Numpy(2)

February 23, 2020

Programación para la Computación cientifica

Taller Evaluado de Numpy

Nico Patalagua

1. Import the numpy package under the name `np`.

[0]:
```
import numpy as np
```

2. Print the numpy version and the configuration.

[3]:
```
np.version.version
np.show_config()
```

```
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
```

3. Create a null vector of size 10.

```
[4]: x = np.zeros(10)
     print(x)
```

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

4. How to find the memory size of any array.

```
[5]: x= np.zeros((10,10))
     print("%d bytes" % (x.size * x.itemsize))
```

```
800 bytes
```

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

```
[6]: %run `python -c "import numpy; numpy.info(numpy.add)"`
```

```
ERROR:root:File `'`python.py'` not found.
```

```
[0]: np.info(np.add)
```

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

```
[11]: x= np.zeros(10)
      x[4] = 1
      print(x)
```

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

7. Create a vector with values ranging from 10 to 49.

```
[12]: x = np.arange(10,50)
      print(x)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

8. Reverse a vector (first element becomes last).

```
[0]: x = np.arange(50)
     x = x[::-1]
     print(x)
```

```
[49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26
 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2
  1  0]
```

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

[13]:
```
x =  np.arange(0, 9).reshape(3,3)
print(x)
```

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

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

[14]:
```
x = np.nonzero([1,2,0,0,4,0])
print(x)
```

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

11. Create a 3x3 identity matrix.

[15]:
```
x = np.eye(3)
print(x)
```

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

12. Create a 3x3x3 array with random values.

[16]:
```
x = np.random.random((3,3,3))
print(x)
```

```
[[[0.57735165 0.13833089 0.30827209]
  [0.36552315 0.32293258 0.0426784 ]
  [0.71007951 0.67311445 0.71098171]]

 [[0.84919953 0.21821926 0.79234011]
  [0.91096478 0.17764479 0.7413266 ]
  [0.08419017 0.33938528 0.39694713]]

 [[0.65839921 0.66264685 0.40583485]
  [0.90937981 0.55666961 0.55895311]
  [0.09508592 0.03643199 0.08781656]]]
```

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

[17]:
```
x = np.random.random((10,10))
xmin, xmax = x.min(), x.max()
print(xmin, xmax)
```

```
0.0011718160683460432 0.999744743587465
```

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

[18]:
```
x = np.random.random(30)
x1 = x.mean()
print(x1)
```

```
0.448837845660199
```

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

[19]:
```
x = np.ones((10,10))
x[1:-1,1:-1] = 0
print(x)
```

```
[[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.]]
```

16. How to add a border (filled with 0's) around an existing array?.

[24]:
```
x= np.ones((5,5))
x= np.pad(x, pad_width=1, mode='constant', constant_values=0)
print(x)
```

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

17. What is the result of the following expression?.

[25]:
```
print(0 * np.nan)
print(np.nan == np.nan)
print(np.inf > np.nan)
print(np.nan - np.nan)
print(np.nan in set([np.nan]))
print(0.3 == 3 * 0.1)
```

```
nan
False
False
nan
```

```
True
False
```

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

```
[26]: x= np.diag(1+np.arange(4),k=-1)
      print(x)
```

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

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

```
[27]: x = np.zeros((8,8),dtype=int)
      x[1::2,::2] = 1
      x[::2,1::2] = 1
      print(x)
```

```
[[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]]
```

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

```
[28]: print(np.unravel_index(99,(6,7,8)))
```

```
(1, 5, 3)
```

21. Create a checkerboard 8x8 matrix using the tile function.

```
[29]: x = np.tile( np.array([[0,1],[1,0]]), (4,4))
      print(x)
```

```
[[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]]
```

22. Normalize a 5x5 random matrix.

```
[30]: x = np.random.random((5,5))
      x = (x - np.mean (x)) / (np.std (x))
      print(x)
```

```
[[-1.14886925 -1.25194505  1.11245463 -0.73728405  0.55572055]
 [-0.13126056  1.06377288  1.46118815  1.48071171 -1.11952055]
 [-1.31799727  1.6558928   0.71259418 -0.46956235 -0.87504612]
 [ 1.21806688  0.09479064  1.06352032 -0.39178823 -0.30580471]
 [-1.59731458  0.56399963 -1.00077051 -0.75145704  0.1159079 ]]
```

23. Create a custom dtype that describes a color as four unsigned bytes (RGBA).

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

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  after removing the cwd from sys.path.
```

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

```
[33]: x = np.dot(np.ones((5,3)), np.ones((3,2)))
      print(x)
```

```
[[3. 3.]
 [3. 3.]
 [3. 3.]
 [3. 3.]
 [3. 3.]]
```

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

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

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

26. What is the output of the following script?.

```
[37]: print(sum(range(5),-1))
      from numpy import *
      print(sum(range(5),-1))
```

```
10
10
```

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

```
[42]: Z = np.arange(11)
      Z**Z
      2 << Z >> 2
      Z <- Z
      1j*Z
      Z/1/1
      Z<Z>Z
```

```
          ␣
   ↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
   ↪last)

        <ipython-input-42-9645ae9e3bce> in <module>()
          5 1j*Z
          6 Z/1/1
    ----> 7 Z<Z>Z


        ValueError: The truth value of an array with more than one element is␣
   ↪ambiguous. Use a.any() or a.all()
```

28. What are the result of the following expressions?.

```
[43]: print(np.array(0) / np.array(0))
      print(np.array(0) // np.array(0))
      print(np.array([np.nan]).astype(int).astype(float))
```

```
nan
0
[-9.22337204e+18]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: RuntimeWarning:
invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: RuntimeWarning:
divide by zero encountered in floor_divide
```

29. How to round away from zero a float array ?.

```
[44]: x = np.random.uniform(-10,+10,10)
      print (np.copysign(np.ceil(np.abs(x)), x))
```

```
[-1. -4.  9.  6.  2.  6. -2. 10.  2. 10.]
```

30. How to find common values between two arrays?.

```
[46]: x1 = np.random.randint(0,10,10)
      x2 = np.random.randint(0,10,10)
      print(np.intersect1d(x1,x2))
```

```
[2 3 4 7 8 9]
```

31. How to ignore all numpy warnings (not recommended)?.

```
[51]: a = np.seterr(all="ignore")
      x = np.ones(1) / 0
      _ = np.seterr(**a)
      n = np.nonzero([1,2,0,0,4,0])
      print(n)
```

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

32. Is the following expressions true?.

```
[52]: np.sqrt(-1) == np.emath.sqrt(-1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: RuntimeWarning:
invalid value encountered in sqrt
  """Entry point for launching an IPython kernel.
```

```
[52]: False
```

33. How to get the dates of yesterday, today and tomorrow?.

```
[59]: yesterday= np.datetime64('today', 'D') - np.timedelta64(1, 'D')
      today= np.datetime64('today', 'D')
      tomorrow= np.datetime64('today', 'D') + np.timedelta64(1, 'D')
      print("Yesterday "+str(yesterday))
      print("Today "+str(today))
      print("Tomorrow "+str(tomorrow))
```

```
Yesterday 2020-02-22
Today 2020-02-23
Tomorrow 2020-02-24
```

34. How to get all the dates corresponding to the month of July 2016?.

```
[60]: x = np.arange('2016-07', '2016-08', dtype='datetime64[D]')
      print(x)
```

```
['2016-07-01' '2016-07-02' '2016-07-03' '2016-07-04' '2016-07-05'
 '2016-07-06' '2016-07-07' '2016-07-08' '2016-07-09' '2016-07-10'
 '2016-07-11' '2016-07-12' '2016-07-13' '2016-07-14' '2016-07-15'
 '2016-07-16' '2016-07-17' '2016-07-18' '2016-07-19' '2016-07-20'
 '2016-07-21' '2016-07-22' '2016-07-23' '2016-07-24' '2016-07-25'
 '2016-07-26' '2016-07-27' '2016-07-28' '2016-07-29' '2016-07-30'
 '2016-07-31']
```

35. How to compute ((A+B)*(-A/2)) in place (without copy)?.

```
[61]: A = np.ones(3)*1
B = np.ones(3)*2
C = np.ones(3)*3
np.add(A,B,out=B)
np.divide(A,2,out=A)
np.negative(A,out=A)
np.multiply(A,B,out=A)
```

```
[61]: array([-1.5, -1.5, -1.5])
```

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

```
[62]: x = np.random.uniform(0,10,10)
print (x - x%1)
print (np.floor(x))
print (np.ceil(x)-1)
print (x.astype(int))
print (np.trunc(x))
```

```
[5. 2. 2. 1. 0. 1. 9. 9. 7. 3.]
[5. 2. 2. 1. 0. 1. 9. 9. 7. 3.]
[5. 2. 2. 1. 0. 1. 9. 9. 7. 3.]
[5 2 2 1 0 1 9 9 7 3]
[5. 2. 2. 1. 0. 1. 9. 9. 7. 3.]
```

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

```
[63]: x = np.zeros((5,5))
x += np.arange(5)
print(x)
```

```
[[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.]]
```

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

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

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

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

[65]:
```
x = np.linspace(0,1,11,endpoint=False)[1:]
print(x)
```

```
[0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
 0.63636364 0.72727273 0.81818182 0.90909091]
```

40. Create a random vector of size 10 and sort it.

[0]:
```
x = np.random.random(10)
x.sort()
print(x)
```

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

[66]:
```
x=np.arange(10)
np.add.reduce(x)
```

[66]:
```
45
```

42. Consider two random array A and B, check if they are equal.

[68]:
```
A = np.random.randint(0,2,5)
B = np.random.randint(0,2,5)
equal = np.allclose(A,B)
print(equal)
equal = np.array_equal(A,B)
print(equal)
```

```
False
False
```

43. Make an array immutable (read-only).

[0]:
```
x = np.zeros(10)
x.flags.writeable = False
```

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

[75]:
```
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)
```

```
[0.42645217 0.19680646 0.42133394 0.36267599 0.89756795 1.02795053
 0.89662918 0.9705275  0.88509266 0.37801906]
[0.0488522  0.46764661 0.47373292 0.97171099 1.38060464 0.39735547
 0.09068007 1.13970647 1.23061848 0.3622604 ]
```

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

[76]:
```
x = np.random.random(10)
x[x.argmax()] = 0
print(x)
```

```
[0.6028588  0.51925526 0.88419502 0.         0.6890356  0.10237085
 0.60649779 0.23543865 0.66018685 0.74023108]
```

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

[78]:
```
x = np.zeros((5,5), [('x',float),('y',float)])
x['x'], x['y'] = np.meshgrid(np.linspace(0,1,5),np.linspace(0,1,5))
print(x)
```

```
[[(0.  , 0.  ) (0.25, 0.  ) (0.5 , 0.  ) (0.75, 0.  ) (1.  , 0.  )]
 [(0.  , 0.25) (0.25, 0.25) (0.5 , 0.25) (0.75, 0.25) (1.  , 0.25)]
 [(0.  , 0.5 ) (0.25, 0.5 ) (0.5 , 0.5 ) (0.75, 0.5 ) (1.  , 0.5 )]
 [(0.  , 0.75) (0.25, 0.75) (0.5 , 0.75) (0.75, 0.75) (1.  , 0.75)]
 [(0.  , 1.  ) (0.25, 1.  ) (0.5 , 1.  ) (0.75, 1.  ) (1.  , 1.  )]]
```

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

[79]:
```
x = np.arange(8)
y = x + 0.5
C = 1.0 / np.subtract.outer(x, y)
print(np.linalg.det(C))
```

```
3638.163637117973
```

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

[80]:
```
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)
```

```
-128
127
-2147483648
2147483647
-9223372036854775808
9223372036854775807
```

```
-3.4028235e+38
3.4028235e+38
1.1920929e-07
-1.7976931348623157e+308
1.7976931348623157e+308
2.220446049250313e-16
```

49. How to print all the values of an array?.

[85]:
```
np.set_printoptions(threshold=np.inf)
x = np.zeros((4, 4))
print(x)
```

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

50. How to find the closest value (to a given scalar) in a vector?.

[87]:
```
x = np.arange(100)
y = np.random.uniform(0,100)
z = (np.abs(x-y)).argmin()
print(x[z])
```

```
95
```

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

[88]:
```
x = np.zeros(10, [ ('position', [ ('x', float, 1),('y', float, 1)]),
                   ('color',    [ ('r', float, 1),('g', float, 1),
                   ('b', float, 1)])])
print(x)
```

```
[((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.))]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

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

[138]:
```
x = np.random.random((10,2))
a,b = np.atleast_2d(x[:,0], x[:,1])
```

```
c = np.sqrt( (a-a.T)**2 + (b-b.T)**2)
print(c)
import scipy.spatial
x = np.random.random((10,2))
c = scipy.spatial.distance.cdist(x,x)
print(c)
```

```
[[0.         0.56756096 0.58923105 0.09994089 0.40572292 0.31704116
  0.73622451 0.79388905 0.52283401 0.27883913]
 [0.56756096 0.         0.07538426 0.46858642 0.54065469 0.26223301
  0.32525109 0.31062702 0.46147842 0.39891568]
 [0.58923105 0.07538426 0.         0.49305635 0.6045951  0.302748
  0.25022674 0.3637136  0.4096562  0.45115566]
 [0.09994089 0.46858642 0.49305635 0.         0.36244961 0.21733365
  0.65370751 0.69604414 0.47601946 0.20279255]
 [0.40572292 0.54065469 0.6045951  0.36244961 0.         0.3404431
  0.83982531 0.59482425 0.78522255 0.19117146]
 [0.31704116 0.26223301 0.302748   0.21733365 0.3404431  0.
  0.51153784 0.48165071 0.45812131 0.1585802 ]
 [0.73622451 0.32525109 0.25022674 0.65370751 0.83982531 0.51153784
  0.         0.56434966 0.34322994 0.66961636]
 [0.79388905 0.31062702 0.3637136  0.69604414 0.59482425 0.48165071
  0.56434966 0.         0.77075548 0.54610431]
 [0.52283401 0.46147842 0.4096562  0.47601946 0.78522255 0.45812131
  0.34322994 0.77075548 0.         0.59405186]
 [0.27883913 0.39891568 0.45115566 0.20279255 0.19117146 0.1585802
  0.66961636 0.54610431 0.59405186 0.        ]]
[[0.         0.22621142 0.3365521  0.69214086 0.40645044 0.61960969
  0.35784017 0.14283045 0.30919754 0.57817499]
 [0.22621142 0.         0.56269268 0.5919511  0.42762934 0.5703187
  0.5716094  0.31482334 0.51961346 0.76678828]
 [0.3365521  0.56269268 0.         0.93384497 0.56532225 0.82031005
  0.13692954 0.30907567 0.14983228 0.37209695]
 [0.69214086 0.5919511  0.93384497 0.         1.01896766 0.15476108
  1.01655495 0.63358057 0.9782845  1.25853556]
 [0.40645044 0.42762934 0.56532225 1.01896766 0.         0.98428742
  0.47297746 0.54579698 0.42914257 0.52275035]
 [0.61960969 0.5703187  0.82031005 0.15476108 0.98428742 0.
  0.91561949 0.53530753 0.88239665 1.16309059]
 [0.35784017 0.5716094  0.13692954 1.01655495 0.47297746 0.91561949
  0.         0.38321066 0.05370532 0.25006934]
 [0.14283045 0.31482334 0.30907567 0.63358057 0.54579698 0.53530753
  0.38321066 0.         0.34736835 0.6283023 ]
 [0.30919754 0.51961346 0.14983228 0.9782845  0.42914257 0.88239665
  0.05370532 0.34736835 0.         0.28096172]
 [0.57817499 0.76678828 0.37209695 1.25853556 0.52275035 1.16309059
  0.25006934 0.6283023  0.28096172 0.        ]]
```

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

[139]:
```
x = (np.random.rand(10)*100).astype(np.float32)
y = x.view(np.int32)
y[:] = x
print(y)
```

```
[11  8 27 73 83 15 83 64  0 40]
```

54. How to read the following file?.

[140]:
```
from io import StringIO
s = StringIO('''1, 2, 3, 4, 5
                6,  ,  , 7, 8
                 ,  , 9,10,11
''')
x = np.genfromtxt(s, delimiter=",", dtype=np.int)
print(x)
```

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

55. What is the equivalent of enumerate for numpy arrays?.

[141]:
```
x = np.arange(9).reshape(3,3)
for i, j in np.ndenumerate(x):
    print(i, j)
for i in np.ndindex(x.shape):
    print(i, x[i])
```

```
(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
```

56. Generate a generic 2D Gaussian-like array.

[142]:
```
x, y = np.meshgrid(np.linspace(-1,1,10), np.linspace(-1,1,10))
a = np.sqrt(x*x+y*y)
sigma, mu = 1.0, 0.0
G = np.exp(-( (a-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]]
```

57. How to randomly place p elements in a 2D array?.

[114]:
```
n = 10
b = 3
x = np.zeros((n,n))
np.put(x, np.random.choice(range(n*n), b, replace=False),1)
print(x)
```

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

58. Subtract the mean of each row of a matrix.

[115]:
```
x = np.random.rand(5, 10)
y = x - x.mean(axis=1, keepdims=True)
y = x - x.mean(axis=1).reshape(-1, 1)
print(y)
```

```
[[-0.02643165 -0.39317509 -0.35971231  0.57613894  0.54455781  0.12606
   0.12385724 -0.17935632 -0.2765568  -0.13538182]
 [ 0.14893599 -0.23193843  0.01038346  0.25468741  0.18528845 -0.49576323
  -0.15309847  0.42638125  0.00325393 -0.14813035]
 [-0.12364215 -0.43062443  0.54318395  0.1957638  -0.06263764 -0.18129665
   0.46977654 -0.24979977  0.13881962 -0.29954329]
 [ 0.22869198 -0.26231646 -0.02945235 -0.09739958 -0.07059637 -0.29119473
   0.05879504  0.1472055   0.10797288  0.2082941 ]
 [-0.26438103  0.13724655  0.12070621 -0.41209831  0.52342965  0.13496162
   0.34529857  0.06175227 -0.41035577 -0.23655977]]
```

59. How to sort an array by the nth column?.

[116]:
```
x = np.random.randint(0,10,(3,3))
print(x)
print(x[x[:,1].argsort()])
```

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

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

[117]:
```
x = np.random.randint(0,3,(3,10))
print((~x.any(axis=0)).any())
```

```
False
```

61. Find the nearest value from a given value in an array.

[120]:
```
x = np.random.uniform(0,1,10)
x1 = 0.5
x2 = x.flat[np.abs(x - x1).argmin()]
print(x2)
```

```
0.5167841573378877
```

62. Considering two arrays with shape (1,3) and (3,1), how to compute their sum using an iterator?.

[122]:
```
a = np.arange(3).reshape(3,1)
b = np.arange(3).reshape(1,3)
```

```
c = np.nditer([a,b,None])
for x,y,z in c: z[...] = x + y
print(c.operands[2])
```

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

63. Create an array class that has a name attribute.

[143]:
```
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")
x = NamedArray(np.arange(10), "range_10")
print (x.name)
```

```
range_10
```

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

[144]:
```
x = np.ones(10)
y = np.random.randint(0,len(x),20)
x+= np.bincount(y, minlength=len(x))
print(x)
np.add.at(x, y, 1)
print(x)
```

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

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

[145]:
```
X = [1,2,3,4,5,6]
I = [1,3,9,3,4,1]
F = np.bincount(I,X)
print(F)
```

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

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

[127]:
```
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]
```

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

[129]:
```
A = np.random.randint(0,10,(3,4,3,4))
sum = A.sum(axis=(-2,-1))
print(sum)
```

```
[[35 48 58 42]
 [57 40 79 42]
 [55 41 69 62]]
```

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

[146]:
```
D = np.random.uniform(0,1,100)
S = np.random.randint(0,10,100)
D_means = D_sums / D_counts
import pandas as pd
print(pd.Series(D).groupby(S).mean())
```

```
0    0.512354
1    0.325508
2    0.572039
3    0.522811
4    0.457726
5    0.486886
6    0.625627
7    0.546124
8    0.502621
9    0.499454
dtype: float64
```

69. How to get the diagonal of a dot product?.

[147]:
```
A = np.random.uniform(0,1,(5,5))
B = np.random.uniform(0,1,(5,5))
np.diag(np.dot(A, B))
np.sum(A * B.T, axis=1)
np.einsum("ij,ji->i", A, B)
```

[147]:
```
array([1.30139978, 0.90810862, 1.04767494, 0.83315991, 1.51263416])
```

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

```
[148]: 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.]
```

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

```
[149]: 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.]]]
```

72. How to swap two rows of an array?.

```
[150]: A = np.arange(25).reshape(5,5)
A[[0,1]] = A[[1,0]]
print(A)
```

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

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

```
[151]: 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)
```

```
[( 2, 41) ( 2, 72) ( 6, 36) ( 6, 91) ( 9, 13) ( 9, 97) (12, 44) (12, 74)
 (12, 75) (12, 81) (13, 25) (13, 36) (13, 40) (13, 66) (13, 97) (23, 24)
 (23, 25) (24, 25) (25, 36) (29, 57) (29, 98) (36, 91) (40, 66) (41, 72)
 (44, 74) (51, 71) (51, 80) (57, 98) (71, 80) (75, 81)]
```

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

```
[152]: 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]
```

75. How to compute averages using a sliding window over an array?.

```
[153]: 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.]
```

76. 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]).

```
[154]: from numpy.lib import stride_tricks
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)
```

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

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

[155]:
```
Z = np.random.randint(0,2,100)
np.logical_not(Z, out=Z)
Z = np.random.uniform(-1.0,1.0,100)
np.negative(Z, out=Z)
```

[155]:
```
array([-0.79186135, -0.22251153, -0.9602394 , -0.28286213, -0.52480295,
        0.89945295,  0.09744696, -0.5580724 ,  0.65430477, -0.15394662,
       -0.301893  ,  0.09393876,  0.83861768, -0.217086  , -0.73610515,
        0.12311777,  0.23825715, -0.17563787,  0.54360351,  0.10832128,
        0.82620605,  0.38251663,  0.46223126,  0.16254478,  0.0097363 ,
        0.58408502, -0.99945968,  0.04634388,  0.01960939, -0.78565661,
        0.84471186,  0.27262877,  0.54253727, -0.32910032,  0.07516819,
        0.31772027,  0.76993192, -0.11891587,  0.45815598, -0.80828575,
        0.541933  , -0.32796762,  0.32857314,  0.31786737, -0.28669799,
        0.69010975,  0.34031381,  0.08072527,  0.87886037, -0.76479989,
       -0.39765319, -0.78954779, -0.0415745 ,  0.20866954,  0.03690498,
       -0.30743702, -0.21189586, -0.63105633, -0.64163023, -0.16449962,
       -0.09426317, -0.09448111, -0.69975312,  0.5500967 ,  0.629684  ,
        0.01520882,  0.03330668,  0.96807353, -0.62261874,  0.25847443,
       -0.19460184,  0.37737429, -0.09600759,  0.72583314,  0.21191501,
       -0.61878103, -0.86738901, -0.24482283, -0.30636618,  0.46751211,
        0.61292885,  0.89751241,  0.13776485,  0.14740913, -0.15069261,
        0.19598187,  0.88286479,  0.10760678,  0.66109918, -0.21004563,
        0.15606538, -0.73563985,  0.09080486, -0.99309202,  0.07449714,
       -0.57243275,  0.61976057,  0.92309133,  0.77058998,  0.8411306 ])
```

78. 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])?.

[156]:
```
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))
```

```
[ 2.91652363 19.03797451 13.35866468 10.76562706  5.90043885 11.19200946
  7.84061858  7.79184811  5.7193492   3.27121367]
```

79. 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])?.

[157]:
```
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]))
```

```
[[11.0815807  11.83547222 11.46486006  3.04045826  6.92413453  7.860907
  12.26070356 17.2654316   7.21897451  3.44633049]
 [ 1.60890251  0.34700867  3.7867407   9.70830462  3.11270472  2.80129025
   1.26279154  7.32017701  0.94040105  5.3756476 ]
 [12.85733723  4.41417269  0.30283182  0.6853388   6.55786477  5.2503043
   3.98028913  7.54637991  4.19880745 12.7530709 ]
 [ 2.84862497  8.29892198  7.66446594 15.54488959  8.93842119  9.20833617
   7.99390846  2.90974744 12.20316041  0.48908196]
 [10.94990529  8.52067682  6.83998505  0.95812226  1.92413113  2.91736948
   8.67122533 13.2298303   2.68911578  6.21182933]
 [ 7.25369361  2.63344185  0.98933669  4.30574039  3.13051506  2.44562173
   2.70439352  7.20615336  3.22217523  5.91425541]
 [ 2.42823008  2.46091963  4.03774453  6.59289838  1.93340403  2.05253741
   3.08574768  8.58788334  0.48807652  1.19158658]
 [ 2.51772748  3.88551784  5.05011533 10.4252372   8.01693623  7.74937834
   3.8268706   0.74856266  9.37138767  4.61712954]
 [ 0.15254349  1.33708499  0.18905478  9.93953967  1.4477279   1.51467107
   0.77643965  4.66590482  4.36867768  1.21981535]
 [ 0.14322544  2.06862928  5.23501443  7.99499499  4.16569634  3.98986473
   2.96231823  8.95458931  0.55110019  4.70970631]]
```

80. 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).

[158]:
```
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)
```

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

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: FutureWarning:
Using a non-tuple sequence for multidimensional indexing is deprecated; use
`arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted
as an array index, `arr[np.array(seq)]`, which will result either in an error or
a different result.
```

81. 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]]?.

[159]:
```
Z = np.arange(1,15,dtype=np.uint32)
R = stride_tricks.as_strided(Z,(11,4),(4,4))
print(R)
```

```
[[ 1  2  3  4]
 [ 2  3  4  5]
 [ 3  4  5  6]
 [ 4  5  6  7]
 [ 5  6  7  8]
 [ 6  7  8  9]
 [ 7  8  9 10]
 [ 8  9 10 11]
 [ 9 10 11 12]
 [10 11 12 13]
 [11 12 13 14]]
```

82. Compute a matrix rank.

```
[160]: Z = np.random.uniform(0,1,(10,10))
       U, S, V = np.linalg.svd(Z)
       rank = np.sum(S > 1e-10)
       print(rank)
```

```
10
```

83. How to find the most frequent value in an array?.

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

```
7
```

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

```
[162]: 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)
```

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

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

  [[0 4 4]
   [2 0 4]
   [2 1 4]]

  [[4 4 4]
   [0 4 0]
   [1 4 0]]
```

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

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

 [[0 3 3]
  [1 1 3]
  [2 1 0]]

 [[3 3 0]
  [1 3 3]
  [1 0 2]]]


[[[4 4 2]
  [3 0 2]
  [3 2 4]]

 [[4 2 0]
  [0 2 1]
  [2 4 1]]

 [[2 0 4]
  [2 1 4]
  [4 1 0]]

 [[0 4 0]
  [1 4 0]
  [1 0 2]]

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

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

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

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


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

 [[0 2 1]
  [2 4 1]
  [0 1 4]]

 [[2 1 4]
  [4 1 0]
  [1 4 4]]

 [[1 4 0]
  [1 0 2]
  [4 4 2]]

 [[4 0 2]
  [0 2 3]
  [4 2 0]]

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

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

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


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

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

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

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

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

 [[3 4 1]
  [0 3 4]
  [1 1 3]]

 [[4 1 1]
  [3 4 1]
  [1 3 1]]]


[[[1 0 1]
  [3 1 3]
  [0 1 2]]

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

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

 [[4 4 2]
  [0 2 0]
  [1 0 0]]

 [[4 2 0]
  [2 0 1]
  [0 0 1]]

 [[2 0 3]
  [0 1 1]
  [0 1 0]]

 [[0 3 4]
  [1 1 3]
  [1 0 0]]
 [[3 4 1]
  [1 3 1]
  [0 0 1]]]


[[[3 1 3]
  [0 1 2]
  [1 2 2]]

 [[1 3 0]
  [1 2 1]
  [2 2 3]]

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

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

 [[2 0 1]
  [0 0 1]
  [4 2 4]]

 [[0 1 1]
  [0 1 0]
  [2 4 2]]

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

 [[1 3 1]
  [0 0 1]
  [2 0 0]]]


[[[0 1 2]
  [1 2 2]
  [2 4 2]]

 [[1 2 1]
  [2 2 3]
  [4 2 1]]

 [[2 1 0]
```

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

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

 [[0 0 1]
  [4 2 4]
  [2 4 2]]

 [[0 1 0]
  [2 4 2]
  [4 2 1]]

 [[1 0 0]
  [4 2 0]
  [2 1 4]]

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


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

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

 [[2 3 4]
  [2 1 2]
  [4 4 4]]

 [[3 4 2]
  [1 2 4]
  [4 4 2]]

 [[4 2 4]
  [2 4 2]
  [4 2 3]]

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

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

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

[163]:
```
class Symetric(np.ndarray):
    def __setitem__(self, index, value):
        i,j = index
        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)
```

```
[[ 1  9  9  9  2]
 [ 9  0  9 18  5]
 [ 9  9  3 42  8]
 [ 9 18 42  1  1]
 [ 2  5  8  1  8]]
```

86. 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)).

[164]:
```
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.]]
```

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

```
[165]: 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)
print(S)
```

```
[[16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]
 [16. 16. 16. 16.]]
```

88. How to implement the Game of Life using numpy arrays?.

```
[166]: def iterate(Z):
    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:])
    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)
print(Z)
```

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

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

89. How to get the n largest values of an array.

[170]:
```python
Z = np.arange(10000)
np.random.shuffle(Z)
n = 5
print (Z[np.argsort(Z)[-n:]])
```

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

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

[171]:
```
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]]
```

91. How to create a record array from a regular array?.

[172]:
```
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)]
```

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

[5]:
```
x = np.random.rand(int(5e7))
%timeit np.power(x,3)
%timeit x*x*x
%timeit np.einsum('i,i,i->i',x,x,x)
```

```
1 loop, best of 3: 3.76 s per loop
10 loops, best of 3: 147 ms per loop
10 loops, best of 3: 127 ms per loop
```

93. 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?.

```
[6]: A = np.random.randint(0,5,(8,3))
     B = np.random.randint(0,5,(2,2))
     C = (A[..., np.newaxis, np.newaxis] == B)
     rows = np.where(C.any((3,1)).all(1))[0]
     print(rows)
```

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

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

```
[7]: Z = np.random.randint(0,5,(10,3))
     print(Z)
     E = np.all(Z[:,1:] == Z[:,:-1], axis=1)
     U = Z[~E]
     print(U)
     U = Z[Z.max(axis=1) != Z.min(axis=1),:]
     print(U)
```

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

95. Convert a vector of ints into a matrix binary representation.

```
[9]: 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]]
```

96. Given a two dimensional array, how to extract unique rows?.

```
[10]: 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 0 0]
 [0 0 1]
 [1 0 1]
 [1 1 1]]
```

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

```
[17]: A = np.random.uniform(0,1,10)
      B = np.random.uniform(0,1,10)
      np.einsum('i->', A)
      np.einsum('i,i->i', A, B)
      np.einsum('i,i', A, B)
      np.einsum('i,j->ij', A, B)
```

```
[17]: array([[4.19763899e-01, 2.99677069e-02, 7.96202669e-02, 6.34913000e-02,
               2.32267402e-01, 2.80635459e-01, 6.00905047e-01, 2.79299847e-01,
               4.03137548e-01, 1.63536245e-02],
              [1.77043116e-01, 1.26394295e-02, 3.35813066e-02, 2.67786192e-02,
               9.79630332e-02, 1.18363147e-01, 2.53442715e-01, 1.17799829e-01,
               1.70030648e-01, 6.89744082e-03],
              [4.70537477e-01, 3.35925249e-02, 8.92509327e-02, 7.11710468e-02,
               2.60361879e-01, 3.14580413e-01, 6.73588999e-01, 3.13083249e-01,
               4.51900045e-01, 1.83317175e-02],
              [3.40980384e-01, 2.43432087e-02, 6.46767129e-02, 5.15749161e-02,
               1.88674224e-01, 2.27964307e-01, 4.88124000e-01, 2.26879370e-01,
               3.27474555e-01, 1.32842895e-02],
              [1.04218164e-01, 7.44032398e-03, 1.97679650e-02, 1.57634965e-02,
               5.76668988e-02, 6.96756258e-02, 1.49191536e-01, 6.93440227e-02,
               1.00090206e-01, 4.06024609e-03],
              [4.37453538e-03, 3.12306024e-04, 8.29756144e-04, 6.61669426e-04,
               2.42055587e-03, 2.92461964e-03, 6.26228314e-03, 2.91070067e-03,
               4.20126521e-03, 1.70427970e-04],
              [6.19935265e-01, 4.42583042e-02, 1.17588509e-01, 9.37681777e-02,
               3.43027959e-01, 4.14461124e-01, 8.87456568e-01, 4.12488604e-01,
               5.95380363e-01, 2.41521212e-02],
              [2.85902826e-01, 2.04111219e-02, 5.42296739e-02, 4.32441717e-02,
               1.58198232e-01, 1.91141904e-01, 4.09278768e-01, 1.90232214e-01,
               2.74578554e-01, 1.11385174e-02],
              [5.35140992e-02, 3.82046871e-03, 1.01504843e-02, 8.09426379e-03,
               2.96108857e-02, 3.57771448e-02, 7.66070936e-02, 3.56068727e-02,
               5.13944691e-02, 2.08486125e-03],
              [7.39597972e-02, 5.28012422e-03, 1.40285975e-02, 1.11867735e-02,
               4.09240767e-02, 4.94462284e-02, 1.05875745e-01, 4.92109019e-02,
               7.10303372e-02, 2.88140728e-03]])
```

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

[19]:
```
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
r = np.zeros_like(x)
r[1:] = np.cumsum(dr)
r_int = np.linspace(0, r.max(), 200)
x_int = np.interp(r_int, r, x)
y_int = np.interp(r_int, r, y)
print(y_int)
```

```
[ 0.00000000e+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]
```

99. 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.

[20]:
```
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.]]
```

100. Compute bootstrapped 95% confidence intervals for the mean of a 1D array X (i.e., resample the elements of an array with replacement N times, compute the mean of each sample, and then compute percentiles over the means).

[21]:
```
X = np.random.randn(100) # random 1D array
N = 1000 # number of bootstrap samples
idx = np.random.randint(0, X.size, (N, X.size))
means = X[idx].mean(axis=1)
confint = np.percentile(means, [2.5, 97.5])
print(confint)
```

```
[-0.31320065  0.09491305]
```