Numpy

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

(**hint**: import … as …)

In [1]:

```
import numpy as np
```

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

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

In [2]:

```
print(np.__version__)
np.__config__.show()
```

```
1.18.5
blas_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['D:/My Files/New folder\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'D:/My Files/New folder\\Library\\include']
blas_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['D:/My Files/New folder\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'D:/My Files/New folder\\Library\\include']
lapack_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['D:/My Files/New folder\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'D:/My Files/New folder\\Library\\include']
lapack_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['D:/My Files/New folder\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'D:/My Files/New folder\\Library\\include']
```

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

(**hint**: np.zeros)

In [4]:

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

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

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

(**hint**: size, itemsize)

In [158]:

```
X = np.zeros((5,5))
print("%d bytes" % (X.size * X.itemsize))
```

```
200 bytes
```

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

(**hint**: np.info)

In [8]:

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

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

Add arguments element-wise.

Parameters
----------
x1, x2 : array_like
    The arrays to be added. If ``x1.shape != x2.shape``, they must be broa
dcastable 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.]])
```

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

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

In [9]:

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

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

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

(**hint**: np.arange)

In [10]:

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

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

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

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

In [12]:

```
X = np.arange(1,20)
X= X[::-1]
print (X)
```

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

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

(**hint**: reshape)

In [13]:

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

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

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

(**hint**: np.nonzero)

In [14]:

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

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

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

(**hint**: np.eye)

In [15]:

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

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

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

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

In [16]:

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

```
[[[0.44642352 0.28777704 0.6080942 ]
  [0.99759356 0.32836193 0.16071025]
  [0.29407163 0.33188006 0.28641725]]

 [[0.36144288 0.34894102 0.92472822]
  [0.9053719  0.57200124 0.76193557]
  [0.15264256 0.54673963 0.01255647]]

 [[0.59379955 0.01533628 0.78597118]
  [0.51846148 0.8476053  0.46402059]
  [0.03417058 0.70385018 0.15380986]]]
```

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

(**hint**: min, max)

In [17]:

```
X = np.random.random((10,10))
print (X)
Xmin, Xmax = X.min(), X.max()
print(Xmin, Xmax)
```

```
[[8.47029832e-01 7.32402339e-01 3.67617289e-01 4.40127607e-01
  5.48683412e-01 7.63470157e-01 5.89844209e-01 4.48436038e-01
  6.28744468e-01 3.95043083e-01]
 [8.28731489e-01 9.40494486e-01 2.04607882e-01 2.59031151e-01
  6.90493935e-01 4.84878885e-01 8.59433826e-01 9.49788903e-01
  8.09493863e-01 2.07164155e-01]
 [9.55915388e-01 9.48989498e-02 5.41988398e-02 6.40923801e-01
  7.43755469e-01 2.50017893e-01 3.31989700e-02 1.34074748e-01
  2.28207952e-01 3.25908172e-02]
 [2.12569942e-01 6.69677338e-01 9.09923093e-01 1.01991643e-01
  2.10794570e-01 9.16393432e-01 5.55584146e-01 1.18959793e-01
  7.56321012e-01 3.62472038e-01]
 [3.42281357e-02 7.12216480e-02 6.95486418e-01 9.78980546e-01
  3.00647015e-01 3.26246658e-01 2.11752356e-01 5.72275792e-01
  4.24713495e-01 9.11965466e-01]
 [9.74393158e-01 5.73210790e-01 3.66722668e-01 1.87183576e-01
  7.54785372e-01 8.92412288e-01 3.46100878e-02 3.50204690e-01
  4.65290049e-01 8.09257041e-01]
 [2.11635956e-02 3.19102609e-01 9.02390283e-02 6.62186375e-01
  6.34803849e-01 5.20057994e-01 1.92220435e-01 6.27785719e-01
  4.29414711e-01 5.87056793e-01]
 [7.05862735e-01 6.18779662e-01 3.69930856e-01 3.24553333e-01
  1.68120436e-01 2.50046570e-02 2.39481072e-01 3.35832622e-03
  9.76893191e-01 2.86534335e-01]
 [3.23354290e-01 2.94933997e-01 2.84407552e-01 2.41035569e-01
  5.14765195e-01 5.47496186e-01 5.98566423e-04 5.58355788e-01
  5.25534583e-01 5.52813941e-01]
 [4.30870867e-01 7.07691760e-01 6.75302367e-01 7.41018835e-01
  6.73512068e-01 5.22597891e-02 4.54712481e-01 3.89290001e-01
  2.25922673e-01 4.20197951e-01]]
0.0005985664226032528 0.9789805462186878
```

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

(**hint**: mean)

In [19]:

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

```
0.5485189949491475
```

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

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

In [22]:

```
X = np.ones((5,5))
X[1:-1,1:-1]=0
print(X)
```

```
[[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.]]
```

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

(**hint**: np.pad)

In [156]:

```
Z = np.ones((5,5))
Z = np.pad(Z, pad_width=1, mode='constant', constant_values=0)
print(Z)
```

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

17. What is the result of the following expression? (★☆☆)

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

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

In [23]:

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

Out[23]:

```
False
```

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

(**hint**: np.diag)

In [24]:

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

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

19. Create a 8x8 matrix and fill it with a checkerboard pattern (★☆☆)

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

In [25]:

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

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

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

(**hint**: np.unravel_index)

In [26]:

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

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

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

(**hint**: np.tile)

In [27]:

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

```
[[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]]
```

22. Normalize a 5x5 random matrix (★☆☆)

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

In [28]:

```
x = np.random.random((5,5))
xmax, xmin = x.max(), x.min()
x= (x-xmin)/(xmax-xmin)
print (x)
```

```
[[0.3650393  0.97584628 0.30174874 0.43023588 0.        ]
 [0.95212917 0.00640874 0.75399831 0.85236766 0.74179728]
 [0.84608112 0.71387    0.32846602 1.         0.05677377]
 [0.82804606 0.21191044 0.20526449 0.14114928 0.82541882]
 [0.5387619  0.82276772 0.84136375 0.36692667 0.75727189]]
```

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

(**hint**: np.dtype)

In [142]:

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

```
<ipython-input-142-b9f1a6f5df9c>:1: FutureWarning: Passing (type, 1) or '1
type' as a synonym of type is deprecated; in a future version of numpy, it
will be understood as (type, (1,)) / '(1,)type'.
  color = np.dtype([("r", np.ubyte, 1),
```

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

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

In [29]:

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

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

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

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

In [141]:

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

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

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

(**hint**: np.sum)

```
# Author: Jake VanderPlas

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

In [35]:

```
#Answer
#10
#10

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

```
10
10
```

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

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

In [38]:

```
#Answer

#Z**Z

#Z/1/1
```

28. What are the result of the following expressions?

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

In [40]:

```
#Answer
#array([-2.14748365e+09])

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

```
<ipython-input-40-19708a4237f9>:4: RuntimeWarning: invalid value encounter
ed in true_divide
  np.array(0) / np.array(0)
<ipython-input-40-19708a4237f9>:5: RuntimeWarning: divide by zero encounte
red in floor_divide
  np.array(0) // np.array(0)
```

Out[40]:

```
array([-2.14748365e+09])
```

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

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

In [133]:

```
def round_array(x,y):
    return np.round(x,y)
test = np.array([32.11, 51.5, 0.112])
print(round_array(test,0))
```

```
[32. 52.  0.]
```

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

(**hint**: np.intersect1d)

In [134]:

```
x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 2, 4])
print(np.intersect1d(x, y))
```

```
[0 2 4]
```

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

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

In [135]:

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

```
D:\My Files\New folder\lib\site-packages\numpy\lib\nanfunctions.py:1115: R
untimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(a, func=_nanmedian, axis=axis, out=out,
```

Out[135]:

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

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

(**hint**: imaginary number)

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

In [132]:

```
#answer
#False

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

```
<ipython-input-132-d32699013579>:4: RuntimeWarning: invalid value encounte
red in sqrt
  np.sqrt(-1) == np.emath.sqrt(-1)
```

Out[132]:

```
False
```

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

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

In [136]:

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

```
2021-09-28
2021-09-27
2021-09-29
```

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

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

In [137]:

```
import numpy as np
print("July, 2016")
print(np.arange('2016-07', '2016-08', dtype='datetime64[D]'))
```

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

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

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

In [138]:

```
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[138]:

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

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

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

In [139]:

```
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))
```

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

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

(**hint**: np.arange)

In [129]:

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

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

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

(**hint**: np.fromiter)

In [128]:

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

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

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

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

(**hint**: np.linspace)

In [127]:

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

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

40. Create a random vector of size 10 and sort it (★★☆)

(**hint**: sort)

In [126]:

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

```
[0.15018179 0.32993291 0.35472573 0.39952795 0.44327432 0.49563011
 0.72603693 0.87051315 0.89002986 0.98835156]
```

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

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

In [143]:

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

Out[143]:

```
45
```

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

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

In [125]:

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

```
False
```

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

(**hint**: flags.writeable)

In [124]:

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

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-124-50d3306bb5ff> in <module>
      1 x = np.zeros(10)
      2 x.flags.writeable = False
----> 3 x[0] = 1

ValueError: assignment destination is read-only
```

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

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

In [122]:

```
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)
```

```
[0.65534316 1.19932662 0.43266388 0.54898428 0.70493748 1.10380939
 0.95577895 1.02432449 0.84425143 0.93499249]
[0.33195148 0.89017287 0.22223046 0.45588732 0.53525642 0.81584485
 1.23718404 0.55593121 0.91427359 1.320212  ]
```

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

(**hint**: argmax)

In [121]:

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

```
[0.         0.49538533 0.35624369 0.27643154 0.19719784 0.27060137
 0.21621554 0.22681718 0.47682576 0.83857719]
```

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

(**hint**: np.meshgrid)

In [120]:

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

```
[[(0.        , 0.        ) (0.11111111, 0.        )
  (0.22222222, 0.        ) (0.33333333, 0.        )
  (0.44444444, 0.        ) (0.55555556, 0.        )
  (0.66666667, 0.        ) (0.77777778, 0.        )
  (0.88888889, 0.        ) (1.        , 0.        )]
 [(0.        , 0.11111111) (0.11111111, 0.11111111)
  (0.22222222, 0.11111111) (0.33333333, 0.11111111)
  (0.44444444, 0.11111111) (0.55555556, 0.11111111)
  (0.66666667, 0.11111111) (0.77777778, 0.11111111)
  (0.88888889, 0.11111111) (1.        , 0.11111111)]
 [(0.        , 0.22222222) (0.11111111, 0.22222222)
  (0.22222222, 0.22222222) (0.33333333, 0.22222222)
  (0.44444444, 0.22222222) (0.55555556, 0.22222222)
  (0.66666667, 0.22222222) (0.77777778, 0.22222222)
  (0.88888889, 0.22222222) (1.        , 0.22222222)]
 [(0.        , 0.33333333) (0.11111111, 0.33333333)
  (0.22222222, 0.33333333) (0.33333333, 0.33333333)
  (0.44444444, 0.33333333) (0.55555556, 0.33333333)
  (0.66666667, 0.33333333) (0.77777778, 0.33333333)
  (0.88888889, 0.33333333) (1.        , 0.33333333)]
 [(0.        , 0.44444444) (0.11111111, 0.44444444)
  (0.22222222, 0.44444444) (0.33333333, 0.44444444)
  (0.44444444, 0.44444444) (0.55555556, 0.44444444)
  (0.66666667, 0.44444444) (0.77777778, 0.44444444)
  (0.88888889, 0.44444444) (1.        , 0.44444444)]
 [(0.        , 0.55555556) (0.11111111, 0.55555556)
  (0.22222222, 0.55555556) (0.33333333, 0.55555556)
  (0.44444444, 0.55555556) (0.55555556, 0.55555556)
  (0.66666667, 0.55555556) (0.77777778, 0.55555556)
  (0.88888889, 0.55555556) (1.        , 0.55555556)]
 [(0.        , 0.66666667) (0.11111111, 0.66666667)
  (0.22222222, 0.66666667) (0.33333333, 0.66666667)
  (0.44444444, 0.66666667) (0.55555556, 0.66666667)
  (0.66666667, 0.66666667) (0.77777778, 0.66666667)
  (0.88888889, 0.66666667) (1.        , 0.66666667)]
 [(0.        , 0.77777778) (0.11111111, 0.77777778)
  (0.22222222, 0.77777778) (0.33333333, 0.77777778)
  (0.44444444, 0.77777778) (0.55555556, 0.77777778)
  (0.66666667, 0.77777778) (0.77777778, 0.77777778)
  (0.88888889, 0.77777778) (1.        , 0.77777778)]
 [(0.        , 0.88888889) (0.11111111, 0.88888889)
  (0.22222222, 0.88888889) (0.33333333, 0.88888889)
  (0.44444444, 0.88888889) (0.55555556, 0.88888889)
  (0.66666667, 0.88888889) (0.77777778, 0.88888889)
  (0.88888889, 0.88888889) (1.        , 0.88888889)]
 [(0.        , 1.        ) (0.11111111, 1.        )
  (0.22222222, 1.        ) (0.33333333, 1.        )
  (0.44444444, 1.        ) (0.55555556, 1.        )
  (0.66666667, 1.        ) (0.77777778, 1.        )
  (0.88888889, 1.        ) (1.        , 1.        )]]
```

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

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

In [144]:

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

```
3638.1636371179666
```

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

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

In [119]:

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

```
-128
127
-2147483648
2147483647
-9223372036854775808
9223372036854775807
-3.4028235e+38
3.4028235e+38
1.1920929e-07
-1.7976931348623157e+308
1.7976931348623157e+308
2.220446049250313e-16
```

49. How to print all the values of an array? (★★☆)

(**hint**: np.set_printoptions)

In [118]:

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

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

(**hint**: argmin)

In [115]:

```
Z = np.arange(50)
v = np.random.uniform(0,50)
index = (np.abs(Z-v)).argmin()
print(Z[index])
```

```
10
```

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

(**hint**: dtype)

In [113]:

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

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

<ipython-input-113-3a6f5eca1821>:1: FutureWarning: Passing (type, 1) or '1
type' as a synonym of type is deprecated; in a future version of numpy, it
will be understood as (type, (1,)) / '(1,)type'.
  Z = np.zeros(10, [ ('position', [ ('x', float, 1),
```

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

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

In [112]:

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

# Much faster with scipy
import scipy
# Thanks Gavin Heverly-Coulson (#issue 1)
import scipy.spatial

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

```
[[0.         0.207225   0.25739715 0.08881192 0.63724801 0.18028463
  0.17033947 0.69189899 0.664974   0.22052447]
 [0.207225   0.         0.46383117 0.29460935 0.72225176 0.25323236
  0.2807207  0.84247138 0.81739437 0.33295503]
 [0.25739715 0.46383117 0.         0.16922212 0.64784389 0.35137343
  0.27676746 0.58689634 0.55918046 0.32263352]
 [0.08881192 0.29460935 0.16922212 0.         0.63774784 0.2277411
  0.16621499 0.65390273 0.6262906  0.23788834]
 [0.63724801 0.72225176 0.64784389 0.63774784 0.         0.47936582
  0.79815818 0.29291873 0.29028435 0.41745583]
 [0.18028463 0.25323236 0.35137343 0.2277411  0.47936582 0.
  0.35049197 0.59150234 0.56709581 0.08275296]
 [0.17033947 0.2807207  0.27676746 0.16621499 0.79815818 0.35049197
  0.         0.81843215 0.7907023  0.38626344]
 [0.69189899 0.84247138 0.58689634 0.65390273 0.29291873 0.59150234
  0.81843215 0.         0.02789748 0.50986399]
 [0.664974   0.81739437 0.55918046 0.6262906  0.29028435 0.56709581
  0.7907023  0.02789748 0.         0.48514867]
 [0.22052447 0.33295503 0.32263352 0.23788834 0.41745583 0.08275296
  0.38626344 0.50986399 0.48514867 0.        ]]
[[0.         0.44806615 0.76228228 0.76521659 0.45099236 0.75733752
  0.26076775 0.47696713 0.69371369 0.53424019]
 [0.44806615 0.         0.37118263 0.40581861 0.62768691 0.58919943
  0.24461499 0.76379576 0.88720243 0.83164177]
 [0.76228228 0.37118263 0.         0.07342581 0.73424052 0.4128657
  0.61097758 0.91191839 0.93998768 0.97513275]
 [0.76521659 0.40581861 0.07342581 0.         0.69420226 0.34286463
  0.63684611 0.87524472 0.88757788 0.93662739]
 [0.45099236 0.62768691 0.73424052 0.69420226 0.         0.48526755
  0.60945107 0.18455884 0.2660493  0.24268899]
 [0.75733752 0.58919943 0.4128657  0.34286463 0.48526755 0.
  0.74617481 0.66555912 0.6015735  0.71385211]
 [0.26076775 0.24461499 0.61097758 0.63684611 0.60945107 0.74617481
  0.         0.69285178 0.87388151 0.75699854]
 [0.47696713 0.76379576 0.91191839 0.87524472 0.18455884 0.66555912
  0.69285178 0.         0.24569425 0.06786548]
 [0.69371369 0.88720243 0.93998768 0.88757788 0.2660493  0.6015735
  0.87388151 0.24569425 0.         0.23023069]
 [0.53424019 0.83164177 0.97513275 0.93662739 0.24268899 0.71385211
  0.75699854 0.06786548 0.23023069 0.        ]]
```

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

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

In [108]:

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

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

(**hint**: np.genfromtxt)

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

In [111]:

```
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
```

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

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

In [145]:

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

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

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

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

In [107]:

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

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

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

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

In [106]:

```
n = 10
p = 4
Z = np.zeros((n,n))
np.put(Z, np.random.choice(range(n*n), p, replace=False),1)
print (Z)
```

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

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

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

In [105]:

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

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

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

Out[105]:

```
array([[-0.04065913, -0.30656162,  0.13340324,  0.12854072, -0.11012729,
        -0.05058212, -0.2521643 , -0.27424341,  0.45418725,  0.31820665],
       [ 0.02758721, -0.5275824 ,  0.43396811,  0.10582152, -0.09069808,
         0.21812103, -0.00178747,  0.20309995, -0.44973282,  0.08120293],
       [-0.17925721, -0.26393512, -0.05984189,  0.36460277,  0.07002368,
         0.45042385,  0.16375638, -0.22791027, -0.08268237, -0.23517981],
       [ 0.45495423, -0.01700992, -0.27308027,  0.47249977, -0.02184339,
        -0.20665364, -0.14107107, -0.06756925,  0.15323875, -0.35346521],
       [-0.2297741 ,  0.17951822,  0.34346736,  0.36336966,  0.04028272,
        -0.04604379, -0.10118861, -0.37839413, -0.15584056, -0.01539677]])
```

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

(**hint**: argsort)

In [104]:

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

```
[[4 6 6]
 [8 8 4]
 [1 2 2]]
[[1 2 2]
 [4 6 6]
 [8 8 4]]
```

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

(**hint**: any, ~)

In [103]:

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

```
False
```

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

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

In [102]:

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

```
0.42878572897431544
```

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

(**hint**: np.nditer)

In [146]:

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

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

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

(**hint**: class method)

In [147]:

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

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

```
range_10
```

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

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

In [101]:

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

```
[6. 3. 1. 2. 3. 2. 5. 2. 3. 3.]
```

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

(**hint**: np.bincount)

In [98]:

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

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

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

(**hint**: np.unique)

In [97]:

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

```
[0 1]
```

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

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

In [96]:

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

```
[[53 57 57 57]
 [65 63 61 53]
 [47 41 52 55]]
```

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

(**hint**: np.bincount)

In [95]:

```
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)
```

```
[0.45629746 0.53474777 0.51256943 0.41619633 0.71834503 0.44281949
 0.45266156 0.59348421 0.81771251 0.49416373]
```

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

(**hint**: np.diag)

In [94]:

```
A = np.random.randint(0,10,(3,3))
B= np.random.randint(0,10,(3,3))
#Slow version

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

# Fast version
np.sum(A * B.T, axis=1)

# Faster version
np.einsum("ij,ji->i", A, B)
```

Out[94]:

```
array([111,  83,  39])
```

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

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

In [93]:

```
Z = np.array([1,2,3,4,5])

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

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

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

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

In [91]:

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

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

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

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

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

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

72. How to swap two rows of an array? (★★★)

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

In [90]:

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

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

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

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

In [89]:

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

```
[( 4, 15) ( 4, 42) ( 8, 50) ( 8, 55) (15, 42) (22, 33) (22, 34) (22, 71)
 (26, 37) (26, 43) (27, 84) (27, 86) (33, 34) (33, 48) (33, 76) (34, 71)
 (37, 43) (39, 75) (39, 88) (48, 76) (50, 55) (51, 68) (51, 73) (52, 70)
 (52, 72) (68, 73) (70, 72) (75, 88) (84, 86)]
```

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

(**hint**: np.repeat)

In [88]:

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

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

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

(**hint**: np.cumsum)

In [87]:

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

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

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

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

In [86]:

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

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

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

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

In [85]:

```
x = np.random.randint(0,2,100)
print ('original: ')
print (x)
print('Negating a boolean: ')
print(np.logical_not(x, out=x))


x = np.random.uniform(-1.0,1.0,10)
print ('original: ')
print (x)
print ('Change the sign of float inplace: ')
print(np.negative(x, out=x))
```

```
original:
[1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 1 1 1 1
 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1
 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 1 1]
Negating a boolean:
[0 0 1 0 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0
 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0
 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 0 0]
original:
[-0.15877833 -0.9665805   0.80134251  0.57005981  0.93512942  0.10152758
 -0.91440921 -0.71247572  0.02428707 -0.56772315]
Change the sign of float inplace:
[ 0.15877833  0.9665805  -0.80134251 -0.57005981 -0.93512942 -0.10152758
  0.91440921  0.71247572 -0.02428707  0.56772315]
```

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

In [83]:

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

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

```
[ 2.35307949 10.00319962  0.38423944  9.542685    2.94037485  8.45954413
  5.5334626   2.80895919  3.55417907  5.57570103]
```

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

In [148]:

```
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]))
```

```
[[10.00017341  0.87885601  7.31978508 17.11004526  4.38012776  7.66994164
  14.63483422  4.5985444  13.5788293   8.88502782]
 [10.94553507 12.59676764  3.85407327  7.2697319  12.36660983  2.36190175
   4.41709681 16.07618238  3.44531536  5.14443951]
 [ 0.7998529  23.08187549 15.07605365  4.9705236   8.64856983 14.49865431
   7.60814129 18.28558043  8.63186829  7.20961103]
 [17.28233442 12.97147754  3.50927852  8.92606052 18.85552942  0.8694154
   5.74421789 21.06815165  4.84817444 10.19270994]
 [ 0.98713164 15.99402864  8.45257312  0.60183968  2.61667554  8.7800504
   1.74637764 10.45192641  2.83512352  5.94278795]
 [ 3.62868196 10.06923954  3.04373511  4.90110221  3.55993671  4.32983955
   2.8525162   3.06197159  1.69612637  5.8243951 ]
 [ 1.50791943  8.70504133  1.4945193   6.74391118  2.28473628  2.52052209
   4.62708353  3.36172613  3.4867265   3.56088111]
 [ 4.69614469 15.9441673   7.77186265  2.42116813  8.20412538  7.10213759
   0.21144041 14.57388651  1.23467252  1.26644364]
 [ 9.82552004 14.24439182  5.56468874  5.49452941 12.25113839  4.13270066
   2.65183232 16.77117489  1.67738134  3.5917704 ]
 [11.74645432  1.00288871  7.39480968 17.54852607  6.18083     8.06297357
  14.98200801  5.99333624 13.94700468 10.26821515]]
```

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

(**hint**: minimum, maximum)

In [81]:

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

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

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

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

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

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

```
<ipython-input-81-f2b5781f38bf>:23: FutureWarning: Using a non-tuple seque
nce for multidimensional indexing is deprecated; use `arr[tuple(seq)]` ins
tead of `arr[seq]`. In the future this will be interpreted as an array ind
ex, `arr[np.array(seq)]`, which will result either in an error or a differ
ent result.
  R[r] = Z[z]
```

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

(**hint**: stride_tricks.as_strided)

In [80]:

```
Z = np.arange(1,15,dtype=int)

def rolling(a, window):
    shape = (a.size - window + 1, window)
    strides = (a.itemsize, a.itemsize)
    return np.lib.stride_tricks.as_strided(a, shape=shape, strides=strides)
R = rolling(Z, 4)
print ('original: ')
print (Z)
print ('after strides: ')
print(R)
```

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

82. Compute a matrix rank (★★★)

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

In [78]:

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

```
10
```

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

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

In [79]:

```
x = np.random.randint(0,10,50)
print (x)
print('rank:', np.bincount(x).argmax())
```

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

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

(**hint**: stride_tricks.as_strided)

In [75]:

```
Z = np.random.randint(0,5,(6,6))
n = 3
i = 1 + (Z.shape[0]-3)
j = 1 + (Z.shape[1]-3)
C = np.lib.stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.strides + Z.stride
s)
print(C)
```

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

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

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

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


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

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

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

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


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

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

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

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


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

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

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

```
    [2 0 2]]

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

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

(**hint**: class method)

In [153]:

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

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

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

```
  File "<ipython-input-153-be9388d23be4>", line 2
    def __setitem__(self,(i,j),value):
                         ^
SyntaxError: invalid syntax
```

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

(**hint**: np.tensordot)

In [66]:

```
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)

# It works, because:
# M is (p,n,n)
# V is (p,n,1)
# Thus, summing over the paired axes 0 and 0 (of M and V independently),
# and 2 and 1, to remain with a (n,1) vector.
```

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

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

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

In [65]:

```
x = np.ones((16,16))
k = 4
S = np.add.reduceat(np.add.reduceat(x, np.arange(0, x.shape[0], k), axis=0),
                                       np.arange(0, x.shape[1], k), axis=1)
print ('input array')
print (x)
print ('block sum')
print (S)
```

```
input array
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
block sum
[[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? (★★★)

In [64]:

```
def iterate(x):
    # Count neighbours
    N = (x[0:-2,0:-2] + x[0:-2,1:-1] + x[0:-2,2:] +
         x[1:-1,0:-2]                + x[1:-1,2:] +
         x[2:  ,0:-2] + x[2:  ,1:-1] + x[2:  ,2:])

    # Apply rules
    birth = (N==3) & (x[1:-1,1:-1]==0)
    survive = ((N==2) | (N==3)) & (x[1:-1,1:-1]==1)
    x[...] = 0
    x[1:-1,1:-1][birth | survive] = 1
    return x

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

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

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

In [63]:

```
x= np.arange(10000)
np.random.shuffle(x)
n = 5

# Slow
print (x[np.argsort(x)[-n:]])

# Fast
print (x[np.argpartition(-x,n)[:n]])
```

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

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

(**hint**: np.indices)

In [62]:

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

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

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

    return ix

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

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

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

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

In [61]:

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

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

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

In [60]:

```
z = np.random.rand(5^7)

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

```
1.34 µs ± 3.4 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
954 ns ± 2.9 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
2.8 µs ± 6.77 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

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

(**hint**: np.where)

In [47]:

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

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

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

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

In [46]:

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

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

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

(**hint**: np.unpackbits)

In [45]:

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

# Author: Daniel T. McDonald

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

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

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

(**hint**: np.ascontiguousarray)

In [44]:

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

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

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

(**hint**: np.einsum)

In [42]:

```
# Make sure to read: http://ajcr.net/Basic-guide-to-einsum/

A= np.arange(3)
B =  np.arange(12).reshape(3,4)
print (A)
#np.einsum('ii->', A)       # np.sum(A)
#np.einsum('i,i->i', A, B) # A * B
#np.einsum('i,i', A, B)    # np.inner(A, B)
#np.einsum('...i,j', A, B)    # np.outer(A, B)
```

```
[0 1 2]
```

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

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

In [41]:

```
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)
```

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

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

In [154]:

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

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

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

(**hint**: np.percentile)

In [155]:

```
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)
```

```
[-0.25089755  0.12213902]
```

In [ ]:

```

```