array - an arrangement of objects , pictures , numbers etc. in columns and rows is called an array.

NumPy array is a central structure of the NumPy library. It is an n-dimensional array object containing rows and columns.

In [2]:
```
import numpy as np
np.arange(1,10)
#it is a static data (the data is fixed)
```

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

In [4]:
```
np.arange(1,51,2)  #odd numbers
```

Out[4]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49])
```

In [5]:
```
np.arange(0,51,2)  #even numbers
```

Out[5]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38, 40, 42, 44, 46, 48, 50])
```

In [11]:
```
np.arange(10,0,-1) # according to the actual rule , first index should always
 have a smaller number compared to the second index,
                    #but if we really want to do it in this way the accuracy is
reduced .(reduced efficiency means that python will take more time)
```

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

In [7]:
```
np.random.randint(1,10)  # we are giving range and a random number in the rang
e is printed
```

Out[7]:
```
5
```

In [9]:
```
np.random.randint(1,50,10) # we are generating 10 random numbers between the g
iven range
#duplicate values are allowed in the repetition
```

Out[9]:
```
array([ 1, 26,  5, 15, 16, 23, 40, 26, 45,  7])
```

In [10]:
```
np.random.randint(1,20,30) # here we need 30 random integers between the given
range , hence the numbers will obviously get repeated .
#this is actually known as manipulation of values - repetitive values
```

Out[10]:
```
array([ 8,  9, 19,  3, 13,  3,  9, 14, 10, 17,  6,  1, 19,  6, 18,  7,  7,
       16, 17, 11,  8, 11, 11, 14,  1, 12, 17,  3, 11,  6])
```

In [19]:
```
np.random.rand(2) # generates positive numbers between 0 and 1 .
```

Out[19]:
```
array([0.78613689, 0.55526455])
```

In [20]:
```
np.random.randn(10)    # here n means normalization
                   # here we are not forcing python to generate only positive
 numbers,we can generate negative numbers also ,but the numbers are always clo
ser to 0 .
                   # generally the range of randn is not defined(frequently we
get value from -4 to +4)
```

Out[20]:
```
array([-0.46519513,  2.23637917, -1.07709817,  0.52593606,  0.4327891 ,
       -0.10508033, -0.04246029, -1.971156  ,  2.70803269, -0.03806371])
```

In [ ]:
```
# arrays are actually of two types 1)create(where we create an array)and autom
atic methods.
```

In [23]:
```
a=np.array([[2,3,4],[6,7,8],[5,7,9]])
a
```

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

In [24]:
```
a.ndim   #ndim means number of dimensions
```

Out[24]:
```
2
```

In [ ]:
```
# 1D array is known as Vector
```

In [25]:
```
a.dtype #to find the data type
```

Out[25]:
```
dtype('int32')
```

In [26]:
```
a.shape  #to find the shape of the array
```

Out[26]:
```
(3, 3)
```

In [27]:
```
a.size   #the total number of elements in the array.
```

Out[27]:
```
9
```

In [28]:
```
a.itemsize #number of bytes aquired by the array
```

Out[28]:
```
4
```

In [30]:
```
a=np.array([[2,3],[6,7,8],[5,7,9]])  #here the dimensions are not equal
a
```

Out[30]:
```
array([list([2, 3]), list([6, 7, 8]), list([5, 7, 9])], dtype=object)
```

In [32]:
```
np.zeros(2) #python will create a one dimensional array where it will create o
nly zeros
```

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

In [34]:
```
np.zeros((2,3))   #python will create 2 rows and 3 coloumns
```

Out[34]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [35]:
```
np.zeros((3,2,3))        #this syntax is to create 3 arrays with 2 rows and 3
 columns each
```

Out[35]:
```
array([[[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]],

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

A standard Normal Distribution has mean 0 and standard deviation 1

In [3]:
```
np.random.randn(5)
```

Out[3]:
```
array([-1.86900415,  0.45090631, -1.02592415, -1.70463674,  0.20080733])
```

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

In [6]:
```
import numpy as np
a=np.arange(100,200,5)
a
```

Out[6]:
```
array([100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160,
       165, 170, 175, 180, 185, 190, 195])
```

Create a 5X4 integer array from a range between 100 to 200 such that the difference between each element is 5.

In [7]:
```
a= np.arange(100, 200, 5).reshape((5,4))
a
```

Out[7]:
```
array([[100, 105, 110, 115],
       [120, 125, 130, 135],
       [140, 145, 150, 155],
       [160, 165, 170, 175],
       [180, 185, 190, 195]])
```

In [9]:
```
np.zeros((3,3))
```

Out[9]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [18]:
```
np.identity(4)
```

Out[18]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [15]:
```
import numpy as np
m=np.ones((3,3))
n=np.ones((3,3))
a=np.identity(3)
print(a)
```

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

Print an array of size M*N in which all the elements are 1's .

In [1]:
```
import numpy as np
M = int(input('enter the value of M :- '))
N = int(input('enter the value of N :- '))
a = np.ones((M,N))
a
```

```
enter the value of M :- 2
enter the value of N :- 5
```

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

Print an array of size M*N in which all the elements are 0's .

In [2]:
```
M = int(input('enter the value of M :- '))
N = int(input('enter the value of N :- '))
a = np.zeros((M,N))
a
```

```
enter the value of M :- 5
enter the value of N :- 6
```

Out[2]:
```
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.],
       [0., 0., 0., 0., 0., 0.]])
```

Give a comma seperated list of nine integers , convert this list into a 3*3 Numpy array .

In [6]:
```
lst = list(range(1,10))
#solution 1
a=np.reshape(lst,(3,3))
a
#solution 2
a=np.array(lst).reshape(3,3)
a
```

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

Print an array of size M*N with its main diagonal elements as 1's and 0's everywhere else .

In [5]:
```
import numpy as np
M = int(input('enter the number of rows :- '))
N = int(input('enter the number of coloumns :- '))
a=np.identity(5)     #(always returns a square matrix(rows = coloumns))
#a=np.eye(n,m)   (where rows are not equal to coloumns)
a
```

```
enter the number of rows :- 5
enter the number of coloumns :- 6
```

Out[5]:
```
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 [12]:
```
#another method with np.eye is the k(which changes the diagonal up and down de
pending upon the value)
a = np.eye(5,5,k=0)
a
```

Out[12]:
```
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.]])
```

Write a numpy program to create a 5X5 array with random normal values with 3 decimal places .

In [18]:
```
#solution1
nparray2=np.random.randn(5,5)
nparray2.round(3)
```

Out[18]:
```
array([[-0.634,  0.143, -2.677,  0.724, -0.064],
       [ 0.752, -0.915, -0.19 ,  0.284, -0.795],
       [-0.102,  1.039, -0.348, -0.107, -0.867],
       [ 0.888,  0.764, -0.221,  1.081, -0.14 ],
       [-0.888,  1.288, -0.485, -1.265,  0.54 ]])
```

In [19]:
```
#solution2
x = np.random.random((5,5))
np.set_printoptions(precision=3, suppress=True)
print (x)
```

```
[[0.812 0.392 0.704 0.272 0.023]
 [0.894 0.895 0.239 0.102 0.173]
 [0.084 0.386 0.672 0.722 0.739]
 [0.54  0.911 0.498 0.506 0.565]
 [0.356 0.185 0.208 0.826 0.167]]
```

create a numpy array with 5 equal spaced elements from 100-200.

In [20]:
```
nparray3 = np.linspace(100,200,5)
nparray3
```

Out[20]:
```
array([100., 125., 150., 175., 200.])
```

In [21]:
```
np.arange(100,201,25)
```

Out[21]:
```
array([100, 125, 150, 175, 200])
```

Create a 1D array of weights of 10 students and retrieve all the weights greater than 68

weights = [74.2, 85, 74, 67.9, 52, 70.5, 86, 51.8, 64, 82]

In [22]:
```
#solution 1
weights = [74.2, 85, 74, 67.9, 52, 70.5, 86, 51.8, 64, 82]

weight = np.array(weights)
[i for i in weights if i>68]
```

Out[22]:
```
[74.2, 85, 74, 70.5, 86, 82]
```

In [23]:
```
#solution 2
weights = [74.2, 85, 74, 67.9, 52, 70.5, 86, 51.8, 64, 82]
a=np.array(weights)
print(a[a>68])
```

```
[74.2 85.  74.  70.5 86.  82. ]
```

Adding columns and rows example

In [24]:
```
A = [[1, 4, 8], [5, 7, 3], [9, 14, 2]]
b = np.array(A)
np.sum(b,axis=0)
```

Out[24]:
```
array([15, 25, 13])
```

Stacking Operations

In [ ]:
```
#stacking means arranging the data
#it is used to convert unstructured data into a structured data
```

In [2]:
```
import numpy as np
abc = np.arange(5)
abc
```

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

In [3]:
```
qwe = np.arange(11,16)
qwe
```

Out[3]:
```
array([11, 12, 13, 14, 15])
```

In [4]:
```
#now we horizontally stack both the arrays
np.hstack([abc,qwe])
```

Out[4]:
```
array([ 0,  1,  2,  3,  4, 11, 12, 13, 14, 15])
```

In [10]:
```
#lets do this operation with 2D arrays
abc2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
abc2
```

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

In [7]:
```
qwe2 = np.array([[11,22,33],[44,45,66]])
qwe2
```

Out[7]:
```
array([[11, 22, 33],
       [44, 45, 66]])
```

In [8]:
```
np.hstack([abc2,qwe2])
```

Out[8]:
```
array([[ 1,  2,  3, 11, 22, 33],
       [ 4,  5,  6, 44, 45, 66]])
```

In [11]:
```
#now we vertically stack both the arrays
np.vstack([abc2,qwe2])
```

Out[11]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [11, 22, 33],
       [44, 45, 66]])
```

Depth stack

In [ ]:
```
#depth means giving a 3 dimensional effect to an array
#to convert a 2D data into a 3D data, we must add a z axis.
#the rows and coloumns should be equal to convert 2D to 3D.
```

In [13]:
```
abc2 = np.array([[1,2,3],[4,5,6],[5,6,7]])
abc2
```

Out[13]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [5, 6, 7]])
```

In [14]:
```
qwe2 = np.array([[11,22,33],[44,45,66],[44,55,66]])
qwe2
```

Out[14]:
```
array([[11, 22, 33],
       [44, 45, 66],
       [44, 55, 66]])
```

In [15]:
```
np.dstack([abc2,qwe2])
```

Out[15]:
```
array([[[ 1, 11],
        [ 2, 22],
        [ 3, 33]],

       [[ 4, 44],
        [ 5, 45],
        [ 6, 66]],

       [[ 5, 44],
        [ 6, 55],
        [ 7, 66]]])
```

Column stack

In [17]:
```
a=np.arange(1,5)
a
```

Out[17]:
```
array([1, 2, 3, 4])
```

In [18]:
```
b=np.arange(11,15)
b
```

Out[18]:
```
array([11, 12, 13, 14])
```

In [20]:
```
np.column_stack((a,b))
```

Out[20]:
```
array([[ 1, 11],
       [ 2, 12],
       [ 3, 13],
       [ 4, 14]])
```

In [ ]:
```
#axis 0 represents rows and axis 1 represents columns.
```

Splitting arrays

In [21]:
```
a=np.arange(10)
a
```

Out[21]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [22]:
```
np.split(a,2)   #splitting into two equal parts
```

Out[22]:
```
[array([0, 1, 2, 3, 4]), array([5, 6, 7, 8, 9])]
```

In [23]:
```
abc=np.array([[1,2,3],[4,5,6],[7,8,9]])
abc
```

Out[23]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [24]:
```
np.split(abc,3)
```

Out[24]:
```
[array([[1, 2, 3]]), array([[4, 5, 6]]), array([[7, 8, 9]])]
```

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

Splitting arrays has two parts.....1)equal number of elements.....2)no equal number of elements

In [2]:
```
import numpy as np
a=np.arange(10)
a
```

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

In [3]:
```
np.split(a,2)          #splitted into two equal arrays....splitting actually d
ivides into equal parts
```

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

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

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

In [7]:
```
np.split(abc,3)
```

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

In [8]:
```
q=np.zeros((3,4))
q
```

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

In [ ]:
```
np.hsplit()    #arranges the data column wise
```

In [9]:
```
np.hsplit(q,2)
```

Out[9]:
```
[array([[0., 0.],
        [0., 0.],
        [0., 0.]]),
 array([[0., 0.],
        [0., 0.],
        [0., 0.]])]
```

In [ ]:
```
np.vsplit()     #arranges the data row wise
```

In [12]:
```
w=np.ones((4,5))
w
```

Out[12]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [13]:
```
np.split(w,2)
```

Out[13]:
```
[array([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]),
 array([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])]
```

In [ ]:
```
#python will not allow negative splitting of numbers
#axis=0 represents row
#axis=1 represents column
#for loop is considered as a slow iterator
#for loop doesnt work with numbers and boolean.
```

Examples for splitting rows and columns

In [4]:
```
import numpy as np
abc = np.random.randn(3,4)
abc
```

Out[4]:
```
array([[-0.77557254,  1.11413022,  1.17641161,  0.13325485],
       [ 0.16702884, -0.07054781,  0.70677681, -1.85221415],
       [ 0.98311902,  1.80458056, -0.85615842,  0.26021801]])
```

In [16]:
```
for data in abc:
    print(data[-2])    #this process is used for splitting coloumns
```

```
0.20979294992662578
-0.18355219263536793
1.2574426988732996
```

In [6]:
```
for data in abc[-1]:    #this process is used for splitting rows
    print(data)
```

```
0.9831190203984321
1.8045805638161267
-0.8561584240578797
0.2602180051733887
```

indexing of arrays

In [8]:
```
weight = np.array([(23,44,67),(67,43,54)])
weight
```

Out[8]:
```
array([[23, 44, 67],
       [67, 43, 54]])
```

In [11]:
```
weight[1][1]  #retrieving the number by giving the rows and columns
```

Out[11]:
```
43
```

In [12]:
```
weight[0,:]   #retrieving all the elements in the first row
```

Out[12]:
```
array([23, 44, 67])
```

Practice session

In [ ]:
```
#comparing the size of lists and array
```

In [9]:
```
import sys
```

In [10]:
```
abc = [1,2,3,4,5]
abc
```

Out[10]:
```
[1, 2, 3, 4, 5]
```

In [12]:
```
sys.getsizeof('abc')
```

Out[12]:
```
52
```

In [2]:
```
import numpy as np
ded = np.array([1,2,3,4,5])
ded
```

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

In [13]:
```
sys.getsizeof('ded')
```

Out[13]:
```
52
```

In [6]:
```
ded.size
```

Out[6]:
```
5
```

In [ ]:
```
sys.getsizeof
```

iterate numpy array using nditer | numpy nditer

In [20]:
```
a = np.arange(12).reshape(3,4)
a
```

Out[20]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [21]:
```
for row in a :
    print(row)
```

```
[0 1 2 3]
[4 5 6 7]
[ 8  9 10 11]
```

In [22]:
```
for row in a :
    for cell in row :
        print(cell)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

there is a C order and a fortran order

In [24]:
```
for x in np.nditer(a,order = 'F'):
    print(x)
```

```
0
4
8
1
5
9
2
6
10
3
7
11
```

In [25]:
```
for x in np.nditer(a,order = 'C'):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

In [26]:
```
yt = np.array([[1,2,3],
               [4,5,6],
               [7,8,9]])
yt
```

Out[26]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [28]:
```
np.savetxt('mypythontext.txt',yt)
```

In [29]:
```
weights = [74.2, 85, 74, 67.9, 52, 70.5, 86, 51.8, 64, 82]
a = np.array(weights)
a
```

Out[29]:
```
array([74.2, 85. , 74. , 67.9, 52. , 70.5, 86. , 51.8, 64. , 82. ])
```

In [36]:
```
[i for i in a if i>68]
```

Out[36]:
```
[74.2, 85.0, 74.0, 70.5, 86.0, 82.0]
```

In [42]:
```
for i in a:
    if i > 68:
        print(i)
```

```
74.2
85.0
74.0
70.5
86.0
82.0
```

In [47]:
```
a = np.random.randint(20,40,10)
a
```

Out[47]:
```
array([30, 23, 30, 21, 24, 25, 26, 20, 28, 27])
```

In [48]:
```
yt = np.arange(1,6)
yt
```

Out[48]:
```
array([1, 2, 3, 4, 5])
```

In [49]:
```
yt**5
```

Out[49]:
```
array([   1,   32,  243, 1024, 3125], dtype=int32)
```

In [52]:
```
z = np.arange(51,100,2)
z
```

Out[52]:
```
array([51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83,
       85, 87, 89, 91, 93, 95, 97, 99])
```

In [53]:
```
A = [[1, 4, 8], [5, 7, 3], [9, 14, 2]]
q = np.array(A)
q
```

Out[53]:
```
array([[ 1,  4,  8],
       [ 5,  7,  3],
       [ 9, 14,  2]])
```

In [55]:
```
q[0]+q[1]+q[2]
```

Out[55]:
```
array([15, 25, 13])
```

In [56]:
```
num = [42, 87, 90, 14, 32, 75, 61, 80, 92]
qw = np.array(num)
qw
```

Out[56]:
```
array([42, 87, 90, 14, 32, 75, 61, 80, 92])
```

In [58]:
```
np.remainder(qw,8)
```

Out[58]:
```
array([2, 7, 2, 6, 0, 3, 5, 0, 4], dtype=int32)
```

In [59]:
```
a = [1,2,3,4,5,6,7,8,9,10,11,12]
ab = np.array(a)
```

In [60]:
```
ab
```

Out[60]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [61]:
```
ab.reshape(3,4)
```

Out[61]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [62]:
```
prices_monday = [10, 30, 50, 60, 90]
a=np.array(prices_monday)
a
```

Out[62]:
```
array([10, 30, 50, 60, 90])
```

In [63]:
```
prices_tuesday = [50, 20, 80, 125, 10, 70]
b = np.array(prices_tuesday)
b
```

Out[63]:
```
array([ 50,  20,  80, 125,  10,  70])
```

In [64]:
```
np.intersect1d(a,b)
```

Out[64]:
```
array([10, 50])
```

In [65]:
```
weights = [57, 69, 54, 65 ,71, 62, 56]
a=np.array(weights)
a
```

Out[65]:
```
array([57, 69, 54, 65, 71, 62, 56])
```

In [66]:
```
heights = [1.59, 1.75, 1.66, 1.74, 1.64, 1.72, 1.53]
b=np.array(heights)
b
```

Out[66]:
```
array([1.59, 1.75, 1.66, 1.74, 1.64, 1.72, 1.53])
```

In [67]:
```
BMI = a/b**2
BMI
```

Out[67]:
```
array([22.54657648, 22.53061224, 19.59645812, 21.46915048, 26.39797739,
       20.9572742 , 23.922423  ])
```

In [68]:
```
array1 = [[[3, 4], [8, 2],[5, 9]]]
ar = np.array(array1)
ar
```

Out[68]:
```
array([[[3, 4],
        [8, 2],
        [5, 9]]])
```

In [73]:
```
r = [[[5,7]]]
br = np.array(r)
br
```

Out[73]:
```
array([[[5, 7]]])
```

In [79]:
```
ty = np.append(ar,br,axis=1)
ty
```

Out[79]:
```
array([[[3, 4],
        [8, 2],
        [5, 9],
        [5, 7]]])
```

In [80]:
```
M = [[15, 17, 45, 56], [7, 42, 15, 63], [54, 3, 61, 41], [0, 87, 16, 20]]
t=np.array(M)
t
```

Out[80]:
```
array([[15, 17, 45, 56],
       [ 7, 42, 15, 63],
       [54,  3, 61, 41],
       [ 0, 87, 16, 20]])
```

In [86]:
```
f=(t).T
f
```

Out[86]:
```
array([[15,  7, 54,  0],
       [17, 42,  3, 87],
       [45, 15, 61, 16],
       [56, 63, 41, 20]])
```

In [87]:
```
A1 = [[5, 9], [7, 6]]
a=np.array(A1)
a
```

Out[87]:
```
array([[5, 9],
       [7, 6]])
```

In [88]:
```
A2 = [[8,-7], [0,4]]
b=np.array(A2)
b
```

Out[88]:
```
array([[ 8, -7],
       [ 0,  4]])
```

In [90]:
```
h=np.vstack([a,b])
h
```

Out[90]:
```
array([[ 5,  9],
       [ 7,  6],
       [ 8, -7],
       [ 0,  4]])
```

In [91]:
```
A1 = [[-81, 75, 40], [27, 67, 52]]
a = np.array(A1)
a
```

Out[91]:
```
array([[-81,  75,  40],
       [ 27,  67,  52]])
```

In [92]:
```
A2 = [[15, 54], [39, 56]]
b = np.array(A2)
b
```

Out[92]:
```
array([[15, 54],
       [39, 56]])
```

In [94]:
```
g=np.hstack([a,b])
g
```

Out[94]:
```
array([[-81,  75,  40,  15,  54],
       [ 27,  67,  52,  39,  56]])
```

In [95]:
```
A = [[4, 7], [2, -3], [8, 1], [0, 9], [5,-1], [8, 3]]
a = np.array(A)
a
```

Out[95]:
```
array([[ 4,  7],
       [ 2, -3],
       [ 8,  1],
       [ 0,  9],
       [ 5, -1],
       [ 8,  3]])
```

In [96]:
```
np.split(a,3)
```

Out[96]:
```
[array([[ 4,  7],
        [ 2, -3]]),
 array([[8, 1],
        [0, 9]]),
 array([[ 5, -1],
        [ 8,  3]])]
```

In [98]:
```
B = [[8, 7, 0, 9], [2, -3, 5, 10], [8, 5, -1,3]]
d = np.array(B)
d
```

Out[98]:
```
array([[ 8,  7,  0,  9],
       [ 2, -3,  5, 10],
       [ 8,  5, -1,  3]])
```

In [102]:
```
y=np.split(d,2,axis=1)
y
```

Out[102]:
```
[array([[ 8,  7],
        [ 2, -3],
        [ 8,  5]]),
 array([[ 0,  9],
        [ 5, 10],
        [-1,  3]])]
```

In [ ]:
```
# concatenate arrays along rows(axis = 0) (row wise)
# concatenate arrays along columns(axis = 1) (column wise)
```

In [8]:
```
a = np.eye(5,6,k=3)
a
```

Out[8]:
```
array([[0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

In [16]:
```
nparray2=np.random.randn(5,5)
nparray2.round(5)
```

Out[16]:
```
array([[-0.67125, -0.5129 , -2.29999,  0.74564,  0.80537],
       [ 0.8028 ,  0.00989,  0.01904, -0.57668,  0.6307 ],
       [-0.14441,  2.27359,  1.1587 ,  0.23474, -0.67473],
       [ 0.34787, -0.5572 , -1.21002, -1.57203, -0.50569],
       [-0.01106, -1.37369, -0.719  , -0.23142,  0.2053 ]])
```

In [ ]:
```
```