**Numpy**

**学习目标**

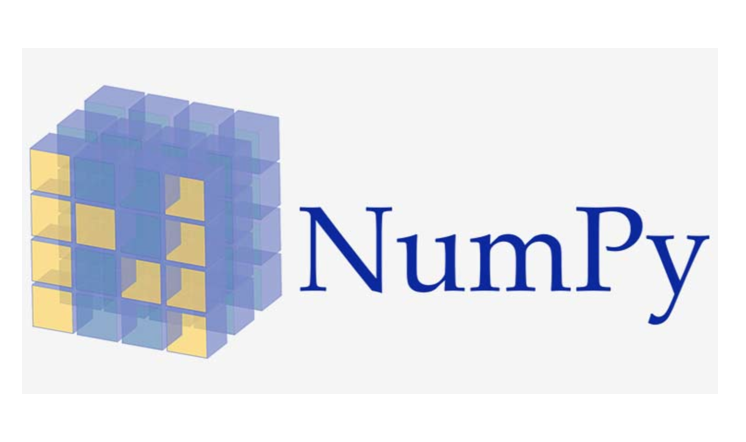
* 了解Numpy运算速度上的优势
* 知道数组的属性，形状、类型
* 应用Numpy实现数组的基本操作
* 应用随机数组的创建实现正态分布应用
* 应用Numpy实现数组的逻辑运算
* 应用Numpy实现数组的统计运算
* 应用Numpy实现数组之间的运算

**Numpy优势**

**学习目标**

* 目标
  + 了解Numpy运算速度上的优势
  + 知道Numpy的数组内存块风格
  + 知道Numpy的并行化运算
* 应用
  + 机器学习,深度学习各种框架的基础库

## 1 Numpy介绍



Numpy（Numerical Python）是一个开源的Python科学计算库，用于快速处理任意维度的数组。

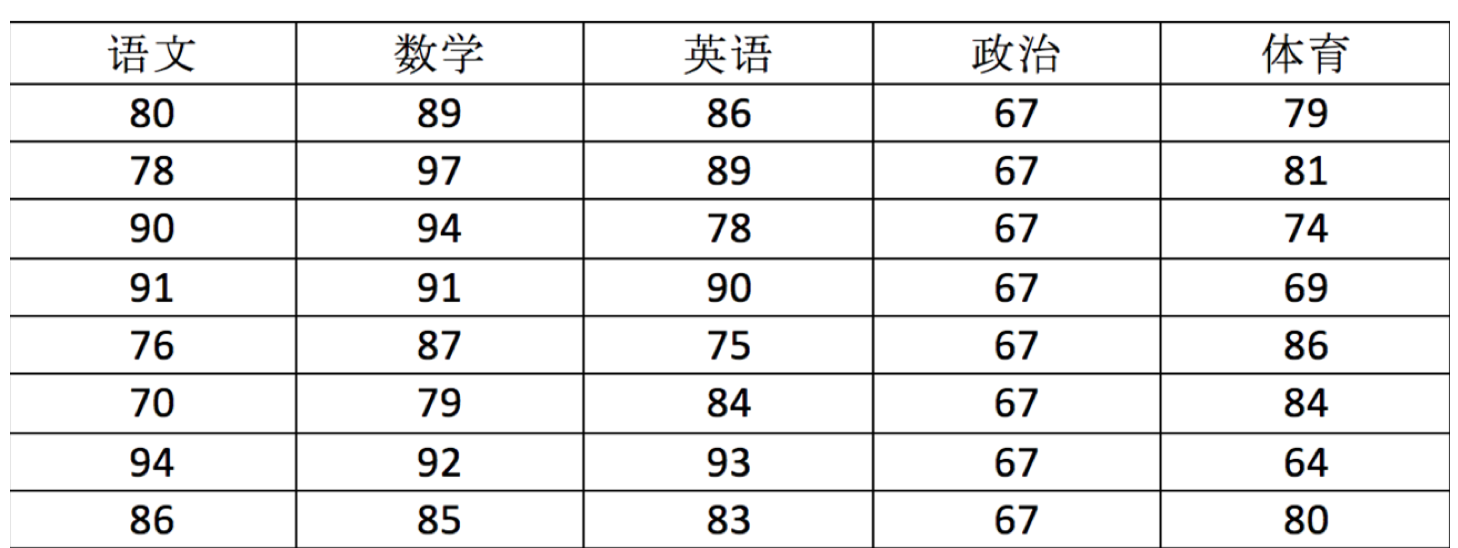
Numpy支持常见的数组和矩阵操作。对于同样的数值计算任务，使用Numpy比直接使用Python要简洁的多。

Numpy使用ndarray对象来处理多维数组，该对象是一个快速而灵活的大数据容器。

## 2 ndarray介绍

NumPy provides an N-dimensional array type, the ndarray, which describes a collection of “items” of the same type.

NumPy提供了一个**N维数组类型ndarray**，它描述了**相同类型**的“items”的集合。



用ndarray进行存储：

import numpy as np

# 创建ndarray

score = np.array([[80, 89, 86, 67, 79],

[78, 97, 89, 67, 81],

[90, 94, 78, 67, 74],

[91, 91, 90, 67, 69],

[76, 87, 75, 67, 86],

[70, 79, 84, 67, 84],

[94, 92, 93, 67, 64],

[86, 85, 83, 67, 80]])

score

返回结果：

array([[80, 89, 86, 67, 79],

[78, 97, 89, 67, 81],

[90, 94, 78, 67, 74],

[91, 91, 90, 67, 69],

[76, 87, 75, 67, 86],

[70, 79, 84, 67, 84],

[94, 92, 93, 67, 64],

[86, 85, 83, 67, 80]])

使用Python列表可以存储一维数组，通过列表的嵌套可以实现多维数组，那么为什么还需要使用Numpy的ndarray呢？

## 3 ndarray与Python原生list运算效率对比

在这里我们通过一段代码运行来体会到ndarray的好处

import random

import time

import numpy as np

a = []

for i in range(100000000):

a.append(random.random())

# 通过%time魔法方法, 查看当前行的代码运行一次所花费的时间

%time sum1=sum(a)

b=np.array(a)

%time sum2=np.sum(b)

其中第一个时间显示的是使用原生Python计算时间,第二个内容是使用numpy计算时间:

CPU times: user 852 ms, sys: 262 ms, total: 1.11 s

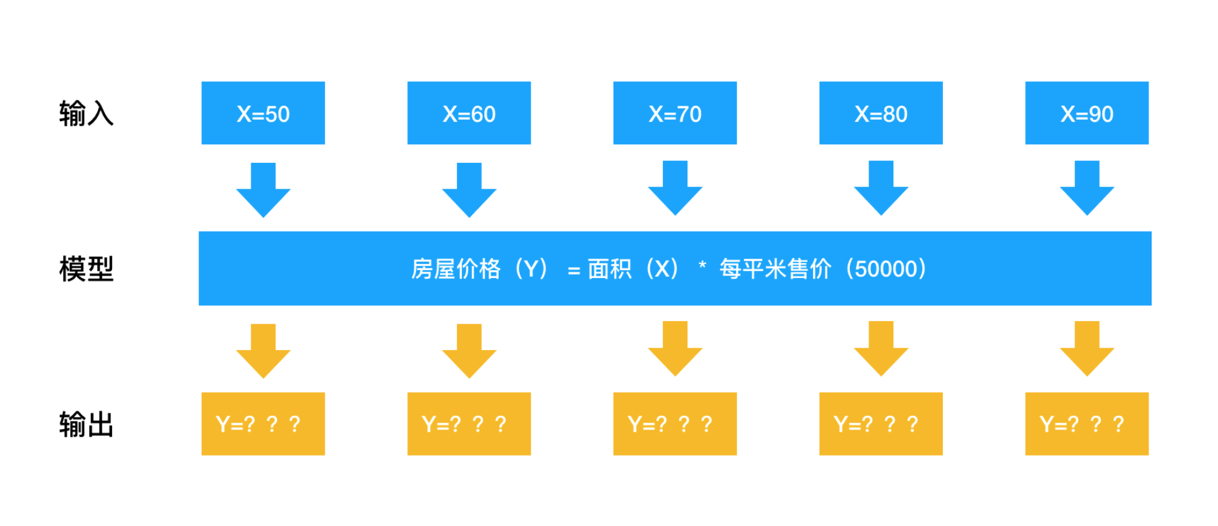
Wall time: 1.13 s

CPU times: user 133 ms, sys: 653 µs, total: 133 ms

Wall time: 134 ms

从中我们看到ndarray的计算速度要快很多，节约了时间。

**机器学习的最大特点就是大量的数据运算**，那么如果没有一个快速的解决方案，那可能现在python也在机器学习领域达不到好的效果。



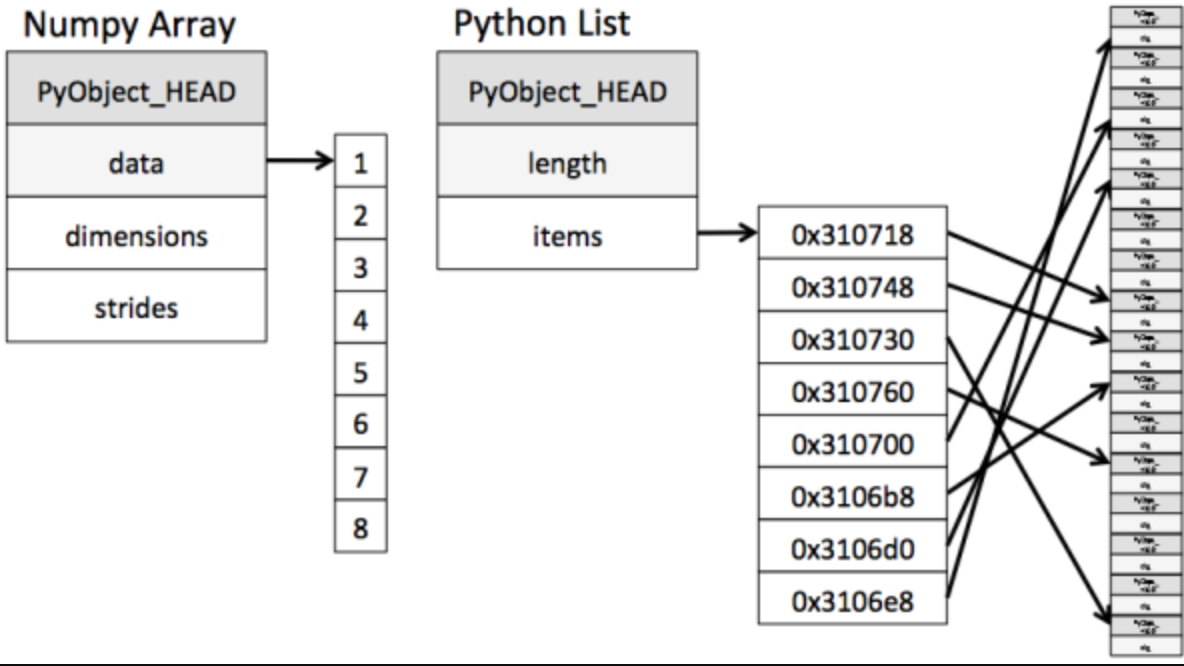
Numpy专门针对ndarray的操作和运算进行了设计，所以数组的存储效率和输入输出性能远优于Python中的嵌套列表，数组越大，Numpy的优势就越明显。

**思考：ndarray为什么可以这么快？**

## 4 ndarray的优势

#### 4.1 内存块风格

ndarray到底跟原生python列表有什么不同呢，请看一张图：



从图中我们可以看出ndarray在存储数据的时候，数据与数据的地址都是连续的，这样就给使得批量操作数组元素时速度更快。

这是因为ndarray中的所有元素的类型都是相同的，而Python列表中的元素类型是任意的，所以ndarray在存储元素时内存可以连续，而python原生list就只能通过寻址方式找到下一个元素，这虽然也导致了在通用性能方面Numpy的ndarray不及Python原生list，但在科学计算中，Numpy的ndarray就可以省掉很多循环语句，代码使用方面比Python原生list简单的多。

#### 4.2 ndarray支持并行化运算（向量化运算）

#### 4.3 效率远高于纯Python代码

Numpy底层使用C语言编写，内部解除了GIL（全局解释器锁），其对数组的操作速度不受Python解释器的限制，所以，其效率远高于纯Python代码。

## 5 小结

* numpy介绍【了解】
  + 一个开源的Python科学计算库
  + 计算器来要比python简洁高效
  + Numpy使用ndarray对象来处理多维数组
* ndarray介绍【了解】
  + NumPy提供了一个N维数组类型ndarray，它描述了相同类型的“items”的集合。
  + 生成numpy对象:np.array()
* ndarray的优势【掌握】
  + 内存块风格
    - list -- 分离式存储,存储内容多样化
    - ndarray -- 一体式存储,存储类型必须一样
  + ndarray支持并行化运算（向量化运算）
  + ndarray底层是用C语言写的,效率更高,释放了GIL

**N维数组-ndarray**

**学习目标**

* 目标
  + 说明数组的属性，形状、类型
* 应用
  + 无

**1 ndarray的属性**

数组属性反映了数组本身固有的信息。

| **属性名字** | **属性解释** |
| --- | --- |
| ndarray.shape | 数组维度的元组 |
| ndarray.ndim | 数组维数 |
| ndarray.size | 数组中的元素数量 |
| ndarray.itemsize | 一个数组元素的长度（字节） |
| ndarray.dtype | 数组元素的类型 |

**2 ndarray的形状**

首先创建一些数组。

# 创建不同形状的数组

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

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

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

分别打印出形状

>>> a.shape

>>> b.shape

>>> c.shape

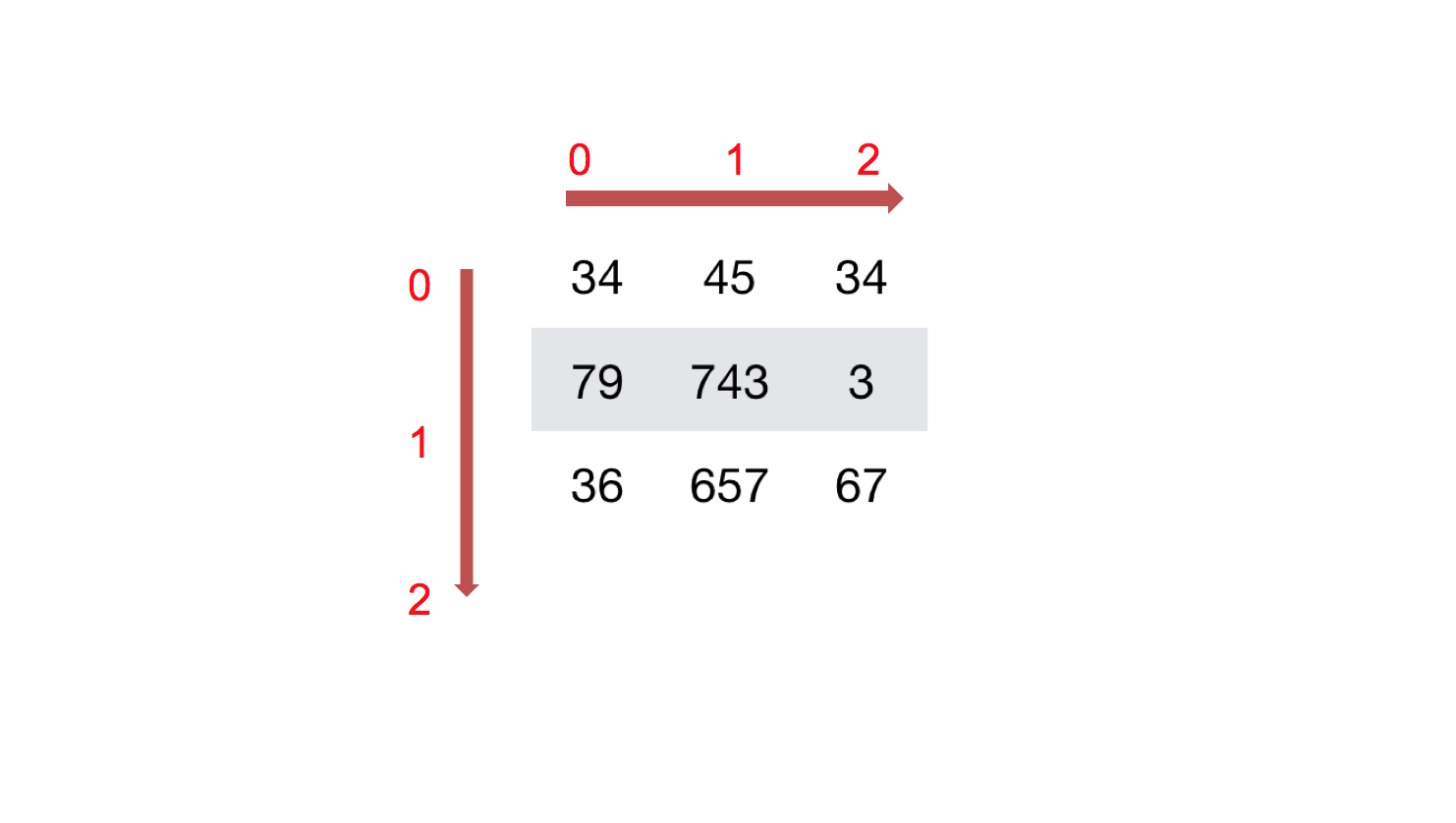
(2, 3) # 二维数组

(4,) # 一维数组

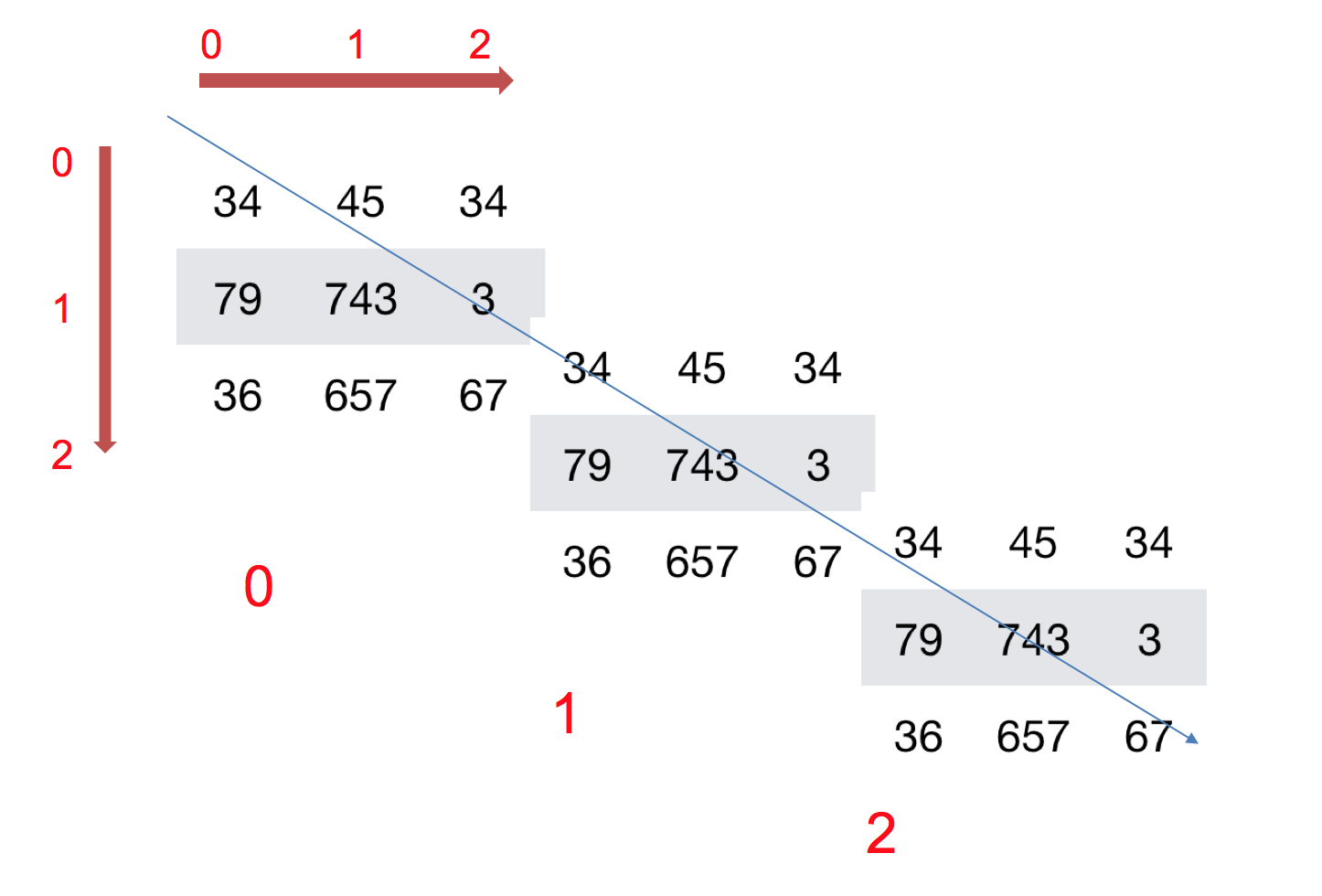
(2, 2, 3) # 三维数组

如何理解数组的形状？

二维数组：



三维数组：



**3 ndarray的类型**

>>> type(score.dtype)

<type 'numpy.dtype'>

dtype是numpy.dtype类型，先看看对于数组来说都有哪些类型

| **名称** | **描述** | **简写** |
| --- | --- | --- |
| np.bool | 用一个字节存储的布尔类型（True或False） | 'b' |
| np.int8 | 一个字节大小，-128 至 127 | 'i' |
| np.int16 | 整数，-32768 至 32767 | 'i2' |
| np.int32 | 整数，-2 **31 至 2**32 -1 | 'i4' |
| np.int64 | 整数，-2 **63 至 2**63 - 1 | 'i8' |
| np.uint8 | 无符号整数，0 至 255 | 'u' |
| np.uint16 | 无符号整数，0 至 65535 | 'u2' |
| np.uint32 | 无符号整数，0 至 2 \*\* 32 - 1 | 'u4' |
| np.uint64 | 无符号整数，0 至 2 \*\* 64 - 1 | 'u8' |
| np.float16 | 半精度浮点数：16位，正负号1位，指数5位，精度10位 | 'f2' |
| np.float32 | 单精度浮点数：32位，正负号1位，指数8位，精度23位 | 'f4' |
| np.float64 | 双精度浮点数：64位，正负号1位，指数11位，精度52位 | 'f8' |
| np.complex64 | 复数，分别用两个32位浮点数表示实部和虚部 | 'c8' |
| np.complex128 | 复数，分别用两个64位浮点数表示实部和虚部 | 'c16' |
| np.object\_ | python对象 | 'O' |
| np.string\_ | 字符串 | 'S' |
| np.unicode\_ | unicode类型 | 'U' |

**创建数组的时候指定类型**

>>> a = np.array([[1, 2, 3],[4, 5, 6]], dtype=np.float32)

>>> a.dtype

dtype('float32')

>>> arr = np.array(['python', 'tensorflow', 'scikit-learn', 'numpy'], dtype = np.string\_)

>>> arr

array([b'python', b'tensorflow', b'scikit-learn', b'numpy'], dtype='|S12')

* 注意：若不指定，整数默认int64，小数默认float64

**4 总结**

知道数组的基本属性，不同形状的维度表示以及数组的类型

**基本操作**

**学习目标**

* 目标
  + 理解数组的各种生成方法
  + 应用数组的索引机制实现数组的切片获取
  + 应用维度变换实现数组的形状改变
  + 应用类型变换实现数组类型改变
  + 应用数组的转换
* 应用
  + 应用正态分布实现模拟股票的涨跌幅数据操作

## 1 生成数组的方法

### 1.1 生成0和1的数组

* **np.ones(shape[, dtype, order])**
* np.ones\_like(a[, dtype, order, subok])
* **np.zeros(shape[, dtype, order])**
* np.zeros\_like(a[, dtype, order, subok])

>>> zero = np.zeros([3, 4])

array([[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.]])

### 1.2 从现有数组生成

#### 1.2.1 生成方式

* **np.array(object[, dtype, copy, order, subok, ndmin])**
* **np.asarray(a[, dtype, order])**

a = np.array([[1,2,3],[4,5,6]])

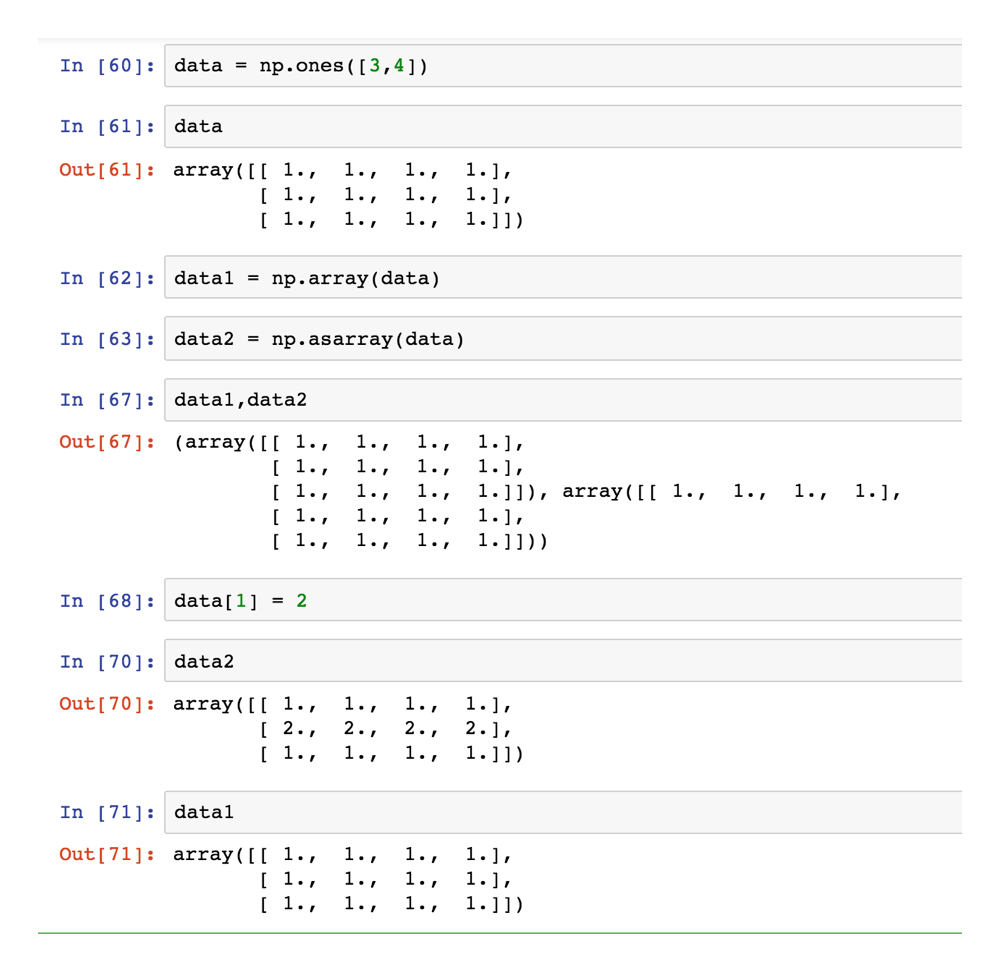
# 从现有的数组当中创建

a1 = np.array(a)

# 相当于索引的形式，并没有真正的创建一个新的

a2 = np.asarray(a)

#### 1.2.2 关于array和asarray的不同



### 1.3 生成固定范围的数组

* np.linspace (start, stop, num, endpoint)

生成等间隔的序列

start 序列的起始值

stop 序列的终止值，

num 要生成的等间隔样例数量，默认为50

endpoint 序列中是否包含stop值，默认为ture

# 生成等间隔的数组

np.linspace(0, 100, 11)

返回结果：

array([ 0., 10., 20., 30., 40., 50., 60., 70., 80., 90., 100.])

* 其它的还有
  + numpy.arange(start,stop, step, dtype)
  + numpy.logspace(start,stop, num)

np.arange(10, 50, 2)

返回结果：

array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,

44, 46, 48])

# 生成10^x

np.logspace(0, 2, 3)

返回结果:

array([ 1., 10., 100.])

### 1.4 生成随机数组

### 1.4.1 使用模块介绍

* np.random模块

### 1.4.2 均匀分布

* np.random.rand(d0, d1, ..., dn)

返回**[0.0，1.0)**内的一组均匀分布的数。

* **np.random.uniform(low=0.0, high=1.0, size=None)**

功能：从一个均匀分布[low,high)中随机采样，注意定义域是左闭右开，即包含low，不包含high.

参数介绍:

low: 采样下界，float类型，默认值为0；

high: 采样上界，float类型，默认值为1；

size: 输出样本数目，为int或元组(tuple)类型，例如，size=(m,n,k), 则输出mnk个样本，缺省时输出1个值。

返回值：ndarray类型，其形状和参数size中描述一致。

* np.random.randint(low, high=None, size=None, dtype='l')

从一个均匀分布中随机采样，生成一个整数或N维整数数组，取数范围：若high不为None时，取[low,high)之间随机整数，否则取值[0,low)之间随机整数。

# 生成均匀分布的随机数

x1 = np.random.uniform(-1, 1, 100000000)

返回结果：

array([ 0.22411206, 0.31414671, 0.85655613, ..., -0.92972446,

0.95985223, 0.23197723])

画图看分布状况：

import matplotlib.pyplot as plt

# 生成均匀分布的随机数

x1 = np.random.uniform(-1, 1, 100000000)

# 画图看分布状况

# 1）创建画布

plt.figure(figsize=(10, 10), dpi=100)

# 2）绘制直方图

plt.hist(x=x1, bins=1000) # x代表要使用的数据，bins表示要划分区间数

# 3）显示图像

plt.show()

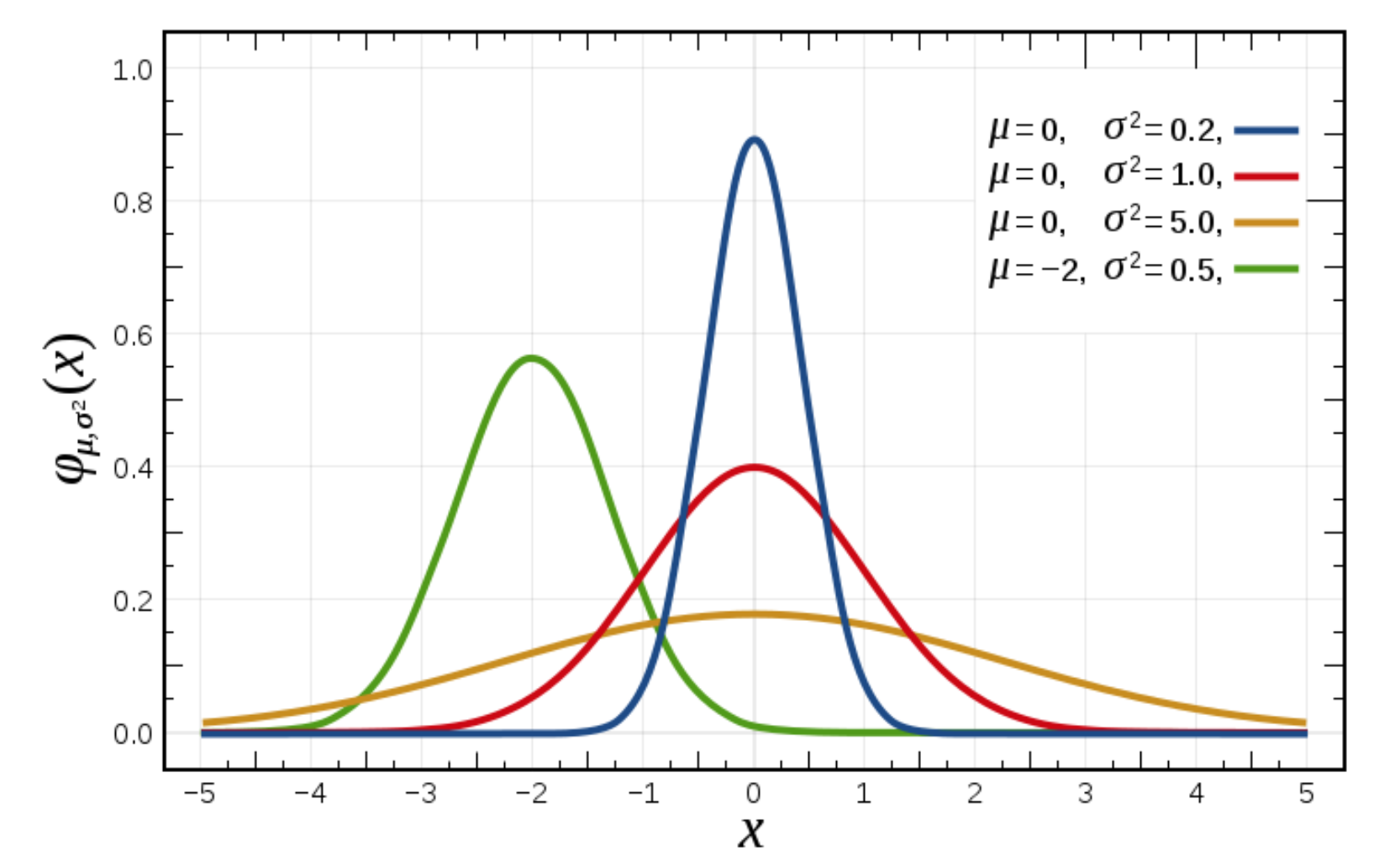


### 1.4.3 正态分布

#### 1.4.3.1 基础概念复习：正态分布（理解）

##### a. 什么是正态分布

正态分布是一种概率分布。正态分布是具有两个参数μ和σ的连续型随机变量的分布，第一参数μ是服从正态分布的随机变量的均值，第二个参数σ是此随机变量的方差，所以正态分布记作**N(μ，σ )**。

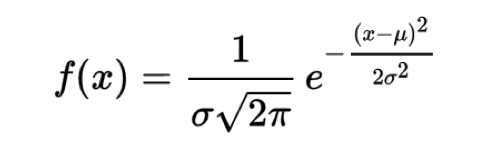


##### b. 正态分布的应用

生活、生产与科学实验中很多随机变量的概率分布都可以近似地用正态分布来描述。

##### c. 正态分布特点

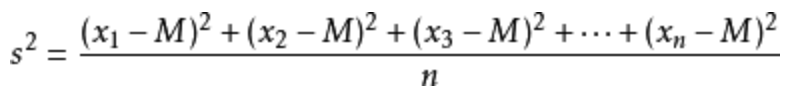
**μ决定了其位置，其标准差σ**决定了分布的幅度。当μ = 0,σ = 1时的正态分布是标准正态分布。



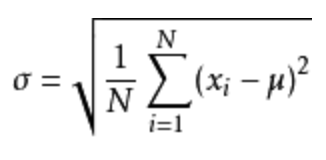
标准差如何来？

###### 方差

是在概率论和统计方差衡量一组数据时离散程度的度量

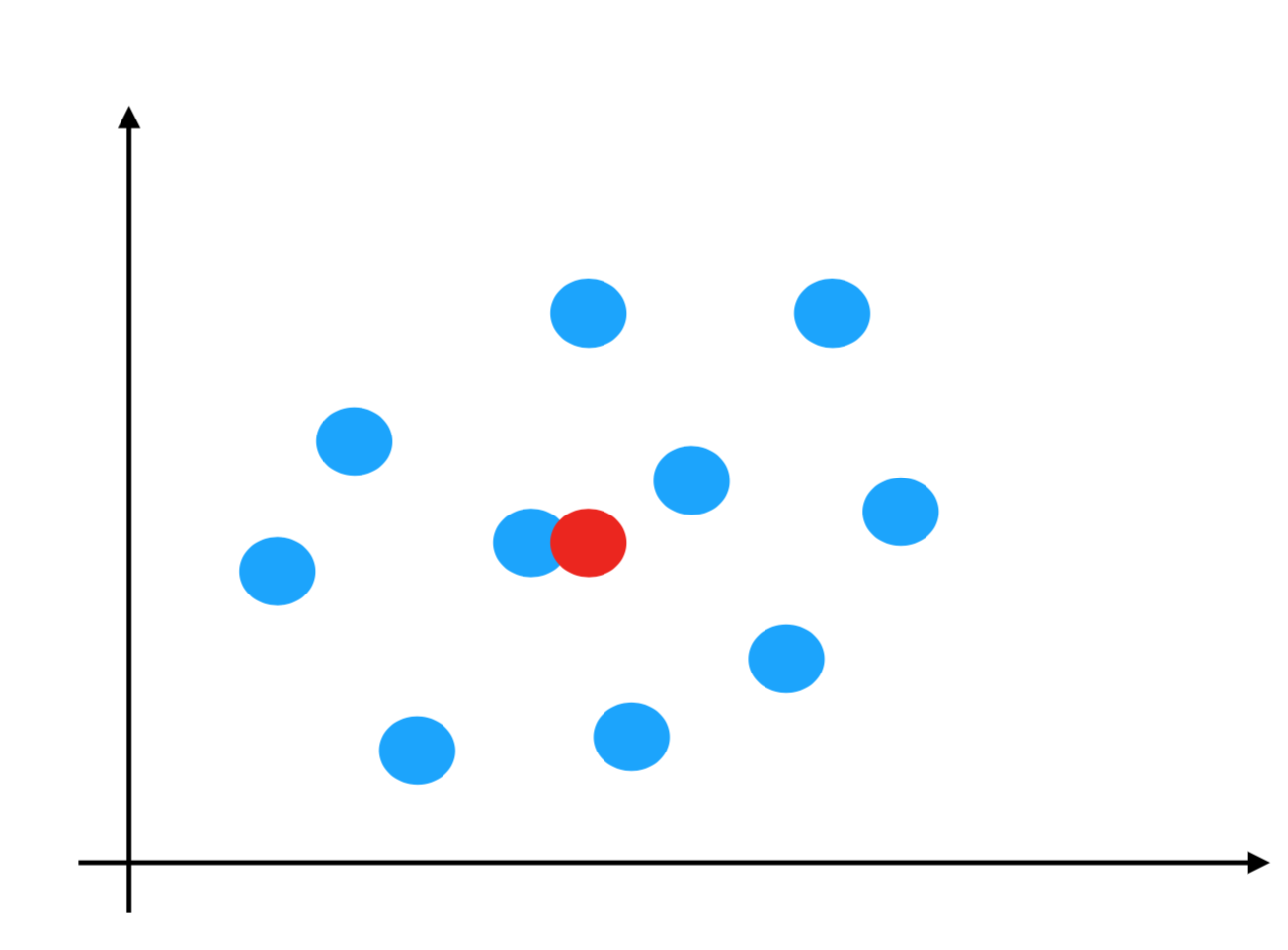


其中M为平均值，n为数据总个数，S为标准差，S^2可以理解一个整体为方差



###### 标准差与方差的意义

可以理解成数据的一个离散程度的衡量



#### 1.4.3.2 正态分布创建方式

* np.random.randn(d0, d1, …, dn)

功能：从标准正态分布中返回一个或多个样本值

* **np.random.normal(loc=0.0, scale=1.0, size=None)**

loc：float

​ 此概率分布的均值（对应着整个分布的中心centre）

scale：float

​ 此概率分布的标准差（对应于分布的宽度，scale越大越矮胖，scale越小，越瘦高）

size：int or tuple of ints

​ 输出的shape，默认为None，只输出一个值

* np.random.standard\_normal(size=None)

返回指定形状的标准正态分布的数组。

x2 = np.random.normal(1.75, 1, 100000000)

返回结果：

array([2.90646763, 1.46737886, 2.21799024, ..., 1.56047411, 1.87969135,

0.9028096 ])

# 生成均匀分布的随机数

x2 = np.random.normal(1.75, 1, 100000000)

# 画图看分布状况

# 1）创建画布

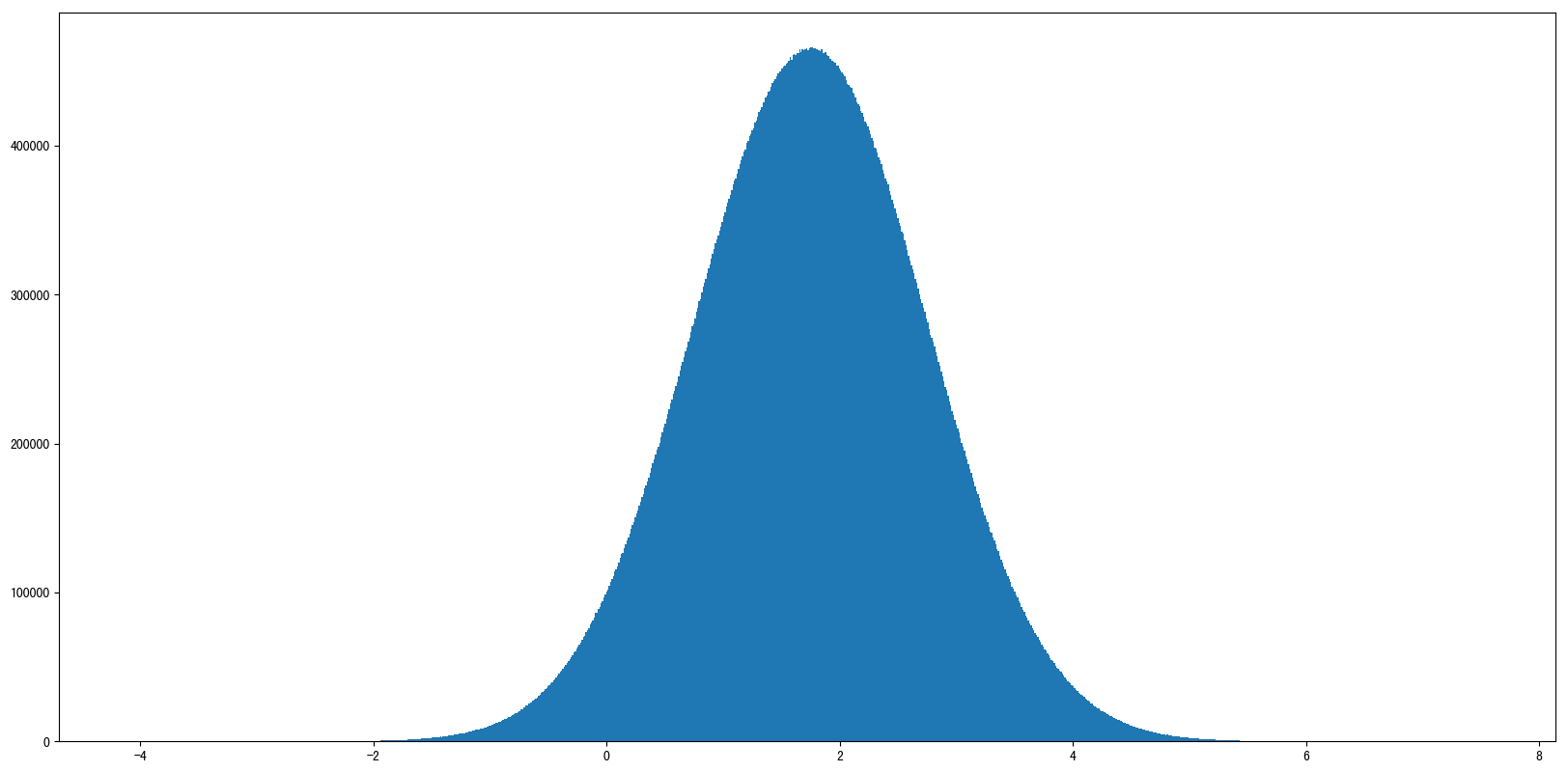
plt.figure(figsize=(20, 10), dpi=100)

# 2）绘制直方图

plt.hist(x2, 1000)

# 3）显示图像

plt.show()



例如：我们可以模拟生成一组股票的涨跌幅的数据

##### 案例：随机生成8只股票2周的交易日涨幅数据

8只股票，**两周(10天)**的涨跌幅数据，如何获取？

* 两周的交易日数量为：2 X 5 =10
* 随机生成涨跌幅在某个正态分布内，比如均值0，方差1

##### 股票涨跌幅数据的创建

# 创建符合正态分布的8只股票10天的涨跌幅数据

stock\_change = np.random.normal(0, 1, (8, 10))

stock\_change

返回结果：

array([[-0.03862668, -1.46128096, -0.75596237, 0.89737022, -1.86978433,

0.38845392, 1.14856354, -1.10912275, 1.28900021, -0.86801677],

[ 1.8701446 , 0.50407342, -0.74396489, -1.69641331, -0.89969227,

2.42459765, 0.78987443, -0.82935223, 0.82546455, 0.40524289],

[-1.7365776 , -0.81501515, -1.08974743, -1.4138561 , 0.85495155,

0.30076624, -2.0813311 , 1.52886975, 0.62945643, -1.48714683],

[-1.12084983, -0.63070289, -0.20677245, -0.49096973, -0.40551104,

-0.46320893, -0.73190969, 0.00392486, 2.524498 , 0.25843191],

[ 0.05001371, 0.52267878, 1.31974783, 0.64840953, 1.56360431,

-0.79233575, 1.47275167, 0.61070343, -0.33247221, -0.57492172],

[ 0.7092757 , 0.00928591, 0.27008373, 0.79050681, -0.95887888,

0.03661459, 0.81303927, -0.54139691, 0.69623969, 0.06570421],

[-1.40481949, 0.49151272, 1.01992235, 1.89152928, -0.76415623,

0.3100241 , 1.27156806, -1.76199057, -0.77419965, -0.17574386],

[-0.21745814, -1.78433763, -0.7953073 , 0.4765644 , -0.2589575 ,

0.97014013, 1.67321489, 1.73370987, 0.29886514, 1.27186522]])

## 2 数组的索引、切片

* 获取第一个股票的前3个交易日的涨跌幅数据

# 二维的数组，两个维度

stock\_change[0, 0:3]

返回结果：

array([-0.03862668, -1.46128096, -0.75596237])

一维、二维、三维的数组如何索引？

# 三维，一维

a1 = np.array([ [[1,2,3],[4,5,6]], [[12,3,34],[5,6,7]]])

# 返回结果

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

[ 4, 5, 6]],

[[12, 3, 34],

[ 5, 6, 7]]])

# 索引、切片

>>> a1[0, 0, 1] # 输出: 2

## 3 形状修改

让刚才的股票行、日期列反过来，变成日期行，股票列

* **ndarray.reshape(shape[, order]) Returns an array containing the same data with a new shape.**

# 在转换形状的时候，一定要注意数组的元素匹配

stock\_change.reshape([10, 8])

stock\_change.reshape([-1,20]) # 数组的形状被修改为: (4, 20), -1: 表示通过待计算

* **ndarray.resize(new\_shape[, refcheck]) Change shape and size of array in-place.**

stock\_change.resize([10, 8])

* **ndarray.T 数组的转置**
  + 将数组的行、列进行互换

stock\_change.shape

(10, 8)

stock\_change.T.shape

(8, 10)

## 4 类型修改

* **ndarray.astype(type)**

stock\_change.astype(np.int32)

* **ndarray.tostring([order])或者ndarray.tobytes([order]) Construct Python bytes containing the raw data bytes in the array.**
  + 转换成bytes

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[12, 3, 34], [5, 6, 7]]])

arr.tostring()

拓展：如果遇到

IOPub data rate exceeded.

The notebook server will temporarily stop sending output

to the client in order to avoid crashing it.

To change this limit, set the config variable

`--NotebookApp.iopub\_data\_rate\_limit`.

这个问题是在jupyer当中对输出的字节数有限制，需要去修改配置文件

创建配置文件

jupyter notebook --generate-config

vi ~/.jupyter/jupyter\_notebook\_config.py

取消注释,多增加

## (bytes/sec) Maximum rate at which messages can be sent on iopub before they

# are limited.

c.NotebookApp.iopub\_data\_rate\_limit = 10000000

**但是不建议这样去修改，jupyter输出太大会崩溃**

## 5 数组的去重

* **np.unique()**

temp = np.array([[1, 2, 3, 4],[3, 4, 5, 6]])

>>> np.unique(temp)

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

## 6 小结

* 创建数组【掌握】
  + 均匀
    - np.random.rand()
    - np.random.uniform(0, 1, 100)
    - np.random.randint(0, 10, 10)
  + **随机（正态分布）**
    - 里面需要关注的参数:均值:u, 标准差:σ
      * u -- 决定了这个图形的左右位置
      * σ -- 决定了这个图形是瘦高还是矮胖
    - np.random.randn()
    - np.random.normal(0, 1, 100)
* 数组索引【知道】
  + 直接进行索引,切片
  + 对象[:, :] -- 先行后列
* 数组形状改变【掌握】
  + 对象.reshape()
    - 没有进行行列互换,新产生一个ndarray
  + 对象.resize()
    - 没有进行行列互换,修改原来的ndarray
  + 对象.T
    - 进行了行列互换
* 数组去重【知道】
  + np.unique(对象)

**ndarray运算**

**学习目标**

* 目标
  + 应用数组的通用判断函数
  + 应用np.where实现数组的三元运算
* 应用
  + 股票涨跌幅数据逻辑运算

## 问题

**如果想要操作符合某一条件的数据，应该怎么做？**

## 1 逻辑运算

# 重新生成8只股票10个交易日的涨跌幅数据

>>> stock\_change = np.random.normal(0, 1, (8, 10))

>>> stock\_change = stock\_change[0:5, 0:5]

# 逻辑判断, 如果涨跌幅大于0.5就标记为True 否则为False

>>> stock\_change > 0.5

array([[ True, False, False, True, False],

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

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

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

[False, False, False, True, True]])

# BOOL赋值, 将满足条件的设置为指定的值-布尔索引

>>> stock\_change[stock\_change > 0.5] = 1

array([[ 1. , -0.72404879, -1.33045773, 1. , 0.3869043 ],

[ 1. , 1. , 0.20815446, -1.67860823, 0.06612823],

[ 1. , 0.42753488, 1. , -0.24375089, 1. ],

[-0.971945 , 1. , -0.95444661, -0.2602084 , -0.48736497],

[-0.32183056, -0.92544956, -0.42126604, 1. , 1. ]])

## 2 通用判断函数

* np.all()

# 判断stock\_change[0:2, 0:5]是否全是上涨的

>>> np.all(stock\_change[0:2, 0:5] > 0)

False

* np.any()

# 判断前5只股票这段期间是否有上涨的

>>> np.any(stock\_change[0:5, :] > 0 )

True

## 3 np.where（三元运算符）

通过使用np.where能够进行更加复杂的运算

* np.where()

# 判断前四个股票前四天的涨跌幅 大于0的置为1，否则为0

temp = stock\_change[:4, :4]

np.where(temp > 0, 1, 0)

* 复合逻辑需要结合np.logical\_and和np.logical\_or使用

# 判断前四个股票前四天的涨跌幅 大于0.5并且小于1的，换为1，否则为0

# 判断前四个股票前四天的涨跌幅 大于0.5或者小于-0.5的，换为1，否则为0

np.where(np.logical\_and(temp > 0.5, temp < 1), 1, 0)

np.where(np.logical\_or(temp > 0.5, temp < -0.5), 1, 0)

## 4 统计运算

**如果想要知道涨幅或者跌幅最大的数据，应该怎么做？**

### 4.1 统计指标

在数据挖掘/机器学习领域，统计指标的值也是我们分析问题的一种方式。常用的指标如下：

* min(a[, axis, out, keepdims])
  + Return the minimum of an array or minimum along an axis.
* max(a[, axis, out, keepdims])
  + Return the maximum of an array or maximum along an axis.
* median(a[, axis, out, overwrite\_input, keepdims])
  + Compute the median along the specified axis.
* mean(a[, axis, dtype, out, keepdims])
  + Compute the arithmetic mean along the specified axis.
* std(a[, axis, dtype, out, ddof, keepdims])
  + Compute the standard deviation along the specified axis.
* var(a[, axis, dtype, out, ddof, keepdims])
  + Compute the variance along the specified axis.

### 4.2 案例：股票涨跌幅统计运算

进行统计的时候，**axis 轴的取值并不一定，Numpy中不同的API轴的值都不一样，在这里，axis 0代表列, axis 1代表行去进行统计**

# 接下来对于这4只股票的4天数据，进行一些统计运算

# 指定行 去统计

print("前四只股票前四天的最大涨幅{}".format(np.max(temp, axis=1)))

# 使用min, std, mean

print("前四只股票前四天的最大跌幅{}".format(np.min(temp, axis=1)))

print("前四只股票前四天的波动程度{}".format(np.std(temp, axis=1)))

print("前四只股票前四天的平均涨跌幅{}".format(np.mean(temp, axis=1)))

如果需要统计出哪一只股票在某个交易日的涨幅最大或者最小？

* np.argmax(temp, axis=)
* np.argmin(temp, axis=)

# 获取股票指定哪一天的涨幅最大

print("前四只股票前四天内涨幅最大{}".format(np.argmax(temp, axis=1)))

print("前四天一天内涨幅最大的股票{}".format(np.argmax(temp, axis=0)))

## 5 小结

* 逻辑运算【知道】
  + 直接进行大于,小于的判断
  + 合适之后,可以直接进行赋值
* 通用判断函数【知道】
  + np.all()
  + np.any()
* 统计运算【掌握】
  + np.max()
  + np.min()
  + np.median()
  + np.mean()
  + np.std()
  + np.var()
  + np.argmax(axis=) — 最大元素对应的下标
  + np.argmin(axis=) — 最小元素对应的下标

**数学：矩阵**

**学习目标**

* 目标
  + 知道什么是矩阵和向量
  + 知道矩阵的加法,乘法
  + 知道矩阵的逆和转置

## 1 矩阵和向量

### 1.1 矩阵

矩阵，英文matrix，**和array的区别矩阵必须是2维的，但是array可以是多维的。**

如图:这个是 3×2 矩阵，即 3 行 2 列，如 m 为行，n 为列，那么 m×n 即 3×2\left[ \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right]​⎣​⎡​​​1​3​5​​​2​4​6​​​⎦​⎤​​矩阵的维数即行数×列数

矩阵元素(矩阵项):A = \left[ \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right]*A*=​⎣​⎡​​​1​3​5​​​2​4​6​​​⎦​⎤​​Aij 指第 i 行，第 j 列的元素。

### 1.2 向量

向量是一种特殊的矩阵，讲义中的向量一般都是列向量，下面展示的就是三维列 向量(3×1)。)A = \left[ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \right]*A*=​⎣​⎡​​​1​2​3​​​⎦​⎤​​

## 2 加法和标量乘法

矩阵的加法:行列数相等的可以加。

例:\left[ \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right] + \left[ \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right] = \left[ \begin{matrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{matrix} \right]​⎣​⎡​​​1​3​5​​​2​4​6​​​⎦​⎤​​+​⎣​⎡​​​1​3​5​​​2​4​6​​​⎦​⎤​​=​⎣​⎡​​​2​6​10​​​4​8​12​​​⎦​⎤​​矩阵的乘法:每个元素都要乘。

例:3 \* \left[ \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right] = \left[ \begin{matrix} 3 & 6 \\ 9 & 12 \\ 15 & 18 \end{matrix} \right]3∗​⎣​⎡​​​1​3​5​​​2​4​6​​​⎦​⎤​​=​⎣​⎡​​​3​9​15​​​6​12​18​​​⎦​⎤​​组合算法也类似。

## 3 矩阵向量乘法

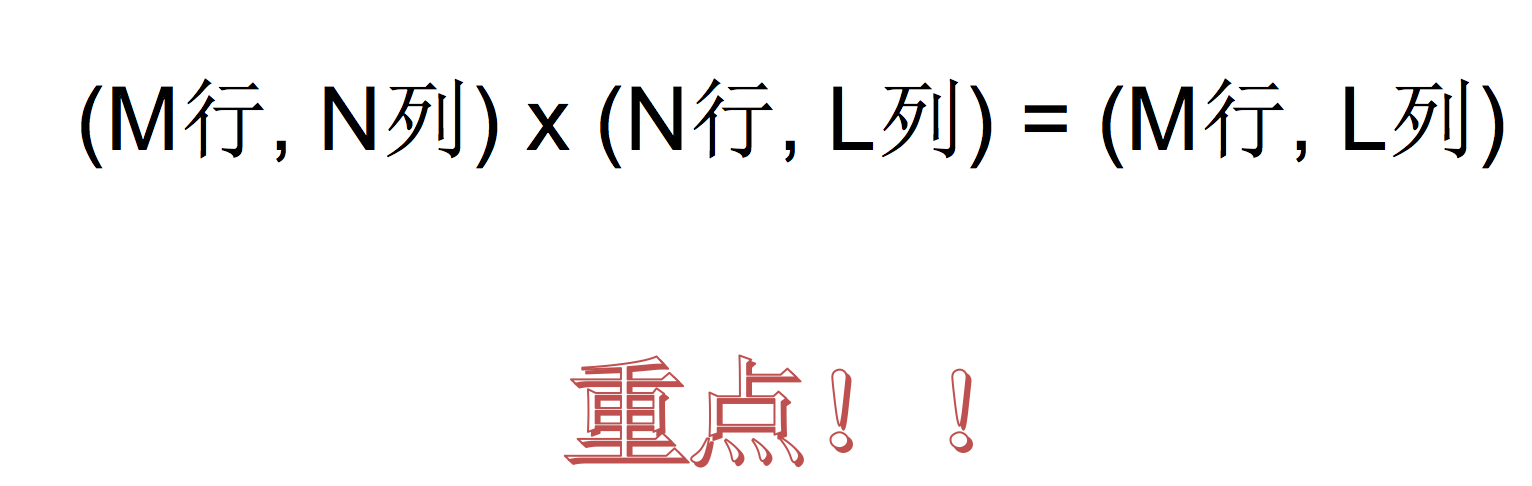
矩阵和向量的乘法如图：m×n 的矩阵乘以 n×1 的向量，得到的是 m×1 的向量

例:\left[ \begin{matrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{matrix} \right] \* \left[ \begin{matrix} 1 \\ 5 \end{matrix} \right] = \left[ \begin{matrix} 16 \\ 4 \\ 7 \end{matrix} \right]​⎣​⎡​​​1​4​2​​​3​0​1​​​⎦​⎤​​∗[​1​5​​]=​⎣​⎡​​​16​4​7​​​⎦​⎤​​

1\*1+3\*5 = 16

4\*1+0\*5 = 4

2\*1+1\*5 = 7

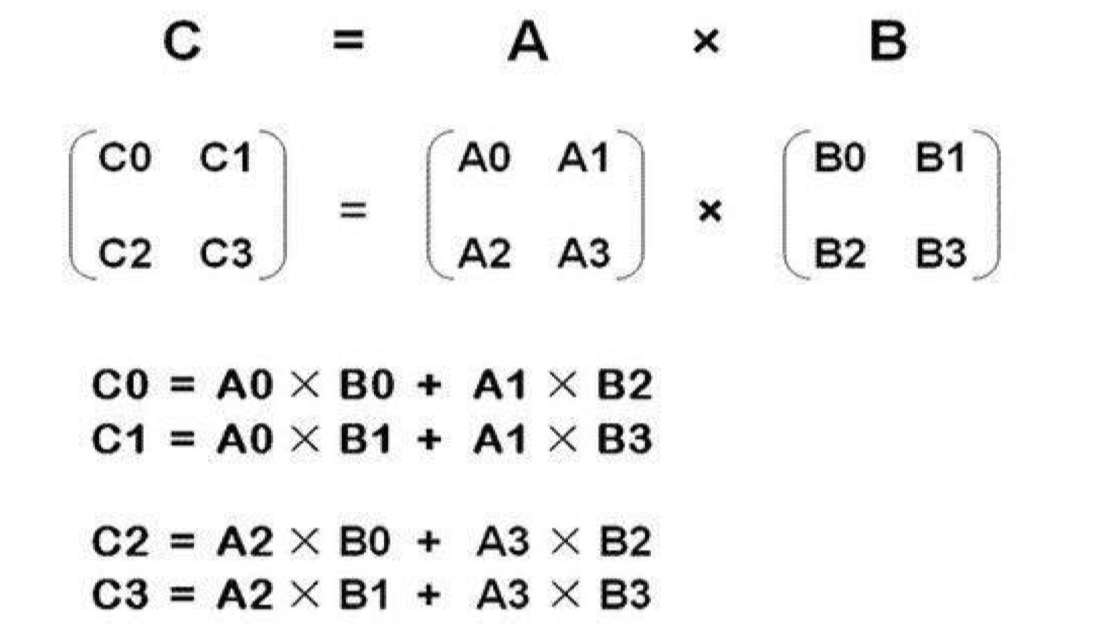


## 4 矩阵乘法

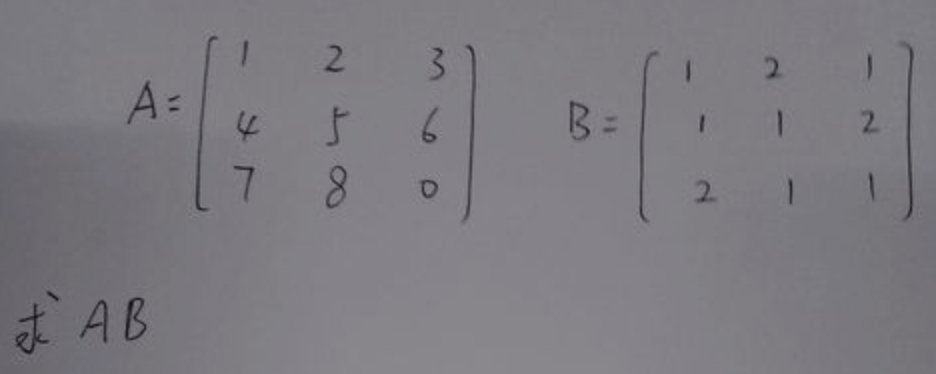
矩阵乘法：

m×n 矩阵乘以 n×o 矩阵，变成 m×o 矩阵。

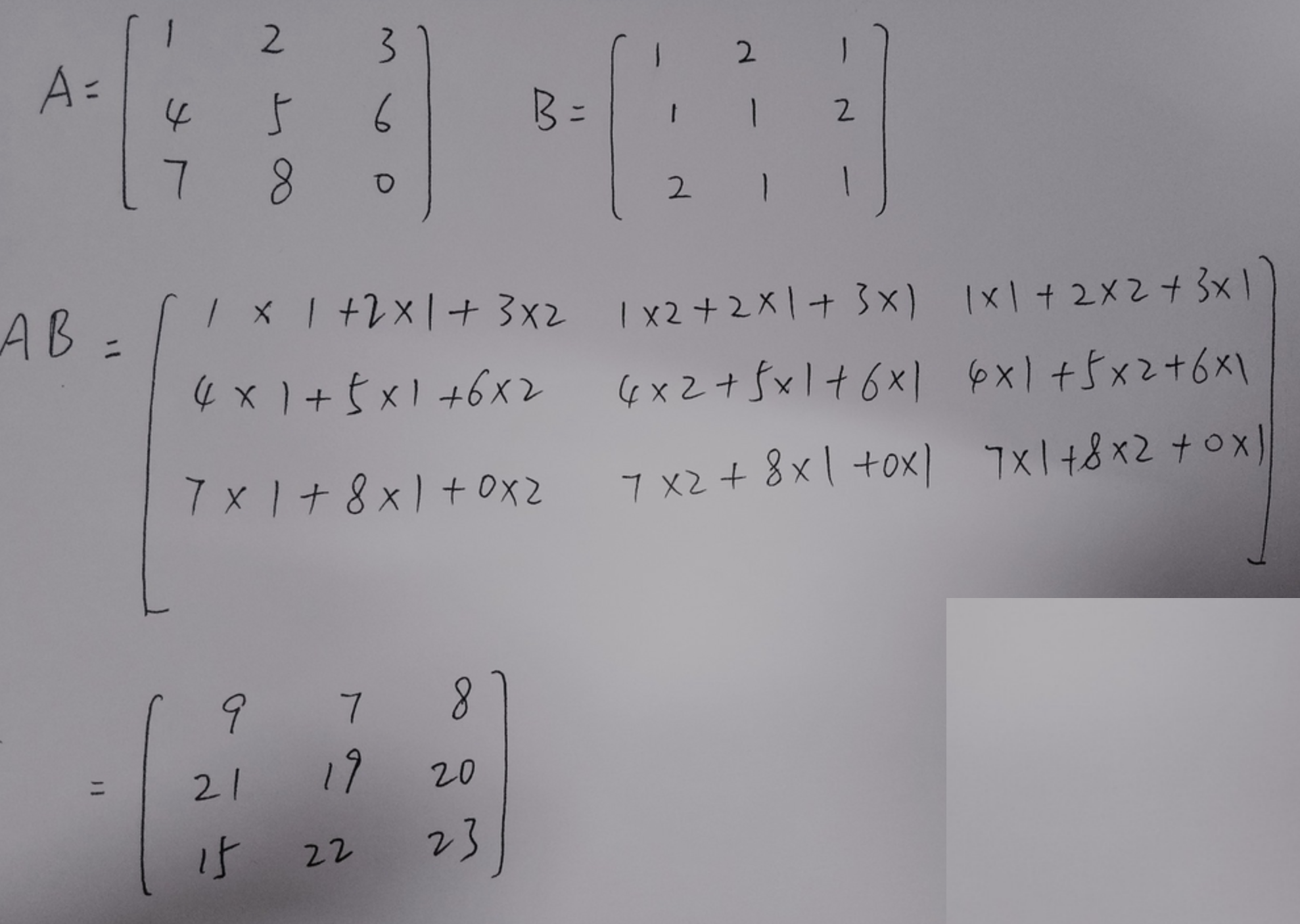
举例：比如说现在有两个矩阵 A 和 B，那 么它们的乘积就可以表示为图中所示的形式。



#### 练一练



答案:

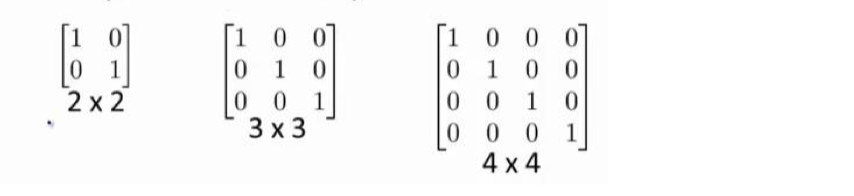


## 5 矩阵乘法的性质

矩阵的乘法不满足交换律：A×B≠B×A

矩阵的乘法满足结合律。即：A×（B×C）=（A×B）×C

单位矩阵：在矩阵的乘法中，有一种矩阵起着特殊的作用，如同数的乘法中的 1,我们称 这种矩阵为**单位矩阵**．它是个方阵，一般用 I 或者 E 表示，从 左上角到右下角的对角线（称为主对角线）上的元素均为 1 以外全都为 0。如：



## 6 逆、转置

矩阵的逆：如矩阵 A 是一个 m×m 矩阵（方阵），如果有逆矩阵，则：

AA-1 = A-1A = I

**低阶矩阵求逆的方法:**

​ 1.待定系数法

​ 2.初等变换

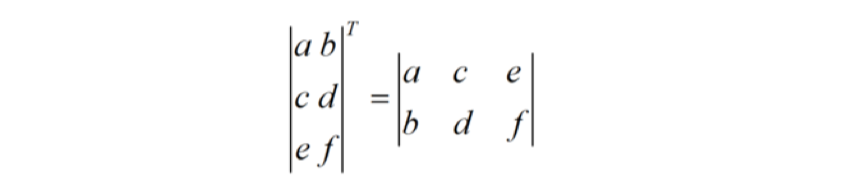
矩阵的转置：设 A 为 m×n 阶矩阵（即 m 行 n 列），第 i 行 j 列的元素是 a(i,j)，即：

A=a(i,j)

定义 A 的转置为这样一个 n×m 阶矩阵 B，满足 B=a(j,i)，即 b (i,j)=a (j,i)（B 的第 i 行第 j 列元素是 A 的第 j 行第 i 列元素），记 AT =B。

直观来看，将 A 的所有元素绕着一条从第 1 行第 1 列元素出发的右下方 45 度的射线作 镜面反转，即得到 A 的转置。

例：



## 7 小结

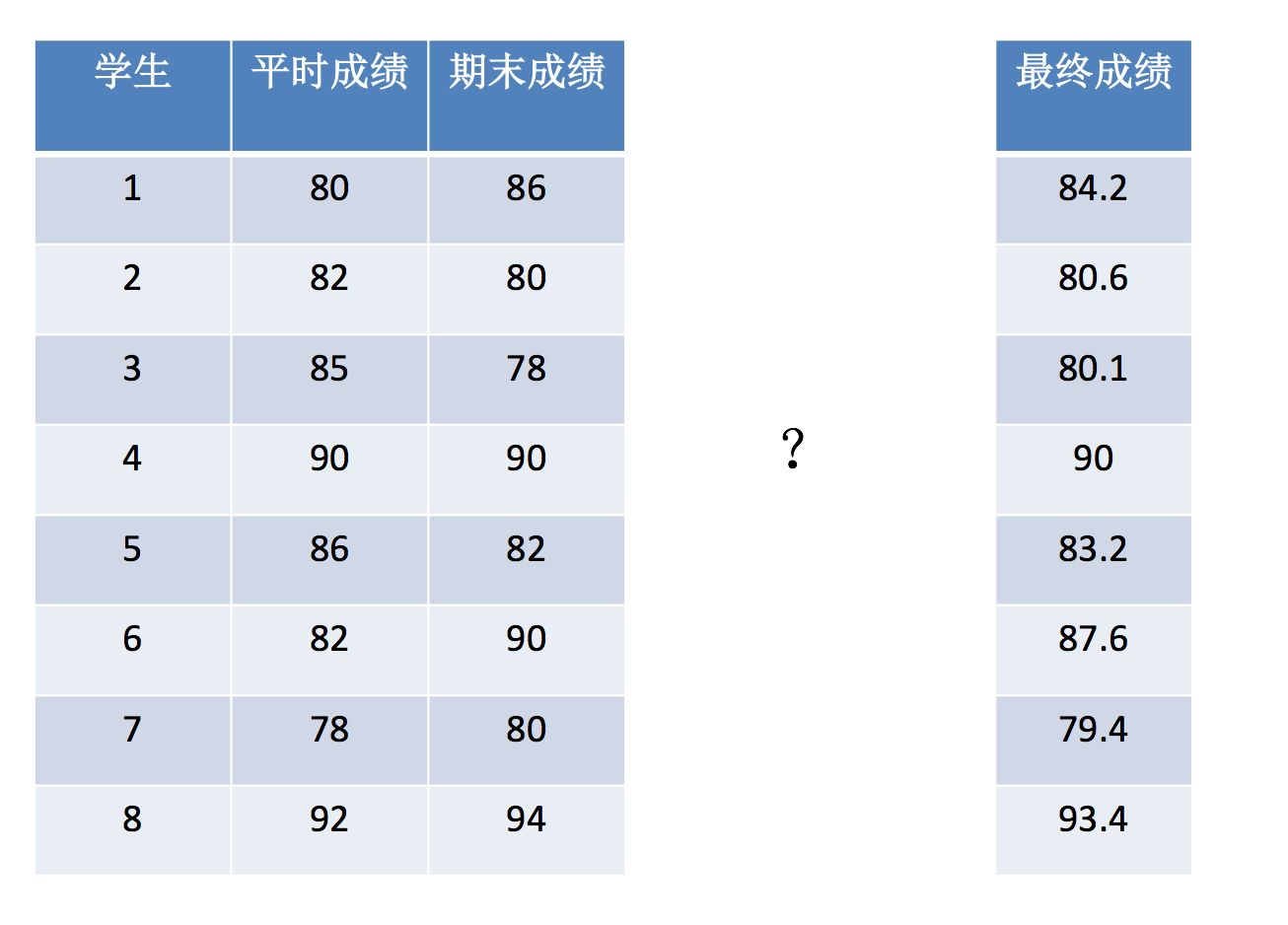
* 1.矩阵和向量【知道】
  + 矩阵就是特殊的二维数组
  + 向量就是一行或者一列的数据
* 2.矩阵加法和标量乘法【知道】
  + 矩阵的加法:行列数相等的可以加。
  + 矩阵的乘法:每个元素都要乘。
* 3.矩阵和矩阵(向量)相乘 【知道】
  + (M行, N列)\*(N行, L列) = (M行, L列)
* 4.矩阵性质【知道】
  + 矩阵不满足交换率,满足结合律
* 5.单位矩阵【知道】
  + 对角线都是1的矩阵,其他位置都为0

**数组间运算**

**学习目标**

* 目标
  + 说明数组间运算的广播机制
  + 知道数组与数之间的运算
  + 知道数组与数组之间的运算
  + 理解矩阵的特点以及运算规则
  + 应用np.matmul实现矩阵运算
* 应用
  + 学生综合成绩矩阵运算

## 1 场景



数据：

[[80, 86],

[82, 80],

[85, 78],

[90, 90],

[86, 82],

[82, 90],

[78, 80],

[92, 94]]

## 2 数组与数的运算

arr = np.array([[1, 2, 3, 2, 1, 4], [5, 6, 1, 2, 3, 1]])

arr + 1

arr / 2

# 可以对比python列表的运算，看出区别

a = [1, 2, 3, 4, 5]

a \* 3

## 3 数组与数组的运算

arr1 = np.array([[1, 2, 3, 2, 1, 4], [5, 6, 1, 2, 3, 1]])

arr2 = np.array([[1, 2, 3, 4], [3, 4, 5, 6]])

上面这个能进行运算吗，结果是不行的！

## 4 广播机制

**执行 broadcast 的前提在于，两个 ndarray 执行的是 element-wise的运算，Broadcast机制的功能是为了方便不同形状的ndarray（numpy库的核心数据结构）进行数学运算。**

当操作两个数组时，numpy会逐个比较它们的shape（构成的元组tuple），只有在下述情况下，两个数组才能够进行数组与数组的运算。

* 维度相等
* shape（其中相对应的一个地方为1）

例如：

Image (3d array): 256 x 256 x 3

Scale (1d array): 3

Result (3d array): 256 x 256 x 3

A (4d array): 9 x 1 x 7 x 1

B (3d array): 8 x 1 x 5

Result (4d array): 9 x 8 x 7 x 5

A (2d array): 5 x 4

B (1d array): 1

Result (2d array): 5 x 4

A (3d array): 15 x 3 x 5

B (3d array): 15 x 1 x 1

Result (3d array): 15 x 3 x 5

如果是下面这样，则不匹配：

A (1d array): 10

B (1d array): 12

A (2d array): 2 x 1

B (3d array): 8 x 4 x 3

**思考：下面两个ndarray是否能够进行运算？**

arr1 = np.array([[1, 2, 3, 2, 1, 4], [5, 6, 1, 2, 3, 1]])

arr2 = np.array([[1], [3]])

## 5 矩阵运算

### 5.1 矩阵乘法api：

* np.matmul
* np.dot

>>> a = np.array([[80, 86],

[82, 80],

[85, 78],

[90, 90],

[86, 82],

[82, 90],

[78, 80],

[92, 94]])

>>> b = np.array([[0.7], [0.3]])

>>> np.matmul(a, b)

array([[81.8],

[81.4],

[82.9],

[90. ],

[84.8],

[84.4],

[78.6],

[92.6]])

>>> np.dot(a,b)

array([[81.8],

[81.4],

[82.9],

[90. ],

[84.8],

[84.4],

[78.6],

[92.6]])

**np.matmul和np.dot的区别:**

二者都是矩阵乘法。 np.matmul中禁止矩阵与标量的乘法。 在矢量乘矢量的內积运算中，np.matmul与np.dot没有区别。

### 5.2 矩阵应用场景

**大部分机器学习算法需要用到**

## 6 小结

* 1.数组运算,满足广播机制,就OK【知道】
  + 1.维度相等
  + 2.shape(其中对应的地方为1,也是可以的)
* 2.矩阵运算【掌握】
  + np.matmul
  + np.dot
  + 注意：二者都是矩阵乘法。 np.matmul中禁止矩阵与标量的乘法。 在矢量乘矢量的內积运算中，np.matmul与np.dot没有区别。