In [ ]:

```
import numpy as np
```

Print the numpy version and the configuration

In [ ]:

```
print(np.__version__)
np.show_config()
```

Create a null vector of size 10

In [ ]:

```
z=np.zeros(10)
print(z)
```

How to get the documentation of the numpy add function from the command line?

In [3]:

```
python -c "import numpy; numpy.info(numpy.add)"
```

```
  File "<ipython-input-3-1663cd30cb7c>", line 1
    python -c "import numpy; numpy.info(numpy.add)"
                                                  ^
SyntaxError: invalid syntax
```

Create a null vector of size 10 but the fifth value which is 1

In [ ]:

```
z=np.zeros(10)
z[4]=1
print(z)
```

Create a vector with values ranging from 10 to 49

In [ ]:

```
z=np.arange(10,50)
print(z)
```

Reverse a vector (first element becomes last)

In [ ]:

```
import numpy as np
z=np.arange(50)
z=z[::-1]
print(z)
```

Create a 3x3 matrix with values ranging from 0 to 8

In [ ]:

```
z=np.arange(9).reshape(3,3)
print(z)
```

In [ ]:

```
z=[[1,2,3],[4,5,66],[8.9,10]]
#print(\n),z
print('\n',z)
```

Find indices of non-zero elements from [1,2,0,0,4,0]

In [ ]:

```
nz = np.nonzero([1,2,0,0,4,0])
print(nz)
```

Create a 3x3 identity matrix

In [ ]:

```
Z = np.eye(3)
print(Z)
```

Create a 3x3x3 array with random values

In [ ]:

```
Z = np.random.random((3,3,3))
print(Z)
```

Create a 10x10 array with random values and find the minimum and maximum values

In [ ]:

```
Z = np.random.random((10,10))
Zmin, Zmax = Z.min(), Z.max()
print(Zmin, Zmax)
```

Create a random vector of size 30 and find the mean value

In [ ]:

```
Z = np.random.random(30)
m = Z.mean()
print(Z)
print(m)
```

Create a 2d array with 1 on the border and 0 inside

In [8]:

```
Z = np.ones((10,10))
Z[1:-1,1:-1] = 0
Z
```

Out[8]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

What is the result of the following expression?

In [7]:

```
0 * np.nan
np.nan == np.nan
np.inf > np.nan
np.nan - np.nan
0.3 == 3 * 0.1
```

Out[7]:

```
False
```

Create a 5x5 matrix with values 1,2,3,4 just below the diagonal

In [ ]:

```
Z = np.diag(1+np.arange(4),k=-1)
print(Z)
```

In [ ]:

```
np.zeros((4,3))
```

Create a 8x8 matrix and fill it with a checkerboard pattern

In [ ]:

```
Z = np.zeros((8,8))
Z[1::2,::2] = 1
Z[::2,1::2] = 1
print(Z)
```

Consider a (6,7,8) shape array, what is the index (x,y,z) of the 100th element

In [28]:

```
print(np.unravel_index(100,(6,7,8)))
```

```
(1, 5, 4)
```

Create a checkerboard 8x8 matrix using the tile function

In [29]:

```
Z = np.tile( np.array([[0,1],[1,0]]), (4,4))
print(Z)
```

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

Normalize a 5x5 random matrix

In [30]:

```
Z = np.random.random((5,5))
print (Z)
Zmax, Zmin = Z.max(), Z.min()
Z = (Z - Zmin)/(Zmax - Zmin)
print(Z)
```

```
[[0.02429511 0.58426263 0.35364069 0.10138726 0.45659093]
 [0.89613084 0.55544752 0.55998302 0.83451781 0.96319725]
 [0.06868255 0.11500329 0.10405114 0.06943634 0.58030985]
 [0.71293239 0.15927079 0.0215707  0.72385381 0.34782926]
 [0.64735475 0.64889829 0.00789273 0.62327401 0.91468775]]
[[0.0171698  0.60333631 0.36192435 0.09786883 0.46969128]
 [0.92979578 0.57317303 0.57792074 0.86530009 1.        ]
 [0.06363398 0.11212191 0.10065734 0.06442303 0.59919859]
 [0.7380261  0.15846053 0.01431793 0.74945849 0.35584102]
 [0.6693803  0.67099605 0.         0.6441729  0.9492209 ]]
```

Create a custom dtype that describes a color as four unisgned bytes (RGBA)

In [ ]:

```
color = np.dtype([("r", np.ubyte, 1),
                  ("g", np.ubyte, 1),
                  ("b", np.ubyte, 1),
                  ("a", np.ubyte, 1)])
```

Multiply a 5x3 matrix by a 3x2 matrix (real matrix product)

In [ ]:

```
Z = np.dot(np.ones((5,3)), np.ones((3,2)))
print(Z)
```

Given a 1D array, negate all elements which are between 3 and 8, in place.

In [ ]:

```
Z = np.arange(11)
Z[(3 < Z) & (Z <= 8)] *= -1
print(Z)
```

In [ ]:

```
print(sum(range(5),-1))
from numpy import *
print(sum(range(5),-1))
```

In [ ]:

```
x=linspace(5,10,num=2)
x
```

Consider an integer vector Z, which of these expressions are legal

In [75]:

```
Z**Z
2 << Z >> 2
Z <- Z
1j*Z
Z/1/1
Z<Z>Z
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-75-4e3654d03fce> in <module>
      1 Z**Z
----> 2 2 << Z >> 2
      3 Z <- Z
      4 1j*Z
      5 Z/1/1

TypeError: ufunc 'left_shift' not supported for the input types, and the i
nputs could not be safely coerced to any supported types according to the
 casting rule ''safe''
```

What are the result of the following expressions?

In [ ]:

```
np.array(0) // np.array(0)
np.array(0) // np.array(0.)
np.array(0) / np.array(0)
np.array(0) / np.array(0.)
```

How to round away from zero a float array ?

In [ ]:

```
Z = np.random.uniform(-10,+10,10)
print (np.trunc(Z + np.copysign(0.5, Z)))
```

Extract the integer part of a random array using 5 different methods

In [ ]:

```
Z = np.random.uniform(0,10,10)

print (Z - Z%1)
print (np.floor(Z))
print (np.ceil(Z)-1)
print (Z.astype(int))
print (np.trunc(Z))
```

In [ ]:

```
print (np.floor(Z))
```

In [ ]:

```
Z = np.random.uniform(0,10,10)
Z
```

In [ ]:

```
print (np.trunc(Z))
```

Create a 5x5 matrix with row values ranging from 0 to 4

In [74]:

```
import numpy as np
Z = np.zeros((5,5))
Z += np.arange(5)
print(Z)
```

```
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
```

In [ ]:

```
Z = np.zeros((5,5))
Z
```

In [ ]:

```
Z = np.arange(5)
Z
```

Consider a generator function that generates 10 integers and use it to build an array

In [ ]:

```
def generate():
    for x in range(10):
        yield x
Z = np.fromiter(generate(),dtype=float,count=-1)
print(Z)
```

Create a vector of size 10 with values ranging from 0 to 1, both excluded

In [ ]:

```
Z = np.linspace(0,1,12,endpoint=True)[1:-1]
print(Z)
```

In [ ]:

```
Z = np.linspace(0,1,num=10)
print(Z)
```

In [ ]:

```
Z = np.linspace(0,5,12,endpoint=True)[1:-1]
print(Z)
```

Create a random vector of size 10 and sort it

In [ ]:

```
Z = np.random.random(10)
Z.sort()
print(Z)
```

In [ ]:

```
Z=np.line[1,2,3,4,5,6,7]
Z.sort()
print(Z)
```

How to sum a small array faster than np.sum?

In [ ]:

```
Z = np.arange(10)
np.add.reduce(Z)
```

Consider two random array A anb B, check if they are equal

In [ ]:

```
A = np.random.randint(0,2,5)
B = np.random.randint(0,2,5)
equal = np.allclose(A,B)
print(equal)
```

Make an array immutable (read-only)

In [ ]:

```
Z = np.zeros(10)
Z.flags.writeable = False
Z[0] = 1
```

Consider a random 10x2 matrix representing cartesian coordinates, convert them to polar coordinates

In [ ]:

```
Z = np.random.random((10,2))
X,Y = Z[:,0], Z[:,1]
R = np.sqrt(X**2+Y**2)
T = np.arctan2(Y,X)
print(R)
print(T)
```

In [ ]:

```
x=np.arange(11)
x[:,5]
```

Create random vector of size 10 and replace the maximum value by 0

In [ ]:

```
Z = np.random.random(10)
Z[Z.argmax()] = 0
print(Z)
```

Create a structured array with x and y coordinates covering the [0,1]x[0,1] area

In [ ]:

```
Z = np.zeros((10,10), [('x',float),('y',float)])
Z['x'], Z['y'] = np.meshgrid(np.linspace(0,1,10),
                             np.linspace(0,1,10))
print(Z)
```

Given two arrays, X and Y, construct the Cauchy matrix C (Cij = 1/(xi - yj))

In [ ]:

```
X = np.arange(8)
Y = X + 0.5
C = 1.0 / np.subtract.outer(X, Y)
print(np.linalg.det(C))
```

In [ ]:

```
y=np.eye(5)
y
```

Print the minimum and maximum representable value for each numpy scalar type

In [ ]:

```
for dtype in [np.int8, np.int32, np.int64]:
   print(np.iinfo(dtype).min)
   print(np.iinfo(dtype).max)
for dtype in [np.float32, np.float64]:
   print(np.finfo(dtype).min)
   print(np.finfo(dtype).max)
   print(np.finfo(dtype).eps)
```

How to print all the values of an array?

In [ ]:

```
np.set_printoptions(threshold=np.nan)
Z = np.zeros((25,25))
print(Z)
```

How to find the closest value (to a given scalar) in an array?

In [ ]:

```
Z = np.arange(100)
v = np.random.uniform(0,100)
index = (np.abs(Z-v)).argmin()
print(Z[index])
```

In [ ]:

```
v = np.random.uniform(0,100)
v
```

Create a structured array representing a position (x,y) and a color (r,g,b)

In [ ]:

```
Z = np.zeros(10, [ ('position', [ ('x', float, 1),
                                   ('y', float, 1)]),
                    ('color',    [ ('r', float, 1),
                                   ('g', float, 1),
                                   ('b', float, 1)])])
print(Z)
```

In [ ]:

```
Z = np.ones(10, [ ('position', [ ('x', float, 0),
                                  ('y', float, 0)]),
                   ('color',    [ ('r', int, 0),
                                  ('g', int, 0),
                                  ('b', int, 0)])])
print(Z)
```

Consider a random vector with shape (100,2) representing coordinates, find point by point distances

In [ ]:

```
Z = np.random.random((10,2))
X,Y = np.atleast_2d(Z[:,0]), np.atleast_2d(Z[:,1])
D = np.sqrt( (X-X.T)**2 + (Y-Y.T)**2)
print(D)
```

Consider a random vector with shape (100,2) representing coordinates, find point by point distances

In [ ]:

```
import scipy
import scipy.spatial
Z = np.random.random((10,2))
D = scipy.spatial.distance.cdist(Z,Z)
print(D)
```

How to convert a float (32 bits) array into an integer (32 bits) in place

In [6]:

```
import numpy as np
Z = np.arange(10, dtype=np.int32)
Z = Z.astype(np.float32, copy=False)
print (Z)
```

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

How to read the following file

In [9]:

```
# File content:
# -------------
1,2,3,4,5
6,,,7,8
,,9,10,11
# -------------

Z = np.genfromtxt("missing.dat", delimiter=",")
```

```
  File "<ipython-input-9-4c8321f65853>", line 4
    6,,,7,8
      ^
SyntaxError: invalid syntax
```

What is the equivalent of enumerate for numpy arrays?

In [10]:

```
Z = np.arange(9).reshape(3,3)
for index, value in np.ndenumerate(Z):
    print(index, value)
for index in np.ndindex(Z.shape):
    print(index, Z[index])
```

```
(0, 0) 0
(0, 1) 1
(0, 2) 2
(1, 0) 3
(1, 1) 4
(1, 2) 5
(2, 0) 6
(2, 1) 7
(2, 2) 8
(0, 0) 0
(0, 1) 1
(0, 2) 2
(1, 0) 3
(1, 1) 4
(1, 2) 5
(2, 0) 6
(2, 1) 7
(2, 2) 8
```

In [12]:

```
Z = np.arange(9).reshape(3,3)
for index, value in np.ndenumerate(Z):
    print(index, value)
```

```
(0, 0) 0
(0, 1) 1
(0, 2) 2
(1, 0) 3
(1, 1) 4
(1, 2) 5
(2, 0) 6
(2, 1) 7
(2, 2) 8
```

Generate a generic 2D Gaussian-like array

In [13]:

```
X, Y = np.meshgrid(np.linspace(-1,1,10), np.linspace(-1,1,10))
D = np.sqrt(X*X+Y*Y)
sigma, mu = 1.0, 0.0
G = np.exp(-( (D-mu)**2 / ( 2.0 * sigma**2 ) ) )
print(G)
```

```
[[0.36787944 0.44822088 0.51979489 0.57375342 0.60279818 0.60279818
  0.57375342 0.51979489 0.44822088 0.36787944]
 [0.44822088 0.54610814 0.63331324 0.69905581 0.73444367 0.73444367
  0.69905581 0.63331324 0.54610814 0.44822088]
 [0.51979489 0.63331324 0.73444367 0.81068432 0.85172308 0.85172308
  0.81068432 0.73444367 0.63331324 0.51979489]
 [0.57375342 0.69905581 0.81068432 0.89483932 0.9401382  0.9401382
  0.89483932 0.81068432 0.69905581 0.57375342]
 [0.60279818 0.73444367 0.85172308 0.9401382  0.98773022 0.98773022
  0.9401382  0.85172308 0.73444367 0.60279818]
 [0.60279818 0.73444367 0.85172308 0.9401382  0.98773022 0.98773022
  0.9401382  0.85172308 0.73444367 0.60279818]
 [0.57375342 0.69905581 0.81068432 0.89483932 0.9401382  0.9401382
  0.89483932 0.81068432 0.69905581 0.57375342]
 [0.51979489 0.63331324 0.73444367 0.81068432 0.85172308 0.85172308
  0.81068432 0.73444367 0.63331324 0.51979489]
 [0.44822088 0.54610814 0.63331324 0.69905581 0.73444367 0.73444367
  0.69905581 0.63331324 0.54610814 0.44822088]
 [0.36787944 0.44822088 0.51979489 0.57375342 0.60279818 0.60279818
  0.57375342 0.51979489 0.44822088 0.36787944]]
```

In [15]:

```
X, Y = np.meshgrid(np.linspace(-1,1,10), np.linspace(-1,1,10))
X
```

Out[15]:

```
array([[-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ],
       [-1.        , -0.77777778, -0.55555556, -0.33333333, -0.11111111,
         0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.        ]])
```

In [34]:

```
p,q=np.meshgrid(np.linspace(1,5,num=3),np.linspace(1,5,num=3))
print (p)
print (q)
```

```
[[1. 3. 5.]
 [1. 3. 5.]
 [1. 3. 5.]]
[[1. 1. 1.]
 [3. 3. 3.]
 [5. 5. 5.]]
```

In [19]:

```
p=np.linspace(1,5,num=3)
p
```

Out[19]:

```
array([1., 3., 5.])
```

How to randomly place p elements in a 2D array

In [31]:

```
n = 10
p = 3
Z = np.zeros((n,n))
np.put(Z, np.random.choice(range(n*n), p, replace=False),1)
print (Z)
```

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

Subtract the mean of each row of a matrix

In [36]:

```
X = np.random.rand(5, 10)

# Recent versions of numpy
Y = X - X.mean(axis=1, keepdims=True)
print (Y)

# Older versions of numpy
#Y = X - X.mean(axis=1).reshape(-1, 1)
```

```
[[ 0.3552966   0.07506684  0.1785039   0.22909465  0.10772008 -0.29702378
  -0.20660405 -0.46078938 -0.20147884  0.22021397]
 [-0.33792143  0.22186499  0.1345527   0.17122829  0.27581006 -0.22262301
   0.11490968  0.14530541 -0.21966743 -0.28345927]
 [-0.60236249  0.10341078  0.23245573  0.32842133 -0.26385268 -0.21250231
   0.11046165  0.27427181  0.3152317  -0.28553551]
 [ 0.41702087 -0.3775656  -0.16289065 -0.27619611  0.0276702   0.09521887
  -0.35550889 -0.08266173  0.35029368  0.36461935]
 [ 0.33255525  0.50155656 -0.12829193  0.48620026 -0.3636681  -0.22053549
   0.39563732 -0.35796506 -0.36114823 -0.28434059]]
```

In [40]:

```
X = np.random.randint(5, 10)
X
```

Out[40]:

```
8
```

In [41]:

```
X = np.randint(5, 10)
X
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-41-2807f4b4aff1> in <module>
----> 1 X = np.randint(5, 10)
      2 X

AttributeError: module 'numpy' has no attribute 'randint'
```

In [43]:

```
v=np.random.randint(0, 10)
v
```

Out[43]:

```
3
```

In [46]:

```
import random
r1 =random.randint(0, 10)
print("Random number between 0 and 10 is % s" % (r1))
```

```
Random number between 0 and 10 is 5
```

In [47]:

```
X = np.random.rand(5, 10)
X
```

Out[47]:

```
array([[0.39765281, 0.89106942, 0.10365195, 0.82354838, 0.95145236,
        0.38822961, 0.69173773, 0.72617645, 0.24026646, 0.84039126],
       [0.84743041, 0.1806665 , 0.35309973, 0.84701158, 0.09434507,
        0.00128798, 0.50222661, 0.90956962, 0.63800427, 0.84437157],
       [0.67681902, 0.14588558, 0.06852285, 0.08009835, 0.27762657,
        0.99463015, 0.57610973, 0.0866701 , 0.95227108, 0.49712417],
       [0.11239393, 0.62269565, 0.28298164, 0.84916617, 0.14763049,
        0.83300577, 0.12212292, 0.98091236, 0.73844636, 0.60679195],
       [0.72535094, 0.03486456, 0.51026099, 0.68595312, 0.44636713,
        0.33335214, 0.85793742, 0.86967407, 0.73380959, 0.97043606]])
```

How to I sort an array by the nth column

In [48]:

```
Z = np.random.randint(0,10,(3,3))
print(Z)
print(Z[Z[:,1].argsort()])
```

```
[[2 0 6]
 [5 4 8]
 [4 2 9]]
[[2 0 6]
 [4 2 9]
 [5 4 8]]
```

In [52]:

```
Z = np.random.randint(0,10,(3,3))
Z
```

Out[52]:

```
array([[6, 8, 5],
       [7, 2, 7],
       [0, 7, 8]])
```

How to tell if a given 2D array has null columns

In [53]:

```
Z = np.random.randint(0,3,(3,10))
print((~Z.any(axis=0)).any())
```

```
False
```

Find the nearest value from a given value in an array

In [54]:

```
Z = np.random.uniform(0,1,10)
z = 0.5
m = Z.flat[np.abs(Z - z).argmin()]
print(m)
```

```
0.44923226201027855
```

In [55]:

```
Z = np.random.uniform(0,1,10)
Z
```

Out[55]:

```
array([0.66323707, 0.06902881, 0.63444483, 0.18724202, 0.9907859 ,
       0.14301724, 0.04307186, 0.18519934, 0.81077038, 0.46295675])
```

Create an array class that has a name attribute

In [56]:

```
class NamedArray(np.ndarray):
    def __new__(cls, array, name="no name"):
        obj = np.asarray(array).view(cls)
        obj.name = name
        return obj
    def __array_finalize__(self, obj):
        if obj is None: return
        self.info = getattr(obj, 'name', "no name")

Z = NamedArray(np.arange(10), "range_10")
print (Z.name)
```

```
range_10
```

Consider a given vector, how to add 1 to each element indexed by a second vector (be careful with repeated indices)

In [57]:

```
Z = np.ones(10)
I = np.random.randint(0,len(Z),20)
Z += np.bincount(I, minlength=len(Z))
print(Z)
```

```
[3. 3. 3. 6. 1. 2. 3. 1. 6. 2.]
```

In [59]:

```
Z = np.ones(10)
I = np.random.randint(0,len(Z),20)
I
```

Out[59]:

```
array([2, 2, 0, 1, 9, 6, 7, 1, 4, 4, 7, 5, 6, 4, 7, 6, 5, 9, 2, 2])
```

How to accumulate elements of a vector (X) to an array (F) based on an index list (I)

In [72]:

```
X = [1,2,3,4,5,6,50,1]
I = [1,3,9,3,4,1,50,50]
y = np.bincount(I,X)
p = np.bincount(X)
F = np.bincount(I)
print(y)
```

```
[ 0.  7.  0.  6.  5.  0.  0.  0.  0.  3.  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.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 51.]
```

In [66]:

```
w = np.array([0.3, 0.5, 0.2, 0.7, 1., -0.6]) # weights
x = np.array([0, 1, 1, 2, 2, 2])
z=np.bincount(x,  weights=w)
print(z)
```

```
[0.3 0.7 1.1]
```

Considering a (w,h,3) image of (dtype=ubyte), compute the number of unique colors

In [73]:

```
w,h = 16,16
I = np.random.randint(0,2,(h,w,3)).astype(np.ubyte)
F = I[...,0]*256*256 + I[...,1]*256 +I[...,2]
n = len(np.unique(F))
print(np.unique(I))
```

```
[0 1]
```

Considering a four dimensions array, how to get sum over the last two axis at once

In [76]:

```
A = np.random.randint(0,10,(3,4,3,4))
sum = A.reshape(A.shape[:-2] + (-1,)).sum(axis=-1)
print(sum)
```

```
[[54 34 39 55]
 [54 46 39 46]
 [66 56 57 65]]
```

In [80]:

```
A = np.random.randint(0,10,(3,4,3,4))
A
```

Out[80]:

```
array([[[[3, 5, 2, 7],
         [2, 1, 8, 5],
         [9, 8, 6, 5]],

        [[0, 3, 6, 0],
         [4, 8, 2, 4],
         [1, 1, 9, 8]],

        [[2, 5, 7, 7],
         [2, 5, 4, 6],
         [1, 6, 0, 5]],

        [[7, 0, 9, 9],
         [4, 2, 7, 8],
         [1, 7, 9, 2]]],


       [[[8, 7, 9, 0],
         [2, 2, 3, 2],
         [5, 3, 6, 0]],

        [[2, 6, 6, 8],
         [7, 3, 7, 5],
         [9, 1, 0, 6]],

        [[8, 0, 6, 6],
         [6, 8, 6, 3],
         [5, 5, 7, 4]],

        [[2, 2, 9, 6],
         [4, 9, 1, 3],
         [7, 7, 6, 6]]],


       [[[3, 1, 5, 7],
         [0, 7, 2, 9],
         [2, 0, 3, 7]],

        [[2, 3, 7, 7],
         [3, 0, 0, 6],
         [8, 8, 2, 4]],

        [[1, 0, 7, 9],
         [1, 0, 4, 3],
         [8, 4, 4, 2]],

        [[1, 7, 3, 6],
         [2, 9, 0, 7],
         [5, 9, 4, 7]]]])
```

Considering a one-dimensional vector D, how to compute means of subsets of D using a vector S of same size describing subset indices

In [81]:

```
D = np.random.uniform(0,1,100)
S = np.random.randint(0,10,100)
D_sums = np.bincount(S, weights=D)
D_counts = np.bincount(S)
D_means = D_sums / D_counts
print(D_means)
```

```
[0.54334866 0.53487252 0.50772645 0.51053123 0.50780497 0.51369697
 0.37528892 0.60736201 0.62196484 0.4281351 ]
```

In [84]:

```
D = np.random.uniform(0,1,100)
S = np.random.randint(0,10,100)
#D_sums = np.bincount(S, weights=D)
D_counts = np.bincount(S)
D_counts
```

Out[84]:

```
array([12,  6, 15, 11,  9,  9,  6, 13, 11,  8], dtype=int64)
```

In [85]:

```
S = np.random.randint(0,10,100)
S
```

Out[85]:

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

How to get the diagonal of a dot product?

In [91]:

```
A=np.array([1,1])
B=np.array([2,2])
y=np.diag(np.dot(A, B))
y
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-91-abf395d1dd55> in <module>
      1 A=np.array([1,1])
      2 B=np.array([2,2])
----> 3 y=np.diag(np.dot(A, B))
      4 y

~\Anaconda3\lib\site-packages\numpy\lib\twodim_base.py in diag(v, k)
    263         return diagonal(v, k)
    264     else:
--> 265         raise ValueError("Input must be 1- or 2-d.")
    266 
    267 

ValueError: Input must be 1- or 2-d.
```

In [53]:

```
A=np.array([[1,2,3],[3,4,5]])
A.ndim
A.shape
A.size
```

Out[53]:

```
6
```

In [99]:

```
type(A)
```

Out[99]:

```
list
```

In [26]:

```
import numpy as np
A=np.array([[[[[1,2,3],[3,4,5]]]]])
A.ndim
len(A)
```

Out[26]:

```
1
```

Consider the vector [1, 2, 3, 4, 5], how to build a new vector with 3 consecutive zeros interleaved between each value

In [7]:

```
Z = np.array([1,2,3,4,5])
nz = 3
Z0 = np.zeros(len(Z) + (len(Z)-1)*(nz))
Z0[::nz+1] = Z
print(Z0)
```

```
[1. 0. 0. 0. 2. 0. 0. 0. 3. 0. 0. 0. 4. 0. 0. 0. 5.]
```

Consider an array of dimension (5,5,3), how to mulitply it by an array with dimensions (5,5)

In [13]:

```
A = np.ones((5,5,3))
B = 2*np.ones((5,5))
print(A * B[:,:,None])
```

```
[[[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]

 [[2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]
  [2. 2. 2.]]]
```

In [52]:

```
A = np.ones((5,5,3))
A.ndim
A.size
```

Out[52]:

```
75
```

In [15]:

```
A = np.ones((5,5,3))
A
```

Out[15]:

```
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.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]])
```

How to swap two rows of an array

In [20]:

```
A = np.arange(25).reshape(5,5)
A[[0,2]] = A[[2,0]]
print(A)
```

```
[[10 11 12 13 14]
 [ 5  6  7  8  9]
 [ 0  1  2  3  4]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

In [19]:

```
A = np.arange(25).reshape(5,5)
A
```

Out[19]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Consider a set of 10 triplets describing 10 triangles (with shared vertices), find the set of unique line segments composing all the triangles

In [21]:

```
faces = np.random.randint(0,100,(10,3))
F = np.roll(faces.repeat(2,axis=1),-1,axis=1)
F = F.reshape(len(F)*3,2)
F = np.sort(F,axis=1)
G = F.view( dtype=[('p0',F.dtype),('p1',F.dtype)] )
G = np.unique(G)
print(G)
```

```
[( 0, 12) ( 0, 15) ( 1, 63) ( 1, 74) (12, 15) (15, 47) (15, 94) (16, 33)
 (16, 49) (18, 23) (18, 35) (23, 35) (27, 46) (27, 53) (33, 49) (37, 59)
 (37, 88) (39, 44) (39, 79) (44, 79) (46, 53) (47, 94) (55, 76) (55, 98)
 (58, 75) (58, 96) (59, 88) (63, 74) (75, 96) (76, 98)]
```

In [22]:

```
faces = np.random.randint(0,100,(10,3))
faces
```

Out[22]:

```
array([[ 0, 22,  4],
       [64, 29, 36],
       [93, 26, 73],
       [42, 27, 90],
       [91, 56, 36],
       [43, 82, 56],
       [47, 91,  0],
       [79, 90, 52],
       [82,  2, 52],
       [32, 53, 16]])
```

In [24]:

```
faces = np.random.randint(0,100,(10,3))
F = np.roll(faces.repeat(2,axis=1),-1,axis=1)
F = F.reshape(len(F)*3,2)
F
```

Out[24]:

```
array([[75, 92],
       [92, 99],
       [99, 75],
       [67, 88],
       [88,  0],
       [ 0, 67],
       [33,  6],
       [ 6,  2],
       [ 2, 33],
       [15, 38],
       [38, 26],
       [26, 15],
       [83, 37],
       [37, 21],
       [21, 83],
       [66, 94],
       [94, 53],
       [53, 66],
       [86, 59],
       [59, 77],
       [77, 86],
       [92, 20],
       [20, 23],
       [23, 92],
       [26, 34],
       [34, 66],
       [66, 26],
       [46, 88],
       [88, 87],
       [87, 46]])
```

Given an array C that is a bincount, how to produce an array A such that np.bincount(A) == C?

In [27]:

```
C = np.bincount([1,1,2,3,4,4,6])
A = np.repeat(np.arange(len(C)), C)
print(A)
```

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

In [29]:

```
C = np.bincount([1,1,2,3,4,4,6])
len(C)
```

Out[29]:

```
7
```

How to compute averages using a sliding window over an array(Doubt)

In [30]:

```
def moving_average(a, n=3) :
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n
Z = np.arange(20)
print(moving_average(Z, n=3))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.]
```

In [31]:

```
def moving_average(a, n=3)
print(moving_average(a, n=3))
```

```
  File "<ipython-input-31-a178e42a6066>", line 1
    def moving_average(a, n=3)
                               ^
SyntaxError: invalid syntax
```

Consider a one-dimensional array Z, build a two-dimensional array whose first row is (Z[0],Z[1],Z[2]) and each subsequent row is shifted by 1 (last row should be (Z[-3],Z[-2],Z[-1]) (Dbt)

In [32]:

```
def rolling(a, window):
    shape = (a.size - window + 1, window)
    strides = (a.itemsize, a.itemsize)
    return stride_tricks.as_strided(a, shape=shape, strides=strides)
Z = rolling(np.arange(10), 3)
print(Z)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-32-a5d4231ab45e> in <module>
      3     strides = (a.itemsize, a.itemsize)
      4     return stride_tricks.as_strided(a, shape=shape, strides=stride
s)
----> 5 Z = rolling(np.arange(10), 3)
      6 print(Z)

<ipython-input-32-a5d4231ab45e> in rolling(a, window)
      2     shape = (a.size - window + 1, window)
      3     strides = (a.itemsize, a.itemsize)
----> 4     return stride_tricks.as_strided(a, shape=shape, strides=stride
s)
      5 Z = rolling(np.arange(10), 3)
      6 print(Z)

NameError: name 'stride_tricks' is not defined
```

In [33]:

```
x = np.zeros(10)
x
```

Out[33]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [34]:

```
x = np.random.normal(size=10000) + np.arange(10000)
x
```

Out[34]:

```
array([-6.46921535e-01,  4.78432051e-01,  3.99046479e+00, ...,
        9.99683232e+03,  9.99763114e+03,  9.99814179e+03])
```

In [36]:

```
shape = (a.size - window + 1, window)
shape
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-36-3b4d0fdb4684> in <module>
----> 1 shape = (a.size - window + 1, window)
      2 shape

NameError: name 'a' is not defined
```

How to negate a boolean, or to change the sign of a float inplace

In [41]:

```
Z = np.random.randint(0,2,100)
np.logical_not(Z, out=Z)
Z

#Z = np.random.uniform(-1.0,1.0,100)
#np.negative(arr, out=arr)
```

Out[41]:

```
array([0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0])
```

In [39]:

```
A = np.random.randint(0,2,5)
A
```

Out[39]:

```
array([0, 0, 0, 0, 1])
```

In [42]:

```
Z = np.random.randint(0,2,100)
Z
```

Out[42]:

```
array([1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0])
```

In [43]:

```
Z = np.random.uniform(-1.0,1.0,100)
np.negative(Z, out=Z)
Z
```

Out[43]:

```
array([-0.89595163, -0.79140102, -0.68351896, -0.88830483,  0.34492942,
       -0.15580163,  0.2498463 ,  0.00863153, -0.06589134,  0.67524672,
        0.2545885 , -0.31044735,  0.44931535, -0.53697688,  0.37009339,
        0.20206281,  0.02296024, -0.48046882,  0.460329  , -0.15324558,
        0.16745814, -0.62862587,  0.63099265,  0.04615883,  0.02451888,
       -0.20880829, -0.86968585,  0.19918381, -0.73622352,  0.19661898,
       -0.41663925, -0.75405898,  0.66105375, -0.92868501, -0.52372114,
        0.14218881, -0.18074783, -0.9323417 , -0.24205218,  0.24281548,
        0.98156181,  0.4057408 ,  0.5554252 ,  0.80210054,  0.88712584,
       -0.26957127,  0.70517663,  0.90357748,  0.49217747, -0.12443021,
       -0.89179557,  0.43409303, -0.07186447, -0.0812468 ,  0.96573225,
       -0.43365313, -0.75499879, -0.76825093,  0.7840027 , -0.07234765,
        0.7234798 , -0.65308287, -0.44997151,  0.0389286 ,  0.46702399,
        0.64597229,  0.16561677, -0.94869851, -0.59761009, -0.29860795,
        0.61653198,  0.62283793,  0.6561587 , -0.3336686 ,  0.50222372,
       -0.99835677, -0.33774469,  0.87032993, -0.00948372, -0.52886156,
        0.34004417,  0.2002819 ,  0.59776397, -0.52663112,  0.96168161,
        0.87932267, -0.69597149,  0.90105645, -0.01945102, -0.79853833,
        0.05116504,  0.78364718, -0.97387076, -0.77378859,  0.95262408,
       -0.53553706, -0.62904616, -0.0755544 , -0.45987273, -0.17798722])
```

Consider 2 sets of points P0,P1 describing lines (2d) and a point p, how to compute distance from p to each line i (P0[i],P1[i])

In [44]:

```
def distance(P0, P1, p):
    T = P1 - P0
    L = (T**2).sum(axis=1)
    U = -((P0[:,0]-p[...,0])*T[:,0] + (P0[:,1]-p[...,1])*T[:,1]) / L
    U = U.reshape(len(U),1)
    D = P0 + U*T - p
    return np.sqrt((D**2).sum(axis=1))

P0 = np.random.uniform(-10,10,(10,2))
P1 = np.random.uniform(-10,10,(10,2))
p  = np.random.uniform(-10,10,( 1,2))
print(distance(P0, P1, p))
```

```
[ 2.80287819  8.60769189  1.53791877  6.85949174 13.59451948  7.16871274
  3.72273682 16.27266646  3.38895062  8.61048034]
```

In [45]:

```
L = (T**2).sum(axis=1)
L
```

```
--------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-45-cf3afeac41b8> in <module>
----> 1 L = (T**2).sum(axis=1)
      2 L

NameError: name 'T' is not defined
```

In [46]:

```
P0 = np.random.uniform(-10,10,(10,2))
P0
```

Out[46]:

```
array([[ 7.47472024, -1.04769704],
       [ 0.0350174 ,  5.02875755],
       [-6.22276782,  7.76112941],
       [ 2.13537571,  9.13174839],
       [-8.00387277, -3.38215681],
       [ 2.77803269,  2.35798283],
       [-7.15977614,  3.81918289],
       [-7.21405589,  9.06367597],
       [ 6.49571129,  2.13601404],
       [-0.81956477, -8.77203339]])
```

In [51]:

```
z=np.sum(axis=1)
z
```

```
--------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-51-daa8b2562453> in <module>
----> 1 z=np.sum(axis=1)
      2 z

TypeError: sum() missing 1 required positional argument: 'a'
```

In [55]:

```
np.sum([[1, 2], [3, 5]], axis=0)
```

Out[55]:

```
array([4, 7])
```

In [58]:

```
np.sum([[1, 2], [3, 5]], axis=1)
```

Out[58]:

```
array([3, 8])
```

Consider 2 sets of points P0,P1 describing lines (2d) and a set of points P, how to compute distance from each point j (P[j]) to each line i (P0[i],P1[i])

In [61]:

```
def distance(P0, P1, p):
    T = P1 - P0
    L = (T**2).sum(axis=1)
    U = -((P0[:,0]-p[...,0])*T[:,0] + (P0[:,1]-p[...,1])*T[:,1]) / L
    U = U.reshape(len(U),1)
    D = P0 + U*T - p
    return np.sqrt((D**2).sum(axis=1))
P0 = np.random.uniform(-10, 10, (10,2))
P1 = np.random.uniform(-10,10,(10,2))
p = np.random.uniform(-10, 10, (10,2))
print np.array([distance(P0,P1,p_i) for p_i in p])
```

```
  File "<ipython-input-61-a95b1671db16>", line 11
    print np.array([distance(P0,P1,p_i) for p_i in p])
           ^
SyntaxError: invalid syntax
```

Consider an arbitrary array, write a function that extract a subpart with a fixed shape and centered on a given element (pad with a fill value when necessary)

In [62]:

```
Z = np.random.randint(0,10,(10,10))
shape = (5,5)
fill  = 0
position = (1,1)

R = np.ones(shape, dtype=Z.dtype)*fill
P  = np.array(list(position)).astype(int)
Rs = np.array(list(R.shape)).astype(int)
Zs = np.array(list(Z.shape)).astype(int)

R_start = np.zeros((len(shape),)).astype(int)
R_stop  = np.array(list(shape)).astype(int)
Z_start = (P-Rs//2)
Z_stop  = (P+Rs//2)+Rs%2

R_start = (R_start - np.minimum(Z_start,0)).tolist()
Z_start = (np.maximum(Z_start,0)).tolist()
R_stop = np.maximum(R_start, (R_stop - np.maximum(Z_stop-Zs,0))).tolist()
Z_stop = (np.minimum(Z_stop,Zs)).tolist()

r = [slice(start,stop) for start,stop in zip(R_start,R_stop)]
z = [slice(start,stop) for start,stop in zip(Z_start,Z_stop)]
R[r] = Z[z]
print(Z)
print(R)
```

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

C:\Users\jaya\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Future
Warning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wil
l be interpreted as an array index, `arr[np.array(seq)]`, which will resul
t either in an error or a different result.
```

In [63]:

```
Z = np.random.randint(0,10,(10,10))
Z
```

Out[63]:

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

Consider an array Z = [1,2,3,4,5,6,7,8,9,10,11,12,13,14], how to generate an array R = [[1,2,3,4], [2,3,4,5], [3,4,5,6], ..., [11,12,13,14]] (Doubt)

In [71]:

```
Z = np.arange(1,15,dtype=int)
R = Z.stride_tricks.as_strided(Z,(11,4),(4,4))
print(R)
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-71-be2680e91ecf> in <module>
      1 Z = np.arange(1,15,dtype=int)
----> 2 R = Z.stride_tricks.as_strided(Z,(11,4),(4,4))
      3 print(R)

AttributeError: 'numpy.ndarray' object has no attribute 'stride_tricks'
```

Compute a matrix rank

In [74]:

```
Z = np.random.uniform(0,1,(10,10))
U, S, V = np.linalg.svd(Z) # Singular Value Decomposition
rank = np.sum(S > 1e-10)
rank
```

Out[74]:

```
10
```

How to find the most frequent value in an array

In [75]:

```
Z = np.random.randint(0,10,50)
print(np.bincount(Z).argmax())
```

```
8
```

In [76]:

```
Z = np.random.randint(0,10,50)
Z
```

Out[76]:

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

Extract all the contiguous 3x3 blocks from a random 10x10 matrix

In [77]:

```
Z = np.random.randint(0,5,(10,10))
n = 3
i = 1 + (Z.shape[0]-3)
j = 1 + (Z.shape[1]-3)
C = stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.strides + Z.strides)
print(C)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-77-24dfd8e3ff0a> in <module>
      3 i = 1 + (Z.shape[0]-3)
      4 j = 1 + (Z.shape[1]-3)
----> 5 C = stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.stri
des + Z.strides)
      6 print(C)

NameError: name 'stride_tricks' is not defined
```

Create a 2D array subclass such that Z[i,j] == Z[j,i]

In [78]:

```
 def __setitem__(self, (i,j), value):
        super(Symetric, self).__setitem__((i,j), value)
        super(Symetric, self).__setitem__((j,i), value)

def symetric(Z):
    return np.asarray(Z + Z.T - np.diag(Z.diagonal())).view(Symetric)

S = symetric(np.random.randint(0,10,(5,5)))
S[2,3] = 42
print(S)
```

```
  File "<ipython-input-78-c68d7795eda5>", line 1
    def __setitem__(self, (i,j), value):
                          ^
SyntaxError: invalid syntax
```

Consider a set of p matrices wich shape (n,n) and a set of p vectors with shape (n,1). How to compute the sum of of the p matrix products at once? (result has shape (n,1)

In [79]:

```
p, n = 10, 20
M = np.ones((p,n,n))
V = np.ones((p,n,1))
S = np.tensordot(M, V, axes=[[0, 2], [0, 1]])
print(S)
```

```
[[200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]
 [200.]]
```

In [81]:

```
p, n = 10, 20
V = np.ones((p,n,1))
V
```

Out[81]:

```
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.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.]],

        [[1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.]],

        [[1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.]],

        [[1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.]],

        [[1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
         [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.]],

       [[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.]],

       [[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.]],

       [[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [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 [84]:

```
V = np.ones((5,3,2))
V
```

Out[84]:

```
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.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

Consider a 16x16 array, how to get the block-sum (block size is 4x4)

In [99]:

```
Z = np.ones([16,16])
k = 4
S = np.add.reduceat(np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0),
                                       np.arange(0, Z.shape[1], k), axis=1)
S
u=np.arange(0, Z.shape[0], k)
print(u)
```

```
[ 0  4  8 12]
```

In [90]:

```
Z = np.ones([16,16])
Z
```

Out[90]:

```
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., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 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 [97]:

```
x=np.multiply.reduceat([0,1,2,3,4,5,6,7],[0,5])
x
```

Out[97]:

```
array([  0, 210], dtype=int32)
```

How to implement the Game of Life using numpy arrays

In [102]:

```
def iterate(Z):
    # Count neighbours
    N = (Z[0:-2,0:-2] + Z[0:-2,1:-1] + Z[0:-2,2:] +
         Z[1:-1,0:-2]                + Z[1:-1,2:] +
         Z[2:  ,0:-2] + Z[2:  ,1:-1] + Z[2:  ,2:])
    Print(N)
    # Apply rules
    birth = (N==3) & (Z[1:-1,1:-1]==0)
    survive = ((N==2) | (N==3)) & (Z[1:-1,1:-1]==1)
    Z[...] = 0
    Z[1:-1,1:-1][birth | survive] = 1
    return Z

Z = np.random.randint(0,2,(50,50))
for i in range(100): Z = iterate(Z)
```

```
--------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-102-08a60ed1500f> in <module>
     13
     14 Z = np.random.randint(0,2,(50,50))
---> 15 for i in range(100): Z = iterate(Z)

<ipython-input-102-08a60ed1500f> in iterate(Z)
      4          Z[1:-1,0:-2]                + Z[1:-1,2:] +
      5          Z[2:  ,0:-2] + Z[2:  ,1:-1] + Z[2:  ,2:])
----> 6     Print(N)
      7     # Apply rules
      8     birth = (N==3) & (Z[1:-1,1:-1]==0)

NameError: name 'Print' is not defined
```

How to get the n largest values of an array

In [107]:

```
Z = np.arange(10000)
k=np.random.shuffle(Z)
n = 5
print (Z[np.argpartition(-Z,n)[:n]])
```

```
[9998 9997 9999 9995 9996]
```

Given an arbitrary number of vectors, build the cartesian product (every combinations of every item)

In [108]:

```
def cartesian(arrays):
    arrays = [np.asarray(a) for a in arrays]
    shape = (len(x) for x in arrays)

    ix = np.indices(shape, dtype=int)
    ix = ix.reshape(len(arrays), -1).T

    for n, arr in enumerate(arrays):
        ix[:, n] = arrays[n][ix[:, n]]

    return ix

print (cartesian(([1, 2, 3], [4, 5], [6, 7])))
```

```
[[1 4 6]
 [1 4 7]
 [1 5 6]
 [1 5 7]
 [2 4 6]
 [2 4 7]
 [2 5 6]
 [2 5 7]
 [3 4 6]
 [3 4 7]
 [3 5 6]
 [3 5 7]]
```

In [117]:

```
grid = np.indices((2, 3))
grid.shape
```

Out[117]:

```
(2, 2, 3)
```

In [116]:

```
y= np.random.randint(0,5,(10,10))
y
```

Out[116]:

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

In [118]:

```
x = np.arange(20).reshape(5, 4)
row, col = np.indices((2, 3))
x[row, col]
```

Out[118]:

```
array([[0, 1, 2],
       [4, 5, 6]])
```

In [119]:

```
x = np.arange(20).reshape(5, 4)
x
```

Out[119]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

How to create a record array from a regular array

In [121]:

```
Z = np.array([("Hello", 2.5, 3),
              ("World", 3.6, 2)])
R = np.core.records.fromarrays(Z.T,
                               names='col1, col2, col3',
                               formats = 'S8, f8, i8')
print(R)
```

```
[(b'Hello', 2.5, 3) (b'World', 3.6, 2)]
```

Consider a large vector Z, compute Z to the power of 3 using 3 different methods

In [123]:

```
x = np.random.rand(5e7)
%timeit np.power(x,3)
#1 loops, best of 3: 574 ms per loop

#%timeit x*x*x
#1 loops, best of 3: 429 ms per loop

#%timeit np.einsum('i,i,i->i',x,x,x)
#1 loops, best of 3: 244 ms per loop
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-123-128a523ecad7> in <module>
----> 1 x = np.random.rand(5e7)
      2 get_ipython().run_line_magic('timeit', 'np.power(x,3)')
      3 #1 loops, best of 3: 574 ms per loop
      4 
      5 #%timeit x*x*x

mtrand.pyx in mtrand.RandomState.rand()

mtrand.pyx in mtrand.RandomState.random_sample()

mtrand.pyx in mtrand.cont0_array()

TypeError: 'float' object cannot be interpreted as an integer
```

In [124]:

```
x = np.random.rand(5e7)
x
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-124-348b7179f092> in <module>
----> 1 x = np.random.rand(5e7)
      2 x

mtrand.pyx in mtrand.RandomState.rand()

mtrand.pyx in mtrand.RandomState.random_sample()

mtrand.pyx in mtrand.cont0_array()

TypeError: 'float' object cannot be interpreted as an integer
```

Consider two arrays A and B of shape (8,3) and (2,2). How to find rows of A that contain elements of each row of B regardless of the order of the elements in B

In [125]:

```
A = np.random.randint(0,5,(8,3))
B = np.random.randint(0,5,(2,2))
C = (A[..., np.newaxis, np.newaxis] == B)
rows = (C.sum(axis=(1,2,3)) >= B.shape[1]).nonzero()[0]
print(rows)
```

```
[0 2 4 5]
```

In [126]:

```
A = np.random.randint(0,5,(8,3))
B = np.random.randint(0,5,(2,2))
print(A)
print(B)
```

```
[[4 4 3]
 [0 0 0]
 [0 0 4]
 [1 2 3]
 [0 0 1]
 [3 3 3]
 [2 2 2]
 [0 4 0]]
[[2 1]
 [3 3]]
```

Considering a 10x3 matrix, extract rows with unequal values (e.g. [2,2,3])

In [127]:

```
Z = np.random.randint(0,5,(10,3))
E = np.logical_and.reduce(Z[:,1:] == Z[:,:-1], axis=1)
U = Z[~E]
print(Z)
print(U)
```

```
[[2 3 0]
 [3 4 3]
 [0 2 2]
 [2 1 2]
 [4 4 3]
 [4 0 3]
 [0 2 2]
 [4 2 1]
 [0 2 1]
 [0 1 4]]
[[2 3 0]
 [3 4 3]
 [0 2 2]
 [2 1 2]
 [4 4 3]
 [4 0 3]
 [0 2 2]
 [4 2 1]
 [0 2 1]
 [0 1 4]]
```

Convert a vector of ints into a matrix binary representation

In [128]:

```
I = np.array([0, 1, 2, 3, 15, 16, 32, 64, 128])
B = ((I.reshape(-1,1) & (2**np.arange(8))) != 0).astype(int)
print(B[:,::-1])
```

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

Given a two dimensional array, how to extract unique rows

In [129]:

```
Z = np.random.randint(0,2,(6,3))
T = np.ascontiguousarray(Z).view(np.dtype((np.void, Z.dtype.itemsize * Z.shape[1])))
_, idx = np.unique(T, return_index=True)
uZ = Z[idx]
print(uZ)
```

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

Considering 2 vectors A & B, write the einsum equivalent of inner, outer, sum, and mul function

In [130]:

```
A = np.array([0, 1, 2])
B = np.array([[ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11]])
np.einsum('i->', A)       # np.sum(A)
np.einsum('i,i->i', A, B) # A * B
np.einsum('i,i', A, B)    # np.inner(A, B)
np.einsum('i,j', A, B)    # np.outer(A, B)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-130-77f536390331> in <module>
      4               [ 8,  9, 10, 11]])
      5 np.einsum('i->', A)       # np.sum(A)
----> 6 np.einsum('i,i->i', A, B) # A * B
      7 np.einsum('i,i', A, B)    # np.inner(A, B)
      8 np.einsum('i,j', A, B)    # np.outer(A, B)

~\Anaconda3\lib\site-packages\numpy\core\einsumfunc.py in einsum(*operand
s, **kwargs)
   1226     # If no optimization, run pure einsum
   1227     if optimize_arg is False:
-> 1228         return c_einsum(*operands, **kwargs)
   1229
   1230     valid_einsum_kwargs = ['out', 'dtype', 'order', 'casting']

ValueError: operand has more dimensions than subscripts given in einstein
 sum, but no '...' ellipsis provided to broadcast the extra dimensions.
```

Considering a path described by two vectors (X,Y), how to sample it using equidistant samples

In [132]:

```
phi = np.arange(0, 10*np.pi, 0.1)
a = 1
x = a*phi*np.cos(phi)
y = a*phi*np.sin(phi)

dr = (np.diff(x)**2 + np.diff(y)**2)**.5 # segment lengths
r = np.zeros_like(x)
r[1:] = np.cumsum(dr)                # integrate path
r_int = np.linspace(0, r.max(), 200) # regular spaced path
x_int = np.interp(r_int, r, x)       # integrate path
y_int = np.interp(r_int, r, y)
print(x_int,y_int)
```

```
[ 0.00000000e+00 -3.73131229e-01 -2.59817608e+00 -3.26212050e+00
 -2.18442687e+00 -2.98929946e-02  2.42923642e+00  4.54913599e+00
  5.92318348e+00  6.35117933e+00  5.82369277e+00  4.46259540e+00
  2.47320794e+00  1.09577220e-01 -2.36575300e+00 -4.71261671e+00
 -6.72701769e+00 -8.25541575e+00 -9.18486120e+00 -9.46381505e+00
 -9.11085788e+00 -8.12875279e+00 -6.63306046e+00 -4.69271059e+00
 -2.44736165e+00 -2.05444585e-02  2.46101146e+00  4.86841760e+00
  7.08937968e+00  9.02539126e+00  1.05948609e+01  1.17357250e+01
  1.24068974e+01  1.25885805e+01  1.22815267e+01  1.15053927e+01
  1.02963689e+01  8.70429550e+00  6.78948686e+00  4.61716636e+00
  2.25853448e+00 -1.98731680e-01 -2.68040566e+00 -5.11543300e+00
 -7.41973991e+00 -9.53891040e+00 -1.14237629e+01 -1.29919305e+01
 -1.42355069e+01 -1.51243232e+01 -1.56061571e+01 -1.57219415e+01
 -1.54217066e+01 -1.47579136e+01 -1.37255236e+01 -1.23634834e+01
 -1.07024632e+01 -8.78327367e+00 -6.65029558e+00 -4.35514246e+00
 -1.94290275e+00  5.28038914e-01  3.00985904e+00  5.45450275e+00
  7.80093594e+00  1.00179639e+01  1.20595602e+01  1.38732623e+01
  1.54564938e+01  1.67497894e+01  1.77435802e+01  1.84439878e+01
  1.87990820e+01  1.88271003e+01  1.85472755e+01  1.79378850e+01
  1.70105456e+01  1.58085777e+01  1.43501808e+01  1.26222364e+01
  1.06905445e+01  8.58498641e+00  6.33562529e+00  3.96179025e+00
  1.51802643e+00 -9.61699044e-01 -3.44349301e+00 -5.88951759e+00
 -8.26006582e+00 -1.05270865e+01 -1.26610269e+01 -1.46343020e+01
 -1.64206033e+01 -1.79708499e+01 -1.92894561e+01 -2.03605967e+01
 -2.11716587e+01 -2.17133504e+01 -2.19797677e+01 -2.19626531e+01
 -2.16476737e+01 -2.10622561e+01 -2.02145809e+01 -1.91158467e+01
 -1.77800930e+01 -1.62239930e+01 -1.44666197e+01 -1.25291882e+01
 -1.04347777e+01 -8.20773276e+00 -5.86978851e+00 -3.45563299e+00
 -9.92556869e-01  1.49191203e+00  3.97031733e+00  6.41557503e+00
  8.80126359e+00  1.11019001e+01  1.32931995e+01  1.53523150e+01
  1.72580567e+01  1.89910866e+01  2.05340895e+01  2.18719167e+01
  2.29917039e+01  2.38829603e+01  2.45376309e+01  2.49501314e+01
  2.51173551e+01  2.50386548e+01  2.47157984e+01  2.41529019e+01
  2.33563391e+01  2.23346314e+01  2.10983193e+01  1.96598169e+01
  1.80332532e+01  1.62343003e+01  1.42799934e+01  1.21885425e+01
  9.97914013e+00  7.67176574e+00  5.28699023e+00  2.84441658e+00
  3.68190102e-01 -2.11846865e+00 -4.59426061e+00 -7.03829202e+00
 -9.43024209e+00 -1.17505186e+01 -1.39803999e+01 -1.61021634e+01
 -1.80991985e+01 -1.99561058e+01 -2.16587804e+01 -2.31850397e+01
 -2.45166960e+01 -2.56568097e+01 -2.65976904e+01 -2.73332424e+01
 -2.78589555e+01 -2.81718833e+01 -2.82706110e+01 -2.81552119e+01
 -2.78066635e+01 -2.72341677e+01 -2.64578863e+01 -2.54844118e+01
 -2.43214986e+01 -2.29779690e+01 -2.14636172e+01 -1.97837046e+01
 -1.79341987e+01 -1.59545785e+01 -1.38585628e+01 -1.16602783e+01
 -9.37416511e+00 -7.01488958e+00 -4.59184632e+00 -2.12995331e+00
  3.50691071e-01  2.83498918e+00  5.30813450e+00  7.75565814e+00
  1.01522531e+01  1.24813227e+01  1.47337756e+01  1.68973642e+01
  1.89604991e+01  2.09071945e+01  2.26943693e+01  2.43420179e+01
  2.58422998e+01  2.71881423e+01  2.83731992e+01  2.93376609e+01
  3.01222558e+01  3.07280750e+01  3.11531219e+01  3.13960177e+01] [ 0.0000
0000e+00  1.74026724e+00  9.81816584e-01 -1.34251287e+00
 -3.53191891e+00 -4.70449474e+00 -4.59573427e+00 -3.33831870e+00
 -1.28956083e+00  1.14234685e+00  3.55645601e+00  5.62188218e+00
  7.09389488e+00  7.82951803e+00  7.78700678e+00  6.99685666e+00
  5.55535240e+00  3.60417006e+00  1.30506373e+00 -1.15819722e+00
 -3.61328132e+00 -5.89130465e+00 -7.87065053e+00 -9.41689057e+00
 -1.04714836e+01 -1.09903787e+01 -1.09354306e+01 -1.03330912e+01
 -9.22690166e+00 -7.67489830e+00 -5.75257980e+00 -3.54832201e+00
 -1.15854355e+00  1.31709956e+00  3.78023622e+00  6.13780326e+00
  8.30540149e+00  1.02098724e+01  1.17910419e+01  1.29972950e+01
  1.37774249e+01  1.41319792e+01  1.40585758e+01  1.35636816e+01
  1.26345244e+01  1.13409083e+01  9.72216360e+00  7.79464305e+00
  5.64544223e+00  3.32503222e+00  8.88107041e-01 -1.59329248e+00
 -4.05992507e+00 -6.45313746e+00 -8.71327144e+00 -1.07900854e+01
 -1.26380653e+01 -1.42147006e+01 -1.54892299e+01 -1.64390858e+01
 -1.70351433e+01 -1.72938948e+01 -1.71685631e+01 -1.67219979e+01
 -1.59044166e+01 -1.47824153e+01 -1.33665280e+01 -1.16678134e+01
 -9.75246391e+00 -7.63091465e+00 -5.35345831e+00 -2.96905840e+00
 -5.09631905e-01  1.97507060e+00  4.44430515e+00  6.85344809e+00
  9.15903461e+00  1.13337844e+01  1.33467125e+01  1.51336120e+01
  1.66966787e+01  1.80162922e+01  1.90746708e+01  1.98127969e+01
  2.02634499e+01  2.04221761e+01  2.02877631e+01  1.98449702e+01
  1.90974465e+01  1.80799531e+01  1.68069397e+01  1.52959771e+01
  1.35665801e+01  1.16227312e+01  9.51601869e+00  7.27388893e+00
  4.92519240e+00  2.49981337e+00  2.82932820e-02 -2.45817963e+00
 -4.92460367e+00 -7.34030894e+00 -9.67635928e+00 -1.19050601e+01
 -1.40002645e+01 -1.59376556e+01 -1.76950019e+01 -1.92523833e+01
 -2.05923861e+01 -2.16995543e+01 -2.25478860e+01 -2.31432552e+01
 -2.34800725e+01 -2.35556687e+01 -2.33702741e+01 -2.29269662e+01
 -2.22315865e+01 -2.12926288e+01 -2.01211000e+01 -1.87303570e+01
 -1.71359212e+01 -1.53552740e+01 -1.34076356e+01 -1.13137308e+01
 -9.09554383e+00 -6.77606668e+00 -4.37904252e+00 -1.92870828e+00
  5.50461201e-01  3.03400451e+00  5.49771780e+00  7.91788934e+00
  1.02715222e+01  1.25365431e+01  1.46919953e+01  1.67182148e+01
  1.85969872e+01  2.03116866e+01  2.18473923e+01  2.31909872e+01
  2.43312337e+01  2.52588296e+01  2.59664444e+01  2.64391214e+01
  2.66748505e+01  2.66796333e+01  2.64545680e+01  2.60026765e+01
  2.53288265e+01  2.44396400e+01  2.33433894e+01  2.20498837e+01
  2.05703459e+01  1.89172843e+01  1.71043584e+01  1.51396391e+01
  1.30387952e+01  1.08294252e+01  8.52903617e+00  6.15551380e+00
  3.72697869e+00  1.26164946e+00 -1.22228693e+00 -3.70679651e+00
 -6.17014862e+00 -8.59028675e+00 -1.09516293e+01 -1.32379536e+01
 -1.54338179e+01 -1.75246263e+01 -1.94966815e+01 -2.13317519e+01
 -2.29952001e+01 -2.44987532e+01 -2.58335652e+01 -2.69919481e+01
 -2.79673583e+01 -2.87543764e+01 -2.93186910e+01 -2.96679958e+01
 -2.98181354e+01 -2.97693731e+01 -2.95229769e+01 -2.90811595e+01
 -2.84133854e+01 -2.75430665e+01 -2.64934034e+01 -2.52713717e+01
 -2.38845608e+01 -2.23353782e+01 -2.06051319e+01 -1.87442619e+01
 -1.67635423e+01 -1.46739620e+01 -1.24866789e+01 -1.01933401e+01
 -7.83474328e+00 -5.42495146e+00 -2.97607515e+00 -5.00072086e-01]
```

Given an integer n and a 2D array X, select from X the rows which can be interpreted as draws from a multinomial distribution with n degrees, i.e., the rows which only contain integers and which sum to n

In [2]:

```
import numpy as np
X = np.asarray([[1.0, 0.0, 3.0, 8.0],
                [2.0, 0.0, 1.0, 1.0],
                [1.5, 2.5, 1.0, 0.0]])
n = 4
M = np.logical_and.reduce(np.mod(X, 1) == 0, axis=-1)
M &= (X.sum(axis=-1) == n)
print(X[M])
```

```
[[2. 0. 1. 1.]]
```

In [ ]: