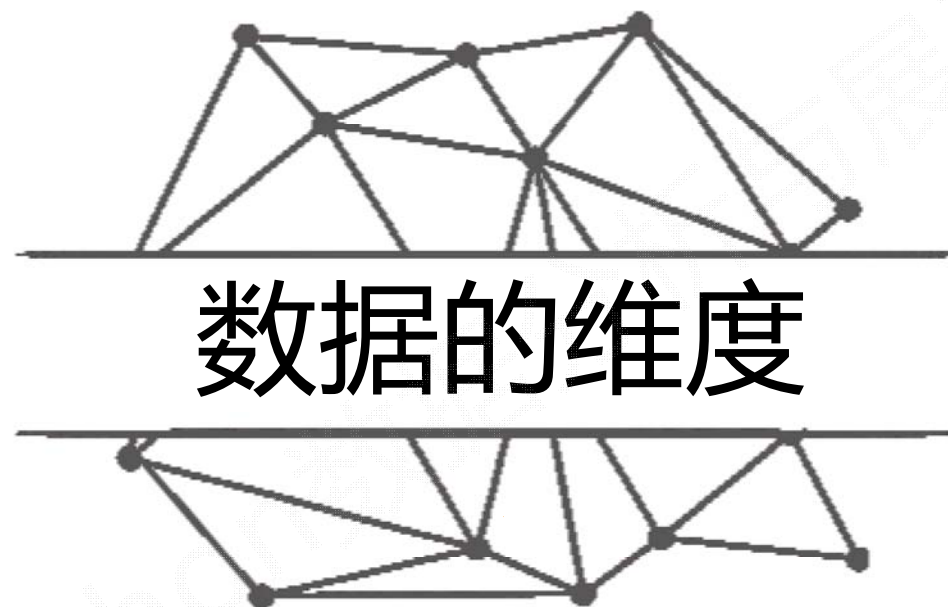


NumPy库入门





数据的维度

从一个数据到一组数据

3.14



3.1413 3.1404
3.1401 3.1376
3.1398 3.1349

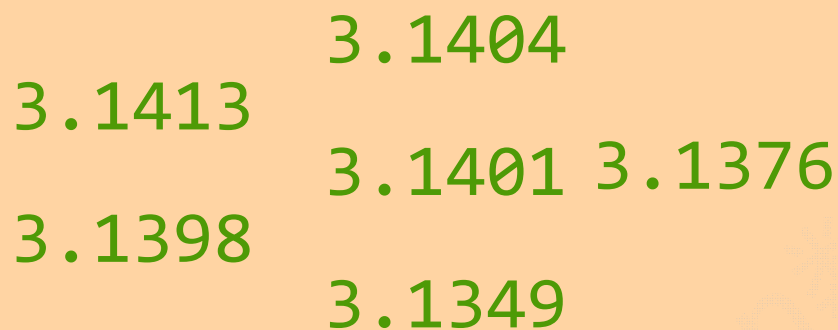
一个数据

表达一个含义

一组数据

表达一个或多个含义

维度：一组数据的组织形式



3.1413 3.1404
3.1398 3.1401 3.1376
3.1349

一组数据

→
3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

或

→
3.1398, 3.1349, 3.1376
3.1413, 3.1404, 3.1401

数据的组织形式

一维数据

一维数据由对等关系的有序或无序数据构成，采用线性方式组织

3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

对应列表、数组和集合等概念

列表和数组

一组数据的有序结构

区别

列表：数据类型可以不同

3.1413, 'pi', 3.1404, [3.1401, 3.1349], '3.1376'

数组：数据类型相同

3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

二维数据

二维数据由多个一维数据构成，是一维数据的组合形式

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分） ▾
1	清华大学	北京	94.0	100.0
2	北京大学	北京	81.2	96.1
3	浙江大学	浙江	77.8	87.2
4	上海交通大学	上海	77.5	89.4
5	复旦大学	上海	71.1	91.8
6	中国科学技术大学	安徽	65.9	91.9
7	南京大学	江苏	65.3	87.1
8	华中科技大学	湖北	63.0	80.6
9	中山大学	广东	62.7	81.1
10	哈尔滨工业大学	黑龙江	61.6	76.4

表格是典型的二维数据
其中，表头是二维数据的一部分

多维数据

多维数据由一维或二维数据在新维度上扩展形成

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分）
1	清华大学	北京市	95.9	100.0
2	北京大学	北京市	82.6	98.9
3	浙江大学	浙江省	80	88.8
4	上海交通大学	上海市	78.7	90.6
5	复旦大学	上海市	70.9	90.4
6	南京大学	江苏省	66.1	90.7
7	中国科学技术大学	安徽省	65.5	90.1
8	哈尔滨工业大学	黑龙江省	63.5	80.9
9	华中科技大学	湖北省	62.9	83.5
10	中山大学	广东省	62.1	81.8

时间维度



2016

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分）
1	清华大学	北京	94.0	100.0
2	北京大学	北京	81.2	96.1
3	浙江大学	浙江	77.8	87.2
4	上海交通大学	上海	77.5	89.4
5	复旦大学	上海	71.1	91.8
6	中国科学技术大学	安徽	65.9	91.9
7	南京大学	江苏	65.3	87.1
8	华中科技大学	湖北	63.0	80.6
9	中山大学	广东	62.7	81.1
10	哈尔滨工业大学	黑龙江	61.6	76.4

2017

高维数据

高维数据仅利用最基本的二元关系展示数据间的复杂结构

```
{  
  "firstName" : "Tian" ,  
  "lastName"  : "Song" ,  
  "address"   : {  
    "streetAddr" : "中关村南大街5号" ,  
    "city"       : "北京市" ,  
    "zipcode"    : "100081"  
  } ,  
  "prof"      : [ "Computer System" , "Security" ]  
}
```

键值对

数据维度的Python表示

数据维度是数据的组织形式

一维数据：列表和集合类型

`[3.1398, 3.1349, 3.1376]` 有序

`{3.1398, 3.1349, 3.1376}` 无序

二维数据：列表类型

`[[3.1398, 3.1349, 3.1376],`

`[3.1413, 3.1404, 3.1401]]`

多维数据：列表类型

数据维度的Python表示

数据维度是数据的组织形式

高维数据：字典类型 或

数据表示格式

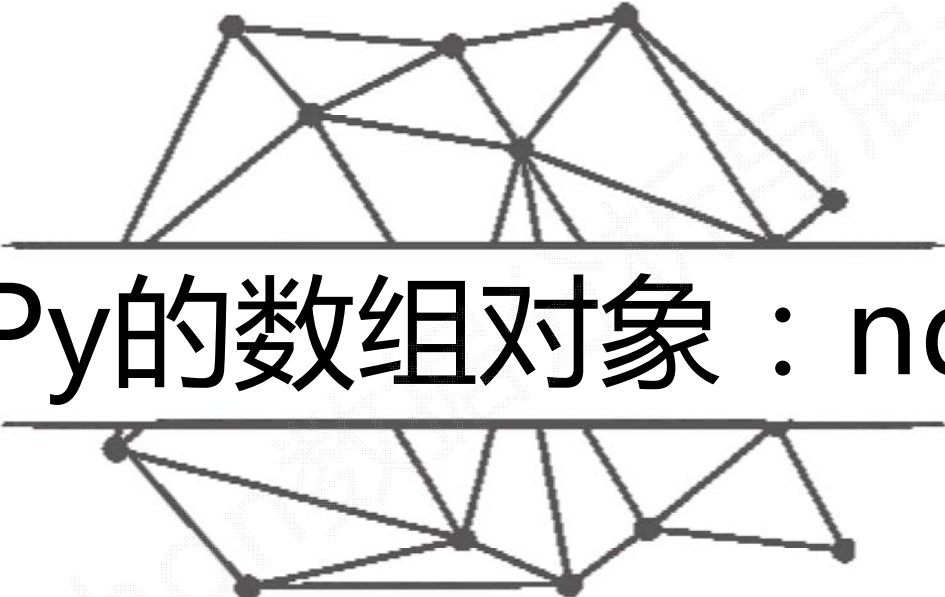
JSON、XML和YAML格式

```
dict = {
```

```
    "firstName" : "Tian",
```

```
    "lastName" : "Song" ,
```

```
}
```



NumPy的数组对象：ndarray

NumPy

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
- 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的引用

```
import numpy as np
```



引入模块的别名

尽管别名可以省略或更改，建议使用上述约定的别名

N维数组对象：ndarray

Python已有列表类型，为什么需要一个数组对象(类型)？

例：计算 A^2+B^3 ，其中，A和B是一维数组

```
def pySum():  
    a = [0, 1, 2, 3, 4]  
    b = [9, 8, 7, 6, 5]  
    c = []  
  
    for i in range(len(a)):  
        c.append(a[i]**2 + b[i]**3)  
  
    return c  
  
print(pySum())
```



```
import numpy as np  
  
def npSum():  
    a = np.array([0, 1, 2, 3, 4])  
    b = np.array([9, 8, 7, 6, 5])  
  
    c = a**2 + b**3  
  
    return c  
  
print(npSum())
```

N维数组对象：ndarray

Python已有列表类型，为什么需要一个数组对象(类型)？

- 数组对象可以去掉元素间运算所需的循环，使一维向量更像单个数据
- 设置专门的数组对象，经过优化，可以提升这类应用的运算速度

观察：科学计算中，一个维度所有数据的类型往往相同

- 数组对象采用相同的数据类型，有助于节省运算和存储空间

N维数组对象：ndarray

ndarray是一个多维数组对象，由两部分构成：

- 实际的数据
- 描述这些数据的元数据（数据维度、数据类型等）

ndarray数组一般要求所有元素类型相同（同质），数组下标从0开始

ndarray实例

IPython提示符

ndarray在程序中的别名是:array

```
In [19]: a = np.array([[0, 1, 2, 3, 4],  
...:                  [9, 8, 7, 6, 5]])
```

np.array()生成一个ndarray数组

```
In [20]: a
```

```
Out[20]:
```

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

```
In [21]: print(a)
```

```
[[0 1 2 3 4]  
 [9 8 7 6 5]]
```

np.array()输出成[]形式，元素由空格分割

轴(axis): 保存数据的维度；秩(rank): 轴的数量

ndarray对象的属性

属性	说明
<code>.ndim</code>	秩，即轴的数量或维度的数量
<code>.shape</code>	ndarray对象的尺度，对于矩阵，n行m列
<code>.size</code>	ndarray对象元素的个数，相当于 <code>.shape</code> 中 $n*m$ 的值
<code>.dtype</code>	ndarray对象的元素类型
<code>.itemsize</code>	ndarray对象中每个元素的大小，以字节为单位

ndarray实例

```
In [22]: a = np.array([[0, 1, 2, 3, 4],  
...:                  [9, 8, 7, 6, 5]])
```

```
In [23]: a.ndim  
Out[23]: 2
```

```
In [24]: a.shape  
Out[24]: (2, 5)
```

```
In [25]: a.size  
Out[25]: 10
```

```
In [26]: a.dtype  
Out[26]: dtype('int32')
```

```
In [27]: a.itemsize  
Out[27]: 4
```



ndarray数组的元素类型

ndarray的元素类型(1)

数据类型	说明
bool	布尔类型，True或False
intc	与C语言中的int类型一致，一般是int32或int64
intp	用于索引的整数，与C语言中ssize_t一致，int32或int64
int8	字节长度的整数，取值： $[-128, 127]$
int16	16位长度的整数，取值： $[-32768, 32767]$
int32	32位长度的整数，取值： $[-2^{31}, 2^{31}-1]$
int64	64位长度的整数，取值： $[-2^{63}, 2^{63}-1]$

ndarray的元素类型(2)

数据类型	说明
uint8	8位无符号整数，取值： $[0, 255]$
uint16	16位无符号整数，取值： $[0, 65535]$
uint32	32位无符号整数，取值： $[0, 2^{32}-1]$
uint64	32位无符号整数，取值： $[0, 2^{64}-1]$
float16	16位半精度浮点数：1位符号位，5位指数，10位尾数
float32	32位半精度浮点数：1位符号位，8位指数，23位尾数
float64	64位半精度浮点数：1位符号位，11位指数，52位尾数

ndarray的元素类型(3)

数据类型	说明
complex64	复数类型，实部和虚部都是32位浮点数
complex128	复数类型，实部和虚部都是64位浮点数

实部(.real) + j虚部(.imag)

ndarray的元素类型

ndarray为什么要支持这么多种元素类型？

对比：Python语法仅支持整数、浮点数和复数3种类型

- 科学计算涉及数据较多，对存储和性能都有较高要求
- 对元素类型精细定义，有助于NumPy合理使用存储空间并优化性能
- 对元素类型精细定义，有助于程序员对程序规模有合理评估

非同质的ndarray对象

```
In [62]: x = np.array([ [0, 1, 2, 3, 4],  
...:                  [9, 8, 7, 6] ])
```

ndarray数组可以由非同质对象构成

```
In [63]: x.shape  
Out[63]: (2,)
```

```
In [64]: x.dtype  
Out[64]: dtype('O')
```

```
In [65]: x  
Out[65]: array([[0, 1, 2, 3, 4], [9, 8, 7, 6]],  
dtype=object)
```

非同质ndarray元素为对象类型

```
In [66]: x.itemsize  
Out[66]: 4
```

```
In [67]: x.size  
Out[67]: 2
```

非同质ndarray对象无法有效发挥NumPy优势，尽量避免使用



ndarray数组的创建



Python 数据科学入门

ndarray数组的创建方法

- 从Python中的列表、元组等类型创建ndarray数组
- 使用NumPy中函数创建ndarray数组，如：arange, ones, zeros等
- 从字节流 (raw bytes) 中创建ndarray数组
- 从文件中读取特定格式，创建ndarray数组

ndarray数组的创建方法

(1) 从Python中的列表、元组等类型创建ndarray数组

```
x = np.array(list/tuple)
```

```
x = np.array(list/tuple, dtype=np.float32)
```

当np.array()不指定dtype时，NumPy将根据数据情况关联一个dtype类型

ndarray数组的创建方法

(1) 从Python中的列表、元组等类型创建ndarray数组

```
In [32]: x = np.array([0, 1, 2, 3])
```

从列表类型创建

```
In [33]: print(x)  
[0 1 2 3]
```

```
In [34]: x = np.array((4, 5, 6, 7))
```

从元组类型创建

```
In [35]: print(x)  
[4 5 6 7]
```

```
In [36]: x = np.array([ [1, 2], [9, 8], (0.1, 0.2)])
```

```
In [37]: print(x)  
[[ 1.  2. ]  
 [ 9.  8. ]  
 [ 0.1 0.2]]
```

从列表和元组混合类型创建

ndarray数组的创建方法

(2) 使用NumPy中函数创建ndarray数组，如：arange, ones, zeros等

函数	说明
<code>np.arange(n)</code>	类似 <code>range()</code> 函数，返回ndarray类型，元素从0到n-1
<code>np.ones(shape)</code>	根据shape生成一个全1数组，shape是元组类型
<code>np.zeros(shape)</code>	根据shape生成一个全0数组，shape是元组类型
<code>np.full(shape, val)</code>	根据shape生成一个数组，每个元素值都是val
<code>np.eye(n)</code>	创建一个正方的n*n单位矩阵，对角线为1，其余为0

ndarray数组的创建方法

```
In [73]: np.arange(10)
Out[73]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [74]: np.ones((3,6))
Out[74]:
array([[ 1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.]])
```

```
In [75]: np.zeros((3,6), dtype=np.int32)
Out[75]:
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])
```

```
In [76]: np.eye(5)
Out[76]:
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

```
In [81]: x = np.ones((2,3,4))
```

```
In [82]: print(x)
[[[ 1.  1.  1.  1.]
   [ 1.  1.  1.  1.]
   [ 1.  1.  1.  1.]]
 [[ 1.  1.  1.  1.]
   [ 1.  1.  1.  1.]
   [ 1.  1.  1.  1.]])
```

```
In [83]: x.shape
Out[83]: (2, 3, 4)
```


ndarray数组的创建方法

(2) 使用NumPy中函数创建ndarray数组，如：arange, ones, zeros等

函数	说明
<code>np.ones_like(a)</code>	根据数组a的形状生成一个全1数组
<code>np.zeros_like(a)</code>	根据数组a的形状生成一个全0数组
<code>np.full_like(a, val)</code>	根据数组a的形状生成一个数组，每个元素值都是val

ndarray数组的创建方法

(3) 使用NumPy中其他函数创建ndarray数组

函数	说明
<code>np.linspace()</code>	根据起止数据等间距地填充数据，形成数组
<code>np.concatenate()</code>	将两个或多个数组合并成一个新的数组

ndarray数组的创建方法

```
In [51]: a = np.linspace(1, 10, 4)
```

```
In [52]: a
```

```
Out[52]: array([ 1.,  4.,  7., 10.])
```

```
In [53]: b = np.linspace(1, 10, 4, endpoint=False)
```

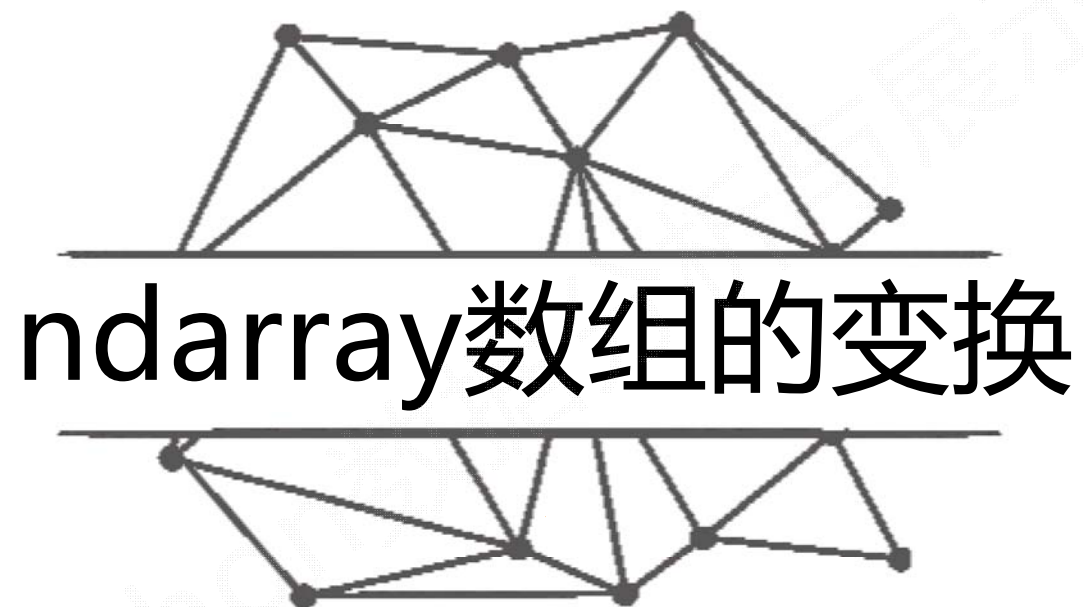
```
In [54]: b
```

```
Out[54]: array([ 1. ,  3.25,  5.5 ,  7.75])
```

```
In [56]: c = np.concatenate((a, b))
```

```
In [57]: c
```

```
Out[57]: array([ 1. ,  4. ,  7. , 10. ,  1. ,  3.25,  5.5 ,  7.75])
```



ndarray数组的变换

ndarray数组的变换

对于创建后的ndarray数组，可以对其进行维度变换和元素类型变换

```
a = np.ones((2, 3, 4), dtype=np.int32)
```

ndarray数组的维度变换

方法	说明
<code>.reshape(shape)</code>	不改变数组元素，返回一个shape形状的数组，原数组不变
<code>.resize(shape)</code>	与 <code>.reshape()</code> 功能一致，但修改原数组
<code>.swapaxes(ax1, ax2)</code>	将数组n个维度中两个维度进行调换
<code>.flatten()</code>	对数组进行降维，返回折叠后的一维数组，原数组不变

ndarray数组的维度变换

```
a = np.ones((2,3,4), dtype=np.int32)
```

```
In [105]: a.reshape((3, 8))
```

```
Out[105]:  
array([[1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1]])
```

```
In [106]: a
```

```
Out[106]:  
array([[[1, 1, 1, 1],  
        [1, 1, 1, 1],  
        [1, 1, 1, 1]],  
       [[1, 1, 1, 1],  
        [1, 1, 1, 1],  
        [1, 1, 1, 1]]])
```

```
In [107]: a.resize((3,8))
```

```
In [108]: a
```

```
Out[108]:  
array([[1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1]])
```

ndarray数组的维度变换

```
a = np.ones((2,3,4), dtype=np.int32)
```

```
In [109]: a.flatten()  
Out[109]: array([1, 1, 1, ..., 1, 1, 1])
```

```
In [110]: a  
Out[110]:  
array([[1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1],  
       [1, 1, 1, ..., 1, 1, 1]])
```

```
In [111]: b = a.flatten()
```

```
In [112]: b  
Out[112]: array([1, 1, 1, ..., 1, 1, 1])
```


ndarray数组的类型变换

`new_a = a.astype(new_type)`

```
In [119]: a = np.ones((2,3,4), dtype=np.int)
```

```
In [120]: a
```

```
Out[120]:
```

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

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

```
In [121]: b = a.astype(np.float)
```

```
In [122]: b
```

```
Out[122]:
```

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

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

`astype()`方法一定会创建新的数组（原始数据的一个拷贝），即使两个类型一致

ndarray数组向列表的转换

```
ls = a.tolist()
```

```
In [128]: a = np.full((2,3,4), 25, dtype=np.int32)
```

```
In [129]: a
```

```
Out[129]:
```

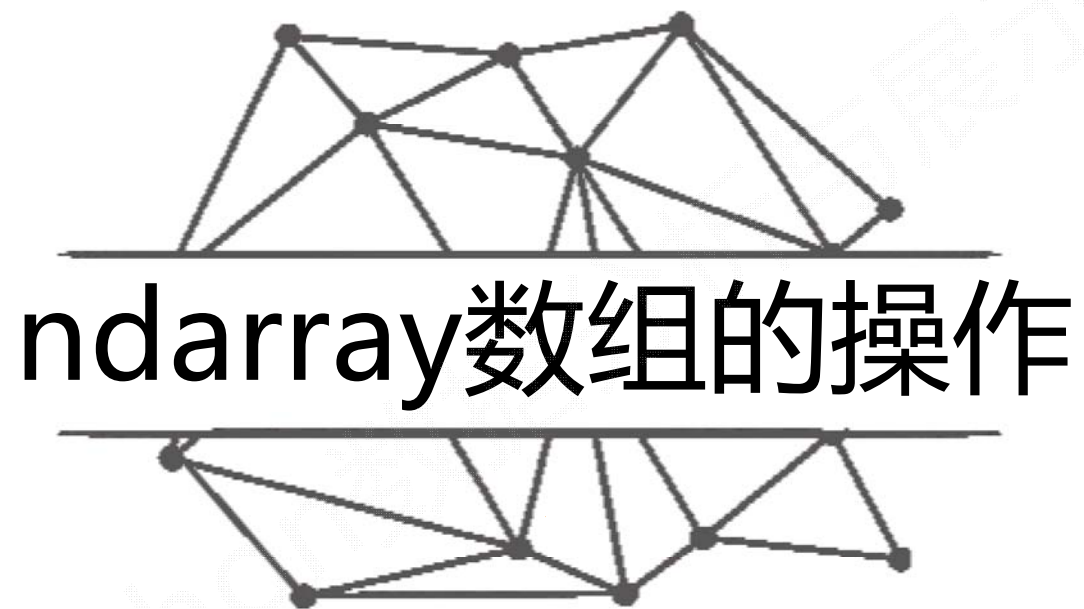
```
array([[[25, 25, 25, 25],
        [25, 25, 25, 25],
        [25, 25, 25, 25]],

       [[25, 25, 25, 25],
        [25, 25, 25, 25],
        [25, 25, 25, 25]]])
```

```
In [130]: a.tolist()
```

```
Out[130]:
```

```
[[[25, 25, 25, 25], [25, 25, 25, 25], [25, 25, 25, 25]],
 [[25, 25, 25, 25], [25, 25, 25, 25], [25, 25, 25, 25]]]
```



ndarray数组的操作

数组的索引和切片

索引：获取数组中特定位置元素的过程

切片：获取数组元素子集的过程

数组的索引和切片

一维数组的索引和切片：与Python的列表类似

```
In [131]: a = np.array([9, 8, 7, 6, 5])
```

```
In [132]: a[2]
```

```
Out[132]: 7
```

```
In [133]: a[ 1 : 4 : 2 ]
```

```
Out[133]: array([8, 6])
```

起始编号：终止编号(不含)：步长，3元素冒号分割

编号0开始从左递增，或-1开始从右递减

数组的索引和切片

多维数组的索引：

```
In [146]: a = np.arange(24).reshape((2,3,4))
```

```
In [147]: a
```

```
Out[147]:
```

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

```
In [148]: a[1, 2, 3]
```

```
Out[148]: 23
```

```
In [149]: a[0, 1, 2]
```

```
Out[149]: 6
```

```
In [150]: a[-1, -2, -3]
```

```
Out[150]: 17
```

每个维度一个索引值，逗号分割

数组的索引和切片

多维数组的切片：

```
In [146]: a = np.arange(24).reshape((2,3,4))
```

```
In [147]: a
```

```
Out[147]:
```

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

```
In [158]: a[:, 1, -3]
```

```
Out[158]: array([ 5, 17])
```

选取一个维度用：

```
In [159]: a[:, 1:3, :]
```

```
Out[159]:
```

```
array([[[ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

每个维度切片方法与一维数组相同

```
In [160]: a[:, :, ::2]
```

```
Out[160]:
```

```
array([[[ 0,  2],
        [ 4,  6],
        [ 8, 10]],
       [[12, 14],
        [16, 18],
        [20, 22]]])
```

每个维度可以使用步长跳跃切片



ndarray数组的运算

数组与标量之间的运算

数组与标量之间的运算作用于数组的每一个元素

实例

```
In [146]: a = np.arange(24).reshape((2,3,4))
```

```
In [147]: a
```

```
Out[147]:
```

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

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

```
In [169]: a.mean()
```

```
Out[169]: 11.5
```

计算a与元素平均值的商

```
In [170]: a = a / a.mean()
```

```
In [171]: a
```

```
Out[171]:
```

```
array([[[ 0.          ,  0.08695652,  0.17391304,  0.26086957],
        [ 0.34782609,  0.43478261,  0.52173913,  0.60869565],
        [ 0.69565217,  0.7826087 ,  0.86956522,  0.95652174]],

       [[ 1.04347826,  1.13043478,  1.2173913 ,  1.30434783],
        [ 1.39130435,  1.47826087,  1.56521739,  1.65217391],
        [ 1.73913043,  1.82608696,  1.91304348,  2.          ]]])
```

NumPy一元函数

对ndarray中的数据执行元素级运算的函数

函数	说明
<code>np.abs(x)</code> <code>np.fabs(x)</code>	计算数组各元素的绝对值
<code>np.sqrt(x)</code>	计算数组各元素的平方根
<code>np.square(x)</code>	计算数组各元素的平方
<code>np.log(x)</code> <code>np.log10(x)</code> <code>np.log2(x)</code>	计算数组各元素的自然对数、10底对数和2底对数
<code>np.ceil(x)</code> <code>np.floor(x)</code>	计算数组各元素的ceiling值 或 floor值

NumPy一元函数

函数	说明
<code>np rint(x)</code>	计算数组各元素的四舍五入值
<code>np.modf(x)</code>	将数组各元素的小数和整数部分以两个独立数组形式返回
<code>np.cos(x)</code> <code>np.cosh(x)</code> <code>np.sin(x)</code> <code>np.sinh(x)</code> <code>np.tan(x)</code> <code>np.tanh(x)</code>	计算数组各元素的普通型和双曲型三角函数
<code>np.exp(x)</code>	计算数组各元素的指数值
<code>np.sign(x)</code>	计算数组各元素的符号值，1(+), 0, -1(-)

NumPy一元函数实例

```
In [176]: a = np.arange(24).reshape((2,3,4))
```

```
In [177]: np.square(a)
```

```
Out[177]:  
array([[[ 0,  1,  4,  9],  
        [16, 25, 36, 49],  
        [64, 81, 100, 121]],  
       [[144, 169, 196, 225],  
        [256, 289, 324, 361],  
        [400, 441, 484, 529]]], dtype=int32)
```

```
In [178]: a = np.sqrt(a)
```

```
In [179]: a
```

```
Out[179]:  
array([[[ 0.,  1.,  1.41421356,  1.73205081],  
        [ 2.,  2.23606798,  2.44948974,  2.64575131],  
        [ 2.82842712,  3.,  3.16227766,  3.31662479]],  
       [[ 3.46410162,  3.60555128,  3.74165739,  3.87298335],  
        [ 4.,  4.12310563,  4.24264069,  4.35889894],  
        [ 4.47213595,  4.58257569,  4.69041576,  4.79583152]])])
```

```
In [180]: np.modf(a)
```

```
Out[180]:
```

```
(array([[[ 0.,  0.,  0.41421356,  0.73205081],  
        [ 0.,  0.23606798,  0.44948974,  0.64575131],  
        [ 0.82842712,  0.,  0.16227766,  0.31662479]],  
       [[ 0.46410162,  0.60555128,  0.74165739,  0.87298335],  
        [ 0.,  0.12310563,  0.24264069,  0.35889894],  
        [ 0.47213595,  0.58257569,  0.69041576,  0.79583152]]]),  
array([[[ 0.,  1.,  1.,  1.],  
        [ 2.,  2.,  2.,  2.],  
        [ 2.,  3.,  3.,  3.]],  
       [[ 3.,  3.,  3.,  3.],  
        [ 4.,  4.,  4.,  4.],  
        [ 4.,  4.,  4.,  4.]])])
```

注意数组是否被真实改变

NumPy二元函数

函数	说明
<code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>**</code>	两个数组各元素进行对应运算
<code>np.maximum(x,y)</code> <code>np.fmax()</code> <code>np.minimum(x,y)</code> <code>np.fmin()</code>	元素级的最大值/最小值计算
<code>np.mod(x,y)</code>	元素级的模运算
<code>np.copysign(x,y)</code>	将数组y中各元素值的符号赋值给数组x对应元素
<code>></code> <code><</code> <code>>=</code> <code><=</code> <code>==</code> <code>!=</code>	算术比较，产生布尔型数组

NumPy二元函数实例

```
In [203]: a = np.arange(24).reshape((2,3,4))
```

```
In [204]: b = np.sqrt(a)
```

```
In [205]: a
```

```
Out[205]:
```

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

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

```
In [206]: b
```

```
Out[206]:
```

```
array([[[ 0.          ,  1.          ,  1.41421356,  1.73205081],
        [ 2.          ,  2.23606798,  2.44948974,  2.64575131],
        [ 2.82842712,  3.          ,  3.16227766,  3.31662479]],

       [[ 3.46410162,  3.60555128,  3.74165739,  3.87298335],
        [ 4.          ,  4.12310563,  4.24264069,  4.35889894],
        [ 4.47213595,  4.58257569,  4.69041576,  4.79583152]])
```

```
In [207]: np.maximum(a, b)
```

```
Out[207]:
```

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

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

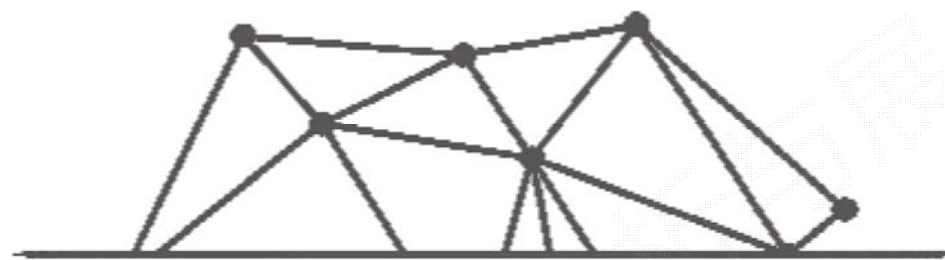
运算结果为浮点数

```
In [208]: a > b
```

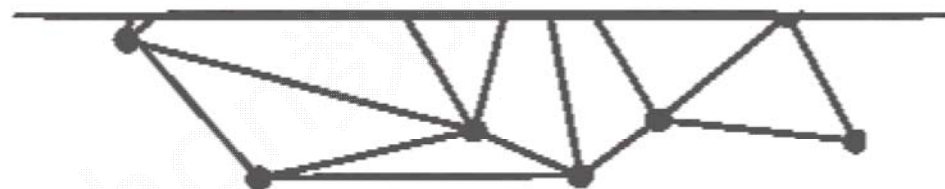
```
Out[208]:
```

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

       [[ True,  True,  True,  True],
        [ True,  True,  True,  True],
        [ True,  True,  True,  True]]], dtype=bool)
```



单元小结



NumPy库入门

数据的维度：一维、多维、高维

ndarray类型属性、创建和变换

.ndim	.reshape(shape)
.shape	.resize(shape)
.size	.swapaxes(ax1,ax2)
.dtype	.flatten()
.itemsize	

np.arange(n)
np.ones(shape)
np.zeros(shape)
np.full(shape,val)
np.eye(n)
np.ones_like(a)
np.zeros_like(a)
np.full_like(a,val)

数组的索引
和切片

数组的运算

一元函数

二元函数



NumPy的随机数函数

NumPy的随机数函数子库

NumPy的random子库

`np.random.*`

`np.random.rand()`

`np.random.randn()`

`np.random.randint()`

np.random的随机数函数(1)

函数	说明
<code>rand(d0,d1,...,dn)</code>	根据d0-dn创建随机数数组，浮点数， $[0,1)$ ，均匀分布
<code>randn(d0,d1,...,dn)</code>	根据d0-dn创建随机数数组，标准正态分布
<code>randint(low[,high,shape])</code>	根据shape创建随机整数或整数数组，范围是 $[low, high)$
<code>seed(s)</code>	随机数种子，s是给定的种子值

函数小试

```
In [5]: import numpy as np
```

```
In [6]: a = np.random.rand(3, 4, 5)
```

```
In [7]: a
```

```
Out[7]:
```

```
array([[[ 0.93859387,  0.01070518,  0.3054871 ,  0.1932416 ,  0.82375036],
        [ 0.15110071,  0.34448139,  0.21612265,  0.43276404,  0.73471971],
        [ 0.00407316,  0.70711519,  0.05127404,  0.67731786,  0.14322067],
        [ 0.88997925,  0.96002098,  0.33277737,  0.59770084,  0.57604945]],

       [[ 0.52441722,  0.14175617,  0.08588264,  0.62617497,  0.72711516],
        [ 0.63504074,  0.21290387,  0.77465841,  0.47369419,  0.78394602],
        [ 0.68891405,  0.3880887 ,  0.60886227,  0.50600248,  0.31468346],
        [ 0.34277096,  0.15791136,  0.14749979,  0.25235406,  0.03123494]],

       [[ 0.6084322 ,  0.51827266,  0.2855457 ,  0.92409508,  0.15750942],
        [ 0.00218532,  0.13749523,  0.73366243,  0.33392875,  0.31355293],
        [ 0.18500307,  0.16201531,  0.66444529,  0.34702364,  0.17776621],
        [ 0.0063955 ,  0.61556899,  0.93334567,  0.97117129,  0.85570959]]])
```

```
In [8]: sn = np.random.randn(3, 4, 5)
```

```
In [9]: sn
```

```
Out[9]:
```

```
array([[[ 0.80082098,  0.81195729, -2.02593352, -0.43980132, -0.84428853],
        [ 0.40876378,  1.0471804 , -0.86464492,  0.20691501, -0.32795862],
        [-0.72705744, -0.95187079, -0.79302798,  1.38466351, -0.64609614],
        [ 0.41632154,  0.08465048, -0.14857112, -0.93631024,  0.13784205]],

       [[-1.08984876,  0.12145932,  1.07303554, -1.95695301, -1.02073863],
        [ 0.60733033,  0.9238731 , -2.17872264, -0.17299356,  0.37281589],
        [-0.09743098, -0.17689769,  0.29176532, -1.282326 , -0.67244174],
        [-0.62478187,  0.24169548, -0.98143442, -0.56389493,  1.15780524]],

       [[ 1.47318384,  0.96500145,  0.80124135, -0.85210635,  0.25740846],
        [ 1.10173455,  0.65815256,  0.10804733, -1.02128938, -0.52060164],
        [-0.86936026,  0.11153553, -0.30380139, -0.36047905,  0.35520664],
        [ 1.27520826,  1.0598634 , -1.5327553 , -0.55896011,  1.92053426]]])
```

```
In [10]: b = np.random.randint(100, 200, (3,4))
```

```
In [11]: b
```

```
Out[11]:
```

```
array([[195, 192, 173, 161],
       [104, 190, 110, 126],
       [198, 122, 191, 178]])
```

函数小试

```
In [10]: b = np.random.randint(100, 200, (3,4))
```

```
In [11]: b
```

```
Out[11]:
```

```
array([[195, 192, 173, 161],  
       [104, 190, 110, 126],  
       [198, 122, 191, 178]])
```

```
In [12]: np.random.seed(10)
```

```
In [13]: np.random.randint(100, 200, (3,4))
```

```
Out[13]:
```

```
array([[109, 115, 164, 128],  
       [189, 193, 129, 108],  
       [173, 100, 140, 136]])
```

```
In [14]: np.random.seed(10)
```

```
In [15]: np.random.randint(100, 200, (3,4))
```

```
Out[15]:
```

```
array([[109, 115, 164, 128],  
       [189, 193, 129, 108],  
       [173, 100, 140, 136]])
```

np.random的随机数函数(2)

函数	说明
<code>shuffle(a)</code>	根据数组a的第1轴进行随排列，改变数组x
<code>permutation(a)</code>	根据数组a的第1轴产生一个新的乱序数组，不改变数组x
<code>choice(a[,size,replace,p])</code>	从一维数组a中以概率p抽取元素，形成size形状新数组 replace表示是否可以重用元素，默认为False

函数小试

```
In [18]: import numpy as np
```

```
In [19]: a = np.random.randint(100, 200, (3,4))
```

```
In [20]: a
```

```
Out[20]:  
array([[116, 111, 154, 188],  
       [162, 133, 172, 178],  
       [149, 151, 154, 177]])
```

```
In [21]: np.random.shuffle(a)
```

```
In [22]: a
```

```
Out[22]:  
array([[116, 111, 154, 188],  
       [149, 151, 154, 177],  
       [162, 133, 172, 178]])
```

```
In [23]: np.random.shuffle(a)
```

```
In [24]: a
```

```
Out[24]:  
array([[162, 133, 172, 178],  
       [116, 111, 154, 188],  
       [149, 151, 154, 177]])
```

```
In [35]: import numpy as np
```

```
In [36]: a = np.random.randint(100, 200, (3,4))
```

```
In [37]: a
```

```
Out[37]:  
array([[117, 146, 107, 175],  
       [128, 133, 184, 196],  
       [188, 144, 105, 104]])
```

```
In [38]: np.random.permutation(a)
```

```
Out[38]:  
array([[128, 133, 184, 196],  
       [188, 144, 105, 104],  
       [117, 146, 107, 175]])
```

```
In [39]: a
```

```
Out[39]:  
array([[117, 146, 107, 175],  
       [128, 133, 184, 196],  
       [188, 144, 105, 104]])
```

a没有变化

函数小试

```
In [54]: import numpy as np
```

```
In [55]: b = np.random.randint(100, 200, (8,))
```

```
In [56]: b
```

```
Out[56]: array([193, 175, 186, 137, 111, 121, 133, 195])
```

```
In [57]: np.random.choice(b, (3,2))
```

```
Out[57]:  
array([[137, 193],  
       [193, 121],  
       [175, 193]])
```

```
In [58]: np.random.choice(b, (3,2), replace=False)
```

```
Out[58]:  
array([[111, 175],  
       [193, 195],  
       [186, 133]])
```

```
In [61]: np.random.choice(b, (3,2), p= b/np.sum(b))
```

```
Out[61]:  
array([[121, 175],  
       [193, 186],  
       [193, 175]])
```


np.random的随机数函数(3)

函数	说明
<code>uniform(low,high,size)</code>	产生具有均匀分布的数组,low起始值,high结束值,size形状
<code>normal(loc,scale,size)</code>	产生具有正态分布的数组,loc均值,scale标准差,size形状
<code>poisson(lam,size)</code>	产生具有泊松分布的数组,lam随机事件发生率,size形状

函数小试

```
In [11]: import numpy as np
```

```
In [12]: u = np.random.uniform(0, 10, (3,4))
```

```
In [13]: u
```

```
Out[13]:
```

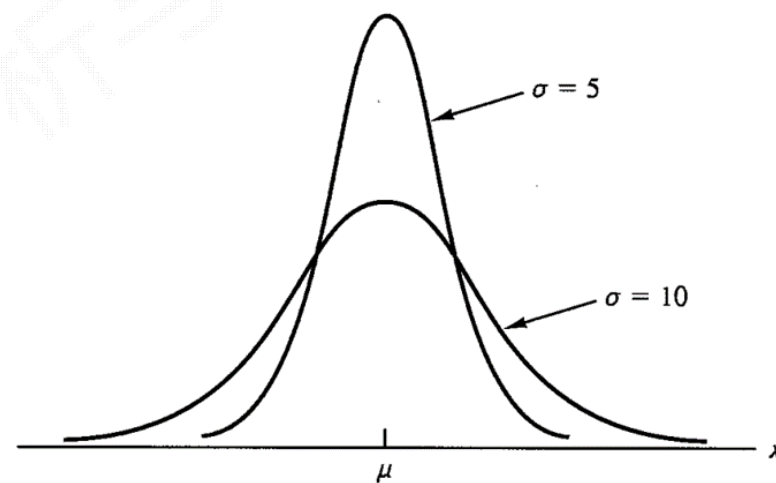
```
array([[ 2.05586231,  2.17210729,  8.23705995,  0.47651788],  
       [ 6.95560704,  6.79999967,  8.48924182,  4.39112863],  
       [ 4.25512913,  5.32043243,  5.28020392,  4.88650053]])
```

```
In [14]: n = np.random.normal(10, 5, (3,4))
```

```
In [15]: n
```

```
Out[15]:
```

```
array([[ 7.99574876,  5.02757645, 11.97332585,  5.48852456],  
       [12.45906894, 19.42856317, 19.11978806, 13.16322858],  
       [10.43019671,  8.46496502,  8.61987727, 15.6517832 ]])
```



正态分布



NumPy的统计函数

NumPy的统计函数

NumPy直接提供的统计类函数

`np.*`

`np.std()`

`np.var()`

`np.average()`

NumPy的统计函数(1)

函数	说明
<code>sum(a, axis=None)</code>	根据给定轴axis计算数组a相关元素之和，axis整数或元组
<code>mean(a, axis=None)</code>	根据给定轴axis计算数组a相关元素的期望，axis整数或元组
<code>average(a,axis=None,weights=None)</code>	根据给定轴axis计算数组a相关元素的加权平均值
<code>std(a, axis=None)</code>	根据给定轴axis计算数组a相关元素的标准差
<code>var(a, axis=None)</code>	根据给定轴axis计算数组a相关元素的方差

`axis=None` 是统计函数的标配参数

函数小试

```
In [20]: import numpy as np
```

```
In [21]: a = np.arange(15).reshape(3, 5)
```

```
In [22]: a
```

```
Out[22]:  
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
In [23]: np.sum(a)
```

```
Out[23]: 105
```

```
In [24]: np.mean(a, axis=1)
```

```
Out[24]: array([ 2.,  7., 12.])
```

```
In [25]: np.mean(a, axis=0)
```

```
Out[25]: array([ 5.,  6.,  7.,  8.,  9.])
```

加权平均

```
In [26]: np.average(a, axis=0, weights=[10, 5, 1])
```

```
Out[26]: array([ 2.1875,  3.1875, 4.1875,  5.1875,  6.1875])
```

```
In [27]: np.std(a)
```

```
Out[27]: 4.3204937989385739
```

$$2*10+7*5+1*12/(10+5+1)=4.1875$$

```
In [28]: np.var(a)
```

```
Out[28]: 18.666666666666668
```

NumPy的统计函数(2)

函数	说明
<code>min(a)</code> <code>max(a)</code>	计算数组a中元素的最小值、最大值
<code>argmin(a)</code> <code>argmax(a)</code>	计算数组a中元素最小值、最大值的降一维后下标
<code>unravel_index(index, shape)</code>	根据shape将一维下标index转换成多维下标
<code>ptp(a)</code>	计算数组a中元素最大值与最小值的差
<code>median(a)</code>	计算数组a中元素的中位数（中值）

函数小试

```
In [34]: import numpy as np
```

```
In [35]: b = np.arange(15,0,-1).reshape(3, 5)
```

```
In [36]: b
```

```
Out[36]:
```

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

```
In [37]: np.max(b)
```

```
Out[37]: 15
```

```
In [38]: np.argmax(b)
```

```
Out[38]: 0
```

扁平化后的下标

```
In [39]: np.unravel_index(np.argmax(b), b.shape)
```

```
Out[39]: (0, 0)
```

重塑成多维下标

```
In [40]: np.ptp(b)
```

```
Out[40]: 14
```

```
In [41]: np.median(b)
```

```
Out[41]: 8.0
```




NumPy的梯度函数



NumPy的梯度函数

函数	说明
<code>np.gradient(f)</code>	计算数组f中元素的梯度，当f为多维时，返回每个维度梯度

梯度：连续值之间的变化率，即斜率

XY坐标轴连续三个X坐标对应的Y轴值：a，b，c，其中，b的梯度是： $(c-a)/2$

函数小试

```
In [49]: import numpy as np
```

```
In [50]: a = np.random.randint(0, 20, (5))
```

```
In [51]: a
```

```
Out[51]: array([15,  3, 12, 13, 14])
```

存在两侧值： $(12-15)/2$

```
In [52]: np.gradient(a)
```

```
Out[52]: array([-12. , -1.5,  5. ,  1. ,  1. ])
```

→ 只有一侧值： $(14-13)/1$

```
In [53]: b = np.random.randint(0, 20, (5))
```

```
In [54]: b
```

```
Out[54]: array([5, 7, 6, 1, 9])
```

```
In [55]: np.gradient(b)
```

```
Out[55]: array([ 2. ,  0.5, -3. ,  1.5,  8. ])
```

函数小试

```
In [56]: import numpy as np
```

```
In [57]: c = np.random.randint(0, 50, (3, 5))
```

```
In [58]: c
```

```
Out[58]:
```

```
array([[18, 49,  1,  5, 26],
       [40, 38, 39, 46, 47],
       [46, 23, 16, 31, 36]])
```

```
In [59]: np.gradient(c)
```

```
Out[59]:
```

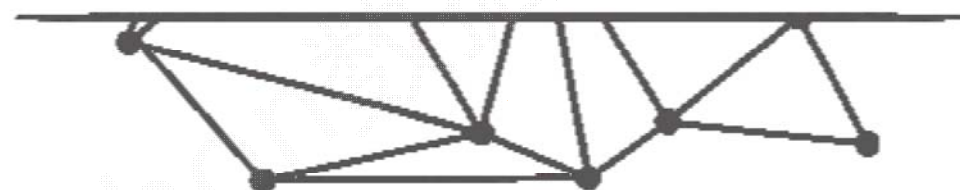
```
[array([[ 22. , -11. ,  38. ,  41. ,  21. ],
       [ 14. , -13. ,   7.5,  13. ,   5. ],
       [  6. , -15. , -23. , -15. , -11. ]]),
 array([[ 31. , -8.5, -22. , 12.5,  21. ],
       [ -2. , -0.5,  4. ,  4. ,  1. ],
       [-23. , -15. ,  4. , 10. ,  5. ]])]
```

最外层维度的梯度

第二层维度的梯度



单元小结



数据存取与函数

CSV文件

`np.loadtxt()`

`np.savetxt()`

多维数据存取

`a.tofile()` `np.fromfile()`

`np.save()` `np.savez()` `np.load()`

随机函数

`np.random.rand()`

`np.random.randn()`

`np.random.randint()`

`np.random.seed()`

`np.random.shuffle()`

`np.random.permutation()`

`np.random.choice()`

数据存取与函数

NumPy的统计函数

`np.sum()`

`np.mean()`

`np.average()`

`np.std()`

`np.var()`

`np.median()`

`np.min()`

`np.max()`

`np.argmin()`

`np.argmax()`

`np.unravel_index()`

`np.ptp()`

NumPy的梯度函数

`np.gradient()`