1.Write a NumPy program to test whether none of the elements of a given array is zero

In [4]:
```
import numpy as np
x = np.array([1, 2, 3, 4])
print("Original array:")
print(x)
print("Test if none of the elements of the said array is zero:")
print(np.all(x))
```

```
Original array:
[1 2 3 4]
Test if none of the elements of the said array is zero:
True
```

2.Write a NumPy program to test whether any of the elements of a given array is non-zero

In [5]:
```
import numpy as np
x = np.array([1, 0, 0, 0])
print("Original array:")
print(x)
print("Test whether any of the elements of a given array is non-zero:")
print(np.any(x))
```

```
Original array:
[1 0 0 0]
Test whether any of the elements of a given array is non-zero:
True
```

3.Create an identity matrix of dimension 4-by-4

Eg: [[1. 0. 0. 0.] [0. 1. 0. 0.] [0. 0. 1. 0.] [0. 0. 0. 1.]]

In [11]:
```
import numpy as np

a = np.identity(4, dtype = float)
print("Matrix b : \n", a)
```

```
Matrix b :
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

4.Convert a 1-D array to a 3-D array

In [12]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

5.Convert all the elements of a numpy array from float to integer datatype

In [13]:
```
import numpy as np
x= np.array([[12.0, 12.51], [2.34, 7.98], [25.23, 36.50]])
print("Original array elements:")
print(x)
print("Convert float values to integer values:")
print(x.astype(int))
```

```
Original array elements:
[[12.   12.51]
 [ 2.34  7.98]
 [25.23 36.5 ]]
Convert float values to integer values:
[[12 12]
 [ 2  7]
 [25 36]]
```

6.Create a 10x10 array with random values and find the minimum and maximum values

In [14]:
```
import numpy as np
x = np.random.random((10,10))
print("Original Array:")
print(x)
xmin, xmax = x.min(), x.max()
print("Minimum and Maximum Values:")
print(xmin, xmax)
```

```
Original Array:
[[0.89840436 0.3207901  0.69798697 0.72365933 0.11016324 0.13054833
  0.4260801  0.19083476 0.28236242 0.35705328]
 [0.21153451 0.58214978 0.4721046  0.42810693 0.65982734 0.59985433
  0.84253209 0.12682682 0.79353337 0.61654655]
 [0.4194293  0.97771038 0.40670735 0.40250986 0.23064899 0.21847601
  0.57542291 0.63783273 0.7125558  0.62063092]
 [0.72907239 0.18828196 0.01813621 0.18733961 0.84600914 0.84436202
  0.81771386 0.22299419 0.85945531 0.92193476]
 [0.16981628 0.58793221 0.72145079 0.51849559 0.40981051 0.05735861
  0.60782019 0.86030153 0.47133647 0.13236702]
 [0.36774247 0.90996901 0.71205464 0.52732304 0.4644732  0.01033907
  0.60751707 0.37690026 0.0550201  0.72653194]
 [0.18350628 0.42320784 0.8370581  0.56181961 0.55701088 0.68059158
  0.7722471  0.69549116 0.47454866 0.10845881]
 [0.46501458 0.77676481 0.63435243 0.76122628 0.33808934 0.90983682
  0.46541517 0.78719255 0.35675402 0.41523285]
 [0.27776934 0.13995153 0.34024246 0.23799221 0.11328323 0.81459471
  0.81479801 0.34928197 0.60128016 0.86496379]
 [0.70417936 0.03250881 0.84350114 0.83271131 0.66901609 0.77759779
  0.68525301 0.00389607 0.3638073  0.52464075]]
Minimum and Maximum Values:
0.003896065648833824 0.9777103841927484
```

7.Create a random vector of size 30 and find the mean value

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

```
0.5470463555271927
```

8.Create a 2d array with 1 on the border and 0 inside

In [21]:
```
import numpy as np
x = np.ones((3,3))
print("Original array:")
print(x)
print("1 on the border and 0 inside in the array")
x[1:-1,1:-1] = 0
print(x)
```

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

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

In [22]:
```
import numpy as np
x = np.ones((3,3))
print("Original array:")
print(x)
print("0 on the border and 1 inside in the array")
x = np.pad(x, pad_width=1, mode='constant', constant_values=0)
print(x)
```

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

10.What is the result of the following expression?

In [91]:
```
0 * np.nan
np.nan == np.nan
np.inf > np.nan
np.nan - np.nan
np.nan in set([np.nan])
0.3 == 3 * 0.1
```

Out[91]:
```
False
```

11.Create a 5x5 matrix with values 1,2,3,4 just below the diagonal

In [27]:
```
import numpy as np
x = np.diag([1, 2, 3, 4])
print(x)
```

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

12.Create a 8x8 matrix and fill it with a checkerboard pattern

In [31]:
```
import numpy as np
print("Checkerboard pattern:")
x = np.zeros((8,8),dtype=int)
print(x)
```

```
Checkerboard pattern:
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
```

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

In [32]:
```
print (np.unravel_index(100, (6,7,8)))
```

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

14.Create a checkerboard 8x8 matrix using the tile function

In [33]:
```
array= np.array([[0,1], [1,0]])
x = np.tile(array,(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]]
```

15.Normalize a 5x5 random matrix

In [34]:
```
Z = np.random.random((5,5))
Zmax, Zmin = Z.max(), Z.min()
Z= (Z-Zmin)/(Zmax-Zmin)
print (Z)
```

```
[[0.1200722  0.22044861 0.17475178 0.97932559 0.57434985]
 [0.60070123 0.8462512  0.05633871 0.84656094 0.44927172]
 [0.94129232 0.43251022 0.63180424 0.98134361 0.3512962 ]
 [1.         0.05940139 0.16346707 0.21188944 0.40202474]
 [0.33949271 0.82238346 0.1174323  0.21586313 0.        ]]
```

16.Create a custom dtype that describes a color as four unsigned bytes (RGBA)

In [40]:
```
RGBA = np.dtype([('red',np.ubyte),('green',np.ubyte),('blue',np.ubyte),('alpha',np.ubyte)])
color = np.array((1,2,3,4),dtype = RGBA)
print(color['red'])
print(color['green'])
print(color['blue'])
print(color['alpha'])
type(color)
```

```
1
2
3
4
```

Out[40]:
```
numpy.ndarray
```

17.Multiply a 5x3 matrix by a 3x2 matrix (real matrix product)

In [41]:
```
arr_1 = np.random.random((5,3))
arr_2 = np.random.random((3,2))
print(arr_1 @ arr_2)
np.dot(arr_1,arr_2)
```

```
[[0.65389974 0.78259958]
 [0.62554468 1.10121228]
 [0.57190617 0.82407007]
 [0.48134484 0.75597861]
 [0.55298894 0.48796371]]
```

Out[41]:
```
array([[0.65389974, 0.78259958],
       [0.62554468, 1.10121228],
       [0.57190617, 0.82407007],
       [0.48134484, 0.75597861],
       [0.55298894, 0.48796371]])
```

18.Given a 1D array, negate all elements which are between 3 and 8, in place

In [44]:
```
arr = np.arange(12)
print(arr>=3)
arr[(arr>=3) & (arr<=8)]*=(-1)
print(arr)

arr = np.arange(12)
```

```
[False False False  True  True  True  True  True  True  True  True  True]
[ 0  1  2 -3 -4 -5 -6 -7 -8  9 10 11]
```

19.What is the output of the following script?

In [92]:
```
print(sum(range(5),-1))
from numpy import *
print(sum(range(5),-1))
```

```
9
10
```

In [ ]:

In [ ]:

20.Consider an integer vector Z, which of these expressions are legal

In [96]:
```
Z = arange(10)
Z**Z
Z_2 = 2 << Z >> 2
print([bin(2) for z in Z_2])
print([bin(z) for z in 2 << Z_2])
```

```
['0b10', '0b10', '0b10', '0b10', '0b10', '0b10', '0b10', '0b10', '0b10', '0b10']
['0b10', '0b100', '0b1000', '0b100000', '0b1000000000', '0b100000000000000000', '0b0', '0b0', '0b0', '0b0']
```

21.What are the result of the following expressions?

In [97]:
```
np.array([np.nan]).astype(int).astype(float)
```

Out[97]:
```
array([-2.14748365e+09])
```

22.How to round away from zero a float array ?

In [47]:
```
arr = np.random.random((3,3))
print(arr)
arr = np.round(arr,2)
print(arr)
```

```
[[0.34147769 0.74865462 0.41809404]
 [0.47413215 0.04688301 0.08671403]
 [0.46452896 0.67905558 0.85268875]]
[[0.34 0.75 0.42]
 [0.47 0.05 0.09]
 [0.46 0.68 0.85]]
```

23.How to find common values between two arrays?

In [55]:
```
x1 = np.random.randint(0,10,10)
x2 = np.random.randint(0,10,10)
print(np.intersect1d(x1,x2))
```

```
[1 3 5]
```

24.How to get the dates of yesterday, today and tomorrow?

In [61]:
```
import numpy as np
yesterday = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
print("Yestraday: ",yesterday)
today     = np.datetime64('today', 'D')
print("Today: ",today)
tomorrow  = np.datetime64('today', 'D') + np.timedelta64(1, 'D')
print("Tomorrow: ",tomorrow)
```

```
Yestraday:  2021-09-05
Today:  2021-09-06
Tomorrow:  2021-09-07
```

25.How to get all the dates corresponding to the month of July 2016

In [63]:
```
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']
```

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

In [64]:
```
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[64]:
```
array([-1.5, -1.5, -1.5])
```

27.Extract the integer part of a random array of positive numbers using 4 different methods

In [65]:
```
x = np.random.uniform(0,10,10)
print (x - x%1)
print (np.floor(x))
print (np.ceil(x)-1)
print (x.astype(int))
```

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

28.Create a 5x5 matrix with row values ranging from 0 to 4

In [70]:
```
x = np.zeros((5,5))
x += np.arange(5)
print(x)
```

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

29.Consider a generator function that generates 10 integers and use it to build an array

In [68]:
```
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.]
```

30.Create a vector of size 10 with values ranging from 0 to 1, both excluded

In [71]:
```
x = np.linspace(0,1,11,endpoint=False)[1:]
print(x)
```

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

31.Create a random vector of size 10 and sort it

In [72]:
```
x = np.random.random(10)
x.sort()
print(x)
```

```
[0.06391535 0.20346433 0.31711716 0.35724936 0.38012448 0.53123162
 0.69979268 0.78639996 0.83430205 0.96295725]
```

32.How to sum a small array faster than np.sum?

In [73]:
```
x = np.arange(10)
np.add.reduce(x)
```

Out[73]:
```
45
```

33.Consider two random array A and B, check if they are equal

In [78]:
```
A = np.random.randint(0,2,5)
B = np.random.randint(0,2,5)
print(A)
print('*'*20)
print(B)
print('*'*20)
```

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

34.Make an array immutable (read-only)

In [84]:
```
import numpy as np

a = np.zeros(7)
print("Before any change ")
print(a)

a[1] = 2
print("Before after first change ")
print(a)
```

```
Before any change
[0. 0. 0. 0. 0. 0. 0.]
Before after first change
[0. 2. 0. 0. 0. 0. 0.]
```

35.Consider a random 10x2 matrix representing cartesian coordinates, convert them to polar coordinates

In [85]:
```
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)
```

```
[1.17781734 0.79796005 0.97329854 0.80856058 1.25104079 1.07871963
 0.74517015 1.0489093  0.8047464  0.64365768]
[1.011613   1.1040655  0.44824563 1.46400021 0.8977177  0.91297135
 1.53993749 0.41066852 1.33428076 0.3651422 ]
```

36.Create random vector of size 10 and replace the maximum value by 0

In [86]:
```
x = np.random.random(10)
x[x.argmax()] = 0
print(x)
```

```
[0.20565314 0.         0.09434376 0.53497185 0.00942981 0.37400594
 0.89123887 0.31867404 0.37791959 0.88318097]
```

37.Create a structured array with x and y coordinates covering the [0,1]x[0,1] area

In [87]:
```
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.  )]]
```

38 Print the minimum and maximum representable value for each numpy scalar type

In [103…
```
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
```

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

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

In [89]:
```
Y = np.random.randint(0,11,10)
X = np.random.randint(11,22,10)
print(Y.reshape(1,10))
Y = np.tile(Y,(10,1))
print(X)
X  = np.tile(X,(10,1)).T
print(X)
C = 1/(X - Y)
```

```
[[ 2  3  7  9 10  5  3  6  4  8]]
[17 20 11 17 17 13 19 13 15 14]
[[17 17 17 17 17 17 17 17 17 17]
 [20 20 20 20 20 20 20 20 20 20]
 [11 11 11 11 11 11 11 11 11 11]
 [17 17 17 17 17 17 17 17 17 17]
 [17 17 17 17 17 17 17 17 17 17]
 [13 13 13 13 13 13 13 13 13 13]
 [19 19 19 19 19 19 19 19 19 19]
 [13 13 13 13 13 13 13 13 13 13]
 [15 15 15 15 15 15 15 15 15 15]
 [14 14 14 14 14 14 14 14 14 14]]
```

40 Create a structured array representing a position (x,y) and a color (r,g,b)

In [90]:
```
import numpy as np
Z = np.zeros(10, [ ('position', [ ('x', float),
                                  ('y', float)]),
                   ('color',    [ ('r', float),
                                  ('g', float),
                                  ('b', float)])])
print(Z)
```

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

In [ ]: