have to use the linspace in that paassing argument (start,stop,how much points in that like below)
l = np.linspace(1,5,5)
print(l)
```
```
[1. 2. 3. 4. 5.]
```

In [58]:
```
#if we want the same number in all row and column then we need to use the full with the arument i.e.((row,colur
m= np.full((3,3),5)
print(m)
```
```
[[5 5 5]
 [5 5 5]
 [5 5 5]]
```

In [59]:
```
#this will print any random values in the row and column(row,column)
n= np.random.random((3,4))
print(n)
```
```
[[0.01895143 0.46790746 0.29178308 0.89767245]
 [0.56564337 0.72539292 0.18306492 0.26201124]
 [0.45594671 0.57263548 0.80163887 0.35893378]]
```

In [64]:
```
#how can we access the array menas changes be done in the array according to the command
#we got the shape of the array by using the method that is shape which can help to get tuple (row,column) form
o = np.array([[23,76,90],[56,75,78]])
print(o)
print(o.shape)
```
```
[[23 76 90]
 [56 75 78]]
(2, 3)
```

In [65]:
```
#use of the shape function
#we change the tuple that we got by using shape function last time
#we can access the individual element of the tuple.
```

In [67]:
```
o.shape = (3,2)
```

In [71]:
```
print(o)
print(o.shape[0])
```
```
[[23 76]
 [90 56]
 [75 78]]
3
```

In [74]:
```
#size
#n.arange(number of points needed)
#will place the numbers zero to till that number
#but the size will count the all the element which will come inbetween 0 to 10 that how it works
p= np.arange(10)
print(p)
print(p.size)
```
```
[0 1 2 3 4 5 6 7 8 9]
10
```

In [75]:
```
#ndim
#ndim returns of the dimnesions of the array it could be 1d,2d or upto nd..
print(o.ndim)
```
```
2
```

In [76]:
```
#it will gives the what type of array it is because array should be homogeous not hetrogeous.
print(p.dtype)
```
```
int32
```

In [89]:
```
#we can do the mathematical calculations with numpy arrays
#like np.subtarct,np.muliply and np.log,np.exp,np.divide,np.sqrt and so on..
q = np.array([5,10])
np.sum(q)
print(np.sum(q))
r= np.array([3,7])
np.subtract(q,r)
print(np.subtract(q,r))
u = np.divide(q,r)
print(u)
r =np.multiply(q,r)
print(r)
#it works like the matrix operations
```
```
15
[2 3]
[1.66666667 1.42857143]
[15 70]
```

In [87]:
```
#sum method with the axis
#in this kind of sum method axis taken zero afterwarrds they sum with the corresponding element in the vertica
#for axis =1 voice vesa.
s = np.sum([[5,1],[6,4]], axis = 0)
print(s)
t = np.sum([[9,6],[23,4]], axis = 1)
print(t)
```
```
[11  5]
[15 27]
```

In [106…
```
# equal function it works to compare  the coresponding element of the array
np.equal(q,r)
print(np.equal(q,r))
# equal function it also works to compare the two array with the comparision of an element it will return in t
np.array_equal(q,r)
print(np.array_equal(q,r))
```
```
[False False]
False
```

In [121…
```
#aggregate function it always work on single array
v = np.array([23,56,98])
print(v)
print(np.min(v))  #min value
print(np.max(v))   #max value
print(np.mean(v))  #mean value
print(np.sum(v))    #sum of vaalue
print(np.std(v))   #standard deviation its nothing but how much the value is differ from the mean value
print(np.median(v)) #median
print(np.corrcoef(v))#ML term
```
```
[23 56 98]
23
98
59.0
177
30.692018506445613
56.0
1.0
```

In [131…
```
#Concept of baroadcasting
#it can happpen with any operation here we taken addition for understading the concept. of it.
#when the one array is three diamensional is added or subtracted with the one dimensional array that time its(
#to get three diamensional array after that it gets merge
#2d array manage itself with the diamensional of the first array then only the addition or subraction can done
#manage the first array.

w= np.array([[1,2,3],[4,4,5]])
wa = np.array([3,4,5])
print(np.sum([w,wa]))
```
```
[[ 4  6  8]
 [ 7  8 10]]
```

In [132…
```
#indexing & sclicing in the numpy array
#here we having the three dimensional array in we want to sclice or index it
i=np.array([[23,76,98],[45,91,34],[12,32,93]])
print(i)
```
```
[[23 76 98]
 [45 91 34]
 [12 32 93]]
```

In [139…
```
#in the 3-d np array here we are calling the 1st row of the array
print(i[0])
```
```
[23 76 98]
```

In [24]:
```
#all_based_on_zero_based_indexing
#when we call for the 2nd dimensional array that time de use the [3,4] But here we having the three dimensiona
#here we want to sclice like [:0, :]
#in that first use for the row and column similarly for the three diamensional array
#here,we having more row and column so three need of indexing themself is necesarrry so that is we have taken
#column
import numpy as np
i=np.array([[23,76,98],[45,91,34],[12,32,93]])
print(i)
#first part of [ : , : ] work for the row 2nd for the column.
print(i[:1,:])
#if we write it down as below for column then we willng get the 1 column   to 0th column and here 1st column i
#so we are getting the zeroth element of zeroth column in the zeroth row that we called first argument.
print(i[ :1,:1])
#similarly in that we are getting the 2nd and first element of column in thhe zeorth row
print(i[:1,1:])
print(i[:2,1:])
#it means hqving the all rows but the column avalible from then 1 to zeroth here 1st coumn is included.and rem
print(i[ : , 1: ])
#here we wanting the 2nd row of the 3x3 array and we wanted the column upto the 3rd element according to index
print(i[2:,:3])
#we want till the 2nd row that why first two row(0,1) come into picture and we had also mention that we want t
print(i[: 2,2:])
#here we wanted the 2nd row and 2nd column upto the end
print(i[2:, 2: ])
#here we will get the entire arrray
print(i[0: ,0:])
#here we wnted the 1st row and coulmn 1st upto the end
print(i[1:,1:])
#here we wanted the 2nd row and 1st collumn till the end
print(i[2:,1:])
#here we wanted the row from the 1st to zeroth and 2 nd will be excluded and 1st column will be excluded.
print(i[:2,:1])
```
```
[[23 76 98]
 [45 91 34]
 [12 32 93]]
[[23 76 98]]
[[23]]
[[76 98]]
[[76 98]
 [91 34]]
[[76 98]
 [91 34]
 [32 93]]
[[12 32 93]]
[[98]
 [34]]
[[93]]
[[23 76 98]
 [45 91 34]
 [12 32 93]]
[[91 34]
 [32 93]]
[[32 93]]
[[23]
 [45]]
```

In [34]:
```
#array manupulation
#this the way we can can make the horizontally concatenation with the two arrays.
aa= np.array([23,98,76])
bb =np.array([45,67,23])
ss =np.concatenate([aa,bb],axis=0)
print(ss)
#if we want the veritically(column wise) concatenation that time we have two multidimensional array at conditi
#column wise stacking
aa1=np.array([[22,78,56],[28,67,45]])
bb1=np.array([[21,98,90],[29,90,78]])
ss1 = np.concatenate([aa1,bb1],axis=1)
print(ss1)
#if we want the horizontally concatenation means row wise that time we need to use the axis=0
#row wise stacking
ss2 = np.concatenate([aa1,bb1],axis=0)
print(ss2)
```
```
[23 98 76 45 67 23]
[[22 78 56 21 98 90]
 [28 67 45 29 90 78]]
[[22 78 56]
 [28 67 45]
 [21 98 90]
 [29 90 78]]
```

In [41]:
```
#v-stack (vertical stacking) and h-stack (horizontal stacking) in numpy array
#stacking- same thing can be concatanted with the small diffrence.
#in stacking we required 1-d array
cc = np.array([23,56,78])
print(cc)
dd = np.array([32,87,55])
print(dd)
#in that we are stacking the two rows of the arrays
ee =np.stack((cc,dd),axis =0)
print(ee)
#similarly, we are stacking the two column of the arrays
ff = np.stack((cc,dd), axis=1)
print(ff)
```
```
[23 56 78]
[32 87 55]
[[23 56 78]
 [32 87 55]]
[[23 32]
 [56 87]
 [78 55]]
```

In [51]:
```
#v-stack (vertical stacking) and h-stack (horizontal stacking) in numpy array
s1 =np.array([[23,87,90],[20,30,56]])
s2 =np.array([[33,67,90],[12,34,89]])
print('stack of a nd b')
print(np.stack((s1,s2)))
print('marix a')
print(s1)
print('matrix b')
print(s2)
```
```
stack of a nd b
[[[23 87 90]
  [20 30 56]]

 [[33 67 90]
  [12 34 89]]]
marix a
[[23 87 90]
 [20 30 56]]
matrix b
[[33 67 90]
 [12 34 89]]
```

In [59]:
```
#horizontal_stacking
print('horizontal staking')
print(np.hstack((s1,s2)))
print('horizontal concatenate')
print(np.concatenate((s1,s2),axis=0))
print('vertical stack')
print(np.vstack((s1,s2)))
print('vertical cancatenate')
print(np.concatenate((s1,s2),axis=1))
print('column stack') #not used generally in data science
print(np.column_stack((s1,s2)))
```
```
horizontal staking
[[23 87 90 33 67 90]
 [20 30 56 12 34 89]]
horizontal concatenate
[[23 87 90]
 [20 30 56]
 [33 67 90]
 [12 34 89]]
vertical stack
[[23 87 90]
 [20 30 56]
 [33 67 90]
 [12 34 89]]
vertical cancatenate
[[23 87 90 33 67 90]
 [20 30 56 12 34 89]]
column stack
[[23 87 90 33 67 90]
 [20 30 56 12 34 89]]
```

In [92]:
```
#split_function
#syntax
#np.split(array,index,axis) here array is any numpy array ,index =int or list
print(s1)
print('split appy on it')#we have pass the argument according to requirement
print('this is about the row based')
print(np.split(s1,2,axis=0))
print('this is about the column based')
print(np.split(s1,3,axis=1))
```
```
[[23 87 90]
 [20 30 56]]
split appy on it
this is about the row based
[array([[23, 87, 90]]), array([[20, 30, 56]])]
this is about the column based
[array([[23],
       [20]]), array([[87],
       [30]]), array([[90],
       [56]])]
```

In [87]:
```
print("if we use the list instaed of integer then let's see")
print('this will be for the row wise')
print(np.split(s1,[2,1],axis=0))

print('lets ckeck the the spliting of the each array')
```
```
if we use the list instaed of integer then let's see
[array([[23, 87, 90],
       [20, 30, 56]]), array([], shape=(0, 3), dtype=int32), array([[20, 30, 56]])]
lets ckeck the the spliting of the each array
```

In [90]:
```
print('part 1')
print(s1[:1])
```
```
part 1
[[23 87 90]]
```

In [114…
```
print('part 2')
print(s1[1:2])
```
```
part 2
[[20 30 56]]
```

In [88]:
```
print('part 3')
print(s1[2:])
```
```
part 3
[]
```

In [135…
```
j =np.array([[23,21,76],[98,67,54]])
print(j)
print('for the column wise we need to take the axis= 1')
k =np.split(j,[1,2],axis=1)
print(k)
print('how the spliting works lets see'),
print('here everything done for the column because its column wise spliting')
```
```
[[23 21 76]
 [98 67 54]]
for the column wise we need to take the axis= 1
[array([[23],
       [98]]), array([[21],
       [67]]), array([[76],
       [54]])]
how the spliting works lets see
here everything done for the column because its column wise spliting
```

In [134…
```
j[:,:1]
```
Out[134…
```
array([[23],
       [98]])
```

In [136…
```
j[:,1:2]
```
Out[136…
```
array([[21],
       [67]])
```

In [137…
```
j[:,2:]
```
Out[137…
```
array([[76],
       [54]])
```

In [ ]: