Numpy

In [8]:

```
import numpy as np
```

In [9]:

```
np.__version__
```

Out[9]:

```
'1.21.3'
```

In [87]:

```
np.__config__.show()
```

```
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
    libraries = ['openblas_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
    libraries = ['openblas_lapack_info']
    language = f77
    define_macros = [('HAVE_CBLAS', None)]
Supported SIMD extensions in this NumPy install:
    baseline = SSE,SSE2,SSE3
    found = SSSE3,SSE41,POPCNT,SSE42,AVX,F16C,FMA3,AVX2,AVX512F,AVX512CD,AVX512_SKX,A
VX512_CLX,AVX512_CNL
    not found =
```

In [19]:

```
def function5():
    z = np.zeros(10)

    return z
function5()
```

Out[19]:

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

In [150…

```
Z = np.zeros((10,10))
print("%d bytes" % (Z.size * Z.itemsize))
```

```
800 bytes
```

In [44]:

```
print(np.info(np.add))
```

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, s
ubok=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
```

In [45]:

```
def function6():
    arr = np.zeros(10, dtype = int)
    arr[4] = 1
    return arr
function6()
```

Out[45]:

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

In [49]:

```
arr = np.arange(10,50)
arr
```

Out[49]:

```
array([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])
```

In [50]:

```
arr = np.arange(10,50)
arr = arr[::-1]
arr
```

Out[50]:

```
array([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])
```

In [51]:

```
arr = np.arange(0,9).reshape(3,3)
arr
```

Out[51]:

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

In [53]:

```
numbers = [1,2,0,0,4,0]
print(f"Numbers:{numbers}")
x = np.nonzero(numbers)
x
```

```
Numbers:[1, 2, 0, 0, 4, 0]
```

Out[53]:

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

In [57]:

```
arr = np.eye(3)
arr
```

Out[57]:

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

In [61]:

```
arr = np.random.random((3,3,3))
arr
```

Out[61]:

```
array([[[0.60463745, 0.25334227, 0.77234308],
        [0.91031164, 0.64826374, 0.86399998],
        [0.59321509, 0.22829223, 0.47751083]],

       [[0.84257276, 0.12545001, 0.58576174],
        [0.93670738, 0.52882034, 0.45813358],
        [0.61914957, 0.80468492, 0.0751875 ]],

       [[0.52290959, 0.15720452, 0.88815775],
        [0.81025234, 0.60563973, 0.82552899],
        [0.03429964, 0.91035575, 0.64240596]]])
```

In [67]:

```
arr = np.random.random((10,10))
arr
```

Out[67]:

```
array([[0.56007357, 0.14373559, 0.63141721, 0.94488136, 0.53697404,
        0.79321641, 0.46991911, 0.83290872, 0.63828618, 0.02540046],
       [0.99209901, 0.18109612, 0.79765051, 0.09526799, 0.23107595,
        0.04302469, 0.57371989, 0.14457724, 0.93755008, 0.19153258],
       [0.43939106, 0.64124415, 0.40463718, 0.57806386, 0.37997946,
        0.15020174, 0.20446588, 0.96140205, 0.01841869, 0.98809561],
       [0.93678239, 0.01744158, 0.99569233, 0.75492814, 0.18086432,
        0.08656752, 0.2308163 , 0.59282938, 0.19743745, 0.42660109],
       [0.32924582, 0.4150288 , 0.57859145, 0.6149558 , 0.53820873,
        0.19931203, 0.65607217, 0.28211769, 0.83352103, 0.58258742],
       [0.23254694, 0.68225924, 0.78186269, 0.57694869, 0.83173009,
        0.35472178, 0.36449354, 0.89284405, 0.91499322, 0.06425951],
       [0.73592766, 0.27360411, 0.95119137, 0.98815034, 0.55370354,
        0.56049426, 0.43830378, 0.96593317, 0.29955691, 0.68167401],
       [0.21745093, 0.7416051 , 0.25682014, 0.60316199, 0.84949356,
        0.52336072, 0.90972974, 0.55037804, 0.34719856, 0.61546392],
       [0.31711393, 0.23880392, 0.35381217, 0.6659009 , 0.43079363,
        0.68543632, 0.66424645, 0.29418989, 0.25996643, 0.75056423],
       [0.49368331, 0.13685358, 0.96861356, 0.01393511, 0.84571714,
        0.7696917 , 0.50707332, 0.51243772, 0.16111963, 0.45612536]])
```

In [70]:

```
from numpy.random import random
m = random((10,10))
print(m)
print()
min, max = m.min(),m.max()
print(f"min({min}) max({max})")
```

```
[[0.10980619 0.5296275  0.22898299 0.4724025  0.32766893 0.09140331
  0.28219089 0.31456141 0.02958125 0.44212124]
 [0.80319496 0.05842923 0.8622534  0.45785831 0.48112221 0.40231849
  0.7476394  0.34342531 0.85935245 0.67932804]
 [0.13551276 0.1425882  0.83040608 0.90710946 0.94639696 0.60545792
  0.59859804 0.97168941 0.21910756 0.42605095]
 [0.78016895 0.12209308 0.15657801 0.4709588  0.80165655 0.34624676
  0.61209621 0.4402181  0.68845464 0.65586524]
 [0.57433175 0.06966152 0.36851505 0.64436717 0.11619354 0.3671595
  0.93321961 0.21594684 0.94171978 0.47944651]
 [0.87380147 0.09328027 0.56961609 0.27911327 0.84285028 0.92413628
  0.79385483 0.64772649 0.35663204 0.81221211]
 [0.2831061  0.93355496 0.53224113 0.87784505 0.51360894 0.92250279
  0.13571457 0.2065226  0.23124718 0.60854912]
 [0.90345557 0.85811973 0.55445728 0.38166728 0.21030222 0.02760191
  0.16202876 0.39473162 0.21936328 0.63137178]
 [0.00975695 0.13325815 0.52049704 0.72317936 0.18601513 0.44873153
  0.51779775 0.67982882 0.72450908 0.52356538]
 [0.9709759  0.78294221 0.88568958 0.30225871 0.12166975 0.24804119
  0.96896867 0.75056054 0.27559984 0.31934189]]

min(0.009756952552697262) max(0.9716894080217193)
```

In [71]:

```
from numpy.random import random
vector = random(30)
print(vector)
print()
print(f"mean({vector.mean()})")
```

```
[0.52011071 0.89600342 0.32903261 0.10346471 0.63935482 0.76268036
 0.93388819 0.66740469 0.5295406  0.81322069 0.17746392 0.79114431
 0.31411425 0.32315412 0.6718197  0.77271262 0.10325337 0.62413092
 0.87139708 0.81784044 0.35266441 0.90845829 0.7723605  0.2815934
 0.09825032 0.75449495 0.76376988 0.54598871 0.5670441  0.90779014]

mean(0.5871382073488072)
```

In [78]:

```
x = np.ones((5,5))
print("Original array:")
print(x)
print("1 on the border and 0 inside in the array")
x[1:-1,1:-1] = 0
print(x)
```

```
Original array:
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
1 on the border and 0 inside in the array
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

In [79]:

```
y = np.ones((3,3))
print("Original array:")
print(y)
print("0 on the border and 1 inside in the array")
y = np.pad(y, pad_width=1, mode='constant', constant_values=0)
print(y)
```

```
Original array:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
0 on the border and 1 inside in the array
[[0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0.]]
```

In [99]:

```
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
```

In [85]:

```
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]]
```

In [83]:

```
y = np.ones((3,3))
print("Checkerboard pattern:")
y = np.zeros((8,8), dtype=int)
y[1::2,::2] = 1
y[::2,1::2] = 1
print(y)
```

```
Checkerboard pattern:
[[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]]
```

In [98]:

```
print(np.unravel_index(99, (6,7,8)))
```

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

In [90]:

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

```
z = np.tile(array,(4,4))
print(z)
```

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

In [91]:

```
z = np.random.random((5,5))
zmax, zmin = z.max(), z.min()
z = (z-zmin)/(zmax-zmin)
print(z)
```

```
[[0.44086235 0.41886218 0.83668995 0.13659531 0.51713332]
 [0.70135599 0.0721743  0.36032196 0.53873304 0.12623708]
 [0.83876726 1.         0.         0.76772999 0.11694327]
 [0.01509948 0.72289305 0.7474733  0.75337823 0.40682624]
 [0.23385396 0.05585925 0.63475077 0.31257803 0.00782802]]
```

In [101…

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

Out[101…

```
dtype([('r', 'u1'), ('g', 'u1'), ('b', 'u1'), ('a', 'u1')])
```

In [102…

```
z = np.ones((5,3)) @ np.ones((3,2))
z
```

Out[102…

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

In [104…

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

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

In [105…

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

```
9
10
```

In [106…

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23632/803206331.py in <module>
----> 1 Z**Z
```

```
      2 2 << Z >> 2
      3 Z <- Z
      4 1j*Z
      5 Z/1/1

ValueError: Integers to negative integer powers are not allowed.
```

In [110…

```
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\swara\AppData\Local\Temp/ipykernel_23632/3912170336.py:1: RuntimeWarning: in
valid value encountered in true_divide
  print(np.array(0) / np.array(0))
C:\Users\swara\AppData\Local\Temp/ipykernel_23632/3912170336.py:2: RuntimeWarning: di
vide by zero encountered in floor_divide
  print(np.array(0) // np.array(0))
```

In [115…

```
z = np.random.uniform(-10,+10,10)
print(np.copysign(np.ceil(np.abs(z)), z))
```

```
[-5.  2. -2. -6.  2.  9. -2.  4. -4.  4.]
```

In [120…

```
z1 = np.random.randint(0,10,10)
z2 = np.random.randint(0,10,10)
print(np.intersect1d(z1, z2))
```

```
[1 2 6 8]
```

In [122…

```
# Suicide mode on
defaults = np.seterr(all="ignore")
Z = np.ones(1) / 0

# Back to sanity
_ = np.seterr(**defaults)

An equivalent way, with a context manager:

with np.errstate(divide='ignore'):
    Z = np.ones(1) / 0
```

```
  File "C:\Users\swara\AppData\Local\Temp/ipykernel_23632/1205115598.py", line 8
    An equivalent way, with a context manager:
       ^
SyntaxError: invalid syntax
```

In [124…

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

```
C:\Users\swara\AppData\Local\Temp/ipykernel_23632/244602691.py:1: RuntimeWarning: inv
alid value encountered in sqrt
  np.sqrt(-1) == np.emath.sqrt(-1)
```

Out[124…

```
False
```

In [128…

```
yesterday = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
today     = np.datetime64('today', 'D')
tomorrow  = np.datetime64('today', 'D') + np.timedelta64(1, 'D')
```

In [129…

```
yesterday
```

Out[129… numpy.datetime64('2021-11-19')

In [130…

```
today
```

Out[130… numpy.datetime64('2021-11-20')

In [131…

```
tomorrow
```

Out[131… numpy.datetime64('2021-11-21')

In [133…

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

```
['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']
```

In [134…

```
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)
```

Out[134… array([-1.5, -1.5, -1.5])

In [137…

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

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

```
[0. 5. 2. 8. 0. 7. 9. 3. 0. 5.]
[0. 5. 2. 8. 0. 7. 9. 3. 0. 5.]
[0. 5. 2. 8. 0. 7. 9. 3. 0. 5.]
[0 5 2 8 0 7 9 3 0 5]
[0. 5. 2. 8. 0. 7. 9. 3. 0. 5.]
```

In [138…

```
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.]]
```

In [140…

```
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.]
```

In [143…

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

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

In [144…

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

```
[0.81147639 0.69331419 0.83266312 0.95096239 0.15798306 0.34550519
 0.17321206 0.20020583 0.73426706 0.75504522]
```

In [146…

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

Out[146…

```
45
```

In [147…

```
A = np.random.randint(0,2,5)
B = np.random.randint(0,2,5)
# Assuming identical shape of the arrays and a tolerance for the comparison of values
equal = np.allclose(A, B)
print(equal)
# Checking both the shape and the element values, no tolerance (values have to be exa
equal = np.array_equal(A, B)
print(equal)
```

```
False
False
```

In [149…

```
array = np.array([9,8,7,6,5,4,3,2,1,0])
print("Before: ",array)
array.flags.writeable = False
array[1] = 20
print("After: ",array)
```

```
Before:  [9 8 7 6 5 4 3 2 1 0]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23632/800247639.py in <module>
      2 print("Before: ",array)
      3 array.flags.writeable = False
----> 4 array[1] = 20
      5 print("After: ",array)

ValueError: assignment destination is read-only
```

In [151…

```
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)
```

```
[0.7995623  1.14938648 0.78643895 0.52663068 0.93558927 1.02653675
 0.30102227 0.20550368 0.83447084 0.34913663]
[0.38490664 0.65767429 0.57876911 1.47643714 0.52626985 0.97417233
 0.22011565 0.16896945 0.37967714 0.59897031]
```

In [153…

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

```
[0.00955572 0.60375138 0.10364639 0.35579176 0.61008004 0.63479358
 0.         0.35169187 0.60608155 0.5550654 ]
```

In [158…

```
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.  )]]
```

In [160…

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

```
3638.1636371179666
```

In [161…

```
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
```

In [167…

```
np.set_printoptions(threshold=np.inf)
Z = np.zeros((16,16))
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.]]
```

In [174…

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

```
75
```

In [180…

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

In [182…

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

```
[[0.         0.65114367 0.70354231 0.31764011 0.97063276 0.43891512
  1.00770585 0.69478514 0.96248516 0.69415891]
 [0.65114367 0.         0.24704938 0.62521649 0.52689502 0.50434509
  0.82419605 0.89427649 0.61767487 0.15649356]
 [0.70354231 0.24704938 0.         0.5584745  0.3035223  0.39170013
  0.57938601 0.72714339 0.37356966 0.094283  ]
 [0.31764011 0.62521649 0.5584745  0.         0.75792042 0.17957711
  0.70890589 0.3899474  0.71215178 0.59220154]
 [0.97063276 0.52689502 0.3035223  0.75792042 0.         0.57851823
  0.41367811 0.77886548 0.14892005 0.3745378 ]
 [0.43891512 0.50434509 0.39170013 0.17957711 0.57851823 0.
  0.57061881 0.40151406 0.53895611 0.43948547]
 [1.00770585 0.82419605 0.57938601 0.70890589 0.41367811 0.57061881
  0.         0.51851804 0.26476856 0.67351296]
 [0.69478514 0.89427649 0.72714339 0.3899474  0.77886548 0.40151406
  0.51851804 0.         0.66653243 0.80023136]
 [0.96248516 0.61767487 0.37356966 0.71215178 0.14892005 0.53895611
  0.26476856 0.66653243 0.         0.46120096]
 [0.69415891 0.15649356 0.094283   0.59220154 0.3745378  0.43948547
  0.67351296 0.80023136 0.46120096 0.        ]]
```

In [185…

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

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

In [191…

```python
from io import StringIO

# Fake file
s = StringIO("""1, 2, 3, 4, 5\n
                6,  ,  , 7, 8\n
                 ,  , 9,10,11\n""")
Z = np.genfromtxt(s, delimiter=",", dtype=np.int64)
print(Z)
```

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

In [192…

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

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

In [193…

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

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

In [198…

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

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

In [200…

```
Y = X - X.mean(axis=1, keepdims=True)
print(Y)
```

```
[[-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
```

```
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]
 [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
   0.33333333  0.55555556  0.77777778  1.        ]]
```

In [202…

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

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

In [203…

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

```
False
```

In [205…

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

```
0.49508485106996625
```

In [208…

```
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]]
```

In [209…

```
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 change")

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

```
range_10
```

In [210…

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

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

In [212…

```
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.]
```

In [213…

```
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(n)
```

```
8
```

In [214…

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

```
[[58 42 51 50]
 [45 54 56 43]
 [40 65 57 78]]
[[58 42 51 50]
 [45 54 56 43]
 [40 65 57 78]]
```

In [215…

```
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)
```

```
[0.55502349 0.60095661 0.64913299 0.50772901 0.38677288 0.38143003
 0.43504922 0.36120243 0.40066056 0.55048803]
```

In [217…

```
A = np.random.uniform(0,1,(5,5))
B = np.random.uniform(0,1,(5,5))
np.diag(np.dot(A, B))
```

Out[217…

```
array([0.47129773, 2.81775459, 1.2435638 , 1.2706859 , 0.26886163])
```

In [218…

```
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.]
```

In [219…

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

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

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

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

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

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

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

In [220…

```
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]]
```

In [221…

```
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)
```

```
[( 3, 59) ( 3, 95) ( 6, 34) ( 6, 55) (13, 13) (13, 73) (15, 18) (15, 75)
 (18, 75) (20, 23) (20, 92) (23, 51) (23, 92) (23, 97) (28, 70) (28, 76)
 (30, 38) (30, 92) (33, 56) (33, 99) (34, 55) (38, 92) (51, 97) (56, 99)
 (57, 61) (57, 88) (59, 95) (61, 88) (70, 76)]
```

In [222…

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

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

In [223…

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

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

In [224…

```
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]]
```

In [225…

```
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[225…

```
array([ 0.81831766,  0.65378255,  0.12786662,  0.42758492,  0.38770368,
        0.42744668,  0.25650595, -0.15369464, -0.53368847,  0.77305608,
        0.27427217,  0.39072616, -0.62727126, -0.55225297, -0.41559632,
        0.95475139, -0.05081362,  0.14993042,  0.52319269,  0.98399698,
       -0.19271792, -0.92236472,  0.46996971, -0.21915513,  0.90721619,
       -0.92377501, -0.84616154, -0.16062923, -0.69756783, -0.53519121,
        0.23532203, -0.83889538, -0.77518965,  0.75548208,  0.78889515,
        0.54268533, -0.05744078,  0.45831979, -0.75485958, -0.94660145,
       -0.28841714, -0.61061473, -0.82395332, -0.66021995,  0.30963342,
        0.57800091, -0.03353375,  0.46271508, -0.50428019,  0.48118836,
       -0.21300639, -0.85669101,  0.0287265 ,  0.58921002, -0.09013246,
        0.28143659,  0.28151586,  0.96949812,  0.13502695,  0.84471824,
       -0.21938008, -0.69964772,  0.87907635,  0.62935636, -0.35490869,
       -0.78190561,  0.83373839, -0.27166575, -0.00809003,  0.98628664,
        0.88152449,  0.45156749, -0.41784618, -0.17650656,  0.61635946,
        0.67023414,  0.24439234,  0.88573295,  0.34634473,  0.48101843,
        0.62533592, -0.42358306,  0.02490128,  0.18741708,  0.6006979 ,
       -0.35584601,  0.30550115, -0.29288197, -0.77110986, -0.73186007,
        0.46151023,  0.55379436,  0.04280915, -0.62582912, -0.06565201,
        0.23781245,  0.55730681,  0.51025762,  0.11900605,  0.38397726])
```

In [226…

```
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))
```

```
[3.90791175 0.98402574 1.74257451 8.62677561 9.9385562  9.62217554
 0.65712493 6.08019127 2.18754887 4.11198861]
```

In [227…

```
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]))
```

```
[[11.10752095 12.18708754  1.63640786 13.13941569  2.10051037  9.46050712
```

```
   3.55926205 11.03094954 12.04001068 12.56847938]
 [ 3.07500361  6.08732272  4.13551442  4.7556248   4.30722001  5.40960478
   9.36336196  3.82923123  4.80583816  5.87070462]
 [ 0.62523136  1.47381545  2.8540195   3.23985095  5.41034673 14.06789527
   2.41366873  8.5403353   1.0187441  11.52835917]
 [ 3.70882483  0.78238094  5.52104776  1.47639443  2.28915303  5.79432057
  10.78511788  0.65041305  2.20496428  3.37583761]
 [ 2.50889317  9.59183607 12.53196458  2.22185064 13.35144455  3.59091521
  17.74136311  3.32667525  6.35966099  1.84127045]
 [ 5.01712121  8.12932251  3.8917624   6.50258331  5.05507833  5.13267652
   9.10884342  4.61204648  6.8524301   6.44639213]
 [ 5.88653072  7.50534165  0.65919478  8.04418089  1.91494767  8.41572682
   5.87955501  7.56156376  6.9618456   9.53445869]
 [ 8.55783872  9.22344578  1.89074955 10.98561401  0.47971789 10.45056046
   3.32054429 10.4811674   9.20482102 12.33314899]
 [ 6.68746275  4.2247619   7.8133623  10.80052048  7.09073572 17.4336321
   2.57573392 14.85092624  5.64343093 17.35829894]
 [ 7.21287485  4.50812709  8.42570742 11.40600772  7.496299   17.95472115
   3.18957497 15.51737678  6.05597574 18.00917171]]
```

In [243…

```
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(position).astype(int)
Rs = np.array(R.shape).astype(int)
Zs = np.array(Z.shape).astype(int)
R_start = np.zeros((len(shape),)).astype(int)
R_stop  = np.array(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)
```

```
[[7 6 8 5 9 9 5 2 8 7]
 [8 2 4 3 4 4 1 7 6 2]
 [8 9 8 2 7 5 2 7 1 9]
 [2 5 4 2 7 8 8 7 8 9]
 [9 2 0 5 3 3 7 5 3 5]
 [1 7 2 6 8 2 8 4 5 9]
 [0 7 4 3 3 8 4 8 4 6]
 [0 8 9 8 3 6 7 2 0 5]
 [5 8 5 9 3 6 2 6 7 5]
 [4 5 6 6 9 9 7 4 8 1]]
[[0 0 0 0 0]
 [0 7 6 8 5]
 [0 8 2 4 3]
 [0 8 9 8 2]
 [0 2 5 4 2]]
C:\Users\swara\AppData\Local\Temp/ipykernel_23632/454330592.py:22: FutureWarning: Usi
ng a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(s
eq)]` instead of `arr[seq]`. In the future this will be interpreted as an array inde
x, `arr[np.array(seq)]`, which will result either in an error or a different result.
  R[r] = Z[z]
```

In [244…

```
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]]
```

In [245…

```
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
```

In [246…

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

```
4
```

In [247…

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

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

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

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

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

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

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

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


 [[[2 0 3]
```

```
   [0 0 0]
   [2 0 0]]

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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


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

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

 [[3 4 4]
```

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

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

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

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

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

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


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

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

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

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

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

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

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

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


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

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

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

 [[3 1 1]
```

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

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

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

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

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

In [249…

```
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  6  9 10  6]
 [ 6  5  0 17 11]
 [ 9  0  2 42  6]
 [10 17 42  7 12]
 [ 6 11  6 12  2]]
```

In [250…

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

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

In [251…

```
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.]]
```

In [252…

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

In [256…

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

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

In [257…

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

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

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

    return ix

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

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

In [258…

```
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)]
```

In [261…

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

%timeit np.power(x,3)
%timeit x*x*x
%timeit np.einsum('i,i,i->i',x,x,x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23632/829540478.py in <module>
----> 1 x = np.random.rand(5e7)
      2 
      3 get_ipython().run_line_magic('timeit', 'np.power(x,3)')
      4 get_ipython().run_line_magic('timeit', 'x*x*x')
      5 get_ipython().run_line_magic('timeit', "np.einsum('i,i,i->i',x,x,x)")

mtrand.pyx in numpy.random.mtrand.RandomState.rand()

mtrand.pyx in numpy.random.mtrand.RandomState.random_sample()

_common.pyx in numpy.random._common.double_fill()

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

In [264…

```
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 5 6 7]
```

In [265…

```
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)
```

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

In [266…

```
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]]
```

In [272…

```
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)

# Author: Andreas Kouzelis
# NumPy >= 1.13
uZ = np.unique(Z, axis=0)
print(uZ)
```

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

In [273…

```
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[273…

```
array([[0.07415631, 0.04048012, 0.01125297, 0.06634409, 0.06198242,
        0.00763816, 0.10694193, 0.09720132, 0.01515775, 0.01717743],
       [0.03227504, 0.01761816, 0.00489763, 0.02887493, 0.0269766 ,
        0.00332436, 0.04654432, 0.04230492, 0.00659711, 0.00747613],
       [0.26983193, 0.14729468, 0.04094609, 0.24140566, 0.22553489,
        0.02779293, 0.38912869, 0.35368561, 0.05515439, 0.06250335],
       [0.49549733, 0.27047992, 0.07519005, 0.44329765, 0.41415386,
        0.05103666, 0.71456417, 0.64947938, 0.10128101, 0.11477606],
       [0.33996466, 0.18557843, 0.05158849, 0.30415004, 0.28415426,
        0.03501666, 0.49026816, 0.44561297, 0.06948971, 0.07874877],
       [0.39119466, 0.21354363, 0.05936247, 0.34998306, 0.32697407,
        0.04029339, 0.56414771, 0.51276333, 0.07996126, 0.09061558],
       [0.54919916, 0.29979445, 0.08333912, 0.4913421 , 0.45903972,
        0.056568  , 0.79200839, 0.71986973, 0.11225781, 0.12721545],
       [0.0529132 , 0.02888403, 0.0080294 , 0.0473389 , 0.04422669,
        0.00545011, 0.07630692, 0.06935664, 0.0108156 , 0.01225671],
       [0.38634229, 0.21089484, 0.05862614, 0.34564187, 0.32291829,
        0.03979359, 0.55715003, 0.50640302, 0.07896942, 0.08949159],
       [0.4436153 , 0.24215879, 0.06731713, 0.39688129, 0.37078906,
        0.04569277, 0.63974431, 0.58147436, 0.09067618, 0.1027582 ]])
```

In [274…

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

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

In [275…

```
X = np.asarray([[1.0, 0.0, 3.0, 8.0],
                [2.0, 0.0, 1.0, 1.0],
                [1.5, 2.5, 1.0, 0.0]])
n = 4
```

```
M = np.logical_and.reduce(np.mod(X, 1) == 0, axis=-1)
M &= (X.sum(axis=-1) == n)
print(X[M])
```

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

In [294…

```
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)
```

```
[-0.23990496  0.1448907 ]
```

In [ ]: