Numpy与金融应用

主要内容：

- Numpy 库、数组的性质、创建 array 数组；
- 数组的索引切片与赋值；
- 数组的常用操作、数组的运算；
- Numpy 矩阵。

1. Numpy 库

Numpy 是 Python 用于高性能科学计算和数据分析的扩展包，主要用来存储和处理多维**数组（矩阵）**，比 Python 自身内置的列表结构要高效很多，其本身由 C 语言开发，是一个非常基础的扩展包，也是 Python 其它许多高级工具的构建基础。

- **ndarray**，一个具有矢量算术运算和复杂广播能力的快速且省空间的多维数组；
- 有许多对数组数据进行快速运算的标准数学函数，矩阵运算无需编写循环；
- 常用的线性代数，傅里叶变换和随机数生成操作。

In [2]:

```
# import numpy
import numpy as np    # 导入 numpy 包
```

2. 数组是有形状的数据结构

数组是有形的，对应维度（dimension）和形状（shape）2个属性：

- 通过 arr.ndim 查看维度，各个维度对应一个轴（axis），**第1维度对应 axis=0，第2维度对应 axis=1，第3维度对应 axis=2，...，第 n 维度对应 axis=n-1**；
- 通过 arr.shape 查看形状；
- 通过 arr.reshape((d1,d2,d3,...,dn)) 来重塑数组的形状，其中的 dn 对应第 n 维中的元素（或元素集合）的个数。

2.1 一维数组

In [3]:

```
# 创建一维数组
a = np.arange(16)
a
```

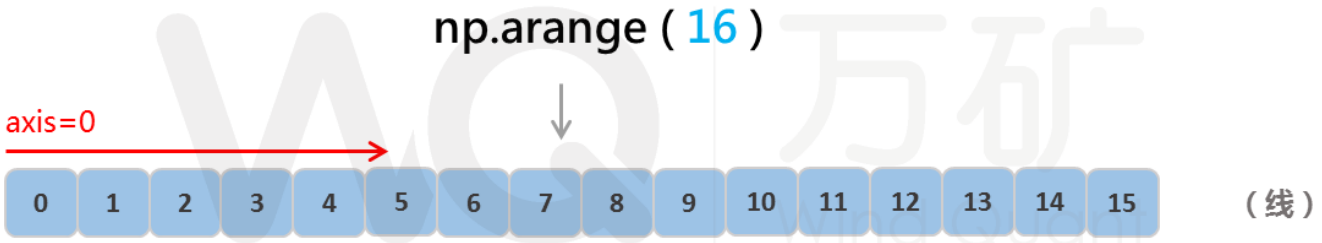
Out[3]:

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

In [4]:

```
print("维度为: ", a.ndim)
print("形状为: ", a.shape)
```

```
维度为:  1
形状为:  (16,)
```



2.2 二维数组

In [5]:

```
# 创建二维数组
b = np.arange(12).reshape((3,4))
b
```

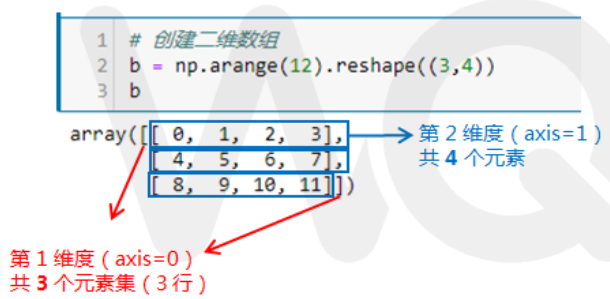
Out[5]:

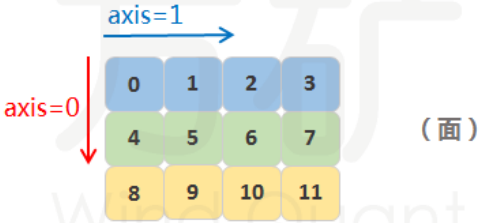
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [6]:

```
# 查看维度和形状
print("维度为: ", b.ndim)
print("形状为: ", b.shape)
```

```
维度为:  2
形状为:  (3, 4)
```





2.3 三维数组

In [9]:

```
# 基于 np.arange(16) 创建 3 维数组
c = np.arange(24).reshape((2,3,4))
c
```

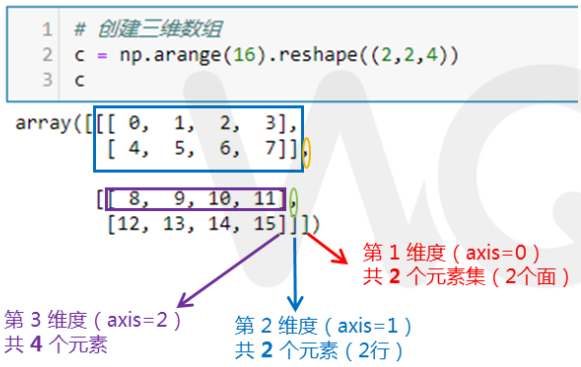
Out[9]:

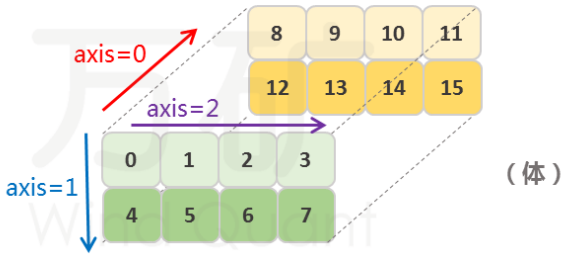
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

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

In [10]:

```
# 查看维度和形状
print("维度为: ", c.ndim)
print("形状为: ", c.shape)
```

```
维度为:  3
形状为:  (2, 3, 4)
```





(体)

2.4 四维数组

In [11]:

```
# 创建四维数组
d = np.arange(32).reshape((2,2,2,4))
d
```

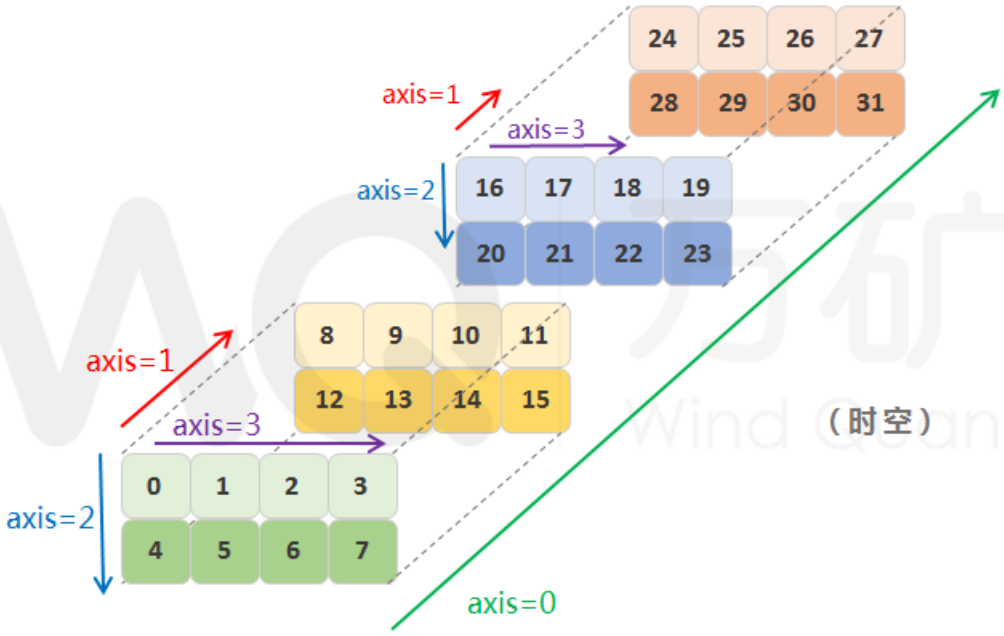
Out[11]:

```
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]]]])
```

In [146]:

```
# 查看维度和形状
print("维度为: ", d.ndim)
print("形状为: ", d.shape)
```

```
维度为:  4
形状为:  (2, 2, 2, 4)
```



3. 创建 array 数组

数组中只包含单一的数据类型，一共有3种常用的数组创建方法：

方法1：使用 `np.array()` 创建数组。

```
numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
```

参数说明：

参数	类型	说明
object	array_like	数组或嵌套的序列，如列表、元组。
dtype	option	数据类型，若为None，类型将被确定为保持序列中的对象所需的最小类型。
copy	bool	是否对object的相关属性进行复制。
order	str	数组内存块的存储方式，共{'K', 'A', 'C', 'F'}四种取值，其中'C'代表C语言的行主序（默认），'F'代表Fortran语言的列主序。
subok	bool	如果为True，则子类将被传递，否则返回的数组将被强制为默认的基类数组。
ndmin	int	用于设置最小维度。

In [12]:

```
# 基于列表创建一维数组
a = [1,2.5,3,4,5] # 注意：数组中只包含单一的数据类型！！！！
np.array(a)
```

Out[12]:

```
array([1. , 2.5, 3. , 4. , 5. ])
```

In [14]:

```
np.array(a,ndmin = 2)
```

Out[14]:

```
array([[1. , 2.5, 3. , 4. , 5. ]])
```

In [15]:

```
# 基于列表创建多维数组

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

Out[15]:

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

In [16]:

```
# 基于元组创建数组
a = (1,2,3,4,5)
np.array(a)
```

Out[16]:

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

In [10]:

```
# 基于元组创建多维数组

```

In [17]:

```
#  创建数组时，可以设置数据类型
a = (1,2,3,4,0)
np.array(a, dtype = 'bool')
```

Out[17]:

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

In [18]:

```
# 设置返回类型
np.array(np.mat('1 2; 3 4'), subok=True)
```

Out[18]:

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

方法2：使用 np.arange()（定隔）和 np.linspace()、np.logspace()（定点）快速生成数组

- 固定数组中各元素之间的步长：

```
np.arange(start,stop,step,dtype)  # 创建等差数列，start为起点（默认值为0），stop为终点（必填，无默认值），step为步长（默认值为1）
```

- 固定某一取值范围内的元素个数：

```
np.linspace(start,stop,num,endpoint,retstep,dtype)  # start为起点（必填，无默认值），stop为终点（必填，无默认值），num为元素个数（默认值为50）
np.logspace(start,stop,num,endpoint,base,dtype)
```

In [19]:

```
print(np.arange(9))   # 从0开始，但不包括 stop 点
print(np.arange(2,9))
print(np.arange(2,9,2))  # 注：使用 print 打印数组显示的形式，没有 逗号 和 array()
```

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

In [ ]:

```
# 生成 0-1 步长为 0.1 的数组
```

In [49]:

```
print(np.linspace(2,9))    #  默认元素个数为 50
print(np.linspace(2,9,11))
```

```
[2.         2.14285714 2.28571429 2.42857143 2.57142857 2.71428571
 2.85714286 3.         3.14285714 3.28571429 3.42857143 3.57142857
 3.71428571 3.85714286 4.         4.14285714 4.28571429 4.42857143
 4.57142857 4.71428571 4.85714286 5.         5.14285714 5.28571429
 5.42857143 5.57142857 5.71428571 5.85714286 6.         6.14285714
 6.28571429 6.42857143 6.57142857 6.71428571 6.85714286 7.
 7.14285714 7.28571429 7.42857143 7.57142857 7.71428571 7.85714286
 8.         8.14285714 8.28571429 8.42857143 8.57142857 8.71428571
 8.85714286 9.        ]
[2.  2.7 3.4 4.1 4.8 5.5 6.2 6.9 7.6 8.3 9. ]
```

In [20]:

```
print(np.logspace(0,9,5))
```

```
[1.00000000e+00 1.77827941e+02 3.16227766e+04 5.62341325e+06
 1.00000000e+09]
```

方法3：通过 Numpy 中的 `random` 模块快速创建随机数组

- **调用方式：**

```
import numpy as np
np.random.seed()    # 方式1：通过 np.random.func() 形式调用

from numpy import random  # 直接从 numpy 库中导 random 模块
random.seed()    # 方式2： 通过 random.func() 形式调用
```

- **常用的函数：**

函数	说明
random	生成 [0.0, 1.0) 范围内的浮点随机数数组。
seed	随机数生成器的种子
permutation	序列的随机排列或者随机排列的范围，不改变原数组
shuffle	序列就地随机排列，改变原数组
rand	均匀分布样本值
randint	给定上下限随机产生整数
randn	正态分布样本值
binomial	二项分布样本值
normal	正态分布样本值
beta	beta分布样本值
chisquare	卡方分布样本值
gamma	Gamma 分布样本值
uniform	[0,1) 均匀分布样本值
choice	从数组中随机选择若干个元素

- 生成的随机数可能会发生变化，可通过seed设置随机数种子来固定生成的随机数；
- 生成随机数时，注意数据的取值范围，数据类型（整型\浮点型）；
- 不同的分布，选择不同的生成方法；
- 生成随机数的方法很多，可通过 **http://numpy.org (http://numpy.org)** 中的说明文档学习相应的使用方法。

In [73]:

```
np.random.normal(0, 1, 10)    # 生成10个正态分布随机数
```

Out[73]:

```
array([ 1.93226028, -0.66253191,  0.29003299,  0.3408019 ,  0.00258583,
        1.09485095, -0.92478678,  2.0710993 , -0.63277521,  0.29992387])
```

随机种子是一个用来生成随机数的一个数，在计算机中是一个无符号整形数 ，生成随机数的算法一定，随机种子一定，生成的随机数就不会变。

In [22]:

```
np.random.seed(1676)  # 设置随机种子数
np.random.random(5)  # 生成的随机数固定不变
```

Out[22]:

```
array([0.39983389, 0.29426895, 0.89541728, 0.71807369, 0.3531823 ])
```

In [24]:

```
np.random.random(5)  # 随机种子只会作用在一个函数上
```

Out[24]:

```
array([0.11451688, 0.74809154, 0.89618379, 0.4089545 , 0.87943532])
```

In [28]:

```
np.random.seed(1676)  # 设置随机种子数
np.random.random(5)  # 生成的随机数固定不变
```

Out[28]:

```
array([0.92866929, 0.40003106, 0.57037287, 0.70225734, 0.51095069])
```

方法4： 创建元素值初始化的数组

方法	说明
np.zeros((d1,d2,d3,...))	创建元素全为0的n维数组，d1,d2,d3,... 代表各维度的元素个数。
np.ones((d1,d2,d3,...))	创建元素全为1的n维数组，d1,d2,d3,... 代表各维度的元素个数。
np.empty((d1,d2,d3,...))	创建空的n维数组，取值无意义，d1,d2,d3,... 代表各维度的元素个数。
np.eye(N,M,k=0)	创建主对角线为1的NxM对角矩阵，参数k用于设置对角线的位置，k=0 代表落在中间对角线上，k=1 代表落在对角线右上方，k=-1 代表落在对角线左上方。
np.identity(n)	创建一个主对角线为 1 其他为 0 的n方阵。

In [171]:

```
print("元素全为0的1维数组：\n",np.zeros(6),'\n')
print("元素全为1的2维数组：\n",np.ones((2,2)),'\n')
print("元素为空（取值无效）的3维数组：\n",np.empty((3,3,4)),'\n')
print("3x2对角方阵：\n",np.eye(3,2),'\n')
print("3维对角方阵：\n",np.identity(3))
```

```
元素全为0的1维数组：
 [0. 0. 0. 0. 0. 0.]

元素全为1的2维数组：
 [[1. 1.]
 [1. 1.]]

元素为空（取值无效）的3维数组：
 [[[1.29501088e-316 6.91254172e-310 6.91254172e-310 6.91254172e-310]
  [6.91254171e-310 6.91254171e-310 6.91254171e-310 6.91254171e-310]
  [6.91254171e-310 6.91254171e-310 6.91254172e-310 6.91254172e-310]]

 [[6.91254172e-310 6.91254172e-310 6.91254172e-310 6.91254172e-310]
  [6.91254172e-310 6.91254172e-310 6.91254172e-310 6.91254172e-310]
  [6.91254172e-310 6.91254172e-310 6.91254172e-310 6.91254172e-310]]

 [[6.91254170e-310 6.91254170e-310 6.91254168e-310 6.91254168e-310]
  [6.91254169e-310 6.91254169e-310 6.91254169e-310 6.91254169e-310]
  [6.91254169e-310 6.91254169e-310 6.91254169e-310 6.91254169e-310]]]

3x2对角方阵：
 [[1. 0.]
 [0. 1.]
 [0. 0.]]

3维对角方阵：
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

例1. 通过 API 函数获取科创板行情数据，并生成数组形式

In [15]:

```
from WindPy import *  # 第一步：导入 WindPy 模块
w.start() # 第二步：启动 API 接口
```

```
COPYRIGHT (C) 2017 Wind Information Co., Ltd. ALL RIGHTS RESERVED.
IN NO CIRCUMSTANCE SHALL WIND BE RESPONSIBLE FOR ANY DAMAGES OR LOSSES
CAUSED BY USING WIND QUANT API FOR PYTHON.
```

In [16]:

```
# 通过 API函数提取科创板的行情数据
datas =  w.wsd("688001.SH,688002.SH,688003.SH,688005.SH", "pct_chg", "2019-07-22", "2019-07-26",
"Fill=Previous;PriceAdj=F")
datas
```

Out[16]:

```
.ErrorCode=0
.RequestID=315
.Codes=[688001.SH,688002.SH,688003.SH,688005.SH]
.Fields=[PCT_CHG]
.Times=[20190722,20190723,20190724,20190725,20190726]
.Data=[[128.77164056059357,-12.198198198198204,5.725425815719283,6.191770186335399,-
7.4026686816340713],[151.00000000000003,-12.569721115537853,6.447938026885391,5.029965753424661,-7.8255
5532912167],[85.88235294117645,-14.556962025316453,8.395061728395058,15.034168564920277,-9.50495049504
95],[86.06311044327572,-15.808600847970922,6.378896882493996,10.27953110910731,-8.646770237712184]]
```

In [17]:

```
# 创建 行情数据的数组
arr = np.array(datas.Data)
arr = arr.round(2)
arr
```

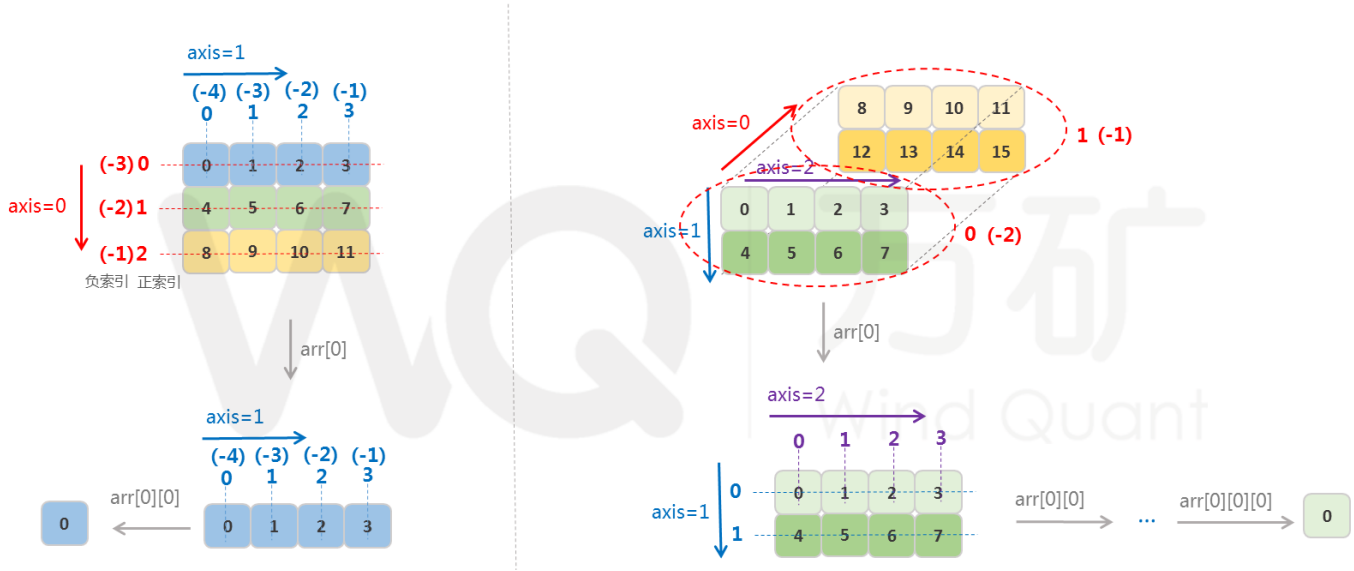
Out[17]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

4. 数组的索引切片与赋值

4.1 索引

获取一个特定位置的元素， 通过arr[下标][下标]...或arr[下标，下标，下标，...]形式进行层层索引操作，**数组是有维度的，每个axis 轴上都会有相应的索引号。**



In [291]:

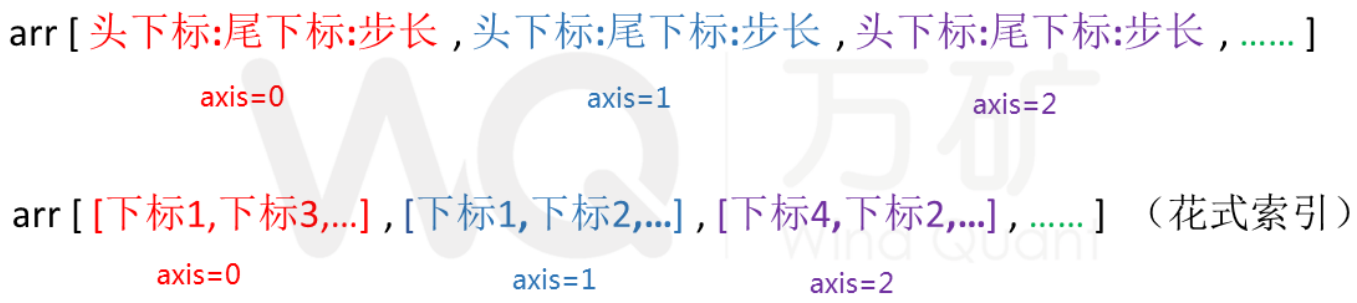
```
# 提取arr中最后一行（最后一只股票）收益率（轴0对应的最后一个元素）
```

In [ ]:

```
# 提取最后一行最后一日的收益率
```

4.2 切片

切片一般是用于获取一段位置上的元素，用arr[头下标：尾下标]、arr[头下标：尾下标：步长]的形式进行切片， 包含头下标，但不包含尾下标，多维数组切片形式如下：



In [292]:

```
# 提取arr中前两行（前两只股票）收益率
arr[:2]
```

Out[292]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83]])
```

In [294]:

```
# 提取 arr 中前两只股票最后两天的收益率
arr[:2,-2:]
```

Out[294]:

```
array([[ 6.19, -7.4 ],
       [ 5.03, -7.83]])
```

In [298]:

```
# 花式索引，只取其中具体的某几个位置的元素
arr[[1,2],[0,3]]
```

Out[298]:

```
array([151.  ,  15.03])
```

In [300]:

```
arr[:,[4,3,2,1,0]]    # 将列倒着排列
```

Out[300]:

```
array([[ -7.4 ,   6.19,   5.73, -12.2 , 128.77],
       [ -7.83,   5.03,   6.45, -12.57, 151.  ],
       [ -9.5 ,  15.03,   8.4 , -14.56,  85.88],
       [ -8.65,  10.28,   6.38, -15.81,  86.06]])
```

问：在不知道元素位置的情况下，如何进行索引切片？

- **例**：想截取股票 '688002.SH' 在 '2019-07-22' 的收益率序列。

4.3 布尔型索引

- 布尔型索引的索引号为bool类型，**False所在位置上的元素不会被截取，True所对应的元素会被截取**。
- **布尔型数组可以被用于索引**，但数组的长度必须与被索引轴长度一致。
- 可将布尔型数组与切片、整数混合使用。

In [18]:

```
arr
```

Out[18]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

In [123]:

```
# 第一步  将提取的数据中的代码和日期转化为  数组形式
codes = np.array(datas.Codes)
times = np.array([i.strftime('%Y-%m-%d') for i in datas.Times])
```

In [90]:

```
# 第二步  生成布尔型数组，作为布尔索引
codes == '688002.SH'
```

Out[90]:

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

In [113]:

```
# 第三步  根据生成的索引进行切片
arr[codes == '688002.SH']
```

Out[113]:

```
array([[151.  , -12.57,   6.45,   5.03,  -7.83]])
```

In [126]:

```
arr[codes == '688002.SH', times == '2019-07-22']
```

Out[126]:

```
array([151.])
```

In [19]:

```
# 多只股票如何选取 ？ 提示：布尔型数组的或、且逻辑运算需使用 | 和 &符号
```

4.4 赋值

- 索引得到的是原数组的一个复制copy，修改索引中的数据不会改变原数组；
- 但切片得到的是原数组的一个视图view，修改切片的内容，会改变原数组，所以建议先对原数组进行copy，再对copy得到的数组进行切片。

In [306]:

```
test = np.array(range(10))
test
```

Out[306]:

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

In [308]:

```
a = test[-1]
a = 0
```

In [309]:

```
test  #最后一个值没有发生变化
```

Out[309]:

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

In [311]:

```
b = test[2:]
b[-1] = 0
```

In [312]:

```
test   #最后一个元素发生变化，原数组被修改
```

Out[312]:

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

In [315]:

```
test = np.array(range(10))
b = test.copy()[2:]
b[-1] = 0
```

In [316]:

```
test    # copy原数组后，不会修改原数组
```

Out[316]:

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

5. 数组的常用操作

问题1：高维数组能否转化为低维数组？低维数组能否转化为高维数组？

可以进行重塑和打平操作来改变数组的维度：

- **重塑：**

```
numpy.reshape(a,newshape,order='C')  # a 代表需要重塑的数组；newshape指重塑后的形状（d1,d2,d3,...）；order 对应数组内存块存储方式

ndarray.reshape(shape,order='C')  # shape指重塑后的形状；order 对应数组内存块存储方式
```

In [20]:

```
a = np.array(range(12))
a
```

Out[20]:

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

In [21]:

```
# 将一维数组转换为2维数组
np.reshape(a, (3,4))
```

Out[21]:

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

In [23]:

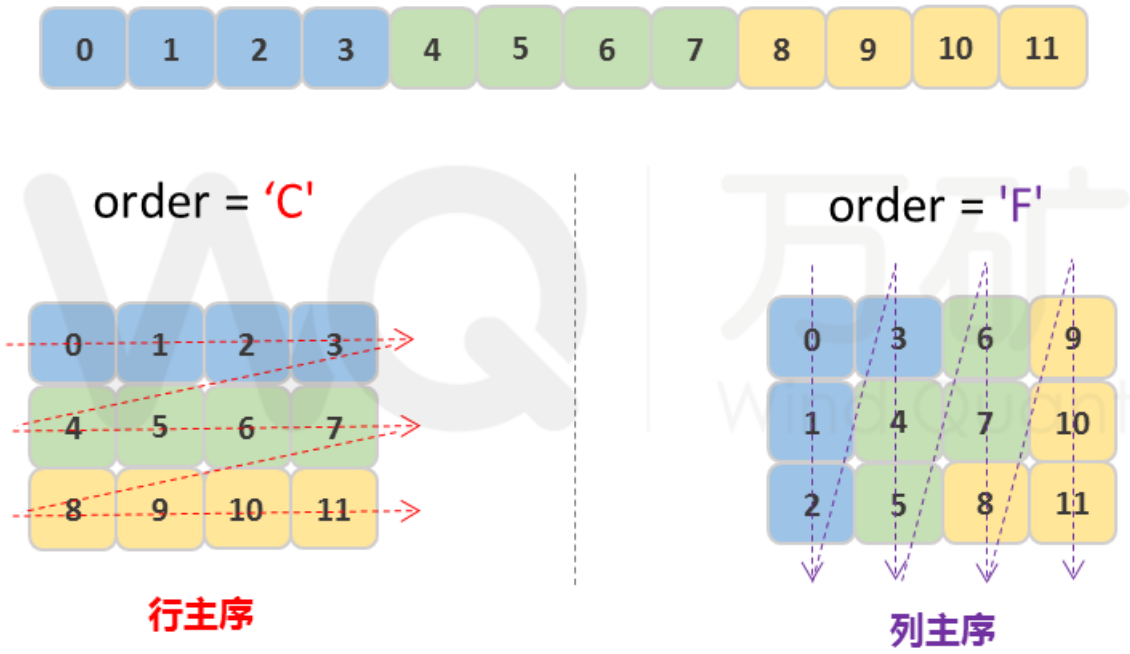
```
# 将一维数组转换为3维数组
a.reshape((2,2,-1))    # 在确定各维度的元素个数时，可以用 -1 代替其中一个维度的元素个数
```

Out[23]:

```
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]]])
```

order参数取值含义

- order = 'C'，代表C语言对应的行主序，即每行元素在内存块中彼此相邻；
- order = 'F'，代表Fortran语言中的列主序，即每列元素在内存块中彼此相邻；
- 数组默认是行主序。



In [322]:

```
np.reshape(a, (3,4))  #数组默认是行主序
```

Out[322]:

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

In [323]:

```
np.reshape(a, (3,4), order = 'F')  #数组默认是行主序
```

Out[323]:

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

- **打平（拉直）：**

```
numpy.ravel(a,order='C')
ndarray.ravel(order='C')  # ravel 按行主序打平时，返回数组的视图view；按列主序打平时，返回的是数组的复制copy

ndarray.flatten(order='C')  # 无论是行主序还是列主序，都复制了原始数组
```

In [344]:

```
b = np.array(range(12)).reshape((3,4))
b
```

Out[344]:

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

In [342]:

```
b = np.array(range(12)).reshape((3,4))
c = b.ravel(order = 'C')
c[-1] = 0
b   # ravel 按行主序打平时，返回数组的视图view，改变打平后的数组会改变原数组
```

Out[342]:

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

In [345]:

```
b = np.array(range(12)).reshape((3,4))
c = b.ravel(order = 'F')
c[-1] = 0
b  # ravel 按列主序打平时，对原数组进行复制，改变打平后的数组 不会改变原数组
```

Out[345]:

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

In [340]:

```
b = np.array(range(12)).reshape((3,4))
c = b.flatten(order = 'F')
c[-1] = 0
b  # 无论是行主序还是列主序，都复制了原始数组
```

Out[340]:

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

In [341]:

```
b = np.array(range(12)).reshape((3,4))
c = b.flatten(order = 'C')
c[-1] = 0
b  # 无论是行主序还是列主序，都复制了原始数组
```

Out[341]:

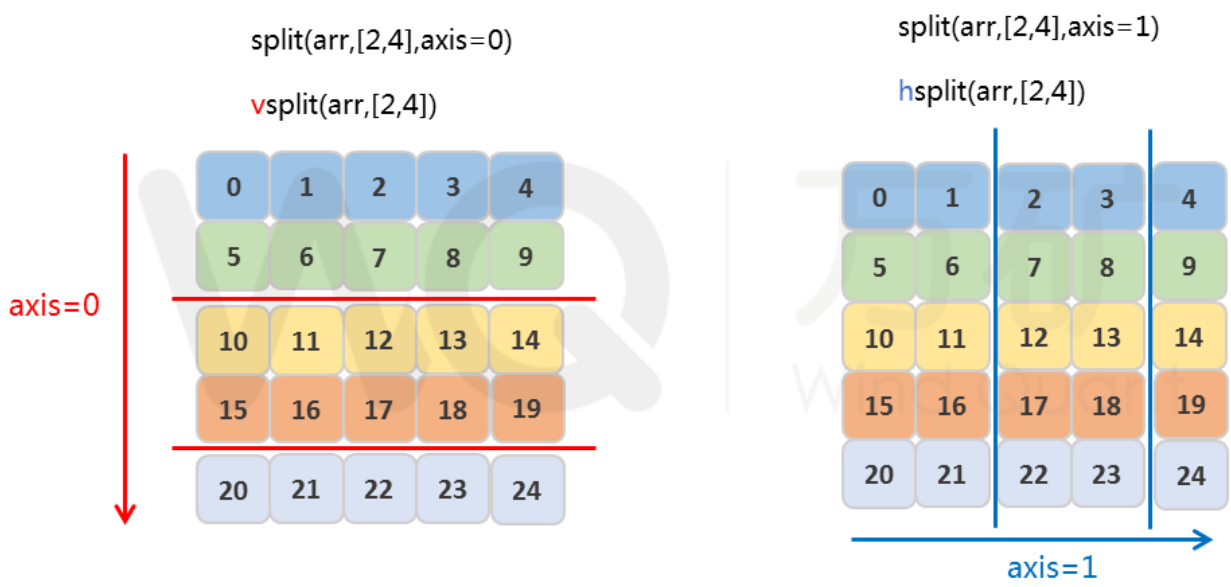
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

问题2： 能否将数组进行拆分或合并

- 分裂：

```
numpy.split(ary,indices_or_section,axis=0)  # ary 代表需分裂的数组；indices_or_section 指分裂的位置；axis指分裂对应的轴
```

方法	说明
numpy.vsplit(ary,indices_or_section)	等价于split(axis=0)，分裂0轴，横向分裂
numpy.hsplit(ary,indices_or_section)	等价于split(axis=1)，分裂1轴，纵向分裂
numpy.dsplit(ary,indices_or_section)	等价于split(axis=2)，分裂2轴，深度分裂



In [348]:

```
arr
```

Out[348]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

In [349]:

```
# 将前2只股票的行情分为一组，后两只分为一组
np.split(arr,2)
```

Out[349]:

```
[array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
        [151.  , -12.57,   6.45,   5.03,  -7.83]]),
 array([[ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
        [ 86.06, -15.81,   6.38,  10.28,  -8.65]])]
```

In [352]:

```
arr1,arr2 = np.vsplit(arr,2)  # 可对分裂后得到的数组进行赋值
arr1
```

Out[352]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83]])
```

In [355]:

```
# 将周1和周2分为一组，周3单独一组，周4和周5分为一组
np.hsplit(arr,[2,3])
```

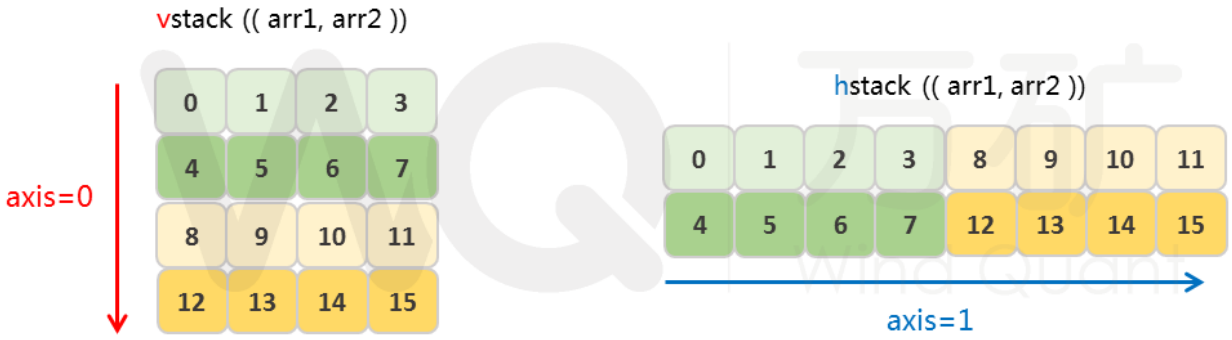
Out[355]:

```
[array([[128.77, -12.2 ],
        [151.  , -12.57],
        [ 85.88, -14.56],
        [ 86.06, -15.81]]), array([[5.73],
        [6.45],
        [8.4 ],
        [6.38]]), array([[ 6.19, -7.4 ],
        [ 5.03, -7.83],
        [15.03, -9.5 ],
        [10.28, -8.65]])]
```

- **合并：**

```
numpy.concatenate((a1,a2,a3,...),axis=0,out=None)  # (a1,a2,a3,...)代表要合并的多个数组；axis指合并对应的轴
```

方法	说明
numpy.vstack((a1,a2,a3,...))	等价于concatenate(axis=0)，合并0轴，竖直合并
numpy.hstack((a1,a2,a3,...))	等价于concatenate(axis=1)，合并1轴，水平合并
numpy.dstack((a1,a2,a3,...))	等价于concatenate(axis=2)，合并2轴，深度合并



- 此外还有 r() *竖直合并和* c() 水平合并的方法。

In [359]:

```
# 通过 API函数提取科创板的行情数据
datas =  w.wsd("688001.SH,688002.SH,688003.SH,688005.SH", "pct_chg", "2019-07-29", "2019-07-30",
"Fill=Previous;PriceAdj=F")
# 创建 行情数据的数组
arr_new = np.array(datas.Data)
arr_new = arr_new.round(2)
arr_new
```

Out[359]:

```
array([[ 9.51,  9.51],
       [10.99, 10.99],
       [ 7.35,  7.35],
       [ 7.07,  7.07]])
```

In [ ]:

```
# 将后两天的行情数据与之前数据合并
```

问题三：还有哪些其他常用的操作？

方法	说明
ndarray.repeat(repeats, axis=None)、numpy.repeat(a, repeats, axis=None)	用于重复元素。
numpy.tile(A, reps)	用于重复某数组的结构。
numpy.sort(a, axis=-1, kind=None, order=None)	升序排列，排序时创建了arr的一个复制品，不会改变原数组。
numpy.insert(arr, obj,values, axis=None)	在某个特定元素之前插入元素。
numpy.delete(arr, obj, axis=None)	删除某些特定元素。

In [360]:

```
# 重复元素
a = np.arange(4)
print(a)
print(a.repeat(3))    # 每个元素都重复n次
print(a.repeat([1,2,3,4]))  # 每个位置上的元素分别重复对应的 [1,2,3,4] 次
```

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

In [361]:

```
#重复结构
small = np.eye(2)
print(small)
big = np.tile(small, (2,2))
print(big)
```

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

In [398]:

```
# 排序
arr = np.random.randint(20,size=(3,3))
print(arr,'\n')
print(np.sort(arr),'\n')
print(np.sort(arr,axis = 0),'\n') # 按 0轴（列）排序
print(np.sort(arr,axis = 1)) # 按 1轴（行）排序，默认对最内层的元素进行排序
```

```
[[14 16  5]
 [13  2 13]
 [ 1  9  5]]

[[ 5 14 16]
 [ 2 13 13]
 [ 1  5  9]]

[[ 1  2  5]
 [13  9  5]
 [14 16 13]]

[[ 5 14 16]
 [ 2 13 13]
 [ 1  5  9]]
```

In [396]:

```
# 获取重新排序后元素所在的原索引的位置
print('排序后元素所在的原索引位置\n',np.argsort(arr))
```

排序后元素所在的原索引位置
 [[2 0 1]
 [1 0 2]
 [0 2 1]]

In [474]:

```
# 插入元素
a = np.arange(5)
print(a)
print(np.insert(a,-1,1000))  #在最后插入1000
```

```
[0 1 2 3 4]
[   0    1    2    3 1000    4]
```

In [475]:

```
# 删除元素
a = np.arange(5)
print(a)
print(np.delete(a,[1,3],None))  # 删除 1和3
```

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

6. 数组的运算

6.1 单个数组的运算

在使用单个数组的一元函数时，对于多维数组，计算时需特别注意对轴（axis）的设置。

In [465]:

```
# 通过 API函数提取科创板的行情数据
datas =  w.wsd("688001.SH,688002.SH,688003.SH,688005.SH", "pct_chg", "2019-07-22", "2019-07-26",
"Fill=Previous;PriceAdj=F")
arr = np.array(datas.Data)
arr = arr.round(2)
arr
```

Out[465]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

- **不涉及多个元素之间的运算**时，是对所有元素都进行相同的运算操作。

函数	说明
abs、fabs	计算整数、浮点数或复数的绝对值。对于非复数，使用fabs更快
sqrt、square、exp	计算各元素的平方根、平方、指数e^x
log、log10、log2、log1p	自然对数、底数10的对数、底数2的对数、ln(1 + x)
sign	计算各元素的正负号：正1,零0,负-1
ceil	计算各元素的取整：大于等于该数的最小整数
floor	计算各元素的取整：小于等于该数的最大整数
rint	各元素四舍五入最接近的整数，dtype不变
modf	将数组各元素的小数和整数部分以两个独立数组的形式返回
isnan、isfinite、isinf	判断各元素是否为NaN、是否有穷、是否为无穷
cos、cosh、sin、sinh、tan、tanh	一般和双曲型的三角函数
arccos、arccosh、arcsin、arcsinh、arctan、arctanh	反三角函数

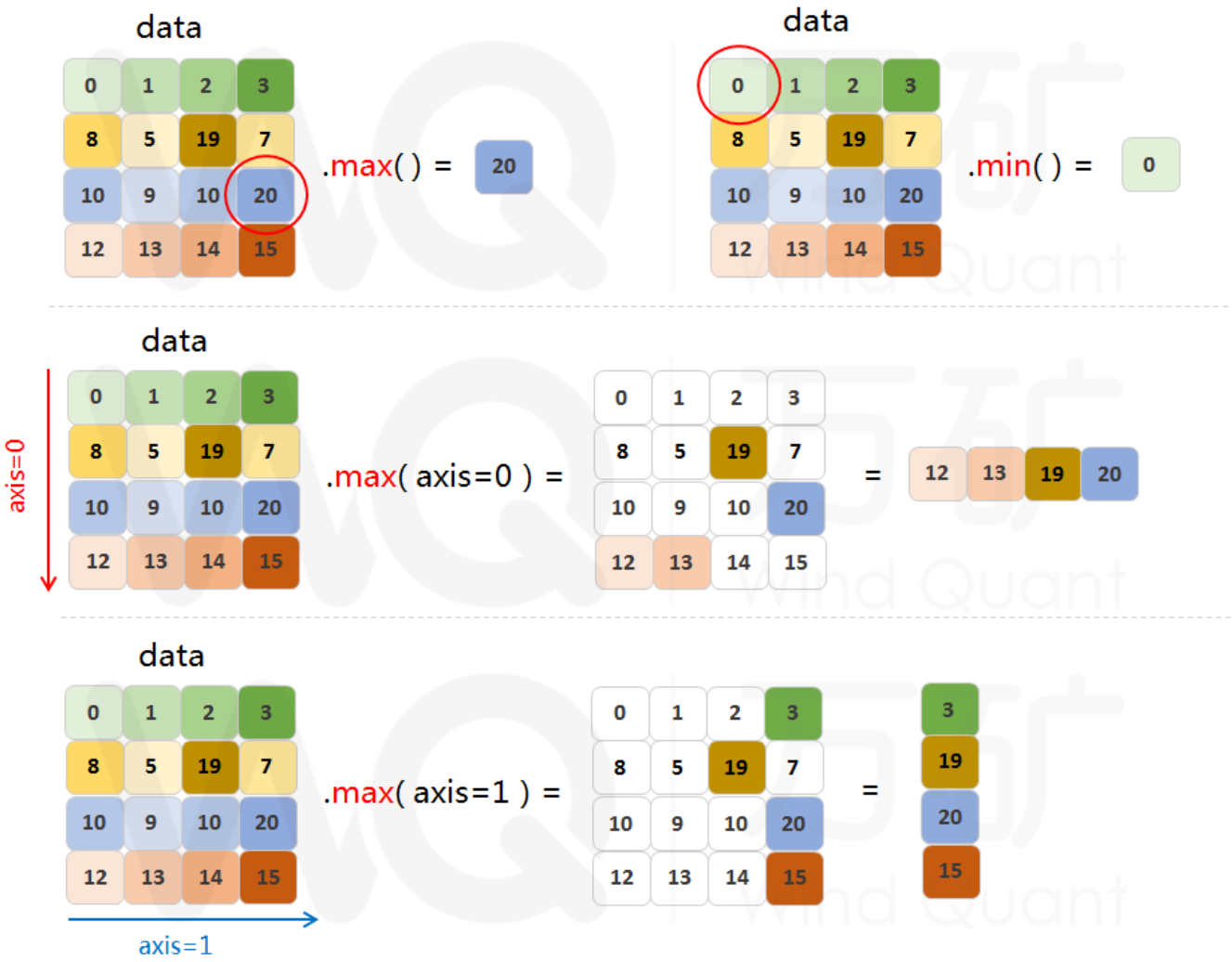
In [0]:

```
# 不涉及多个元素之间的运算时，是对所有元素都进行相同的运算操作

print(np.abs(arr),'\n')  # 取绝对值

print(np.square(arr),'\n')  # 取平方

print(np.ceil(arr),'\n')  # 取大于该数的最小整数

print(np.floor(arr),'\n')  # 取小于该数的最大整数

print(np.sign(arr),'\n')  # 计算各元素的正负号：正1,零0,负-1
```

```
[[128.77  12.2    5.73   6.19   7.4 ]
 [151.    12.57   6.45   5.03   7.83]
 [ 85.88  14.56   8.4   15.03   9.5 ]
 [ 86.06  15.81   6.38  10.28   8.65]]

[[16581.7129   148.84      32.8329    38.3161    54.76  ]
 [22801.       158.0049    41.6025    25.3009    61.3089]
 [ 7375.3744   211.9936    70.56     225.9009    90.25  ]
 [ 7406.3236   249.9561    40.7044   105.6784    74.8225]]

[[129. -12.   6.   7.  -7.]
 [151. -12.   7.   6.  -7.]
 [ 86. -14.   9.  16.  -9.]
 [ 87. -15.   7.  11.  -8.]]

[[128. -13.   5.   6.  -8.]
 [151. -13.   6.   5.  -8.]
 [ 85. -15.   8.  15. -10.]
 [ 86. -16.   6.  10.  -9.]]

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

- 涉及多个元素之间的运算时，此时特别需要注意对轴（axis）的设置。

函数	说明
sum、mean	数组全部或者按某个轴的方向进行求和、求均值
std、var	标准差、方差，自由度可以调整
min、max、argmin、argmax	最小和最大值、最小和最大元素的索引
cumsum、cumprod	数组全部或者按某个轴的方向进行累计和、累计积



In [0]:

```
# 求和函数

print(arr.sum(),'\n')   # 不设置轴时，默认全部元素求和

print(arr.sum(axis=0),'\n')   # axis=0，列向求和

print(arr.sum(axis=1),'\n')   # axis=1，横向求和
```

```
426.68

[451.71 -55.14  26.96  36.53 -33.38]

[121.09 142.08  85.25  78.26]
```

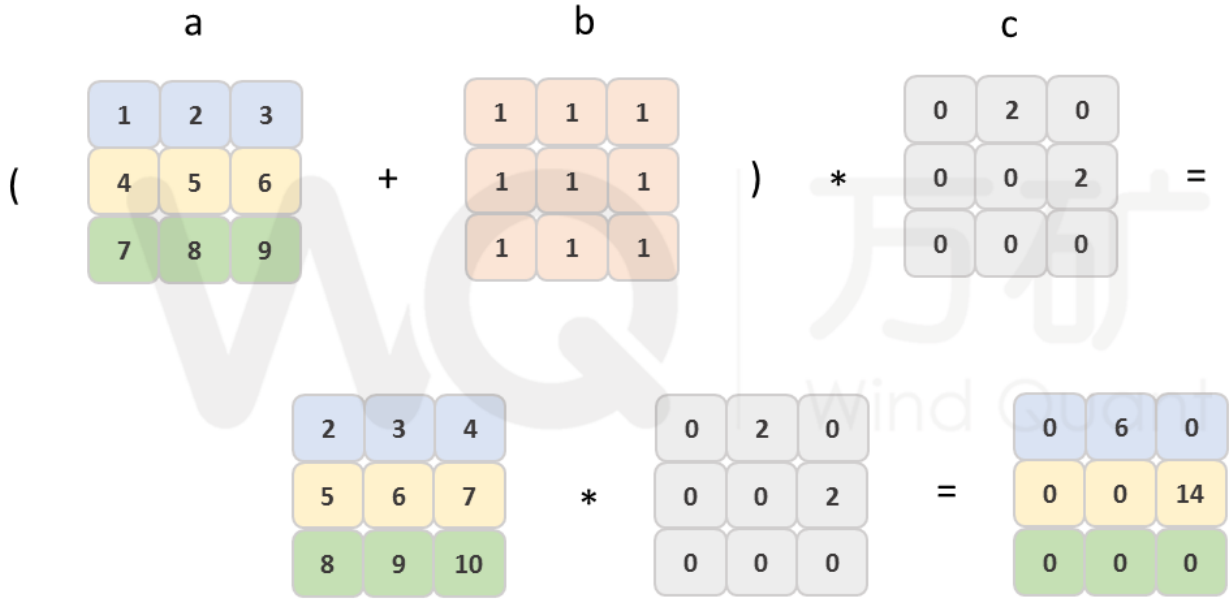
In [0]:

```
# 求均值函数

print(arr.mean(),'\n')  # 不设置轴时，默认全部元素求和

print(arr.mean(axis=0),'\n')  # axis=0，列向求和

print(arr.mean(axis=1),'\n')  # axis=0，横向求和
```

```
21.334

[112.9275 -13.785    6.74     9.1325  -8.345 ]

[24.218 28.416 17.05  15.652]
```

6.2 数组间的运算

- 对于**形状（shape）相同**的数组，在进行数组将加（+）、减（-）、乘（* ）、除（/）算术运算或大于（>）、等于（=）、小于（<）等等关系运算时，就是数组间各位置对应的元素进行计算。



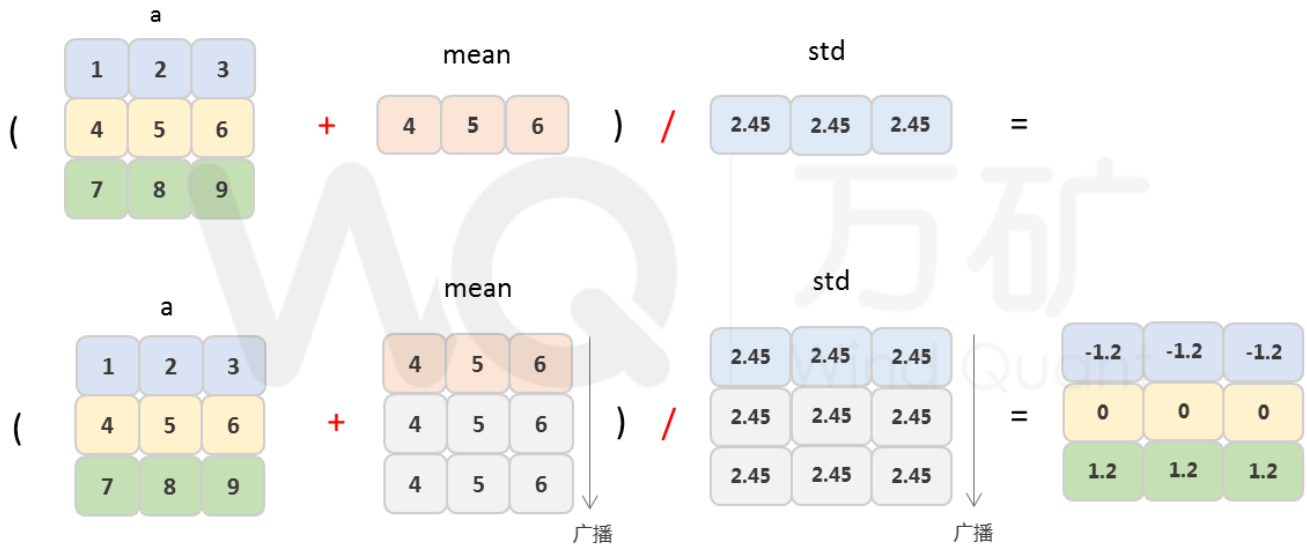
In [447]:

```
a = np.arange(1,10).reshape((3,3))
print(a,'\n')
b = np.ones((3,3))
print(b,'\n')
c = np.eye(3,3,k=1) *2
print(c,'\n')

# 计算（a+b）*c
```

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

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

[[0. 2. 0.]
 [0. 0. 2.]
 [0. 0. 0.]]
```

形状不同的数组之间还能进行运算吗？

- 对于**形状（shape）不同**的数组，在进行运算操作时，会触发广播机制。
- **广播机制**：对于形状小的数组，会适当复制元素，使得两个数组之间的形状相同，然后再对各位置上互相对应的元素进行操作。



In [466]:

```
arr
```

Out[466]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

In [462]:

```
# 对每个交易日内，四只科创股的收益率，进行标准化处理
# 第一步：计算均值
mean = arr.mean(axis=0)
# 第二步：计算标准差
std = arr.std(axis = 0)
# 第三步：对每个收益率进行标准化处理
arr2 = (arr-mean)/std
```

任何形状不同的数组，在运算过程中，都会触发广播吗？

在进行广播之前，会检查各数组的形状是否相容，只有形状相容的数组，才会进行广播，否则会报错。

In [ ]:

```
# 对每只科创股的一周的收益率，进行标准化处理
```

如何判断各数组的形状是否相容？

- 检查shape返回的形状元组（d1,d2,...,dn）的最后一个元素，若最后一个元素相等或其中有一个等于1，就可能是相容的。

其他运算函数

函数	说明
add、multiply	数组中对应的元素相加、相乘
substract	第一个数组减去第二个数组中的元素
divide、floor_divide	除法、向下圆整除法(余数直接舍弃)
power	对于第一个数组中的元素，根据第二个数组中的对应元素，进行幂运算
maximum、fmax	元素级的最大值、fmax功能相同只是忽略NaN
minimum、fmin	元素级的最小值、fmin功能相同只是忽略NaN
mod	元素级的求余
copysign	将第二个数组中的值的符号复制给第一个数组中的值
greater、greater_equal、less、less_equal、equal、not_equal	元素级的比较运算，产生True或者False为元素的数组
logical_and、logical_or、logical_xor	元素级的逻辑判断(且、或者、不等于)

7. Numpy 矩阵

7.1 二维数组转矩阵

```
numpy.mat(data,dtype=None)
numpy.asmatrix(data,dtype=None)   # 针对二维数组
```

In [470]:

```
np.mat(arr)
```

Out[470]:

```
matrix([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
        [151.  , -12.57,   6.45,   5.03,  -7.83],
        [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
        [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

In [469]:

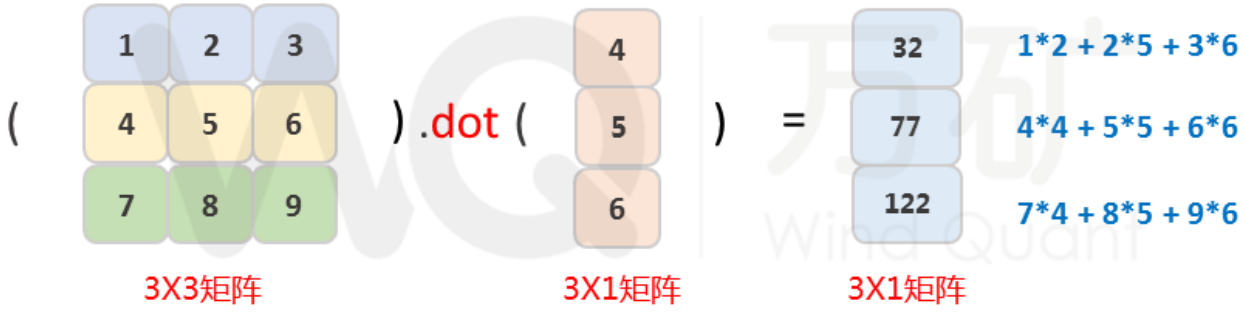
```
np.asmatrix(arr)
```

Out[469]:

```
matrix([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
        [151.  , -12.57,   6.45,   5.03,  -7.83],
        [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
        [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

7.2 矩阵点乘

```
numpy.dot(a,b,out=None)
```



In [482]:

```
a = np.arange(1,10).reshape((3,3))
b = np.array([4,5,6])
np.dot(a,b)
```

Out[482]:

```
array([ 32,  77, 122])
```

7.3 矩阵性质

In [471]:

```
x = np.random.randint(1,10,(3,3))
print('原矩阵：\n',x)
print()
print('矩阵对角线：\n',np.diag(x))
print()
print('矩阵上三角：\n',np.triu(x))
print()
print('矩阵下三角：\n',np.tril(x))
print()
print('矩阵的迹：\n',np.trace(x))
print()
print('矩阵的转置：\n',x.T)
```

```
原矩阵：
 [[4 5 1]
 [6 9 6]
 [9 5 9]]

矩阵对角线：
 [4 9 9]

矩阵上三角：
 [[4 5 1]
 [0 9 6]
 [0 0 9]]

矩阵下三角：
 [[4 0 0]
 [6 9 0]
 [9 5 9]]

矩阵的迹：
 22

矩阵的转置：
 [[4 6 9]
 [5 9 5]
 [1 6 9]]
```

7.4 矩阵运算

numpy下有一个子模块`linalg`，是一个线性代数运算库，关于矩阵运算主要使用该库来完成。

numpy.linalg函数

函数	说明
diag	以一维数组的形式返回方阵的对角线元素或将一维数组转化为方阵
dot、trace、det	矩阵乘法、矩阵的迹运算、矩阵行列式
eig、inv、pinv	方阵的特征值和特征向量、方阵的逆、矩阵的Moore-Penrose伪逆
qr、svd	矩阵的QR分解、奇异值分解
solve	解线性方程组Xβ = y，其中X为方阵
lstsq	计算Xβ = y的最小二乘解

In [472]:

```
import numpy.linalg as la
```

In [479]:

```
arr
```

Out[479]:

```
array([[128.77, -12.2 ,   5.73,   6.19,  -7.4 ],
       [151.  , -12.57,   6.45,   5.03,  -7.83],
       [ 85.88, -14.56,   8.4 ,  15.03,  -9.5 ],
       [ 86.06, -15.81,   6.38,  10.28,  -8.65]])
```

In [480]:

```
arr_ = np.delete(arr,-1,axis=1)  #删除最后一天的数据
arr_
```

Out[480]:

```
array([[128.77, -12.2 ,   5.73,   6.19],
       [151.  , -12.57,   6.45,   5.03],
       [ 85.88, -14.56,   8.4 ,  15.03],
       [ 86.06, -15.81,   6.38,  10.28]])
```

In [481]:

```
print('矩阵的行列式：\n',la.det(arr_))
print()
print('矩阵的逆：\n',la.inv(arr_))
print()
print('矩阵的特征值分解：\n',la.eig(arr_))
print()
print('矩阵的奇异值分解：\n',la.svd(arr_))
```

```
矩阵的行列式：
 -3761.127155110001

矩阵的逆：
 [[ 6.55648615e-02 -3.73579943e-02 -5.69284130e-04 -2.03676499e-02]
 [ 8.71949361e-02 -1.09984795e-02  1.37989492e-01 -2.48871242e-01]
 [-2.01843685e+00  1.50672224e+00  2.19049319e-01  1.57879372e-01]
 [ 8.37905356e-01 -6.39274794e-01  8.10381134e-02 -2.12945991e-01]]

矩阵的特征值分解：
 (array([122.33325492,  -1.53596851,   1.60407202,  12.47864157]), array([[-0.55317647,  0.03687606, -0.01370074, -0.09414866],
       [-0.65060664,  0.38255053,  0.25141973, -0.25490575],
       [-0.38062244, -0.68849927,  0.96118892,  0.81479023],
       [-0.35472997,  0.61503171, -0.11276645,  0.51212885]]))

矩阵的奇异值分解：
 (array([[-0.55110672, -0.2479144 , -0.02564565, -0.79634297],
       [-0.64470986, -0.47106432, -0.08753551,  0.59563842],
       [-0.37483972,  0.70150732, -0.60310019,  0.06043855],
       [-0.37433375,  0.47383998,  0.79243296,  0.08602277]]), array([235.10098301,  15.27126962,  2.87841174,   0.36394479]), array([[-0.98988782,  0.11145588, -0.05467065, -0.06863533],
       [-0.13296418, -0.57359491,  0.29184517,  0.75374784],
       [-0.04089637, -0.81087356, -0.25078864, -0.52717797],
       [-0.02775096, -0.0324118 ,  0.9213808 , -0.38631177]]))
```