Numpy

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

(**hint**: import … as …)

```
import numpy as np
```

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

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

```
import numpy
print(numpy.__version__)
print(numpy.show_config)
```

```
1.21.6
<function show at 0x7f9d7505b5e0>
```

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

(**hint**: np.zeros)

```
import numpy as np
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)

```
import numpy as np
x = np.array([100,20,34])
print("size of the array: ",x.size)
print("memory size of one array: ",x.itemsize)
```

```
size of the array:  3
memory size of one array:  8
```

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

(**hint**: np.info)

```
import numpy as np
print(np.info(np.add))
```

Show hidden output

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

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

```
x = np.zeros(10)
x[5]=1
print(x)
```

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

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

(**hint**: np.arange)

```
x = np.arange(10,49)
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]
```

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

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

```
x= np.arange(1,5)
x = x[::-1]
print(x)
```

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

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

(**hint**: reshape)

```
x = np.arange(2,11).reshape(3,3)
print(x)
```

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

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

(**hint**: np.nonzero)

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

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

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

(**hint**: np.eye)

```
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)

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

Show hidden output

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

(**hint**: min, max)

```
x=np.random.random((10,10))
print(x)
xmin, xmax = x.min(), x.max()
print(xmin,xmax)
```

```
[[0.22812774 0.98114319 0.61608549 0.77164288 0.60468925 0.25291732
  0.24025992 0.82710107 0.58147559 0.07335923]
 [0.19359053 0.70007579 0.66460191 0.19706742 0.34317614 0.51563843
  0.52009296 0.73428571 0.19297962 0.09326177]
 [0.15192322 0.52173882 0.64607426 0.28050129 0.14073757 0.92249426
  0.14921805 0.5718952  0.76021679 0.11497576]
 [0.50071634 0.35674354 0.94694021 0.10941063 0.98915121 0.91897778
  0.71426698 0.33977374 0.52402953 0.03836708]
 [0.86027502 0.30683549 0.27019092 0.24352192 0.29483536 0.86399323
  0.07221691 0.30723846 0.28319193 0.57685459]
 [0.86678527 0.72523108 0.20636759 0.07469122 0.2379274  0.01958532
  0.21575142 0.91844935 0.22411331 0.15048557]
 [0.6870546  0.58959133 0.46947814 0.91286846 0.61614965 0.93292121
  0.89511872 0.70594516 0.56102915 0.22246591]
 [0.64144898 0.19875749 0.35856469 0.9369513  0.11311041 0.71516394
  0.17319295 0.19271646 0.64971038 0.48745143]
 [0.33063264 0.39723588 0.7648105  0.11507874 0.82213893 0.03088771
  0.38398321 0.69191369 0.49914635 0.4581111 ]
 [0.1416837  0.84367263 0.63165019 0.06356484 0.10687561 0.67190598
  0.27222982 0.45594891 0.80140191 0.14738638]]
0.0195853150544798 0.9891512050545748
```

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

(**hint**: mean)

```
x=np.arange(30)
print(x)
xmean = x.mean()
print(xmean)
```

```
[ 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]
14.5
```

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

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

```
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)

```
x=np.ones((5,5))
a=np.pad(x,pad_width=1,mode="constant",constant_values=0)
print(a)
```

```
[[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


#No output occur's.
```

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

(**hint**: np.diag)

```
x=np.diag([1,2,3,4,5])
print(x)
```

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

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

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

```
x = np.ones((3,3))
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)

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

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

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

(**hint**: np.tile)

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

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

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

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

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

```
[[0.77792137 0.65740397 0.62734512 0.7165577  0.46324212]
 [0.70208312 0.7168933  0.42088851 0.77171124 0.36393159]
 [0.3218494  0.88452412 0.70719377 0.52135728 0.31429118]
 [1.         0.7807109  0.84776576 0.63837593 0.49957297]
 [0.10934465 0.         0.40158757 0.84756333 0.07694299]]
```

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

(**hint**: np.dtype)

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

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

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

```
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**: >, <=)

```
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))
```

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
```

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)
```

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

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

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

```
[  4. -10.  -2.  -7.  -6.  10.  -4.   7.  10.  -8.]
```

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

(**hint**: np.intersect1d)

```
a=np.random.randint(0,10,10)
b=np.random.randint(0,10,10)
print(np.intersect1d(a,b))
```

```
[0 4 5 9]
```

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

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

```
defaults=np.seterr(all="ignore")
x=np.ones(1)/0
_=np.seterr(**defaults)
with np.errstate(all="ignore"):
  np.arange(3)/0
```

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

(**hint**: imaginary number)

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

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

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

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

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

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

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

```
['2016-07-01' '2016-07-02' '2016-07-03' '2016-07-04' '2016-07-05'
 '2016-07-06' '2016-07-07' '2016-07-08' '2016-07-09' '2016-07-10'
 '2016-07-11' '2016-07-12' '2016-07-13' '2016-07-14' '2016-07-15'
```

```
       '2016-07-16' '2016-07-17' '2016-07-18' '2016-07-19' '2016-07-20'
       '2016-07-21' '2016-07-22' '2016-07-23' '2016-07-24' '2016-07-25'
       '2016-07-26' '2016-07-27' '2016-07-28' '2016-07-29' '2016-07-30'
       '2016-07-31']
```

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

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

```
A = np.ones(3)*1
B = np.ones(3)*2
np.add(A,B,out=B)
np.divide(A,2,out=A)
np.multiply(A,B,out=A)
```

```
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)

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

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

```
[4. 9. 1. 1. 9. 0. 3. 7. 1. 9.]
[4. 9. 1. 1. 9. 0. 3. 7. 1. 9.]
[4. 9. 1. 1. 9. 0. 3. 7. 1. 9.]
[4 9 1 1 9 0 3 7 1 9]
[4. 9. 1. 1. 9. 0. 3. 7. 1. 9.]
```

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

(**hint**: np.arange)

```
x = np.zeros((5,5))
x = np.arange(5)
print(x)

x = np.tile(np.arange(0,5),(5,1))
print(x)
```

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

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

(**hint**: np.fromiter)

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

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

(**hint**: np.linspace)

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

```
[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)

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

```
[0.18244658 0.22427441 0.30839178 0.54890365 0.55787285 0.73190594
 0.77969942 0.85895455 0.96263935 0.98499326]
```

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

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

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

```
45
```

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

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

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

```
False
False
```

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

(**hint**: flags.writeable)

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-342e1d67599c> in <module>
      1 z = np.zeros(10)
      2 z.flags.writeable = False
----> 3 z[10] = 1

ValueError: assignment destination is read-only
```

SEARCH STACK OVERFLOW

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

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

```
a=np.random.random((10,2))
x,y = a[:,0],a[:,1]
r = np.sqrt(x**2+y**2)
t = np.arctan2(y,x)
print(r)
print(t)
```

```
[0.97061959 0.97723283 0.91328996 1.28592896 0.80189457 1.22063711
 0.95411096 0.42933702 0.6514394  0.86851421]
```

```
[0.41916683 1.52376241 1.37581472 0.72456613 0.34490515 0.80712584
 0.96938205 0.28032067 0.62535049 1.30700201]
```

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

(**hint**: argmax)

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

```
[0.26785751 0.59130115 0.         0.66760022 0.70649301 0.39887152
 0.51196608 0.67856283 0.07467642 0.49701637]
```

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

(**hint**: np.meshgrid)

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

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

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

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

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

```
3638.163637117973
```

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

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

```
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)

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

```
print(Z)
```

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

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

(**hint**: argmin)

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

```
11
```

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

(**hint**: dtype)

```
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-61-6f1aa4d7a2ff>:1: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version o
  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)

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

```
[[0.         0.44302087 0.4172199  0.58758119 0.93457606 0.97530011
  0.53839935 0.79647712 0.20037463 0.80127856]
 [0.44302087 0.         0.2522872  0.44378622 0.51572261 0.54895497
  0.16725628 0.5610029  0.32957636 0.43735385]
 [0.4172199  0.2522872  0.         0.67477085 0.56102271 0.74600974
  0.41933334 0.39009369 0.22264889 0.39614479]
 [0.58758119 0.44378622 0.67477085 0.         0.84339738 0.61573156
  0.31754367 1.00274209 0.62908855 0.84253538]
 [0.93457606 0.51572261 0.56102271 0.84339738 0.         0.42582846
  0.54097785 0.45492621 0.76768363 0.20560229]
 [0.97530011 0.54895497 0.74600974 0.61573156 0.42582846 0.
  0.44050531 0.83000673 0.87807858 0.57382298]
 [0.53839935 0.16725628 0.41933334 0.31754367 0.54097785 0.44050531
  0.         0.69646161 0.47177941 0.52499234]
 [0.79647712 0.5610029  0.39009369 1.00274209 0.45492621 0.83000673
  0.69646161 0.         0.59613821 0.25988122]
 [0.20037463 0.32957636 0.22264889 0.62908855 0.76768363 0.87807858
  0.47177941 0.59613821 0.         0.61627478]
 [0.80127856 0.43735385 0.39614479 0.84253538 0.20560229 0.57382298
  0.52499234 0.25988122 0.61627478 0.        ]]
```

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

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

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

```
[85 92 50 65 26 18 37 20 13 84]
```

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

(**hint**: np.genfromtxt)

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

```
from io import StringIO

s = StringIO('''1, 2, 3, 4, 5
                6,  ,  , 7, 8
                 ,  , 9,10,11
''')
Z = np.genfromtxt(s, delimiter=",", dtype=np.int)
print(Z)
```

```
[[ 1  2  3  4  5]
 [ 6 -1 -1  7  8]
 [-1 -1  9 10 11]]
<ipython-input-65-a40f5bb89038>:7: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, us
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Z = np.genfromtxt(s, delimiter=",", dtype=np.int)
```

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

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

```
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)

```
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)

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

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

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

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

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

```
X = np.random.rand(5, 10)
Y = X - X.mean(axis=1, keepdims=True)
Y = X - X.mean(axis=1).reshape(-1, 1)
print(Y)
```

```
[[-0.43398154  0.35782663 -0.48018163  0.05761502  0.37476506 -0.12385651
  -0.02591016  0.31847269  0.05188987 -0.09663943]
 [ 0.47341632 -0.08933695  0.52856756  0.2357073   0.36289168 -0.3221994
  -0.42465115 -0.12422512 -0.25388004 -0.3862902 ]
 [-0.03868429  0.2166373   0.33657332  0.34915726 -0.1819267   0.12654943
  -0.19714677  0.04249201 -0.34537469 -0.30827687]
 [-0.31766085 -0.01317382  0.13289975 -0.39746447  0.31176527  0.38700121
  -0.36973598  0.04910591  0.33027232 -0.11300935]
 [ 0.13143664 -0.38510988  0.14384424 -0.35005082 -0.02606678  0.5243581
  -0.30988759  0.08913887 -0.18036681  0.36270402]]
```

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

(**hint**: argsort)

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

```
[[4 1 1]
 [9 1 8]
 [2 8 6]]
[[4 1 1]
 [9 1 8]
 [2 8 6]]
```

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

(**hint**: any, ~)

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

Z=np.array([
    [0,1,np.nan],
    [1,2,np.nan],
    [4,5,np.nan]])
```

```
False
```

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

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

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

```
0.50290637354078
```

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

(**hint**: np.nditer)

```
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)

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

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

```
range_10
```

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

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

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

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

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

(**hint**: np.bincount)

```
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)

```
w, h = 256, 256
I = np.random.randint(0, 4, (h, w, 3)).astype(np.ubyte)
colors = np.unique(I.reshape(-1, 3), axis=0)
n = len(colors)
print(n)
```

```
64
```

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

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

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

```
[[51 54 58 43]
 [61 43 53 52]
 [46 68 53 55]]
[[51 54 58 43]
 [61 43 53 52]
 [46 68 53 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)

```
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)
```

```
[0.33608566 0.46562843 0.45418418 0.42257878 0.56275874 0.43937003
 0.50103301 0.46191289 0.43737502 0.77159045]
```

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

(**hint**: np.diag)

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

```
array([1.20289072, 1.20184049, 1.74299179, 1.35492481, 1.13468716])
```

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])

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

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

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

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

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

Show hidden output

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

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

```
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)

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

```
[( 0, 41) ( 0, 56) ( 3, 40) ( 3, 63) ( 3, 84) ( 3, 89) (11, 65) (11, 97)
 (15, 15) (15, 31) (15, 34) (15, 68) (18, 29) (18, 72) (25, 81) (25, 88)
 (29, 72) (31, 34) (35, 66) (35, 93) (40, 89) (41, 56) (63, 84) (65, 97)
 (66, 93) (69, 72) (69, 84) (72, 84) (81, 88)]
```

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

(**hint**: np.repeat)

```
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)

```
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)

```
from numpy.lib import stride_tricks

def rolling(a, window):
    shape = (a.size - window + 1, window)
    strides = (a.strides[0], a.strides[0])
    return stride_tricks.as_strided(a, shape=shape, strides=strides)
Z = rolling(np.arange(10), 3)
print(Z)
```

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

```
[6 7 8]
[7 8 9]]
```

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

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

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

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

```
array([-0.46346931, -0.87374229,  0.41788389, -0.98323977, -0.09866067,
       -0.14750384,  0.3272814 ,  0.07753095,  0.00186959,  0.85745254,
        0.21415912,  0.9984971 , -0.13846509, -0.48439165,  0.07631377,
        0.58374539, -0.79311296, -0.66902814,  0.70726484, -0.94427504,
        0.6325173 ,  0.4303672 , -0.68929779, -0.7670332 ,  0.01866789,
       -0.91851308, -0.34742061, -0.19541206,  0.99031863,  0.02020378,
       -0.1488848 , -0.9695073 ,  0.88938198,  0.51230326, -0.93335834,
        0.30388011, -0.07466544,  0.51547982, -0.44260082, -0.8015584 ,
       -0.88547385, -0.22253157, -0.80223951,  0.25522654, -0.38280295,
        0.33678363,  0.89967739,  0.78198991,  0.27342336,  0.36593487,
        0.98571586, -0.26267182,  0.43278837, -0.84238083,  0.0529646 ,
       -0.59788058, -0.77090168, -0.91353903,  0.45152529,  0.31920047,
        0.5849248 ,  0.624691  , -0.01364287,  0.46993903,  0.7723568 ,
       -0.99585571,  0.3307658 ,  0.24796615,  0.86150826,  0.79229196,
        0.55485545,  0.57613992, -0.80939156, -0.5923763 , -0.49411832,
        0.80409825,  0.72052235, -0.35462374, -0.64936013,  0.49981987,
        0.10266119, -0.99648531,  0.89172753, -0.50039527, -0.83706032,
       -0.76333326, -0.83879873, -0.9110501 , -0.47378409, -0.31882541,
       -0.77572453, -0.50316446, -0.04882832,  0.95662614,  0.09724442,
        0.18324766, -0.86693659,  0.04364497, -0.29950197,  0.17781631])
```

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])? (★★★)

```
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))
```

```
[ 0.72320674  2.01244682  7.36412057  3.21621013  6.58189891  4.78713465
  5.61019575  6.05443652 10.25986966  2.14183158]
```

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])? (★★★)

```
P0 = np.random.uniform(-10, 10, (10,2))
P1 = np.random.uniform(-10,10,(10,2))
p = np.random.uniform(-10, 10, (10,2))
print(np.array([distance(P0,P1,p_i) for p_i in p]))
```

```
[[2.87122594e+00 4.03478501e+00 7.34587351e+00 6.00752042e+00
  5.22331054e+00 3.58898641e+00 8.30215854e+00 1.22107422e+01
  5.96174185e+00 6.49852464e-01]
 [9.81795835e+00 1.85420089e+00 9.90771284e+00 6.20756237e+00
  1.04140328e+00 8.35980711e+00 1.20331431e+01 6.96987611e+00
  1.34530003e+00 1.34906276e+00]
 [1.08133898e+01 8.70095729e+00 3.41322404e+00 2.07187241e+00
  1.20526567e-02 4.53012850e+00 6.79957368e+00 1.70199267e+00
  4.43912878e+00 1.00734602e+01]
 [5.71701750e+00 1.76014854e+01 5.46348617e+00 1.06909957e+01
  6.54911980e+00 3.57246590e+00 1.81985853e+00 5.93459397e+00
  1.16565539e+00 1.83552141e+01]
 [5.60935870e-01 1.57801927e+01 4.06817796e+00 6.84768229e+00
```

```
 1.02730956e+01 4.94500747e+00 1.81766572e+00 1.76108352e+00
 5.24126874e+00 1.36925690e+01]
[8.76256607e+00 7.12622109e+00 4.78317459e+00 4.43574645e-01
 1.24027206e+00 4.52026021e+00 7.48555024e+00 2.29973522e+00
 1.64768442e+00 7.19458065e+00]
[2.01489092e+00 1.74962954e+00 9.47785338e+00 8.94666738e+00
 5.30812120e+00 4.61094523e+00 9.89996448e+00 1.56824629e+01
 7.59265022e+00 2.51291291e+00]
[2.17771817e+00 1.38118944e+01 2.36775330e+00 3.64476472e+00
 1.19816684e+01 5.04303498e+00 1.01005912e+00 6.95604861e+00
 8.93046291e+00 1.00102692e+01]
[8.91051798e-01 2.06380742e+00 9.10039163e+00 8.95909438e+00
 6.31123476e+00 3.86345575e+00 9.32746441e+00 1.65730299e+01
 8.78581433e+00 2.67252466e+00]
[7.56050168e+00 6.89795001e+00 4.92029611e+00 1.11174294e+00
 2.16131008e+00 4.07649808e+00 7.32787279e+00 3.87664461e+00
 2.10135631e-01 6.34406619e+00]]
```

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)

```
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)
```

```
[[1 9 8 4 2 2 4 0 3 0]
 [6 2 9 1 6 4 7 0 1 8]
 [4 6 8 5 4 2 1 4 2 6]
 [1 8 0 7 0 2 5 6 1 5]
 [1 6 6 1 7 6 1 1 9 1]
 [7 3 1 4 8 5 4 7 7 0]
 [2 4 8 6 4 5 8 8 9 0]
 [5 0 1 9 3 5 1 0 1 7]
 [1 8 5 3 1 8 4 9 8 1]
 [0 3 3 5 1 3 2 7 7 7]]
[[0 0 0 0 0]
 [0 1 9 8 4]
 [0 6 2 9 1]
 [0 4 6 8 5]
 [0 1 8 0 7]]
<ipython-input-95-f2b5781f38bf>:23: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tupl
  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)

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

```
print(R)
```

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

82. Compute a matrix rank (★★★)

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

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

```
10
```

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

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

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

```
3
```

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

(**hint**: stride_tricks.as_strided)

```
Z = np.random.randint(0,5,(10,10))
n = 3
i = 1 + (Z.shape[0]-3)
j = 1 + (Z.shape[1]-3)
C = stride_tricks.as_strided(Z, shape=(i, j, n, n), strides=Z.strides + Z.strides)
print(C)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

 [[0 1 0]
```

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

(**hint**: class method)

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

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

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

```
[[ 3  2  2 13  9]
 [ 2  3 14  5 11]
 [ 2 14  8 42 15]
 [13  5 42  6  6]
 [ 9 11 15  6  4]]
```

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)

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

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

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

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

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

```
Z = np.ones((16,16))
k = 4
S = np.add.reduceat(np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0),
                                    np.arange(0, Z.shape[1], k), axis=1)
print(S)
```

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

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

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

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

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

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

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

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

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

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

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

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

(**hint**: np.indices)

```
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)

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

```
[(b'Hello', 2.5, 3) (b'World', 3.6, 2)]
```

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

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

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

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

```
3.75 s ± 17.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
136 ms ± 1.07 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
130 ms ± 15 ms per loop (mean ± std. dev. of 7 runs, 10 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)

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

C = (A[..., np.newaxis, np.newaxis] == B)
rows = np.where(C.any((3,1)).all(1))[0]
print(rows)
```

```
[1 5 6]
```

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

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

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

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

(**hint**: np.unpackbits)

```
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]]
```

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

(**hint**: np.ascontiguousarray)

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

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

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

(**hint**: np.einsum)

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

np.einsum('i->', A)
np.einsum('i,i->i', A, B)
np.einsum('i,i', A, B)
np.einsum('i,j->ij', A, B)
```

```
array([[0.13704571, 0.13088786, 0.13096513, 0.01985631, 0.00315395,
        0.06748103, 0.05194566, 0.03617219, 0.25007248, 0.14390748],
       [0.39083631, 0.37327493, 0.37349532, 0.05662758, 0.00899466,
        0.19244701, 0.1481422 , 0.10315831, 0.71317378, 0.41040518],
       [0.31957015, 0.30521096, 0.30539116, 0.04630195, 0.00735455,
        0.1573557 , 0.12112954, 0.08434814, 0.58313172, 0.33557078],
       [0.26076449, 0.2490476 , 0.24919464, 0.0377817 , 0.0060012 ,
        0.12839991, 0.0988399 , 0.06882683, 0.47582681, 0.27382076],
       [0.11036057, 0.10540176, 0.10546399, 0.01598995, 0.00253983,
        0.05434132, 0.04183096, 0.02912884, 0.2013791 , 0.11588624],
       [0.15903673, 0.15189076, 0.15198044, 0.02304255, 0.00366005,
        0.07830937, 0.06028112, 0.04197655, 0.29020033, 0.16699957],
       [0.40017638, 0.38219533, 0.38242098, 0.05798084, 0.00920961,
        0.19704605, 0.15168245, 0.10562355, 0.73021696, 0.4202129 ],
       [0.49216555, 0.47005116, 0.47032868, 0.07130899, 0.01132664,
        0.24234132, 0.18654993, 0.1299034 , 0.89807306, 0.51680789],
       [0.38911722, 0.37163308, 0.3718525 , 0.0563785 , 0.0089551 ,
        0.19160054, 0.14749059, 0.10270457, 0.71003688, 0.40860001],
       [0.39093809, 0.37337213, 0.37359258, 0.05664232, 0.008997  ,
        0.19249713, 0.14818077, 0.10318517, 0.71335949, 0.41051205]])
```

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

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

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

dr = (np.diff(x)**2 + np.diff(y)**2)**.5
r = np.zeros_like(x)
r[1:] = np.cumsum(dr)
```

```
r_int = np.linspace(0, r.max(), 200)
x_int = np.interp(r_int, r, x)
y_int = np.interp(r_int, r, y)
```

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

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

```
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)

```
X = np.random.randn(100)
N = 1000
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)
```

```
[-0.23490123  0.14573651]
```