Array creation routines

Ones and zeros

In [3]:

```
import numpy as np
```

Create a new array of 2*2 integers, without initializing entries.

In [9]:

```
a=np.zeros((2,2))
```

Let X = np.array([1,2,3], [4,5,6], np.int32). Create a new array with the same shape and type as X.

In [22]:

```
X = np.array([[1,2,3], [4,5,6]], np.int32)
X
```

Out[22]:

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

Create a 3-D array with ones on the diagonal and zeros elsewhere.

In [23]:

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

Out[23]:

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

In [ ]:

```

```

Create a new array of 3*2 float numbers, filled with ones.

In [24]:

```
a=np.ones((3,2))
a
```

Out[24]:

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

Let x = np.arange(4, dtype=np.int64). Create an array of ones with the same shape and type as X.

In [29]:

```
x=np.arange(4,dtype=np.int64)
y=np.full_like(x,1)
y
```

Out[29]:

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

Create a new array of 3*2 float numbers, filled with zeros.

In [25]:

```
a=np.zeros((3,2))
a
```

Out[25]:

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

Let x = np.arange(4, dtype=np.int64). Create an array of zeros with the same shape and type as X.

In [26]:

```
x=np.arange(4,dtype=np.int64)
y=np.full_like(x,0)
y
```

Out[26]:

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

Create a new array of 2*5 uints, filled with 6.

In [43]:

```
a=np.full((2,5),6,dtype=np.uint32)
a
```

Out[43]:

```
array([[6, 6, 6, 6, 6],
       [6, 6, 6, 6, 6]], dtype=uint32)
```

In [ ]:

```

```

Let x = np.arange(4, dtype=np.int64). Create an array of 6's with the same shape and type as X.

In [46]:

```
x = np.arange(4, dtype=np.int64)
y=np.full_like(x,6)
y
```

Out[46]:

```
array([6, 6, 6, 6], dtype=int64)
```

From existing data

Create an array of [1, 2, 3].

In [47]:

```
a=np.arange(1,4,1)
a
```

Out[47]:

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

Let x = [1, 2]. Convert it into an array.

In [48]:

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

Out[48]:

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

Let X = np.array([[1, 2], [3, 4]]). Convert it into a matrix.

In [49]:

```
X = np.array([[1, 2], [3, 4]])
y=np.matrix(X)
y
```

Out[49]:

```
matrix([[1, 2],
        [3, 4]])
```

Let x = [1, 2]. Conver it into an array of `float` .

In [53]:

```
x = [1, 2]
y=np.asfarray(x)
y
```

Out[53]:

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

Let x = np.array([30]). Convert it into scalar of its single element, i.e. 30.

In [54]:

```
x = np.array([30])
x[0]
```

Out[54]:

```
30
```

Let x = np.array([1, 2, 3]). Create a array copy of x, which has a different id from x.

In [56]:

```
x = np.array([1, 2, 3])
y=np.copy(x)
print(id(x),x)
print(id(y),y)
```

```
3015254559888 [1 2 3]
3015254473584 [1 2 3]
```

Numerical ranges

Create an array of 2, 4, 6, 8, ..., 100.

In [57]:

```
a=np.arange(2,101,2)
a
```

Out[57]:

```
array([  2,   4,   6,   8,  10,  12,  14,  16,  18,  20,  22,  24,  26,
        28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48,  50,  52,
        54,  56,  58,  60,  62,  64,  66,  68,  70,  72,  74,  76,  78,
        80,  82,  84,  86,  88,  90,  92,  94,  96,  98, 100])
```

Create a 1-D array of 50 evenly spaced elements between 3. and 10., inclusive.

In [67]:

```
a=np.linspace(3,10,50)
a
```

Out[67]:

```
array([ 3.        ,  3.14285714,  3.28571429,  3.42857143,  3.57142857,
        3.71428571,  3.85714286,  4.        ,  4.14285714,  4.28571429,
        4.42857143,  4.57142857,  4.71428571,  4.85714286,  5.        ,
        5.14285714,  5.28571429,  5.42857143,  5.57142857,  5.71428571,
        5.85714286,  6.        ,  6.14285714,  6.28571429,  6.42857143,
        6.57142857,  6.71428571,  6.85714286,  7.        ,  7.14285714,
        7.28571429,  7.42857143,  7.57142857,  7.71428571,  7.85714286,
        8.        ,  8.14285714,  8.28571429,  8.42857143,  8.57142857,
        8.71428571,  8.85714286,  9.        ,  9.14285714,  9.28571429,
        9.42857143,  9.57142857,  9.71428571,  9.85714286, 10.        ])
```

Create a 1-D array of 50 element spaced evenly on a log scale between 3. and 10., exclusive.

In [73]:

```
a=np.logspace(3,9,50)
a
```

Out[73]:

```
array([1.00000000e+03, 1.32571137e+03, 1.75751062e+03, 2.32995181e+03,
       3.08884360e+03, 4.09491506e+03, 5.42867544e+03, 7.19685673e+03,
       9.54095476e+03, 1.26485522e+04, 1.67683294e+04, 2.22299648e+04,
       2.94705170e+04, 3.90693994e+04, 5.17947468e+04, 6.86648845e+04,
       9.10298178e+04, 1.20679264e+05, 1.59985872e+05, 2.12095089e+05,
       2.81176870e+05, 3.72759372e+05, 4.94171336e+05, 6.55128557e+05,
       8.68511374e+05, 1.15139540e+06, 1.52641797e+06, 2.02358965e+06,
       2.68269580e+06, 3.55648031e+06, 4.71486636e+06, 6.25055193e+06,
       8.28642773e+06, 1.09854114e+07, 1.45634848e+07, 1.93069773e+07,
       2.55954792e+07, 3.39322177e+07, 4.49843267e+07, 5.96362332e+07,
       7.90604321e+07, 1.04811313e+08, 1.38949549e+08, 1.84206997e+08,
       2.44205309e+08, 3.23745754e+08, 4.29193426e+08, 5.68986603e+08,
       7.54312006e+08, 1.00000000e+09])
```

Building matrices

Let X = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]). Get the diagonal of X, that is, [0, 5, 10].

In [79]:

```
X = np.array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11]])
y=np.diag(X)
y
```

Out[79]:

```
array([ 0,  5, 10])
```

Create a 2-D array whose diagonal equals [1, 2, 3, 4] and 0's elsewhere.

In [ ]:

```
```

Create an array which looks like below. array([[0., 0., 0., 0., 0.], [1., 0., 0., 0., 0.], [1., 1., 0., 0., 0.]])

In [96]:

```
a=np.tri(3,5,-1)
a
```

Out[96]:

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

Create an array which looks like below. array([[0, 0, 0], [4, 0, 0], [7, 8, 0], [10, 11, 12]])

In [95]:

```
a=np.tril(np.arange(1,13).reshape(4,3),-1)
a
```

Out[95]:

```
array([[ 0,  0,  0],
       [ 4,  0,  0],
       [ 7,  8,  0],
       [10, 11, 12]])
```

Create an array which looks like below. array([[1, 2, 3], [4, 5, 6], [0, 8, 9], [0, 0, 12]])

In [94]:

```
a=np.triu(np.arange(1,13).reshape(4,3),-1)
a
```

Out[94]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 0,  8,  9],
       [ 0,  0, 12]])
```

Array manipulation routines

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Q1. Let x be a ndarray [10, 10, 3] with all elements set to one. Reshape x so that the size of the second dimension equals 150.

In [4]:

```
x = np.ones([10,10,3])
y=x.reshape(2,150)
print(y)
```

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

Q2. Let x be array [[1, 2, 3], [4, 5, 6]]. Convert it to [1 4 2 5 3 6].

In [7]:

```
x= np.array([[1,2,3],[4,5,6]])
y=np.column_stack(x)
np.concatenate(y)
```

Out[7]:

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

Q3. Let x be array [[1, 2, 3], [4, 5, 6]]. Get the 5th element.

In [8]:

```
x[1,1]
```

Out[8]:

```
5
```

Q4. Let x be an arbitrary 3-D array of shape (3, 4, 5). Permute the dimensions of x such that the new shape will be (4,3,5).

In [52]:

```
x=np.zeros(shape=(3,4,5))
y=x.reshape(4,3,5)
y.shape
```

Out[52]:

```
(4, 3, 5)
```

Q5. Let x be an arbitrary 2-D array of shape (3, 4). Permute the dimensions of x such that the new shape will be (4,3).

In [62]:

```
x=np.zeros(shape=(3,4))
y=x.reshape(4,3)
y.shape
```

Out[62]:

```
(4, 3)
```

Q5. Let x be an arbitrary 2-D array of shape (3, 4). Insert a nex axis such that the new shape will be (3, 1, 4).

In [60]:

```
x=np.zeros(shape=(3,4))
y=x.reshape(3,1,4)
y.shape
```

Out[60]:

```
(3, 1, 4)
```

Q6. Let x be an arbitrary 3-D array of shape (3, 4, 1). Remove a single-dimensional entries such that the new shape will be (3, 4).

In [63]:

```
x=np.zeros(shape=(3,4,1))
np.squeeze(x).shape
```

Out[63]:

```
(3, 4)
```

Q7. Lex x be an array
[[1 2 3]
[4 5 6].

and y be an array
[[7 8 9]
[10 11 12]].
Concatenate x and y so that a new array looks like
[[1, 2, 3, 7, 8, 9],
[4, 5, 6, 10, 11, 12]].

In [10]:

```
x= np.array([[ 1, 2, 3],[ 4, 5, 6]])
y = np.array([[ 7, 8, 9],[10, 11, 12]])
np.concatenate((x,y),axis=1)
```

Out[10]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

Q8. Lex x be an array
[[1 2 3]
[4 5 6].

and y be an array
[[7 8 9]
[10 11 12]].
Concatenate x and y so that a new array looks like
[[1 2 3]
[4 5 6]
[7 8 9]
[10 11 12]]

In [11]:

```
np.concatenate((x,y),axis=0)
```

Out[11]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

Q8. Let x be an array [1 2 3] and y be [4 5 6]. Convert it to [[1, 4], [2, 5], [3, 6]].

In [12]:

```
x1 = np.column_stack(np.array([1,2,3]))
y1 = np.column_stack(np.array([4,5,6]))
x1 = np.column_stack(np.concatenate((x1,y1),axis=0))
print(x1)
```

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

Q9. Let x be an array [[1],[2],[3]] and y be [[4], [5], [6]]. Convert x to [[[1, 4]], [[2, 5]], [[3, 6]]].

In [64]:

```
x=np.array([[1],[2],[3]])
y=np.array([[4],[5],[6]])
z=np.concatenate((x,y))
z.reshape(3,2)
```

Out[64]:

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

Q10. Let x be an array [1, 2, 3, ..., 9]. Split x into 3 arrays, each of which has 4, 2, and 3 elements in the original order.

In [18]:

```
a = np.split(np.arange(1,10),[4,6])
print(a)
```

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

Q11. Let x be an array
[[[0., 1., 2., 3.],
[4., 5., 6., 7.]],

[[8., 9., 10., 11.],
[12., 13., 14., 15.]]].
Split it into two such that the first array looks like
[[[0., 1., 2.],
[4., 5., 6.]],

[[8., 9., 10.],
[12., 13., 14.]]].

and the second one look like:

[[[3.],
[7.]],

[[11.],
[15.]]].

In [19]:

```
x = np.arange(16).reshape(2,2,4)
np.dsplit(x,np.array([3,6]))
```

Out[19]:

```
[array([[[ 0,  1,  2],
         [ 4,  5,  6]],

        [[ 8,  9, 10],
         [12, 13, 14]]]),
 array([[[ 3],
         [ 7]],

        [[11],
         [15]]]),
 array([], shape=(2, 2, 0), dtype=int32)]
```

Q12. Let x be an array
[[0., 1., 2., 3.],
[4., 5., 6., 7.],
[8., 9., 10., 11.],
[12., 13., 14., 15.]].
Split it into two arrays along the second axis.

In [20]:

```
x = np.arange(0,16).reshape(4,4)
print(np.split(x,2,axis=1))
```

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

Q13. Let x be an array
[[0., 1., 2., 3.],
[4., 5., 6., 7.],

[8., 9., 10., 11.],
[12., 13., 14., 15.]].
Split it into two arrays along the first axis.

In [21]:

```
x = np.arange(16).reshape((4,4))
print(np.split(x,2))
```

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

Q14. Let x be an array [0, 1, 2]. Convert it to
[[0, 1, 2, 0, 1, 2],
[0, 1, 2, 0, 1, 2]].

In [23]:

```
x = np.array([0,1,2])
c = np.concatenate([x,x])
d=np.stack([c,c])
print(d)
```

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

Q15. Let x be an array [0, 1, 2]. Convert it to
[0, 0, 1, 1, 2, 2].

In [24]:

```
x = np.array([0,1,2])
y=np.concatenate(np.column_stack(np.vstack([x,x])))
print(y)
```

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

Q16. Let x be an array [0, 0, 0, 1, 2, 3, 0, 2, 1, 0].
remove the leading the trailing zeros.

In [25]:

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

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

Q17. Let x be an array [2, 2, 1, 5, 4, 5, 1, 2, 3]. Get two arrays of unique elements and their counts.

In [29]:

```
x = np.array([2,2,1,5,4,5,1,2,3])
np.unique(x,return_counts=True)
```

Out[29]:

```
(array([1, 2, 3, 4, 5]), array([2, 3, 1, 1, 2], dtype=int64))
```

Q18. Lex x be an array
[[1 2]
[3 4].
Flip x along the second axis.

In [30]:

```
x = np.array([[1,2],[3,4]])
print(np.flip(x ,axis=1))
```

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

Q19. Lex x be an array
[[1 2]
[3 4].
Flip x along the first axis.

In [31]:

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

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

Q20. Lex x be an array
[[1 2]
[3 4].
Rotate x 90 degrees counter-clockwise.

In [32]:

```
x = np.array([[1,2],[3,4]])
y=np.rot90(x,1)
print(y)
```

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

Q21 Lex x be an array
[[1 2 3 4]
[5 6 7 8].
Shift elements one step to right along the second axis.

In [45]:

```
a=np.array([[1,2,3,4],[5,6,7,8]])
np.roll(a,1,axis=1)
```

Out[45]:

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

In [ ]:

String operations

In [1]:

```
from __future__ import print_function
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Q1. Concatenate x1 and x2.

In [12]:

```
x1 = np.array(['Hello', 'Say'], dtype=str)
x2 = np.array([' world', ' something'], dtype=str)
x3=np.char.add(x1,x2)
print(x3)
```

```
['Hello world' 'Say something']
```

Q2. Repeat x three time element-wise.

In [20]:

```
x = np.array(['Hello ', 'Say '], dtype=str)
y=np.char.multiply(x,3)
print(y)
```

```
['Hello Hello Hello ' 'Say Say Say ']
```

Q3-1. Capitalize the first letter of x element-wise.
Q3-2. Lowercase x element-wise.
Q3-3. Uppercase x element-wise.
Q3-4. Swapcase x element-wise.
Q3-5. Title-case x element-wise.

In [22]:

```
x = np.array(['heLLo woRLd', 'Say sOmething'], dtype=str)
capitalized = np.char.capitalize(x)
lowered = np.char.lower(x)
uppered = np.char.upper(x)
swapcased = np.char.swapcase(x)
titlecased = np.char.title(x)
print("capitalized =", capitalized)
print("lowered =", lowered)
print("uppered =", uppered)
print("swapcased =", swapcased)
print("titlecased =", titlecased)
```

```
capitalized = ['Hello world' 'Say something']
lowered = ['hello world' 'say something']
uppered = ['HELLO WORLD' 'SAY SOMETHING']
swapcased = ['HEllO WOrlD' 'sAY SoMETHING']
titlecased = ['Hello World' 'Say Something']
```

Q4. Make the length of each element 20 and the string centered / left-justified / right-justified with paddings of _ .

In [23]:

```
x = np.array(['hello world', 'say something'], dtype=str)
centered = np.char.center(x,20,'_')
left = np.char.ljust(x,20,'_')
right = np.char.rjust(x,20,'_')

print("centered =", centered)
print("left =", left)
print("right =", right)
```

```
centered = ['____hello world_____' '___say something____']
left = ['hello world_________' 'say something_______']
right = ['_________hello world' '_______say something']
```

Q5. Encode x in cp500 and decode it again.

In [25]:

```
x = np.array(['hello world', 'say something'], dtype=str)
encoded = np.char.encode(x,'cp500')
decoded = np.char.decode(encoded ,'cp500')
print("encoded =", encoded)
print("decoded =", decoded)
```

```
encoded = [b'\x88\x85\x93\x93\x96@\xa6\x96\x99\x93\x84'
 b'\xa2\x81\xa8@\xa2\x96\x94\x85\xa3\x88\x89\x95\x87']
decoded = ['hello world' 'say something']
```

Q6. Insert a space between characters of x.

In [27]:

```
x = np.array(['hello world', 'say something'], dtype=str)
np.char.join(' ',x)
```

Out[27]:

```
array(['h e l l o   w o r l d', 's a y   s o m e t h i n g'], dtype='<U25')
```

Q7-1. Remove the leading and trailing whitespaces of x element-wise.
Q7-2. Remove the leading whitespaces of x element-wise.
Q7-3. Remove the trailing whitespaces of x element-wise.

In [29]:

```
x = np.array(['    hello world    ', '\tsay something\n'], dtype=str)
stripped = np.char.strip(x)
lstripped = np.char.lstrip(x)
rstripped = np.char.rstrip(x)
print("stripped =", stripped)
print("lstripped =", lstripped)
print("rstripped =", rstripped)
```

```
stripped = ['hello world' 'say something']
lstripped = ['hello world    ' 'say something\n']
rstripped = ['    hello world' '\tsay something']
```

Q8. Split the element of x with spaces.

In [37]:

```
x = np.array(['Hello my name is John'], dtype=str)
sparr = np.char.split(x)
print(sparr)
```

```
[list(['Hello', 'my', 'name', 'is', 'John'])]
```

Q9. Split the element of x to multiple lines.

In [39]:

```
x = np.array(['Hello\nmy name is John'], dtype=str)
np.char.splitlines(x)
```

Out[39]:

```
array([list(['Hello', 'my name is John'])], dtype=object)
```

Q10. Make x a numeric string of 4 digits with zeros on its left.

In [41]:

```
x = np.array(['34'], dtype=str)
print(np.char.ljust(x,4,'0'))
```

```
['3400']
```

Q11. Replace "John" with "Jim" in x.

In [42]:

```
x = np.array(['Hello nmy name is John'], dtype=str)
print("{}".format(np.char.replace(x,'John','jim')))
```

```
['Hello nmy name is jim']
```

Comparison

Q12. Return x1 == x2, element-wise.

In [43]:

```
x1 = np.array(['Hello', 'my', 'name', 'is', 'John'], dtype=str)
x2 = np.array(['Hello', 'my', 'name', 'is', 'Jim'], dtype=str)
np.char.equal(x1,x2)
```

Out[43]:

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

Q13. Return x1 != x2, element-wise.

In [44]:

```
x1 = np.array(['Hello', 'my', 'name', 'is', 'John'], dtype=str)
x2 = np.array(['Hello', 'my', 'name', 'is', 'Jim'], dtype=str)
np.char.not_equal(x1,x2)
```

Out[44]:

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

String information

Q14. Count the number of "l" in x, element-wise.

In [45]:

```
x = np.array(['Hello', 'my', 'name', 'is', 'Lily'], dtype=str)
np.char.count(x,'l')
```

Out[45]:

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

Q15. Count the lowest index of "l" in x, element-wise.

In [46]:

```
x = np.array(['Hello', 'my', 'name', 'is', 'Lily'], dtype=str)
np.char.find(x,'l')
```

Out[46]:

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

Q16-1. Check if each element of x is composed of digits only.
Q16-2. Check if each element of x is composed of lower case letters only.
Q16-3. Check if each element of x is composed of upper case letters only.

In [48]:

```
x = np.array(['Hello', 'I', 'am', '20', 'years', 'old'], dtype=str)
out1 = np.char.isdigit(x)
out2 = np.char.islower(x)
out3 = np.char.isupper(x)
print("Digits only =", out1)
print("Lower cases only =", out2)
print("Upper cases only =", out3)
```

```
Digits only = [False False False  True False False]
Lower cases only = [False False  True False  True  True]
Upper cases only = [False  True False False False False]
```

Q17. Check if each element of x starts with "hi".

In [49]:

```
x = np.array(['he', 'his', 'him', 'his'], dtype=str)
np.char.startswith(x,'hi')
```

Out[49]:

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

In [ ]:

Input and Output

In [1]:

```
from __future__ import print_function
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

In [3]:

```
from datetime import date
print(date.today())
```

```
2023-02-02
```

NumPy binary files (NPY, NPZ)

Q1. Save x into `temp.npy` and load it.

In [4]:

```
x = np.arange(10)
np.save("temp.npy",x)
import os
if os.path.exists('temp.npy'):
    x2 = np.load("temp.npy")
    print(np.array_equal(x, x2))
```

```
True
```

Q2. Save x and y into a single file 'temp.npz' and load it.

In [5]:

```
x = np.arange(10)
y = np.arange(11, 20)
np.savez("temp.npz",x=x,y=y)

with np.load("temp.npz") as data:
    x2 = data['x']
    y2 = data['y']
    print(np.array_equal(x, x2))
    print(np.array_equal(y, y2))
```

```
True
True
```

Text files

Q3. Save x to 'temp.txt' in string format and load it.

In [8]:

```
x = np.arange(10).reshape(2, 5)
header = 'num1 num2 num3 num4 num5'
np.savetxt("temp.txt",x)
np.loadtxt("temp.txt")
```

Out[8]:

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

Q4. Save `x`, `y`, and `z` to 'temp.txt' in string format line by line, then load it.

In [11]:

```
x = np.arange(10)
y = np.arange(11, 21)
z = np.arange(22, 32)
np.savetxt("temp.txt",[x,y,z])
np.loadtxt("temp.txt")
```

Out[11]:

```
array([[ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],
       [11., 12., 13., 14., 15., 16., 17., 18., 19., 20.],
       [22., 23., 24., 25., 26., 27., 28., 29., 30., 31.]])
```

Q5. Convert `x` into bytes, and load it as array.

In [21]:

```
x = np.array([1, 2, 3, 4])
x_bytes = x.tobytes()
x2 = np.frombuffer(x_bytes, dtype=x.dtype)
print(np.array_equal(x, x2))
```

```
True
```

Q6. Convert `a` into an ndarray and then convert it into a list again.

In [15]:

```
a = [[1, 2], [3, 4]]
x = np.array(a)
a2 = x.tolist()
print(a == a2)
```

```
True
```

String formatting¶

Q7. Convert `x` to a string, and revert it.

In [17]:

```
x = np.arange(10).reshape(2,5)
x_str = str(x)
print(x_str, "\n", type(x_str))
x_str = x_str.replace("[", "") # [] must be stripped
x_str = x_str.replace("]", "")
x2 = np.fromstring(x_str, dtype=x.dtype, sep=" ").reshape(x.shape)
assert np.array_equal(x, x2)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
 <class 'str'>
```

Text formatting options

Q8. Print `x` such that all elements are displayed with precision=1, no suppress.

In [18]:

```
x = np.random.uniform(size=[10,100])
np.set_printoptions(precision=1, suppress=True)
print(x)
```

```
[[0.7 0.9 0.5 0.6 0.8 0.2 0.7 0.2 0.2 0.4 0.8 0.3 0.5 0.1 0.  0.4 0.8 0.9
  0.7 0.8 0.5 0.5 0.3 0.4 0.7 0.  0.9 0.4 0.9 0.2 0.4 0.  0.5 0.4 0.8 1.
  0.7 0.2 0.4 0.9 0.5 1.  0.7 0.2 0.3 0.7 0.7 0.8 0.2 0.2 0.7 0.  0.7 0.7
  0.9 0.6 0.8 0.8 0.8 0.8 0.3 0.4 0.  0.3 0.1 0.3 0.3 0.6 0.7 0.7 0.8 0.1
  0.6 0.5 0.6 0.8 0.9 0.8 0.8 0.1 0.6 0.2 1.  0.6 0.7 0.7 0.9 0.4 0.3 0.7
  0.8 0.2 0.1 0.3 0.3 0.5 0.4 0.8 0.5 0.9]
 [0.5 0.6 0.6 0.7 0.4 0.1 0.  0.5 0.  0.4 0.3 0.8 0.7 0.2 0.3 0.3 1.  0.9
  0.6 0.4 0.3 0.7 0.9 0.1 0.9 0.8 0.5 0.2 0.1 0.8 0.1 0.3 0.6 0.4 0.6 0.4
  0.8 0.1 0.7 0.3 1.  0.7 0.1 0.1 0.6 0.7 0.3 0.4 0.2 0.1 1.  0.2 0.  0.5
  0.4 0.9 0.2 0.9 0.3 0.2 0.5 0.5 0.5 0.2 0.6 0.7 0.8 0.6 0.7 0.2 0.1 0.1
  0.1 0.  0.2 0.  0.9 0.8 0.9 0.9 0.1 0.5 0.6 0.5 0.2 0.5 0.9 0.  0.3 0.3
  0.2 0.1 0.3 0.  0.3 0.  0.7 0.4 0.9 0.3]
 [0.8 1.  0.6 0.4 0.1 0.9 0.7 0.8 0.7 0.7 0.7 0.9 0.4 0.  1.  0.8 0.8 0.9
  0.2 0.9 0.8 0.4 0.4 0.6 0.6 0.1 0.6 0.8 0.5 0.5 0.9 0.  1.  1.  0.2 0.7
  0.  0.9 0.1 0.6 0.9 0.5 0.8 0.5 0.8 0.7 0.5 0.4 0.5 0.2 0.9 0.9 0.5 0.1
  0.  0.9 0.4 0.2 0.7 0.3 1.  0.3 0.9 0.9 0.4 0.1 0.  0.9 0.7 0.1 0.9 0.3
  0.3 0.7 0.5 0.5 1.  0.1 0.1 0.9 0.1 0.  0.8 0.7 0.1 0.8 0.3 0.9 0.8 1.
  0.5 0.2 0.  0.7 0.9 0.8 0.7 0.2 0.7 0.8]
 [0.5 0.5 0.9 0.4 1.  0.5 0.8 0.2 0.2 0.7 0.5 0.1 0.6 0.6 0.2 0.2 0.8 0.8
  0.  1.  0.9 0.6 1.  0.2 0.7 0.2 0.3 0.3 0.8 0.2 0.6 0.9 0.5 0.8 0.3 0.5
  0.6 0.6 1.  0.  0.6 0.8 0.1 0.2 1.  0.7 0.7 0.5 1.  0.8 0.9 0.3 0.8 0.7
  0.4 0.3 0.4 0.2 0.5 0.  1.  0.7 0.6 0.2 0.3 0.8 0.8 0.8 0.3 0.8 0.8 0.8
  0.3 0.6 0.5 0.6 0.1 0.6 0.1 0.5 0.4 0.5 0.8 0.9 0.  1.  0.8 0.8 0.6 0.2
  0.7 0.5 0.9 0.3 0.  0.6 0.8 0.8 0.7 0.6]
 [0.4 0.9 0.8 0.9 0.1 0.6 1.  0.5 0.9 0.1 0.1 0.1 0.3 1.  0.2 0.5 0.2 0.9
  0.1 0.1 0.2 0.4 0.2 0.1 1.  0.3 0.7 0.1 0.3 0.6 0.1 0.6 0.7 0.8 0.9 0.5
  0.9 0.6 0.8 0.8 0.5 0.7 1.  0.8 0.6 1.  0.3 0.7 0.7 0.2 0.  0.7 0.1 0.2
  0.6 0.2 0.6 0.6 0.9 0.3 0.8 0.6 0.3 0.2 0.2 0.4 0.  0.2 0.9 0.8 0.5 0.2
  0.8 0.  0.8 0.8 0.  0.8 1.  0.9 0.7 0.7 1.  0.1 0.3 0.4 1.  0.4 0.8 0.7
  0.2 0.2 0.3 0.3 0.4 0.2 0.3 0.2 0.4 0.4]
 [0.3 0.7 0.9 0.3 0.2 0.9 0.9 0.  0.8 0.8 0.6 0.4 0.6 0.5 0.8 0.9 0.3 0.9
  0.3 0.9 0.6 0.5 0.7 0.9 0.4 0.9 0.6 0.8 0.6 0.9 0.4 0.3 0.2 0.3 0.7 0.4
  0.  0.2 0.  0.2 0.2 1.  1.  0.3 0.7 0.4 0.6 0.2 0.1 0.6 0.6 0.7 0.1 0.3
  0.7 0.2 0.9 0.6 0.1 0.9 1.  0.5 0.5 0.2 0.  0.6 1.  0.5 0.1 0.2 0.8 0.
  0.7 0.  0.2 0.5 0.4 0.1 0.1 0.2 0.5 0.  0.2 0.5 0.  0.8 0.3 0.1 0.8 0.9
  0.7 0.9 0.4 0.6 0.  1.  0.2 0.3 0.6 0.4]
 [0.3 0.2 0.3 1.  0.4 0.  0.7 0.4 0.2 0.9 0.  0.5 0.3 0.7 1.  0.7 0.7 0.8
  0.7 0.5 0.8 0.7 0.1 0.8 0.8 0.2 0.1 0.4 0.7 0.4 0.7 0.3 0.6 0.3 0.6 0.7
  0.4 0.3 0.3 0.2 0.2 0.9 0.2 0.3 0.7 0.4 0.9 0.7 0.5 0.7 0.4 0.1 0.5 0.1
  0.  0.3 0.4 0.3 0.1 0.7 0.2 1.  0.6 0.1 0.2 0.1 0.9 0.5 0.5 0.1 0.3 0.1
  0.7 0.4 0.3 0.8 0.1 0.  0.2 0.4 0.  0.1 0.3 1.  0.4 0.7 0.8 0.8 0.6 0.9
  0.7 0.7 0.2 0.5 0.7 0.1 0.9 0.4 0.4 0.4]
 [1.  0.1 0.3 0.2 0.7 0.4 0.8 0.7 1.  0.5 0.3 0.1 0.2 0.5 0.6 0.6 0.4 1.
  0.5 0.6 0.5 0.6 0.6 0.6 0.8 0.7 0.5 0.3 0.9 0.7 0.3 0.5 0.4 0.4 0.9 0.7
  0.  0.7 0.1 0.7 0.4 0.  0.4 0.3 0.3 0.  0.3 0.4 0.1 0.8 1.  0.7 0.4 0.3
  0.5 0.6 0.8 0.5 0.4 0.  0.1 0.3 0.8 0.4 1.  0.3 0.9 1.  0.2 0.2 0.5 0.2
  0.7 0.4 0.2 0.9 0.1 0.6 0.1 0.6 0.4 0.2 0.1 0.6 0.  0.1 0.2 0.4 1.  0.5
  0.6 0.  0.1 0.3 0.5 1.  0.6 1.  0.2 0.7]
 [0.9 0.  0.6 0.2 0.1 0.5 0.8 0.1 0.5 0.6 0.5 0.3 0.2 0.4 0.5 0.9 0.6 0.7
  0.4 0.4 0.6 0.9 0.6 0.3 0.3 0.4 0.5 0.7 0.1 0.5 0.4 0.5 0.1 1.  0.3 0.6
  0.9 0.3 0.8 0.8 0.5 0.  0.2 0.3 1.  0.4 0.8 0.3 0.7 0.6 0.7 0.4 0.6 0.
  0.  0.7 0.9 0.5 0.1 0.9 0.9 0.3 0.8 0.9 0.4 0.3 0.8 0.7 0.7 0.1 0.2 0.6
  0.1 0.9 0.8 0.3 0.1 0.4 0.4 0.5 0.6 0.7 0.4 0.8 0.1 0.  1.  0.7 0.4 0.5
  0.  0.6 0.3 0.8 1.  0.7 0.1 0.3 0.8 0.4]
 [0.2 0.1 0.4 1.  0.9 0.3 0.4 0.3 0.7 0.3 0.2 0.8 0.4 0.8 0.8 0.2 0.9 0.6
  0.3 0.6 0.1 0.4 0.5 0.3 1.  0.2 0.7 0.5 0.3 0.8 0.9 0.7 0.9 0.6 0.9 0.2
  0.3 0.1 0.9 0.7 0.6 0.9 0.  0.2 0.3 0.7 0.8 0.7 0.5 0.2 0.1 0.9 0.5 0.3
  0.3 0.8 0.8 0.1 1.  0.4 0.6 0.5 0.  0.9 0.7 0.6 0.4 0.9 0.5 0.1 0.8 0.1
  0.6 0.4 0.3 0.8 0.4 0.6 0.6 0.1 0.3 0.4 0.4 0.3 0.7 0.6 0.5 0.9 0.4 0.3
  0.8 0.8 0.5 1.  0.9 1.  0.2 0.3 0.  0.2]]
```

Base-n representations

Q9. Convert 12 into a binary number in string format.

In [19]:

```
print(np.binary_repr(12))
```

```
1100
```

Q10. Convert 12 into a hexadecimal number in string format.

In [20]:

```
np.base_repr(1100,base=16)
```

Out[20]:

```
'44C'
```

Linear algebra

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Matrix and vector products

Q1. Predict the results of the following code.

In [4]:

```
x = [1,2]
y = [[4, 1], [2, 2]]
print(np.dot(x, y))
print(np.dot(y, x))
print(np.matmul(x, y))
print(np.inner(x, y))
print(np.inner(y, x))
```

```
[8 5]
[6 6]
[8 5]
[6 6]
[6 6]
```

Q2. Predict the results of the following code.

In [5]:

```
x = [[1, 0], [0, 1]]
y = [[4, 1], [2, 2], [1, 1]]
print(np.dot(y, x))
print(np.matmul(y, x))
```

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

Q3. Predict the results of the following code.

In [6]:

```
x = np.array([[1, 4], [5, 6]])
y = np.array([[4, 1], [2, 2]])
print(np.vdot(x, y))
print(np.vdot(y, x))
print(np.dot(x.flatten(), y.flatten()))
print(np.inner(x.flatten(), y.flatten()))
print((x*y).sum())
```

```
30
30
30
30
30
```

Q4. Predict the results of the following code.

In [7]:

```
x = np.array(['a', 'b'], dtype=object)
y = np.array([1, 2])
print(np.inner(x, y))
print(np.inner(y, x))
print(np.outer(x, y))
print(np.outer(y, x))
```

```
abb
abb
[['a' 'aa']
 ['b' 'bb']]
[['a' 'b']
 ['aa' 'bb']]
```

Decompositions

Q5. Get the lower-trianglular `L` in the Cholesky decomposition of x and verify it.

In [8]:

```
x = np.array([[4, 12, -16], [12, 37, -43], [-16, -43, 98]], dtype=np.int32)
y=np.linalg.cholesky(x)
print(y)
```

```
[[ 2.  0.  0.]
 [ 6.  1.  0.]
 [-8.  5.  3.]]
```

Q6. Compute the qr factorization of x and verify it.

In [9]:

```
x = np.array([[12, -51, 4], [6, 167, -68], [-4, 24, -41]], dtype=np.float32)
q,r = np.linalg.qr(x)
print("q = {},\n\n r = {}".format(q,r))
```

```
q = [[-0.85714287  0.3942857   0.33142856]
 [-0.42857143 -0.9028571  -0.03428571]
 [ 0.2857143  -0.17142858  0.94285715]],

 r = [[ -14.  -21.   14.]
 [   0. -175.   70.]
 [   0.    0.  -35.]]
```

Q7. Factor x by Singular Value Decomposition and verify it.

In [10]:

```
x = np.array([[1, 0, 0, 0, 2], [0, 0, 3, 0, 0], [0, 0, 0, 0, 0], [0, 2, 0, 0, 0]], dtype=np.float32)
u,s,vh = np.linalg.svd(x)
print("u = {}\n s= {}\n vh = {}".format(u,s,vh))
```

```
u = [[ 0.  1.  0.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  0.  0. -1.]
 [ 0.  0.  1.  0.]]
 s= [3.       2.236068 2.       0.      ]
 vh = [[-0.         0.         1.        -0.         0.       ]
 [ 0.4472136 -0.        -0.        -0.         0.8944272]
 [-0.         1.         0.        -0.         0.       ]
 [ 0.         0.         0.         1.         0.       ]
 [-0.8944272  0.         0.         0.         0.4472136]]
```

Matrix eigenvalues

Q8. Compute the eigenvalues and right eigenvectors of x. (Name them eigenvals and eigenvecs, respectively)

In [11]:

```
x = np.diag((1, 2, 3))
x = np.diag((1, 2, 3))
eigenvals, eigenvecs = np.linalg.eig(x)
print("eigenvals = {} \neigenvecs=\n{}".format(eigenvals,eigenvecs))
```

```
eigenvals = [1. 2. 3.]
eigenvecs=
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Q9. Predict the results of the following code.

In [12]:

```
print(np.array_equal(np.dot(x, eigenvecs), eigenvals * eigenvecs))
```

```
True
```

Norms and other numbers

Q10. Calculate the Frobenius norm and the condition number of x.

In [13]:

```
x = np.arange(1, 10).reshape((3, 3))
print(np.linalg.norm(x,ord='fro'))
print(np.linalg.cond(x,p='fro'))
```

```
16.881943016134134
4.561770736605098e+17
```

Q11. Calculate the determinant of x.

In [14]:

```
x = np.arange(1, 5).reshape((2, 2))
np.linalg.det(x)
```

Out[14]:

```
-2.0000000000000004
```

Q12. Calculate the rank of x.

In [15]:

```
x = np.eye(4)
np.linalg.matrix_rank(x)
```

Out[15]:

```
4
```

Q13. Compute the sign and natural logarithm of the determinant of x.

In [16]:

```
x = np.arange(1, 5).reshape((2, 2))
np.linalg.slogdet(x)
```

Out[16]:

```
(-1.0, 0.6931471805599455)
```

Q14. Return the sum along the diagonal of x.

In [17]:

```
x = np.eye(4)
np.trace(x)
```

Out[17]:

```
4.0
```

Solving equations and inverting matrices

Q15. Compute the inverse of x.

In [21]:

```
x = np.array([[1., 2.], [3., 4.]])
print(np.linalg.inv(x))
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

In [ ]:

```

```

Logic functions

In [2]:

```
import numpy as np
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.21.5'
```

Truth value testing

Q1. Let x be an arbitrary array. Return True if none of the elements of x is zero. Remind that 0 evaluates to False in python.

In [4]:

```
x = np.array([1,2,3])
print(np.all(x))

x = np.array([1,0,3])
print(np.all(x))
```

```
True
False
```

Q2. Let x be an arbitrary array. Return True if any of the elements of x is non-zero.

In [13]:

```
x = np.array([1,0,0])
print(np.any(x))
x = np.array([0,0,0])
print(np.any(y))
```

```
True
False
```

Array contents

Q3. Predict the result of the following code.

In [14]:

```
x = np.array([1, 0, np.nan, np.inf])
print(np.isfinite(x))
```

```
[ True  True False False]
```

Q4. Predict the result of the following code.

In [15]:

```
x = np.array([1, 0, np.nan, np.inf])
print(np.isinf(x))
```

```
[False False False  True]
```

Q5. Predict the result of the following code.

In [16]:

```
x = np.array([1, 0, np.nan, np.inf])
print(np.isnan(x))
```

```
[False False  True False]
```

Array type testing

Q6. Predict the result of the following code.

In [17]:

```
x = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
print(np.iscomplex(x))
```

```
[ True False False False False  True]
```

Q7. Predict the result of the following code.

In [18]:

```
x = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
print(np.isreal(x))
```

```
[False  True  True  True  True False]
```

Q8. Predict the result of the following code.

In [21]:

```
print(np.isscalar(3))
print(np.isscalar([3]))
print(np.isscalar(True))
```

```
True
False
True
```

Logical operations

Q9. Predict the result of the following code.

In [22]:

```
print(np.logical_and([True, False], [False, False]))
print(np.logical_or([True, False, True], [True, False, False]))
print(np.logical_xor([True, False, True], [True, False, False]))
print(np.logical_not([True, False, 0, 1]))
```

```
[False False]
[ True False  True]
[False False  True]
[False  True  True False]
```

Comparison

Q10. Predict the result of the following code.

In [23]:

```
print(np.allclose([3], [2.999999]))
print(np.array_equal([3], [2.999999]))
```

```
True
False
```

Q11. Write numpy comparison functions such that they return the results as you see.

In [3]:

```
x = np.array([4, 5])
y = np.array([2, 5])
print(np.greater(x, y))
print(np.greater_equal(x, y))
print(np.less(x, y))
print(np.less_equal(x, y))
```

```
[ True False]
[ True  True]
[False False]
[False  True]
```

Q12. Predict the result of the following code.

In [4]:

```
print(np.equal([1, 2], [1, 2.000001]))
print(np.isclose([1, 2], [1, 2.000001]))
```

```
[ True False]
[ True  True]
```

In [ ]:

```

```

Mathematical functions

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Trigonometric functions

Q1. Calculate sine, cosine, and tangent of x, element-wise.

In [3]:

```
x = np.array([0., 1., 30, 90])
print("sine: {} \ncosine: {} \ntangent: {}".format(np.sin(x),np.cos(x),np.tan(x)))
```

```
sine: [ 0.          0.84147098 -0.98803162  0.89399666]
cosine: [ 1.          0.54030231  0.15425145 -0.44807362]
tangent: [ 0.          1.55740772 -6.4053312  -1.99520041]
```

Q2. Calculate inverse sine, inverse cosine, and inverse tangent of x, element-wise.

In [5]:

```
x = np.array([-1., 0, 1.])
print("inversesine: {} \ninversecosine: {} \ninversetangent: {}".format(np.arcsin(x),np.arccos(x),np.arctan(x)))
```

```
inversesine: [-1.57079633  0.          1.57079633]
inversecosine: [3.14159265 1.57079633 0.        ]
inversetangent: [-0.78539816  0.          0.78539816]
```

Q3. Convert angles from radians to degrees.

In [6]:

```
x = np.array([-np.pi, -np.pi/2, np.pi/2, np.pi])
deg = np.degrees(x)
print(deg)
```

```
[-180.  -90.   90.  180.]
```

Q4. Convert angles from degrees to radians.

In [8]:

```
x = np.array([-180.,  -90.,   90.,  180.])
rad=np.deg2rad(x)
print(rad)
```

```
[-3.14159265 -1.57079633  1.57079633  3.14159265]
```

Hyperbolic functions

Q5. Calculate hyperbolic sine, hyperbolic cosine, and hyperbolic tangent of x, element-wise.

In [9]:

```
x = np.array([-1., 0, 1.])
print(np.sinh(x))
print(np.cosh(x))
print(np.tanh(x))
```

```
[-1.17520119  0.          1.17520119]
[1.54308063 1.         1.54308063]
[-0.76159416  0.          0.76159416]
```

Rounding

Q6. Predict the results of these, paying attention to the difference among the family functions.

In [10]:

```
x = np.array([2.1, 1.5, 2.5, 2.9, -2.1, -2.5, -2.9])

out1 = np.around(x)
out2 = np.floor(x)
out3 = np.ceil(x)
out4 = np.trunc(x)
out5 = [round(elem) for elem in x]

print(out1)
print(out2)
print(out3)
print(out4)
print(out5)
```

```
[ 2.  2.  2.  3. -2. -2. -3.]
[ 2.  1.  2.  2. -3. -3. -3.]
[ 3.  2.  3.  3. -2. -2. -2.]
[ 2.  1.  2.  2. -2. -2. -2.]
[2, 2, 2, 3, -2, -2, -3]
```

Q7. Implement out5 in the above question using numpy.

In [11]:

```
print(np.round(x))
```

```
[ 2.  2.  2.  3. -2. -2. -3.]
```

Sums, products, differences

Q8. Predict the results of these.

In [12]:

```
x = np.array(
    [[1, 2, 3, 4],
     [5, 6, 7, 8]])

outs = [np.sum(x),
        np.sum(x, axis=0),
        np.sum(x, axis=1, keepdims=True),
        "",
        np.prod(x),
        np.prod(x, axis=0),
        np.prod(x, axis=1, keepdims=True),
        "",
        np.cumsum(x),
        np.cumsum(x, axis=0),
        np.cumsum(x, axis=1),
        "",
        np.cumprod(x),
        np.cumprod(x, axis=0),
        np.cumprod(x, axis=1),
        "",
        np.min(x),
        np.min(x, axis=0),
        np.min(x, axis=1, keepdims=True),
        "",
        np.max(x),
        np.max(x, axis=0),
        np.max(x, axis=1, keepdims=True),
        "",
        np.mean(x),
        np.mean(x, axis=0),
        np.mean(x, axis=1, keepdims=True)]

for out in outs:
    if out == "":
        pass
        print
    else:
        pass
        print("->", out)
```

```
-> 36
-> [ 6  8 10 12]
-> [[10]
 [26]]
-> 40320
-> [ 5 12 21 32]
-> [[  24]
 [1680]]
-> [ 1  3  6 10 15 21 28 36]
-> [[ 1  2  3  4]
 [ 6  8 10 12]]
-> [[ 1  3  6 10]
 [ 5 11 18 26]]
-> [    1     2     6    24   120   720  5040 40320]
-> [[ 1  2  3  4]
 [ 5 12 21 32]]
-> [[   1    2    6   24]
 [   5   30  210 1680]]
-> 1
-> [1 2 3 4]
-> [[1]
 [5]]
-> 8
-> [5 6 7 8]
-> [[4]
 [8]]
-> 4.5
-> [3. 4. 5. 6.]
-> [[2.5]
 [6.5]]

C:\Users\khlbg\AppData\Local\Temp\ipykernel_16696\3184732887.py:34: FutureWarning: elementwise comparison failed; returning
scalar instead, but in the future will perform elementwise comparison
  if out == "":
```

Q9. Calculate the difference between neighboring elements, element-wise.

In [13]:

```
x = np.array([1, 2, 4, 7, 0])
print(np.diff(x))
```

```
[ 1  2  3 -7]
```

Q10. Calculate the difference between neighboring elements, element-wise, and prepend [0, 0] and append[100] to it.

In [17]:

```
x = np.array([1, 2, 4, 7, 0])
print(np.ediff1d(x, to_begin=[0, 0], to_end=[100]))
```

```
[  0   0   1   2   3  -7 100]
```

Q11. Return the cross product of x and y.

In [18]:

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
z=np.cross(x,y)
print(z)
```

```
[-3  6 -3]
```

Exponents and logarithms

Q12. Compute e^x, element-wise.

In [19]:

```
x = np.array([1., 2., 3.], np.float32)
print(np.exp(x))
```

```
[ 2.718282   7.3890557 20.085537 ]
```

Q13. Calculate exp(x) - 1 for all elements in x.

In [20]:

```
x = np.array([1., 2., 3.], np.float32)
z=np.exp(x)-1
print(z)
```

```
[ 1.718282   6.3890557 19.085537 ]
```

Q14. Calculate 2^p for all p in x.

In [23]:

```
x = np.array([1., 2., 3.], np.float32)
r1=np.exp2(x)
r2=2**x
assert np.allclose(r1,r2)
print(r1)
```

```
[2. 4. 8.]
```

Q15. Compute natural, base 10, and base 2 logarithms of x element-wise.

In [24]:

```
x = np.array([1, np.e, np.e**2])
print("natural log = {}".format(np.log(x)))
print("common log = {}".format(np.log10(x)))
print("base 2 log = {}".format(np.log2(x)))
```

```
natural log = [0. 1. 2.]
common log = [0.         0.43429448 0.86858896]
base 2 log = [0.         1.44269504 2.88539008]
```

Q16. Compute the natural logarithm of one plus each element in x in floating-point accuracy.

In [25]:

```
x = np.array([1e-99, 1e-100])
print(np.log1p(x))
```

```
[1.e-099 1.e-100]
```

Floating point routines

Q17. Return element-wise True where signbit is set.

In [26]:

```
x = np.array([-3, -2, -1, 0, 1, 2, 3])
print(np.signbit(x))
```

```
[ True  True  True False False False False]
```

Q18. Change the sign of x to that of y, element-wise.

In [27]:

```
x = np.array([-1, 0, 1])
y = -1.1
np.copysign(x,y)
```

Out[27]:

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

Arithmetic operations

Q19. Add x and y element-wise.

In [30]:

```
x = np.array([1, 2, 3])
y = np.array([-1, -2, -3])
print(x+y)
```

```
[0 0 0]
```

Q20. Subtract y from x element-wise.

In [36]:

```
x = np.array([3, 4, 5])
y = np.array(3)
print(np.subtract(x,y))
```

```
[0 1 2]
```

Q21. Multiply x by y element-wise.

In [37]:

```
x = np.array([3, 4, 5])
y = np.array([1,0,-1])
print(np.multiply(x,y))
```

```
[ 3  0 -5]
```

Q22. Divide x by y element-wise in two different ways.

In [41]:

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

x/y
print(np.floor_divide(x,y))
```

```
[3.         2.         1.66666667]
[3. 2. 1.]
```

Q23. Compute numerical negative value of x, element-wise.

In [46]:

```
x = np.array([1, -1])
print(np.negative(x))
```

```
[-1  1]
```

Q24. Compute the reciprocal of x, element-wise.

In [47]:

```
x = np.array([1., 2., .2])
print(np.reciprocal(x))
```

```
[1.  0.5 5. ]
```

Q25. Compute x^y, element-wise.

In [49]:

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

```
[[ 1  4]
 [ 3 16]]
```

Q26. Compute the remainder of x / y element-wise in two different ways.

In [50]:

```
x = np.array([-3, -2, -1, 1, 2, 3])
y = 2
print(np.remainder(x,y))
print(np.fmod(x,y))
```

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

Miscellaneous

Q27. If an element of x is smaller than 3, replace it with 3. And if an element of x is bigger than 7, replace it with 7.

In [51]:

```
x = np.arange(10)
print(np.clip(x,3,7))
```

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

Q28. Compute the square of x, element-wise.

In [52]:

```
x = np.array([1, 2, -1])
print(np.square(x))
```

```
[1 4 1]
```

Q29. Compute square root of x element-wise.

In [53]:

```
x = np.array([1., 4., 9.])
print(np.sqrt(x))
```

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

Q30. Compute the absolute value of x.

In [54]:

```
x = np.array([[1, -1], [3, -3]])
print(np.abs(x))
```

```
[[1 1]
 [3 3]]
```

Q31. Compute an element-wise indication of the sign of x, element-wise.

In [55]:

```
x = np.array([1, 3, 0, -1, -3])
print(np.sign(x))
```

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

In [ ]:

Random Sampling

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Simple random data

Q1. Create an array of shape (3, 2) and populate it with random samples from a uniform distribution over [0, 1).

In [3]:

```
np.random.rand(3,2)
```

Out[3]:

```
array([[0.29242216, 0.16534313],
       [0.73247743, 0.65175047],
       [0.55577736, 0.45815693]])
```

Q2. Create an array of shape (1000, 1000) and populate it with random samples from a standard normal distribution. And verify that the mean and standard deviation is close enough to 0 and 1 repectively.

In [4]:

```
x = np.random.standard_normal(size=(1000,1000))
print(np.mean(x))
print(np.std(x))
```

```
-0.0005427181403533473
1.0007258191643718
```

Q3. Create an array of shape (3, 2) and populate it with random integers ranging from 0 to 3 (inclusive) from a discrete uniform distribution.

In [5]:

```
np.random.randint(0,4,size=(3,2))
```

Out[5]:

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

Q4. Extract 1 elements from x randomly such that each of them would be associated with probabilities .3, .5, .2. Then print the result 10 times.

In [6]:

```
x = [b'3 out of 10', b'5 out of 10', b'2 out of 10']
for i in range(10):
    print(np.random.choice(x,p=[0.3, 0.5,0.2]))
```

```
b'2 out of 10'
b'5 out of 10'
b'5 out of 10'
b'3 out of 10'
b'5 out of 10'
b'5 out of 10'
b'2 out of 10'
b'3 out of 10'
b'3 out of 10'
b'5 out of 10'
```

Q5. Extract 3 different integers from 0 to 9 randomly with the same probabilities.

In [7]:

```
np.random.choice(range(10),size=3)
```

Out[7]:

```
array([7, 4, 4])
```

Permutations

Q6. Shuffle numbers between 0 and 9 (inclusive).

In [8]:

```
x = np.arange(10)
np.random.shuffle(x)
print(x)
```

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

In [88]:

```
# Or
```

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

Random generator

Q7. Assign number 10 to the seed of the random generator so that you can get the same value next time.

In [9]:

```
np.random.seed(1)
np.random.rand(2)
```

Out[9]:

```
array([0.417022  , 0.72032449])
```

Set routines

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Making proper sets

Q1. Get unique elements and reconstruction indices from x. And reconstruct x.

In [4]:

```
x = np.array([1, 2, 6, 4, 2, 3, 2])
out, indices = np.unique(x, return_inverse=True)
print ("unique elements =", out)
print ("reconstruction indices =", indices)
print ("reconstructed =", out[indices])
```

```
unique elements = [1 2 3 4 6]
reconstruction indices = [0 1 4 3 1 2 1]
reconstructed = [1 2 6 4 2 3 2]
```

Boolean operations

Q2. Create a boolean array of the same shape as x. If each element of x is present in y, the result will be True, otherwise False.

In [5]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1])
print (np.in1d(x, y))
```

```
[ True  True False False  True]
```

Q3. Find the unique intersection of x and y.

In [6]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
print (np.intersect1d(x, y))
```

```
[0 1]
```

Q4. Find the unique elements of x that are not present in y.

In [7]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
print (np.setdiff1d(x, y))
```

```
[2 5]
```

Q5. Find the xor elements of x and y.

In [8]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
out1 = np.setxor1d(x, y)
out2 = np.sort(np.concatenate((np.setdiff1d(x, y), np.setdiff1d(y, x))))
assert np.allclose(out1, out2)
print (out1)
```

```
[2 4 5]
```

Q6. Find the union of x and y.

In [10]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
out1 = np.union1d(x, y)
out2 = np.sort(np.unique(np.concatenate((x, y))))
assert np.allclose(out1, out2)
print( np.union1d(x, y))
```

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

In [ ]:

```
```

Soring, searching, and counting

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Sorting

Q1. Sort x along the second axis.

In [3]:

```
x = np.array([[1,4],[3,1]])
output = np.sort(x, axis=1)
x.sort(axis=1)
print(output)
```

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

Q2. Sort pairs of surnames and first names and return their indices. (first by surname, then by name).

In [4]:

```
surnames = ('Hertz', 'Galilei', 'Hertz')
first_names = ('Heinrich', 'Galileo', 'Gustav')
print( np.lexsort((first_names, surnames)))
```

```
[1 2 0]
```

Q3. Get the indices that would sort x along the second axis.

In [6]:

```
x = np.array([[1,4],[3,1]])
result = np.argsort(x, axis=1)
print( result)
```

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

Q4. Create an array such that its fifth element would be the same as the element of sorted x, and it divide other elements by their value.

In [8]:

```
x = np.random.permutation(10)
print ("x =", x)
print( "\nCheck the fifth element of this new array is 5, the first four elements are all same")
output = (np.partition(x, 5))
x.partition(5)
print (output)
```

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

Check the fifth element of this new array is 5, the first four elements are all same
[1 0 2 3 4 5 6 7 8 9]
```

Q5. Create the indices of an array such that its third element would be the same as the element of sorted x, and it divide other elements by their value.

In [13]:

```
x = np.random.permutation(10)
print ("x =", x)
partitioned = np.partition(x, 3)
indices = np.argpartition(x, 3)
print ("partitioned =", partitioned)
print ("indices =", partitioned)
```

```
x = [4 1 7 0 2 3 5 9 6 8]
partitioned = [0 1 2 3 4 7 5 9 6 8]
indices = [0 1 2 3 4 7 5 9 6 8]
```

Searching

Q6. Get the maximum and minimum values and their indices of x along the second axis.

In [16]:

```
x = np.random.permutation(10).reshape(2,5)
print("x=",x)
print("maximum values=",np.max(x,1))
print("max indices=",np.argmax(x,1))
print("minimum values=",np.min(x,1))
print("min indices=",np.argmin(x,1))
```

```
x= [[3 6 2 1 9]
 [0 7 4 8 5]]
maximum values= [9 8]
max indices= [4 3]
minimum values= [1 0]
min indices= [3 0]
```

Q7. Get the maximum and minimum values and their indices of x along the second axis, ignoring NaNs.

In [17]:

```
x = np.array([[np.nan, 4], [3, 2]])
print ("maximum values ignoring NaNs =", np.nanmax(x, 1))
print( "max indices =", np.nanargmax(x, 1))
print( "minimum values ignoring NaNs =", np.nanmin(x, 1))
print ("min indices =", np.nanargmin(x, 1))
```

```
maximum values ignoring NaNs = [4. 3.]
max indices = [1 0]
minimum values ignoring NaNs = [4. 2.]
min indices = [1 1]
```

Q8. Get the values and indices of the elements that are bigger than 2 in x.

In [18]:

```
x = np.array([[1, 2, 3], [1, 3, 5]])
print( "Values bigger than 2 =", x[x>2])
print ("Their indices are ", np.nonzero(x > 2))
assert( np.array_equiv(x[x>2], x[np.nonzero(x > 2)]))
assert (np.array_equiv(x[x>2], np.extract(x > 2, x)))
```

```
Values bigger than 2 = [3 3 5]
Their indices are  (array([0, 1, 1], dtype=int64), array([2, 1, 2], dtype=int64))
```

Q9. Get the indices of the elements that are bigger than 2 in the flattend x.

In [19]:

```
x = np.array([[1, 2, 3], [1, 3, 5]])
print( np.flatnonzero(x>2))
assert (np.array_equiv(np.flatnonzero(x), x.ravel().nonzero()))
```

```
[2 4 5]
```

Q10. Check the elements of x and return 0 if it is less than 0, otherwise the element itself.

In [23]:

```
x=np.arange(-5,4).reshape(3,3)
print(np.where(x < 0, 0, x))
```

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

Q11. Get the indices where elements of y should be inserted to x to maintain order.

In [25]:

```
x = [1, 3, 5, 7, 9]
y = [0, 4, 2, 6]
array = np.searchsorted(x, y)
print(array)
```

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

Counting

Q12. Get the number of nonzero elements in x.

In [26]:

```
x = [[0,1,7,0,0],[3,0,0,2,19]]
print (np.count_nonzero(x))
```

```
5
```

In [ ]:

```
```

Statistics

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.21.5'
```

Order statistics

Q1. Return the minimum value of x along the second axis.

In [3]:

```
x = np.arange(4).reshape((2, 2))
print("x=\n", x)
print("ans=\n", np.amin(x, 1))
```

```
x=
 [[0 1]
 [2 3]]
ans=
 [0 2]
```

Q2. Return the maximum value of x along the second axis. Reduce the second axis to the dimension with size one.

In [4]:

```
x = np.arange(4).reshape((2, 2))
print("x=\n", x)
print("ans=\n", np.amax(x, 1, keepdims=True))
```

```
x=
 [[0 1]
 [2 3]]
ans=
 [[1]
 [3]]
```

Q3. Calcuate the difference between the maximum and the minimum of x along the second axis.

In [5]:

```
x = np.arange(10).reshape((2, 5))
print("x=\n", x)
y = np.ptp(x, 1)
z = np.amax(x, 1) - np.amin(x, 1)
print("ans=\n", y)
```

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

Q4. Compute the 75th percentile of x along the second axis.

In [6]:

```
x = np.arange(1, 11).reshape((2, 5))
print("x=\n", x)
print("ans=\n", np.percentile(x, 75, 1))
```

```
x=
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
ans=
 [4. 9.]
```

Averages and variances

Q5. Compute the median of flattened x.

In [7]:

```
x = np.arange(1, 10).reshape((3, 3))
print("x=\n", x)
print("ans=\n", np.median(x))
```

```
x=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
ans=
 5.0
```

Q6. Compute the weighted average of x.

In [8]:

```
x = np.arange(5)
weights = np.arange(1, 6)
y = np.average(x, weights=weights)
z = (x*(weights/weights.sum())).sum()
print(y)
```

```
2.6666666666666665
```

Q7. Compute the mean, standard deviation, and variance of x along the second axis.

In [10]:

```
x = np.arange(5)
print("x=\n",x)
a = np.mean(x)
b = np.average(x)
print("mean=\n", a)
c = np.std(x)
d = np.sqrt(np.mean((x - np.mean(x)) ** 2 ))
print("std=\n", c)
e = np.var(x)
f = np.mean((x - np.mean(x)) ** 2 )
print("variance=\n", e)
```

```
x=
 [0 1 2 3 4]
mean=
 2.0
std=
 1.4142135623730951
variance=
 2.0
```

Correlating

Q8. Compute the covariance matrix of x and y.

In [12]:

```
x=np.array([0,1,2])
y=np.array([2,1,0])
print("ans=\n",np.cov(x,y))
```

```
ans=
 [[ 1. -1.]
 [-1.  1.]]
```

Q9. In the above covariance matrix, what does the -1 mean?

Type *Markdown* and LaTeX: α^2

Q10. Compute Pearson product-moment correlation coefficients of x and y.

In [13]:

```
x = np.array([0, 1, 3])
y = np.array([2, 4, 5])
print("ans=\n", np.corrcoef(x, y))
```

```
ans=
 [[1.         0.92857143]
 [0.92857143 1.        ]]
```

Q11. Compute cross-correlation of x and y.

In [14]:

```
x = np.array([0, 1, 3])
y = np.array([2, 4, 5])
print("ans=\n", np.correlate(x, y))
```

```
ans=
 [19]
```

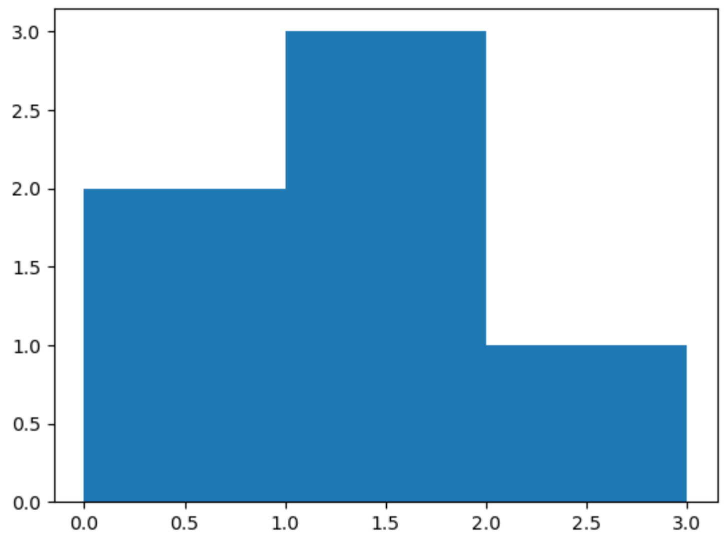
Histograms

Q12. Compute the histogram of x against the bins.

In [15]:

```
x = np.array([0.5, 0.7, 1.0, 1.2, 1.3, 2.1])
bins = np.array([0, 1, 2, 3])
print("ans=\n", np.histogram(x, bins))
import matplotlib.pyplot as plt
%matplotlib inline
plt.hist(x, bins=bins)
plt.show()
```

```
ans=
 (array([2, 3, 1], dtype=int64), array([0, 1, 2, 3]))
```

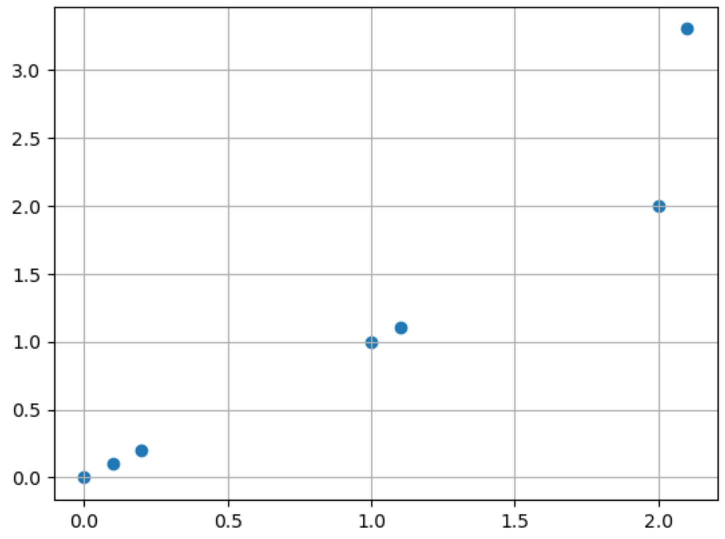


Q13. Compute the 2d histogram of x and y.

In [16]:

```
xedges = [0, 1, 2, 3]
yedges = [0, 1, 2, 3, 4]
x = np.array([0, 0.1, 0.2, 1., 1.1, 2., 2.1])
y = np.array([0, 0.1, 0.2, 1., 1.1, 2., 3.3])
H, xedges, yedges = np.histogram2d(x, y, bins=(xedges, yedges))
print("ans=\n", H)
plt.scatter(x, y)
plt.grid()
```

```
ans=
 [[3. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 1. 1.]]
```



Q14. Count number of occurrences of 0 through 7 in x.

In [17]:

```
x = np.array([0, 1, 1, 3, 2, 1, 7])
print("ans=\n", np.bincount(x))
```

```
ans=
 [1 3 1 1 0 0 0 1]
```

Q15. Return the indices of the bins to which each value in x belongs.

In [18]:

```
x = np.array([0.2, 6.4, 3.0, 1.6])
bins = np.array([0.0, 1.0, 2.5, 4.0, 10.0])
print("ans=\n", np.digitize(x, bins))
```

```
ans=
 [1 4 3 2]
```

In [ ]:

```

```