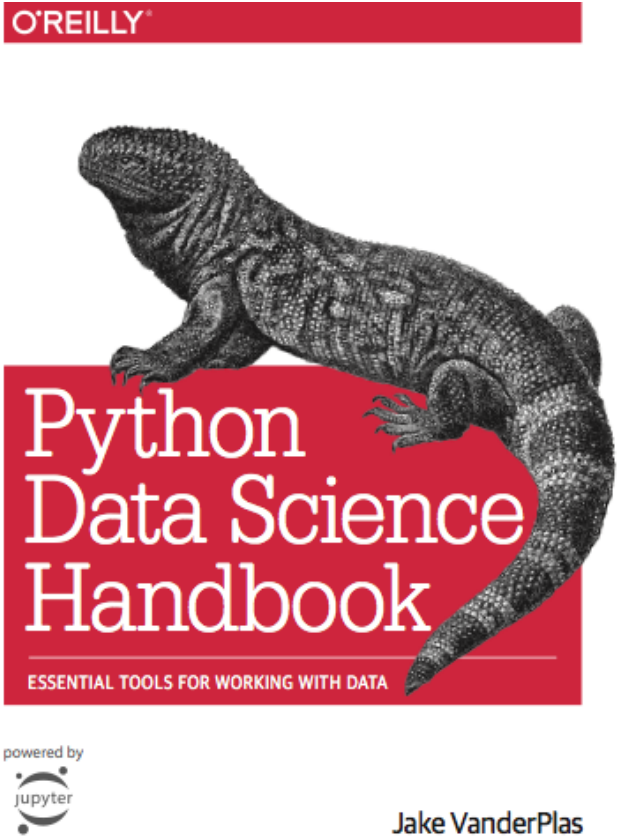
NumPy Tutorial

王文中

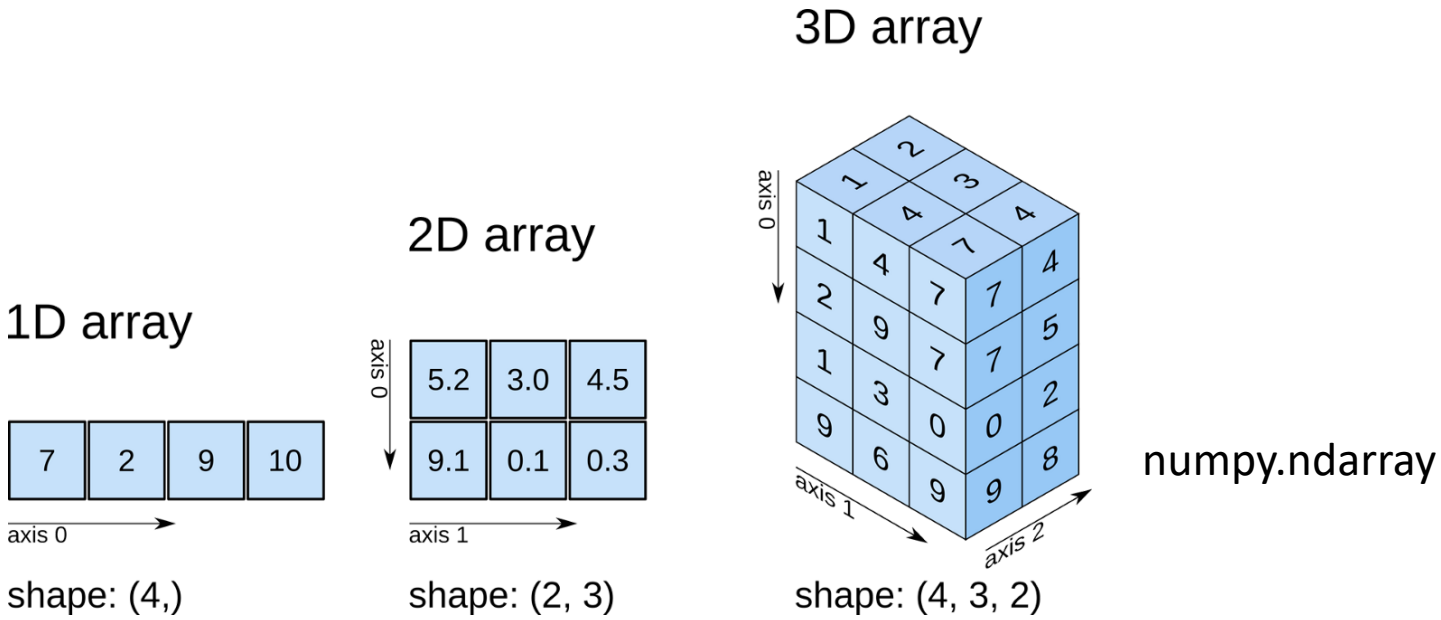
安徽大学计算机学院



https://jakevdp.github.io/PythonDataScienceHandbook/

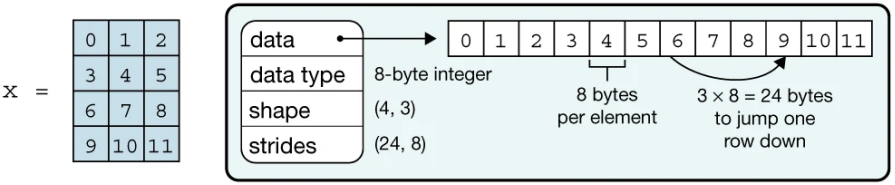
NumPy

NumPy is the primary array programming library for the Python language.

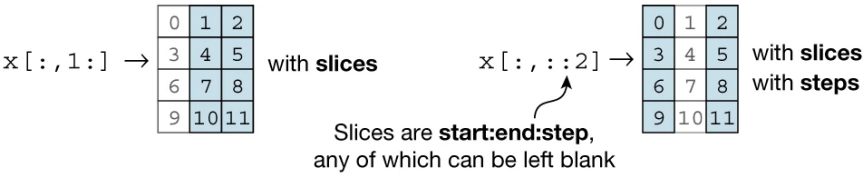


ndarray

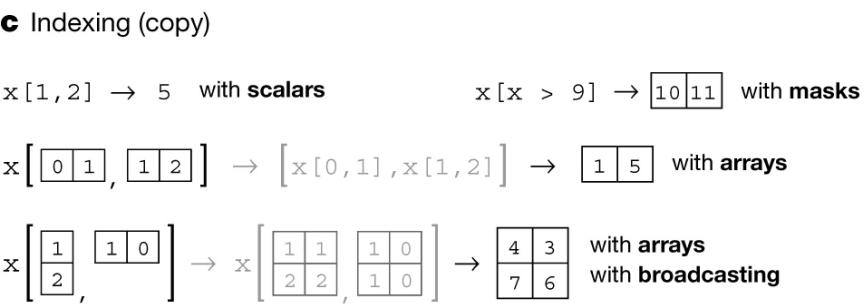
a Data structure



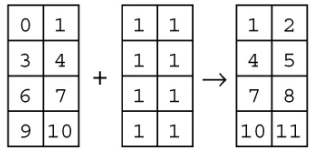
b Indexing (view)



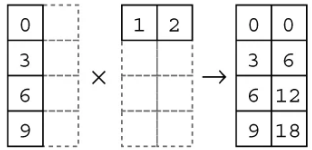
c Indexing (copy)



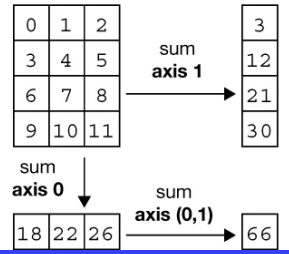
d Vectorization



e Broadcasting



f Reduction



g Example

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

In [2]: x = np.arange(12)

In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```

```
>>> import numpy as np
>>> np.__version__
'1.20.1'

>>> dir(np)
['ALLOW_THREADS', 'AxisError', 'BUFSIZE', … 'array', … ,
'transpose', … , 'vstack', 'warnings', 'where', 'who',
'zeros', 'zeros_like']


>>> dir(np.ndarray)
['T', … , 'astype', … , 'max', 'mean', 'min', … , 'reshape',
'resize', 'round', …, 'trace', 'transpose', 'var', 'view']
```

创建一个ndarray对象

创建ndarray对象

```
#从序列创建一个ndarray
>>> np.array([1,2,3])
array([1, 2, 3])

#指定数组元素的类型
>>> np.array([1,2,3],dtype = 'float')
array([1., 2., 3.])

#创建多维数组
>>> np.array([[1,2,3],[4,5,6]])
array([[1, 2, 3],
       [4, 5, 6]])
```

```
#创建多维数组
>>> np.array([[[1],[2],[3]],[[4],[5],[6]]])
array([[[1],
        [2],
        [3]],

       [[4],
        [5],
        [6]]])
```

方法1:从序列创建一个ndarray

创建ndarray对象

```
>>> np.zeros((3,4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
>>> np.ones((2,3))
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
>>> np.ones((2,2,3))
array([[[1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.]]])
```

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
>>> np.identity(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

方法2:创建特殊的ndarray

创建ndarray对象

```
>>> np.random.rand(2,3)
array([[0.56694182, 0.99506098, 0.51948246],
       [0.15170283, 0.80824191, 0.87334274]])


>>> np.random.randint(-5,5,(2,3))
array([[ 1,  4,  1],
       [-3, -2,  0]])

>>> np.random.normal(0,1,(2,3))
array([[ 2.78230738, -1.51830277, -0.70990982],
       [-0.25289697, -0.96932164,  0.30035351]])
```

方法3:用随机数函数创建ndarray

创建ndarray对象

```
>>> np.empty((2,3))
array([[1.37961370e-306, 1.37960012e-306, 4.22802739e-307],
       [1.24611470e-306, 8.34423493e-308, 1.44635488e-307]])

>>> x = np.random.rand(2,3)
>>> x.shape
(2, 3)

>>> np.empty_like(x)
array([[1.37961370e-306, 1.37960012e-306, 4.22802739e-307],
       [1.24611470e-306, 8.34423493e-308, 1.44635488e-307]])
```

```
>>> np.full((2,3),fill_value = 0)
array([[0, 0, 0],
       [0, 0, 0]])

>>> np.full((2,3),fill_value = np.inf)
array([[inf, inf, inf],
       [inf, inf, inf]])

>>> np.full_like(x,fill_value = 255)
array([[255., 255., 255.],
       [255., 255., 255.]])
```

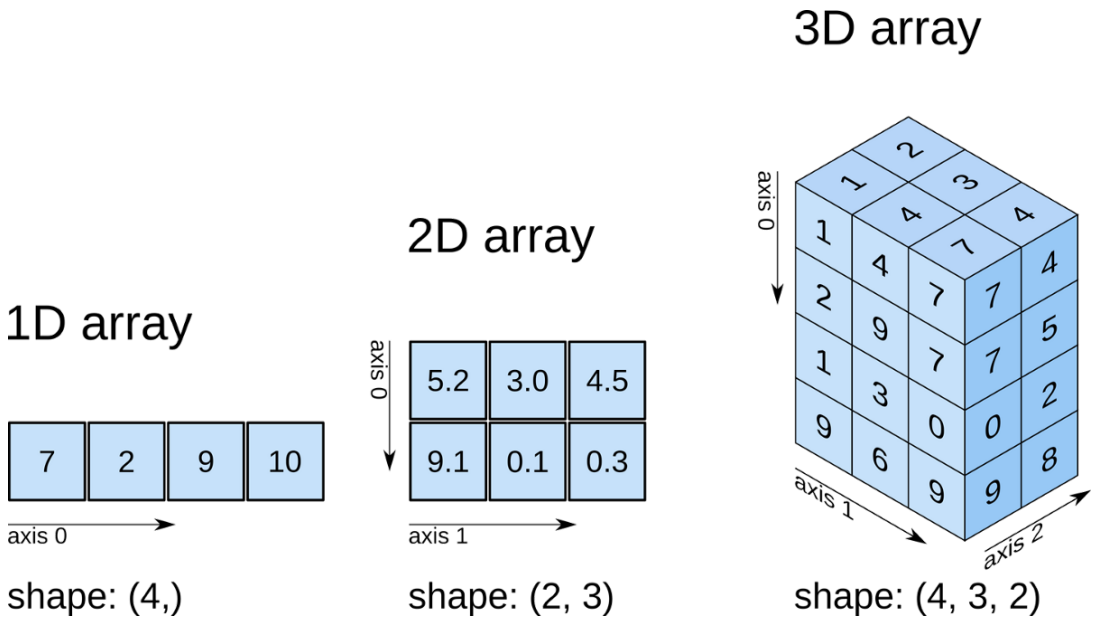
方法4:用np.empty,np.full等函数创建ndarray

创建ndarray对象

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.linspace(1,5,10)
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])

>>> x = np.logspace(1,5,10)
>>> x
array([1.00000000e+01, 2.78255940e+01, 7.74263683e+01, 2.15443469e+02,
       5.99484250e+02, 1.66810054e+03, 4.64158883e+03, 1.29154967e+04,
       3.59381366e+04, 1.00000000e+05])
>>> np.log10(x)
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

方法5:用np.arange等函数创建一维ndarray

ndarray对象的属性



维度: ndarray.ndim
形状: ndarray.shape
跨度: ndarray.strides

```
>>> x = np.random.rand(2)
>>> x
array([0.25814068, 0.89113847])
>>> x.shape
(2,)
>>> x.ndim
1
```

```
>>> y = np.random.rand(2,3)
>>> y.shape
(2, 3)
>>> y.ndim
2
>>> y
array([[0.34426797, 0.7299161 , 0.12344461],
       [0.26655586, 0.44549793, 0.29565539]])
```

```
>>> z = np.random.randint(-5,5,(2,2,3))
>>> z.shape
(2, 2, 3)
>>> z.ndim
3
```

```
>>> x = np.array(7)
>>> x
array(7)
>>> x.shape
()
>>> x.ndim
0
```

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.dtype
dtype('int32')
>>> x.itemsize
4
>>> x.size
6
```

```
>>> y = np.array([[1,2,3],[4,5,6]],dtype=np.float64)
>>> y
array([[1., 2., 3.],
       [4., 5., 6.]])
>>> y.dtype
dtype('float64')
>>> y.itemsize
8
>>> y.size
6
```

ndarray元素的存储结构

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x.itemsize
4
>>> x.size
6

>>> x.data
<memory at 0x0000014CEA1125F0>

>>> x.strides
(12, 4)
>>> np.array(x.strides)//x.itemsize
array([3, 1], dtype=int32)
```

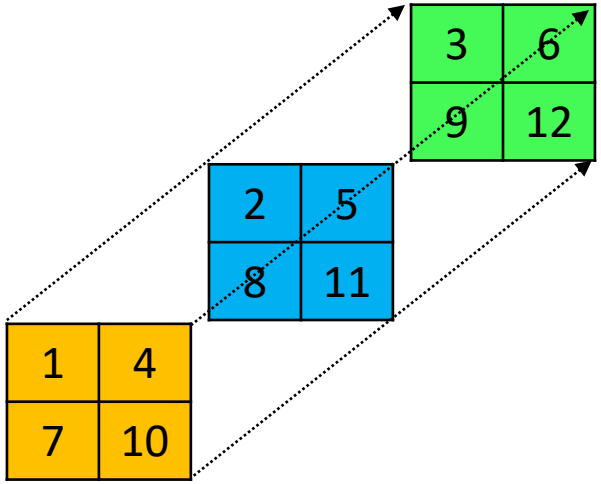
1	2	3	4	5	6

默认按行存储(C语言数组的存储方式)

ndarray元素的存储结构

```
>>> x = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
>>> x
array([[[ 1,  2,  3],
        [ 4,  5,  6]],
       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> x.shape
(2, 2, 3)
>>> x.strides
(24, 12, 4)
>>> x.size
12
>>> x.itemsize
4
>>> x.data
<memory at 0x0000014CEA0BEA90>
```

1	2	3	4	5	6	7	8	9	10	11	12



访问数组元素

用下标访问数组元素

```
>>> x = np.arange(10)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> x[2]
2
>>> x[1:9:2]
array([1, 3, 5, 7])

>>> x = np.array([[1,2,3,4,5,6],[7,8,9,10,11,12]])
>>> x
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
>>> x[:,::-1]
array([[ 6,  5,  4,  3,  2,  1],
       [12, 11, 10,  9,  8,  7]])
>>> x[:,4:0:-1]
array([[ 5,  4,  3,  2],
       [11, 10,  9,  8]])
```

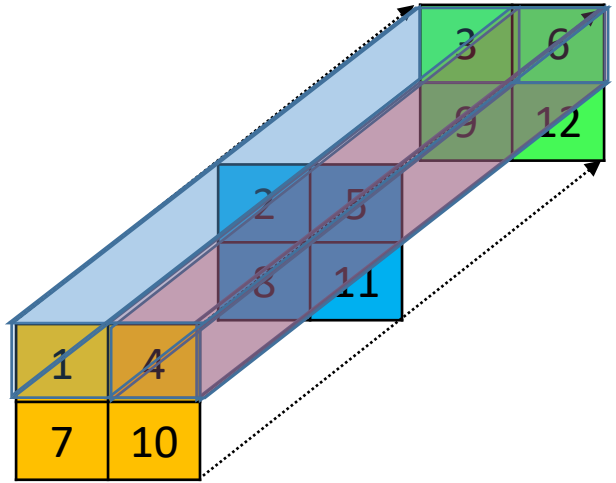
用下标访问数组元素

$X[i_0, i_1, \ldots, i_{d-1}], i_j \in [0, X.shape[j] - 1], d = X.ndim$

$X[i_0, i_1, \ldots, i_{d-1}]$在内存中的位置是:$X.data + \sum_{j=0}^{d-1} i_j \times X.strides[j]$

```
>>> x = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
>>> x
array([[[ 1,  2,  3],
        [ 4,  5,  6]],
       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> x.shape
(2, 2, 3)
>>> x.strides
(24, 12, 4)
>>> x[0,1,2]
6
```

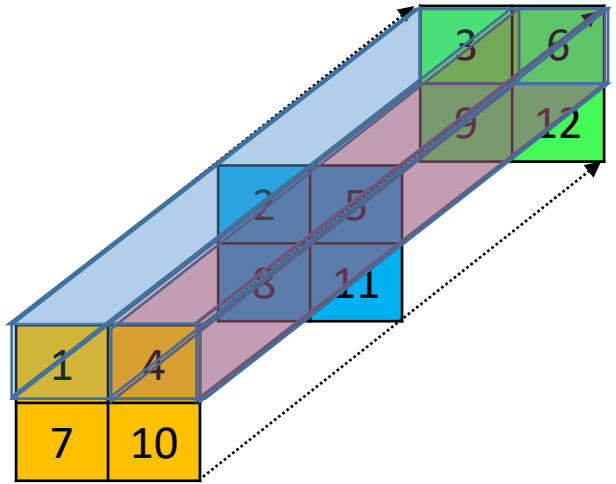
1	2	3	4	5	6	7	8	9	10	11	12



数组切片(Slicing)

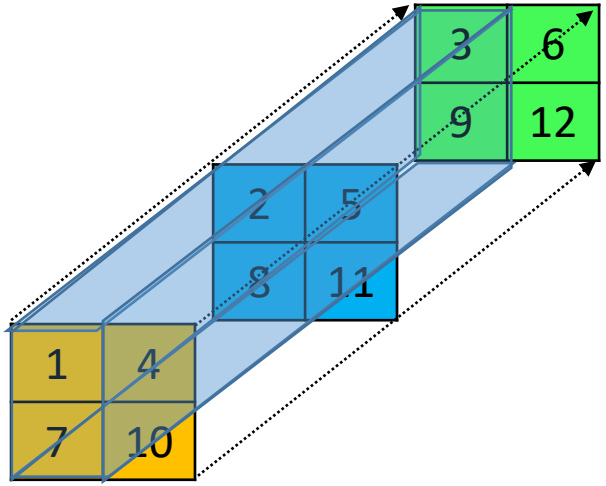
```
>>> x = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
>>> x
array([[[ 1,  2,  3],
        [ 4,  5,  6]],
       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> x[0]
array([[1, 2, 3],
       [4, 5, 6]])
>>> x[0].shape
(2, 3)
>>> x[0][1]
array([4, 5, 6])
>>> x[0,1]
array([4, 5, 6])
```

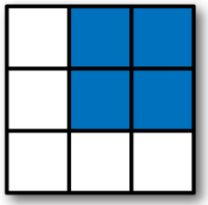
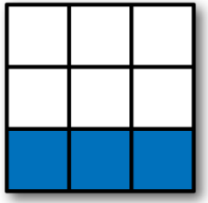
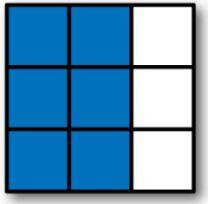
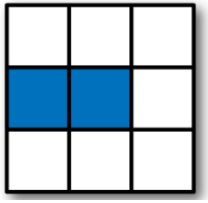
1	2	3	4	5	6	7	8	9	10	11	12



数组切片

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

>>> x[:,:1,:]
array([[[1, 2, 3]],

       [[7, 8, 9]]])

>>> x[:,:1,:2]
array([[[1, 2]],

       [[7, 8]]])
```



	Expression	Shape
	`arr[:2, 1:]`	`(2, 2)`
	`arr[2]`	`(3,)`
	`arr[2, :]`	`(3,)`
	`arr[2:, :]`	`(1, 3)`
	`arr[:, :2]`	`(3, 2)`
	`arr[1, :2]`	`(2,)`
	`arr[1:2, :2]`	`(1, 2)`

布尔索引(Boolean Indexing)

```
>>> x = np.random.randint(-5,5,(5,10))
>>> x
array([[ 3,  4,  2, -5, -1, -5, -5, -5,  4,  0],
       [ 3, -2,  1, -2,  0,  2, -3, -4,  3, -2],
       [ 2, -1, -4,  1, -3, -2,  2, -2,  1,  2],
       [ 1,  1, -4,  4,  3, -1,  1,  4,  4,  2],
       [-1, -1,  1,  4,  2,  3,  0,  3, -1,  2]])
>>> y = np.random.rand(5)
>>> y
array([0.31187091, 0.52794402, 0.85590955, 0.16405146, 0.52598888])
>>> y>0.5
array([False,  True,  True, False,  True])
>>> x[y>0.5,:]
array([[ 3, -2,  1, -2,  0,  2, -3, -4,  3, -2],
       [ 2, -1, -4,  1, -3, -2,  2, -2,  1,  2],
       [-1, -1,  1,  4,  2,  3,  0,  3, -1,  2]])
```

布尔索引(Boolean Indexing)

```
>>> x = np.random.randn(4,5)
>>> x
array([[-0.01311172, -0.42492337,  1.70629494,  0.09026107, -1.57295181],
       [-1.47649472,  1.57279371, -0.3945336 , -0.549308  ,  0.31950348],
       [ 0.13884932,  0.54295891, -0.2315363 ,  0.44926544,  1.08650511],
       [-0.83692313,  0.14914834,  0.55520105, -0.00895871,  0.98064664]])
>>> x>1
array([[False, False,  True, False, False],
       [False,  True, False, False, False],
       [False, False, False, False,  True],
       [False, False, False, False, False]])
>>> x[x>1]
array([1.70629494, 1.57279371, 1.08650511])
```

布尔索引(Boolean Indexing)

```
>>> x = np.random.randn(4,5)
>>> x
array([[-0.01311172, -0.42492337,  1.70629494,  0.09026107, -1.57295181],
       [-1.47649472,  1.57279371, -0.3945336 , -0.549308  ,  0.31950348],
       [ 0.13884932,  0.54295891, -0.2315363 ,  0.44926544,  1.08650511],
       [-0.83692313,  0.14914834,  0.55520105, -0.00895871,  0.98064664]])
#把x截断到[-1,1]范围
>>> x[x>1] = 1
>>> x[x<-1] = -1
>>> x
array([[-0.01311172, -0.42492337,  1.        ,  0.09026107, -1.        ],
       [-1.        ,  1.        , -0.3945336 , -0.549308  ,  0.31950348],
       [ 0.13884932,  0.54295891, -0.2315363 ,  0.44926544,  1.        ],
       [-0.83692313,  0.14914834,  0.55520105, -0.00895871,  0.98064664]])
```

布尔索引(Boolean Indexing)

```
>>> x = np.log(np.random.randint(-5,5,(3,5)))
>>> x
array([[0.        ,       -inf,        nan, 0.        , 1.09861229],
       [       nan,        nan,       -inf, 1.09861229, 1.38629436],
       [       nan, 0.        ,        nan,        nan,        nan]])

#清洗异常值
>>> x[np.isnan(x)] = 0
>>> x[np.isinf(x)] = 1
>>> x
array([[0.        , 1.        , 0.        , 0.        , 1.09861229],
       [0.        , 0.        , 1.        , 1.09861229, 1.38629436],
       [0.        , 0.        , 0.        , 0.        , 0.        ]])
```

高级(花样)索引(Fancy Indexing)

用整数数组作为索引

```
>>> x = np.empty((5,8))
>>> for i in range(8):
...     x[:,i] = i
>>> x
array([[0., 1., 2., 3., 4., 5., 6., 7.],
       [0., 1., 2., 3., 4., 5., 6., 7.],
       [0., 1., 2., 3., 4., 5., 6., 7.],
       [0., 1., 2., 3., 4., 5., 6., 7.],
       [0., 1., 2., 3., 4., 5., 6., 7.]])
>>> x[:,[5,3,7,0]]
array([[5., 3., 7., 0.],
       [5., 3., 7., 0.],
       [5., 3., 7., 0.],
       [5., 3., 7., 0.],
       [5., 3., 7., 0.]])
```

```
>>> x[:,[-1,-3,-5]]
array([[7., 5., 3.],
       [7., 5., 3.],
       [7., 5., 3.],
       [7., 5., 3.],
       [7., 5., 3.]])
```

高级(花样)索引(Fancy Indexing)

用整数数组作为索引

```
>>> x = np.random.randint(-5,5,(5,10))
>>> x
array([[-4,  1, -4,  0,  1,  3,  1,  2, -3, -4],
       [-5,  4,  3,  4,  1, -3,  3,  1,  2, -1],
       [-2,  1,  3,  3, -2, -4,  4, -1,  3, -1],
       [ 3, -2, -4,  2, -3, -4,  4,  2, -4,  0],
       [ 4,  2, -1, -2,  4,  2, -3,  4,  0,  4]])

>>> x[[1,2,3],[4,6,7]]
array([1, 4, 2])
```

高级(花样)索引(Fancy Indexing)

```
>>> y = np.random.randint(0,4,(10,))
>>> y
array([1, 3, 2, 0, 0, 2, 1, 1, 2, 2])
#把整数label转换为one-hot vector
>>> h = np.zeros((4,y.shape[0]))#h的每一列表示一个one-hot vector
>>> h
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., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
>>> h[y, np.arange(y.shape[0])] = 1
>>> h
array([[0., 0., 0., 1., 1., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 1., 1., 0., 0.],
       [0., 0., 1., 0., 0., 1., 0., 0., 1., 1.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

数组操作

ndarray.reshape

```
>>> x = np.arange(15)
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])

>>> x.reshape((3,5))
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

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

ndarray.reshape

```
>>> x = np.arange(15).reshape((1,3,5))
>>> x
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14]]])

>>> x.reshape((-1,5))
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
>>> x.reshape((5,-1))
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])

>>> x.reshape(-1)
array([ 0,  1,  2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14])
```

ndarray.transpose

```
>>> x = np.arange(15).reshape(3,5)
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> x.transpose()
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
>>> x.T
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

ndarray.transpose

```
>>> x = np.arange(24).reshape((2,3,4))
>>> x
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]]])
```

```
>>> x[1,2,3]
23
>>> y[3,1,2]
23
```

```
>>> y = x.transpose((2,0,1))
>>> y
array([[[ 0,  4,  8],
        [12, 16, 20]],

       [[ 1,  5,  9],
        [13, 17, 21]],

       [[ 2,  6, 10],
        [14, 18, 22]],

       [[ 3,  7, 11],
        [15, 19, 23]]])
```

ndarray.squeeze, ndarray.newaxis

```
>>> x = np.arange(24).reshape((2,3,4))
>>> x
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]]])
```

```
>>> x.shape
(2, 3, 4)
>>> y = x[:,np.newaxis,:,:]
>>> y.shape
(2, 1, 3, 4)

>>> z = y.squeeze()
>>> z.shape
(2, 3, 4)
```

ndarray.flatten, ndarray. ravel

```
>>> x = np.arange(15).reshape((3,5))
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> y = x.flatten()
>>> y
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> z = x.ravel()
>>> z
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> x[0] = -1
>>> y
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> z
array([-1, -1, -1, -1, -1,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

ndarray.repeat

```
>>> x = np.random.randint(0,5,(2,3))
>>> x
array([[3, 0, 4],
       [1, 2, 0]])

>>> x.repeat(2,axis = 0)
array([[3, 0, 4],
       [3, 0, 4],
       [1, 2, 0],
       [1, 2, 0]])

>>> x.repeat(2,axis = 1)
array([[3, 3, 0, 0, 4, 4],
       [1, 1, 2, 2, 0, 0]])
```

np.split

```
>>> x = np.arange(9.0)
>>> np.split(x, 3)
[array([0.,  1.,  2.]), array([3.,  4.,  5.]), array([6.,  7.,  8.])]

>>> x = np.arange(8.0)
>>> np.split(x, [3, 5, 6, 10])
[array([0.,  1.,  2.]),
 array([3.,  4.]),
 array([5.]),
 array([6.,  7.]),
 array([], dtype=float64)]
```

np.hsplit, np.vsplit

```
>>> x = np.arange(16.0).reshape(4, 4)
>>> x
array([[ 0.,   1.,   2.,   3.],
       [ 4.,   5.,   6.,   7.],
       [ 8.,   9.,  10.,  11.],
       [12.,  13.,  14.,  15.]])

>>> np.vsplit(x, 2)
[array([[0., 1., 2., 3.],
        [4., 5., 6., 7.]]),
 array([[ 8.,  9., 10., 11.],
        [12., 13., 14., 15.]])]

>>> np.vsplit(x, np.array([3, 6]))
[array([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]]),
 array([[12., 13., 14., 15.]]),
 array([], shape=(0, 4), dtype=float64)]
```

```
>>> np.hsplit(x, 2)
[array([[  0.,   1.],
        [  4.,   5.],
        [  8.,   9.],
        [12.,  13.]]),
 array([[  2.,   3.],
        [  6.,   7.],
        [10.,  11.],
        [14.,  15.]])]
>>> np.hsplit(x, np.array([3, 6]))
[array([[ 0.,   1.,   2.],
        [ 4.,   5.,   6.],
        [ 8.,   9.,  10.],
        [12.,  13.,  14.]]),
 array([[ 3.],
        [ 7.],
        [11.],
        [15.]]),
 array([], shape=(4, 0), dtype=float64)]
```

np.dsplit

```
>>> x = np.arange(16.0).reshape(2, 2, 4)
>>> x
array([[[ 0.,   1.,   2.,   3.],
        [ 4.,   5.,   6.,   7.]],
       [[ 8.,   9.,  10.,  11.],
        [12.,  13.,  14.,  15.]]])

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

```
>>> np.dsplit(x, np.array([3, 6]))
[array([[[ 0.,   1.,   2.],
         [ 4.,   5.,   6.]],
        [[ 8.,   9.,  10.],
         [12.,  13.,  14.]]]),
 array([[[ 3.],
         [ 7.]],
        [[11.],
         [15.]]]),
 array([], shape=(2, 2, 0), dtype=float64)]
```

np.stack

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([2, 3, 4])
>>> np.stack((a, b))#np.stack((a, b), axis=0)
array([[1, 2, 3],
       [2, 3, 4]])

>>> np.stack((a, b), axis=-1)#np.stack((a, b), axis=1)
array([[1, 2],
       [2, 3],
       [3, 4]])
```

np.stack

```
>>> arrays = [np.random.randn(3, 4) for _ in range(10)]
>>> np.stack(arrays, axis=0).shape
(10, 3, 4)

>>> np.stack(arrays, axis=1).shape
(3, 10, 4)

>>> np.stack(arrays, axis=2).shape
(3, 4, 10)
```

np.hstack,np.vstack

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([2, 3, 4])

>>> np.hstack((a,b))
array([1, 2, 3, 2, 3, 4])

>>> np.vstack((a,b))
array([[1, 2, 3],
       [2, 3, 4]])
```

np.hstack,np.vstack

```
>>> import cv2 as cv
>>> imgs = [cv.imread('%d.jpg'%(i)) for i in range(4)]
>>> M = np.hstack(imgs)
>>> cv.namedWindow('images'),cv.imshow('images',M),cv.waitKey(0),cv.destroyAllWindows()
```

np.hstack,np.vstack


```
M = np.vstack((np.hstack(imgs[:2]),np.hstack(imgs[2:])))
```

```
>>> import cv2 as cv
>>> im = cv.imread('sunflower.jpg')
>>> im.shape
(854, 1280, 3)
>>> im = im[:,:,[2,1,0]]
>>> r,g,b = np.dsplit(im)
>>> M = np.hstack((r,g,b))
```

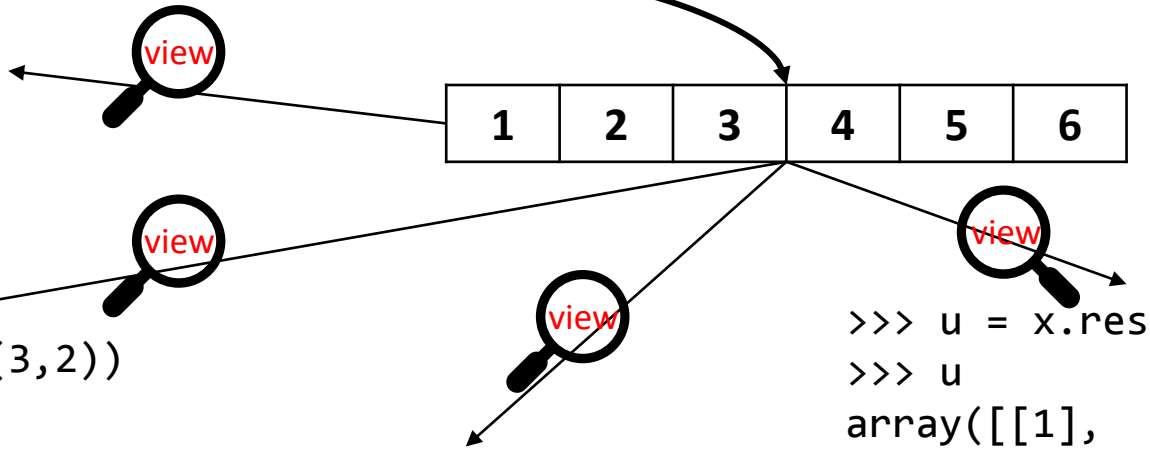
```
>>> r,g,b = np.dsplit(im)
>>> r.shape
(854, 1280, 1)
>>> r = r.squeeze()
>>> r.shape
(854, 1280)
>>> zeros = np.full_like(r,fill_value = 0)
>>> R = np.stack((r,zeros,zeros), axis = 2)
>>> G = np.stack((zeros,g,zeros), axis = 2)
>>> B = np.stack((zeros,zeros,b), axis = 2)
>>> M = np.hstack((R,G,B))
```

```
>>> R = im.copy()
>>> R[:,:,[1,2]] = 0

>>> G = im.copy()
>>> G[:,:,[0,2]] = 0

>>> B = im.copy()
>>> B[:,:,[0,1]] = 0

>>> M = np.hstack((R,G,B))
```

数组的视图(view)

视图

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



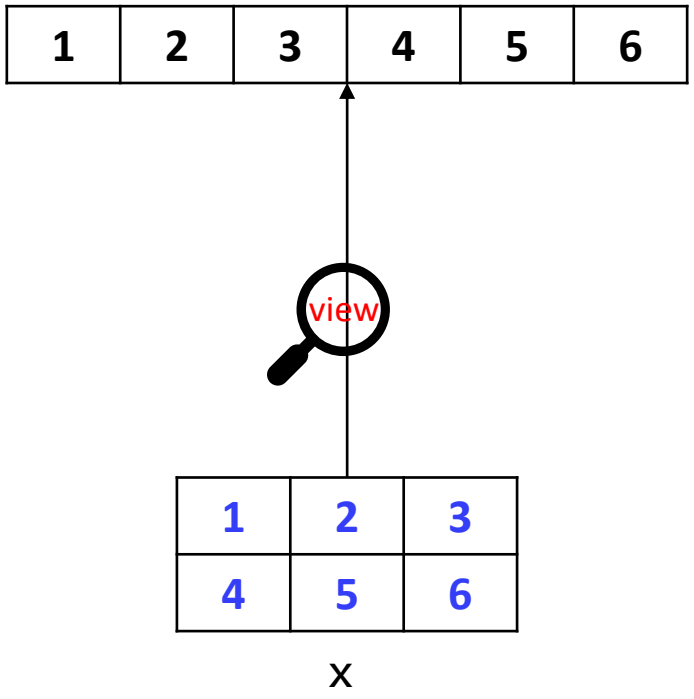

```
>>> y = x.reshape((3,2))
>>> y
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
>>> z = x.reshape((1,-1))
>>> z
array([[1, 2, 3, 4, 5, 6]])
```

```
>>> u = x.reshape((-1,1))
>>> u
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

视图

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x.shape
(2, 3)
>>> x.strides
(12, 4)
>>> x.itemsize
4
>>> np.array(x.strides)//x.itemsize
array([3, 1], dtype=int32)
```

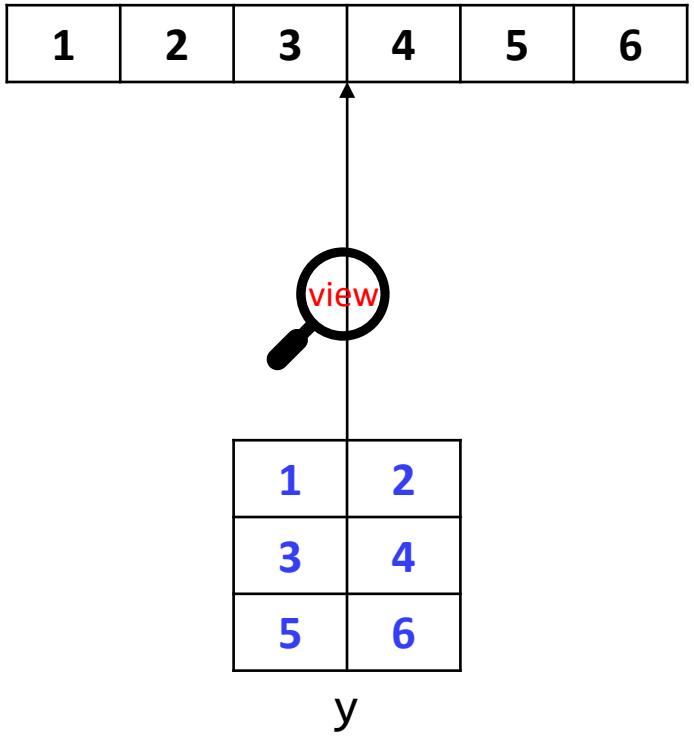



```
&(x[i+1,j]) - &(x[i,j]) == x.strides[0]

&(x[i,j+1]) - &(x[i,j]) == x.strides[1]
```

视图

```
>>> y = x.reshape((3,2))
>>> y.strides
(8, 4)
>>> np.array(y.strides)//y.itemsize
array([2, 1], dtype=int32)
```

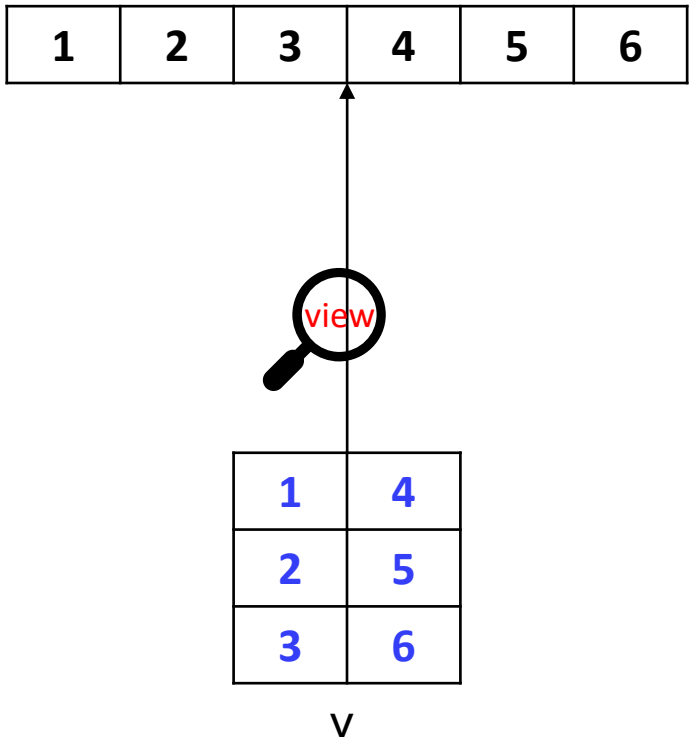
```
&(y[i+1,j]) - &(y[i,j]) == y.strides[0]

&(y[i,j+1]) - &(y[i,j]) == y.strides[1]
```



视图

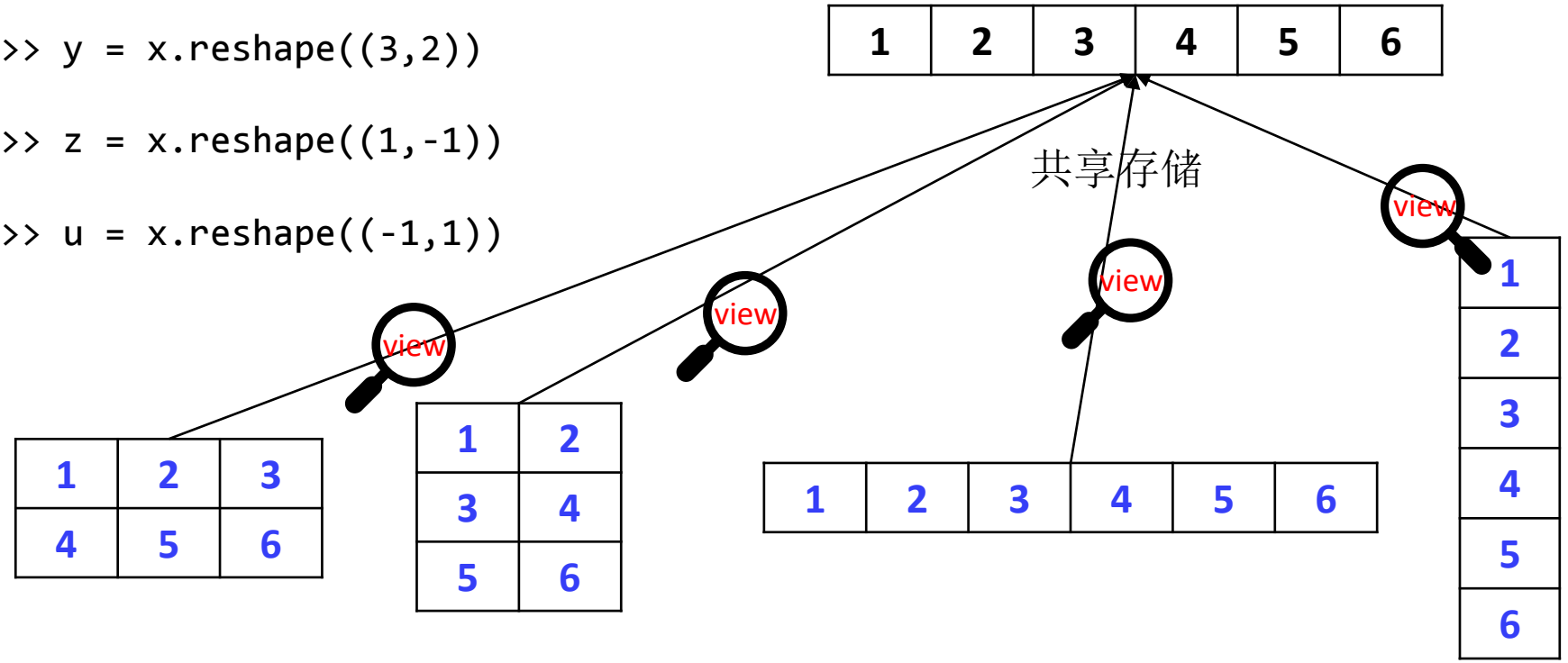
```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x
array([[1, 2, 3],
       [4, 5, 6]])

>>> v = x.T
>>> v.strides
(4, 12)
>>> np.array(v.strides)//v.itemsize
array([1, 3], dtype=int32)
```

```
&(v[i+1,j]) - &(v[i,j]) == v.strides[0]

&(v[i,j+1]) - &(v[i,j]) == v.strides[1]
```



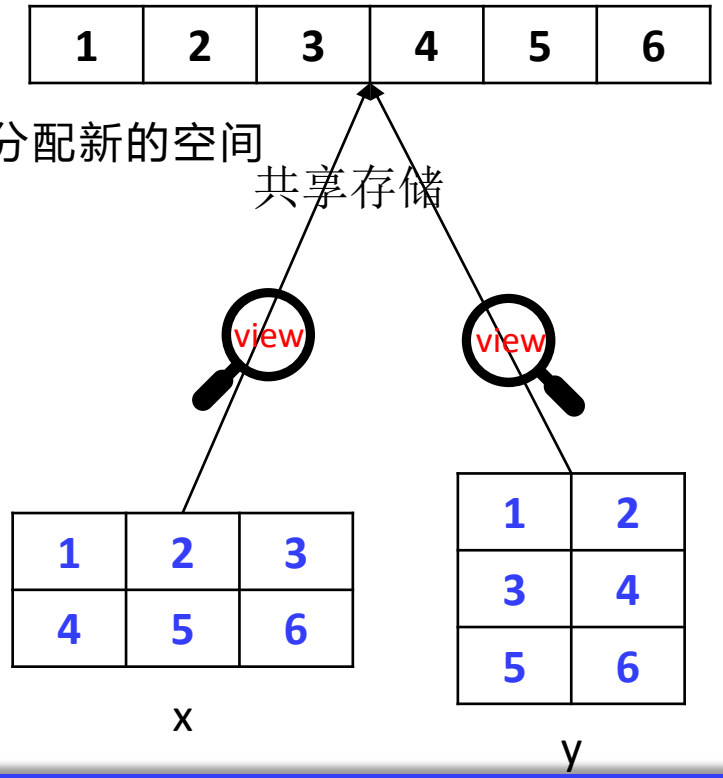
视图之间共享存储

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

>>> y = x.reshape((3,2))

>>> z = x.reshape((1,-1))

>>> u = x.reshape((-1,1))
```



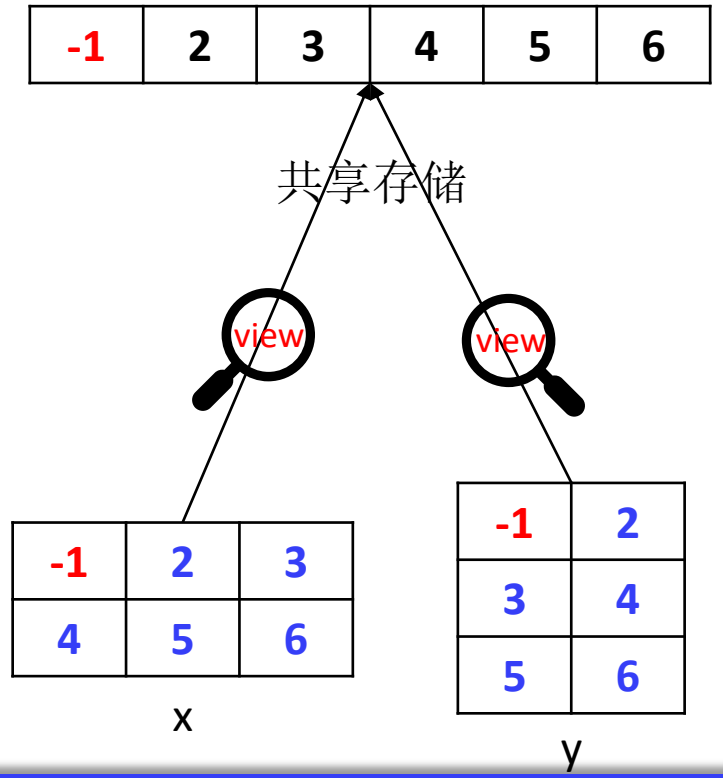
视图之间共享存储

```
#创建数组x,x"拥有"数组的存储空间
>>> x = np.array([[1,2,3],[4,5,6]])

#获取数组的一个新视图,不需要为数组元素分配新的空间
>>> y = x.reshape((3,2))

>>> x.base is None
True

>>> y.base
array([[-1,  2,  3],
       [ 4,  5,  6]])
>>> y.base is x
True
```



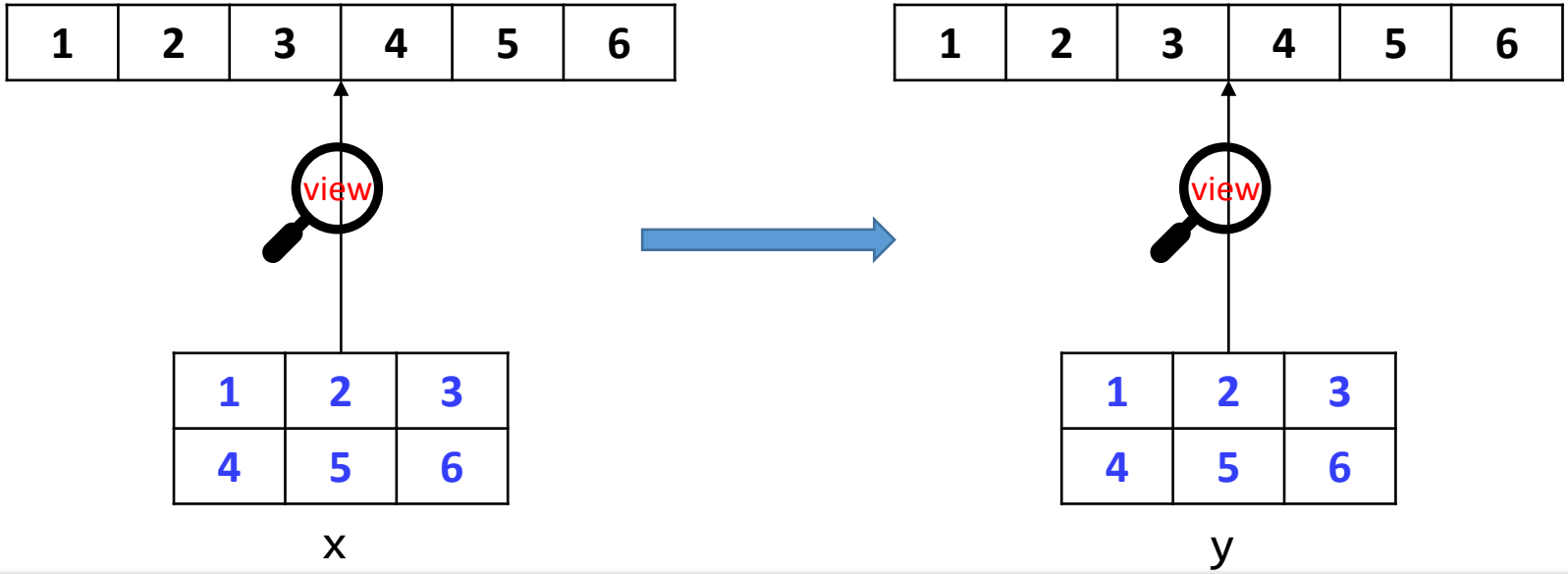
视图之间共享存储

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

>>> y = x.reshape((3,2))

>>> x[0,0] = -1

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



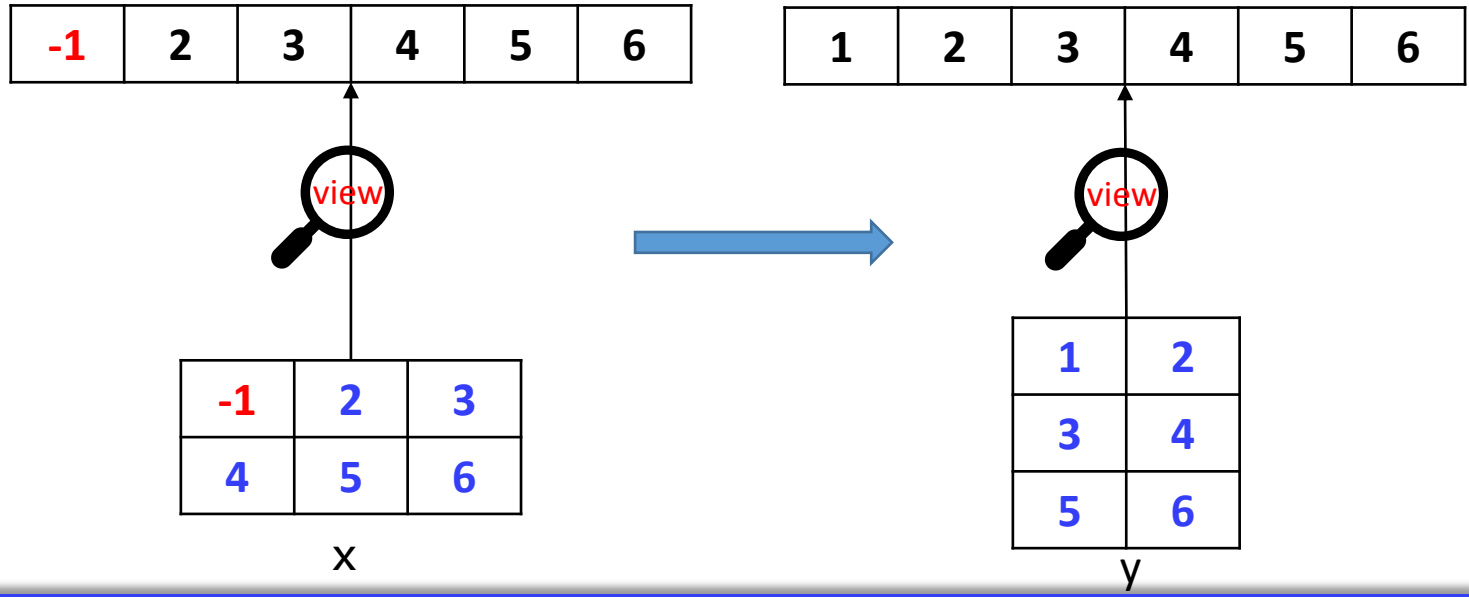
ndarray.copy

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> y = x.copy()
>>> y.base is x
False
```



ndarray.copy

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> y = x.copy().reshape((3,2))
>>> x[0,0] = -1
>>> y
array([[1, 2],
       [3, 4],
       [5, 6]])
```



视图

```
>>> x = np.random.rand(2,3)

>>> y = x[:,np.newaxis,:]
>>> y.base is x
True

>>> z = y.squeeze()
>>> z.base is x
True

>>> u = x.flatten()
>>> u.base is x
False
```

```
>>> v = x.ravel()
>>> v.base is x
True

>>> w = x.transpose((1,0))
>>> w.base is x
True

>>> r = x.repeat(2,0)
>>> r.base is x
False
```

数组上的运算

Universal function (ufunc)

```
from math import sqrt
for _ in range(1000):
    sum([sqrt(x) for x in range(10000)])
```

1.627s

```
import numpy as np
for _ in range(1000):
    np.sqrt(np.arange(10000)).sum()
```

0.857s

Universal funcrion (ufunc)

```
>>> x = np.arange(10)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.sqrt(x)
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

ufunc:A universal function is a function that operates on ndarrays in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features. That is, a ufunc is a “vectorized” wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs.

算术运算

Operator	Equivalent ufunc	Description
+	np.add	Addition (e.g., 1 + 1 = 2)
-	np.subtract	Subtraction (e.g., 3 - 2 = 1)
-	np.negative	Unary negation (e.g., -2)
*	np.multiply	Multiplication (e.g., 2 * 3 = 6)
/	np.divide	Division (e.g., 3 / 2 = 1.5)
//	np.floor_divide	Floor division (e.g., 3 // 2 = 1)
**	np.power	Exponentiation (e.g., 2 ** 3 = 8)
%	np.mod	Modulus/remainder (e.g., 9 % 4 = 1)

算术运算

```
>>> x = np.random.randint(1,5,(2,3))
>>> y = np.random.randint(1,5,(2,3))
>>> x
array([[2, 3, 1],
       [1, 4, 1]])
>>> y
array([[4, 4, 3],
       [2, 2, 4]])

>>> -x
array([[-2, -3, -1],
       [-1, -4, -1]])
>>> x + y
array([[6, 7, 4],
       [3, 6, 5]])

>>> x - y
array([[-2, -1, -2],
       [-1,  2, -3]])
```

```
>>> x / y
array([[0.5   , 0.75  , 0.33333333],
       [0.5   , 2.    , 0.25      ]])

>>> x // y
array([[0, 0, 0],
       [0, 2, 0]], dtype=int32)

>>> x * y
array([[ 8, 12,  3],
       [ 2,  8,  4]])

>>> x ** y
array([[16, 81,  1],
       [ 1, 16,  1]], dtype=int32)

>>> x % y
array([[2, 3, 1],
       [1, 0, 1]], dtype=int32)
```

关系运算与逻辑运算(bitwise)

Operator	Equivalent ufunc
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

Operator	Equivalent ufunc
&	np.bitwise_and
|	np.bitwise_or
^	np.bitwise_xor
~	np.bitwise_not

关系运算

```
>>> x = np.arange(4)
>>> x
array([0, 1, 2, 3])

>>> x<=2
array([ True,  True,  True, False])

>>> x>3
array([False, False, False, False])

>>> x != 2
array([ True,  True, False,  True])

>>> x*2 == x**2
array([ True, False,  True, False])

>>> np.sqrt(x) == x
array([ True,  True, False, False])
```

```
>>> x = np.random.randint(-5,5,(2,3))
>>> y = np.random.randint(-5,5,(2,3))
>>> x
array([[ 3, -2, -4],
       [-5, -5,  2]])
>>> y
array([[ 1, -3, -1],
       [ 4,  1, -5]])
>>> x<=y
array([[False, False,  True],
       [ True,  True, False]])
>>> x[x<=y]
array([-4, -5, -5])

>>> (y>0) & (y<x)
array([[ True, False, False],
       [False, False, False]])
```

例：判断闰年

```
>>> year = 2000
>>> ((year % 4 == 0) and (year % 100 != 0)) or (year % 400 == 0)
True
```

```
>>> years = np.arange(1990,2023)
>>> years
array([1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
       2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022])
>>> isLeap = (((years % 4 ==0) & (years % 100 != 0)) | (years % 400 ==0))
>>> years[isLeap]
array([1992, 1996, 2000, 2004, 2008, 2012, 2016, 2020])
```

```
>>> isLeap = (((years % 4 ==0) and (years % 100 != 0)) or (years % 400 ==0))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than one element is ambiguous.
Use a.any() or a.all()
```

例：肤色检测

肤色像素判别规则：20<R-G<80

```
>>> im = cv.imread('0.jpg')
>>> B,G,R = im[:,:,0],im[:,:,1],im[:,:,2]
>>> skinMask = (R-G>20) & (R-G<80)
>>> skinImg = im.copy();
>>> skinImg[~skinMask,:] = 0
```

np.where

C++:

```
int x = 2;
int y = x>0?1:-1;
```

numpy:

```
>>> x = np.random.randn(3,2)
>>> x
array([[ 0.40217239, -0.29639506],
       [-1.19111173,  1.38473396],
       [-0.66014868, -0.31229027]])

>>> np.where(x>0,1,-1)
array([[ 1, -1],
       [-1,  1],
       [-1, -1]])
```

```
>>> def relu(x):
...     return np.where(x>0,x,0)
...
>>> x = np.random.randint(-5,5,(2,3))
>>> x
array([[ 0,  2, -4],
       [ 3,  0, -2]])
>>> relu(x)
array([[0, 2, 0],
       [3, 0, 0]])
```

例：肤色检测

肤色像素判别规则：20<R-G<80

```
>>> im = cv.imread('0.jpg')
>>> B,G,R = np.dsplit(im,3)
>>> skinMask = (R-G>20) & (R-G<80)
>>> skinImg = np.where(skinMask,im,0)
```

就地操作(in-place operation)

```
>>> x = np.random.randint(-5,5,(1,6))
>>> x
array([[-5,  2,  0, -3, -3, -5]])

>>> y = np.random.randint(-5,5,(1,6))
>>> y
array([[ 1, -4,  4,  1,  1, -3]])

>>> z = x + y
>>> z
array([[-4, -2,  4, -2, -2, -8]])

>>> np.add(x,y,out = y)
array([[-4, -2,  4, -2, -2, -8]])
>>> y
array([[-4, -2,  4, -2, -2, -8]])
```

```
>>> x = np.arange(8).reshape((2,4))
>>> x
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

>>> np.power(2,x[0],out=x[1])
array([1, 2, 4, 8])
>>> x
array([[0, 1, 2, 3],
       [1, 2, 4, 8]])
```

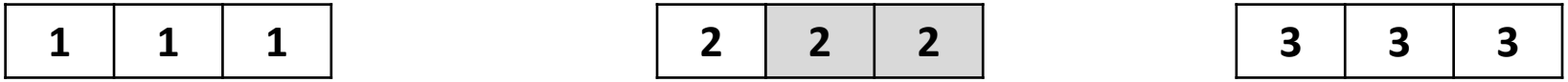
数组运算中的广播(BroadCasting)

广播(BroadCasting)

```
>>> x = np.random.randint(-5,5,(2,3))
>>> x
array([[-5,  2,  3],
       [-1,  1,  3]])

>>> y = np.random.randint(-5,5,(2,1))
>>> y
array([[-1],
       [ 0]])
```

```
>>> x + y
array([[-6,  1,  2],
       [-1,  1,  3]])

>>> x + 2
array([[-3,  4,  5],
       [ 1,  3,  5]])
```

广播(BroadCasting)



广播(BroadCasting)

1	1	1
1	1	1
1	1	1

0	1	2
0	1	2
0	1	2

1	2	3
1	2	3
1	2	3

广播(BroadCasting)

1	1	1
1	1	1
1	1	1

0	1	2
0	1	2
0	1	2

1	2	3
1	2	3
1	2	3

广播(BraoadCasting)

```
>>> x = np.arange(12).reshape(3,4)
>>> y = np.arange(12).reshape(4,3)
>>> x + y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (3,4) (4,3)
```

例：彩色图像灰度化

$$GRAY = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

```
>>> im = cv.imread('sunflower.jpg').astype(cv.float32)/255
>>> W = np.array([0.1140, 0.5870, 0.2989])
>>> gray = (im * W).sum(axis = 2)
```

数组元素的聚合运算

Function Name	NaN-safe Version	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute median of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any	N/A	Evaluate whether any elements are true
np.all	N/A	Evaluate whether all elements are true

min,max,sum

```
>>> x = np.random.randint(0,10,(5,10))
>>> x
array([[2, 7, 6, 8, 5, 1, 2, 7, 2, 7],
       [8, 2, 3, 4, 6, 9, 6, 3, 0, 5],
       [6, 2, 1, 8, 5, 7, 3, 0, 9, 6],
       [6, 3, 7, 5, 6, 8, 8, 8, 1, 6],
       [7, 3, 8, 1, 4, 0, 2, 6, 8, 1]])
>>> x.sum()
238
>>> x.sum(axis = 0)
array([29, 17, 25, 26, 26, 25, 21, 24, 20, 25])
>>> x.sum(axis = 0).ndim
1
>>> x.sum(axis = 0, keepdims = True)
array([[29, 17, 25, 26, 26, 25, 21, 24, 20, 25]])
```

np.maximum, np.minimum

```
>>> x = np.random.randint(0,10,(3,10))
>>> y = np.random.randint(0,10,(3,10))
>>> x
array([[0, 6, 3, 5, 2, 4, 8, 2, 7, 7],
       [0, 0, 5, 5, 6, 0, 8, 9, 2, 4],
       [8, 6, 9, 9, 5, 5, 0, 1, 4, 0]])
>>> y
array([[0, 6, 8, 7, 8, 8, 9, 0, 1, 9],
       [9, 3, 1, 9, 9, 0, 2, 7, 7, 8],
       [4, 0, 5, 9, 1, 4, 8, 9, 9, 3]])
>>> z = np.maximum(x,y)
>>> z
array([[0, 6, 8, 7, 8, 8, 9, 2, 7, 9],
       [9, 3, 5, 9, 9, 0, 8, 9, 7, 8],
       [8, 6, 9, 9, 5, 5, 8, 9, 9, 3]])
```

prod

```
>>> x = np.random.randint(1,5,(2,3))
>>> x
array([[4, 1, 2],
       [2, 3, 2]])

>>> x.prod()
96
>>> x.prod(axis = 0)
array([8, 3, 4])
>>> x.prod(axis = 1)
array([ 8, 12])
>>> x.prod(initial = 5)
480
```

argmin,argmax

```
>>> x = np.random.randint(1,5,(3,5))
>>> x
array([[3, 4, 1, 1, 1],
       [1, 4, 4, 3, 3],
       [1, 4, 1, 1, 1]])

>>> x.argmin(axis = 0)
array([1, 0, 0, 0, 0], dtype=int64)
>>> x.argmax(axis = 1)
array([1, 1, 1], dtype=int64)
>>> x.argmin()
2
```

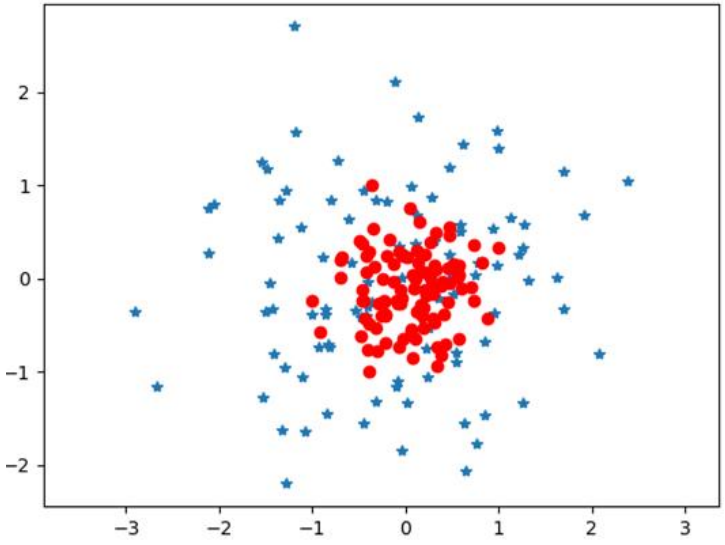
any,all

```
>>> x = np.random.randint(-5,5,(3,5))
>>> x
array([[-4, -1, -5,  1,  0],
       [ 1, -5,  0, -5, -3],
       [-5, -3,  4, -4,  4]])
>>> x.any()
True
>>> x.all()
False

>>> (x>0).any()
True
>>> (x>0).all()
False
```

例：特征归一化

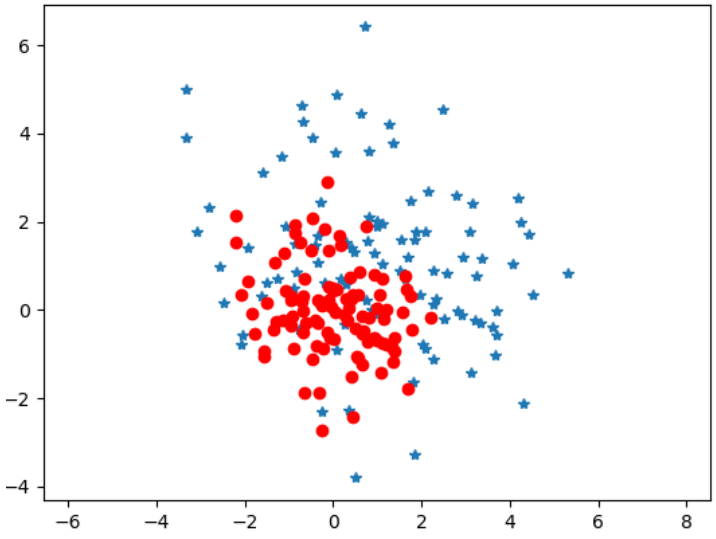
```
>>> x = np.random.randn(2,100)
```

```
>>> minx = x.min(axis = 1, keepdims = True)
>>> maxx = x.max(axis = 1, keepdims = True)
>>> minx
array([[-2.90092376],
       [-2.20297544]])
>>> maxx
array([[2.38442347],
       [2.70580991]])
>>> y = 2*(x - minx)/(maxx - minx) - 1
>>> y.min(axis = 1)
array([-1., -1.])
>>> y.max(axis = 1)
array([1., 1.])
```



例：特征标准化

```
>>> x = np.random.randn(2,100)*2 + 1
>>> std = x.std(axis = 1, keepdims = True)
>>> std
array([[1.96023919],
       [1.81613181]])
>>> miu = x.mean(axis = 1,keepdims = True)
>>> miu
array([[0.99654516],
       [1.13055334]])

>>> y = (x - miu) / std
>>> y.std(axis = 1)
array([1., 1.])
>>> y.mean(axis = 1)
array([-2.22044605e-18,  3.08086889e-17])
```



矩阵乘积运算

np.dot, @

```
>>> x = np.random.rand(2,3)
>>> y = np.random.rand(3,4)
>>> x.dot(y)
array([[0.65404399, 0.51001177, 0.2614941 , 0.15208298],
       [0.98426831, 0.93180514, 0.5526937 , 0.51949789]])

>>> x@y
array([[0.65404399, 0.51001177, 0.2614941 , 0.15208298],
       [0.98426831, 0.93180514, 0.5526937 , 0.51949789]])

>>> np.dot(x,y)
array([[0.65404399, 0.51001177, 0.2614941 , 0.15208298],
       [0.98426831, 0.93180514, 0.5526937 , 0.51949789]])
```

例：sigmoid神经元

```
>>> def sigmoid(z):
...     return 1 / (1 + np.exp(-z))
...
>>> def sigmoidCell(X,W,b):
...     Z = W.T @ X + b
...     return sigmoid(Z)
...
>>> X = np.random.randn(2,5)
>>> W = np.random.randn(2,1)
>>> b = 0
>>> rho = sigmoidCell(X,W,b)
>>> rho
array([[0.71146343, 0.88812752, 0.28855789, 0.43348845, 0.68182186]])
```

例：softmax神经元

```
>>> def softmax(z):
...     z = z - z.max(axis = 0,keepdims = True)
...     expz = np.exp(z)
...     rho = expz / expz.sum(axis = 0,keepdims = True)
...     return rho
...
>>> def softmaxCell(X,W,b):
...     Z = W.T@X + b
...     return softmax(Z)
...
>>> X = np.random.randn(2,5)
>>> W = np.random.randn(2,3)
>>> b = np.zeros((3,1))
>>> rho = softmaxCell(X,W,b)
>>> rho
array([[0.38445404, 0.87811646, 0.08823561, 0.04816973, 0.83945355],
       [0.0857069 , 0.00572758, 0.72731163, 0.34196217, 0.05975182],
       [0.52983905, 0.11615596, 0.18445276, 0.6098681 , 0.10079462]])
>>> rho.sum(axis = 0)
array([1., 1., 1., 1., 1.])
```

编写神经元模型

sigmoid神经元

```
class SigmoidNeuron:
    def __init__(self, in_features):
        self.in_features = in_features
        self.W = np.zeros((self.in_features,1))
        self.b = 0

    def sigmoid(z):
        return 1/(1 + np.exp(-z))

    def predict(self, x):
        z = self.W.T @ x + self.b
        a = SigmoidNeuron.sigmoid(z)
        return a

    def evaluate(self, x,y):
        rho = self.predict(x)
        yhat = np.where(rho>=0.5,1,0)
        err = (yhat!=y).astype('float').mean()
        return err
```

sigmoid神经元

```
def fit(self, X, Y, lr = 0.1):
    self.W, self.b = np.zeros((self.in_features,1)),0 #initialize params
    loss0 = np.Inf
    epsilon = 1e-6

    while(True):
        #prediction
        rho = self.predict(X) #1 x n
        #loss
        loss = -(np.log(rho[Y==1]).sum() + np.log(1 - rho[Y==0]).sum())/X.shape[1]
        if np.abs(loss - loss0) < epsilon:
            break
        loss0 = loss
        #error:
        e = rho - Y#1 x n
        #dw,db:
        dw = X @ e.T /n
        db = e.mean()
        #update params:
        self.W = self.W - lr * dw
        self.b = self.b - lr * db
```

测试SigmoidNeuron

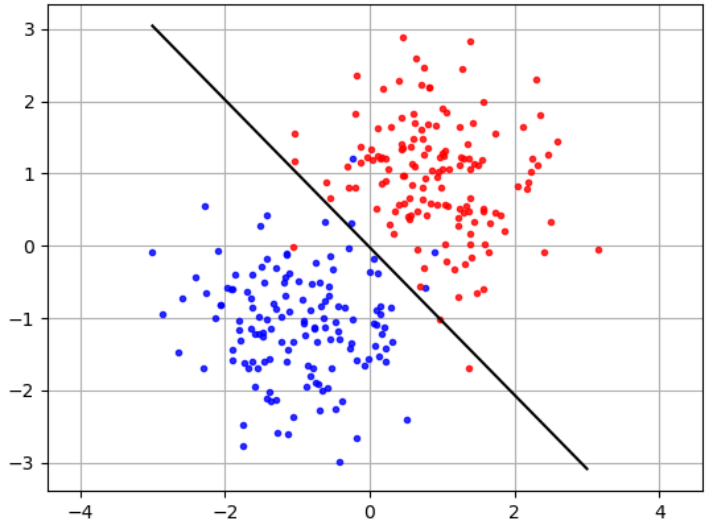
```
cell = SigmoidNeuron(in_features = 5)
print('W = ',cell.W)
print('b = ',cell.b)

x = np.random.randn(5,10)
a = cell.predict(x)
print(a)
```

```
W =  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
b =  0
[[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]]
```

二维点云分类

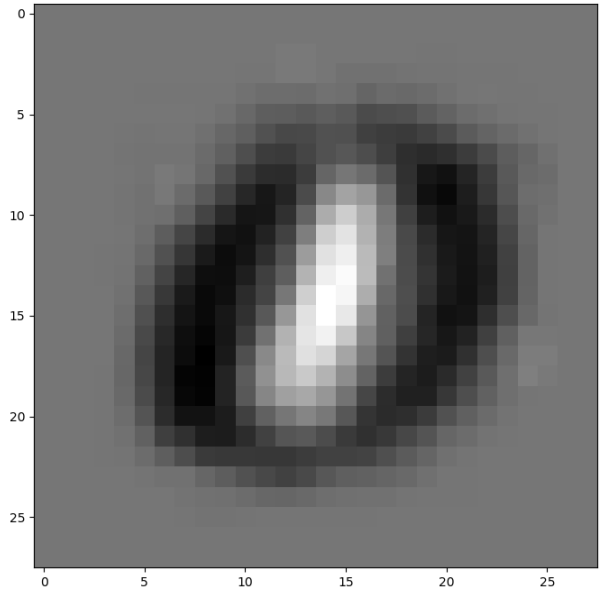
```
#生成一组2维随机样本
n = 200
x1 = np.random.normal([[1],[1]],0.8,(2,n))
x0 = np.random.normal([[-1],[-1]],0.8,(2,n))
x = np.hstack((x0,x1))
y = np.hstack((np.zeros((n,)),np.ones((n,))))

x_train,y_train,x_test,y_test = splitData(x,y)

lr = SigmoidNeuron(in_features = 2)
lr.fit(x_train,y_train,lr=1)

#计算训练误差:
err = lr.evaluate(x_train,y_train)
print('training error = %.2f%%'%(err*100))

#计算测试误差
err = lr.evaluate(x_test,y_test)
print('test error = %.2f%%'%(err*100))
```

```
training error = 2.14%
test error = 2.50%
```



MNIST 0-1分类

```
def testMNIST():
    data = pd.read_csv('data/MNIST/mnist_test.csv').to_numpy().astype(np.float32)
    y,x = data[:,0],data[:,1:]
    zero_ones = y<2
    x = x[zero_ones,:].T
    y = y[zero_ones]

    x = 2*x/255 - 1

    x_train,y_train,x_test,y_test = splitData(x,y)

    lr = SigmoidNeuron(in_features = x_train.shape[0])
    lr.fit(x_train,y_train,lr=1)

    #计算训练误差:
    err = lr.evaluate(x_train,y_train)
    print('training error = %.2f%%'%(err*100))

    #计算测试误差
    err = lr.evaluate(x_test,y_test)
    print('test error = %.2f%%'%(err*100))
```




```
training error = 0.00%
test error = 0.00%
```

乳腺癌细胞识别

Wisconsin Diagnostic Breast Cancer, WDBC数据:
baohan 569个样本（恶性212，良性357），每个样本有三十个特征。

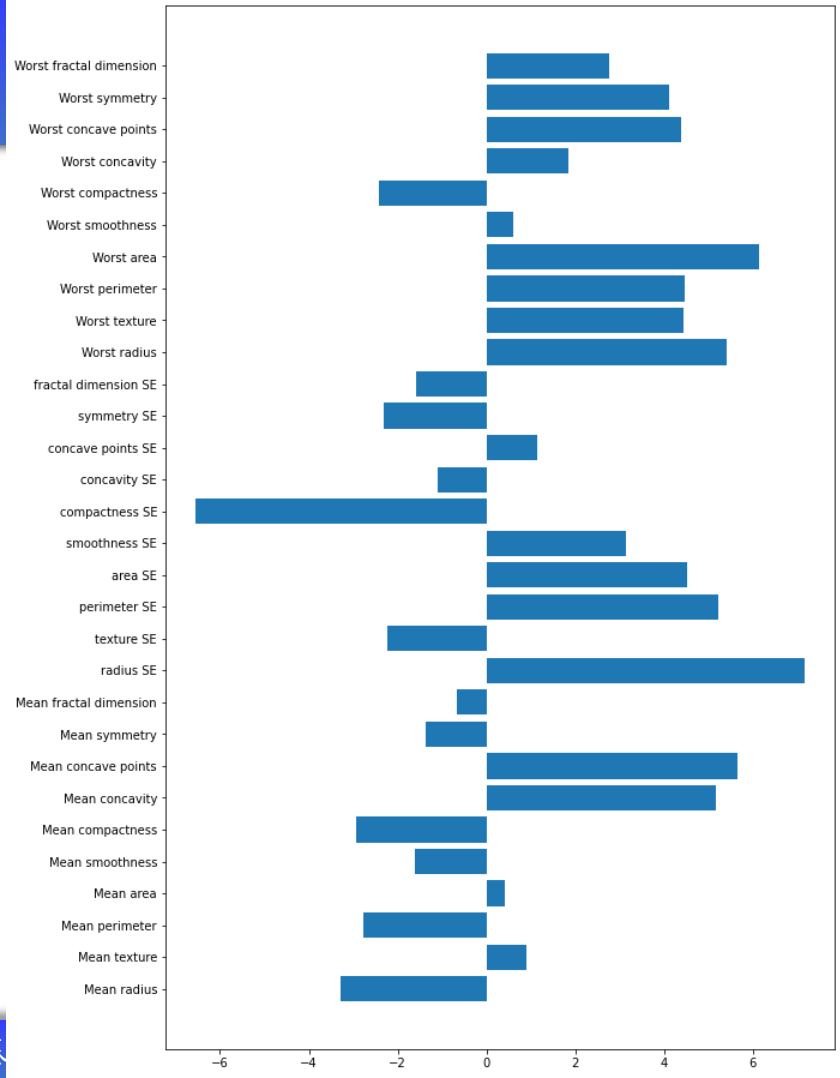
```
import pandas as pd
df = pd.read_csv('data/breast-cancer-wisconsin/wdbc.data')
data = np.vstack((df.columns.to_numpy(),df.to_numpy()))
print(data.shape)
print(data[0])
```

```
(569, 32)
['842302' 'M' '17.99' '10.38' '122.8' '1001' '0.1184' '0.2776' '0.3001'
 '0.1471' '0.2419' '0.07871' '1.095' '0.9053' '8.589' '153.4' '0.006399'
 '0.04904' '0.05373' '0.01587' '0.03003' '0.006193' '25.38' '17.33'
 '184.6' '2019' '0.1622' '0.6656' '0.7119' '0.2654' '0.4601' '0.1189']
```

乳腺癌细胞识别

```
y = np.where(data[:,1]=='M',1.0,0.0).T
x = data[:,2:].astype(np.float32).T

#归一化到[-1,1]区间
maxV = x.max(axis = 1, keepdims = True)
minV= x.min(axis = 1, keepdims = True)
x = 2*(x - minV) / (maxV - minV)

lr = SigmoidNeuron(in_features = x.shape[0])
lr.fit(x_train,y_train, lr = 1)

err = lr.evaluateErrorRate(x_train,y_train)
print('training error = %.2f%%'%(err*100))

err = lr.evaluateErrorRate(x_test,y_test)
print('test error = %.2f%%'%(err*100))
```

```
training error = 1.26%
test error = 1.75%
```



练习

- 1.编写一个用于线性回归的神经元模型，并用于拟合以下线性方程：$y = 2x + 1$。请根据该方程生成带噪声的训练样本，用于训练你编写的模型。
- 2. 用上面的神经元模型，拟合以下多项式方程:$y = x^3 + 2x^2 + x - 1$。请根据该方程生成带噪声的训练样本，用于训练你编写的模型。
- 3.编写一个Softmax回归器模型，并用于识别MNIST手写体数字。手写体数字样本见mnist_train.csv, mnist_test.csv。