```
"""
Registration : xxxx
Description  : Basics of Numerical Python
Author       : AKB
"""

import numpy as np
import warnings
warnings.filterwarnings("ignore")
print('Floor of 8.92 : ', np.floor(8.92), ', Ceil of 8.92 : ', np.ceil(8.92))

#=============================================================================================#
print                        ('~~~ NUMPY ARRAY ~~~')                                        #
#=============================================================================================#
#=== Shape, Dimension Check 1D           ==#
a = np.array([1,2,3,4,5,6])                # convert 1D list into 1D array (vector)
print ('1D array : ', a)
print ('Size of array : ',          a.size)
print ('Shape of array (tuple) : ', a.shape)
print ('Dimension of array : '    , a.ndim)
print ('Array data type : '       , a.dtype)
a = np.array([1.0, 2, 3, 4, 5, 6]);  print ('Datatype (float) : ',   a.dtype)
a = np.array([1, 2+1j, 3, 4, 5, 6]); print ('Datatype (complex) : ', a.dtype)
a = np.array([1.0, 2.0, 3, 4, 5, 6], dtype=int); print ('Typecast float to (floor) int :
', a)
a = np.array([1.0, 2.0, 3, 4, 5, 6]); print ("Typecast float to (floor) int : \n",
a.astype(int))

#=== Reshape, Resize, Dimension Check 2D  ==#
b = np.array([[1,2,3],[4,5,6]])             # convert 2D nested list into 2D array
(matrix)
print ('\n2D array : \n',           b)
print ('Shape of array (tuple) : ', b.shape)
print ('Dimension of array : ',     b.ndim)
print ('Reshape in 2X3 matrix: \n', b.reshape(2,3))
print ('Reshape in 3X2 matrix: \n', b.reshape(3,2))

#=== Reshape, Resize, Dimension Check 3D  ==#
c = np.array([1,2,3,4,5,6,7,8,9,10,11,12])  # convert 1D list into 1D array (vector)
print ('\n1D array : ', c)
print ('Shape of array (tuple) : ', c.shape)
print ('Dimension of array : ', c.ndim)
print ('Reshape in 2X3X2 array: \n', c.reshape(2,3,2))  # Nested list
print ('Check array dimension : ',
np.array([1,2,3,4,5,6,7,8,9,10,11,12]).reshape(2,3,2).ndim)
print ("Reshape doesn't affect original array c : ", c) # or assign d = c.reshape(2,3,2)
c.resize(2,3,2)
print ('Resize affects original array c :\n', c)
print ('Converting Tuple into Array :\n', np.array([(1,2,3),(4,5,6)]))
print ('Direct Definition :\n', np.array([[[1,2],[3,4],[5,6]],[[7,8],[9,10],[11,12]]]))

#=============================================================================================#
print                        ('\n~~~ SPECIAL ARRAY ~~~')                                     #
#=============================================================================================#
d = np.zeros(3);                    print ('Zero 1D array : '               , d)
d = np.zeros((3,3));                print ('Null Marix : \n'                , d)
d = np.ones(3);                     print ('1D array of Ones: '             , d)
d = np.ones((3,3));                 print ('Marix of Ones : \n'             , d)
d = np.eye(3);                      print ('Identity Matrix of order 3 : \n', d)
d = np.eye(3,5);                    print ("Identity 3-row, 5-column array : \n", d)
d = np.identity(3);                 print ("Identity array with 1's in main diagonal :
\n", d)
d = np.full((3,3),5);               print ('3X3 Constant Matrix : \n'       , d)
d = np.empty(3); d = d.fill(0);     print ('Populate void array with 0 : \n', d)
d = np.random.random((3,3));        print ('Random matrix : \n'             , d)
d = np.arange(10);                  print ('Create 0-9 1D array : '         , d)
```

```
d = np.arange(1.1,4.9,0.4);         print ('Create 1D array from 1.1 excluding 4.9 with
stride 0.5 : \n', d)
d = np.linspace(1.1,4.7,10);        print ('Create 1D array from 1.1 including 4.9 using
linspace  : \n', d)
d = np.arange(1,12,2).reshape(2,3); print ('Array creation & reshape :\n'   , d)
d = np.logspace(1,3,20);            print ('Create 1D log array from 10^1 to 10^3 with 20
points : \n', d)

#=============================================================================================#
print                   ('\n~~~ ALGEBRA WITH ARRAY ~~~')
#
#=============================================================================================#
print      ('1D ARRAY')

a = np.array([1,2,3,4,5,6]);
b = np.array([7,8,9,10,11,12]);
print ('a =', a, ' b =', b, ', Full Array a :', a[::])
print ('Addition of respective elements :', a+b)        # np.add(a,b) does the same
print ('Concatenate a & b : ', np.concatenate([a,b]))   # in list, a+b is concatenation of
a & b
print ('Substraction of respective elements : '  , b-a) # np.substract(b,a) does the same
print ('Multiplication of respective elements : ', b*a) # np.multiply(a,b)  does the same
print ('Division of respective elements : '      , b.astype(float)/a)
print ('a^3+b^2 :', pow(a,3)+pow(b,2))                  # we can also use a**3+b**2
print ('Sum of a :', sum(a), ', Difference of a :', np.diff(a))
print ('Product of a :', np.prod(a))
print ('Max of a :', max(a), ', Min of a :', min(a))
print ('Index of minimum element is ', np.argmin(a), ', minimum of a :', a[np.argmin(a)])
print ('Index of maximum element is ', np.argmax(a), ', maximum of a :', a[np.argmax(a)])
print ('Mean of a :', np.mean(a), ', Median of a :', np.median(a)) # np.mean(a) =
np.average(a)
print ('Variance of a :', np.var(a))
print ('Std. Dev. of a :', np.std(a), ' Which is sqrt(var) :', np.sqrt(np.var(a)))
print ('Sin(a) :', np.sin(a), ', Exp(a) :', np.exp(a), ', sqrt(a) :', np.sqrt(a))
def f(x): return x*x;
print (f(a))
print ('Alternate element of a : ', a[::2], ', First 3 element of a : ', a[:3:])
print ('From 1 to 4 in stride 2 : ', a[1:4:2])
print ('Composite a : ', np.arange(10)[1:5:2])
print ('a.b Inner product : ', np.inner(a,b), ' which is same as : ', sum(a*b))
print ('Inner product with scalar : ', np.inner(a,2))
print ('a.b Vector complex-conjugate dot product : ', np.vdot(a,b)) # for 1D array,
np.dot() = np.vdot() = np.inner()
x = np.array([1,2,3]); y = np.array([-1,3,0]);
print ('Cross product : ', np.cross(x,y))

print     ('\n2D/3D ARRAY')

a = np.array([1,2,3,4,5,6,7,8,9]).reshape(3,3); # same as np.array([[1,2,3],[4,5,6],
[7,8,9]])
b = np.array([4,5,6,7,8,9,10,11,12]).reshape(3,3);
print ('a = \n', a, a[:,:], ', b = \n', b, ', Transpose b = \n', b.T)
print ('Flatten a = \n', np.ravel(a), ', which is same as = \n', a.flatten())
print ('Trace b = ', np.trace(b), ', Rank b = ', np.linalg.matrix_rank(b))
print ('First column of a : ', a[:,0], ', First row of a : ', a[0,:])
print ('Grid : \n', np.mgrid[0:3, 0:2], ', Grid with stride : \n', np.mgrid[0:2:0.5,
0:3])
a = np.array([[1,2,3],[4,5,6]]); b = np.array([[1,0,1],[0,1,0]]);
print ('a :\n', a, ', \nb : \n', b, ', \na.b Inner product : \n', np.inner(a,b), 'a*bT
Matrix product : \n', np.dot(a,b.T))
print ('5*Identity Matrix : \n', np.inner(np.eye(3),5)) # for 2D array, np.vdot() \neq
np.inner()
a = np.random.randn(3,3); print(a)
print ('1st row = ', a[0,:], ', 2nd row = ', a[1,:], ', 3rd row = ', a[2,:])
print ('Vertically stack rows to get a = \n', np.vstack((a[0,:],a[1,:],a[2,:])))
print ('Extract 1st, 2nd, 3rd column (python row-array) & convert row to column')
```

```
a0 = a[:,0]; a0 = a0.reshape(-1,1); # convert row->column
a1 = a[:,1]; a1 = a1.reshape(-1,1);
a2 = a[:,2]; a2 = a2.reshape(-1,1);
print ('1st column = \n', a0, ', 2nd column = \n', a1, ', 3rd column = \n', a2)
print ('Horizontal stack columns to get a = \n', np.hstack((a0,a1,a2)))

#=============================================================================================#

#=============================================================================================#
print   ('\nGEOMETRY COMPUTATION & LINEAR ALGEBRA WITH ARRAY')
#
#=============================================================================================#
a = np.array([2,-3,0]); b = np.array([1,1,-1]); c = np.array([3,0,-1]);
print ('Volume of Parallelopiped a.bxc : ', np.vdot(a, np.cross(b,c)), ' which is same as
', np.dot(a, np.cross(b,c)))

# Matrix a * Vector x = Vector b; Solution : Vector x = a^-1 * b
a = np.array([[1,2,-1],[2,1,4],[3,3,4]]); b = np.array([1,2,1]);
print ('Solution vector x : ', np.linalg.solve(a,b))

# Eigenvalues and Eigenvectors
A = np.array([[1,1,1],[1,2,3],[1,4,9]]);
print ('A : \n', A, ', \nA^-1 : \n', np.linalg.inv(A))
print ('Identity A*A^-1 : \n', np.dot(A, np.linalg.inv(A)))
A = np.array([[1,2],[3,4]]);
print ('Eigenvalue & Eigenvector : \n', np.linalg.eig(A)) # Tuple of two arrays
eigen_val, eigen_vec = np.linalg.eig(A);
print ('Eigenvalue : ', eigen_val, ', Eigenvalue directly : ', np.linalg.eigvals(A))
print ('Eigenvector : ', eigen_vec)
print ('1st Eigenvector : ', eigen_vec[:,0], ',\n 2nd Eigenvector : ', eigen_vec[:,1])


"""
Results:
Floor of 8.92 :  8.0 , Ceil of 8.92 :  9.0
~~~ NUMPY ARRAY ~~~
1D array :  [1 2 3 4 5 6]
Shape of array (tuple) :  (6,)
Dimension of array :  1
Integer data type:  int64
Floating point datatype:  float64
Complex datatype:  complex128

2D array :
[[1 2 3]
 [4 5 6]]
Shape of array (tuple) :  (2, 3)
Dimension of array :  2
Reshape in 2X3 matrix:
[[1 2 3]
 [4 5 6]]
Reshape in 3X2 matrix:
[[1 2]
 [3 4]
 [5 6]]

1D array :  [ 1  2  3  4  5  6  7  8  9 10 11 12]
Shape of array (tuple) :  (12,)
Dimension of array :  1
Reshape in 2X3X2 array:
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]]
```

```
  [11 12]]]
Check array dimension :  3
Reshape doesn't affect original array c :  [ 1  2  3  4  5  6  7  8  9 10 11 12]
Resize affects original array c :
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
Converting Tuple into Array :
[[1 2 3]
 [4 5 6]]
Direct Definition :
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

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

~~~ SPECIAL ARRAY ~~~
Zero 1D array :  [ 0.  0.  0.]
Null Marix :
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
1D array of Ones:  [ 1.  1.  1.]
Marix of Ones :
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
Identity Matrix of order 3 :
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
3X3 Constant Matrix :
[[ 5.  5.  5.]
 [ 5.  5.  5.]
 [ 5.  5.  5.]]
Random matrix :
[[ 0.06342705  0.57982624  0.81059203]
 [ 0.93842308  0.50303311  0.92540096]
 [ 0.19309967  0.88882978  0.15316389]]
Create 0-9 1D array :  [0 1 2 3 4 5 6 7 8 9]
Create 1D array from 1.1 excluding 4.9 with stride 0.5 :
[ 1.1  1.5  1.9  2.3  2.7  3.1  3.5  3.9  4.3  4.7]
Create 1D array from 1.1 including 4.9 using linspace  :
[ 1.1  1.5  1.9  2.3  2.7  3.1  3.5  3.9  4.3  4.7]
Array creation & reshape :
[[ 1  3  5]
 [ 7  9 11]]

~~~ ALGEBRA WITH ARRAY ~~~
1D ARRAY
a = [1 2 3 4 5 6]  b = [ 7  8  9 10 11 12] , Full Array a : [1 2 3 4 5 6]
Addition of respective elements : [ 8 10 12 14 16 18] Using add() method :  [ 8 10 12 14
16 18]
Concatenate a & b :  [ 1  2  3  4  5  6  7  8  9 10 11 12]
Substraction of respective elements :  [6 6 6 6 6 6]
Multiplication of respective elements :  [ 7 16 27 40 55 72]
Division of respective elements :  [7 4 3 2 2 2]
a^3+b^2 : [ 50  72 108 164 246 360]
Sum of a : 21 , Difference of a : [1 1 1 1 1]
```

```
Product of a : 720
Max of a : 6 , Min of a : 1
Mean of a : 3.5 , Median of a : 3.5
Variance of a : 2.91666666667
Std. Dev. of a : 1.70782512766  Which is sqrt(var) : 1.70782512766
Sin(a) : [ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155 ] ,
Exp(a) : [   2.71828183    7.3890561    20.08553692   54.59815003  148.4131591
403.42879349] ,
Sqrt(a) : [ 1.          1.41421356  1.73205081  2.          2.23606798  2.44948974]
[ 1  4  9 16 25 36]
Alternate element of a :  [1 3 5] , First 3 element of a :  [1 2 3]
From 1 to 4 in stride 2 :  [2 4]
Composite a :  [1 3]
a.b Inner product :  217  which is same as :  217
Inner product with scalar :  [ 2  4  6  8 10 12] a.b Vector product :  217
Cross product :  [-9 -3  5]

2D/3D ARRAY
a =
[[1 2 3]
 [4 5 6]
 [7 8 9]] [[1 2 3]
 [4 5 6]
 [7 8 9]] , b =
[[ 4  5  6]
 [ 7  8  9]
 [10 11 12]] , Transpose b =
[[ 4  7 10]
 [ 5  8 11]
 [ 6  9 12]]
Trace b =  24 , Rank b =  2
First column of a :  [1 4 7] , First row of a :  [1 2 3]
Grid :
[[[0 0]
  [1 1]
  [2 2]]

 [[0 1]
  [0 1]
  [0 1]]] , Grid with stride :
[[[ 0.   0.   0. ]
  [ 0.5  0.5  0.5]
  [ 1.   1.   1. ]
  [ 1.5  1.5  1.5]]

 [[ 0.   1.   2. ]
  [ 0.   1.   2. ]
  [ 0.   1.   2. ]
  [ 0.   1.   2. ]]]
a :
[[1 2 3]
 [4 5 6]] ,
b :
[[1 0 1]
 [0 1 0]] ,
a.b Inner product :
[[ 4  2]
 [10  5]] 5*Identity Matrix :
[[ 5.  0.  0.]
 [ 0.  5.  0.]
 [ 0.  0.  5.]]

GEOMETRY COMPUTATION & LINEAR ALGEBRA WITH ARRAY
Volume of Parallelopiped a.bxc :  4
Solution vector x :  [ 7. -4. -2.]
A :
```

```
[[1 1 1]
 [1 2 3]
 [1 4 9]] ,
A^-1 :
[[ 3.  -2.5  0.5]
 [-3.   4.  -1. ]
 [ 1.  -1.5  0.5]]
Identity A*A^-1 :
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
Eigenvalue & Eigenvector :
(array([-0.37228132,  5.37228132]), array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]]))
Eigenvalue :  [-0.37228132  5.37228132] , Eigenvalue directly :  [-0.37228132  5.37228132]
Eigenvector :  [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
1st Eigenvector :  [-0.82456484  0.56576746] ,
 2nd Eigenvector :  [-0.41597356 -0.90937671]
"""
```