Numpy

1. Import the numpy package under the name `np` (★☆☆)

(**hint**: import … as …)

In [2]:
```
import numpy as np
```

2. Print the numpy version and the configuration (★☆☆)

(**hint**: np.__version__, np.show_config)

In [2]:
```
print(np.__version__)
print(np.show_config())
```

```
1.23.4
openblas64__info:
    library_dirs = ['D:\\a\\numpy\\numpy\\build\\openblas64__info']
    libraries = ['openblas64__info']
    language = f77
    define_macros = [('HAVE_CBLAS', None), ('BLAS_SYMBOL_SUFFIX', '64_'), ('HAVE_BLAS_ILP64', No
ne)]
blas_ilp64_opt_info:
    library_dirs = ['D:\\a\\numpy\\numpy\\build\\openblas64__info']
    libraries = ['openblas64__info']
    language = f77
    define_macros = [('HAVE_CBLAS', None), ('BLAS_SYMBOL_SUFFIX', '64_'), ('HAVE_BLAS_ILP64', No
ne)]
openblas64__lapack_info:
    library_dirs = ['D:\\a\\numpy\\numpy\\build\\openblas64__lapack_info']
    libraries = ['openblas64__lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None), ('BLAS_SYMBOL_SUFFIX', '64_'), ('HAVE_BLAS_ILP64', No
ne), ('HAVE_LAPACKE', None)]
lapack_ilp64_opt_info:
    library_dirs = ['D:\\a\\numpy\\numpy\\build\\openblas64__lapack_info']
    libraries = ['openblas64__lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None), ('BLAS_SYMBOL_SUFFIX', '64_'), ('HAVE_BLAS_ILP64', No
ne), ('HAVE_LAPACKE', None)]
Supported SIMD extensions in this NumPy install:
    baseline = SSE,SSE2,SSE3
    found = SSSE3,SSE41,POPCNT,SSE42,AVX,F16C,FMA3,AVX2
    not found = AVX512F,AVX512CD,AVX512_SKX,AVX512_CLX,AVX512_CNL,AVX512_ICL
None
```

3. Create a null vector of size 10 (★☆☆)

(**hint**: np.zeros)

In [6]:
```
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 (★☆☆)

(**hint**: size, itemsize)

In [5]:
```
print("Memory size of numpy array in bytes:",
      x.size * x.itemsize)
```

```
Memory size of numpy array in bytes: 80
```

5. How to get the documentation of the numpy add function from the command line? (★☆☆)

(**hint**: np.info)

In [7]:
```
print(np.info(np.add))
```

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[,
signature, extobj])

Add arguments element-wise.

Parameters
----------
x1, x2 : array_like
    The arrays to be added.
    If ``x1.shape != x2.shape``, they must be broadcastable to a common
    shape (which becomes the shape of the output).
out : ndarray, None, or tuple of ndarray and None, optional
    A location into which the result is stored. If provided, it must have
    a shape that the inputs broadcast to. If not provided or None,
    a freshly-allocated array is returned. A tuple (possible only as a
    keyword argument) must have length equal to the number of outputs.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
add : ndarray or scalar
    The sum of `x1` and `x2`, element-wise.
    This is a scalar if both `x1` and `x2` are scalars.

Notes
-----
Equivalent to `x1` + `x2` in terms of array broadcasting.

Examples
--------
>>> np.add(1.0, 4.0)
5.0
>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> np.add(x1, x2)
array([[  0.,   2.,   4.],
       [  3.,   5.,   7.],
       [  6.,   8.,  10.]])

The ``+`` operator can be used as a shorthand for ``np.add`` on ndarrays.

>>> x1 = np.arange(9.0).reshape((3, 3))
>>> x2 = np.arange(3.0)
>>> x1 + x2
array([[ 0.,  2.,  4.],
       [ 3.,  5.,  7.],
       [ 6.,  8., 10.]])
None
```

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

(**hint**: array[4])

In [9]:
```
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 (★☆☆)

(**hint**: np.arange)

In [10]:
```
v = np.arange(10,50)
print("Original vector:")
print(v)
```

```
Original vector:
[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) (★☆☆)

(**hint**: array[::-1])

In [11]:
```
x = np.arange(1, 11)
print("Original array:")
print(x)
print("Reverse array:")
x = x[::-1]
print(x)
```

```
Original array:
[ 1  2  3  4  5  6  7  8  9 10]
Reverse array:
[10  9  8  7  6  5  4  3  2  1]
```

9. Create a 3x3 matrix with values ranging from 0 to 8 (★☆☆)

(**hint**: reshape)

In [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] (★☆☆)

(**hint**: np.nonzero)

In [15]:
```
x = np.array([1,2,0,0,4,0])
print(x)
np.nonzero(x)
```

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

Out[15]:
```
(array([0, 1, 4], dtype=int64),)
```

11. Create a 3x3 identity matrix (★☆☆)

(**hint**: np.eye)

In [16]:
```
x = np.eye(3)
print(x)
```

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

12. Create a 3x3x3 array with random values (★☆☆)

(**hint**: np.random.random)

In [17]:
```
x = np.random.random((3,3,3))
print(x)
```

```
[[[0.85120877 0.50336793 0.50105321]
  [0.47026765 0.69871386 0.60506446]
  [0.71946291 0.10121237 0.85542661]]

 [[0.83765256 0.36688355 0.595501  ]
  [0.00303706 0.2730718  0.7394878 ]
  [0.75731516 0.20868914 0.6632824 ]]

 [[0.41493608 0.66850476 0.66036291]
  [0.20330441 0.51298658 0.07087709]
  [0.54732113 0.21538883 0.065564  ]]]
```

13. Create a 10x10 array with random values and find the minimum and maximum values (★☆☆)

(**hint**: min, max)

In [18]:
```
x = np.random.random((10,10))
print("Original Array:")
print(x)
xmin, xmax = x.min(), x.max()
print("Minimum and Maximum Values:")
print(xmin, xmax)
```

```
Original Array:
[[0.8858279  0.00513343 0.22415951 0.69616959 0.16073009 0.12840889
  0.09567875 0.3012956  0.880026   0.19125381]
 [0.14791652 0.81403657 0.41634313 0.55478787 0.33990593 0.64256306
  0.26113566 0.75216435 0.9486391  0.74030603]
 [0.01651238 0.24105192 0.86689481 0.37878352 0.55395597 0.98730138
  0.7513223  0.49477729 0.09384359 0.30912464]
 [0.96179413 0.40949746 0.58332016 0.27797468 0.52142314 0.24029889
  0.96457038 0.30539383 0.40263945 0.55553253]
 [0.32782602 0.0995882  0.82494144 0.41603061 0.78379338 0.16589087
  0.38703483 0.0400196  0.23078904 0.98690493]
 [0.40413288 0.27041552 0.89209253 0.2664043  0.60684998 0.99255068
  0.50935404 0.06181492 0.04583386 0.56127608]
 [0.28885642 0.04240628 0.24555954 0.44589611 0.20861985 0.04039362
  0.50432498 0.94750994 0.82773739 0.19107412]
 [0.24789527 0.06388299 0.11000077 0.26332092 0.86469273 0.88447477
  0.58336095 0.65358244 0.5323352  0.02579737]
 [0.25923951 0.48583739 0.42230887 0.62368884 0.43437952 0.71132717
  0.01323914 0.85981195 0.87799323 0.4467971 ]
 [0.89598298 0.55178063 0.98972389 0.26044079 0.93731481 0.11265069
  0.27932989 0.31225238 0.41147269 0.2117078 ]]
Minimum and Maximum Values:
0.005133428733660406 0.9925506848857093
```

14. Create a random vector of size 30 and find the mean value (★☆☆)

(**hint**: mean)

In [19]:
```
x = np.random.random(30)
m = x.mean()
print(m)
```

```
0.5721043719821004
```

15. Create a 2d array with 1 on the border and 0 inside (★☆☆)

(**hint**: array[1:-1, 1:-1])

In [20]:
```
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? (★☆☆)

(**hint**: np.pad)

In [21]:
```
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? (★☆☆)

(**hint**: NaN = not a number, inf = infinity)

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

In [ ]:
```
nan
False
False
nan
```

```
False
nan
```

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

(**hint**: np.diag)

In [22]:
```
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 (★☆☆)

(**hint**: array[::2])

In [23]:
```
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?

(**hint**: np.unravel_index)

In [24]:
```
print(np.unravel_index(100,(6,7,8)))
```

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

21. Create a checkerboard 8x8 matrix using the tile function (★☆☆)

(**hint**: np.tile)

In [25]:
```
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 (★☆☆)

(**hint**: (x - min) / (max - min))

In [26]:
```
x= np.random.random((5,5))
print("Original Array:")
print(x)
xmax, xmin = x.max(), x.min()
x = (x - xmin)/(xmax - xmin)
print("After normalization:")
print(x)
```

```
Original Array:
[[0.90833432 0.27790854 0.02723855 0.99373289 0.55419898]
 [0.01535892 0.72827542 0.88390282 0.09353681 0.00916852]
 [0.32283511 0.17867176 0.62163618 0.56199459 0.8394597 ]
 [0.46244947 0.55203986 0.60321597 0.47678861 0.41389742]
 [0.49014233 0.19254536 0.18269429 0.79320203 0.9144493 ]]
After normalization:
[[0.91326258 0.27295323 0.01835332 1.         0.55357525]
 [0.00628745 0.73038079 0.88844806 0.08569098 0.        ]
 [0.31858413 0.17216065 0.6220697  0.56149307 0.84330817]
 [0.46038732 0.55138227 0.6033607  0.47495126 0.41107409]
 [0.48851434 0.18625175 0.17624624 0.7963253  0.91947343]]
```

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

(**hint**: np.dtype)

In [28]:
```
color = np.dtype([("r", np.ubyte, 1),
                  ("g", np.ubyte, 1),
                  ("b", np.ubyte, 1),
                  ("a", np.ubyte, 1)])
```

```
C:\Users\Karthi\AppData\Local\Temp\ipykernel_11416\2911720781.py:1: 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'.
  color = np.dtype([("r", np.ubyte, 1),
```

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

(**hint**: np.dot | @)

In [29]:
```
x = np.arange(15).reshape((5, 3))
y = np.arange(6).reshape((3, 2))
np.dot(x, y)
```

Out[29]:
```
array([[ 10,  13],
       [ 28,  40],
       [ 46,  67],
       [ 64,  94],
       [ 82, 121]])
```

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

(**hint**: >, <=)

In [31]:
```
x = np.arange(11)
x[(x >= 3) & (x <= 8)] = -1
print(x)
```

```
[ 0  1  2 -1 -1 -1 -1 -1 -1  9 10]
```

26. What is the output of the following script? (★☆☆)

(**hint**: np.sum)

```
# Author: Jake VanderPlas

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

In [ ]:

27. Consider an integer vector Z, which of these expressions are legal? (★☆☆)

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

In [ ]:

28. What are the result of the following expressions?

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

In [33]:
```
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
[-2.14748365e+09]
C:\Users\Karthi\AppData\Local\Temp\ipykernel_11416\3912170336.py:1: RuntimeWarning: invalid valu
e encountered in divide
  print(np.array(0) / np.array(0))
C:\Users\Karthi\AppData\Local\Temp\ipykernel_11416\3912170336.py:2: RuntimeWarning: divide by ze
ro encountered in floor_divide
  print(np.array(0) // np.array(0))
```

29. How to round away from zero a float array ? (★☆☆)

(**hint**: np.uniform, np.copysign, np.ceil, np.abs)

In [34]:
```
x = np.random.uniform(-10,+10,10)
print(np.copysign(np.ceil(np.abs(x)), x))
```

```
[-10.  -8.  -7.  -8.  -5.   4.   5.   8.  -5.   3.]
```

30. How to find common values between two arrays? (★☆☆)

(**hint**: np.intersect1d)

In [37]:
```
x1 = np.random.randint(0,10,10)
```

```
x2 = np.random.randint(0,10,10)
print(np.intersect1d(x1,x2))
```

```
[1 4 5 6 7 9]
```

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

(**hint**: np.seterr, np.errstate)

In [38]:
```
data = np.random.random(1000).reshape(10, 10,10) * np.nan
np.seterr(all="ignore")
np.nanmedian(data, axis=[1, 2])
```

```
C:\Users\Karthi\PycharmProjects\pythonProject\venv\lib\site-packages\numpy\lib\nanfunctions.py:1
217: RuntimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(a, func=_nanmedian, axis=axis, out=out,
```

Out[38]:
```
array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan])
```

32. Is the following expressions true? (★☆☆)

(**hint**: imaginary number)

```
np.sqrt(-1) == np.emath.sqrt(-1)
```

In [39]:
```
np.sqrt(-1) == np.emath.sqrt(-1)
```

Out[39]:
```
False
```

33. How to get the dates of yesterday, today and tomorrow? (★☆☆)

(**hint**: np.datetime64, np.timedelta64)

In [42]:
```
today = np.datetime64('today', 'D')
print("Today: ", today)
yesterday = np.datetime64('today', 'D')
- np.timedelta64(1, 'D')
print("Yestraday: ", yesterday)
tomorrow = np.datetime64('today', 'D')
+ np.timedelta64(1, 'D')

print("Tomorrow: ", tomorrow)
```

```
Today:  2022-11-06
Yestraday:  2022-11-06
Tomorrow:  2022-11-06
```

34. How to get all the dates corresponding to the month of July 2016? (★★☆)

(**hint**: np.arange(dtype=datetime64['D']))

In [43]:
```
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)? (★★☆)

(**hint**: np.add(out=), np.negative(out=), np.multiply(out=), np.divide(out=))

In [44]:
```
A = np.ones(3)*1
B = np.ones(3)*2
np.add(A,B,out=B)
np.divide(A,2,out=A)
np.negative(A,out=A)
np.multiply(A,B,out=A)
```

Out[44]:
```
array([-1.5, -1.5, -1.5])
```

36. Extract the integer part of a random array using 5 different methods (★★☆)

(**hint**: %, np.floor, np.ceil, astype, np.trunc)

In [45]:
```
Z = np.random.uniform(0,10,10)

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

```
[4. 0. 0. 8. 3. 7. 3. 1. 0. 5.]
[4. 0. 0. 8. 3. 7. 3. 1. 0. 5.]
[4. 0. 0. 8. 3. 7. 3. 1. 0. 5.]
[4 0 0 8 3 7 3 1 0 5]
[4. 0. 0. 8. 3. 7. 3. 1. 0. 5.]
```

37. Create a 5x5 matrix with row values ranging from 0 to 4 (★★☆)

(**hint**: np.arange)

In [48]:
```
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 (★☆☆)

(**hint**: np.fromiter)

In [49]:
```
def generate():
    for x in range(10):
```

```
        yield x
Z = np.fromiter(generate(),dtype=float,count=-1)
print(Z)
```

```
[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 (★★☆)

(**hint**: np.linspace)

In [50]:
```
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 (★★☆)

(**hint**: sort)

In [51]:
```
x = np.random.random(10)
x.sort()
print(x)
```

```
[0.02686435 0.03635556 0.142623   0.18120865 0.23161733 0.32864311
 0.48883962 0.51497123 0.7479827  0.97344089]
```

41. How to sum a small array faster than np.sum? (★★☆)

(**hint**: np.add.reduce)

In [52]:
```
Z = np.arange(10)
np.add.reduce(Z)
```

Out[52]:
```
45
```

42. Consider two random array A and B, check if they are equal (★★☆)

(**hint**: np.allclose, np.array_equal)

In [53]:
```
x = np.random.randint(0,2,6)
print("First array:")
print(x)
y = np.random.randint(0,2,6)
print("Second array:")
print(y)
print("Test above two arrays are equal or not!")
array_equal = np.allclose(x, y)
print(array_equal)
```

```
First array:
[0 1 0 0 0 1]
Second array:
[0 1 0 0 1 0]
Test above two arrays are equal or not!
False
```

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

(**hint**: flags.writeable)

In [55]:
```
x = np.zeros(10)
x.flags.writeable = False
print("Test the array is read-only or not:")
print("Try to change the value of the first element:")
x[0] = 1
```

```
Test the array is read-only or not:
Try to change the value of the first element:
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [55], line 5
      3 print("Test the array is read-only or not:")
      4 print("Try to change the value of the first element:")
----> 5 x[0] = 1

ValueError: assignment destination is read-only
```

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

(**hint**: np.sqrt, np.arctan2)

In [56]:
```
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)
```

```
[0.56545983 1.01800767 1.06691187 0.98873855 1.36193839 0.99848716
 1.23459307 0.5331449  0.74376735 0.78080847]
[0.14325456 1.27965608 0.43743505 1.31485693 0.81591354 0.3463516
 0.84571992 1.27472814 0.95528993 1.13808166]
```

45. Create random vector of size 10 and replace the maximum value by 0 (★★☆)

(**hint**: argmax)

In [57]:
```
x = np.random.random(15)
print("Original array:")
print(x)
x[x.argmax()] = -1
print("Maximum value replaced by -1:")
print(x)
```

```
Original array:
[0.6274672  0.41558401 0.13635386 0.52751064 0.72383983 0.2909096
 0.53445332 0.14477827 0.79803549 0.47788456 0.2674997  0.20371924
 0.25253864 0.86316714 0.37664417]
Maximum value replaced by -1:
[ 0.6274672   0.41558401  0.13635386  0.52751064  0.72383983  0.2909096
  0.53445332  0.14477827  0.79803549  0.47788456  0.2674997   0.20371924
  0.25253864 -1.          0.37664417]
```

46. Create a structured array with `x` and `y` coordinates covering the [0,1]x[0,1] area (★★☆)

(**hint**: np.meshgrid)

In [58]:
```
Z = np.zeros((5,5), [('x',float),('y',float)])
```

```
Z['x'], Z['y'] = np.meshgrid(np.linspace(0,1,5),
                             np.linspace(0,1,5))
print(Z)
```

```
[[(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))

(**hint**: np.subtract.outer)

In [59]:
```
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 (★★☆)

(**hint**: np.iinfo, np.finfo, eps)

In [60]:
```
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? (★★☆)

(**hint**: np.set_printoptions)

In [61]:
```
np.set_printoptions(threshold=float("inf"))
Z = np.zeros((40,40))
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. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  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? (★★☆)

(**hint**: argmin)

In [63]:

```
x = np.arange(100)
print("Original array:")
print(x)
a = np.random.uniform(0,100)
print("Value to compare:")
print(a)
index = (np.abs(x-a)).argmin()
print(x[index])
```

```
Original 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 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 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
Value to compare:
25.07748737536797
25
```

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

(**hint**: dtype)

In [64]:

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

```
C:\Users\Karthi\AppData\Local\Temp\ipykernel_11416\274409719.py:1: 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'.
  Z = np.zeros(10, [ ('position', [ ('x', float, 1),
```

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

(**hint**: np.atleast_2d, T, np.sqrt)

In [65]:
```
a= np.random.random((10,2))
x,y = np.atleast_2d(a[:,0], a[:,1])
d = np.sqrt( (x-x.T)**2 + (y-y.T)**2)
print(d)
```

```
[[0.         0.77977375 0.43200719 0.36604681 0.68103187 0.34312417
  0.54793295 0.6381005  0.74224131 0.23047054]
 [0.77977375 0.         0.49719433 0.54906995 0.88432418 0.47240169
  0.83934353 0.81332097 0.65676383 0.95001427]
 [0.43200719 0.49719433 0.         0.07113998 0.43592022 0.13707919
  0.35659231 0.36635063 0.34537465 0.51225845]
 [0.36604681 0.54906995 0.07113998 0.         0.43331893 0.12171505
  0.33564231 0.36788203 0.39057951 0.44163179]
 [0.68103187 0.88432418 0.43592022 0.43331893 0.         0.55447897
  0.13486267 0.07163494 0.29134037 0.59216813]
 [0.34312417 0.47240169 0.13707919 0.12171505 0.55447897 0.
  0.45652775 0.48809869 0.48162961 0.47998658]
 [0.54793295 0.83934353 0.35659231 0.33564231 0.13486267 0.45652775
  0.         0.11566999 0.33891903 0.46048224]
 [0.6381005  0.81332097 0.36635063 0.36788203 0.07163494 0.48809869
  0.11566999 0.         0.24051482 0.57164967]
 [0.74224131 0.65676383 0.34537465 0.39057951 0.29134037 0.48162961
  0.33891903 0.24051482 0.         0.7424873 ]
 [0.23047054 0.95001427 0.51225845 0.44163179 0.59216813 0.47998658
  0.46048224 0.57164967 0.7424873  0.        ]]
```

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

(**hint**: astype(copy=False))

In [66]:
```
Z = (np.random.rand(10)*100).astype(np.float32)
Y = Z.view(np.int32)
Y[:] = Z
print(Y)
```

```
[18 67 41 53 52 43 92 86 68 11]
```

54. How to read the following file? (★★☆)

(**hint**: np.genfromtxt)

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

In [69]:
```
from io import StringIO
s = StringIO('''1, 2, 3, 4, 5
                6,  ,  , 7, 8
                 ,  , 9,10,11
```

```
''')
Z = np.genfromtxt(s, delimiter=",", dtype=np.int)
print(Z)
```

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

```
C:\Users\Karthi\AppData\Local\Temp\ipykernel_11416\4147279938.py:6: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Z = np.genfromtxt(s, delimiter=",", dtype=np.int)
```

55. What is the equivalent of enumerate for numpy arrays? (★★☆)

(**hint**: np.ndenumerate, np.ndindex)

In [70]:
```
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
```

56. Generate a generic 2D Gaussian-like array (★★☆)

(**hint**: np.meshgrid, np.exp)

In [71]:
```
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]]
```

57. How to randomly place p elements in a 2D array? (★★☆)

(**hint**: np.put, np.random.choice)

In [72]:
```
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. 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. 0. 0. 0. 0. 0. 0. 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.]]
```

58. Subtract the mean of each row of a matrix (★★☆)

(**hint**: mean(axis=,keepdims=))

In [73]:
```
X = np.random.rand(5, 10)

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

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

print(Y)
```

```
[[-0.27887003 -0.54171952 -0.13157477  0.10104573 -0.10419819  0.3030246
   0.43855275 -0.08572746  0.35553477 -0.05606788]
 [ 0.09770969  0.04588653 -0.47261463 -0.4170308   0.38895876  0.05293738
  -0.01274458  0.25955215  0.07224089 -0.01489539]
 [ 0.12195238  0.00068794 -0.52428593  0.23855619  0.27667408 -0.15780432
  -0.23516903  0.2973676  -0.22437974  0.20640084]
 [-0.41687486  0.21319305  0.39230291 -0.32700889  0.17493531  0.25627164
   0.16090959 -0.02311737 -0.1049385  -0.32567288]
 [-0.18793298  0.495798   -0.30988918 -0.15173201  0.12691848  0.51956571
  -0.36575861 -0.17979206 -0.18729335  0.24011601]]
```

59. How to sort an array by the nth column? (★★☆)

(**hint**: argsort)

In [74]:
```
Z = np.random.randint(0,10,(3,3))
print(Z)
print(Z[Z[:,1].argsort()])
```

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

60. How to tell if a given 2D array has null columns? (★★☆)

(**hint**: any, ~)

In [75]:
```
print("Original array:")
nums = np.random.randint(0,3,(4,10))
print(nums)
print("\nTest whether the said array has null columns or not:")
print((~nums.any(axis=0)).any())
```

```
Original array:
[[2 2 2 1 0 0 2 2 0 0]
 [1 0 1 1 2 0 0 1 2 0]
 [0 1 1 0 0 2 0 2 2 0]
 [0 2 2 0 1 1 0 0 1 0]]

Test whether the said array has null columns or not:
True
```

61. Find the nearest value from a given value in an array (★★☆)

(**hint**: np.abs, argmin, flat)

In [76]:
```
Z = np.random.uniform(0,1,10)
z = 0.5
m = Z.flat[np.abs(Z - z).argmin()]
print(m)
```

```
0.3834625963390236
```

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

(**hint**: np.nditer)

In [77]:
```
A = np.arange(3).reshape(3,1)
B = np.arange(3).reshape(1,3)
it = np.nditer([A,B,None])
for x,y,z in it: z[...] = x + y
print(it.operands[2])
```

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

63. Create an array class that has a name attribute (★★☆)

(**hint**: class method)

In [78]:
```
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.name = getattr(obj, 'name', "no name")

Z = NamedArray(np.arange(10), "range_10")
print (Z.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)? (★★★)

(**hint**: np.bincount | np.add.at)

In [79]:
```
Z = np.ones(10)
I = np.random.randint(0,len(Z),20)
Z += np.bincount(I, minlength=len(Z))
print(Z)
```

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

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

(**hint**: np.bincount)

In [80]:
```
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 (★★★)

(**hint**: np.unique)

In [81]:
```
w, h = 256, 256
I = np.random.randint(0, 4, (h, w, 3)).astype(np.ubyte)
colors = np.unique(I.reshape(-1, 3), axis=0)
```

```
n = len(colors)
print(n)
```

```
64
```

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

(**hint**: sum(axis=(-2,-1)))

In [82]:
```
A = np.random.randint(0,10,(3,4,3,4))
# solution by passing a tuple of axes (introduced in numpy 1.7.0)
sum = A.sum(axis=(-2,-1))
print(sum)
# solution by flattening the last two dimensions into one
# (useful for functions that don't accept tuples for axis argument)
sum = A.reshape(A.shape[:-2] + (-1,)).sum(axis=-1)
print(sum)
```

```
[[28 45 57 57]
 [49 62 63 51]
 [56 47 37 50]]
[[28 45 57 57]
 [49 62 63 51]
 [56 47 37 50]]
```

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? (★★★)

(**hint**: np.bincount)

In [83]:
```
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)
```

```
[0.55404597 0.39242288 0.69128272 0.53345737 0.26144548 0.55496727
 0.54024641 0.44263176 0.55706675 0.49284397]
```

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

(**hint**: np.diag)

In [85]:
```
A = np.random.uniform(0,1,(5,5))
B = np.random.uniform(0,1,(5,5))
np.diag(np.dot(A, B))
```

Out[85]:
```
array([1.58677445, 1.10715645, 1.58262354, 0.94869282, 0.94305722])
```

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

(**hint**: array[::4])

In [86]:
```
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)? (★★★)

(**hint**: array[:, :, None])

In [87]:
```
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? (★★★)

(**hint**: array[[]] = array[[]])

In [88]:
```
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 (★★★)

(**hint**: repeat, np.roll, np.sort, view, np.unique)

In [89]:
```
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, 18) ( 0, 92) ( 7,  7) ( 7, 17) ( 7, 19) ( 7, 42) ( 7, 46) ( 7, 73)
 ( 9, 26) ( 9, 32) ( 9, 53) ( 9, 59) ( 9, 60) ( 9, 79) (10, 17) (10, 22)
 (17, 22) (18, 92) (19, 46) (26, 53) (32, 60) (34, 34) (34, 86) (42, 73)
 (44, 70) (44, 88) (59, 79) (70, 88)]
```

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

(**hint**: np.repeat)

In [90]:
```
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? (★★★)

(**hint**: np.cumsum)

In [91]:
```
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]) (★★★)

(**hint**: from numpy.lib import stride_tricks)

In [93]:
```
from numpy.lib import stride_tricks

def rolling(a, window):
    shape = (a.size - window + 1, window)
    strides = (a.strides[0], a.strides[0])
    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? (★★★)

(**hint**: np.logical_not, np.negative)

In [94]:
```
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)
```

Out[94]:
```
array([ 0.28540488,  0.90287851, -0.77287605,  0.7353899 ,  0.52749302,
        0.67760383,  0.28249118,  0.70624071,  0.34876908,  0.35970185,
        0.83560391, -0.89922   , -0.17779489, -0.49671865,  0.45047752,
       -0.32807356, -0.7316412 , -0.84595484,  0.60555132,  0.44850001,
       -0.44975838, -0.56649648,  0.93759338, -0.33852171,  0.35571886,
        0.74244705, -0.02038736,  0.58122157,  0.72524273,  0.5193706 ,
        0.86494688,  0.58353557,  0.55654949, -0.44536647, -0.67291805,
       -0.17098583, -0.04984809,  0.40242209,  0.57014732,  0.95058825,
        0.1347241 ,  0.84989967,  0.94214063,  0.21264974, -0.08676216,
       -0.15144811,  0.67365151,  0.66243336, -0.00274423, -0.65567004,
        0.79370558, -0.35507672,  0.68671568,  0.15559659,  0.96127715,
       -0.35239911, -0.24873985,  0.15732002, -0.02570114, -0.39532512,
        0.06497932, -0.25525731, -0.7880869 ,  0.74665161, -0.22372934,
        0.15558249,  0.41512754,  0.32524696, -0.47091519, -0.45536523,
        0.50520931, -0.38969227, -0.42604626,  0.61547144, -0.1613881 ,
        0.20579789,  0.4825892 ,  0.45428185,  0.5624528 ,  0.03240527,
       -0.01497646, -0.03376873,  0.50105648,  0.3741023 , -0.50418152,
       -0.03589371,  0.1429046 ,  0.18716565, -0.83599666, -0.00229628,
        0.91615761,  0.52192701,  0.92803106, -0.07220527,  0.62820275,
       -0.53334483, -0.80192287, -0.05013882,  0.43023467, -0.01674865])
```

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])? (★★★)

In [95]:
```
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))
```

```
[ 4.12885645  7.54550411  7.91113081 10.10040262 14.61489139 12.4283293
  1.80056966  7.07918016  6.87795153  1.12191953]
```

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])? (★★★)

In [96]:
```
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]))
```

```
[[ 2.93337695  0.60352838  4.3771031   4.84726565  1.14397954  3.32036781
   8.21953923  3.34053269  1.4512524   8.11342263]
 [ 7.90265847  1.18681179  3.82142114 11.60383733  1.43291234  7.19109155
   0.66942834  5.95538104  3.16719678 14.89798756]
 [ 6.33071511  3.06566816  5.633057    4.28268385  0.88189088  0.31891482
   9.15319189  7.1992046   2.83621912  7.53622318]
 [ 0.92337067 11.8522691   9.63191215  2.40855676 13.27962687 15.81443903
  14.44508205  5.83868688  9.89415811  0.89144626]
 [ 1.7370818  14.12899043 13.52317735  6.28773767 15.97420877 17.41900497
  18.34721367  4.90561994 13.06335573  2.98911851]
 [ 4.05889508  1.46273906  4.76309938  4.69579522  0.42851351  2.28507138
   8.49603436  4.64050475  1.95410852  7.95768411]
 [ 5.33381461  9.92245468  4.45655706  2.51456982 10.69052569 14.88932415
   9.36719463  8.30171647  6.50628403  5.82059429]
 [ 6.59485206  0.71242486  7.49040229  2.15875283  1.55139739  2.47904179
  11.14479846  6.23519842  0.36960824  5.41645337]
 [ 2.36998621  8.46897486  1.55266136 11.70392818  6.94378842  3.96407701
   1.8252105   6.81044775  9.59163403 14.95545818]
 [ 3.09337668  4.73930364  3.73129069 12.79966591  3.91986016  0.6465315
   0.15049787  0.71933373  7.59687205 16.07102243]]
```

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) (★★★)

(**hint**: minimum, maximum)

In [97]:
```
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)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [97], line 23
     21 r = [slice(start,stop) for start,stop in zip(R_start,R_stop)]
     22 z = [slice(start,stop) for start,stop in zip(Z_start,Z_stop)]
---> 23 R[r] = Z[z]
     24 print(Z)
     25 print(R)

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and integer or
 boolean arrays are valid indices
```

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]]? (★★★)

(**hint**: stride_tricks.as_strided)

In [99]:
```
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 (★★★)

(**hint**: np.linalg.svd) (suggestion: np.linalg.svd)

In [100…
```
Z = np.random.uniform(0,1,(10,10))
U, S, V = np.linalg.svd(Z) # Singular Value Decomposition
rank = np.sum(S > 1e-10)
print(rank)
```

```
10
```

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

(**hint**: np.bincount, argmax)

In [101…
```
Z = np.random.randint(0,10,50)
print(np.bincount(Z).argmax())
```

```
2
```

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

(**hint**: stride_tricks.as_strided)

In [102…
```
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)
```

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

  [[0 0 3]
```

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


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

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

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

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

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

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

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

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


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

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

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

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

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

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

 [[4 2 2]
```

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

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


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

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

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

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

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

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

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

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


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

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

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

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

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

 [[3 2 2]
```

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

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

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


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

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

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

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

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

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

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

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


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

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

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

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

 [[3 4 2]
```

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

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

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

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

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

(**hint**: class method)

In [103…
```
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)
```

```
[[ 4  4  1 11 16]
 [ 4  8  0 12 16]
 [ 1  0  0 42  8]
 [11 12 42  7  8]
 [16 16  8  8  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)) (★★★)

(**hint**: np.tensordot)

In [104…
```
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)? (★★★)

(**hint**: np.add.reduceat)

In [105…

```
arra1 = np.ones((16,16))
k = 5
print("Original arrays:")
print(arra1)
result = np.add.reduceat(np.add.reduceat(arra1, np.arange(0, arra1.shape[0], k), axis=0),
                                       np.arange(0, arra1.shape[1], k), axis=1)
print("\nBlock-sum (4x4) of the said array:")
print(result)
```

```
Original arrays:
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]

Block-sum (4x4) of the said array:
[[25. 25. 25.  5.]
 [25. 25. 25.  5.]
 [25. 25. 25.  5.]
 [ 5.  5.  5.  1.]]
```

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

In [106…

```
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:])

    # 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)
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 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 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 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 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 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 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
  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 1 1 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 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
  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 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 1 1 0 0 0 0 0 0 0 0 1 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 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 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 1 0 0 0 0 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 1 0 0 0 0 0 1 1 1 0 0 0 0 1 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 1 1 1 1 0 1 0 0 0 0 0 1 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 1 1 1 1 1 0 0 0 0 0 1 1 0 1 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 1 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1
  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 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 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 1 0 0 1 0 0 0 1 0 1 1 0 0 0 1 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 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0
  0 1 1 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 1 0 1 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 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 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 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 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 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 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 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 (★★★)

(**hint**: np.argsort | np.argpartition)

In [107...

```
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) (★★★)

(**hint**: np.indices)

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]]
```

91. How to create a record array from a regular array? (★★★)

(**hint**: np.core.records.fromarrays)

In [109…
```
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 (★★★)

(**hint**: np.power, *, np.einsum)

In [111…
```
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)
```

```
2.29 s ± 87 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
380 ms ± 6.96 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
211 ms ± 9.89 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

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? (★★★)

(**hint**: np.where)

In [112…
```
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)
```

```
[0 2 4 5 7]
```

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

In [113…
```
Z = np.random.randint(0,5,(10,3))
print(Z)
# solution for arrays of all dtypes (including string arrays and record arrays)
E = np.all(Z[:,1:] == Z[:,:-1], axis=1)
U = Z[~E]
print(U)
# soluiton for numerical arrays only, will work for any number of columns in Z
U = Z[Z.max(axis=1) != Z.min(axis=1),:]
print(U)
```

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

95. Convert a vector of ints into a matrix binary representation (★★★)

(**hint**: np.unpackbits)

In [114…
```
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])

# Author: Daniel T. McDonald

I = np.array([0, 1, 2, 3, 15, 16, 32, 64, 128], dtype=np.uint8)
print(np.unpackbits(I[:, np.newaxis], axis=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]]
[[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? (★★★)

(**hint**: np.ascontiguousarray)

In [115…

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

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

(**hint**: np.einsum)

In [116…

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

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->ij', A, B)    # np.outer(A, B)
```

Out[116]:
```
array([[1.86862240e-03, 1.00016263e-01, 5.21444062e-02, 6.63782345e-02,
        1.09712393e-01, 9.69661848e-02, 1.12014968e-01, 6.00771763e-02,
        2.58982285e-02, 1.08482057e-01],
       [1.51590666e-02, 8.11374837e-01, 4.23017795e-01, 5.38488717e-01,
        8.90033999e-01, 7.86631294e-01, 9.08713480e-01, 4.87371830e-01,
        2.10097541e-01, 8.80052989e-01],
       [7.41749554e-03, 3.97014498e-01, 2.06987190e-01, 2.63488363e-01,
        4.35503278e-01, 3.84907214e-01, 4.44643350e-01, 2.38476316e-01,
        1.02803003e-01, 4.30619462e-01],
       [1.05529589e-02, 5.64837236e-01, 2.94483131e-01, 3.74868020e-01,
        6.19595680e-01, 5.47612060e-01, 6.32599370e-01, 3.39283084e-01,
        1.46259052e-01, 6.12647417e-01],
       [1.07657905e-02, 5.76228849e-01, 3.00422254e-01, 3.82428342e-01,
        6.32091659e-01, 5.58656276e-01, 6.45357607e-01, 3.46125731e-01,
        1.49208798e-01, 6.25003264e-01],
       [1.27850416e-02, 6.84307369e-01, 3.56769992e-01, 4.54157290e-01,
        7.50647908e-01, 6.63438853e-01, 7.66402040e-01, 4.11045696e-01,
        1.77194669e-01, 7.42230000e-01],
       [5.08956567e-03, 2.72414233e-01, 1.42025686e-01, 1.80794356e-01,
        2.98823575e-01, 2.64106737e-01, 3.05095099e-01, 1.63632168e-01,
        7.05389889e-02, 2.95472511e-01],
       [1.15475903e-02, 6.18073949e-01, 3.22238585e-01, 4.10199864e-01,
        6.77993455e-01, 5.99225276e-01, 6.92222760e-01, 3.71260998e-01,
        1.60044176e-01, 6.70390308e-01],
       [7.15116816e-04, 3.82759574e-02, 1.99555253e-02, 2.54027735e-02,
        4.19866403e-02, 3.71087006e-02, 4.28678299e-02, 2.29913753e-02,
        9.91118307e-03, 4.15157942e-02],
       [5.47216651e-03, 2.92892584e-01, 1.52702265e-01, 1.94385313e-01,
        3.21287211e-01, 2.83960584e-01, 3.28030187e-01, 1.75932982e-01,
        7.58416567e-02, 3.17684235e-01]])
```

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

(**hint**: np.cumsum, np.interp)

In [118…
```
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)
```

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. (★★★)

(**hint**: np.logical_and.reduce, np.mod)

In [119…
```
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). (★★★)

(**hint**: np.percentile)

In [120…

```
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)
```

```
[-0.22021422  0.17192439]
```

In [ ]: