numpy

In [6]:
```
import numpy as np

a = np.array([1, 2, 3, 4, 5], dtype='i')  # integer type
b = np.array((1, 3, 6, 9, 12), dtype='f')  # float type
print(a)
print(b)
```

```
[1 2 3 4 5]
[ 1.  3.  6.  9. 12.]
```

In [7]:
```
type(a)
```

Out[7]:
```
numpy.ndarray
```

In [8]:
```
type(b)
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
a.dtype
```

Out[9]:
```
dtype('int32')
```

In [10]:
```
b.dtype
```

Out[10]:
```
dtype('float32')
```

In [12]:
```
c = np.array([[1, 2, 3], [4, 5, 6]])
print(c.ndim)
print(c)
```

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

In [14]:
```
d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(d.ndim)
print(d)
```

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

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

In [15]:
```
print(d.shape[0], d.shape[1], d.shape[2])
```

```
2 2 3
```

In [17]:
```
print(d[1, 0, 2])
```

```
9
```

In [20]:
```
print(np.shape(d))
```

```
(2, 2, 3)
```

In [21]:
```
d.shape
```

Out[21]:
```
(2, 2, 3)
```

In [22]:
```
b.shape
```

Out[22]:
```
(5,)
```

In [23]:
```
c.size
```

Out[23]:
```
6
```

In [25]:
```
c.nbytes
```

Out[25]:
```
24
```

In [26]:
```
d.nbytes
```

Out[26]:
```
48
```

In [28]:
```
A = np.arange(100)
print(A)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

In [30]:
```
B = np.arange(20, 100)
print(B)
```

```
[20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
 92 93 94 95 96 97 98 99]
```

In [31]:
```
C = np.arange(20, 100, 3)
print(C)
```

```
[20 23 26 29 32 35 38 41 44 47 50 53 56 59 62 65 68 71 74 77 80 83 86 89
 92 95 98]
```

In [32]:
```
print(list(range(10)))
```

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

In [33]:
```
D = np.random.permutation(np.arange(10))
print(D)
```

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

In [34]:
```
np.random.randint
```

Out[34]:
```
<function RandomState.randint>
```

In [35]:
```
np.random.randint(20, 30)
```

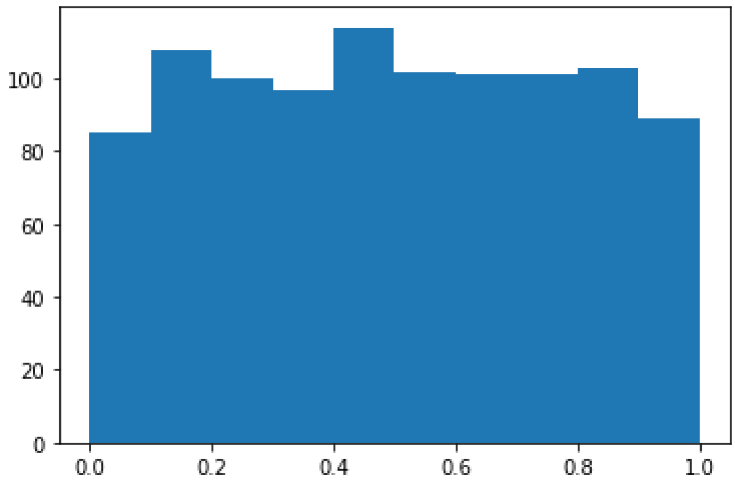
Out[35]:
```
22
```

In [36]:
```
E = np.random.rand(1000)
```

In [38]:
```
import matplotlib.pyplot as plt
```

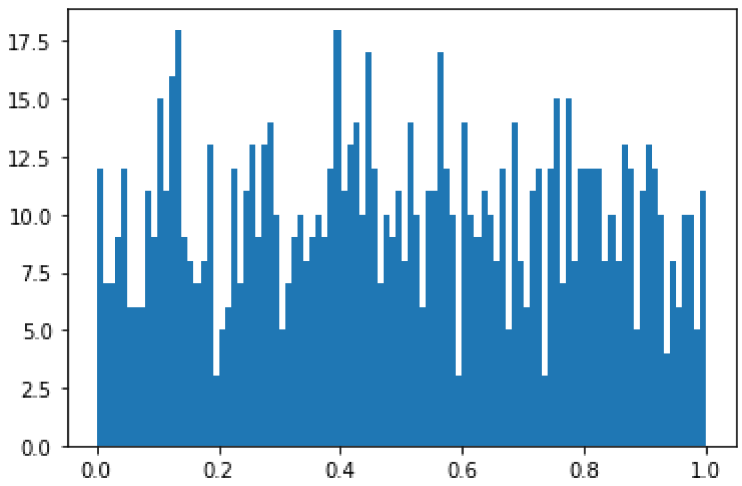
In [39]:
```
plt.hist(E)
```

Out[39]:
```
(array([ 85., 108., 100.,  97., 114., 102., 101., 101., 103.,  89.]),
 array([5.62133819e-04, 1.00489502e-01, 2.00416870e-01, 3.00344237e-01,
        4.00271605e-01, 5.00198973e-01, 6.00126341e-01, 7.00053709e-01,
        7.99981077e-01, 8.99908445e-01, 9.99835813e-01]),
 <a list of 10 Patch objects>)
```



In [40]:
```
plt.hist(E, bins = 100)
```

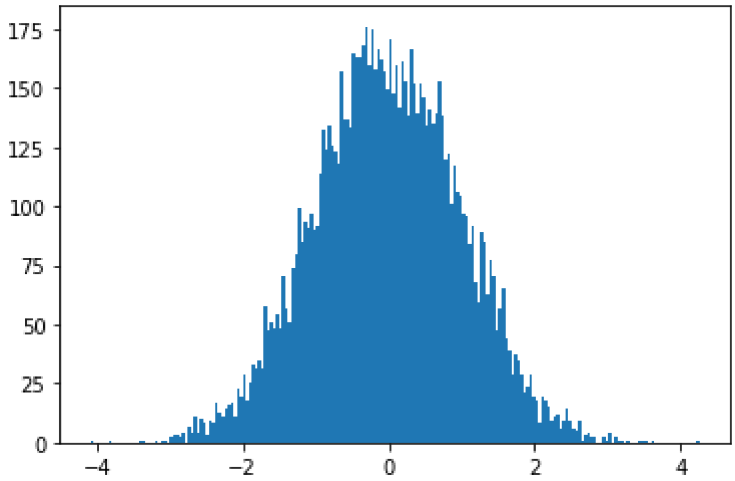
Out[40]:
```
(array([12.,  7.,  7.,  9., 12.,  6.,  6.,  6., 11.,  9., 15., 11., 16.,
        18.,  9.,  8.,  7.,  8., 13.,  3.,  5.,  6., 12.,  7., 11., 13.,
         9., 13., 14., 10.,  5.,  7.,  9., 10.,  8.,  9., 10.,  9., 12.,
        18., 11., 13., 14., 10., 17., 12.,  7., 10.,  9., 11.,  8., 14.,
        10.,  6., 11., 11., 17., 12., 10.,  3., 14., 10.,  9., 11., 10.,
         8., 12.,  5., 14.,  8.,  6., 11., 12.,  3., 12., 15.,  7., 15.,
         8., 12., 12., 12., 12.,  8., 10.,  8., 13., 12.,  5., 11., 13.,
        12., 10.,  4.,  8.,  6., 10., 10.,  5., 11.]),
 array([5.62133819e-04, 1.05548706e-02, 2.05476074e-02, 3.05403442e-02,
        4.05330810e-02, 5.05258178e-02, 6.05185545e-02, 7.05112913e-02,
        8.05040281e-02, 9.04967649e-02, 1.00489502e-01, 1.10482238e-01,
        1.20474975e-01, 1.30467712e-01, 1.40460449e-01, 1.50453186e-01,
        1.60445922e-01, 1.70438659e-01, 1.80431396e-01, 1.90424133e-01,
        2.00416870e-01, 2.10409606e-01, 2.20402343e-01, 2.30395080e-01,
        2.40387817e-01, 2.50380554e-01, 2.60373290e-01, 2.70366027e-01,
        2.80358764e-01, 2.90351501e-01, 3.00344237e-01, 3.10336974e-01,
        3.20329711e-01, 3.30322448e-01, 3.40315185e-01, 3.50307921e-01,
        3.60300658e-01, 3.70293395e-01, 3.80286132e-01, 3.90278869e-01,
        4.00271605e-01, 4.10264342e-01, 4.20257079e-01, 4.30249816e-01,
        4.40242552e-01, 4.50235289e-01, 4.60228026e-01, 4.70220763e-01,
        4.80213500e-01, 4.90206236e-01, 5.00198973e-01, 5.10191710e-01,
        5.20184447e-01, 5.30177184e-01, 5.40169920e-01, 5.50162657e-01,
        5.60155394e-01, 5.70148131e-01, 5.80140868e-01, 5.90133604e-01,
        6.00126341e-01, 6.10119078e-01, 6.20111815e-01, 6.30104551e-01,
        6.40097288e-01, 6.50090025e-01, 6.60082762e-01, 6.70075499e-01,
        6.80068235e-01, 6.90060972e-01, 7.00053709e-01, 7.10046446e-01,
        7.20039183e-01, 7.30031919e-01, 7.40024656e-01, 7.50017393e-01,
        7.60010130e-01, 7.70002866e-01, 7.79995603e-01, 7.89988340e-01,
        7.99981077e-01, 8.09973814e-01, 8.19966550e-01, 8.29959287e-01,
        8.39952024e-01, 8.49944761e-01, 8.59937498e-01, 8.69930234e-01,
        8.79922971e-01, 8.89915708e-01, 8.99908445e-01, 9.09901182e-01,
        9.19893918e-01, 9.29886655e-01, 9.39879392e-01, 9.49872129e-01,
        9.59864865e-01, 9.69857602e-01, 9.79850339e-01, 9.89843076e-01,
        9.99835813e-01]),
 <a list of 100 Patch objects>)
```



In [41]:
```
F = np.random.randn(10000)
plt.hist(F, bins=200)
```

Out[41]: (array([1.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,
                   0.,   0.,   0.,   0.,   0.,   1.,   1.,   0.,   0.,   0.,   1.,
                   0.,   1.,   1.,   0.,   2.,   3.,   3.,   2.,   4.,   0.,   7.,
                   4., 11.,   4., 10.,   8.,   3.,   9.,   8., 17., 13., 11.,
                  14., 16., 17., 11., 23., 19., 29., 18., 25., 33., 31.,
                  35., 31., 58., 47., 51., 48., 54., 48., 70., 57., 51.,
                  74., 80., 99., 85., 93., 91., 97., 90., 92., 114., 132.,
                 124., 134., 126., 123., 118., 157., 137., 137., 133., 165., 163.,
                 163., 168., 176., 160., 175., 158., 166., 162., 157., 149., 171.,
                 148., 160., 142., 161., 153., 138., 166., 152., 139., 152., 146.,
                 134., 141., 135., 139., 153., 138., 120., 122., 101., 117., 106.,
                 104., 97., 96., 84., 92., 68., 59., 89., 85., 63., 77.,
                  70., 47., 57., 65., 44., 39., 29., 37., 35., 29., 21.,
                  24., 29., 19., 18.,   8., 19., 18., 15.,   9., 11., 12.,
                   6.,   9., 14.,   9.,   6.,   5.,   9.,   1.,   3.,   4.,   2.,
                   2.,   0.,   0.,   2.,   1.,   4.,   0.,   2.,   1.,   1.,   0.,
                   1.,   0.,   0.,   0.,   1.,   1.,   1.,   0.,   1.,   0.,   0.,
                   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
                   0.,   1.]),
          array([-4.09036316e+00, -4.04858566e+00, -4.00680815e+00, -3.96503065e+00,
                 -3.92325315e+00, -3.88147565e+00, -3.83969815e+00, -3.79792064e+00,
                 -3.75614314e+00, -3.71436564e+00, -3.67258814e+00, -3.63081064e+00,
                 -3.58903313e+00, -3.54725563e+00, -3.50547813e+00, -3.46370063e+00,
                 -3.42192312e+00, -3.38014562e+00, -3.33836812e+00, -3.29659062e+00,
                 -3.25481312e+00, -3.21303561e+00, -3.17125811e+00, -3.12948061e+00,
                 -3.08770311e+00, -3.04592561e+00, -3.00414810e+00, -2.96237060e+00,
                 -2.92059310e+00, -2.87881560e+00, -2.83703810e+00, -2.79526059e+00,
                 -2.75348309e+00, -2.71170559e+00, -2.66992809e+00, -2.62815058e+00,
                 -2.58637308e+00, -2.54459558e+00, -2.50281808e+00, -2.46104058e+00,
                 -2.41926307e+00, -2.37748557e+00, -2.33570807e+00, -2.29393057e+00,
                 -2.25215307e+00, -2.21037556e+00, -2.16859806e+00, -2.12682056e+00,
                 -2.08504306e+00, -2.04326556e+00, -2.00148805e+00, -1.95971055e+00,
                 -1.91793305e+00, -1.87615555e+00, -1.83437804e+00, -1.79260054e+00,
                 -1.75082304e+00, -1.70904554e+00, -1.66726804e+00, -1.62549053e+00,
                 -1.58371303e+00, -1.54193553e+00, -1.50015803e+00, -1.45838053e+00,
                 -1.41660302e+00, -1.37482552e+00, -1.33304802e+00, -1.29127052e+00,
                 -1.24949302e+00, -1.20771551e+00, -1.16593801e+00, -1.12416051e+00,
                 -1.08238301e+00, -1.04060550e+00, -9.98828003e-01, -9.57050501e-01,
                 -9.15272999e-01, -8.73495496e-01, -8.31717994e-01, -7.89940492e-01,
                 -7.48162990e-01, -7.06385488e-01, -6.64607986e-01, -6.22830484e-01,
                 -5.81052982e-01, -5.39275480e-01, -4.97497977e-01, -4.55720475e-01,
                 -4.13942973e-01, -3.72165471e-01, -3.30387969e-01, -2.88610467e-01,
                 -2.46832965e-01, -2.05055463e-01, -1.63277961e-01, -1.21500458e-01,
                 -7.97229564e-02, -3.79454543e-02, 3.83204783e-03, 4.56095499e-02,
                  8.73870520e-02, 1.29164554e-01, 1.70942056e-01, 2.12719558e-01,
                  2.54497060e-01, 2.96274563e-01, 3.38052065e-01, 3.79829567e-01,
                  4.21607069e-01, 4.63384571e-01, 5.05162073e-01, 5.46939575e-01,
                  5.88717077e-01, 6.30494579e-01, 6.72272082e-01, 7.14049584e-01,
                  7.55827086e-01, 7.97604588e-01, 8.39382090e-01, 8.81159592e-01,
                  9.22937094e-01, 9.64714596e-01, 1.00649210e+00, 1.04826960e+00,
                  1.09004710e+00, 1.13182460e+00, 1.17360211e+00, 1.21537961e+00,
                  1.25715711e+00, 1.29893461e+00, 1.34071212e+00, 1.38248962e+00,
                  1.42426712e+00, 1.46604462e+00, 1.50782212e+00, 1.54959963e+00,
                  1.59137713e+00, 1.63315463e+00, 1.67493213e+00, 1.71670963e+00,
                  1.75848714e+00, 1.80026464e+00, 1.84204214e+00, 1.88381964e+00,
                  1.92559714e+00, 1.96737465e+00, 2.00915215e+00, 2.05092965e+00,
                  2.09270715e+00, 2.13448466e+00, 2.17626216e+00, 2.21803966e+00,
                  2.25981716e+00, 2.30159466e+00, 2.34337217e+00, 2.38514967e+00,
                  2.42692717e+00, 2.46870467e+00, 2.51048217e+00, 2.55225968e+00,
                  2.59403718e+00, 2.63581468e+00, 2.67759218e+00, 2.71936968e+00,
                  2.76114719e+00, 2.80292469e+00, 2.84470219e+00, 2.88647969e+00,
                  2.92825720e+00, 2.97003470e+00, 3.01181220e+00, 3.05358970e+00,
                  3.09536720e+00, 3.13714471e+00, 3.17892221e+00, 3.22069971e+00,
                  3.26247721e+00, 3.30425471e+00, 3.34603222e+00, 3.38780972e+00,
                  3.42958722e+00, 3.47136472e+00, 3.51314222e+00, 3.55491973e+00,

```
       3.59669723e+00,  3.63847473e+00,  3.68025223e+00,  3.72202974e+00,
       3.76380724e+00,  3.80558474e+00,  3.84736224e+00,  3.88913974e+00,
       3.93091725e+00,  3.97269475e+00,  4.01447225e+00,  4.05624975e+00,
       4.09802725e+00,  4.13980476e+00,  4.18158226e+00,  4.22335976e+00,
       4.26513726e+00]),
<a list of 200 Patch objects>)
```



In [42]:
```
G = np.random.rand(2, 3)
```

In [43]:
```
G
```

Out[43]:
```
array([[0.38453861, 0.60299934, 0.52695136],
       [0.53043403, 0.354032  , 0.4453608 ]])
```

In [44]:
```
G.ndim
```

Out[44]:
```
2
```

In [54]:
```
H = np.random.rand(2, 3, 4, 2)
print(H)
```

```
[[[[4.93504351e-01 6.94400759e-01]
   [6.80040183e-01 2.83026163e-01]
   [2.94096631e-01 2.63370756e-01]
   [4.60102429e-01 4.34998005e-01]]

  [[2.70451259e-01 5.16003445e-01]
   [4.16385915e-01 1.18561658e-01]
   [7.17358186e-01 3.51836560e-01]
   [9.71011546e-01 8.54532100e-01]]

  [[5.58678587e-01 5.66918249e-01]
   [3.34746745e-01 1.54939078e-01]
   [6.70903346e-04 1.74018033e-01]
   [4.30127756e-01 7.67229466e-01]]]


 [[[9.20180970e-01 3.60198039e-01]
   [6.40739993e-01 5.69109029e-01]
   [8.78250964e-01 4.59798302e-01]
   [4.54692263e-01 6.46930754e-01]]

  [[2.94783811e-01 6.42706276e-01]
   [4.03828862e-01 9.50846667e-01]
   [4.82340862e-01 5.04715889e-01]
   [6.35203722e-01 3.00903184e-02]]

  [[1.64508338e-01 9.66594648e-01]
   [7.70899537e-01 1.54189228e-01]
   [9.87443312e-02 1.89043840e-01]
   [1.10949270e-01 1.88407117e-01]]]]
```

In [46]:
```
H.ndim
```

Out[46]: 4

In [47]:
```
I = np.arange(100).reshape(4, 25)
```

In [48]:
```
I.shape
```

Out[48]: (4, 25)

In [52]:
```
J = np.arange(100).reshape(4, 5, 5)
print(J)
```

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

 [[25 26 27 28 29]
  [30 31 32 33 34]
  [35 36 37 38 39]
  [40 41 42 43 44]
  [45 46 47 48 49]]

 [[50 51 52 53 54]
  [55 56 57 58 59]
  [60 61 62 63 64]
  [65 66 67 68 69]
  [70 71 72 73 74]]

 [[75 76 77 78 79]
  [80 81 82 83 84]
  [85 86 87 88 89]
  [90 91 92 93 94]
  [95 96 97 98 99]]]
```

In [50]:
```
J.shape
```

Out[50]:
```
(4, 5, 5)
```

In [56]:
```
np.zeros(3)
```

Out[56]:
```
array([0., 0., 0.])
```

In [57]:
```
np.ones(4)
```

Out[57]:
```
array([1., 1., 1., 1.])
```

In [58]:
```
np.eye(5)
```

Out[58]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Slicing and Indexing

A[start:end:step]

a[1:5] index 1 til 5 but not 5

a[:5] index 0 till 5 but not 5

a[2:] index 2 till end including last element

a[::-1] from end till start (reverse the array]

a[::2] from start till end every other element

a[::2,:] ?

A[index_array]

a[[1, 4, 6]] index 1, 4, and 6 elements

a[[True, True, False, False, True, True, True, False]]

Assuming array has 8 elements, the above returns all the elements corresponding to True index

a[a < 8]

a[a < 8 & a > 4] difference between (and, &)?

a[a < 8 and a > 4] ?

In [59]:
```
A = np.arange(100)
```

In [61]:
```
b = A[3:10]
print(b)
```

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

In [62]:
```
b[0] = -1200
```

In [63]:
```
b
```

Out[63]:
```
array([-1200,     4,     5,     6,     7,     8,     9])
```

In [64]:
```
A
```

Out[64]:
```
array([    0,     1,     2, -1200,     4,     5,     6,     7,     8,
           9,    10,    11,    12,    13,    14,    15,    16,    17,
          18,    19,    20,    21,    22,    23,    24,    25,    26,
          27,    28,    29,    30,    31,    32,    33,    34,    35,
          36,    37,    38,    39,    40,    41,    42,    43,    44,
          45,    46,    47,    48,    49,    50,    51,    52,    53,
          54,    55,    56,    57,    58,    59,    60,    61,    62,
          63,    64,    65,    66,    67,    68,    69,    70,    71,
          72,    73,    74,    75,    76,    77,    78,    79,    80,
          81,    82,    83,    84,    85,    86,    87,    88,    89,
          90,    91,    92,    93,    94,    95,    96,    97,    98,
          99])
```

In [65]:
```
# Copy the array to another memory address
b = A[3:10].copy()
```

In [66]:
```
A[::5]
```

Out[66]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
       85, 90, 95])
```

In [67]:
```
A[::-5]
```

Out[67]:
```
array([99, 94, 89, 84, 79, 74, 69, 64, 59, 54, 49, 44, 39, 34, 29, 24, 19,
       14,  9,  4])
```

In [69]:
```
B = (A == -1200)*np.arange(A.size)
```

In [70]:
```
B
```

Out[70]:
```
array([0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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 [71]:
```
A
```

Out[71]:
```
array([    0,     1,     2, -1200,     4,     5,     6,     7,     8,
           9,    10,    11,    12,    13,    14,    15,    16,    17,
          18,    19,    20,    21,    22,    23,    24,    25,    26,
          27,    28,    29,    30,    31,    32,    33,    34,    35,
          36,    37,    38,    39,    40,    41,    42,    43,    44,
          45,    46,    47,    48,    49,    50,    51,    52,    53,
          54,    55,    56,    57,    58,    59,    60,    61,    62,
          63,    64,    65,    66,    67,    68,    69,    70,    71,
          72,    73,    74,    75,    76,    77,    78,    79,    80,
          81,    82,    83,    84,    85,    86,    87,    88,    89,
          90,    91,    92,    93,    94,    95,    96,    97,    98,
          99])
```

In [72]:
```
idx = np.argwhere(A==-1200)[0][0]
print(idx)
```

```
3
```

In [73]:
```
A[idx] = 3
```

In [74]:
```
A
```

Out[74]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

In [76]:
```
A = np.round(10 * np.random.rand(5, 4))
```

In [77]:
```
A
```

Out[77]:
```
array([[10.,  7.,  9.,  7.],
       [ 5.,  0.,  0.,  8.],
       [ 5.,  3.,  6.,  4.],
       [ 1.,  6.,  6.,  9.],
       [ 2.,  7.,  8.,  9.]])
```

In [78]:
```
A[1, 2]
```

Out[78]:
```
0.0
```

In [79]:
```
A[1, :]
```

Out[79]:
```
array([5., 0., 0., 8.])
```

In [80]:
```
A[:, 1]
```

Out[80]:
```
array([7., 0., 3., 6., 7.])
```

In [81]:
```
A[1:3, 2:4]
```

Out[81]:
```
array([[0., 8.],
       [6., 4.]])
```

In [82]:
```
A.T
```

Out[82]:
```
array([[10.,  5.,  5.,  1.,  2.],
       [ 7.,  0.,  3.,  6.,  7.],
       [ 9.,  0.,  6.,  6.,  8.],
       [ 7.,  8.,  4.,  9.,  9.]])
```

In [83]:
```
import numpy.linalg as la
```

In [97]:
```
B = np.round(10 * np.random.rand(2, 2))
```

In [98]:
```
B
```

Out[98]:
```
array([[8., 8.],
       [9., 6.]])
```

In [99]:
```
la.inv(B)
```

Out[99]:
```
array([[-0.25      ,  0.33333333],
       [ 0.375     , -0.33333333]])
```

In [100]:
```
A.sort(axis=0)
```

In [101]:
```
A
```

Out[101]:
```
array([[ 1.,  0.,  0.,  4.],
       [ 2.,  3.,  6.,  7.],
       [ 5.,  6.,  6.,  8.],
       [ 5.,  7.,  8.,  9.],
       [10.,  7.,  9.,  9.]])
```

In [102]:
```
A.sort(axis=1)
```

In [103]:
```
A
```

Out[103]:
```
array([[ 0.,  0.,  1.,  4.],
       [ 2.,  3.,  6.,  7.],
       [ 5.,  6.,  6.,  8.],
       [ 5.,  7.,  8.,  9.],
       [ 7.,  9.,  9., 10.]])
```

In [105]:
```
A[A<5]
```

Out[105]:
```
array([0., 0., 1., 4., 2., 3.])
```

In [106]:
```
A
```

Out[106]:
```
array([[ 0.,  0.,  1.,  4.],
       [ 2.,  3.,  6.,  7.],
       [ 5.,  6.,  6.,  8.],
       [ 5.,  7.,  8.,  9.],
       [ 7.,  9.,  9., 10.]])
```

In [108]:
```
C = np.arange(100)
```

In [109]:
```
D = C[[3, 5, 6]]
```

In [110]:
```
D
```

Out[110]:
```
array([3, 5, 6])
```

In [111]:
```
D[0] = -4
```

In [112]:
```
D
```

Out[112]:
```
array([-4,  5,  6])
```

In [113]:
```
C
```

Out[113]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

In [114]:
```
E = A[A<10]
```

In [115]:
```
E
```

Out[115]:
```
array([0., 0., 1., 4., 2., 3., 6., 7., 5., 6., 6., 8., 5., 7., 8., 9., 7.,
       9., 9.])
```

In [116]:
```
E = A[(A<40) & (A>30)]
```

In [117]:
```
E
```

Out[117]:
```
array([], dtype=float64)
```

In [118]:
```
# &, and
# |, or
# ~, not
```

Broadcasting

A = A + 5

A + [1, 3]

In [119]:
```
A = np.round(10*np.random.rand(2, 3))
```

In [120]:
```
A
```

Out[120]:
```
array([[0., 9., 6.],
       [8., 5., 5.]])
```

In [121]:
```
A + 3
```

Out[121]:
```
array([[ 3., 12.,  9.],
       [11.,  8.,  8.]])
```

In [122]:
```
A + np.arange(2).reshape(2,1)
```

Out[122]:
```
array([[0., 9., 6.],
       [9., 6., 6.]])
```

In [124]:
```
B = np.round(10 * np.random.rand(2, 2))
```

In [125]:
```
B
```

Out[125]:
```
array([[9., 2.],
       [3., 9.]])
```

In [126]:
```
C = np.hstack((A, B))
```

In [127]:
```
C
```

Out[127]:
```
array([[0., 9., 6., 9., 2.],
       [8., 5., 5., 3., 9.]])
```

In [133]:
```
A = np.random.permutation(np.arange(10))
```

In [134]:
```
A
```

Out[134]:
```
array([8, 7, 3, 6, 0, 2, 5, 1, 9, 4])
```

In [135]:
```
A.sort()
```

In [136]:
```
A
```

Out[136]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [137]:
```
np.sort(A)
```

Out[137]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [138]:
```
A = A[::-1]
```

In [139]:
```
A
```

Out[139]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

In [140]:
```
A = np.array(["abc", "hello", "hi"])
```

In [141]:
```
A.sort()  # alphabetic order
```

In [142]:
```
A
```

Out[142]:
```
array(['abc', 'hello', 'hi'], dtype='<U5')
```

Seed: ufuncs

```
b = np.random.rand(1000000)
%timeit sum(b)
%timeit np.sum(b)
%timeit
s = 0
for x in b:
    s += x
```

In [144]:
```
b = np.random.rand(1000000)
%timeit sum(b)
%timeit np.sum(b)
```

```
165 ms ± 11.5 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
1.14 ms ± 30 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [145]:
```
def my_sum(G):
    s = 0
    for x in b:
        s += x
    return s
```

In [147]:
```
%timeit my_sum(b)
```

```
206 ms ± 7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Pandas

In [1]:
```
import pandas as pd
```

In [2]:
```
print(pd.__version__)
```

```
1.0.1
```

In [5]:
```
A = pd.Series([2, 3, 4, 5], index = ['a', 'b', 'c', 'd'])
print(A)
```

```
a    2
b    3
c    4
d    5
dtype: int64
```

In [6]:
```
A.values
```

Out[6]:
```
array([2, 3, 4, 5], dtype=int64)
```

In [7]:
```
type(A.values)
```

Out[7]:
```
numpy.ndarray
```

In [8]:
```
type(A)
```

Out[8]:
```
pandas.core.series.Series
```

In [9]:
```
A.index
```

Out[9]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [11]:
```
A['a']  # like a dictionary
```

Out[11]:
```
2
```

In [12]:
```
A['a': 'c']
```

Out[12]:
```
a    2
b    3
c    4
dtype: int64
```

In [13]:
```
grades_dict = {'A': 4, 'B': 3.5, 'C': 3, 'D': 2.5}
grades = pd.Series(grades_dict)
```

In [14]:
```
grades.values
```

Out[14]:
```
array([4. , 3.5, 3. , 2.5])
```

In [15]:
```
marks_dict = {'A': 85, 'B': 75, 'C': 65, 'D': 55}
marks = pd.Series(marks_dict)
```

In [16]:
```
marks
```

Out[16]:
```
A    85
B    75
C    65
D    55
dtype: int64
```

In [17]:
```
marks['A']
```

Out[17]:
```
85
```

In [18]:
```
marks[0:2]
```

Out[18]:
```
A    85
B    75
dtype: int64
```

In [19]:
```
D = pd.DataFrame({'Marks': marks, 'Grades': grades})
```

In [20]:
```
D
```

Out[20]:

	Marks	Grades
A	85	4.0
B	75	3.5
C	65	3.0
D	55	2.5

In [21]:
```
D.T
```

Out[21]:

	A	B	C	D
Marks	85.0	75.0	65.0	55.0
Grades	4.0	3.5	3.0	2.5

In [22]:
```
D.values
```

Out[22]:
```
array([[85. ,  4. ],
       [75. ,  3.5],
       [65. ,  3. ],
       [55. ,  2.5]])
```

In [23]:
```
D.values[2, 0]
```

Out[23]:
```
65.0
```

In [24]:
```
D.columns
```

Out[24]:
```
Index(['Marks', 'Grades'], dtype='object')
```

In [25]:
```
D['ScaledMarks'] = 100 * D['Marks']/90
```

In [26]:
```
D
```

Out[26]:

	Marks	Grades	ScaledMarks
A	85	4.0	94.444444
B	75	3.5	83.333333
C	65	3.0	72.222222
D	55	2.5	61.111111

In [27]:
```
del D['ScaledMarks']
```

In [28]:
```
D
```

Out[28]:

	Marks	Grades
A	85	4.0
B	75	3.5
C	65	3.0
D	55	2.5

In [29]:
```
G = D[D['Marks']>70]
```

In [30]:
```
G
```

Out[30]:

	Marks	Grades
A	85	4.0
B	75	3.5

In [33]:
```
A = pd.DataFrame([{'a': 1, 'b': 4}, {'b': -3, 'c': 9}])
```

In [34]:
```
A
```

Out[34]:

	a	b	c
0	1.0	4	NaN
1	NaN	-3	9.0

In [35]:
```
A.fillna(0)
```

Out[35]:

	a	b	c
0	1.0	4	0.0
1	0.0	-3	9.0

In [36]:
```
A.dropna?
```

In [37]:
```
A = pd.Series(['a', 'b', 'c'], index=[1,3,5])
```

In [38]:
```
A[1]
```

Out[38]:
```
'a'
```

In [39]:
```
A[1:3]
```

Out[39]:
```
3    b
5    c
dtype: object
```

In [40]:
```
A.loc[1:3]
```

Out[40]:
```
1    a
3    b
dtype: object
```

In [41]:
```
A.iloc[1:3]
```

Out[41]:
```
3    b
5    c
dtype: object
```

In [42]:
```
D.iloc[2,:]
```

Out[42]:
```
Marks     65.0
Grades     3.0
Name: C, dtype: float64
```

In [44]:
```
D.iloc[::-1]
```

Out[44]:

	Marks	Grades
D	55	2.5
C	65	3.0
B	75	3.5
A	85	4.0

In [46]:
```
from sklearn.impute import SimpleImputer
```

In [47]:
```
df = pd.read_csv('D:/Programming/Jupyter/Hello/data/covid_19_data.csv')
```

In [48]:
```
df.head()
```

Out[48]:

	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	1/31/20 14:00	5806	204.0	141.0
1	Zhejiang	Mainland China	1/31/20 14:00	538	NaN	14.0
2	Guangdong	Mainland China	1/31/20 14:00	436	NaN	11.0
3	Henan	Mainland China	1/31/20 14:00	352	2.0	3.0
4	Hunan	Mainland China	1/31/20 14:00	332	NaN	2.0

In [49]:
```
df.head(10)
```

Out[49]:

	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	1/31/20 14:00	5806	204.0	141.0
1	Zhejiang	Mainland China	1/31/20 14:00	538	NaN	14.0
2	Guangdong	Mainland China	1/31/20 14:00	436	NaN	11.0
3	Henan	Mainland China	1/31/20 14:00	352	2.0	3.0
4	Hunan	Mainland China	1/31/20 14:00	332	NaN	2.0
5	Jiangxi	Mainland China	1/31/20 14:00	240	NaN	7.0
6	Anhui	Mainland China	1/31/20 14:00	237	NaN	3.0
7	Chongqing	Mainland China	1/31/20 14:00	211	NaN	1.0
8	Shandong	Mainland China	1/31/20 14:00	184	NaN	2.0
9	Sichuan	Mainland China	1/31/20 14:00	177	1.0	1.0

In [52]:
```
df.drop(['Last Update'], axis=1, inplace=True)  # drop columns
```

In [54]:
```
df.head(10)
```

Out[54]:

	Province/State	Country/Region	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204.0	141.0
1	Zhejiang	Mainland China	538	NaN	14.0
2	Guangdong	Mainland China	436	NaN	11.0
3	Henan	Mainland China	352	2.0	3.0
4	Hunan	Mainland China	332	NaN	2.0
5	Jiangxi	Mainland China	240	NaN	7.0
6	Anhui	Mainland China	237	NaN	3.0
7	Chongqing	Mainland China	211	NaN	1.0
8	Shandong	Mainland China	184	NaN	2.0
9	Sichuan	Mainland China	177	1.0	1.0

In [81]:
```
df.rename(columns={'Province/State': 'Province', 'Country/Region': 'Country'}, inplac
e=True)
```

In [82]:
```
df.head()
```

Out[82]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204.0	141.0
1	Zhejiang	Mainland China	538	NaN	14.0
2	Guangdong	Mainland China	436	NaN	11.0
3	Henan	Mainland China	352	2.0	3.0
4	Hunan	Mainland China	332	NaN	2.0

In [83]:
```
# date formar
# df['Date'] = pd.to_datetime(df['Date'])
```

In [84]:
```
df.describe()
```

Out[84]:

	Confirmed	Deaths	Recovered
count	10.000000	3.000000	10.000000
mean	851.300000	69.000000	18.500000
std	1744.822885	116.914499	43.272393
min	177.000000	1.000000	1.000000
25%	217.500000	1.500000	2.000000
50%	286.000000	2.000000	3.000000
75%	415.000000	103.000000	10.000000
max	5806.000000	204.000000	141.000000

In [96]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Province   10 non-null     object 
 1   Country    10 non-null     object 
 2   Confirmed  10 non-null     int64  
 3   Deaths     3 non-null      float64
 4   Recovered  10 non-null     float64
dtypes: float64(2), int64(1), object(2)
memory usage: 528.0+ bytes
```

In [108]:
```
df.head()
```

Out[108]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204.0	141.0
1	Zhejiang	Mainland China	538	NaN	14.0
2	Guangdong	Mainland China	436	NaN	11.0
3	Henan	Mainland China	352	2.0	3.0
4	Hunan	Mainland China	332	NaN	2.0

In [103]:
```
df.fillna('99999')
```

Out[103]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204	141.0
1	Zhejiang	Mainland China	538	99999	14.0
2	Guangdong	Mainland China	436	99999	11.0
3	Henan	Mainland China	352	2	3.0
4	Hunan	Mainland China	332	99999	2.0
5	Jiangxi	Mainland China	240	99999	7.0
6	Anhui	Mainland China	237	99999	3.0
7	Chongqing	Mainland China	211	99999	1.0
8	Shandong	Mainland China	184	99999	2.0
9	Sichuan	Mainland China	177	1	1.0

In [104]:
```
imputer = SimpleImputer(strategy='constant')
```

In [105]:
```
df2 = pd.DataFrame(imputer.fit_transform(df),columns=df.columns)
```

In [106]:
```
df2
```

Out[106]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204	141
1	Zhejiang	Mainland China	538	missing_value	14
2	Guangdong	Mainland China	436	missing_value	11
3	Henan	Mainland China	352	2	3
4	Hunan	Mainland China	332	missing_value	2
5	Jiangxi	Mainland China	240	missing_value	7
6	Anhui	Mainland China	237	missing_value	3
7	Chongqing	Mainland China	211	missing_value	1
8	Shandong	Mainland China	184	missing_value	2
9	Sichuan	Mainland China	177	1	1

In [107]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Province   10 non-null     object
 1   Country    10 non-null     object
 2   Confirmed  10 non-null     object
 3   Deaths     10 non-null     object
 4   Recovered  10 non-null     object
dtypes: object(5)
memory usage: 528.0+ bytes
```

In [101]:
```
df2 = df.groupby('Country')[['Country', 'Confirmed', 'Deaths', 'Recovered']].sum().reset_index()
```

In [111]:
```
df2  # Should add more data
```

Out[111]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204	141
1	Zhejiang	Mainland China	538	missing_value	14
2	Guangdong	Mainland China	436	missing_value	11
3	Henan	Mainland China	352	2	3
4	Hunan	Mainland China	332	missing_value	2
5	Jiangxi	Mainland China	240	missing_value	7
6	Anhui	Mainland China	237	missing_value	3
7	Chongqing	Mainland China	211	missing_value	1
8	Shandong	Mainland China	184	missing_value	2
9	Sichuan	Mainland China	177	1	1

In [112]:
```
df3 = df2[df2['Confirmed']>100]
```

In [113]:
```
df3
```

Out[113]:

	Province	Country	Confirmed	Deaths	Recovered
0	Hubei	Mainland China	5806	204	141
1	Zhejiang	Mainland China	538	missing_value	14
2	Guangdong	Mainland China	436	missing_value	11
3	Henan	Mainland China	352	2	3
4	Hunan	Mainland China	332	missing_value	2
5	Jiangxi	Mainland China	240	missing_value	7
6	Anhui	Mainland China	237	missing_value	3
7	Chongqing	Mainland China	211	missing_value	1
8	Shandong	Mainland China	184	missing_value	2
9	Sichuan	Mainland China	177	1	1

In [ ]:
```

```