#REFERENCE

https://github.com/ageron/handson-ml2/blob/master/tools_numpy.ipynb
(https://github.com/ageron/handson-ml2/blob/master/tools_numpy.ipynb)

In [1]:
```
import numpy as np
```

In [2]:
```
np.zeros(5)
```

Out[2]:
```
array([0., 0., 0., 0., 0.])
```

In [3]:
```
np.zeros((3,4))
```

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

In [4]:
```
a = np.zeros((3,4))
a
```

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

In [5]:
```
a.shape
```

Out[5]:
```
(3, 4)
```

In [6]:
```
a.shape[0]
```

Out[6]:
```
3
```

In [7]:
```
a.ndim  # equal to len(a.shape)
```

Out[7]:
```
2
```

In [8]:
```
a.size
```

Out[8]:
```
12
```

In [9]:
```
np.zeros((2,3,4))
```

Out[9]:
```
array([[[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 [10]:
```
1 type(np.zeros((3,4)))
```

Out[10]:
```
numpy.ndarray
```

In [11]:
```
1 np.ones((3,4))
```

Out[11]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [12]:
```
1 np.full((3,4), np.pi)
```

Out[12]:
```
array([[3.14159265, 3.14159265, 3.14159265, 3.14159265],
       [3.14159265, 3.14159265, 3.14159265, 3.14159265],
       [3.14159265, 3.14159265, 3.14159265, 3.14159265]])
```

In [13]:
```
1 np.empty((2,3))
```

Out[13]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [14]:
```
1 np.array([[1,2,3,4], [10, 20, 30, 40]])
```

Out[14]:
```
array([[ 1,  2,  3,  4],
       [10, 20, 30, 40]])
```

In [15]:
```
1 np.arange(1, 5)
```

Out[15]:
```
array([1, 2, 3, 4])
```

In [16]:
```
1 np.arange(1.0, 5.0)
```

Out[16]:
```
array([1., 2., 3., 4.])
```

In [17]:
```
1 np.arange(1, 5, 0.5)
```

Out[17]:
```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

In [18]:
```
1 print(np.arange(0, 5/3, 1/3)) # depending on floating point errors, the max
2 print(np.arange(0, 5/3, 0.333333333))
3 print(np.arange(0, 5/3, 0.333333334))
```

```
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667]
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667]
[0.         0.33333333 0.66666667 1.         1.33333334]
```

In [19]:
```
1 print(np.linspace(0, 5/3, 6))
```

```
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667]
```

In [20]:
```
1 np.random.rand(3,4)
```

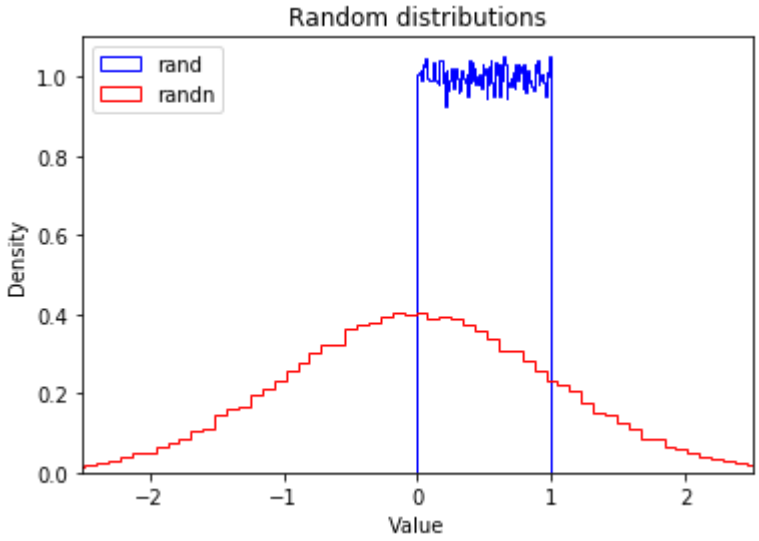
Out[20]:
```
array([[0.84625688, 0.4259509 , 0.52547951, 0.82440238],
       [0.36892663, 0.08427768, 0.95484925, 0.6663687 ],
       [0.0643052 , 0.16238608, 0.1176385 , 0.65327855]])
```

In [21]:
```
1 np.random.randn(3,4)
```

Out[21]:
```
array([[ 0.41190254,  0.44254128,  0.23678673,  1.09468869],
       [ 1.53545807, -1.0029811 ,  0.13970212,  2.08104554],
       [-0.68317997,  1.1570703 ,  1.10464409, -1.10421329]])
```

In [22]:
```
1 %matplotlib inline
2 import matplotlib.pyplot as plt
```

In [23]:
```
1 plt.hist(np.random.rand(100000), density=True, bins=100, histtype="step", co
2 plt.hist(np.random.randn(100000), density=True, bins=100, histtype="step", c
3 plt.axis([-2.5, 2.5, 0, 1.1])
4 plt.legend(loc = "upper left")
5 plt.title("Random distributions")
6 plt.xlabel("Value")
7 plt.ylabel("Density")
8 plt.show()
```



In [24]:
```
def my_function(z, y, x):
    return x * y + z

np.fromfunction(my_function, (3, 2, 10))
```

Out[24]:
```
array([[[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.]],

       [[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]],

       [[ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.],
        [ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.]]])
```

In [25]:
```
c = np.arange(1, 5)
print(c.dtype, c)
```

```
int32 [1 2 3 4]
```

In [26]:
```
c = np.arange(1.0, 5.0)
print(c.dtype, c)
```

```
float64 [1. 2. 3. 4.]
```

In [27]:
```
d = np.arange(1, 5, dtype=np.complex64)
print(d.dtype, d)
```

```
complex64 [1.+0.j 2.+0.j 3.+0.j 4.+0.j]
```

In [28]:
```
e = np.arange(1, 5, dtype=np.complex64)
e.itemsize
```

Out[28]:
```
8
```

In [29]:
```
f = np.array([[1,2],[1000, 2000]], dtype=np.int32)
f.data
```

Out[29]:
```
<memory at 0x000002608AE0C668>
```

In [30]:
```
if (hasattr(f.data, "tobytes")):
    data_bytes = f.data.tobytes() # python 3
else:
    data_bytes = memoryview(f.data).tobytes() # python 2

data_bytes
```

Out[30]:
```
b'\x01\x00\x00\x00\x02\x00\x00\x00\xe8\x03\x00\x00\xd0\x07\x00\x00'
```

In [31]:
```
g = np.arange(24)
print(g)
print("Rank:", g.ndim)
```

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

In [32]:
```
g.shape = (6, 4)
print(g)
print("Rank:", g.ndim)
```

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

In [33]:
```
g.shape = (2, 3, 4)
print(g)
print("Rank:", g.ndim)
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
Rank: 3
```

In [34]:
```
g2 = g.reshape(4,6)
print(g2)
print("Rank:", g2.ndim)
```

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

In [35]:
```
g2[1, 2] = 999 # [row,column]
g2
```

Out[35]:
```
array([[  0,   1,   2,   3,   4,   5],
       [  6,   7, 999,   9,  10,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23]])
```

In [36]:
```
g
```

Out[36]:
```
array([[[  0,   1,   2,   3],
        [  4,   5,   6,   7],
        [999,   9,  10,  11]],

       [[ 12,  13,  14,  15],
        [ 16,  17,  18,  19],
        [ 20,  21,  22,  23]]])
```

In [37]:
```
g.ravel()
```

Out[37]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7, 999,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23])
```

In [38]:
```
a = np.array([14, 23, 32, 41])
b = np.array([5,  4,  3,  2])
print("a + b  =", a + b)
print("a - b  =", a - b)
print("a * b  =", a * b)
print("a / b  =", a / b)
print("a // b  =", a // b)
print("a % b  =", a % b)
print("a ** b =", a ** b)
```

```
a + b  = [19 27 35 43]
a - b  = [ 9 19 29 39]
a * b  = [70 92 96 82]
a / b  = [ 2.8         5.75        10.66666667 20.5       ]
a // b  = [ 2  5 10 20]
a % b  = [4 3 2 1]
a ** b = [537824 279841  32768   1681]
```

Broadcasting

In general, when NumPy expects arrays of the same shape but finds that this is not the case, it applies the so-called broadcasting rules:

In [39]:
```
h = np.arange(5).reshape(1, 1, 5)
h
```

Out[39]:
```
array([[[0, 1, 2, 3, 4]]])
```

In [40]:
```
h + [10, 20, 30, 40, 50]  # same as: h + [[[10, 20, 30, 40, 50]]]
```

Out[40]:
```
array([[[10, 21, 32, 43, 54]]])
```

In [41]:
```
k = np.arange(6).reshape(2, 3)
k
```

Out[41]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [42]:
```
k + [[100], [200]]  # same as: k + [[100, 100, 100], [200, 200, 200]]
```

Out[42]:
```
array([[100, 101, 102],
       [203, 204, 205]])
```

In [43]:
```
k + [100, 200, 300]  # after rule 1: [[100, 200, 300]], and after rule 2: [[
```

Out[43]:
```
array([[100, 201, 302],
       [103, 204, 305]])
```

In [44]:
```
k + 1000  # same as: k + [[1000, 1000, 1000], [1000, 1000, 1000]]
```

Out[44]:
```
array([[1000, 1001, 1002],
       [1003, 1004, 1005]])
```

In [45]:
```
try:
    k + [33, 44]
except ValueError as e:
    print(e)
```

```
operands could not be broadcast together with shapes (2,3) (2,)
```

In [46]:
```
k1 = np.arange(0, 5, dtype=np.uint8)
print(k1.dtype, k1)
```

```
uint8 [0 1 2 3 4]
```

In [47]:
```
k2 = k1 + np.array([5, 6, 7, 8, 9], dtype=np.int8)
print(k2.dtype, k2)
```

```
int16 [ 5  7  9 11 13]
```

In [48]:
```
k3 = k1 + 1.5
print(k3.dtype, k3)
```

```
float64 [1.5 2.5 3.5 4.5 5.5]
```

In [49]:
```
m = np.array([20, -5, 30, 40])
m < [15, 16, 35, 36]
```

Out[49]:
```
array([False,  True,  True, False])
```

In [50]:
```
m < 25  # equivalent to m < [25, 25, 25, 25]
```

Out[50]:
```
array([ True,  True, False, False])
```

In [51]:
```
m[m < 25]
```

Out[51]:
```
array([20, -5])
```

In [52]:
```
a = np.array([[-2.5, 3.1, 7], [10, 11, 12]])
print(a)
print("mean =", a.mean())
```

```
[[-2.5  3.1  7. ]
 [10.  11.  12. ]]
mean = 6.766666666666667
```

In [53]:
```
for func in (a.min, a.max, a.sum, a.prod, a.std, a.var):
    print(func.__name__, "=", func())
```

```
min = -2.5
max = 12.0
sum = 40.6
prod = -71610.0
std = 5.084835843520964
var = 25.855555555555554
```

In [54]:
```
c=np.arange(24).reshape(2,3,4)
c
```

Out[54]:
```
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]]])
```

In [55]:
```
c.sum(axis=0)  # sum across matrices
```

Out[55]:
```
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])
```

In [56]:
```
c.sum(axis=1)  # sum across rows
#[0+4+8,1+5+9,2+6+10,3+7+11],[12+16+28,14+22+30,16+24+32,18+26+34]
```

Out[56]:
```
array([[12, 15, 18, 21],
       [48, 51, 54, 57]])
```

In [57]:
```
[0+4+8,1+5+9,2+6+10,3+7+11],[12+16+20,13+17+21,14+18+22,15+19+23]
```

Out[57]:
```
([12, 15, 18, 21], [48, 51, 54, 57])
```

In [58]:
```
c.sum(axis=(0,2))  # sum across matrices and columns
```

Out[58]:
```
array([ 60,  92, 124])
```

In [59]:
```
0+1+2+3 + 12+13+14+15, 4+5+6+7 + 16+17+18+19, 8+9+10+11 + 20+21+22+23
```

Out[59]:
```
(60, 92, 124)
```

In [60]:
```
a = np.array([[-2.5, 3.1, 7], [10, 11, 12]])
np.square(a)
```

Out[60]:
```
array([[  6.25,   9.61,  49.  ],
       [100.  , 121.  , 144.  ]])
```

In [61]:
```
print("Original ndarray")
print(a)
for func in (np.abs, np.sqrt, np.exp, np.log, np.sign, np.ceil, np.modf, np.
    print("\n", func.__name__)
    print(func(a))
```

```
Original ndarray
[[-2.5  3.1  7. ]
 [10.  11.  12. ]]

 absolute
[[ 2.5  3.1  7. ]
 [10.  11.  12. ]]

 sqrt
[[       nan 1.76068169 2.64575131]
 [3.16227766 3.31662479 3.46410162]]

 exp
[[8.20849986e-02 2.21979513e+01 1.09663316e+03]
 [2.20264658e+04 5.98741417e+04 1.62754791e+05]]

 log
[[       nan 1.13140211 1.94591015]
 [2.30258509 2.39789527 2.48490665]]

 sign
[[-1.  1.  1.]
 [ 1.  1.  1.]]

 ceil
[[-2.  4.  7.]
 [10. 11. 12.]]

 modf
(array([[-0.5,  0.1,  0. ],
       [ 0. ,  0. ,  0. ]]), array([[-2.,  3.,  7.],
       [10., 11., 12.]]))

 isnan
[[False False False]
 [False False False]]

 cos
[[-0.80114362 -0.99913515  0.75390225]
 [-0.83907153  0.0044257   0.84385396]]

D:\anaconda\lib\site-packages\ipykernel_launcher.py:5: RuntimeWarning: invalid
value encountered in sqrt
  """
D:\anaconda\lib\site-packages\ipykernel_launcher.py:5: RuntimeWarning: invalid
value encountered in log
  """
```

In [62]:
```
a = np.array([1, -2, 3, 4])
b = np.array([2, 8, -1, 7])
np.add(a, b)  # equivalent to a + b
```

Out[62]:
```
array([ 3,  6,  2, 11])
```

In [63]:
```
np.greater(a, b)  # equivalent to a > b
```

Out[63]:
```
array([False, False,  True, False])
```

In [64]:
```
np.maximum(a, b)
```

Out[64]:
```
array([2, 8, 3, 7])
```

In [65]:
```
np.copysign(a, b)
```

Out[65]:
```
array([ 1.,  2., -3.,  4.])
```

In [66]:
```
a = np.array([1, 5, 3, 19, 13, 7, 3])
a[3]
```

Out[66]:
```
19
```

In [67]:
```
a[2:5]
```

Out[67]:
```
array([ 3, 19, 13])
```

In [68]:
```
a[2:-1]
```

Out[68]:
```
array([ 3, 19, 13,  7])
```

In [69]:
```
a[:2]
```

Out[69]:
```
array([1, 5])
```

In [70]:
```
a[2::1]#skip 1-1=0 from indesx 2
```

Out[70]:
```
array([ 3, 19, 13,  7,  3])
```

In [71]:
```
a[2::2]#skip 2-1=1 from indesx 2
```

Out[71]:
```
array([ 3, 13,  3])
```

In [72]:
```
a[2::3]#skip 3-1=2 from indesx 2
```

Out[72]:
```
array([3, 7])
```

In [73]:
```
a[2::-1]#skip 1-1=0 from indesx 2(reverse order)
```

Out[73]: array([3, 5, 1])

In [74]:
```
a[4::-2]#skip 2-1=1 from indesx 4(reverse order)
```

Out[74]: array([13,  3,  1])

In [75]:
```
a
```

Out[75]: array([ 1,  5,  3, 19, 13,  7,  3])

In [76]:
```
a[3]=999
a
```

Out[76]: array([  1,   5,   3, 999,  13,   7,   3])

In [77]:
```
a[2:5] = [997, 998, 999]
a
```

Out[77]: array([  1,   5, 997, 998, 999,   7,   3])

In [78]:
```
a[2:5] = -1
a
```

Out[78]: array([ 1,  5, -1, -1, -1,  7,  3])

In [79]:
```
try:
    a[2:5] = [1,2,3,4,5,6]  # too long
except ValueError as e:
    print(e)
```

```
cannot copy sequence with size 6 to array axis with dimension 3
```

In [80]:
```
try:
    del a[2:5]
except ValueError as e:
    print(e)
```

```
cannot delete array elements
```

In [81]:
```
a_slice = a[2:6]
a_slice[1] = 1000
a  # the original array was modified!
```

Out[81]: array([   1,    5,   -1, 1000,   -1,    7,    3])

In [82]:
```
a[3] = 2000
a_slice  # similarly, modifying the original array modifies the slice!
```

Out[82]: array([  -1, 2000,   -1,    7])

In [83]:
```
another_slice = a[2:6].copy()
another_slice[1] = 3000
a  # the original array is untouched
```

Out[83]:
```
array([   1,    5,   -1, 2000,   -1,    7,    3])
```

In [84]:
```
a[3] = 4000
another_slice  # similary, modifying the original array does not affect the
```

Out[84]:
```
array([  -1, 3000,   -1,    7])
```

In [85]:
```
a
```

Out[85]:
```
array([   1,    5,   -1, 4000,   -1,    7,    3])
```

In [86]:
```
b = np.arange(48).reshape(4, 12)
b
```

Out[86]:
```
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]])
```

In [87]:
```
b[1, 2]  # row 1, col 2
```

Out[87]:
```
14
```

In [88]:
```
b[1, :]  # row 1, all columns
```

Out[88]:
```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

In [89]:
```
b[:, 1]  # all rows, column 1
```

Out[89]:
```
array([ 1, 13, 25, 37])
```

In [90]:
```
b[1, :]
```

Out[90]:
```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

In [91]:
```
b[1:2, :] #The first expression returns row 1 as a 1D array of shape (12,),
          #while the second returns that same row as a 2D array of shape (1,
```

Out[91]:
```
array([[12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]])
```

In [92]:
```
b
```

Out[92]:
```
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]])
```

In [93]:
```
b[(0,2), 2:5]  # rows 0 and 2, columns 2 to 4 (5-1)
```

Out[93]:
```
array([[ 2,  3,  4],
       [26, 27, 28]])
```

In [94]:
```
b[:, (-1, 2, -1)]  # all rows, columns -1 (last), 2 and -1 (again, and in th
```

Out[94]:
```
array([[11,  2, 11],
       [23, 14, 23],
       [35, 26, 35],
       [47, 38, 47]])
```

In [95]:
```
b[(-1, 2, -1, 2), (5, 9, 1, 9)]  # returns a 1D array with b[-1, 5], b[2, 9]
```

Out[95]:
```
array([41, 33, 37, 33])
```

In [96]:
```
c = b.reshape(4,2,6)
c
```

Out[96]:
```
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]]])
```

In [97]:
```
c[2, 1, 4]  # matrix 2, row 1, col 4
```

Out[97]:
```
34
```

In [98]:
```
c[2, :, 3]  # matrix 2, all rows, col 3
```

Out[98]:
```
array([27, 33])
```

In [99]:
```
c[2, 1]  # Return matrix 2, row 1, all columns.  This is equivalent to c[2,
```

Out[99]:
```
array([30, 31, 32, 33, 34, 35])
```

In [100]:
```
c[2, ...]  #  matrix 2, all rows, all columns.  This is equivalent to c[2, :
```

Out[100]:
```
array([[24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

In [101]:
```
c[2, 1, ...]  # matrix 2, row 1, all columns.  This is equivalent to c[2, 1,
```

Out[101]:
```
array([30, 31, 32, 33, 34, 35])
```

In [102]:
```
c[2, ..., 3]  # matrix 2, all rows, column 3.  This is equivalent to c[2, :,
```

Out[102]:
```
array([27, 33])
```

In [103]:
```
c[..., 3]  # all matrices, all rows, column 3.  This is equivalent to c[:, :
```

Out[103]:
```
array([[ 3,  9],
       [15, 21],
       [27, 33],
       [39, 45]])
```

In [104]:
```
b = np.arange(48).reshape(4, 12)
b
```

Out[104]:
```
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]])
```

In [105]:
```
rows_on = np.array([True, False, True, False])
b[rows_on, :]  # Rows 0 and 2, all columns. Equivalent to b[(0, 2), :]
```

Out[105]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]])
```

In [106]:
```
cols_on = np.array([False, True, False] * 4)
b[:, cols_on]  # All rows, columns 1, 4, 7 and 10
```

Out[106]:
```
array([[ 1,  4,  7, 10],
       [13, 16, 19, 22],
       [25, 28, 31, 34],
       [37, 40, 43, 46]])
```

In [107]:
```
[False, True, False] * 4
```

Out[107]:
```
[False,
 True,
 False,
 False,
 True,
 False,
 False,
 True,
 False,
 False,
 True,
 False]
```

In [108]:
```
b[np.ix_(rows_on, cols_on)]
```

Out[108]:
```
array([[ 1,  4,  7, 10],
       [25, 28, 31, 34]])
```

In [109]:
```
np.ix_(rows_on, cols_on)
```

Out[109]:
```
(array([[0],
        [2]], dtype=int64), array([[ 1,  4,  7, 10]], dtype=int64))
```

In [110]:
```
b
```

Out[110]:
```
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]])
```

In [111]:
```
b[b % 3 == 1]
```

Out[111]:
```
array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46])
```

In [112]:
```
c = np.arange(24).reshape(2, 3, 4)  # A 3D array (composed of two 3x4 matric
c
```

Out[112]:
```
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]]])
```

In [113]:
```
for m in c:
    print("Item:")
    print(m)
```

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

In [114]:
```
for i in range(len(c)):  # Note that len(c) == c.shape[0]
    print("Item:")
    print(c[i])
```

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

In [115]:
```
for i in c.flat:
    print("Item:", i)
```

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

In [116]:
```
q1 = np.full((3,4), 1.0)
q1
```

Out[116]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [117]:
```
q2 = np.full((4,4), 2.0)
q2
```

Out[117]:
```
array([[2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.]])
```

In [118]:
```
q3 = np.full((3,4), 3.0)
q3
```

Out[118]:
```
array([[3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

In [119]:
```
q4 = np.vstack((q1, q2, q3))
q4
```

Out[119]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

In [120]:
```
q4.shape
```

Out[120]:
```
(10, 4)
```

In [121]:
```
q5 = np.hstack((q1, q3))
q5
```

Out[121]:
```
array([[1., 1., 1., 1., 3., 3., 3., 3.],
       [1., 1., 1., 1., 3., 3., 3., 3.],
       [1., 1., 1., 1., 3., 3., 3., 3.]])
```

In [122]:
```
q5.shape
```

Out[122]:
```
(3, 8)
```

In [123]:
```
try:
    q5 = np.hstack((q1, q2, q3))
except ValueError as e:
    print(e)
```

```
all the input array dimensions except for the concatenation axis must match exa
ctly
```

In [124]:
```
q7 = np.concatenate((q1, q2, q3), axis=0)  # Equivalent to vstack
q7
```

Out[124]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]])
```

In [125]:
```
q7.shape
```

Out[125]:
```
(10, 4)
```

In [126]:
```
q8 = np.stack((q1, q3))
q8
```

Out[126]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[3., 3., 3., 3.],
        [3., 3., 3., 3.],
        [3., 3., 3., 3.]]])
```

In [127]:
```
q8.shape
```

Out[127]:
```
(2, 3, 4)
```

In [128]:
```
r = np.arange(24).reshape(6,4)
r
```

Out[128]:
```
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]])
```

In [129]:
```
r1, r2, r3 = np.vsplit(r, 3)
r1
```

Out[129]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [130]:
```
1 r2
```

Out[130]:
```
array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [131]:
```
1 r3
```

Out[131]:
```
array([[16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [132]:
```
1 r4, r5 = np.hsplit(r, 2)
2 r4
```

Out[132]:
```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13],
       [16, 17],
       [20, 21]])
```

In [133]:
```
1 r5
```

Out[133]:
```
array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15],
       [18, 19],
       [22, 23]])
```

In [134]:
```
1 t = np.arange(24).reshape(4,2,3)
2 t
```

Out[134]:
```
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]]])
```

In [135]:
```
1 t.shape
```

Out[135]:
```
(4, 2, 3)
```

In [136]:
```
t1 = t.transpose((1,2,0))
t1
```

Out[136]:
```
array([[[ 0,  6, 12, 18],
        [ 1,  7, 13, 19],
        [ 2,  8, 14, 20]],

       [[ 3,  9, 15, 21],
        [ 4, 10, 16, 22],
        [ 5, 11, 17, 23]]])
```

In [137]:
```
t1.shape
```

Out[137]:
```
(2, 3, 4)
```

In [138]:
```
t2 = t.transpose()  # equivalent to t.transpose((2, 1, 0))
t2
```

Out[138]:
```
array([[[ 0,  6, 12, 18],
        [ 3,  9, 15, 21]],

       [[ 1,  7, 13, 19],
        [ 4, 10, 16, 22]],

       [[ 2,  8, 14, 20],
        [ 5, 11, 17, 23]]])
```

In [139]:
```
t2.shape
```

Out[139]:
```
(3, 2, 4)
```

In [140]:
```
t3 = t.swapaxes(0,1)  # equivalent to t.transpose((1, 0, 2))
t3
```

Out[140]:
```
array([[[ 0,  1,  2],
        [ 6,  7,  8],
        [12, 13, 14],
        [18, 19, 20]],

       [[ 3,  4,  5],
        [ 9, 10, 11],
        [15, 16, 17],
        [21, 22, 23]]])
```

In [141]:
```
t3.shape
```

Out[141]:
```
(2, 4, 3)
```

In [142]:
```
1 m1 = np.arange(10).reshape(2,5)
2 m1
```

Out[142]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

In [143]:
```
1 m1.T
```

Out[143]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

In [144]:
```
1 m2 = np.arange(5)
2 m2
```

Out[144]:
```
array([0, 1, 2, 3, 4])
```

In [145]:
```
1 m2.T
```

Out[145]:
```
array([0, 1, 2, 3, 4])
```

In [146]:
```
1 m2r = m2.reshape(1,5)
2 m2r
```

Out[146]:
```
array([[0, 1, 2, 3, 4]])
```

In [147]:
```
1 m2r.T
```

Out[147]:
```
array([[0],
       [1],
       [2],
       [3],
       [4]])
```

In [148]:
```
1 n1 = np.arange(10).reshape(2, 5)
2 n1
```

Out[148]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

In [149]:
```
1 n2 = np.arange(15).reshape(5,3)
2 n2
```

Out[149]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

In [150]:
```
n1.dot(n2)
```

Out[150]:
```
array([[ 90, 100, 110],
       [240, 275, 310]])
```

In [151]:
```
import numpy.linalg as linalg

m3 = np.array([[1,2,3],[5,7,11],[21,29,31]])
m3
```

Out[151]:
```
array([[ 1,  2,  3],
       [ 5,  7, 11],
       [21, 29, 31]])
```

In [152]:
```
linalg.inv(m3)
```

Out[152]:
```
array([[-2.31818182,  0.56818182,  0.02272727],
       [ 1.72727273, -0.72727273,  0.09090909],
       [-0.04545455,  0.29545455, -0.06818182]])
```

In [153]:
```
linalg.pinv(m3) # both are same inv or pinv
```

Out[153]:
```
array([[-2.31818182,  0.56818182,  0.02272727],
       [ 1.72727273, -0.72727273,  0.09090909],
       [-0.04545455,  0.29545455, -0.06818182]])
```

In [154]:
```
m3.dot(linalg.inv(m3))
```

Out[154]:
```
array([[ 1.00000000e+00, -1.11022302e-16,  0.00000000e+00],
       [-1.33226763e-15,  1.00000000e+00, -1.11022302e-16],
       [ 2.88657986e-15,  0.00000000e+00,  1.00000000e+00]])
```

In [155]:
```
np.eye(3)
```

Out[155]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [156]:
```
m3
```

Out[156]:
```
array([[ 1,  2,  3],
       [ 5,  7, 11],
       [21, 29, 31]])
```

In [157]:
```
q, r = linalg.qr(m3)
q
```

Out[157]:
```
array([[-0.04627448,  0.98786672,  0.14824986],
       [-0.23137241,  0.13377362, -0.96362411],
       [-0.97176411, -0.07889213,  0.22237479]])
```

In [158]:
```
1 r
```

Out[158]:
```
array([[-21.61018278, -29.89331494, -32.80860727],
       [  0.        ,   0.62427688,   1.9894538 ],
       [  0.        ,   0.        ,  -3.26149699]])
```

In [159]:
```
1 q.dot(r)  # q.r equals m3
```

Out[159]:
```
array([[ 1.,  2.,  3.],
       [ 5.,  7., 11.],
       [21., 29., 31.]])
```

In [160]:
```
1 linalg.det(m3)  # Computes the matrix determinant
```

Out[160]:
```
43.99999999999997
```

In [161]:
```
1 m3
```

Out[161]:
```
array([[ 1,  2,  3],
       [ 5,  7, 11],
       [21, 29, 31]])
```

In [162]:
```
1 eigenvalues, eigenvectors = linalg.eig(m3)
2 eigenvalues # λ
```

Out[162]:
```
array([42.26600592, -0.35798416, -2.90802176])
```

In [163]:
```
1 eigenvectors # v
```

Out[163]:
```
array([[-0.08381182, -0.76283526, -0.18913107],
       [-0.3075286 ,  0.64133975, -0.6853186 ],
       [-0.94784057, -0.08225377,  0.70325518]])
```

In [164]:
```
1 m3.dot(eigenvectors) - eigenvalues * eigenvectors  # m3.v - λ*v = 0
```

Out[164]:
```
array([[ 6.66133815e-15,  1.66533454e-15, -3.21964677e-15],
       [ 7.10542736e-15,  5.52335955e-15, -4.88498131e-15],
       [ 3.55271368e-14,  5.08620923e-15, -1.02140518e-14]])
```

In [165]:
```
1 m4 = np.array([[1,0,0,0,2], [0,0,3,0,0], [0,0,0,0,0], [0,2,0,0,0]])
2 m4
```

Out[165]:
```
array([[1, 0, 0, 0, 2],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0]])
```

In [166]:
```
U, S_diag, V = linalg.svd(m4)
U
```

Out[166]:
```
array([[ 0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  0.,  0., -1.],
       [ 0.,  0.,  1.,  0.]])
```

In [167]:
```
S_diag
```

Out[167]:
```
array([3.        , 2.23606798, 2.        , 0.        ])
```

In [168]:
```
S = np.zeros((4, 5))
S[np.diag_indices(4)] = S_diag
S  # Σ
```

Out[168]:
```
array([[3.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 2.23606798, 0.        , 0.        , 0.        ],
       [0.        , 0.        , 2.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ]])
```

In [169]:
```
V
```

Out[169]:
```
array([[-0.        ,  0.        ,  1.        ,  0.        ,  0.        ],
       [ 0.4472136 ,  0.        ,  0.        ,  0.        ,  0.89442719],
       [-0.        ,  1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  1.        ,  0.        ],
       [-0.89442719,  0.        ,  0.        ,  0.        ,  0.4472136 ]])
```

In [170]:
```
U.dot(S).dot(V) # U.Σ.V == m4
```

Out[170]:
```
array([[1., 0., 0., 0., 2.],
       [0., 0., 3., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 2., 0., 0., 0.]])
```

In [171]:
```
np.diag(m3)  # the values in the diagonal of m3 (top left to bottom right)
```

Out[171]:
```
array([ 1,  7, 31])
```

In [172]:
```
np.trace(m3)  # equivalent to np.diag(m3).sum()
```

Out[172]:
```
39
```

In [173]:
```
coeffs  = np.array([[2, 6], [5, 3]])
depvars = np.array([6, -9])
solution = linalg.solve(coeffs, depvars)
solution
```

Out[173]:
```
array([-3.,  2.])
```

In [174]:
```
coeffs.dot(solution), depvars  # yep, it's the same
```

Out[174]:
```
(array([ 6., -9.]), array([ 6, -9]))
```

In [175]:
```
np.allclose(coeffs.dot(solution), depvars)
```

Out[175]:
```
True
```

In [176]:
```
import math
data = np.empty((768, 1024))
for y in range(768):
    for x in range(1024):
        data[y, x] = math.sin(x*y/40.5)  # BAD! Very inefficient.
```

In [177]:
```
x_coords = np.arange(0, 1024)  # [0, 1, 2, ..., 1023]
y_coords = np.arange(0, 768)   # [0, 1, 2, ..., 767]
X, Y = np.meshgrid(x_coords, y_coords)
X
```

Out[177]:
```
array([[   0,    1,    2, ..., 1021, 1022, 1023],
       [   0,    1,    2, ..., 1021, 1022, 1023],
       [   0,    1,    2, ..., 1021, 1022, 1023],
       ...,
       [   0,    1,    2, ..., 1021, 1022, 1023],
       [   0,    1,    2, ..., 1021, 1022, 1023],
       [   0,    1,    2, ..., 1021, 1022, 1023]])
```

In [178]:
```
Y
```

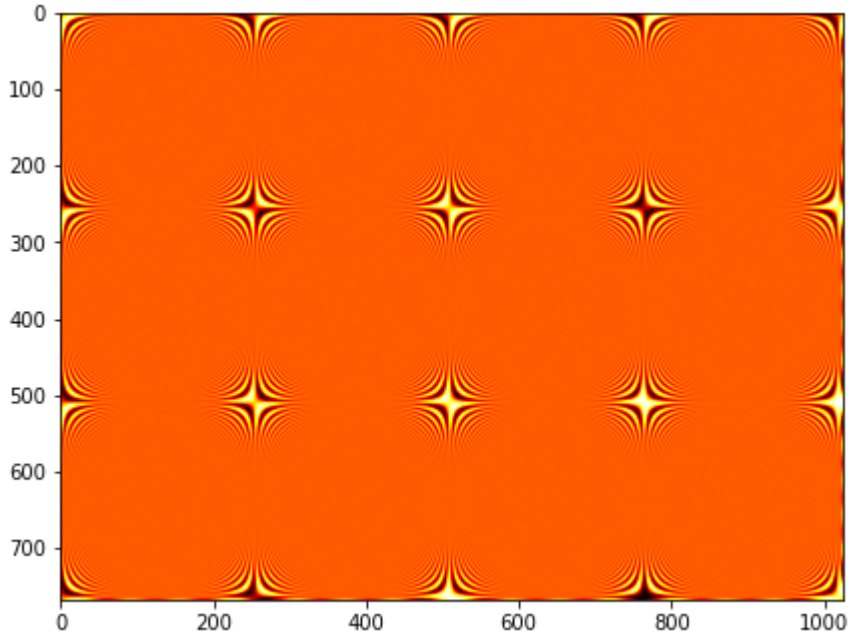
Out[178]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  1,   1,   1, ...,   1,   1,   1],
       [  2,   2,   2, ...,   2,   2,   2],
       ...,
       [765, 765, 765, ..., 765, 765, 765],
       [766, 766, 766, ..., 766, 766, 766],
       [767, 767, 767, ..., 767, 767, 767]])
```

In [179]:
```
data = np.sin(X*Y/40.5)
```

In [180]:
```
1 data
```

Out[180]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.02468885, 0.04936265, ..., 0.07705885, 0.1016508 ,
        0.12618078],
       [0.        , 0.04936265, 0.09860494, ..., 0.15365943, 0.20224852,
        0.25034449],
       ...,
       [0.        , 0.03932283, 0.07858482, ..., 0.6301488 , 0.59912825,
        0.56718092],
       [0.        , 0.06398059, 0.12769901, ..., 0.56844086, 0.51463783,
        0.45872596],
       [0.        , 0.08859936, 0.17650185, ..., 0.50335246, 0.42481591,
        0.34293805]])
```

In [181]:
```
1 import matplotlib.pyplot as plt
2 import matplotlib.cm as cm
3 fig = plt.figure(1, figsize=(7, 6))
4 plt.imshow(data, cmap=cm.hot, interpolation="bicubic")
5 plt.show()
```




In [182]:
```
1 a = np.random.rand(2,3)
2 a
```

Out[182]:
```
array([[0.34436763, 0.42532011, 0.95940086],
       [0.53245229, 0.9751774 , 0.16471651]])
```

In [183]:
```
1 np.save("my_array", a)
```

In [184]:
```
with open("my_array.npy", "rb") as f:
    content = f.read()

content
```

Out[184]:
```
b"\x93NUMPY\x01\x00v\x00{'descr': '<f8', 'fortran_order': False, 'shape': (2,
3), }                                                            \n\x1a\xab-\x82
\x1e\n\xd6?hb9\xd5q8\xdb?\x97\xf1\xe8oi\xb3\xee?s\xee0`\xd9\t\xe1?sj=>\xa74\xe
f?\xf8\x9c;5n\x15\xc5?"
```

In [185]:
```
a_loaded = np.load("my_array.npy")
a_loaded
```

Out[185]:
```
array([[0.34436763, 0.42532011, 0.95940086],
       [0.53245229, 0.9751774 , 0.16471651]])
```

In [186]:
```
np.savetxt("my_array.csv", a)
```

In [187]:
```
with open("my_array.csv", "rt") as f:
    print(f.read())
```

```
3.443676253579482927e-01 4.253201086299100986e-01 9.594086228761145560e-01
5.324522856457875042e-01 9.751774039862780574e-01 1.647165069509595181e-01
```

In [188]:
```
np.savetxt("my_array.csv", a, delimiter=",")
```

In [189]:
```
a_loaded = np.loadtxt("my_array.csv", delimiter=",")
a_loaded
```

Out[189]:
```
array([[0.34436763, 0.42532011, 0.95940086],
       [0.53245229, 0.9751774 , 0.16471651]])
```

In [190]:
```
b = np.arange(24, dtype=np.uint8).reshape(2, 3, 4)
b
```

Out[190]:
```
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]]], dtype=uint8)
```

In [191]:
```
np.savez("my_arrays", my_a=a, my_b=b)
```

In [192]:
```
with open("my_arrays.npz", "rb") as f:
    content = f.read()

repr(content)[:180] + "[...]"
```

Out[192]:
```
'b"PK\\x03\\x04\\x14\\x00\\x00\\x00\\x00\\x00\\x00\\x00!\\x00\\x1a\\x19\\xf39
\\xb0\\x00\\x00\\x00\\xb0\\x00\\x00\\x00\\x08\\x00\\x00\\x00my_a.npy\\x93NUMPY
\\x01\\x00v\\x00{\'descr\': \'<f8\', \'fortran_order\': False,[...]'
```

In [193]:
```
my_arrays = np.load("my_arrays.npz")
my_arrays
```

Out[193]:
```
<numpy.lib.npyio.NpzFile at 0x2608e32e448>
```

In [194]:
```
my_arrays.keys()
```

Out[194]:
```
KeysView(<numpy.lib.npyio.NpzFile object at 0x000002608E32E448>)
```

In [195]:
```
my_arrays["my_a"]
```

Out[195]:
```
array([[0.34436763, 0.42532011, 0.95940086],
       [0.53245229, 0.9751774 , 0.16471651]])
```

# refer below link for more

## https://numpy.org/doc/stable/reference/index.html (https://numpy.org/doc/stable/reference/index.htm

In [ ]:
```

```