

# NumPy

机器学习是当今科技领域最热门的话题之一。作为一种强大的工具，Python语言在机器学习领域大放异彩。而在Python生态系统中，NumPy库扮演着重要的角色，它提供了丰富的功能和高效的数据结构，使得机器学习任务更加便捷和高效。

## 一. NumPy简介

NumPy (Numerical Python) 是一个开源的Python库，提供了高性能的多维数组对象和用于处理数组的函数。它是许多科学计算和数据分析任务的基础库之一。

NumPy 是一个 Python 包。它代表 “Numeric Python”。它是一个由多维数组对象和用于处理数组的例程集合组成的库。

Numeric, 即 NumPy 的前身, 是由 Jim Hugunin 开发的。也开发了另一个包 Numarray, 它拥有一些额外的功能。2005年, Travis Oliphant 通过将 Numarray 的功能集成到 Numeric 包中来创建 NumPy 包。这个开源项目有很多贡献者。

NumPy是一个开源的Python科学计算基础库, 包含:

- 一个强大的N维数组对象 ndarray;
  - 广播功能函数;
  - 整合C/C++/Fortran代码的工具, 底层算法在设计时就有着优异的性能, 使得numpy比纯python代码高效得多;
  - 线性代数、傅里叶变换、随机数生成等功能; NumPy是SciPy、Pandas等数据处理或科学计算库的基础。
- Numpy的主要对象是同种元素的多维数组。这是一个所有的元素都是一种类型、通过一个正整数元组索引的元素表格(通常元素是数字)。在Numpy中维度(dimensions)叫做轴(axes), 轴的个数叫做秩(rank)。

## 二.安装NumPy

在开始之前, 我们需要安装NumPy库, 安装的方式有两种:

方式一:通过命令行进行安装

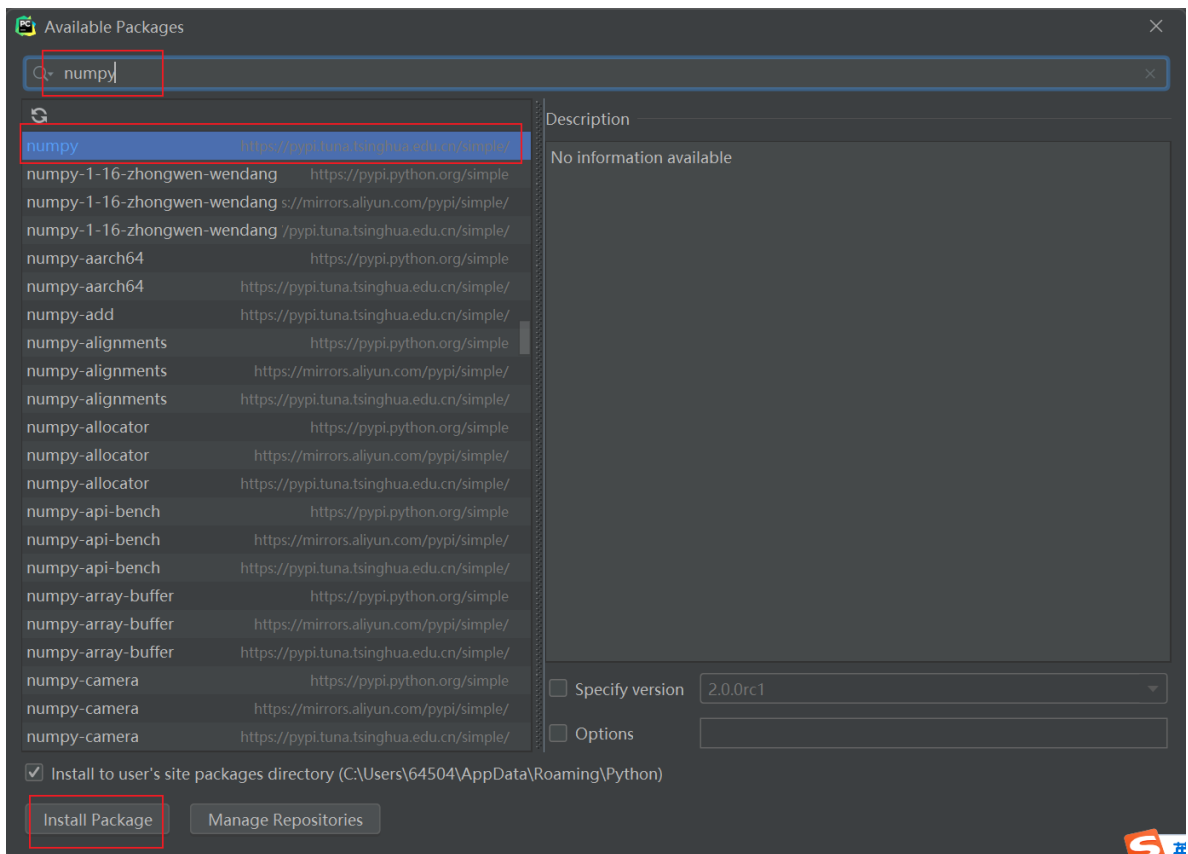


```
html > body > script > check_pwd()

Terminal: Local x +
Microsoft Windows [版本 10.0.22000.2538]
(c) Microsoft Corporation。保留所有权利。

D:\hkyx\数据分析\python基础\day01\代码\demo1>pip install numpy
```

方式二:



### 三.体验Numpy 多维数组对象

题目：数组a与数组b相加，数组a是1~N数字的立方，数组b是1~N数字的平方

```
def arr_add(n):  
    a = [i**3 for i in range(1,n+1)]  
    b = [i**2 for i in range(1,n+1)]  
    c = []  
    for i in range(n):  
        c.append(a[i]+b[i])  
    return c  
print(arr_add(3))
```

使用NumPy实现:

```
使用Numpy计算上面的题目:  
import numpy as np  
def arr_add(n):  
    a = np.arange(1,n+1) ** 3  
    b = np.arange(1,n+1) ** 2  
    return a+b  
print(arr_add(3))
```

由于Python是先循环遍历再计算，Numpy直接计算，计算数量越大越节省时间。

#### 3.1 创建数组的方法

```
import numpy as np
a = np.array([1,2,3,4,5])
b = np.array(range(1,6))
c = np.arange(1,6)
print(a)
print(b)
print(c)
print(a.dtype) # int32或int64
print(type(a)) # <class 'numpy.ndarray'>
```



以上三种方法结果是一样的，注意一下输入结果是数组

**array:** 将输入数据（可以是列表、元组、数组以及其它序列）转换为ndarray(Numpy数组)，如不显示指明数据类型，将自动推断，默认复制所有输入数据。

**arange:** Python内建函数range的数组版，返回一个数组。

array的属性:

- **shape:** 返回一个元组，表示 array的维度 [形状，几行几列] （2，3）两行三列，（2，2，3）两个两行三列
- **ndim:** 返回一个数字，表示array的维度的数目
- **size:** 返回一个数字，表示array中所有数据元素的数目
- **dtype:** 返回array中元素的数据类型

### 3.1.1 arange创建数字序列

使用arange创建数字序列:

```
np.arange([开始,]结束[,步长],dtype=None)
np.arange(5) 返回 array([0,1,2,3,4])
np.arange(1,10,2) 返回 array([1,3,5,7,9])
```

### 3.1.2 使用ones创建全是1的数组

```
np.ones(shape,dtype=None,order='C')
a=np.ones(3) # 返回 array([1. 1. 1.])
b=np.ones((2,3))
```

参数:

**shape:** 整数或者整型元组定义返回数组的形状；可以是一个数（创建一维向量），也可以是一个元组（创建多维向量）

**dtype :** 数据类型，可选定义返回数组的类型。

### 3.1.3 ones\_like创建形状相同的数组

```
np.ones_like(a, dtype=float, order='C', subok=True)
```

返回：与a相同形状和数据类型的数组，并且数组中的值都为1

参数：

**a**：用a的形状和数据类型，来定义返回数组的属性

**dtype**：数据类型，可选

案例：

案例1：以下数组是x

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

```
>>> np.ones_like(x)
```

```
array([[1, 1, 1],  
       [1, 1, 1]])
```

案例2：以下数组是y

```
array([ 0., 1., 2.])
```

```
>>> np.ones_like(y)
```

```
array([ 1., 1., 1.])
```

### 3.1.4 full创建指定值的数组

```
np.full(shape, fill_value, dtype=None, order='C')
```

参数：

**shape**：整数或者整型元组定义返回数组的形状；可以是一个数（创建一维向量），也可以是一个元组（创建多维向量）

**fill\_value**：标量（就是纯数值变量）

**dtype**：数据类型，可选定义返回数组的类型。

案例：

```
np.full(3, 520) 返回 [520 520 520]
```

```
np.full((2, 4), 520)
```

### 3.1.5 full\_like创建开关相同的指定值数组

```
np.full_like(a, fill_value, dtype=None)
```

例：

案例1：以下数组是x

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

```
>>> np.zeros_like(x, 520)
```

```
array([[520, 520, 520],  
       [520, 520, 520]])
```

案例2：以下数组是y

```
array([ 0., 1., 2.])
```

```
>>> np.zeros_like(y, 520)
```

```
array([ 520., 520., 520.])
```

## 3.2 给数据指定数据类型

```
import numpy as np
a = np.array(range(1,8),dtype=float) # 修改数据类型
b = np.array(range(1,8),dtype='float32') # 修改数据类型和位数
print(a)
print(b)
print(a.dtype)
print(b.dtype)
print(type(a))
print(type(b))
```

### 3.3 多维数组

```
import numpy as np
a = np.array([1,2,3,4,5,6])
b = np.array([
    [1,2,3],
    [4,5,6]
])
print(a)
print(b)
```

`np.array([[1,2],[3,4]])`



1	2
3	4

**a是一维数组，b是二维数组**

**一维数组的定义：**当数组中每个元素都只带有一个下标时，称这样的数组为一维数组，一维数组实质上是一组相同类型数据的线性集合。

**二维数组的定义：**二维数组本质上是以数组作为数组元素的数组，即数组的数组。

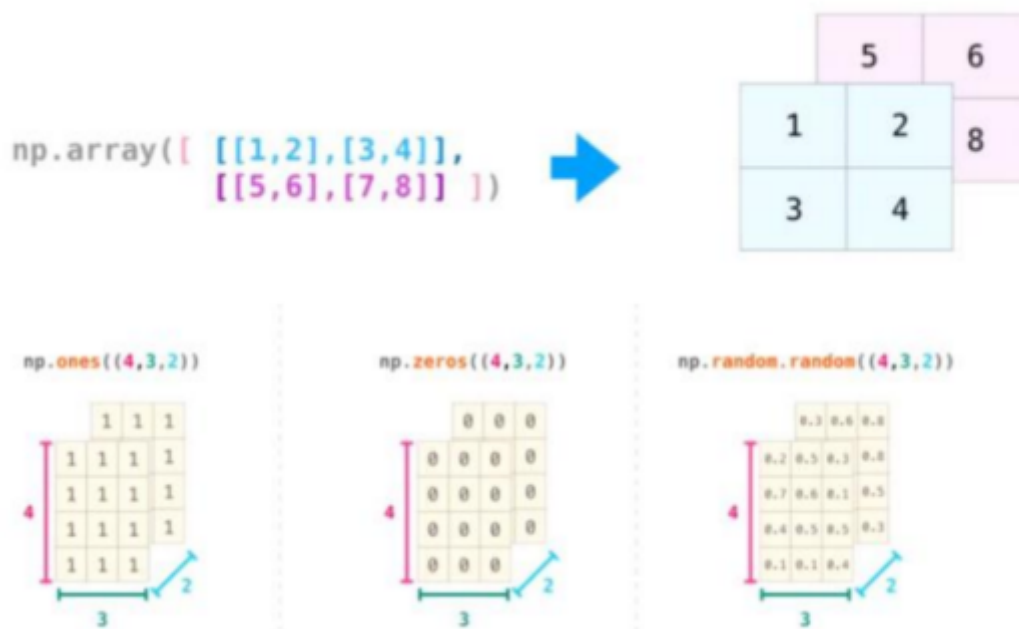
```
print(a.shape) # 返回一个元组，查看矩阵或者数组的维数(有几个数就是几维)，就是几乘几的数组
print(b.shape)
print(a.ndim) # 返回数组维度数目
print(b.ndim)
print(a.size) # 返回数组中所有元素的数目
print(b.size)
print(a.dtype) # 返回数组中所有元素的数据类型
print(b.dtype)
```

使用 `np.ones()`、`np.zeros()` 等方法：



这样就很容易理解括号里 (3,2) 的含义。

Numpy 不仅可以处理上述的一维数组和二维矩阵（二维数组我们习惯叫它为矩阵），还可以处理任意 N 维的数组，方法也大同小异。



### 3.3.1 reshape不改值修改形状

```
import numpy as np
import random
a = np.arange(10).reshape(2,5) # 变成2行5列
b = a.reshape(10) # 变回1行1列
c = a.flatten() # 不清楚对方什么阵型，直接转一维
print(b)
```

能变成2行5列，就能变回1行10个元素，想让他变成一维就给它传一个数，这个数必需是这个数组所有\*\*值的全部个数。

`a.reshape(1,10)` 等于 `a.reshape(10)`

但是如果我不清楚这个数组中有多少个数，但是仍然想转成一维

`a.flatten()`

作用：将一个多维数组展开变成一个一维数组

### 3.3.2 数组计算

```
import numpy as np
a = np.arange(10).reshape(2,5)
print(a)
```

arange: 生成0-9共10个数  
reshape: 这个方法是在不改变数据内容的情况下，改变一个数组的格式

```
D:\python39\python.e
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

```
print(a+1)
```

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

```
print(a*3)
```

```
[[ 0  3  6  9 12]
 [15 18 21 24 27]]
```

凡是形状一样的数组，假设数组a和数组b，可以直接用a+b 或 a-b

```
import numpy as np
import random
a = np.arange(10).reshape(2,5)
b = np.random.randn(2,5)
print(a+b)
print(a-b)
```

总结：

- (1) 形状一样的数组按对应位置进行计算。
- (2) 一维和多维数组是可以计算的，只要它们在某一维度上是一样的形状，仍然是按位置计算。

**例如：a是一个2行5列的数组，b是一个1行5列的数组**

a	<table><tr><td>3</td><td>4</td><td>5</td><td>7</td><td>2</td></tr><tr><td>2</td><td>5</td><td>3</td><td>5</td><td>4</td></tr></table>	3	4	5	7	2	2	5	3	5	4
3	4	5	7	2							
2	5	3	5	4							
b	<table><tr><td>2</td><td>1</td><td>3</td><td>2</td><td>2</td></tr></table>	2	1	3	2	2					
2	1	3	2	2							
a-b	<table><tr><td>1</td><td>3</td><td>2</td><td>5</td><td>0</td></tr><tr><td>0</td><td>4</td><td>0</td><td>3</td><td>2</td></tr></table>	1	3	2	5	0	0	4	0	3	2
1	3	2	5	0							
0	4	0	3	2							
a*b	<table><tr><td>6</td><td>4</td><td>15</td><td>14</td><td>0</td></tr><tr><td>4</td><td>5</td><td>9</td><td>10</td><td>8</td></tr></table>	6	4	15	14	0	4	5	9	10	8
6	4	15	14	0							
4	5	9	10	8							

### 3.3.3 广播原则

- 相同形状：数组计算

x	2	2	3
	1	2	3

y	1	1	3
	2	2	4

x*y	2	2	9
	2	4	12

- 不同形状：数组计算

x	2	2	3
	1	2	3
y	1	1	3
x*y	2	2	9
	1	2	9

广播的原则：

如果两个数组的后缘维度（**trailing dimension**，即从末尾开始算起的维度）的轴长度相符，或其中的一方的长度为1，则认为它们是广播兼容的。广播会在缺失和（或）长度为1的维度上进行。

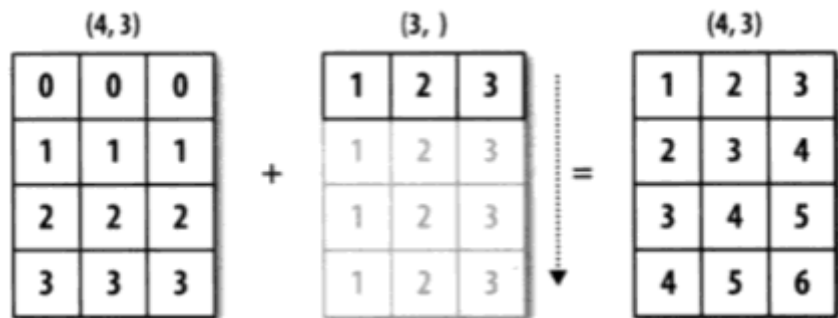
这句话乃是理解广播的核心。广播主要发生在两种情况，一种是两个数组的维数不相等，但是它们的后缘维度的轴长相符，另外一种是有的一方的长度为1。

**数组维度不同，后缘维度的轴长相符：**



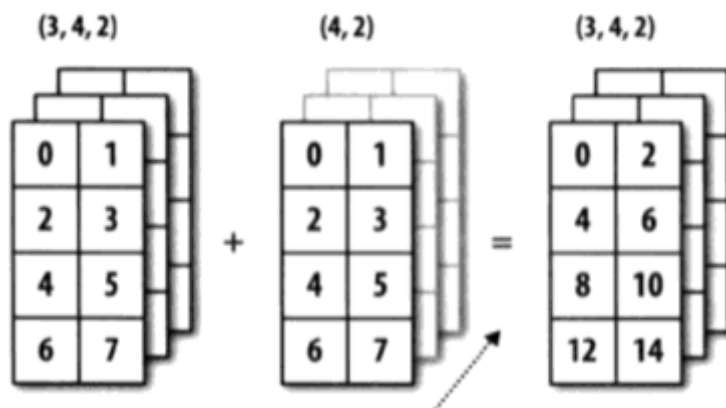
x	0	0	0
	1	1	1
	2	2	2
	3	3	3
y	1	2	3
x+y	1	2	3
	2	3	4
	3	4	5
	4	5	6

x是4行3列 (4, 3)  
y是1行3列 (3, )  
后缘维度都是3，所以后缘维度相同



上例中x的shape为 (4,3) , y的shape为 (3, ) 。可以说x是二维的，而y是一维的。但是它们的后缘维度相等，x的第二维长度为3，和y的维度相同。x和y的 shape并不一样，但是它们可以执行相加操作，这就是通过广播完成的。

同样的例子还有：



从上面的图可以看到，(3,4,2) 和 (4,2) 的维度是不相同的，前者为3维，后者为2维。但是它们后缘维度的轴长相同，都为 (4,2) 。同样，还有一些例子：(4,2,3) 和 (2,3) 是兼容的，(4,2,3) 还和 (3) 是兼容的，后者需要在两个轴上面进行扩展。

数组维度相同，其中有个轴为1：

(4, 3)				(4, 1)				(4, 3)		
0	0	0	+	1	1	1	=	1	1	1
1	1	1		2	2	2		3	3	3
2	2	2		3	3	3		5	5	5
3	3	3		4	4	4		7	7	7

x的shape为 (4,3) ， y的shape为 (4,1) ， 它们都是二维的，但是第二个数组在1轴上的长度为1， 所以可以进行广播。

### 3.4 基础索引与切片

```
import numpy as np
a = np.arange(10) # 一维数组使用小写
A = np.arange(20).reshape(4,5) # 多维数组使用大写
```

	data	data[0]	data[1]	data[0:2]	data[1:]
0	1	1		1	
1	2		2	2	2
2	3				3

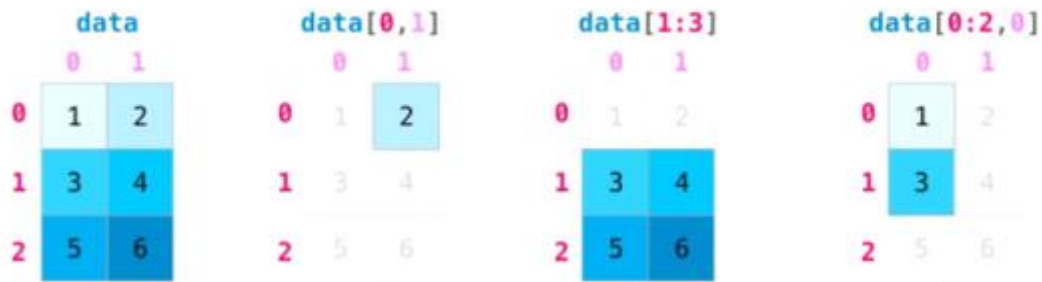
- 一维数组

```
a = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(a[3],a[5],a[-1]) # 返回 3 5 9
print(a[2:4]) # 返回 array([2, 3])
```

- 二维数组

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

## 矩阵索引



A[0,0] # 取数组A的0行0列, 返回值0  
A[-1,2] # 取最后一行的第2列, 返回值17  
A[2] # 取第2行所有的列, 返回array([10, 11, 12, 13, 14])  
A[-1] # 取最后1行  
A[0:-1] # 取除了最后1行之外其它所有行  
A[0:2,2:4] #取0和1行, 2和3列  
A[:,2] # 取所有行中的第2列

- 切片修改

由于Numpy经常处理大数组, 避免每次都复制, 所以切片修改时直接修改了数组  
a[4:6] = 520 # 返回array([ 0, 1, 2, 3, 520, 520, 6, 7, 8, 9])  
A[:,1,:2]=520 # 修改0行1列的两个值

## 3.5 布尔索引

- 一维数组

```
import numpy as np
数组 = np.arange(10)
print(数组)
筛选 = 数组 > 5
print(筛选) # 返回False和True
print(数组[筛选]) # 返回6 7 8 9
```

实例1: 把一维数组进行01化处理

假设这10个数字, 我想让大于5的数字变成1, 小于等于5的数字变成0

```
arr1[arr1<=5] = 0 # 小于5的重新赋值为0
arr1[arr1>5] = 1 # 大于5的重新赋值为1
print(arr1)
```

实例2: 进行自增量的操作, 给大于5的加上520

```
arr1[arr1>5] += 520
print(arr1)
```

- 二维数组

```
import numpy as np
arr1 = np.arange(1,21).reshape(4,5)
print(arr1)
test = arr1>10
print(arr1) # 返回一个布尔数组，即有行又有列
print(arr1[test]) # 返回所有为True的对应数字组成的数组，以一维数组展现
```

```
# 例：把第3例大于5的行筛选出来并重新赋值为520
import numpy as np
数组 = np.arange(1,21).reshape(4,5)
print(数组)
print("-"*30)
print(数组[:,3]) # 所有行，第3列
print("-"*30)
筛选 = 数组[:,3] > 5 # 所有行第3列，大于5的
数组[数组[:,3]>5] = 520
print(数组)
```

```
例子：找出偶数或小于7的数
import numpy as np
数组 = np.arange(10)
print(数组)
print("-"*30)
条件 = (数组%2==0) | (数组<7)
print(条件)
print("-"*30)
print(数组[条件])
```

## 3.6 神奇索引

**神奇索引：使用整数数组进行数据索引。**

a     

3	6	7	9	5	2	7
---	---	---	---	---	---	---

**a[[2,3,5]] # 返回对应下标的一数组 array([7,9,2])**

```
import numpy as np
数组 = np.arange(36).reshape(9,4)
print(数组)
print("-"*15)
print(数组[[4,3,0,6]]) # 返回第4行，第3行，第0行，第6行

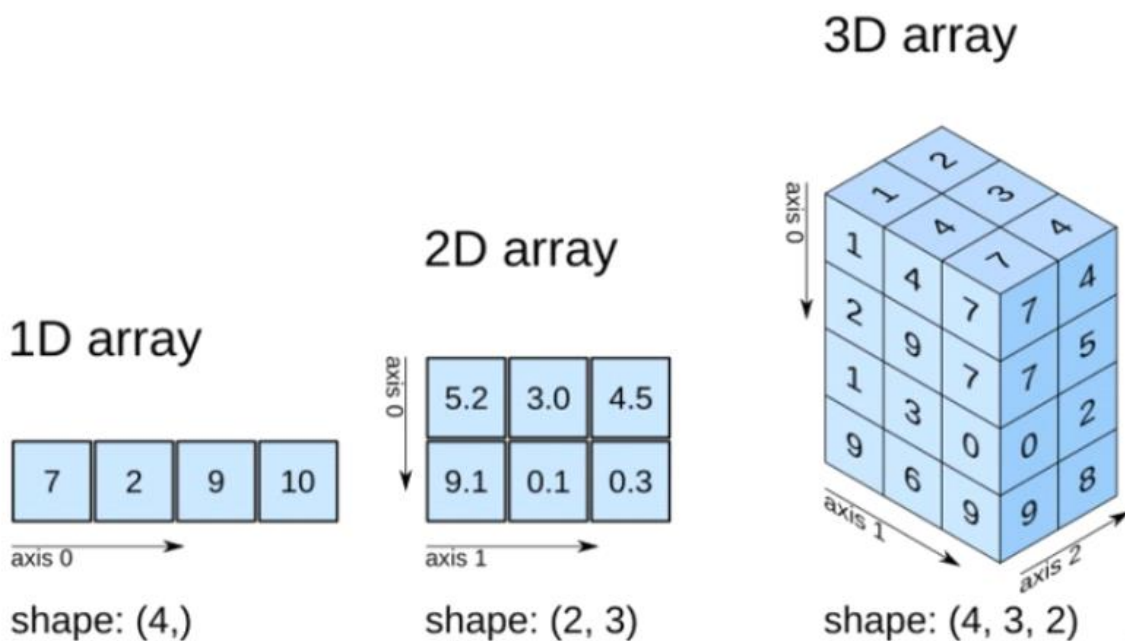
import numpy as np
数组 = np.arange(32).reshape((8,4))
print(数组)
读取 = 数组[[1,5,7,2],[0,3,1,2]] # 取第1行第0列，第5行第3列，第7行第1列，第2行第2列
print(读取)

import numpy as np
数组 = np.arange(10)
索引 = np.array([[0,2],[1,3]])
print(数组)
print("-"*30)
print(数组[索引])
```

## 实例：获取数组中最大的前N个数字

```
# 获取数组中最大的前N个数字
import numpy as np
数组 = np.random.randint(1,100,10)
print(数组)
# 数组.argsort()会返回排序后的下标
# 取最大值对应的3个下标，因为默认升序，所以要用-3,从倒数第3个到最后一个
下标 = 数组.argsort()[-3:]
print(下标) # 返回的是最大3个数在数组中的下标
# 将下标传给数组
最大 = 数组[下标]
print(f'最大的三个数是{最大}')
```

## 四.Numpy的轴



```
数组=np.array([[[1,2],[4,5],[7,8]],[[8,9],[11,12],[14,15]],[[10,11],[13,14],[16,17]],[[19,20],[22,23],[25,26]]])
```

```
print(数组.shape) # 返回 (4, 3, 2)
```

最内层一对 [ ] 可以代表一个1维数组  
加粗的一对 [ ] 里面有3个一维数组，也就是2维数组  
最外层的一对 [ ] 里面有3个2维数组也就是3维数组  
0轴是行，1轴是列，2轴是纵深  
数组的shape维度是(4,3,2)，元组的索引为 [ 0,1,2 ]  
假设维度是(2, 3)，元组的索引为[0,1]  
假设维度是(4, ) 元组的索引为[0]

可以看到轴编号和shape元组的索引是对应的，所以这个编号可以理解是高维nd.array.shape产生的元组的索引

我们知道shape(4,3,2)表示数组的维度，既然shape的索引可以看做轴编号，那么一条轴其实就是一个维度

0轴对应的是最高维度3维，1轴对应2维，2轴对应的就是最低维度的1维

总结：凡是提到轴，先看数组的维度，有几维就有几个轴

## 2.1 沿轴切片

```
import numpy as np
数组=np.array([ [1,2,3] , [4,5,6] , [7,8,9] ])
print(数组)
print(数组.shape)
```

数组的维度是（3，3），这个元组的索引是 [0,1]，表示这个2维数组有两条轴：0轴和1轴

- 首先看1个参数的切片操作

```
print(数组[0:2])
```

这里有个很重要的概念， :2 是切片的第一个参数，约定俗成第一个参数就代表0轴  
0轴表示2维，所以这个切片是在2维这个维度上切的，又叫“沿0轴切”。

		1轴		
		0	1	2
0轴	0	1	2	3
	1	4	5	6
	2	7	8	9

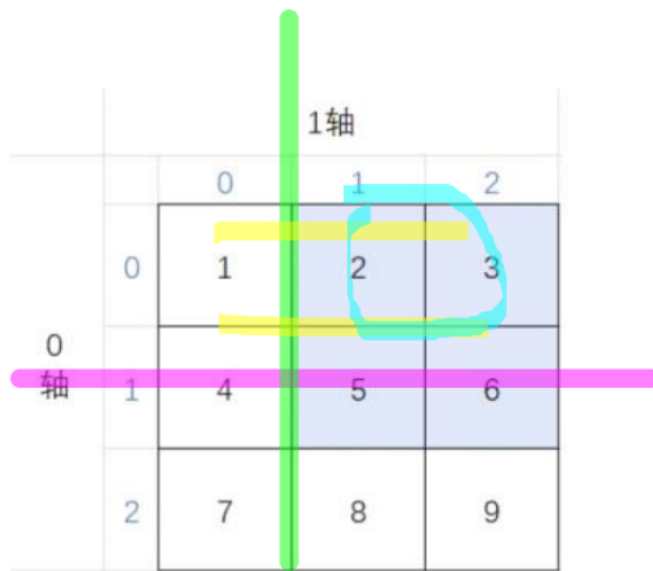
这个2维数据是由3个1维数组组成的，这3个1维数组当然也有索引号也是[0,1,2]，[:2]就表示它要切取2维（0轴）上3个1维数组中的索引[0]和索引[1]，于是得到([1, 2, 3])和([4, 5, 6])这两个1维数组。

- 再看第二个参数的切片操作

```
print(数组[:2,1:])
```

就是在两个维度（轴）上各切一刀，第1个参数就是2维（0轴），:2表示切取2维（0轴）上的索引[0]和索引[1]，即([1, 2, 3])和([4, 5, 6])这两个1维数组

第2个参数就是1维（1轴），1:表示切取1维（1轴）上的索引[1]和索引[2]，即对数组([1, 2, 3])取([2,3])，对数组([4, 5, 6])取([5,6])



## 五.numpy数组装换轴

data

1	2
3	4
5	6

data.T

1	3	5
2	4	6

reshape() 用法:

data

1
2
3
4
5
6

data.reshape(2,3)

1	2	3
4	5	6

Dimensions: 2 (rows), 3 (columns)

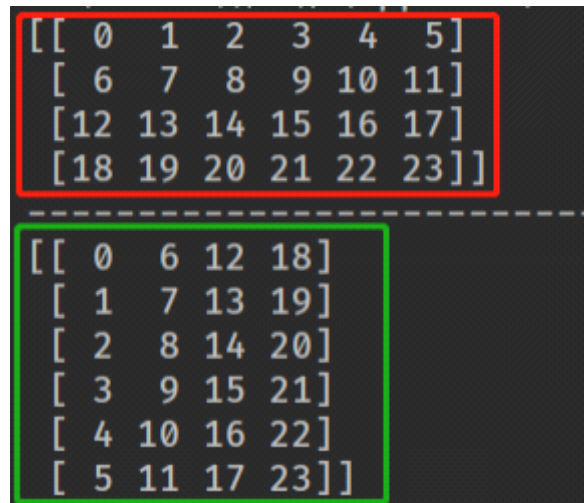
data.reshape(3,2)

1	2
3	4
5	6

Dimensions: 3 (rows), 2 (columns)

### 5.1 transpose方法【行列装置】

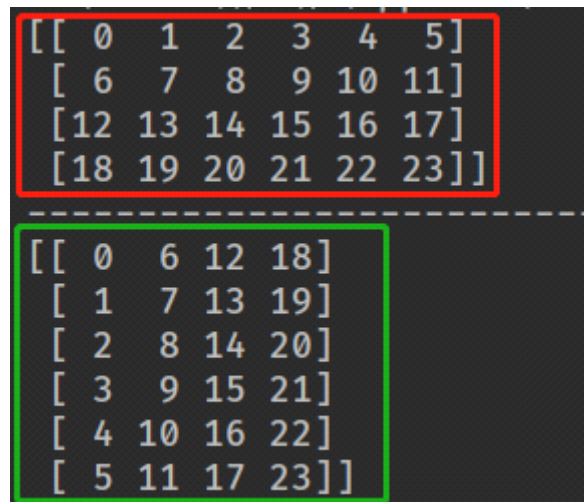
```
import numpy as np
数组=np.arange(24).reshape((4,6))
print(数组)
print("-"*30)
print(数组.transpose())
```



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

## 5.2 swapaxes方法【轴装置】

```
import numpy as np
数组=np.arange(24).reshape((4,6))
print(数组)
print("-"*30)
print(数组.swapaxes(1,0))
```



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

## 六.使用数组进行面向数组编程

使用Numpy数组可以使你利用简单的数组表达式完成多种数据操作任务，而无须写大量的循环，这种利用数组表达式来替代循环的方法，称向量化。

通常，向量化的数组操作会比纯Python的等价实现在速度上快一到两个数量级（甚至更多）

### 6.1 数学和统计方法



方法	描述
sum	沿着轴向计算所有元素的累和，0长度的数组累和为0
average	加权平均，参数可以指定weights
prod	所有元素的乘积
mean	数学平均，0长度的数组平均值为NaN
std,var	标准差和方差，可以选择自由度调整（默认分母是n）
min,max	最小和最大值
argmin,argmax	最小和最大值的 <b>位置</b>
cumsum	从0开始元素累积和
cumprod	从1开始元素累积积
median	中位数
prercentile	0-100百分位数
quantile	0-1分位数

### 6.1.1 平均数，加权平均数，中位数，众数

为筹备班级的初中毕业联欢会，班长对全班学生爱吃哪几种水果作了民意调查。那么最终买什么水果，下面的调查数据中最值得关注的是（ ）

A. 中位数 B. 平均数 C. 众数 D. 加权平均数

1、平均数：所有数加在一起求平均

2、中位数：对于有限的数集，可以通过把所有观察值高低排序后找出正中间的一个作为中位数。如果观察值有偶数个，通常取最中间的两个数值的平均数作为中位数。

3、众数：出现次数最多的那个数

4、加权平均数：加权平均值即将各数值乘以相应的权数，然后加总求和得到总体值，再除以总的单位数。加权平均值的大小不仅取决于总体中各单位的数值（变量值）的大小，而且取决于各数值出现的次数（频数），由于各数值出现的次数对其在平均数中的影响起着权衡轻重的作用，因此叫做权数。因为加权平均值是根据权数的不同进行的平均数的计算，所以又叫加权平均数。在日常生活中，人们常常把“权数”理解为事物所占的“权重”

x占a% y占b% z占c% n占m%

加权平均数=  $(ax+by+cz+mn) / (x+y+z+n)$

### 6.1.2 一维数组

沿轴向进行计算，一维数组只有一个0轴

```
import numpy as np
a = np.array([1,2,3,4,3,5,3,6])
print(f'数组: {a}')
print(np.sum(a))
print(np.prod(a))
print(np.cumsum(a)) # 从0开始元素的累积和
print(np.cumprod(a)) # 从1开始元素的累积积
print(np.max(a))
print(np.min(a))
print(np.argmax(a)) # 最大值所在的下标
print(np.argmin(a)) # 最小值所在的下标
print(np.mean(a)) # 平均数
print(np.median(a)) # 中位数
```

```
print(np.average(a)) # 加权平均
counts = np.bincount(a) # 统计非负整数的个数，不能统计浮点数
print(np.argmax(counts)) # 返回众数，此方法不能用于二维数组
```

### 6.1.3 二维数组

```
import numpy as np
from scipy import stats
a = np.array([[1,3,6],[9,2,3],[2,3,3]])
print(f'数组: \n{a}')
print('-'*30)
print(np.sum(a))
print(np.prod(a))
print(np.cumsum(a)) # 从0开始元素的累积和，返回一维数组
print(np.cumprod(a)) # 从1开始元素的累积积，返回一维数组
print(np.max(a))
print(np.min(a))
print(np.argmax(a))
print(np.argmin(a))
print(np.mean(a))
print(np.median(a))
print(np.average(a))
注意：数组的众数不建议在Numpy里面计算，在Pandas里面计算更简单。
将一维数组转成Pandas的Series，然后调用mode()方法
```

### 6.1.4 Numpy的axis参数的用途

axis=0代表行，axis=1代表列  
所有的数学和统计函数都有这个参数，都可以使用  
我们想按行或按列使用时使用这个参数

```
import numpy as np
a = np.array([[1,3,6],[9,3,2],[1,4,3]])
print(f'数组: \n{a}')
print('-'*30)
print(np.sum(a,axis=0)) # 每行中的每个对应元素相加，返回一维数组
print('-'*30)
print(np.sum(a,axis=1)) # 每列中的每个元素相加，返回一维数组
```

其中思路正好是反的：axis=0 求每列的和。axis=1求每行的和。

## 6.2 数组中满足条件个数的计算

### 6.2.1 将条件逻辑作为数组操作

```
import numpy as np
a = np.array([[1,3,6],[9,3,2],[1,4,3]])
print(f'数组: \n{a}')
print('-'*30)
print(a>3)
print('-'*30)
print(np.where(a>3,520,1314))
```

### 6.2.2 布尔值数组方法 any和all

```
import numpy as np
a = np.array([[1,3,6],[9,3,2],[1,4,3]])
print(f'数组:\n{a}')
print('-'*30)
print((a>3).sum()) # 数组中大于3的数有多少个
```

对于布尔值数组，有两个常用方法any和all。

any: 检查数组中是否至少有一个True

all: 检查是否每个值都是True

```
import numpy as np
a = np.array([False,False,True,False])
print(a.any())
print(a.all())
```

## 6.2.3 按值大小排序

```
ndarray.sort(axis=-1, kind='quicksort', order=None)
或者: ndarray.sort(axis=-1, kind='quicksort', order=None)
```

参数	描述
axis	排序沿数组的（轴）方向，0表示按行，1表示按列，None表示展开来排序，默认值为-1，表示沿最后的轴排序
kind	排序的算法，提供了快排'quicksort'、混排'mergesort'、堆排'heapsort'，默认为'quicksort'
order	排序的字段名，可指定字段排序，默认为None

一维数组:

```
import numpy as np
a = np.array([3,6,7,9,2,1,8,5,4])
a.sort()
print(a)
```

二维数组:

```
import numpy as np
a = np.array([[0,12,48],[4,18,14],[7,1,99]])
print(f'数组: \n{a}')
print('-'*30)
print(np.sort(a)) # 默认按最后的轴排序，就是（行，列）（0，1）
print('-'*30)
print(np.sort(a,axis=0)) # 按行排序
```

## 6.2.4 从大到小的索引 argsort

**numpy.argsort(a, axis=-1, kind='quicksort', order=None)**

对数组沿给定轴执行间接排序，并使用指定排序类型返回数据的索引数组。这个索引数组用于构造排序后的数组。

参数类似于sort()

一维数组:

```
import numpy as np
x = np.array([59, 29, 39])
a = np.argsort(x)
print(f'索引升序: {a}') # 升序
```

```
# argsort函数返回的是数组值从小到大的索引值,[3, 1, 2]从小到大为[1, 2, 3],期对应的索引为[1, 2, 0]
print(f'数组升序: {x[a]}') # 以排序后的顺序重构原数组
b = np.argsort(-x) # 降序
print(f'索引降序: {b}')
print(f'数组升序: {x[b]}')

二维数组:
import numpy as np
x = np.array([[0, 12, 48], [4, 18, 14], [7, 1, 99]])
a1 = np.argsort(x)
print(f'索引排序: \n{a1}')
print('-'*30)
# 以排序后的顺序重构原数组, 注意与一维数组的形式不一样
print(np.array([np.take(x[i], x[i].argsort()) for i in range(3)]))
```

### 6.2.5 唯一值与其他集合逻辑unique和in1d

- 去重复

```
import numpy as np
姓名 = np.array(['孙悟空', '猪八戒', '孙悟空', '沙和尚', '孙悟空', '唐僧'])
print(np.unique(姓名))
数组 = np.array([1, 3, 1, 3, 5, 3, 1, 3, 7, 3, 5, 6])
print(np.unique(数组))检查
```

- 检查一个数组中的值是否在另外一个数组中, 并返回一个布尔数组:

```
import numpy as np
a = np.array([6, 0, 0, 3, 2, 5, 6])
print(np.in1d(a, [2, 3, 6]))
```

## 七. Numpy常用random随机函数

函数名	说明
<code>seed([seed])</code>	设定随机种子, 这样每次生成的随机数会相同
<code>rand(d0, d1, ..., dn)</code>	返回数据在[0, 1)之间, 具有均匀分布
<code>randn(d0, d1, ..., dn)</code>	返回数据具有标准正态分布(均值0, 方差1)
<code>randint(low[, high, size, dtype])</code>	生成随机整数, 包含low, 不包含high
<code>random([size])</code>	生成[0.0, 1.0)的随机数
<code>choice(a[, size, replace, p])</code>	<b>a</b> 是一维数组, 从它里面生成随机结果
<code>shuffle(x)</code>	把一个数组 <b>x</b> 进行随机排列
<code>permutation(x)</code>	把一个数组 <b>x</b> 进行随机排列, 或者数字的全排列
<code>normal([loc, scale, size])</code>	按照平均值 <b>loc</b> 和方差 <b>scale</b> 生成高斯分布的数字
<code>uniform([low, high, size])</code>	在[low, high)之间生成均匀分布的数字

### 7.1 seed 向随机数生成器传递随机状态种子

只要`random.seed( * )` `seed`里面的值一样，那随机出来的结果就一样。所以说，`seed`的作用是让随机结果可重现。也就是说当我们设置相同的`seed`，每次生成的随机数相同。如果不设置`seed`，则每次会生成不同的随机数。使用同一个种子，每次生成的随机数序列都是相同的。

```
import random
random.seed(10)
print(random.random()) # random.random()用来随机生成一个0到1之间的浮点数，包括零。
print(random.random())
print(random.random()) # 这里没有设置种子，随机数就不一样了
注意：这里不一定就写10，你写几都行，只要写上一个整数，效果都是一样的，写0都行，但是不能为空，为空就相当于没有用seed
seed只限在这一台电脑上，如果换台电脑值就变了
```

## 7.2 rand 返回[0,1]之间

```
import numpy as np
一维 = np.random.rand(3)
print(一维)
print('-'*30)
二维 = np.random.rand(2,3)
print(二维)
print('-'*30)
三维 = np.random.rand(2,3,4)
print(三维)
```

## 7.3 randint 随机整数

```
import numpy as np
a = np.random.randint(3)
print(f'随机0至3之间的整数是: {a}')
```

```
b = np.random.randint(1,10)
print(f'随机1至10之间的整数是: {b}')
```

```
c = np.random.randint(1,10,size=(5,))
print(f'随机1至10之间取5个元素组成一维数组{c}')
```

```
d = np.random.randint(1,20,size=(3,4))
print(f'随机1至20之间取12个元素组成二维数组: \n{d}')
```

```
e = np.random.randint(1,20,size=(2,3,4))
print(f'随机1至20之间取24个元素组成三维数组: \n{e}')
```

## 7.4 random 生成0.0至1.0的随机数

```
import numpy as np
一维 = np.random.random(3)
print(f'生成3个0.0至1.0的随机数:\n{一维}')
```

```
二维 = np.random.random(size=(2,3))
print(f'生成2行3列共6个数的0.0至1.0的随机数:\n{二维}')
```

```
三维 = np.random.random(size=(3,2,3))
print(f'生成三块2行3列，每块6个数的0.0至1.0的随机数:\n{三维}')
```

## 7.5 choice 从一维数组中生成随机数

```

import numpy as np
# 第一参数是一个1维数组，如果只有一个数字那就看成range(5)
# 第二参数是维度和元素个数，一个数字是1维，数字是几就是几个元素
a = np.random.choice(5,3)
print(f'从range(5)中拿随机数，生成只有3个元素的一维数组是：{a}')

import numpy as np
b = np.random.choice(5,(2,3))
print(f'从range(5)中拿随机数，生成2行3列的数组是：\n{b}')

import numpy as np
d = np.random.choice([1,2,9,4,8,6,7,5],(2,3))
print(f'从[1,2,9,4,8,6,7,5]数组中拿随机数，生成2行3列的数组是：\n{d}')

```

## 7.6 shuffle(数组)把一个数进行随机排列

```

import numpy as np
一维数组 = np.arange(10)
print(f'没有随机排列前的一维数组{一维数组}')
np.random.shuffle(一维数组)
print(f'随机排列后的一维数组{一维数组}')

import numpy as np
二维数组 = np.arange(20).reshape(4,5)
print(f'没有随机排列前的二维数组\n{二维数组}\n')
np.random.shuffle(二维数组)
print(f'随机排列后的二维数组\n{二维数组}')

import numpy as np
三维数组 = np.arange(12).reshape(2,2,3)
print(f'没有随机排列前的三维数组\n{三维数组}\n')
np.random.shuffle(三维数组)
print(f'随机排列后的三维数组\n{三维数组}')

```

## 7.7 permutation(数组) 把一个数组随机排列或者数字全排列

```

import numpy as np
# 与上面讲的np.random.shuffle(一维数组)效果一样，就是把一维数组重新排序了
排列 = np.random.permutation(10) # 这里的10就看成是range(10)
print(排列)

import numpy as np
二维数组 = np.arange(9).reshape((3,3))
print(f'没有随机排列前的二维数组是\n{二维数组}\n')
排序后 = np.random.permutation(二维数组)
print(f'随机排列后的二维数组是\n{排序后}\n')
print(f'看一下原来的二维数组变了吗？\n{二维数组}')

```