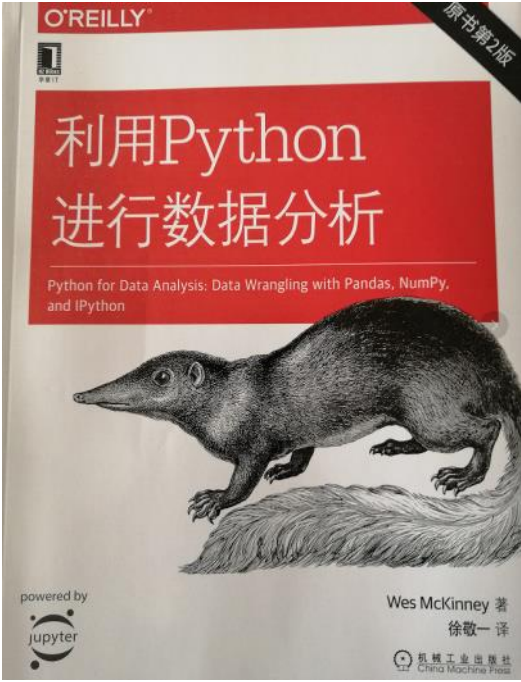
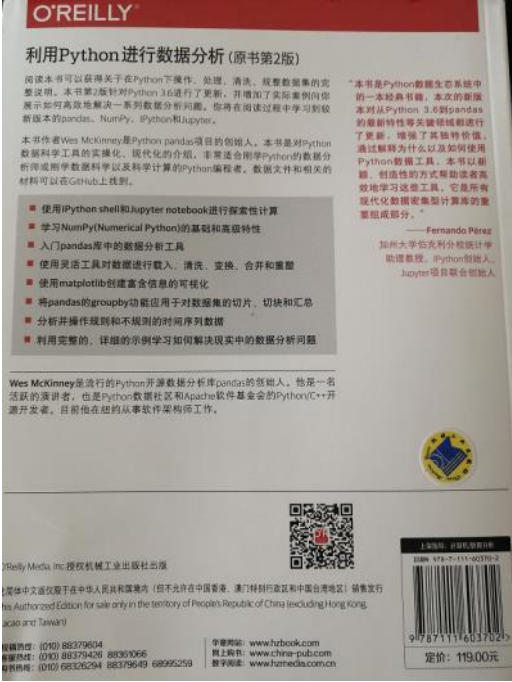


参考书籍

2020年4月4日 13:26



正面



反面



我的订单

Numpy介绍及安装

2020年4月3日 15:49

同样的数值计算，使用Numpy比直接编写Python实现 **代码更简洁、性能更高效**。它是目前Python数值计算中最为重要的基础包。

Numpy帮我们处理数值型数据的
在Pandas和机器学习中都会用到Numpy

安装：

1、找到pip3.exe所在的文件夹，复制路径

我的路径是：C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\Scripts

2、按Win+R,输入CMD确定

3、进入后，先输入cd 路径 回车,如图1

4、输入 pip3 install numpy 回车,如图1

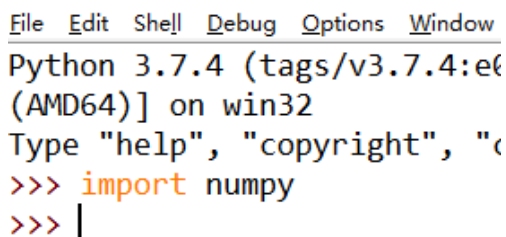
5、在Python编译器中，导入模块不报错就证明安装成功了。如图2



```
Microsoft Windows [版本 10.0.18362.720]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\孙艺航>cd C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\Scripts
C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\Scripts>pip3 install numpy
```

图1



```
File Edit Shell Debug Options Window
Python 3.7.4 (tags/v3.7.4:ec07720, Dec 4 2019) on win32
Type "help", "copyright", "credits()" or "quit()" for more
>>> import numpy
>>> |
```

图2

序章：数组和矩阵的区别

2020年4月3日 17:46

- **matrix**是**array**的分支，**matrix**和**array**在很多时候都是通用的，用哪一个都一样；
- 如果两个可以通用，那就选择**array**，因为**array**更灵活，速度更快，很多人把二维的**array**也翻译成矩阵。
- 相同点：索引方式相同，都可以使用**a[i][j]**，**a[i,j]**进行索引
- **matrix**（矩阵）：具有相对简单的运算符号，比如两个**matrix**相乘（求内积），就是用符号*
- **array**（数组）：两个一维数组相乘用*，对应元素相乘

```
1 import numpy as np
2 a=np.array([1,2,3,4])
3 b=np.array([2,3,4,5])
4 print(a*b)
5 print(sum(a*b))
6
7 #结果
8 [ 2  6 12 20]
9 40
```

- 多维数组也可以看成矩阵，如果进行矩阵相乘（求内积），则需要用到**np.dot()**，如**np.dot(arr.T,arr)**

```
import numpy as np

"""rand函数的作用就是从标准正态分布中返回一个或多个样本值，
   这里返回2行3列共6个数的数组。
   标准正态分布俗称高斯分布，均值为0，方差为1的正态分布。"""

arr=np.random.rand(2,3)
b=np.dot(arr.T,arr) #把数组当成矩阵，对应元素相乘求内积
#将数组转化成矩阵
arr_Matrix=mat(arr) #等同于arr_Matrix=matrix(arr)，mat是matrix的缩写
c=arr_Matrix.T * arr_Matrix #矩阵直接相乘

#将结果输出
print("arr:\n{}\ntype(arr):{}".format(arr,type(arr)))
print("数组（矩阵）内积：\n",b)

print("arr_Matrix:\n{}\ntype(arr_Matrix):{}".format(arr_Matrix,type(arr_Matrix)))
print("矩阵乘积：\n",c)

#比较两者乘积是否相等
print("数组矩阵内积=矩阵乘积：\n",b==c)
```

#运行结果如下:

arr:

```
[[0.93383594 0.70401786 0.95323492]
 [0.7970047  0.21436019 0.24329003]]
```

type(arr):<class 'numpy.ndarray'>

数组（矩阵）内积:

```
[[1.50726605 0.82828326 1.08406832]
 [0.82828326 0.54159144 0.72324611]
 [1.08406832 0.72324611 0.96784685]]
```

arr_Matrix:

- ```
[[0.93383594 0.70401786 0.95323492]
 [0.7970047 0.21436019 0.24329003]]
```

type(arr\_Matrix):<class 'numpy.matrixlib.defmatrix.matrix'>

矩阵乘积:

```
[[1.50726605 0.82828326 1.08406832]
 [0.82828326 0.54159144 0.72324611]
 [1.08406832 0.72324611 0.96784685]]
```

数组矩阵内积=矩阵乘积:

```
[[True True True]
 [True True True]
 [True True True]]
```

## 一、体验Numpy 多维数组对象

2020年4月3日 16:01

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 数组相加(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(数组相加(3))
```

使用Numpy计算上面的题目：

```
import numpy as np
def 数组相加(n):
 a = np.arange(1,n+1) ** 3
 b = np.arange(1,n+1) ** 2
 return a+b
print(数组相加(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。



## 1.1 创建数组的方法

2020年4月3日 16:42

```
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

## 1.1.1 arange 创建数字序列

2020年4月3日 21:03

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])`

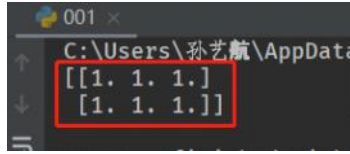
## 1.1.2 使用ones创建全是1的数组

2020年4月3日 21:07

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

`a=np.ones(3)` # 返回 `array([1. 1. 1.])`

`b=np.ones((2,3))`



`b=np.ones((5,), dtype=np.int)` # 返回 `[1 1 1 1 1]`

### 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。

**order:** { 'C' , 'F' }, 可选规定返回数组元素在内存的存储顺序: C (C语言) -rowmajor; F (Fortran) column-major。



### 1.1.3 ones\_like创建形状相同的数组

2020年4月3日 21:24

```
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

**order顺序：**{'C', 'F', 'A'或'K'}，可选,覆盖结果的内存布局。

**subok：**bool，可选。True：使用a的内部数据类型，False：使用a数组的数据类型，默认为True

**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.])
```

## 1.1.4 zeros 创建全是0的数组

2020年4月3日 21:38

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

`np.zeros(10)` 返回: `[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]`

`np.zeros((2,4))` 返回

```
C:\Users\孙艺航\AppData\Local\Programs\Python\Python38-64\Python38\python.exe
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```



### 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。

**order:** { 'C' , 'F' }, 可选规定返回数组元素在内存的存储顺序: C (C语言) -rowmajor; F (Fortran) column-major。

## 1.1.5 zeros\_like创建形状相同的数组

2020年4月3日 21:41

**np.zeros(a,dtype=None)**

**案例1:以下数组是x**

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

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

```
array([[0, 0, 0],
 [0, 0, 0]])
```

**案例2:以下数组是y**

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

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

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

## 1.1.6 full创建指定值的数组

2020年4月3日 21:47

`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。

**order**：{ 'C' , 'F' }, 可选规定返回数组元素在内存的存储顺序：C（C语言）-rowmajor；F（Fortran）column-major。

例：

`np.full(3,520)` 返回 `[520 520 520]`

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

```
C:\Users\孙艺航\AppData
[[520 520 520 520]
 [520 520 520 520]]
```

## 1.1.7 full\_like创建开关相同的指定值数组

2020年4月3日 21:47

**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.])
```

## 1.1.8 使用random模块生成随机数组

2020年4月3日 21:38

```
import random
np.random.randn(d0,d1,.....,dn)
```

`np.random.random(3)`



|        |
|--------|
| 0.5967 |
| 0.0606 |
| 0.2223 |

传1个数就是一维，2个数就是二维，n个数就是n维

例如：

```
a = np.random.randn() # 一个随机数
b = np.random.randn(3) # 3个数
c = np.random.randn(3,2) # 3行2列
d = np.random.randn(3,2,4) # 3块，每块是2行4列
```

需要四舍五入：

```
np.round(a,2) # 变量a，保留两位小数点
```

## 1.2 给数据指定数据类型

2020年4月3日 16:57

```
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))
```

### 1.3 多维数组

2020年4月3日 17:21

```
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：

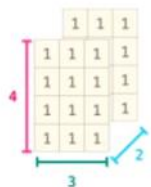


这样就很容易理解括号里 (3,2) 的含义。

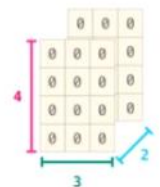
Numpy 不仅可以处理上述的一维数组和二维矩阵（二维数组我们习惯叫它为矩阵），还可以处理任意 N 维的数组，方法也大同小异。



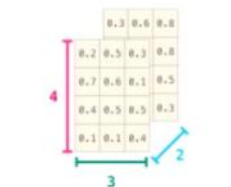
`np.ones((4,3,2))`



`np.zeros((4,3,2))`



`np.random.random((4,3,2))`





## 1.3.1 reshape不改值修改形状

2020年4月3日 22:46

```
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**

**注意区别：**

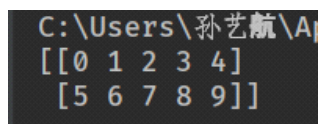
**shape：返回数组的维度**

**reshape:不改变数组的值，修改形状**

## 1.3.2 数组计算

2020年4月3日 23:08

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



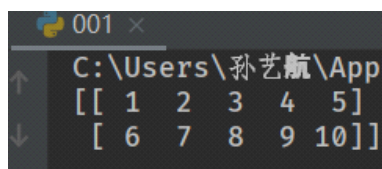
```
C:\Users\孙艺航\AppData\Local\Temp\ipykernel_1001\1001.py:1:
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

**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

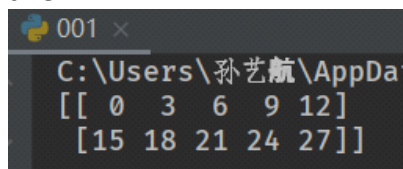
**a.shape** # 返回维度 (2, 5)

**a+1**



```
C:\Users\孙艺航\AppData\Local\Temp\ipykernel_1001\1001.py:1:
[[1 2 3 4 5]
 [6 7 8 9 10]]
```

**a\*3**



```
C:\Users\孙艺航\AppData\Local\Temp\ipykernel_1001\1001.py:1:
[[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)
```

**a+1** 数组中每个数都+1

**a\*2** 数组中每个数都\*2

**a/2** 数组中每个数都除2

注：如果分母为0，0/0返回nan，其它数/0返回inf

nan(NAN,Nan):not a number表示不是一个数字

inf(-inf,inf):infinity,inf表示正无穷, -inf表示负无穷

a+b 对应位置的数字进行计算 (加减乘除)

```
import numpy as np
a = np.arange(6)
b = np.arange(24).reshape(4,6)
print(a)
print(b)
print(b-a)
```

总结:

(1) 形状一样的数组按对应位置进行计算。

(2) 一维和 multidimensional 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

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 4 | 5 | 7 | 2 |
| 2 | 5 | 3 | 5 | 4 |

b

|   |   |   |   |   |
|---|---|---|---|---|
| 2 | 1 | 3 | 2 | 2 |
|---|---|---|---|---|

a-b

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 3 | 2 | 5 | 0 |
| 0 | 4 | 0 | 3 | 2 |

a\*b

|   |   |    |    |   |
|---|---|----|----|---|
| 6 | 4 | 15 | 14 | 0 |
| 4 | 5 | 9  | 10 | 8 |

a

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 5 | 7 | 6 | 4 |
| 2 | 3 | 5 | 3 | 8 |
| 5 | 6 | 7 | 3 | 9 |
| 4 | 5 | 2 | 1 | 3 |
| 6 | 7 | 6 | 9 | 4 |

b

|   |
|---|
| 3 |
| 2 |
| 3 |
| 1 |
| 3 |

a+b

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 2 | 4 | 3 | 1 |
| 0 | 1 | 3 | 1 | 6 |
| 2 | 3 | 4 | 0 | 6 |
| 3 | 4 | 1 | 0 | 2 |
| 3 | 4 | 3 | 6 | 1 |

a+b

|    |    |    |    |    |
|----|----|----|----|----|
| 9  | 15 | 21 | 18 | 12 |
| 4  | 6  | 10 | 6  | 16 |
| 15 | 18 | 21 | 9  | 27 |
| 4  | 5  | 2  | 1  | 3  |
| 18 | 21 | 18 | 27 | 12 |

### 1.3.3 广播规则

2020年4月4日 9:52

|     |   |   |    |
|-----|---|---|----|
| x   | 2 | 2 | 3  |
|     | 1 | 2 | 3  |
| y   | 1 | 1 | 3  |
|     | 2 | 2 | 4  |
| x*y | 2 | 2 | 9  |
|     | 2 | 4 | 12 |

相同形状：数组计算

|     |   |   |   |
|-----|---|---|---|
| x   | 2 | 2 | 3 |
|     | 1 | 2 | 3 |
| y   | 1 | 1 | 3 |
|     |   |   |   |
| x*y | 2 | 2 | 9 |
|     | 1 | 2 | 9 |

不同形状：数组计算

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

|        |   |       |   |        |
|--------|---|-------|---|--------|
| (4, 3) |   | (3, ) |   | (4, 3) |
| 0 0 0  | + | 1 2 3 | = | 1 2 3  |
| 1 1 1  |   | 1 2 3 |   | 2 3 4  |
| 2 2 2  |   | 1 2 3 |   | 3 4 5  |
| 3 3 3  |   | 1 2 3 |   | 4 5 6  |

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：

|           |   |        |   |           |
|-----------|---|--------|---|-----------|
| (3, 4, 2) |   | (4, 2) |   | (3, 4, 2) |
| 0 1       | + | 0 1    | = | 0 2       |
| 2 3       |   | 2 3    |   | 4 6       |
| 4 5       |   | 4 5    |   | 8 10      |
| 6 7       |   | 6 7    |   | 12 14     |

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。

思考:

- ☒ (3, 2, 3) 和 (2, 3)
- ☒ (3, 2, 3) 和 (3, )
- ☒ (3, 2, 2) 和 (2, 1)
- ☐ (3, 2, 3) 和 (2, 4)

2020年4月4日 10:42

```
a = np.arange(10)
```

## # 一维数组使用小写

|   | data |
|---|------|
| 0 | 1    |
| 1 | 2    |
| 2 | 3    |

```
data[0]
```

```
data[1]
```

|   |
|---|
| 1 |
| 2 |

|   |
|---|
| 2 |
| 3 |

```
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])
```

语法: **序列**[开始位置下标: 结束位置下标: 步长]

步长是选取间隔，正负整数均可，默认为1

不包结束位置下标对应的数据，正负整数均可

## 矩阵索引

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

## # 二维数组索引用行列坐标，类似于我们Excel中的offset

用最快的方式学会excel【Excel教程持续更新ing...】



|   | data |   |
|---|------|---|
|   | 0    | 1 |
| 0 | 1    | 2 |
| 1 | 3    | 4 |
| 2 | 5    | 6 |

|   | 0 | 1 |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |
| 2 | 5 | 6 |

```
data[1:3]
```

| 0 | 1 |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

data[0:2,0]

|   | 0 | 1 |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |
| 2 | 5 | 6 |

<https://www.bilibili.com/video/BV1Z7411c7U6>

## 第19集 offset函数就是个游戏机

**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

### 三、切片修改 (注意: 会修改原来的数组)

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**

## 1.5 布尔索引

2020年4月4日 11:50

### 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

```
数组[数组<=5] = 0 # 小于5的重新赋值为0
```

```
数组[数组>5] = 1 # 大于5的重新赋值为1
```

```
print(数组)
```

实例2: 进行自增量的操作, 给大于5的加上520

```
数组[数组>5] += 520
```

```
print(数组)
```

### 二维数组

```
import numpy as np
```

```
数组 = np.arange(1,21).reshape(4,5)
```

```
print(数组)
```

```
筛选 = 数组>10
```

```
print(筛选) # 返回一个布尔数组, 即有行又有列
```

```
print(数组[筛选]) # 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)
```

```
C:\Users\孙艺航\AppData\Lo
[[1 2 3 4 5]
 [6 7 8 9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]

[4 9 14 19]

[[1 2 3 4 5]
 [520 520 520 520 520]
 [520 520 520 520 520]
 [520 520 520 520 520]]
```

一、所有的行第3列改成520

```
import numpy as np
```

```
数组 = np.arange(1,21).reshape(4,5)
```

```
数组[:,3] = 520
```

```
print(数组)
```

二、所有行第3列, 大于5的改成520

```
import numpy as np
```

```
数组 = np.arange(1,21).reshape(4,5)
```

```
筛选 = 数组[:,3]>5 # 所有行第3列, 大于5的
```

```
数组[:,3][筛选] = 520
```

```
print(数组)
```

```
import numpy as np
```

```
数组 = np.array([[10,20,30],[50,40,10],[10,1,10]])
```

```
print(数组)
```

```
筛选 = 数组>25
```

```
print(筛选)
```

```
print(数组[筛选])
```

条件组合: 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])
```

```
C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\python.exe
[0 1 2 3 4 5 6 7 8 9]

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

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



## 1.6 神奇索引

2020年4月4日 11:50

神奇索引：使用整数数组进行数据索引。

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 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(36).reshape(9,4)
```

```
print(数组)
```

```
print(""*30)
```

```
print(数组[:,[1,2]]) # 取所有行的, 第1列和第2列
```

```
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])
```

```
C:\Users\孙艺航\AppData\Local\Prog
[0 1 2 3 4 5 6 7 8 9]

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

实例：获取数组中最大的前N个数字

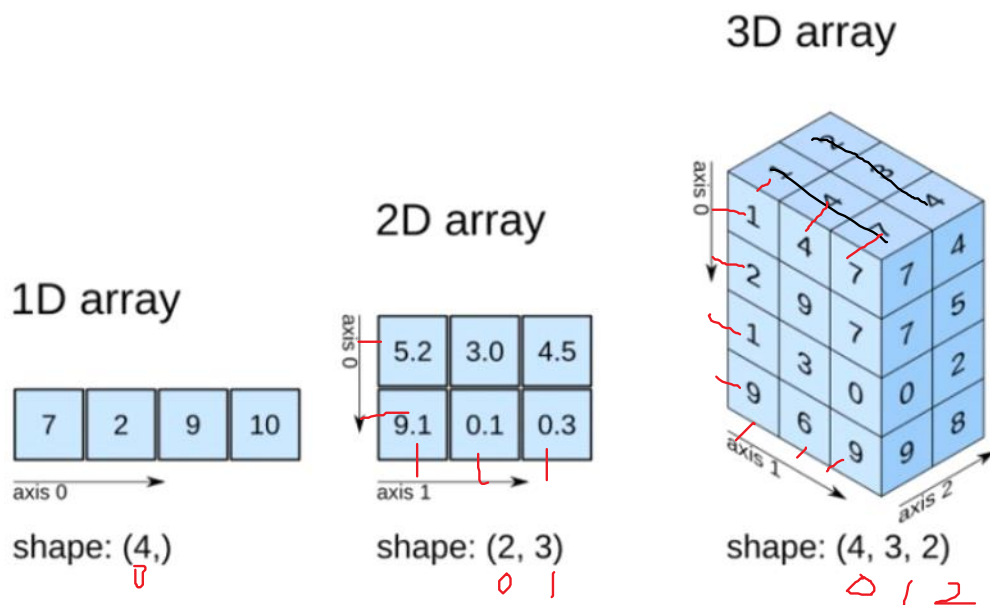
# 获取数组中最大的前N个数字

```
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}')
```

## 二、Numpy的轴

2020年4月4日 14:25



`import numpy as np`

```
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为高维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

2020年4月4日 20:31

```
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。

**首先看2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])

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

## 2.2 传入轴编号怎么理解

2020年4月4日 20:54

```
import numpy as np
```

```
数组=np.arange(16).reshape((2, 2, 4))
```

```
print(数组)
```

```
print(数组.shape)
```

就是3维数组的轴编号

数组的维度： (2, 2, 4)    元组索引（下标）： [0,1,2]

我们转换它：

3维数组的1维（2轴）上是4个一维数组，每个1维数组都有一个由0,1两个轴编号组成的索引 [0,0], [0,1], [1,0], [1,1]，transpose方法传入的参数是轴编号 (1, 0, 2) 在就是把元组的索引顺序改变成 [1,0,2] 也就是把数组 [0,1] 的一维数组变成数组 [1,0]

|    |   | 1轴                 |                      |
|----|---|--------------------|----------------------|
|    |   | 0                  | 1                    |
| 0轴 | 0 | 0,1,2,3<br>(0,0)   | 4,5,6,7<br>(0,1)     |
|    | 1 | (1,0)<br>8,9,10,11 | (1,1)<br>12,13,14,15 |

输出

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

### 三、numpy数组转置换轴

2020年4月5日 9:09

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 |

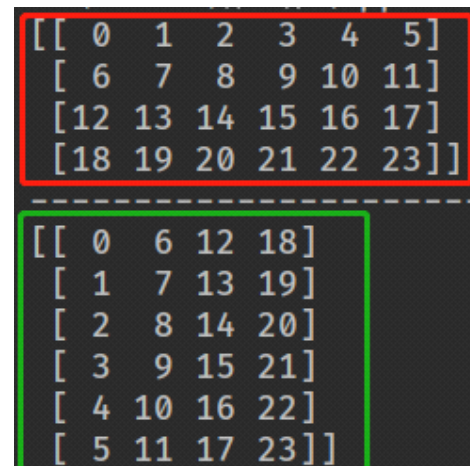
data.reshape(3,2)

|   |   |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

## 3.1 transpose方法 【行列转置】

2020年4月5日 9:21

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



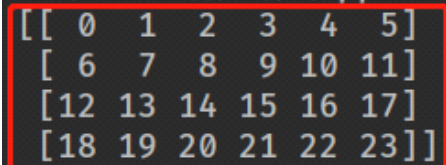
```
[[0 1 2 3 4 5]
 [6 7 8 9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]

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

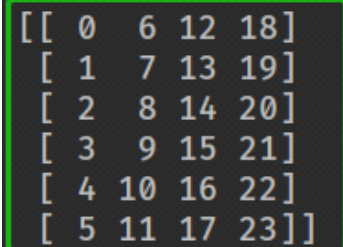
## 3.2 swapaxes方法 【轴转置】

2020年4月5日 9:27

```
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]]
```



## 四、Numpy常用random随机函数

2020年4月5日 9:37

| 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x进行随机排列              |
| <code>permutation(x)</code>                    | 把一个数组x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loc和方差scale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   |

## 4.1 seed 向随机数生成器传递随机状态种子

2020年4月5日 9:55

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

## 4.2 rand 返回[0,1]之间，从均匀分布中抽取样本

2020年4月5日 10:10

```
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)
```

```
[0.91393273 0.64177439 0.12127377] 一维

[[0.42006266 0.30584327 0.82347489] 二维
 [0.9332625 0.46752797 0.74903966]]

[[[0.62717205 0.74551462 0.74248745 0.14876918]
 [0.97212127 0.00945164 0.33643652 0.61088034]
 [0.31781924 0.5675992 0.14537249 0.11590312]] 三维
 [[0.26506813 0.57189897 0.71386175 0.91001165]
 [0.74445198 0.27596353 0.23805552 0.80191919]
 [0.24824561 0.41018213 0.8119142 0.65113587]]]
```

我们数据分析的三部曲：Numpy、Pandas、matplotlib，后期我们在使用matplotlib画图时会用到均匀分布

```
Microsoft Windows [版本 10.0.18362.720]
(c) 2019 Microsoft Corporation. 保留所有权利。
C:\Users\孙艺航>cd C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\Scripts
C:\Users\孙艺航\AppData\Local\Programs\Python\Python37\Scripts>pip3 install matplotlib
```

```
import numpy as np
import matplotlib.pyplot as plt
绘制正弦曲线
x轴 = np.linspace(-10,10,100) # 在[-10,10]闭区间或半闭区间中，数量为100
y轴 = np.sin(x轴) # sin正弦、cos余弦
plt.plot(x轴,y轴)
plt.show()

#加入噪声后
import numpy as np
import matplotlib.pyplot as plt
绘制正弦曲线
x轴 = np.linspace(-10,10,100) # 在[-10,10]闭区间中，数量为100
y轴 = np.sin(x轴) + np.random.rand(len(x轴)) # 生成均匀分布，len(x轴)就是维度，相加就是定义元素的相加
plt.plot(x轴,y轴)
plt.show()
```

### 4.3 randn 返回标准正态分布随机数（浮点数）平均数0，方差1 【了解】

2020年4月5日 10:25

randn生成一个从标准正态分布中得到的随机标量，标准正态分布即 $N(0, 1)$ 。

randn (n)和randn(a1,a2,...)的用法和rand相同

```
import numpy as np
```

```
一维 = np.random.randn(3)
```

```
print(一维)
```

```
print('-'*30)
```

```
二维 = np.random.randn(2,3)
```

```
print(二维)
```

```
print('-'*30)
```

```
三维 = np.random.randn(2,3,4)
```

```
print(三维)
```

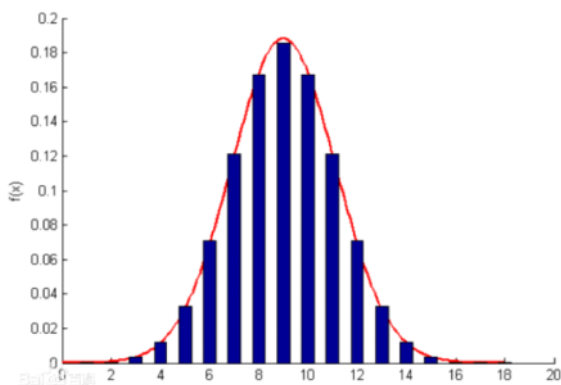
```
print('-'*30)
```

```
[0.76388878 -0.93828205 0.12701948] 一维

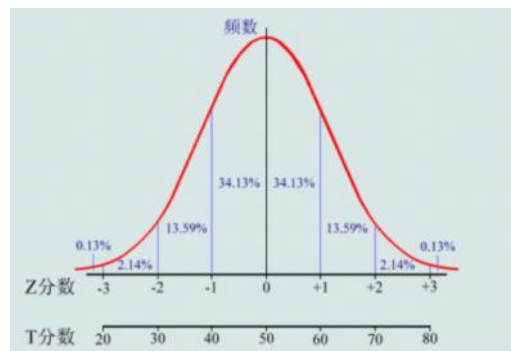
[[-1.43339927 -0.35867966 -0.40617194] 二维
 [-0.43001871 -0.96275831 -0.39123271]]

[[0.98045942 0.35491842 -0.54718563 -0.40427562] 三维
 [-1.02663406 -0.12362132 -0.34739243 -0.81949418]
 [0.20234654 1.50702542 0.30846399 0.42042206]]

[[-0.14340575 0.07440955 -2.00803861 0.94657717]
 [0.82196822 1.28403362 1.13573602 1.04508304]
 [1.27178304 -1.71929955 2.79839521 -0.68944059]]
```



正态分布



标准正态分布

## 4.4 randint 随机整数

2020年4月5日 11:03

```
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}')
```

## 4.5 random 生成0.0至1.0的随机数

2020年4月5日 11:25

```
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}')
```

## 4.6 choice 从一维数组中生成随机数

2020年4月5日 11:42

```
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
```

```
c = np.random.choice([1,2,9,4,8,6,7,5],3)
```

```
print(f'从[1,2,9,4,8,6,7,5]数组中拿随机数，3个元素：{c}')
```

```
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}')
```

## 4.7 shuffle(数组)把一个数进行随机排列

2020年4月5日 11:54

```
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}')
```

没有随机排列前的一维数组[0 1 2 3 4 5 6 7 8 9]  
随机排列后的一维数组[3 0 6 9 1 2 8 5 7 4]

```
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}\n')
```

没有随机排列前的二维数组

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

随机排列后的二维数组

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

☑ 注意：多维数组随机排列只按行随机，列是不变的

```
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}\n')
```

没有随机排列前的三维数组

```
[[[0 1 2]
 [3 4 5]]
```

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

随机排列后的三维数组

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

```
 [[0 1 2]
 [3 4 5]]]
```



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

2020年4月5日 12:02

```
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}')
```

## 4.9 normal 生成正态分布数字

2020年4月5日 12:24

正态分布，又叫常态分布，又叫高斯分布

`normal [平均值, 方差, size]`

```
import numpy as np
```

```
数组 = np.random.normal(1,10,10) # 平均值1, 方差10, 10个数
```

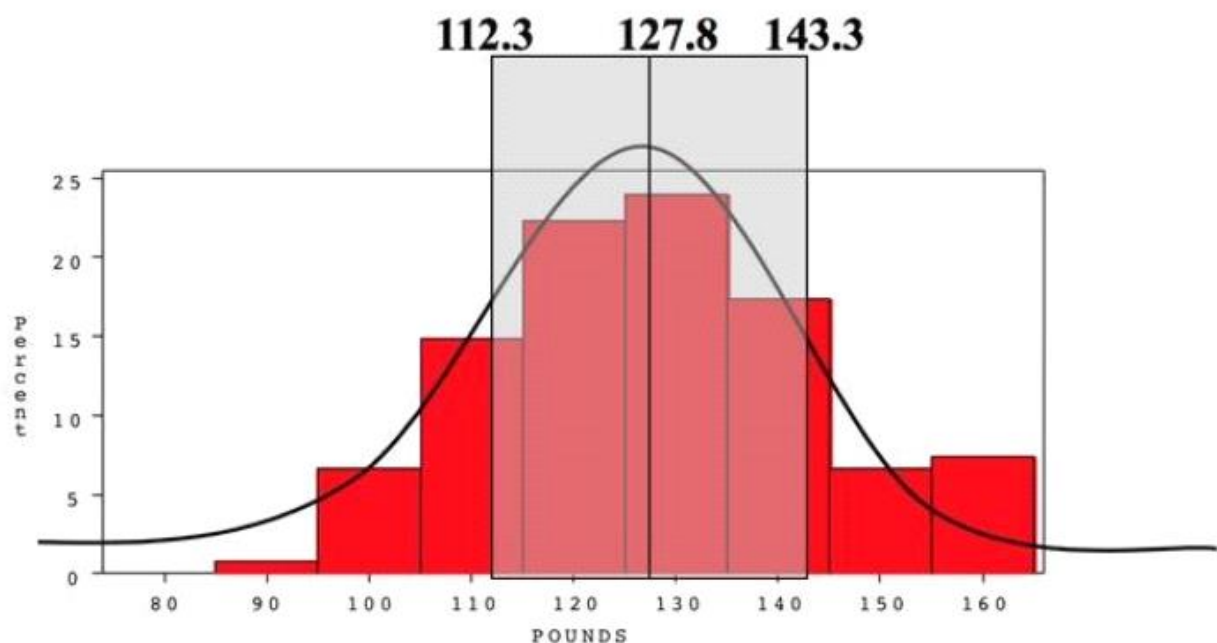
```
print(数组)
```

举一个例子：

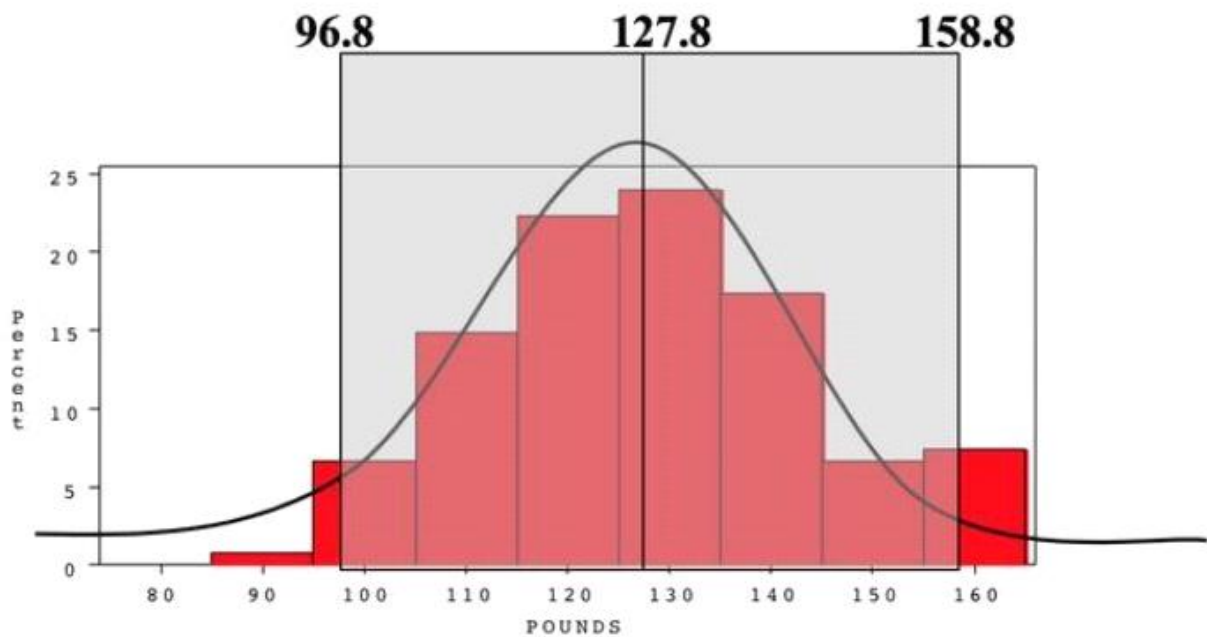
检查一些示例数据：

女性体重的平均值= 127.8

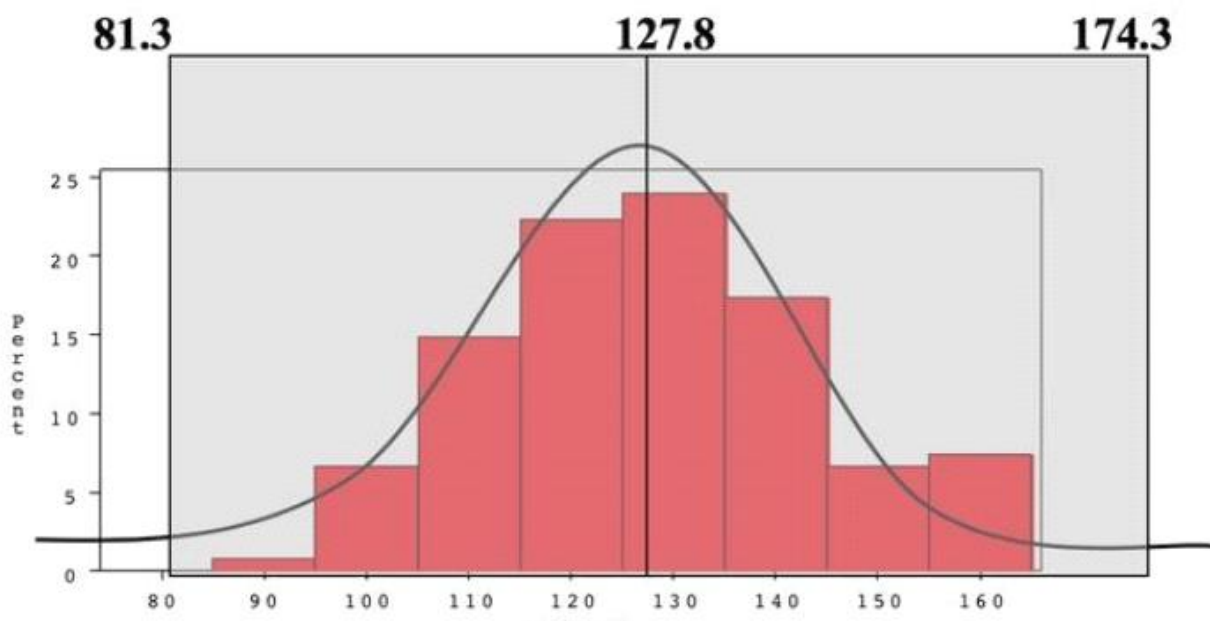
标准偏差 (SD) = 15.5



一个标准差的范围



两个标准差的范围



### 高斯分布重要量的性质

1. 密度函数关于平均值对称
2. 平均值是它的众数 (statistical mode) 以及中位数 (median)
3. 函数曲线下68.268949%的面积在平均值左右的一个标准差范围内
4. 95.449974%的面积在平均值左右两个标准差 $2\sigma$ 的范围内
5. 99.730020%的面积在平均值左右三个标准差 $3\sigma$ 的范围

**其中第3-5条称为68-95-99.7法则**

## 4.10 uniform 均匀分布

2020年4月5日 12:37

```
import numpy as np
数组 = np.random.uniform(1,10,10)
print(f'在1到10之间生成10个随机数: \n{数组}')
```

```
import numpy as np
数组 = np.random.uniform(1,10,(2,3))
print(f'在1到10之间生成2行3列共计6个随机数: \n{数组}')
```

## 六、通用函数：快速的逐元素数组函数

2020年4月5日 13:05

通用函数也可以称为 `ufunc`，是一种在 `ndarray` 数据中进行逐元素操作的函数。某些简单函数接受了一个或者多个标量数值，并产生一个或多个标量结果，而通用函数就是对这些简单函数的向量化封装。

有很多 `ufunc` 是简单的逐元素转换，比如 `sqrt` 和 `exp` 函数：就是一元通用函数

```
import numpy as np
数组 = np.arange(10)
print(数组)
print(np.sqrt(数组)) # 返回正的平方根
print(np.exp(数组)) # 计算每个元素的自然指数值e的x次方
```

介绍一下二元通用函数：比如 `add` 和 `maximum` 则会接受两个数组并返回一个数组结尾结果，所以叫做二元通用函数。

```
import numpy as np
x = np.random.randn(8)
y = np.random.randn(8)
print(x)
print('-----')
print(y)
print('-----')
print(np.maximum(x,y)) # 对位比较大小，取大的，生成新的数组返回，逐个元素地将 x和 y 中元素的最大值计算出来
```

| 一元ufunc              | 描述                                                          | 调用方式                                                          |
|----------------------|-------------------------------------------------------------|---------------------------------------------------------------|
| abs, fabs            | 计算整数、浮点数或者复数的绝对值，对于非复数，可以使用更快的fabs                          | np.abs(arr)<br>np.fabs(arr)                                   |
| sqrt                 | 计算各个元素的平方根，相当于 $arr ** 0.5$ ，要求arr的每个元素必须是非负数               | np.sqrt(arr)                                                  |
| square               | 计算各个元素的平方，相当于 $arr ** 2$                                    | np.square(arr)                                                |
| exp                  | 计算各个元素的指数e的x次方                                              | np.exp(arr)                                                   |
| log、log10、log2、log1p | 分别计算自然对数、底数为10的log、底数为2的log以及 $\log(1+x)$ ；要求arr中的每个元素必须为正数 | np.log(arr)<br>np.log10(arr)<br>np.log2(arr)<br>np.log1p(arr) |
| sign                 | 计算各个元素的正负号：1 正数，0：零，-1：负数                                   | np.sign(arr)                                                  |
| ceil                 | 计算各个元素的ceiling值，即大于等于该值的最小整数                                | np.ceil(arr)                                                  |
| floor                | 计算各个元素的floor值，即小于等于该值的最大整数                                  | np.floor(arr)                                                 |
| rint                 | 将各个元素值四舍五入到最接近的整数，保留dtype的类型                                | np.rint(arr)                                                  |

| 一元ufunc                                      | 描述                                            | 调用方式                                               |
|----------------------------------------------|-----------------------------------------------|----------------------------------------------------|
| modf                                         | 将数组中元素的小数位和整数位以两部分独立数组的形式返回                   | np.modf(arr)                                       |
| isnan                                        | 返回一个表示“那些值是NaN(不是一个数字)”的布尔类型数组                | np.isnan(arr)                                      |
| isfinite、isinf                               | 分别一个表示“那些元素是有穷的(非inf、非NaN)”或者“那些元素是无穷的”的布尔型数组 | np.isfinite(arr)<br>np.isinf(arr)                  |
| cos、cosh、sin、sinh、tan、tanh                   | 普通以及双曲型三角函数                                   | np.cos(arr)<br>np.sin(arr)<br>np.tan(arr)          |
| arccos、arccosh、arcsin、arcsinh、arctan、arctanh | 反三角函数                                         | np.arccos(arr)<br>np.arcsin(arr)<br>np.arctan(arr) |

| 二元ufunc                                                           | 描述                                  | 调用方式                                                                               |
|-------------------------------------------------------------------|-------------------------------------|------------------------------------------------------------------------------------|
| mod                                                               | 元素级的求模计算（除法取余）                      | np.mod(arr1,arr2)                                                                  |
| dot                                                               | 求两个数组的点积                            | np.dot(arr1,arr2)                                                                  |
| greater、<br>greater_equal、<br>less、less_equal、<br>equal、not_equal | 执行元素级别的比较运算，最终返回一个布尔型数组             | np.greater(arr1, arr2)<br>np.less(arr1, arr2)<br>np.equal(arr1, arr2)              |
| logical_and、<br>logical_or、<br>logical_xor                        | 执行元素级别的布尔逻辑运算，相当于中缀运算符&、 、^         | np.logical_and(arr1,arr2)<br>np.logical_or(arr1,arr2)<br>np.logical_xor(arr1,arr2) |
| power                                                             | 求解对数组中的每个元素进行给定次数的指数值，类似于: arr ** 3 | np.power(arr, 3)                                                                   |

## 七、使用数组进行面向数组编程

2020年4月6日 9:19

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）



## 7.1 数学和统计方法

2020年4月6日 9:38

| 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                   |

7.1.1 平均数，加权平均数，中位数，众数

2020年4月6日 10:45

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。

|     |     |      |
|-----|-----|------|
| 马云  | 孙兴华 | 马化腾  |
| 最有钱 | 最穷  | 第二有钱 |

|    |     |     |     |       |
|----|-----|-----|-----|-------|
| 马云 | 马化腾 | 孙兴华 | 吃低保 | 吃不上低保 |
|----|-----|-----|-----|-------|

|    |     |       |     |     |       |
|----|-----|-------|-----|-----|-------|
| 马云 | 马化腾 | B站小伙伴 | 孙兴华 | 吃低保 | 吃不上低保 |
|----|-----|-------|-----|-----|-------|

- 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)

## 7.1.2 一维数组

2020年4月6日 9:46

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
```

Numpy中没有直接的方法求众数，但是可以这样实现：

```
import numpy as np
bincount () : 统计非负整数的个数，不能统计浮点数
counts = np.bincount(nums)
#返回众数
np.argmax(counts)
```

## 7.1.3 二维数组

2020年4月6日 11:03

```
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

```
import numpy as np
import pandas as pd

nums = np.random.randint(1, 10, size=20)
ser = pd.Series(nums)
print(ser.mode())
```

将二维数组转成Pandas的DataFrame,然后调用mode()方法

```
import numpy as np
import pandas as pd

nums = np.random.randint(1, 10, size=20).reshape(4, 5)
df = pd.DataFrame(nums)
print(df.mode())
```

## 7.1.4 Numpy的axis参数的用途

2020年4月6日 11:49

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。

数组对应到现实中的一种解释：

- 行：每行对应一个样本数据
- 列：每列代表样本的一个特征

数据标准化：

- 对于机器学习、神经网络来说，不同列的量纲是相同的，收敛更快。
- 有两个特征，一个是商品单价1元至50元，另一个是销售数量3千个至1万个，这两个数字不可比，所以需要都做标准化。
- 比如在Excel里，单价一个列，销售数量一个列，不同列代表不同特征，所以用axis=0做计算
- 标准化一般使用：通过均值和方差实现

数组 = (数组 - mean(数组, axis=0)) / std(数组, axis=0)

## 7.2 数组中满足条件个数的计算

2020年4月6日 12:31

## 7.2.1 将条件逻辑作为数组操作

2020年4月6日 12:15

```
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))
```



## 7.2.2 布尔值数组方法 any和all

2020年4月6日 12:25

```
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())
```

## 7.2.3 按值大小排序 sort

2020年4月6日 13:03

`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
```

拓展: 按字段名排序【有需要的自己看一下】

```
>>dt = np.dtype([('name', 'S10'),('age', int)])
>>a = np.array([("Mike",21),("Nancy",25),("Bob", 17), ("Jane",27)], dtype = dt)
>>np.sort(a, order = 'name')
array([(b'Bob', 17), (b'Jane', 27), (b'Mike', 21), (b'Nancy', 25)],
 dtype=[('name', 'S10'), ('age', '<i4')])

>>np.sort(a, order = 'age')
array([(b'Bob', 17), (b'Mike', 21), (b'Nancy', 25), (b'Jane', 27)],
 dtype=[('name', 'S10'), ('age', '<i4')])
```



## 7.2.4 从大到小的索引 argsort

2020年4月6日 13:30

`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)]))
```

## 7.2.5 根据键值的字典序进行排序 lexsort

2020年4月6日 13:44

lexsort(keys, axis=-1)

lexsort()根据键值的字典序进行排序，支持对数组按指定行或列的顺序排序，间接排序，不修改原数组，返回索引。一般对一维数组使用argsort()。默认按最后一行元素有小到大排序，返回最后一行元素排序后索引所在位置。

| 参数     | 描述                                           |
|--------|----------------------------------------------|
| 'axis' | 数组排序时的基准，axis=0，按行排列；axis=1，按列排列             |
| 'keys' | 排序的参照物包括数组或包含N维的的元组，默认值为最后一行，（如果为二维数组则指最后一列） |

```
1 >>import numpy as np
2 >>x=np.array([[0,12,48],[4,18,14],[7,1,99]])
3
4 >>np.lexsort(x)
5 array([1, 0, 2], dtype=int64) #返回索引值
6
7 >>a=np.array([1,5,1,4,3,4,4])
8 >>b=np.array([9,4,0,4,0,2,1])
9 >> ind=np.lexsort((b,a))
10 >>ind #将长度相同的a, b组合，再根据a值的大小进行排序，再考虑b值
11 array([2, 0, 4, 6, 5, 3, 1], dtype=int64)
12
13 >>list(zip(a[ind],b[ind]))
14 [(1, 0), (1, 9), (3, 0), (4, 1), (4, 2), (4, 4), (5, 4)]
15
16 >>> c=[[1,5,1,4,3,4,4],[9,4,0,4,0,2,1]]
17 >>> np.lexsort(c) # 此种情况与先b后a的情况一致
18 array([2, 4, 6, 5, 3, 1, 0], dtype=int64)
19
```

```
20 #其他方法
21 #按最后一列顺序排序
22 >>x[np.lexsort(x.T)]
23 array([[4, 18, 14],
24 [0, 12, 48],
25 [7, 1, 99]])
26
27 #按最后一列逆序排序
28 >>x[np.lexsort(-x.T)]
29 array([[7, 1, 99],
30 [0, 12, 48],
31 [4, 18, 14]])
32
33 #按第一列顺序排序
34 >>x[np.lexsort(x[:,::-1].T)]
35 array([[0, 12, 48],
36 [4, 18, 14],
37 [7, 1, 99]])
38
39 #按最后一行顺序排序
40 >>x.T[np.lexsort(x)].T
41 array([[12, 0, 48],
42 [18, 4, 14],
43 [1, 7, 99]])
44
45 #按第一行顺序排序
46 >>x.T[np.lexsort(x[:,::-1,:])].T
47 array([[0, 12, 48],
48 [4, 18, 14],
49 [7, 1, 99]])
```

## 7.2.6 唯一值与其它集合逻辑 unique和in1d

2020年4月6日 13:46

### 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]))
```

### 附：数组的集合操作 x和y是两个不同的数组

| 方法               | 描述                       |
|------------------|--------------------------|
| unique(x)        | 计算x的唯一值，并排序              |
| intersect1d(x,y) | 计算x和y的交集，并排序             |
| union1d(x,y)     | 计算x和y的并集，并排序             |
| in1d(x,y)        | 计算x中的元素是否包含在y中，返回一个布尔值数组 |
| setdiff1d(x,y)   | 差集，在x中但不在y中的x元素          |
| setxor1d(x,y)    | 异或集，在x或y中，但不属于x、y交集的元素   |

## 八、浅拷贝与深拷贝

2020年4月5日 13:27

### 浅拷贝

`a=b` 不能这样赋值，因为a和b相互影响，在内存里a变了b也会发生变化

`a=b[:]` 视图操作，一种切片，会创建新的对象a，但是a的数据完全由b保管，他们两个的数据变化是一致的

### 深拷贝

`a=b.copy()` 复制，a和b互不影响，相当于是重新开辟了一个空间保存b的值，然后让a指向b.copy()

# 附录：

2020年4月5日 9:48



## 1.random -- 生成伪随机数

2020年4月5日 9:48

模块概述:

**random** 模块实现了各种分布的伪随机数生成器。

函数概要: 随机数状态的相关函数

**random.seed(a=None, version=2)**

-- 初始化随机数生成器

参数: 含义

**a**

1. 如果省略该参数或者将其值设置为 **None** (默认), 将使用当前系统时间作为随机数种子 (如果操作系统提供了随机性来源, 则用它来代替系统时间)
2. 如果参数 **a** 为整数, 则直接被用作随机数种子
3. 该参数的值也可以是字符串、字节、字节数组等

**version**

1. 如果 **version=2** (默认), 字符串、字节或字节数组对象的每一个位都将比转换成整数使用
2. **version=1**, 用于从旧版本的 python 中复制随机序列, 字符串和字节算法生成更窄的种子范围

**random.getstate()**

-- 返回捕获当前生成器内部状态的对象。

-- 返回的对象可以传递给下面的 **setstate()** 函数, 用于恢复状态。

**random.setstate(state)**

-- 设置生成器的内部状态

-- 传入一个先前利用 **getstate()** 函数获得的状态对象, 使得生成器恢复到这个状态。

-- 小甲鱼: **getstate()** 和 **setstate()** 两个函数搭配使用, 可以重现之前获取到的随机值。

**random.getrandbits(k)**

-- 返回一个不大于 **k** 位的 Python 整数 (十进制), 比如 **k=10**, 则返回的结果是在  $0 \sim 2^{10}$  之间的整数。

整数相关的随机函数

**random.randrange(stop)**

**random.randrange(start, stop[, step])**

-- 从 **range(start, stop, step)** 中随机选择一个元素返回。

-- 功能相当于 **choice(range(start, stop, step))**, 但它不会创建一个 **range** 对象。

-- 传递的位置参数应该与 **range()** 模式匹配。

-- 不应该使用关键字参数, 因为函数可能以未定义的方式使用它们。

参数: 含义

**start**

1. 指定起始值
2. 如果省略该参数, 其默认值是 **0**

**stop**

指定结束值

**step**

1. 指定步长, 其值可以是正数也可以是负数

2. 如果省略该参数，其默认值是 1
3. 如果该参数被设置为 0，Python 将抛出 ValueError 异常

```
random.randint(a, b)
```

```
-- 返回一个随机整数 N，返回是：a <= N <= b
```

#### 序列相关的随机函数

```
random.choice(seq)
```

```
-- 从 seq 参数指定的序列中返回一个随机元素。
-- 如果 seq 是空序列，Python 将抛出 IndexError 异常。
```

```
random.choices(population, weights=None, *, cum_weights=None, k=1)
```

```
-- 从 population 参数指定的序列中随机抽取 k 个元素并返回。
-- weights 参数是指定相对权重列表，cum_weights 参数是指定累积权重列表（相对权重 [10, 5, 30, 5] 等同于累积权重 [10, 15, 45, 50]），两个参数不能同时存在（注：如果同时存在，Python 将抛出 TypeError 异常）。
-- 如果没有指定 weights 相对权重和 cum_weights 累积权重，那么每个元素被选中的概率是相同的。
-- 如果指定任一权重参数，那么其长度必须与 population 参数指定的序列长度一致。
```

```
random.shuffle(x[, random])
```

```
-- 原地打乱 x 参数指定的（可变）序列。
-- 可选参数是一个 0 参数函数，其返回一个范围在 [0.0, 1.0) 之间的随机浮点数，默认是使用 random() 函数。
-- 如果要打乱一个不可变序列（比如字符串），可以使用 sample(x, k=len(x)) 函数实现，它会生成一个元素打乱后的列表。
```

```
random.sample(population, k)
```

```
-- 从 population 参数指定的序列或集合中，随机抽取 k 个不重复的元素构成新序列并返回。
-- 该函数返回的是一个新的随机序列，不会破坏原序列，常用于不重复的随机抽样。
-- 如果 k 参数的值大于 population 参数指定的序列或集合的元素个数，Python 抛出 ValueError 异常。
-- 如果要从一个整数区间中随机抽取一定数量的整数，推荐使用 range() 对象作为参数（比如 sample(range(10000000), k=60)），这样实现的效率非常高并且节省内存空间。
```

#### 实值分布相关的随机函数

以下函数生成特定的随机实数分布。函数参数是根据分布方程式中的相应变量命名的，这在普通数学操作中经常用到；这些方程式大多数都可以在统计文本中被找到。

```
random.random()
```

```
-- 返回一个范围在 [0.0, 1.0) 之间的随机浮点数。
```

```
random.uniform(a, b)
```

```
-- 返回一个随机的浮点数 N。
-- 如果 a <= b，则 a <= N <= b。
-- 如果 b < a，则 b <= N <= a。
```

```
random.triangular(low, high, mode)
```

```
-- 返回一个三角分布的随机浮点数 N，其中 low <= N <= high。
-- 众数值通过 mode 参数指定。
-- low 和 high 参数的默认值是 0 和 1，mode 参数的默认值是边界之间的中心点。
```

```
random.betavariate(alpha, beta)
```

```
-- 返回一个 Beta 分布的随机浮点数。
-- 参数 alpha 和 beta 都应该大于 0。
-- 返回值的返回在 0 到 1 之间。
```

```
random.expovariate(lambd)
```

```
-- 指数分布。
-- lambd 参数的值是 1/期望值 的结果，所以是一个非 0 值。
-- 如果 lambd 参数的值为正，则返回值的范围为 0 到正无穷大；如果 lambd 参数的值为负，则返回值的范围为 0 到负无穷大。
```

```
random.gammavariate(alpha, beta)
```

```
-- Gamma 分布（不是 Gamma 函数！）。
-- 条件参数 alpha > 0，且 beta > 0。
-- 概率分布函数如下：
```

```
 x ** (alpha - 1) * math.exp(-x / beta)
pdf(x) = -----
 math.gamma(alpha) * beta ** alpha
复制代码
```

```
random.gauss(mu, sigma)
```

```
-- 高斯分布。
-- mu 参数指定的是均值，sigma 参数指定的是标准差。
-- 该函数要比下面的 normalvariate() 函数（正态分布函数）稍微快一些。
```

```
random.lognormvariate(mu, sigma)
```

```
-- 对数正态分布。
-- mu 参数指定的是均值，sigma 参数指定的是标准差。
-- mu 参数可以是任意值，但 sigma 参数的值必须大于 0。
```

```
random.normalvariate(mu, sigma)
```

```
-- 正态分布。
-- mu 参数指定的是均值，sigma 参数指定的是标准差。
```

```
random.vonmisesvariate(mu, kappa)
```

-- 卡方分布。  
-- mu 参数指定的是平均角度，以 0 到  $2\pi$  之间的弧度表示。  
-- kappa 参数指定的是自由度，必须大于或者等于 0。  
-- 如果 kappa 参数的值等于 0，该分布在 0 到  $2\pi$  范围内减少到一个均匀的随机角。

`random.paretovariate(alpha)`

-- 帕累托分布（小甲鱼：20% 的人口拥有 80% 的财产就是这个）。  
-- alpha 参数指定的是形状参数。

`random.weibullvariate(alpha, beta)`

-- Weibull 分布。  
-- alpha 参数指定的是比例参数。  
-- beta 参数指定的是形状参数。

## 替代生成器

`class random.SystemRandom([seed])`

-- 使用 `os.urandom()` 函数的类，利用操作系统提供的源来生成随机数。  
-- 该类并不是所有操作系统都适用。  
-- 该类不依赖于软件状态，并且序列不可重现。所以，`seed()` 方法没有效果而被忽略，`getstate()` 和 `setstate()` 方法如果被调用则抛出 `NotImplementedError` 异常。

## 关于“再现”的备注

有时候，能够再现伪随机数生成器给出的序列是有用的。

通过重新使用相同的种子值（只要多个线程没有运行），相同的伪随机数序列就应该可以被生成。

随着 Python 版本的迭代，随机模块的算法和种子设定函数有可能会发生改变，但以下两个方面保证不会有变动：

如果添加了新的种子生成方法，则将提供向后兼容的方案  
当提供的种子一样时，`random()` 方法将生成一样的伪随机数

## 一、基本用法

```

01. >>> random() # 返回一个浮点数 x: 0.0 <= x < 1.0
02. 0.37444887175646646
03.
04. >>> uniform(2.5, 10.0) # 返回一个浮点数 x: 2.5 <= x < 10.0
05. 3.1800146073117523
06.
07. >>> expovariate(1 / 5) # 指数分布, 平均到达时间为 5 秒
08. 5.148957571865031
09.
10. >>> randrange(10) # 从 0 ~ 9 (包含) 中随机取出一个整数
11. 7
12.
13. >>> randrange(0, 101, 2) # 从 0 ~ 100 (包含) 中随机取出一个偶数
14. 26
15.
16. >>> choice(['win', 'lose', 'draw']) # 从序列中随机取出一个元素
17. 'draw'
18.
19. >>> deck = 'ace two three four'.split()
20. >>> shuffle(deck) # 打乱一个列表
21. >>> deck
22. ['four', 'two', 'ace', 'three']
23.
24. >>> sample([10, 20, 30, 40, 50], k=4) # 在给定的序列中生成 4 个随机数
25. [40, 10, 50, 30]

```

## 二、情景模拟

```

01. >>> # 模拟黑红绿轮盘 (带权重的随机抽取)
02. >>> choices(['red', 'black', 'green'], [18, 18, 2], k=6)
03. ['red', 'green', 'black', 'black', 'red', 'black']
04.
05. >>> # 模拟从 52 张扑克牌中抽取 20 张不可替代的牌
06. >>> # 并确定 “10” 以上的牌 (10, J, Q, K) 所占比例
07. >>> deck = collections.Counter(tens=16, low_cards=36)
08. >>> seen = sample(list(deck.elements()), k=20)
09. >>> seen.count('tens') / 20
10. 0.15
11.
12. >>> # 模拟扔硬币, 并假设扔到 “字” 的概率十 60%
13. >>> # 统计扔 7 次硬币, 有 5 次以上扔到 “字” 的概率
14. >>> def trial():
15. ... return choices('HT', cum_weights=(0.60, 1.00), k=7).count('H') >= 5
16. ...
17. >>> sum(trial() for i in range(10000)) / 10000
18. 0.4169
19.
20. >>> # 取 5 个样本的中位数, 统计中间数值的概率
21. >>> def trial():
22. ... return 2500 <= sorted(choices(range(10000), k=5))[2] < 7500
23. ...
24. >>> sum(trial() for i in range(10000)) / 10000
25. 0.7958

```

## 三、自助抽样法 (统计学)

```

01. # http://statistics.about.com/od/Applications/a/Example-Of-Bootstrapping.htm
02. from statistics import fmean as mean
03. from random import choices
04.
05. data = 1, 2, 4, 4, 10
06. means = sorted(mean(choices(data, k=5)) for i in range(20))
07. print(f'The sample mean of {mean(data):.1f} has a 90% confidence '
08. f'interval from {means[1]:.1f} to {means[-2]:.1f}')

```

## 四、重新抽样 (统计学)

```

01. # Example from "Statistics is Easy" by Dennis Shasha and Manda Wilson
02. from statistics import fmean as mean
03. from random import shuffle
04.
05. drug = [54, 73, 53, 70, 73, 68, 52, 65, 65]
06. placebo = [54, 51, 58, 44, 55, 52, 42, 47, 58, 46]
07. observed_diff = mean(drug) - mean(placebo)
08.
09. n = 10000
10. count = 0
11. combined = drug + placebo
12. for i in range(n):
13. shuffle(combined)
14. new_diff = mean(combined[:len(drug)]) - mean(combined[len(drug):])
15. count += (new_diff >= observed_diff)
16.
17. print(f'{n} label reshufflings produced only {count} instances with a difference')
18. print(f'at least as extreme as the observed difference of {observed_diff:.1f}.')
19. print(f'The one-sided p-value of {count / n:.4f} leads us to reject the null')
20. print(f'hypothesis that there is no difference between the drug and the placebo.')

```

## 五、模拟单个服务器队列中的到达时间和服务传递：

```
01. from random import expovariate, gauss
02. from statistics import mean, median, stdev
03.
04. average_arrival_interval = 5.6
05. average_service_time = 5.0
06. stdev_service_time = 0.5
07.
08. num_waiting = 0
09. arrivals = []
10. starts = []
11. arrival = service_end = 0.0
12. for i in range(20000):
13. if arrival <= service_end:
14. num_waiting += 1
15. arrival += expovariate(1.0 / average_arrival_interval)
16. arrivals.append(arrival)
17. else:
18. num_waiting -= 1
19. service_start = service_end if num_waiting else arrival
20. service_time = gauss(average_service_time, stdev_service_time)
21. service_end = service_start + service_time
22. starts.append(service_start)
23.
24. waits = [start - arrival for arrival, start in zip(arrivals, starts)]
25. print(f'Mean wait: {mean(waits):.1f}. Stdev wait: {stdev(waits):.1f}.')
26. print(f'Median wait: {median(waits):.1f}. Max wait: {max(waits):.1f}.')
```