100 numpy exercises

This is a collection of exercises that have been collected in the numpy mailing list, on stack overflow and in the numpy documentation. The goal of this collection is to offer a quick reference for both old and new users but also to provide a set of exercises for those who teach.

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

```
import numpy as np
```

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

```
np.__version__
```

```
'1.21.5'
```

```
np.show_config()
```

```
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/usr/local/lib']
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/usr/local/lib']
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/usr/local/lib']
lapack_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/usr/local/lib']
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_KNL,AVX512_KNM,AVX512_SKX,AVX512_CLX,AVX512_
```

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

```
np.zeros(10)
```

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

4. How to find the memory size of any array (★☆☆)

```
Z = np.zeros((10,10))
Z
```

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

```
Z.size
```

```
100
```

```
Z.itemsize
```

```
8
```

```
Z.size * Z.itemsize #MEMORY
```

```
800
```

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

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

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

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.]])
```

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

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

```
Z[4] = 1
```

```
Z
```

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

7. Create a vector with values ranging from 10 to 49 (★☆☆)

```
x=np.arange(10,50) #note , its 50 not 49
x
```

```
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])
```

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

```
x[::-1]
```

```
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])
```

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

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

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

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

```
x=np.array([1,2,0,0,4,0])
x
```

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

```
np.argwhere(x,0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-b848909557d6> in <module>()
----> 1 np.argwhere(x,0)
```

```
np.nonzero(x)
```

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

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

```
np.eye(3)
```

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

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

```
  np.random.randint(3,3,3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-5b54de9f47b6> in <module>()
----> 1 np.random.randint(3,3,3)

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

_bounded_integers.pyx in numpy.random._bounded_integers._rand_int64()

ValueError: low >= high
```

SEARCH STACK OVERFLOW

```
np.random.randint((3,3,(3,3,3))) #add one more braces inside
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: VisibleDeprecationWar
  """Entry point for launching an IPython kernel.
```

```
np.random.random((3,3,3))# not randint
```

```
array([[[0.6977442 , 0.20416829, 0.96702556],
        [0.05188695, 0.9145523 , 0.15437944],
        [0.99184318, 0.42600523, 0.20397499]],

       [[0.31650324, 0.04030352, 0.76402286],
        [0.34365502, 0.00759509, 0.26145993],
        [0.93664801, 0.59715833, 0.26614562]],

       [[0.25081411, 0.8024025 , 0.18072935],
        [0.34728064, 0.44297413, 0.217606  ],
        [0.51011187, 0.72585442, 0.09871864]]])
```

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

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

```
array([[0.01943133, 0.27069681, 0.60323781, 0.4770491 , 0.81409224,
        0.81515706, 0.46458096, 0.30596462, 0.40903368, 0.42713229],
       [0.50743919, 0.35666367, 0.45304385, 0.65049699, 0.61840875,
        0.04107798, 0.81020464, 0.0207848 , 0.67539884, 0.79388979],
       [0.89939263, 0.43325208, 0.17449046, 0.43622591, 0.51162517,
        0.96619454, 0.28931904, 0.35740686, 0.02418466, 0.29880814],
       [0.0675786 , 0.90845283, 0.20104986, 0.58595045, 0.14748858,
        0.95949125, 0.13856229, 0.60524566, 0.18702735, 0.21423269],
       [0.80964714, 0.69686054, 0.075854  , 0.83677016, 0.06325936,
        0.20895813, 0.90239827, 0.92038674, 0.57306975, 0.62490518],
       [0.53026323, 0.82890902, 0.88121336, 0.3972008 , 0.00218967,
        0.71802561, 0.19867863, 0.3426098 , 0.89399731, 0.73972333],
       [0.63602029, 0.46900905, 0.28153368, 0.14968016, 0.7876176 ,
        0.45874501, 0.03961024, 0.60796428, 0.92691516, 0.75090294],
       [0.88723104, 0.7423842 , 0.33755016, 0.51336954, 0.89118285,
        0.56955187, 0.74415205, 0.17826542, 0.99767183, 0.13952729],
       [0.87434532, 0.68991009, 0.82028658, 0.81968729, 0.73874594,
        0.51547079, 0.33693871, 0.54375905, 0.7492097 , 0.16137994],
       [0.11580309, 0.71659424, 0.02192274, 0.70064924, 0.57749201,
        0.65041531, 0.37583802, 0.70517274, 0.21769918, 0.15524889]])
```

```
np.max(x)
```

```
0.9976718290757352
```

```
np.min(x)
```

```
0.002189671132236115
```

ALTERNATE WAY

```
x.min()
```

```
0.002189671132236115
```

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

```
np.random.random(30)
```

```
array([0.88965471, 0.79320441, 0.30567681, 0.43386822, 0.77535678,
       0.44518492, 0.07049122, 0.1221952 , 0.07174543, 0.29354383,
       0.75284719, 0.52326025, 0.85849464, 0.40810945, 0.03078085,
       0.90546776, 0.9585818 , 0.5114471 , 0.62214248, 0.08599294,
       0.65528599, 0.11015828, 0.56968571, 0.12403381, 0.63935414,
       0.92474963, 0.77398099, 0.82708799, 0.04930414, 0.89719595])
```

```
np.mean(np.random.random(30))
```

```
0.5459374272003783
```

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

```
Z = np.ones((10,10))
Z
```

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

```
Z[1:-1,1:-1]=0
```

```
Z
```

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

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

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

```
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.]])
```

```
np.pad(Z, pad_width=1, mode='constant', constant_values=0)
```

```
array([[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? (★☆☆)

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

NaN = not a number

inf = infinity

```
np.nan
```

```
nan
```

```
0 * np.nan
```

```
nan
```

```
np.nan == np.nan #no 2 nulls are same
```

```
False
```

```
np.inf
```

```
inf
```

```
np.inf > np.nan
```

```
False
```

```
np.nan - np.nan
```

```
nan
```

```
np.nan in set([np.nan])
```

```
True
```

```
0.3 == 3 * 0.1
```

```
False
```

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

```
np.diag(np.arange(1,5),k=-1)
```

```
array([[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 (★☆☆)

```
Z = np.zeros((8,8),dtype=int)
Z
```

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

```
Z[::2,1::2] = 1
Z
```

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

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

```
np.unravel_index(99,(6,7,8)) #note99 for 100th element
```

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

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

```
  np.tile( np.array([[0,1],[1,0]]), (8,8))
```

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

```
np.tile( np.array([[0,8],[8,0]]), (8,8))
```

```
array([[0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0],
       [0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8],
       [8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0, 8, 0]])
```

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

```
array([[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 (★☆☆)

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

```
array([[0.54591316, 0.16521022, 0.51043447, 0.56330583, 0.43894501],
       [0.95418016, 0.38359994, 0.57440997, 0.47777888, 0.42097204],
       [0.08315526, 0.39284743, 0.25396442, 0.15768061, 0.17499726],
       [0.95486611, 0.19951392, 0.10455624, 0.83216696, 0.12909813],
       [0.07935674, 0.06974181, 0.94367251, 0.2720761 , 0.14612555]])
```

```
Z = (Z - np.mean (Z)) / (np.std (Z))
Z
```

```
array([[ 5.43557258e-01, -8.10984116e-01,  4.17324033e-01,
         6.05440382e-01,  1.62964482e-01],
       [ 1.99617157e+00, -3.39533333e-02,  6.44948914e-01,
         3.01135439e-01,  9.90166424e-02],
       [-1.10293575e+00, -1.05073294e-03, -4.95196595e-01,
        -8.37774497e-01, -7.76161838e-01],
       [ 1.99861216e+00, -6.88931536e-01, -1.02679104e+00,
         1.56204851e+00, -9.39470989e-01],
       [-1.11645089e+00, -1.15066079e+00,  1.95878536e+00,
        -4.30755248e-01, -8.78887385e-01]])
```

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

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

```
color
```

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

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

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

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

```
(np.ones((5,3))).dot(np.ones((3,2))) #alternate
```

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

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

```
Z = np.arange(11)
Z
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
Z[(3 < Z) & (Z < 8)]=-1 *Z[(3 < Z) & (Z < 8)]
Z
```

```
array([ 0,  1,  2,  3, -4, -5, -6, -7,  8,  9, 10])
```

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

```
# Author: Jake VanderPlas

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

```
range(5) #1,2,3,4
```

```
range(0, 5)
```

```
sum(range(5),-1) #
```

```
9
```

```
from numpy import *
range(5)
```

```
    range(0, 5)
```

```
sum(range(5),-1)
```

```
    10
```

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
```

```
Z
```

```
    array([ 0,  1,  2,  3, -4, -5, -6, -7,  8,  9, 10])
```

```
  Z**Z
```

```
    ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)
    <ipython-input-65-8f6ea73f5220> in <module>()
    ----> 1 Z**Z

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

SEARCH STACK OVERFLOW

```
Z=array([ 0,  1,  2,  3, 4, 5, 6, 7,  8,  9, 10])
Z**Z
##just bcz -ve in above case it was not allowed
```

```
    array([          1,           1,           4,          27,         256,
                  3125,       46656,      823543,    16777216,   387420489,
           10000000000])
```

```
2 << Z #ask sir doubt
```

```
    array([   2,    4,    8,   16,   32,   64,  128,  256,  512, 1024, 2048])
```

```
Z >>2
```

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

```
Z=array([ 0,  1,  2,  3, 4, 5, 6, 7,  8,  9, 10])
2 << Z >>2
```

```
array([  0,   1,   2,   4,   8,  16,  32,  64, 128, 256, 512])
```

```
Z <- Z
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False])
```

```
Z>-Z
```

```
array([False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

```
1j*Z
```

```
array([0. +0.j, 0. +1.j, 0. +2.j, 0. +3.j, 0. +4.j, 0. +5.j, 0. +6.j,
       0. +7.j, 0. +8.j, 0. +9.j, 0.+10.j])
```

```
Z/1/1
```

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

```
  Z<Z>Z
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-74-6d2bd9eb1fd1> in <module>()
----> 1 Z<Z>Z

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

SEARCH STACK OVERFLOW

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)
```

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

```
array(0)
```

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: inva
  """Entry point for launching an IPython kernel.
```

```
nan
```

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divic
  """Entry point for launching an IPython kernel.
0
```

```
np.nan
```

```
nan
```

```
np.array([np.nan])
```

```
array([nan])
```

```
np.array([np.nan]).astype(int)
```

```
array([-9223372036854775808])
```

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

```
array([-9.22337204e+18])
```

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

```
Z = np.random.uniform(-10,+10,10)
Z
```

```
array([ 3.71843579, -9.22626642,  7.33354825, -0.02527729,  5.76493224,
       -0.07557423, -8.6721194 ,  9.83425666,  2.76948343, -5.9170587 ])
```

```
Z = np.random.uniform(-10,+10,10)
```

```
np.where(Z>0,np.floor(Z))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-84-dcda33d4d0e3> in <module>()
----> 1 np.where(Z>0,np.floor(Z))

<__array_function__ internals> in where(*args, **kwargs)

ValueError: either both or neither of x and y should be given
```

SEARCH STACK OVERFLOW

```
np.where(Z>0, np.ceil(Z), np.floor(Z))
```

```
array([ 8.,  6.,  6., -7., -3.,  6., -6.,  6., -4.,  2.])
```

```
Z
```

```
array([ 7.16883033,  5.09910127,  5.16203894, -6.90894365, -2.83108999,
        5.08949344, -5.9822664 ,  5.87847291, -3.40688992,  1.60253243])
```

```
#Alternate way
np.abs(Z)
```

```
array([7.16883033, 5.09910127, 5.16203894, 6.90894365, 2.83108999,
       5.08949344, 5.9822664 , 5.87847291, 3.40688992, 1.60253243])
```

```
np.ceil(np.abs(Z))
```

```
array([8., 6., 6., 7., 3., 6., 6., 6., 4., 2.])
```

```
np.copysign(np.ceil(np.abs(Z)), Z)
```

```
array([ 8.,  6.,  6., -7., -3.,  6., -6.,  6., -4.,  2.])
```

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

```
Z1 = np.random.randint(0,10,10)
Z1
```

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

```
Z2 = np.random.randint(0,10,10)
Z2
```

```
array([2, 4, 8, 9, 1, 2, 5, 4, 2, 4])
```

```
np.intersect1d(Z1,Z2)
```

```
array([2, 5, 8])
```

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

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

```
Z
```

```
array([inf])
```

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

```
#for a section of code
# Equivalently with a context manager
with np.errstate(all="ignore"):
    np.arange(3) / 0
```

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

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

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: inval
  """Entry point for launching an IPython kernel.
nan
```

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

```
1j
```

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: inval
  """Entry point for launching an IPython kernel.
False
```

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

```
yesterday = np.datetime64('today') - 1
yesterday
```

```
numpy.datetime64('2022-04-14')
```

```
today     = np.datetime64('today')
today
```

```
numpy.datetime64('2022-04-15')
```

```
tomorrow  = np.datetime64('today') + 1
tomorrow
```

```
numpy.datetime64('2022-04-16')
```

(★★☆)

(★★★)

CAN IGNORE IF NO CONCEPTUAL CLARITY , AS THEY ARE PURE TECHNICAL AND SCIENTIFIC JUST SKIM THROUGH

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

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

```
array(['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'], dtype='datetime64[D]')
```

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

```
  A = np.ones(3)*1
  B = np.ones(3)*2
  A
```

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

```
B
```

```
array([2., 2., 2.])
```

```
#A+B=A
np.add(A,B,out=B)
```

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

```
#-A/2
np.divide(-A,2,out=A)
```

```
array([-0.5, -0.5, -0.5])
```

```
np.multiply(A,B,out=A)
```

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

36. Extract the integer part of a random array of positive numbers using 4 different methods (★★☆)

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

```
Z
```

```
array([7.44671072, 2.78830268, 4.9326987 , 1.25740948, 4.85246936,
       1.23571306, 6.18399196, 3.4015102 , 3.05842036, 6.51827369])
```

```
Z - Z%1
```

```
array([7., 2., 4., 1., 4., 1., 6., 3., 3., 6.])
```

```
Z // 1
```

```
array([7., 2., 4., 1., 4., 1., 6., 3., 3., 6.])
```

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

```
array([7., 2., 4., 1., 4., 1., 6., 3., 3., 6.])
```

```
Z.astype(int)
```

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

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

```
array([7., 2., 4., 1., 4., 1., 6., 3., 3., 6.])
```

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

```
np.tile(np.arange(0, 5), (5,1))
```

```
array([[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]])
```

```
#alternate
np.zeros(5,5) #wrong add one more braces inside
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-117-b1bc850401c5> in <module>()
      1 #alternate
----> 2 np.zeros(5,5)

TypeError: Cannot interpret '5' as a data type
```

SEARCH STACK OVERFLOW

```
np.zeros((5,5))+np.arange(5)
```

```
array([[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 (★☆☆)

```
def generate():
    for x in range(10):
        yield x
```

```
Z = np.fromiter(generate(),dtype=int,count=-1)
Z
```

```
array([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 (★★☆)

```
np.linspace(0,1,11)
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

```
np.linspace(0,1,11,endpoint=False)
```

```
array([0.        , 0.09090909, 0.18181818, 0.27272727, 0.36363636,
       0.45454545, 0.54545455, 0.63636364, 0.72727273, 0.81818182,
       0.90909091])
```

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

```
array([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 (★★☆)

```
np.sort(np.random.random(10))
```

```
array([0.44484896, 0.62886842, 0.69026463, 0.69781374, 0.70779775,
       0.74350698, 0.84397244, 0.93840938, 0.94732511, 0.9496494 ])
```

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

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

```
45
```

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

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

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

```
B
```

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

```
# Assuming identical shape of the arrays and a tolerance for the comparison of values
np.allclose(A,B)
```

```
True
```

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

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

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

```
Z.flags.writeable = False
```

```
Z[0] = 1
```

```
---------------------------------------------------------------------------
```

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

```
Z = np.random.random((10,2))
Z
```

```
array([[0.50427402, 0.61182224],
       [0.17394129, 0.52922144],
       [0.06852744, 0.13679527],
       [0.48231202, 0.34693225],
       [0.43004733, 0.30625353],
       [0.44964221, 0.71459662],
       [0.56589609, 0.42598812],
       [0.50978139, 0.19326762],
       [0.81652672, 0.9521585 ],
       [0.51891606, 0.04090298]])
```

```
X,Y = Z[:,0], Z[:,1]
X,Y
```

```
(array([0.50427402, 0.17394129, 0.06852744, 0.48231202, 0.43004733,
        0.44964221, 0.56589609, 0.50978139, 0.81652672, 0.51891606]),
 array([0.61182224, 0.52922144, 0.13679527, 0.34693225, 0.30625353,
        0.71459662, 0.42598812, 0.19326762, 0.9521585 , 0.04090298]))
```

```
R = np.sqrt(X**2+Y**2)
T = np.arctan2(Y,X)
R,T
```

```
(array([0.7928548 , 0.55707352, 0.15299986, 0.59412698, 0.52795069,
        0.8442905 , 0.70831085, 0.54518752, 1.25432121, 0.52052563]),
 array([0.88146262, 1.25324508, 1.10638988, 0.62356901, 0.61882666,
        1.00916443, 0.64526988, 0.36237663, 0.86193346, 0.07866123]))
```

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

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

```
array([0.67066294, 0.43473416, 0.1677296 , 0.22958723, 0.56459298,
       0.14903044, 0.34830234, 0.62671201, 0.04445742, 0.74295259])
```

```
Z[Z.argmax()] = 0
Z
```

```
array([0.67066294, 0.43473416, 0.1677296 , 0.22958723, 0.56459298,
       0.14903044, 0.34830234, 0.62671201, 0.04445742, 0.        ])
```

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

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

```
array([[(0., 0.), (0., 0.), (0., 0.), (0., 0.), (0., 0.)],
       [(0., 0.), (0., 0.), (0., 0.), (0., 0.), (0., 0.)],
       [(0., 0.), (0., 0.), (0., 0.), (0., 0.), (0., 0.)],
       [(0., 0.), (0., 0.), (0., 0.), (0., 0.), (0., 0.)],
       [(0., 0.), (0., 0.), (0., 0.), (0., 0.), (0., 0.)]],
      dtype=[('x', '<f8'), ('y', '<f8')])
```

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

```
array([[(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.  )]], dtype=[('x', '<f8'), ('y', '<f8')])
```

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

```
X = np.arange(8)
Y = X + 0.5
X,Y
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5]))
```

```
C = 1.0 / np.subtract.outer(X, Y)
C
```

```
array([[-2.        , -0.66666667, -0.4       , -0.28571429, -0.22222222,
        -0.18181818, -0.15384615, -0.13333333],
       [ 2.        , -2.        , -0.66666667, -0.4       , -0.28571429,
        -0.22222222, -0.18181818, -0.15384615],
       [ 0.66666667,  2.        , -2.        , -0.66666667, -0.4       ,
        -0.28571429, -0.22222222, -0.18181818],
       [ 0.4       ,  0.66666667,  2.        , -2.        , -0.66666667,
        -0.4       , -0.28571429, -0.22222222],
       [ 0.28571429,  0.4       ,  0.66666667,  2.        , -2.        ,
        -0.66666667, -0.4       , -0.28571429],
       [ 0.22222222,  0.28571429,  0.4       ,  0.66666667,  2.        ,
        -2.        , -0.66666667, -0.4       ],
       [ 0.18181818,  0.22222222,  0.28571429,  0.4       ,  0.66666667,
         2.        , -2.        , -0.66666667],
       [ 0.15384615,  0.18181818,  0.22222222,  0.28571429,  0.4       ,
         0.66666667,  2.        , -2.        ]])
```

```
np.linalg.det(C)
```

```
3638.163637117973
```

48. Print the minimum and maximum representable value for each numpy scalar type (★★☆)

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

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

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

```
-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? (★★☆)

```
np.set_printoptions(threshold=float("inf"))
Z = np.zeros((40,40))
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.]])
```

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

```
Z = np.arange(100)
Z
```

```
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])
```

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

```
46.64555981718884
```

```
index = (np.abs(Z-v)).argmin()
index
```

```
47
```

```
Z[index]
```

```
47
```

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

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: Passi

array([((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
       ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
       ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
       ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.)),
       ((0., 0.), (0., 0., 0.)), ((0., 0.), (0., 0., 0.))],
      dtype=[('position', [('x', '<f8'), ('y', '<f8')]), ('color', [('r', '<f8'), ('
```

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

```
Z = np.random.random((10,2))
Z
```

```
array([[0.06274621, 0.18303131],
       [0.80361964, 0.60441334],
       [0.93047263, 0.75502316],
       [0.24379998, 0.38540171],
       [0.79276932, 0.65975361],
       [0.69058351, 0.91076353],
       [0.1054969 , 0.80666616],
       [0.88149715, 0.81914384],
       [0.53718953, 0.01477258],
       [0.79047848, 0.61992075]])
```

```
X,Y = np.atleast_2d(Z[:,0], Z[:,1])
X,Y
```

```
(array([[0.06274621, 0.80361964, 0.93047263, 0.24379998, 0.79276932,
         0.69058351, 0.1054969 , 0.88149715, 0.53718953, 0.79047848]]),
 array([[0.18303131, 0.60441334, 0.75502316, 0.38540171, 0.65975361,
         0.91076353, 0.80666616, 0.81914384, 0.01477258, 0.61992075]]))
```

```
D = np.sqrt( (X-X.T)**2 + (Y-Y.T)**2)
D
```

```
array([[0.        , 0.85232403, 1.03929006, 0.27154051, 0.87189328,
        0.96113155, 0.62509843, 1.03681834, 0.50339594, 0.84880307],
       [0.85232403, 0.        , 0.19691369, 0.60113571, 0.05639393,
        0.32653883, 0.7268298 , 0.2284165 , 0.64704036, 0.02032658],
       [1.03929006, 0.19691369, 0.        , 0.7798329 , 0.16744697,
        0.28601024, 0.82659057, 0.08068495, 0.83823774, 0.19455339],
       [0.27154051, 0.60113571, 0.7798329 , 0.        , 0.61370702,
        0.6896525 , 0.44338638, 0.77122624, 0.47269798, 0.59485844],
       [0.87189328, 0.05639393, 0.16744697, 0.61370702, 0.        ,
        0.27101277, 0.70279917, 0.18242224, 0.69377342, 0.03989868],
       [0.96113155, 0.32653883, 0.28601024, 0.6896525 , 0.27101277,
        0.        , 0.59427485, 0.21175974, 0.90902668, 0.30751996],
       [0.62509843, 0.7268298 , 0.82659057, 0.44338638, 0.70279917,
        0.59427485, 0.        , 0.77610056, 0.90191683, 0.70998142],
       [1.03681834, 0.2284165 , 0.08068495, 0.77122624, 0.18242224,
        0.21175974, 0.77610056, 0.        , 0.87496335, 0.21903022],
       [0.50339594, 0.64704036, 0.83823774, 0.47269798, 0.69377342,
        0.90902668, 0.90191683, 0.87496335, 0.        , 0.65601799],
       [0.84880307, 0.02032658, 0.19455339, 0.59485844, 0.03989868,
        0.30751996, 0.70998142, 0.21903022, 0.65601799, 0.        ]])
```

```
# Much faster with scipy
import scipy
import scipy.spatial

D = scipy.spatial.distance.cdist(Z,Z)
D
```

```
array([[0.        , 0.85232403, 1.03929006, 0.27154051, 0.87189328,
        0.96113155, 0.62509843, 1.03681834, 0.50339594, 0.84880307],
       [0.85232403, 0.        , 0.19691369, 0.60113571, 0.05639393,
        0.32653883, 0.7268298 , 0.2284165 , 0.64704036, 0.02032658],
       [1.03929006, 0.19691369, 0.        , 0.7798329 , 0.16744697,
        0.28601024, 0.82659057, 0.08068495, 0.83823774, 0.19455339],
       [0.27154051, 0.60113571, 0.7798329 , 0.        , 0.61370702,
        0.6896525 , 0.44338638, 0.77122624, 0.47269798, 0.59485844],
       [0.87189328, 0.05639393, 0.16744697, 0.61370702, 0.        ,
        0.27101277, 0.70279917, 0.18242224, 0.69377342, 0.03989868],
       [0.96113155, 0.32653883, 0.28601024, 0.6896525 , 0.27101277,
        0.        , 0.59427485, 0.21175974, 0.90902668, 0.30751996],
       [0.62509843, 0.7268298 , 0.82659057, 0.44338638, 0.70279917,
        0.59427485, 0.        , 0.77610056, 0.90191683, 0.70998142],
       [1.03681834, 0.2284165 , 0.08068495, 0.77122624, 0.18242224,
        0.21175974, 0.77610056, 0.        , 0.87496335, 0.21903022],
       [0.50339594, 0.64704036, 0.83823774, 0.47269798, 0.69377342,
        0.90902668, 0.90191683, 0.87496335, 0.        , 0.65601799],
       [0.84880307, 0.02032658, 0.19455339, 0.59485844, 0.03989868,
        0.30751996, 0.70998142, 0.21903022, 0.65601799, 0.        ]])
```

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

```
Z = (np.random.rand(10)*100).astype(np.float32)
Z
```

```
array([11.459753, 99.0671  , 71.6015  , 92.5468  , 96.90511 , 38.297672,
       74.80695 , 23.247725, 19.38841 , 18.960337], dtype=float32)
```

```
Y = Z.view(np.int32)
Y
```

```
array([1094146854, 1120281179, 1116681208, 1119426550, 1119997803,
       1108947153, 1117101353, 1102707543, 1100684151, 1100459717],
      dtype=int32)
```

```
Y[:] = Z
```

```
Y
```

```
array([11, 99, 71, 92, 96, 38, 74, 23, 19, 18], dtype=int32)
```

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

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

```
from io import StringIO

# Fake file
s = StringIO('''1, 2, 3, 4, 5
                6,  ,  , 7, 8
                 ,  , 9,10,11
''')
s
```

```
<_io.StringIO at 0x7fe82b3aeeb0>
```

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: `
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
```

```
  """Entry point for launching an IPython kernel.
array([[ 1,  2,  3,  4,  5],
       [ 6, -1, -1,  7,  8],
       [-1, -1,  9, 10, 11]])
```

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

```
Z = np.arange(9).reshape(3,3)
Z
```

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

```
for index, value in np.ndenumerate(Z):
    print(index, value)
```

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

```
  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
```

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

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

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

```
D = np.sqrt(X*X+Y*Y)
D
```

```
array([[1.41421356, 1.26686158, 1.1439589 , 1.05409255, 1.0061539 ,
        1.0061539 , 1.05409255, 1.1439589 , 1.26686158, 1.41421356],
       [1.26686158, 1.09994388, 0.95581392, 0.84619701, 0.7856742 ,
        0.7856742 , 0.84619701, 0.95581392, 1.09994388, 1.26686158],
       [1.1439589 , 0.95581392, 0.7856742 , 0.64788354, 0.56655772,
        0.56655772, 0.64788354, 0.7856742 , 0.95581392, 1.1439589 ],
       [1.05409255, 0.84619701, 0.64788354, 0.47140452, 0.35136418,
        0.35136418, 0.47140452, 0.64788354, 0.84619701, 1.05409255],
       [1.0061539 , 0.7856742 , 0.56655772, 0.35136418, 0.15713484,
        0.15713484, 0.35136418, 0.56655772, 0.7856742 , 1.0061539 ],
       [1.0061539 , 0.7856742 , 0.56655772, 0.35136418, 0.15713484,
        0.15713484, 0.35136418, 0.56655772, 0.7856742 , 1.0061539 ],
       [1.05409255, 0.84619701, 0.64788354, 0.47140452, 0.35136418,
        0.35136418, 0.47140452, 0.64788354, 0.84619701, 1.05409255],
       [1.1439589 , 0.95581392, 0.7856742 , 0.64788354, 0.56655772,
        0.56655772, 0.64788354, 0.7856742 , 0.95581392, 1.1439589 ],
       [1.26686158, 1.09994388, 0.95581392, 0.84619701, 0.7856742 ,
        0.7856742 , 0.84619701, 0.95581392, 1.09994388, 1.26686158],
       [1.41421356, 1.26686158, 1.1439589 , 1.05409255, 1.0061539 ,
        1.0061539 , 1.05409255, 1.1439589 , 1.26686158, 1.41421356]])
```

```
sigma, mu = 1.0, 0.0
sigma, mu
```

```
(1.0, 0.0)
```

```
  G = np.exp(-( (D-mu)**2 / ( 2.0 * sigma**2 ) ) )
  G
```

```
array([[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? (★★☆)

```
n = 10
p = 3
Z = np.zeros((n,n))
Z
```

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

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

```
  Z
```

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

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

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

```
array([[0.22837235, 0.35785663, 0.51645213, 0.21298743, 0.81100656,
        0.38225686, 0.34647627, 0.23096201, 0.07891913, 0.3449573 ],
       [0.49329889, 0.23617302, 0.41487764, 0.0649475 , 0.69812518,
        0.6895348 , 0.26457269, 0.11574557, 0.1240249 , 0.03031892],
       [0.16073547, 0.17612508, 0.97657463, 0.99765113, 0.90837821,
        0.01672011, 0.76532456, 0.81899095, 0.02519778, 0.1528949 ],
       [0.95068524, 0.2404214 , 0.12211532, 0.13763248, 0.00869075,
        0.52484418, 0.3655848 , 0.69020284, 0.36783387, 0.19859371],
       [0.65288963, 0.17056715, 0.55028956, 0.25840999, 0.75861344,
        0.9173282 , 0.69489207, 0.87260123, 0.39397592, 0.96488651]])
```

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

```
array([[-0.12265231,  0.00683196,  0.16542747, -0.13803724,  0.4599819 ,
         0.03123219, -0.00454839, -0.12006266, -0.27210554, -0.00606737],
       [ 0.18013698, -0.07698889,  0.10171573, -0.24821441,  0.38496326,
         0.37637289, -0.04858922, -0.19741635, -0.18913701, -0.28284299],
       [-0.33912381, -0.32373421,  0.47671535,  0.49779185,  0.40851893,
        -0.48313917,  0.26546528,  0.31913167, -0.47466151, -0.34696439],
       [ 0.59002478, -0.12023906, -0.23854514, -0.22302798, -0.35196971,
         0.16418372,  0.00492434,  0.32954238,  0.00717341, -0.16206675],
       [ 0.02944426, -0.45287822, -0.07315581, -0.36503538,  0.13516806,
         0.29388283,  0.0714467 ,  0.24915586, -0.22946945,  0.34144114]])
```

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

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

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

```
Z[Z[:,1].argsort()]
```

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

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

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

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

```
(~Z.any(axis=0))
```

```
array([False, False, False, False, False, False, False, False, False,
       False])
```

```
(~Z.any(axis=0)).any()
```

```
False
```

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

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

```
array([0.67364564, 0.83130718, 0.82469004, 0.57070424, 0.04665237,
       0.03436864, 0.4747631 , 0.28797376, 0.19240777, 0.20023882])
```

```
z = 0.5
```

```
m = Z.flat[np.abs(Z - z).argmin()]
m
```

```
0.4747631015876774
```

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

```
A = np.arange(3).reshape(3,1)
B = np.arange(3).reshape(1,3)
A,B
```

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

```
it = np.nditer([A,B,None])
```

```
it
```

```
<numpy.nditer at 0x7fe81dfeec10>
```

```
for x,y,z in it:
    z[...] = x + y
it
```

```
<numpy.nditer at 0x7fe81dfeec10>
```

```
it.operands
```

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

```
(it.operands[2])
```

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

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

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

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

```
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)? (★★★)

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

```

```
[2. 4. 2. 2. 6. 5. 2. 3. 2. 2.]
```

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

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

```
array([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 (★★☆)

```

```

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

```

```

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? (★★★)

```

```

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

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

A,B
```

```
(array([[0.99885454, 0.08079166, 0.47248997, 0.74926622, 0.80444523],
        [0.54533218, 0.3798197 , 0.44502215, 0.91648091, 0.22940312],
        [0.06875742, 0.06412907, 0.86885477, 0.9406479 , 0.86498716],
        [0.37963295, 0.69559779, 0.40579072, 0.07828983, 0.66638694],
        [0.94639669, 0.86420745, 0.09070027, 0.22505297, 0.28971566]]),
 array([[0.24238051, 0.77761457, 0.09568111, 0.61851717, 0.92890418],
        [0.04880672, 0.97970341, 0.70366773, 0.69161396, 0.80387524],
        [0.21348037, 0.09522553, 0.53355522, 0.80787989, 0.62261162],
        [0.46790341, 0.44630139, 0.83775784, 0.57806546, 0.17777794],
        [0.15116942, 0.46291305, 0.1729916 , 0.70770015, 0.28746587]]))
```

```
np.diag(np.dot(A, B))
```

```
array([0.81910512, 1.35376677, 1.45295701, 1.5605836 , 1.75359068])
```

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

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

```
array([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)? (★★★)

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

```
array([[[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? (★★★)

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

```
    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]])
```

```
A[[0,1]] = A[[1,0]]
A
```

```
    array([[ 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 (★★★)

```

```

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

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

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

```
from numpy.lib.stride_tricks import sliding_window_view

Z = np.arange(20)
Z
```

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

```
sliding_window_view(Z, window_shape=3).mean(axis=-1)
```

```
    array([ 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]) (★★★)

[ ] ↳ *1 cell hidden*

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

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

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

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

```
array([-0.41273198,  0.46851447, -0.27292932, -0.62305076,  0.88398141,
        0.85442969, -0.16473009, -0.27316391,  0.42763054, -0.32536966,
       -0.98955585, -0.22633948, -0.64380144,  0.77268015,  0.27284333,
       -0.03231775,  0.75589104,  0.72323115,  0.68351703,  0.07189598,
        0.93442069,  0.36868421, -0.97532564,  0.39700952, -0.39724768,
        0.77404151, -0.4179374 , -0.02743144, -0.62543999,  0.2007554 ,
       -0.2223689 ,  0.14396035,  0.61653595, -0.43916179,  0.90580135,
       -0.6650992 , -0.19463045,  0.59895353,  0.97758044,  0.15007997,
        0.67968005,  0.01680675,  0.26794388,  0.27152838,  0.35083009,
       -0.62674535,  0.35107073,  0.51897092,  0.14355935, -0.35597032,
       -0.67654521, -0.28286937,  0.95290535, -0.94147351, -0.2215758 ,
       -0.38988694,  0.87481798,  0.55853355, -0.61170518,  0.31856737,
       -0.65844006, -0.35477487,  0.23850028,  0.43103639,  0.98970487,
       -0.00298793, -0.49630759, -0.05717953,  0.00321372,  0.58313569,
        0.74949367,  0.96286951, -0.41775429, -0.29600333,  0.34077737,
        0.08003323,  0.40868204,  0.69773172, -0.2915089 , -0.73797018,
       -0.14265238,  0.93933114,  0.76181367, -0.83553527,  0.86627663,
       -0.57578345,  0.219023  ,  0.18314615,  0.87184502, -0.40898657,
        0.71445   , -0.57803326,  0.54445464, -0.8684261 , -0.46143807,
        0.62143282, -0.54827184,  0.49337799, -0.2459928 ,  0.62125324])
```

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])? (★★★)

[ ] ↳ *1 cell hidden*

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])? (★★★)

[ ] ↳ *1 cell hidden*

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) (★★★)

[ ] ↳ *1 cell hidden*

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]]? (★★★)

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

# Author: Jeff Luo (@Jeff1999)

Z = np.arange(1, 15, dtype=np.uint32)
print(sliding_window_view(Z, window_shape=4))
```

82. Compute a matrix rank (★★★)

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

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

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

```
np.bincount(Z).argmax()
```

```
8
```

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

[ ] ↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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)) (★★★)

[ ] ↳ *1 cell hidden*

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

```
Z = np.ones((16,16))
k = 4
windows = np.lib.stride_tricks.sliding_window_view(Z, (k, k))
S = windows[::k, ::k, ...].sum(axis=(-2, -1))
S
```

```
array([[16., 16., 16., 16.],
       [16., 16., 16., 16.],
       [16., 16., 16., 16.],
       [16., 16., 16., 16.]])
```

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

[ ] ↳ *1 cell hidden*

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

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

```
array([9995, 9996, 9997, 9998, 9999])
```

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

[ ] ↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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

```
x = np.random.rand(int(1000))

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

```
The slowest run took 82.75 times longer than the fastest. This could mean that an int
10000 loops, best of 5: 76.1 µs per loop
The slowest run took 11.85 times longer than the fastest. This could mean that an int
100000 loops, best of 5: 2.13 µs per loop
The slowest run took 37.93 times longer than the fastest. This could mean that an int
100000 loops, best of 5: 4.34 µs per loop
```

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

[ ] ↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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

↳ *1 cell hidden*

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

[ ] ↳ *1 cell hidden*

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. (★★★)

[ ] ↳ *1 cell hidden*

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). (★★★)

[ ] ↳ *1 cell hidden*