

Python数据分析

(5)

殷传涛 教授

chuantao.yin@buaa.edu.cn

库

- 常用的第三方库（其他机构开发）：

数据分析

numpy: <http://www.numpy.org/> 开源数值计算扩展第三方库

scipy: <https://pypi.org/project/scipy/> 专为科学以及工程计算的第三方库

pandas: <http://pandas.pydata.org/> 可高效地操作大型数据集的第三方库

网络爬虫

requests: <https://pypi.org/project/requests/> 简洁且简单的处理HTTP请求的第三方库

scrapy: <https://scrapy.org/> 快速、高层次的Web获取框架

文本处理

pdfminer: <https://pypi.org/project/pdfminer/> 从PDF文档中提取各类信息的第三方库

openpyxl: <https://pypi.org/project/openpyxl/> 处理Microsoft Excel文档的Python第三方库

python-docx: <https://pypi.org/project/python-docx/> 处理Microsoft Word文档的Python第三方库

beautifulsoup4: <https://pypi.org/project/beautifulsoup4/> 从HTML和XML文件中解析出数据的第三方库

NumPy介绍

[Install](#) [Documentation](#) [Learn](#) [Community](#) [About Us](#) [News](#) [Contribute](#) [English](#) ▾



The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 1.26. [VIEW ALL RELEASES](#)

NumPy 2.0 release date: June 16 [2024-05-23](#)

Powerful N-dimensional arrays

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

Numerical computing tools

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Open source

Distributed under a liberal [BSD license](#), NumPy is developed and maintained [publicly on GitHub](#) by a vibrant, responsive, and diverse [community](#).

Interoperable

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

Performant

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

Easy to use

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

NumPy介绍

▶ NumPy (Numerical Python) 的特点

- NumPy的核心是`ndarray`对象。封装了同类数据类型的`n`维数组。
- Python中用列表(list)保存一组值，可以用来当作数组使用，不过由于列表的元素可以是任何对象，因此列表中所保存的是对象的指针。这样为了保存一个简单的[1,2,3]，需要有3个指针和三个整数对象。对于数值运算来说这种结构显然比较浪费内存和CPU计算时间。
- Python还提供了一个array模块，array对象和列表不同，它直接保存数值，和C语言的一维数组比较类似。但是由于它不支持多维，也没有各种运算函数，因此也不适合做数值运算。
- NumPy数组在创建时具有固定的大小，这与Python列表（可以动态增长）不同。
- NumPy数组中的所有元素都必须具有相同的数据类型，因此在内存中的大小将相同。
- NumPy数组有助于对大量数据进行高级数学运算和其他类型的运算。与使用Python的内置序列相比，可以更高效地执行，并且代码更少。

安装NumPy

- ▶ 在Python中安装和使用NumPy

```
pip install numpy
```

```
import numpy as np
```

- ▶ 直接下载和安装Python的集成安装包Anaconda，包含了我们所有的工具
 - Numpy, pandas, matplotlib, tensorflow, jupyternotebook等
 - <https://www.anaconda.com/products/individual>

Anaconda科学包



Data Exploration and Transformation

- pandas
- Intake
- NumPy
- Dask
- Apache Airflow



Visualization

- Matplotlib
- seaborn
- Plotly
- Bokeh
- HoloViews



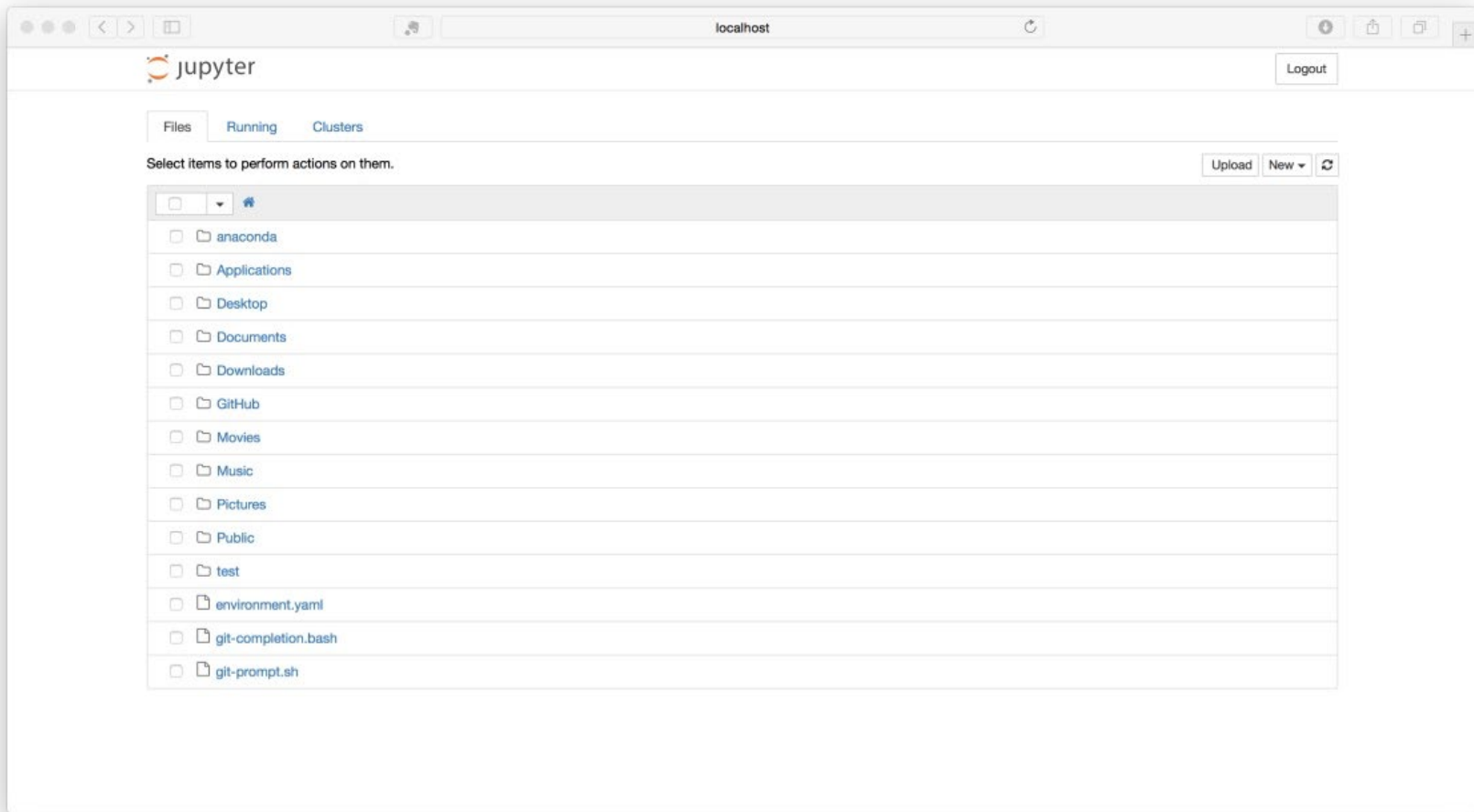
AI and Machine Learning

- scikit-learn
- TensorFlow
- Keras
- PyTorch
- XGBoost


Jupyter Notebook

- ▶ 开发环境
 - Python shell
 - IDE: Pycharm, Eclipse等
- ▶ Jupyter Notebook 开发环境适合数据分析，程序、结果、说明在同一个窗口中。
 - 支持多语言：Python, R, Matlab等
 - 分享便捷：支持以网页的形式分享远程运行
 - 交互式展现：输出图片、视频、数学公式，甚至可以呈现一些互动的可视化内容

Jupyter Notebook



Jupyter Notebook

jupyter Untitled1 最后检查: 9 分钟前 (未保存改变)  Logout


File Edit View Insert Cell Kernel Widgets Help 可信的 Python 3

运行 代码

```
In [1]: print("hello")
hello

In [ ]:

In [ ]:
```

jupyter ch-02 最后检查: 17 小时前 (未保存改变)  Logout

File Edit View Insert Cell Kernel Widgets Help 可信的 Python 3

运行 Markdown

numpy arrays

```
In [2]: a = np.arange(5)

In [3]: a.dtype
Out[3]: dtype('int32')
```

NumPy数组

- ▶ 必须先导入numpy
 - `import numpy as np`（通常都用np来代替numpy）
- ▶ 创建一维数组
 - 创建函数 `array()`
 - `arange()`函数，返回等差一维数组
 - 查看数组的数据类型`dtype`
 - 查看数组的形状

```
In [12]: a=np.array([0,1,2,3,4])  
         b=np.arange(5)
```

```
In [13]: a.dtype  
         b.dtype
```

```
Out[13]: dtype('int32')
```

```
In [14]: print(a)  
         print(b)
```

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

```
In [15]: print(a.shape)  
         print(b.shape)
```

```
(5,)  
(5,)
```

NumPy数组

- ▶ 创建多维数组
 - 二维数组（矩阵）
 - 查看形状

```
In [8]: m = np.array([np.arange(2), np.arange(2)])
```

```
In [9]: m
```

```
Out[9]: array([[0, 1],  
              [0, 1]])
```

```
In [10]: m.shape
```

```
Out[10]: (2, 2)
```

Numpy中的主要函数

```
In [33]: import numpy as np
         dir(np)
```

```
'amax',
'amin',
'angle',
'any',
'append',
'apply_along_axis',
'apply_over_axes',
'arange',
'arccos',
'arccosh',
'arcsin',
'arcsinh',
'arctan',
'arctan2',
'arctanh',
'argmax',
'argmin',
'argpartition',
'argsort',
```

```
In [34]: help("numpy.arange")
```

Help on built-in function arange in numpy:

```
numpy.arange = arange(...)
arange([start,] stop[, step,], dtype=None)
```

Return evenly spaced values within a given interval.

Values are generated within the half-open interval ``[start, stop)`` (in other words, the interval including ``start`` but excluding ``stop``). For integer arguments the function is equivalent to the Python built-in ``range`` function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use ``numpy.linspace`` for these cases.

Parameters

start : number, optional
Start of interval. The interval includes this value. The default start value is 0.

stop : number
End of interval. The interval does not include this value, except in some cases where ``step`` is not an integer and floating point round-off affects the length of ``out``.

step : number, optional
Spacing between values. For any output ``out``, this is the distance between two adjacent values, ``out[i+1] - out[i]``. The default step size is 1. If ``step`` is specified as a position argument, ``start`` must also be given.

dtype : dtype
The type of the output array. If ``dtype`` is not given, infer the data type from the other input arguments.

Returns

arange : ndarray
Array of evenly spaced values.

NumPy数组的数值类型

Numpy 的类型	描述
np.int8	字节 (-128到127)
np.int16	整数 (-32768至32767)
np.int32	整数 (-2147483648至2147483647)
np.int64	整数 (-9223372036854775808至9223372036854775807)
np.uint8	无符号整数 (0到255)
np.uint16	无符号整数 (0到65535)
np.uint32	无符号整数 (0到4294967295)
np.uint64	无符号整数 (0到18446744073709551615)
np.intp	用于索引的整数, 通常与索引相同 ssize_t
np.uintp	整数大到足以容纳指针
np.float32	
np.float64 / np.float_	请注意, 这与内置python float的精度相匹配。
np.complex64	复数, 由两个32位浮点数 (实数和虚数组件) 表示
np.complex128 / np.complex_	请注意, 这与内置python 复合体的精度相匹配。

数值类型的转化及定义

► 转化类型

► 定义数值类型

```
In [17]: np.float64(42)
```

```
Out[17]: 42.0
```

```
In [18]: np.int8(42.0)
```

```
Out[18]: 42
```

```
In [19]: np.bool(42)
```

```
Out[19]: True
```

```
In [20]: np.bool(0)
```

```
Out[20]: False
```

```
In [21]: np.bool(42.0)
```

```
Out[21]: True
```

```
In [22]: np.float(True)
```

```
Out[22]: 1.0
```

```
In [23]: np.float(False)
```

```
Out[23]: 0.0
```

```
In [24]: np.arange(7, dtype=np.uint16)
```

```
Out[24]: array([0, 1, 2, 3, 4, 5, 6], dtype=uint16)
```

NumPy数组

- ▶ 一维数组的索引和切片
 - 和列表、字符串等序列型数据一样

```
>>> a = np.arange(10)**3
>>> a
array([ 0,  1,  8, 27, 64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; from start to position 6, exclusive, set even
>>> a
array([-1000,  1, -1000,  27, -1000,  125,  216,  343,  512,  729])
>>> a[::-1]           # reversed a
array([ 729,  512,  343,  216,  125, -1000,  27, -1000,  1, -1000])
```

NumPy数组

- ▶ 多维数组中元素的索引
 - 按照元素在数组中的位置
 - 每一维度从0到n-1

```
In [11]: a = np.array([[1, 2], [3, 4]])
```

```
In [12]: a
```

```
Out[12]: array([[1, 2],  
               [3, 4]])
```

```
In [13]: a[0,0]
```

```
Out[13]: 1
```

```
In [14]: a[0,1]
```

```
Out[14]: 2
```

```
In [15]: a[1,0]
```

```
Out[15]: 3
```

```
In [16]: a[1,1]
```

```
Out[16]: 4
```


数组形状

► 改变数组的形状

► 多维变一维

- `ravel()`
- `flatten()`

► 多维变多维

- `resize()`
- `reshape()`函数

```
In [26]: b = np.arange(24).reshape(2, 3, 4)
```

```
In [27]: print(b)

[[[ 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 [28]: print(b.ravel())

[ 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 [29]: print(b.flatten())

[ 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 [66]: b.shape = (6, 4)
```

```
In [67]: print(b)

[[ 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 [33]: b.resize((2, 12))
```

```
In [34]: print(b)

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

NumPy数组

► 转置 transpose()

```
In [67]: print(b)
```

```
[[ 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 [32]: print(b.transpose())
```

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

堆叠数组

▶ 两个数组相叠加

▶ 水平叠加

- `hstack()`
- `concatenate()`

```
In [68]: a = np.arange(9).reshape(3,3)
```

```
In [69]: a
```

```
Out[69]: array([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])
```

```
In [70]: b = 2 * a
```

```
In [71]: b
```

```
Out[71]: array([[ 0,  2,  4],
                [ 6,  8, 10],
                [12, 14, 16]])
```

```
In [72]: np.hstack((a, b))
```

```
Out[72]: array([[ 0,  1,  2,  0,  2,  4],
                [ 3,  4,  5,  6,  8, 10],
                [ 6,  7,  8, 12, 14, 16]])
```

```
In [73]: np.concatenate((a, b), axis=1)
```

```
Out[73]: array([[ 0,  1,  2,  0,  2,  4],
                [ 3,  4,  5,  6,  8, 10],
                [ 6,  7,  8, 12, 14, 16]])
```

堆叠数组

▶ 垂直叠加

- `vstack()`
- `concatenate()`

```
In [74]: np.vstack((a, b))
```

```
Out[74]: array([[ 0,  1,  2],  
                [ 3,  4,  5],  
                [ 6,  7,  8],  
                [ 0,  2,  4],  
                [ 6,  8, 10],  
                [12, 14, 16]])
```

```
In [75]: np.concatenate((a, b), axis=0)
```

```
Out[75]: array([[ 0,  1,  2],  
                [ 3,  4,  5],  
                [ 6,  7,  8],  
                [ 0,  2,  4],  
                [ 6,  8, 10],  
                [12, 14, 16]])
```

拆分数组

- ▶ 横向拆分
 - `hsplit()`
 - `split()`

```
In [76]: a
```

```
Out[76]: array([[0, 1, 2],  
               [3, 4, 5],  
               [6, 7, 8]])
```

```
In [77]: np.hsplit(a, 3)
```

```
Out[77]: [array([[0],  
                [3],  
                [6]]),  
          array([[1],  
                [4],  
                [7]]),  
          array([[2],  
                [5],  
                [8]])]
```

```
In [78]: np.split(a, 3, axis=1)
```

```
Out[78]: [array([[0],  
                [3],  
                [6]]),  
          array([[1],  
                [4],  
                [7]]),  
          array([[2],  
                [5],  
                [8]])]
```

拆分数组

► 纵向拆分

- vsplit()
- split()

```
In [79]: np.vsplit(a, 3)
```

```
Out[79]: [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

```
In [80]: np.split(a, 3, axis=0)
```

```
Out[80]: [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]
```

数组的属性

- ▶ 形状 shape
- ▶ 数据类型 dtype
- ▶ 维度ndim
- ▶ 元素数量size
- ▶ 元素的字节数itemsize
- ▶ 数组的总字节数nbytes
- ▶ 转置T

```
In [7]: b = np.arange(24).reshape(2, 12)
```

```
In [8]: b
```

```
Out[8]: 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 [9]: b.shape
```

```
Out[9]: (2, 12)
```

```
In [10]: b.dtype
```

```
Out[10]: dtype('int32')
```

```
In [11]: b.ndim
```

```
Out[11]: 2
```

```
In [12]: b.size
```

```
Out[12]: 24
```

```
In [13]: b.itemsize
```

```
Out[13]: 4
```

```
In [14]: b.nbytes
```

```
Out[14]: 96
```

```
In [15]: b.size * b.itemsize
```

```
Out[15]: 96
```

```
In [88]: b.resize(6,4)
```

```
In [89]: b
```

```
Out[89]: 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 [90]: b.T
```

```
Out[90]: array([[ 0,  4,  8, 12, 16, 20],  
               [ 1,  5,  9, 13, 17, 21],  
               [ 2,  6, 10, 14, 18, 22],  
               [ 3,  7, 11, 15, 19, 23]])
```

数组的属性

- ▶ 实部 real
- ▶ 虚部 imag

```
In [91]: b = np.array([1.j + 1, 2.j + 3])
```

```
In [92]: b
```

```
Out[92]: array([1.+1.j, 3.+2.j])
```

```
In [93]: b.real
```

```
Out[93]: array([1., 3.])
```

```
In [94]: b.imag
```

```
Out[94]: array([1., 2.])
```

```
In [95]: b.dtype
```

```
Out[95]: dtype('complex128')
```


数组与列表的互相转换

▶ 数组转化为列表

```
In [111]: b = np.array([1.j + 1, 2.j + 3])
```

```
In [112]: b.tolist()
```

```
Out[112]: [(1+1j), (3+2j)]
```

```
In [113]: b.astype(int)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\ipykernel  
art  
"""Entry point for launching an IPython kernel.
```

```
Out[113]: array([1, 3])
```

```
In [114]: b.astype('complex')
```

```
Out[114]: array([1.+1.j, 3.+2.j])
```

Numpy的高级索引

► Numpy的索引和切片

◦ 正常的切片索引

- `a[start:stop:step]`
- 注意冒号的使用
- 可省略

◦ 省略号的使用

```
In [25]: import numpy as np
a = np.arange(10)
print(a)
print(a[5])
print(a[2:5])
print(a[:8])
print(a[1:9:2])
print(a[::-1])
```

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

```
In [26]: a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a.shape)
print(a[...,1]) # 第2列元素
print(a[1,...]) # 第2行元素
print(a[...,1:]) # 第2列及剩下的所有元素
print(a[1:,...]) # 第2行及剩下的所有元素
```

```
(3, 3)
[2 4 5]
[3 4 5]
[[2 3]
 [4 5]
 [5 6]]
[[3 4 5]
 [4 5 6]]
```

Numpy的高级索引

► 综合使用冒号和省略号进行索引

```
import numpy as np

a = np.arange(24).reshape(4,6)
print("a数组\n", a)

b = a[0:3, 2:4]
c = a[0:3, [1, 2]]
d = a[..., 2:]
print("b数组\n", b)
print("c数组\n", c)
print("d数组", d)
```

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

b数组
[[ 2  3]
 [ 8  9]
 [14 15]]

c数组
[[ 1  2]
 [ 7  8]
 [13 14]]

d数组
[[ 2  3  4  5]
 [ 8  9 10 11]
 [14 15 16 17]
 [20 21 22 23]]
```

Numpy的高级索引

▶ 用数组进行索引

- 生成数组的格式和索引号数组的格式相同

```
import numpy as np
x = np.array([[1, 2], [3, 4], [5, 6]])
print(x.shape)
#索引号为1*3的一维数组
print(x[2,1])           #取数组中(2,1)位置处的元素
print(x[[0,1,2], [0,1,0]]) # 取数组中(0,0), (1,1)和(2,0)位置处的元素
```

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

```
import numpy as np

x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
print (x)
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
#索引号为2*2的二维数组
y = x[rows,cols]
print ('第[0,0]、[0,2]、[3,0]、[3,2]位置上的元素')
print (y)
```

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

第[0,0]、[0,2]、[3,0]、[3,2]位置上的元素

```
[[ 0  2]
 [ 9 11]]
```

Numpy的高级索引

布尔索引

- 使用布尔表达式或条件筛选元素

```
import numpy as np

x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
print('我们的数组是:')
print(x)
print('\n')
# 现在我们会打印出大于 5 的元素
print('大于 5 的元素是:')
print(x[x>5])
```

我们的数组是:

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

大于 5 的元素是:

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

```
import numpy as np

a = np.array([1, 2+6j, 5, 3.5+5j])
print(a[np.iscomplex(a)])
```

```
[2. +6.j 3.5+5.j]
```

```
import numpy as np

a = np.array([np.nan, 1, 2, np.nan, 3, 4, 5])
# np.nan为numpy数组中的空值
print(a)
print(a[~np.isnan(a)])
# ~ 取补运算符
```

```
[nan 1.  2. nan 3.  4.  5.]
[1. 2. 3. 4. 5.]
```

Numpy的高级索引

- ▶ 花式索引：用一维数组作为索引
 - 对于一维数组，索引对应位置元素
 - 对于二维数组，索引对应行的元素

```
import numpy as np

x=np.arange(32)*2
print(x, "\n")
print (x[[4,2,1,7]])
#用一维数组[4, 2, 1, 7]做为索引
```

```
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
 48 50 52 54 56 58 60 62]
```

```
[ 8  4  2 14]
```

```
import numpy as np
```

```
x=np.arange(32).reshape((8,4))
print(x, "\n")
print (x[[4,2,1,7]])
#用一维数组[4, 2, 1, 7]做为索引
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]]
```

```
[[16 17 18 19]
 [ 8  9 10 11]
 [ 4  5  6  7]
 [28 29 30 31]]
```

Numpy的高级索引

▶ 花式索引

- 用多维数组索引
- 用`np.ix_()`函数生成一个 4×4 的索引矩阵

```
x[1,0] x[1,3] x[1,1] x[1,2]  
x[5,0] x[5,3] x[5,1] x[5,2]  
x[7,0] x[7,3] x[7,1] x[7,2]  
x[2,0] x[2,3] x[2,1] x[2,2]
```

```
: import numpy as np  
  
x=np.arange(32).reshape((8,4))  
print(x, "\n")  
a=np.ix_([1,5,7,2], [0,3,1,2])  
print(a, "\n")  
print(x[a], "\n")  
print (x[[1,5,7,2], [0,3,1,2]])
```

```
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]  
 [12 13 14 15]  
 [16 17 18 19]  
 [20 21 22 23]  
 [24 25 26 27]  
 [28 29 30 31]]
```

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

```
[[ 4  7  5  6]  
 [20 23 21 22]  
 [28 31 29 30]  
 [ 8 11  9 10]]
```

```
[ 4 23 29 10]
```


Numpy的高级索引

```
import numpy as np
import scipy.misc
import matplotlib.pyplot as plt
```

```
face = np.array(scipy.misc.face())
#导入数据集
```

```
print(face.shape)
plt.imshow(face)
plt.show()
#显示数据集
```

```
xmax = face.shape[0] #x轴上的像素数
ymax = face.shape[1] #y轴上的像素数
```

```
face = face[:min(xmax, ymax), :min(xmax, ymax)]
#把face图片切割为方正的图片 (用xmax和ymax中较小的那个)
```

```
xmax = face.shape[0]
ymax = face.shape[1]
print(face.shape)
```

```
face[range(xmax), range(ymax)] = 0
face[range(xmax-1, -1, -1), range(ymax)] = 0
```

```
plt.imshow(face)
plt.show()
```

```
print(face) #RGB格式存储的图片数据
```

(768, 1024, 3)

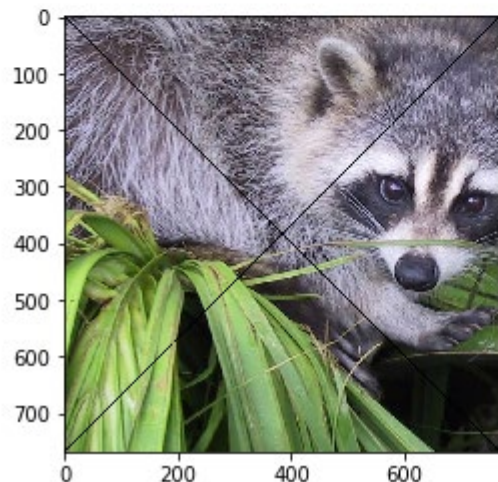


```
[[[ 0  0  0]
 [138 129 148]
 [153 144 165]
 ...
 [ 65  66  84]
 [ 67  68  86]
 [ 0  0  0]]]
```

```
[[ 89  82 100]
 [ 0  0  0]
 [130 122 143]
```

```
...
 [ 67  68  88]
 [ 0  0  0]
 [ 69  70  90]]]
```

(768, 768, 3)



```
[[[ 73  66  84]
 [ 94  87 105]
 [ 0  0  0]
```

```
...
 [ 0  0  0]
 [ 69  70  90]
 [ 68  69  89]]]
```


Numpy中的数学运算

► 数组的加、减、乘、除

```
import numpy as np
```

```
a = np.arange(9, dtype = np.float_).reshape(3,3)
```

```
print ('第一个数组:')
```

```
print (a)
```

```
print ('第二个数组:')
```

```
b = a*2+1
```

```
print (b)
```

```
print ('两个数组相加:')
```

```
print (a+b)
```

```
print (np.add(a,b))
```

```
print ('\n')
```

```
print ('两个数组相减:')
```

```
print (a-b)
```

```
print (np.subtract(a,b))
```

```
print ('\n')
```

```
print ('两个数组相乘:')
```

```
print (a*b)
```

```
print (np.multiply(a,b))
```

```
print ('\n')
```

```
print ('两个数组相除:')
```

```
print (a/b)
```

```
print (np.divide(a,b))
```

第一个数组:

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

第二个数组:

```
[[1. 3. 5.]  
 [7. 9. 11.]  
 [13. 15. 17.]]
```

两个数组相加:

```
[[1. 4. 7.]  
 [10. 13. 16.]  
 [19. 22. 25.]]  
[[1. 4. 7.]  
 [10. 13. 16.]  
 [19. 22. 25.]]
```

两个数组相减:

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

两个数组相乘:

```
[[ 0.   3.  10.]  
 [ 21.  36.  55.]  
 [ 78. 105. 136.]]  
[[ 0.   3.  10.]  
 [ 21.  36.  55.]  
 [ 78. 105. 136.]]
```

两个数组相除:

```
[[0.          0.33333333 0.4          ]  
 [0.42857143 0.44444444 0.45454545]  
 [0.46153846 0.46666667 0.47058824]]  
[[0.          0.33333333 0.4          ]  
 [0.42857143 0.44444444 0.45454545]  
 [0.46153846 0.46666667 0.47058824]]
```

Numpy中的数学运算

▶ 倒数、幂次方、取余

```
import numpy as np
```

```
a = np.array([0.25, 1.25, 1, 100])
```

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

```
print(np.reciprocal(a)) #求倒数
```

```
print(np.power(a, 2)) #求幂次方
```

```
print(np.power(a, b)) #求幂次方
```

```
[4.  0.8  1.  0.01]
```

```
[6.2500e-02 1.5625e+00 1.0000e+00 1.0000e+04]
```

```
[6.250000e-02 1.953125e+00 1.000000e+00 1.000000e+02]
```

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
b = np.array([3, 5, 7])
```

```
print('第一个数组:')
```

```
print(a)
```

```
print('\n')
```

```
print('第二个数组:')
```

```
print(b)
```

```
print('\n')
```

```
print('调用 mod() 函数:')
```

```
print(np.mod(a, b))
```

```
print('\n')
```

```
print('调用 remainder() 函数:')
```

```
print(np.remainder(a, b))
```

第一个数组:

```
[10 20 30]
```

第二个数组:

```
[3 5 7]
```

调用 mod() 函数:

```
[1 0 2]
```

调用 remainder() 函数:

```
[1 0 2]
```

Numpy中的数学运算

- 广播：两个数组计算时，理论上应该形状相同。当两个数组形状不同时，会自动触发广播机制，numpy会进行维度扩展，以完成计算

```
: import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])
print(a + b, "\n")
print(a*b, "\n")

a = np.array([[ 0,  0,  0],
               [10, 10, 10],
               [20, 20, 20],
               [30, 30, 30]])
b = np.array([1, 2, 3])
print(a + b, "\n")
print(a*b, "\n")
```

[11 22 33 44]

[10 40 90 160]

[[1 2 3]

[11 12 13]

[21 22 23]

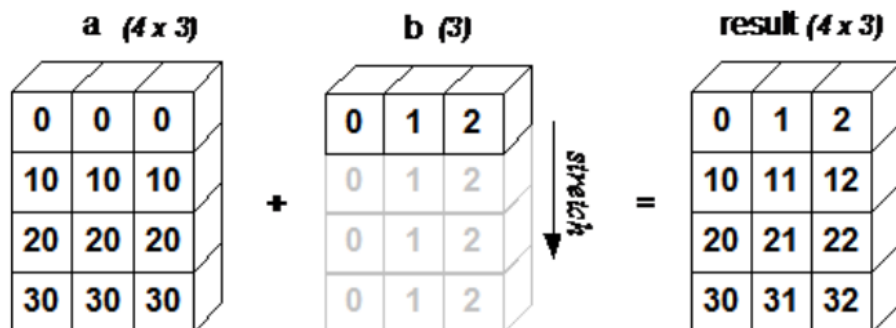
[31 32 33]]

[[0 0 0]

[10 20 30]

[20 40 60]

[30 60 90]]



Numpy中的数学运算

► 广播

```
import scipy.io.wavfile as sw
import matplotlib.pyplot as plt
import urllib
import numpy as np

request = urllib.request.Request('http://www.thesoundarchive.com/
response = urllib.request.urlopen(request)
print(response.info())
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'wb')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = sw.read(WAV_FILE)
print("Data type", data.dtype, "Shape", data.shape)

plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)

newdata = data * 0.2
newdata = newdata.astype(np.uint8)
print("Data type", newdata.dtype, "Shape", newdata.shape)

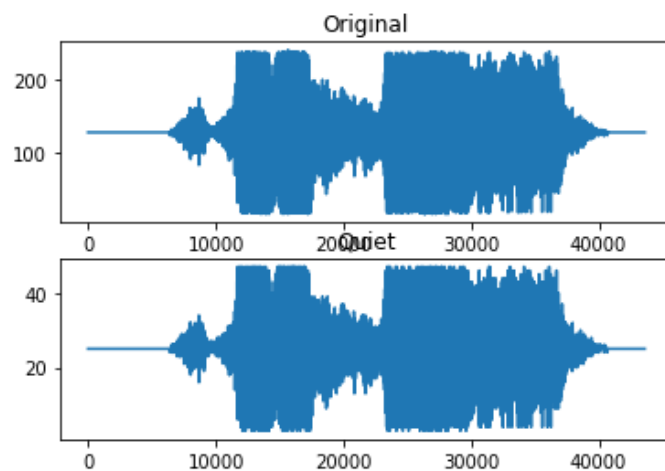
sw.write("quiet.wav",
        sample_rate, newdata)

plt.subplot(2, 1, 2)
plt.title("Quiet")
plt.plot(newdata)

plt.show()
```

Content-Type: audio/wav
Last-Modified: Tue, 12 Aug 2014 15:53:38 GMT
Accept-Ranges: bytes
ETag: "ac20ba9445b6cf1:0"
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Date: Sat, 06 Jun 2020 11:57:59 GMT
Connection: close
Content-Length: 43642

Data type uint8 Shape (43584,)
Data type uint8 Shape (43584,)



Numpy中的统计函数

- ▶ 最小值与最大值
- ▶ `amin()` 与 `amax()`

```
import numpy as np

a = np.array([[3, 7, 5], [8, 4, 3], [2, 4, 9]])
print('我们的数组是:')
print(a)

print('在水平方向上找最小值:')
print(np.amin(a, 1))

print('在竖直方向上找最小值:')
print(np.amin(a, 0))

print('找最大值:')
print(np.amax(a))

print('在竖直方向上找最大值:')
print(np.amax(a, axis = 0))
```

我们的数组是：

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

在水平方向上找最小值：

```
[3 3 2]
```

在竖直方向上找最小值：

```
[2 4 3]
```

找最大值：

```
9
```

在竖直方向上找最大值：

```
[8 7 9]
```

Numpy中的统计函数

- ▶ 最大值与最小值的差值（最大值 - 最小值）
- ▶ `ptp()` 函数

```
import numpy as np

a = np.array([[1, 7, 5], [8, 4, 3], [2, 4, 9]])
print('我们的数组是：')
print(a)

print('调用 ptp() 函数：')
print(np.ptp(a))

print('沿水平方向 调用 ptp() 函数：')
print(np.ptp(a, axis = 1))

print('竖直方向调用 ptp() 函数：')
print(np.ptp(a, axis = 0))
```

我们的数组是：

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

调用 `ptp()` 函数：

8

沿水平方向 调用 `ptp()` 函数：

```
[6 5 7]
```

竖直方向调用 `ptp()` 函数：

```
[7 3 6]
```

Numpy中的统计函数

► 排序 sort()函数

sort(a, axis, kind, order)

```
import numpy as np

a = np.array([[9, 7, 5, 6], [3, 1, 6, 2]])
print ('我们的数组是：')
print (a)
print ('\n')
print ('调用 sort() 函数：')
print (np.sort(a))
print ('\n')
print ('按列排序：')
print (np.sort(a, axis = 0))
```

Numpy中的统计函数

- ▶ argsort() 函数
- ▶ 返回数组值从小到大的索引值

```
import numpy as np

x = np.array([[3, 1, 2], [5, 7, 6]])
print ('我们的数组是：')
print (x)
print ('\n')
print ('对 x 调用 argsort() 函数：')
y = np.argsort(x)
print (y)
```

我们的数组是：

```
[[3 1 2]
 [5 7 6]]
```

对 x 调用 argsort() 函数：

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


Numpy中的统计函数

- ▶ `argmax()` 和 `argmin()` 函数
- ▶ 返回最大值和最小值的索引

```
import numpy as np

a = np.array([[30, 40, 70], [80, 20, 10], [50, 90, 60], [60, 90, 100]])
print('我们的数组是:')
print(a)
print('\n')

print(np.argmax(a))
print(a.flatten())
print('沿轴 0 的最大值索引:')
print(np.argmax(a, axis = 0))
print('沿轴 1 的最大值索引:')
print(np.argmax(a, axis = 1))

print('\n')
print(np.argmin(a))
print(minindex)
print('沿轴 0 的最小值索引:')
print(np.argmin(a, axis = 0))
print('沿轴 1 的最小值索引:')
print(np.argmin(a, axis = 1))
```

我们的数组是:

```
[[ 30  40  70]
 [ 80  20  10]
 [ 50  90  60]
 [ 60  90 100]]
```

11

```
[ 30  40  70  80  20  10  50  90  60  60  90 100]
```

沿轴 0 的最大值索引:

```
[1 2 3]
```

沿轴 1 的最大值索引:

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

5

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

沿轴 0 的最小值索引:

```
[0 1 1]
```

沿轴 1 的最小值索引:

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

Numpy中的统计函数

- ▶ nonzero()函数
- ▶ 返回输入数组中非零元素的索引

```
import numpy as np

a = np.array([[30, 40, 0], [0, 20, 10], [50, 0, 60], [70, 0, 60]])
print ('我们的数组是：')
print (a)
print ('\n')
print ('调用 nonzero() 函数：')
print (np.nonzero (a))
```

我们的数组是：

```
[[30 40  0]
 [ 0 20 10]
 [50  0 60]
 [70  0 60]]
```

调用 nonzero() 函数：

```
(array([0, 0, 1, 1, 2, 2, 3, 3], dtype=int32), array([0, 1, 1, 2, 0, 2, 0, 2], dtype=int32))
```

Numpy中的统计函数

- ▶ where() 函数
- ▶ 返回输入数组中满足给定条件的元素的索引

```
import numpy as np

x = np.arange(9.).reshape(3, 3)
print('我们的数组是:')
print(x)
print('大于 3 的元素的索引:')
y = np.where(x > 3)
print(y)
print('使用这些索引来获取满足条件的元素:')
print(x[y])
```

我们的数组是:

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

大于 3 的元素的索引:

```
(array([1, 1, 2, 2, 2], dtype=int32), array([1, 2, 0, 1, 2], dtype=int32))
```

使用这些索引来获取满足条件的元素:

```
[4. 5. 6. 7. 8.]
```

Numpy中的统计函数

- ▶ `extract()` 函数
- ▶ 根据某个条件从数组中抽取元素，返回满条件的元素

```
import numpy as np

x = np.arange(9.).reshape(3, 3)
print('我们的数组是:')
print(x)
# 定义条件, 选择偶数元素
condition = np.mod(x, 2) == 0
print('按元素的条件值:')
print(condition)
print('使用条件提取元素:')
print(np.extract(condition, x))

condition = (x >= 4)
print('按元素的条件值:')
print(condition)
print('使用条件提取元素:')
print(np.extract(condition, x))
```

我们的数组是:

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

按元素的条件值:

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

使用条件提取元素:

```
[0. 2. 4. 6. 8.]
```

按元素的条件值:

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

使用条件提取元素:

```
[4. 5. 6. 7. 8.]
```

Numpy中的统计函数

- ▶ 算术平均值
- ▶ mean()函数

```
import numpy as np

a = np.array([[1, 2, 3], [3, 4, 5], [4, 5, 6]])
print('我们的数组是:')
print(a)

print('调用 mean() 函数:')
print(np.mean(a))

print('沿竖直方向调用 mean() 函数:')
print(np.mean(a, axis = 0))

print('沿水平方向调用 mean() 函数:')
print(np.mean(a, axis = 1))
```

我们的数组是：

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

调用 mean() 函数：

```
3.6666666666666665
```

沿竖直方向调用 mean() 函数：

```
[2.66666667 3.66666667 4.66666667]
```

沿水平方向调用 mean() 函数：

```
[2. 4. 5.]
```

Numpy中的统计函数

- ▶ 加权平均值
- ▶ average() 函数

```
import numpy as np

a = np.array([1, 2, 3, 4])
print('我们的数组是:')
print(a)
print('\n')
print('调用 average() 函数:')
print(np.average(a))
print('\n')
# 不指定权重时相当于 mean 函数
wts = np.array([99, 3, 2, 1])
print('再次调用 average() 函数:')
print(np.average(a, weights = wts))
print('\n')
# 如果 returned 参数设为 true, 则返回权重的和
print('权重的和:')
print(np.average([1, 2, 3, 4], weights = wts, returned = True))
```

我们的数组是:

[1 2 3 4]

调用 average() 函数:

2.5

再次调用 average() 函数:

1.0952380952380953

权重的和:

(1.0952380952380953, 105.0)

Numpy中的统计函数

► average()函数的沿轴计算

```
import numpy as np

a = np.arange(24).reshape(6,4)
print('我们的数组是:')
print(a)

print('在水平方向上做加权平均计算:')
wt1 = np.array([1,2,4,5])
print(np.average(a, axis = 1, weights = wt1))
print('水平方向计算结果与权重之和:')
print(np.average(a, axis = 1, weights = wt1, returned = True))

print('\n')
print('在竖直方向上做加权平均计算:')
wt2 = np.array([10,2,3,4,5,6])
print(np.average(a, axis = 0, weights = wt2))
print('竖直方向计算结果与权重之和:')
print(np.average(a, axis = 0, weights = wt2, returned = True))
```

我们的数组是:

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

在水平方向上做加权平均计算:

```
[ 2.08333333  6.08333333 10.08333333 14.08333333 18.08333333 22.08333333]
```

水平方向计算结果与权重之和:

```
(array([ 2.08333333,  6.08333333, 10.08333333, 14.08333333, 18.08333333,
        22.08333333]), array([12., 12., 12., 12., 12., 12.]))
```

在竖直方向上做加权平均计算:

```
[ 9.33333333 10.33333333 11.33333333 12.33333333]
```

竖直方向计算结果与权重之和:

```
(array([ 9.33333333, 10.33333333, 11.33333333, 12.33333333]), array([30., 30., 30., 30., 30., 30.]))
```

Numpy中的统计函数

- ▶ 中位数
- ▶ median() 函数

```
import numpy as np

a = np.array([[30,65,70],[80,95,10],[50,90,60],[40,95,20]])
print ('我们的数组是：')
print (a)
print ('\n')
print ('调用 median() 函数：')
print (np.median(a))
print ('\n')
print ('沿轴 0 调用 median() 函数：')
print (np.median(a, axis = 0))
print ('\n')
print ('沿轴 1 调用 median() 函数：')
print (np.median(a, axis = 1))
```

我们的数组是：

```
[[30 65 70]
 [80 95 10]
 [50 90 60]
 [40 95 20]]
```

调用 median() 函数：

62.5

沿轴 0 调用 median() 函数：

```
[45.  92.5 40.]
```

沿轴 1 调用 median() 函数：

```
[65. 80. 60. 40.]
```


Numpy中的统计函数

- ▶ 百分位数
- ▶ percentile()()函数

```
import numpy as np

a = np.array([[10, 7, 4], [3, 2, 1]])
print('我们的数组是:')
print(a)

print(np.percentile(a, 70))

print()
# axis 为 0, 在纵列上求
print(np.percentile(a, 70, axis=0))
print()
# axis 为 1, 在横行上求
print(np.percentile(a, 70, axis=1))
print()
# 保持维度不变
print(np.percentile(a, 70, axis=1, keepdims=True))
```

我们的数组是:

```
[[10  7  4]
 [ 3  2  1]]
5.5
```

```
[7.9 5.5 3.1]
```

```
[8.2 2.4]
```

```
[[8.2]
 [2.4]]
```

Numpy中的统计函数

- ▶ 标准差与方差
- ▶ std()函数 与var()函数
- ▶ $\text{std} = \sqrt{\text{mean}((x - x.\text{mean}())^2)}$

```
import numpy as np

print (np.std([1,2,3,4]))

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

```
1.118033988749895
1.25
```

```
import numpy as np

a = np.arange(24).reshape(6,4)
print ('我们的数组是：')
print (a)

print(np.std(a, axis=1))
print(np.std(a, axis=0))

print(np.var(a, axis=1))
print(np.var(a, axis=0))
```

我们的数组是：

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
[1.11803399 1.11803399 1.11803399 1.11803399 1.11803399 1.11803399]
[6.83130051 6.83130051 6.83130051 6.83130051]
[1.25 1.25 1.25 1.25 1.25 1.25]
[46.66666667 46.66666667 46.66666667 46.66666667]
```

Numpy数组的视图和副本

- ▶ 副本是一个数据的完整的拷贝。
 - 对副本进行修改，不会影响到原始数据，物理内存不在同一位置。
 - Python 中的`deepcopy()`函数。
 - numpy中的`ndarray.copy()` 函数
- ▶ 视图是数据的一个别称或引用。
 - 通过该别称或引用亦便可访问、操作原有数据。如果对视图进行修改，它会影响到原始数据，物理内存存在同一位置。
 - numpy 的切片操作返回原数据的视图。
 - 调用 ndarray 的 `view()` 函数产生一个视图。

Numpy数组的视图和副本

► 用copy()创建副本

```
import numpy as np

a = np.array([[10, 10], [2, 3], [4, 5]])
print (a)

b = a.copy()
print (b)

# b 和 a 完全不相关

b[0,0] = 100
b.resize(1,6)
print (b)
print (a)
```

```
[[10 10]
 [ 2  3]
 [ 4  5]]
[[10 10]
 [ 2  3]
 [ 4  5]]
[[100  10   2   3   4   5]]
[[10 10]
 [ 2  3]
 [ 4  5]]
```

Numpy数组的视图和副本

► 用view()创建视图

```
import numpy as np

a = np.array([[10,10], [2,3], [4,5]])
print (a)

b = a.view()
print (b)

# b 引用 a 的内部数据, 但表现方式可以不一致

b[0,0] = 100
b.resize(1,6)
print (b)
print (a)
```

```
[[10 10]
 [ 2  3]
 [ 4  5]]
[[10 10]
 [ 2  3]
 [ 4  5]]
[[100 10  2  3  4  5]]
[[100 10]
 [  2  3]
 [  4  5]]
```

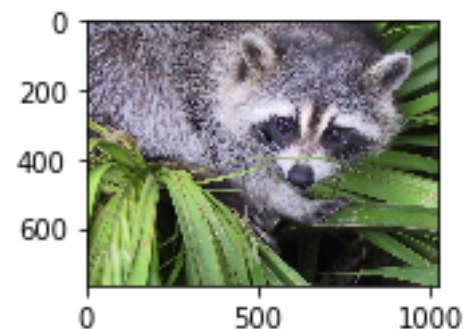
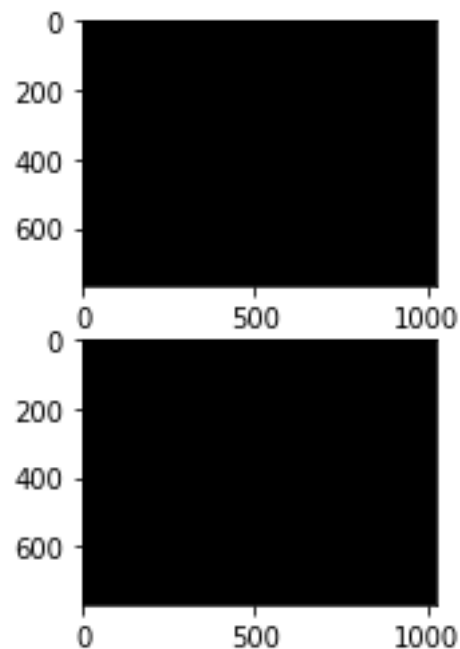
Numpy数组的视图和副本

```
import scipy.misc
import matplotlib.pyplot as plt

face = np.array(scipy.misc.face())
acopy = face.copy()
aview = face.view()

aview.flat=0

plt.subplot(221)
plt.imshow(face)
plt.subplot(222)
plt.imshow(acopy)
plt.subplot(223)
plt.imshow(aview)
plt.show()
```



Numpy的文件输入与输出

- ▶ `save()` 把arr数组存储到文件file中
 - `save(file, arr, allow_pickle=True, fix_imports=True)`
 - `.npy`为numpy文件格式
- ▶ `load()` 读取文件中的数组
 - `load(file)`

```
import numpy as np

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

# 保存到 outfile.npy 文件上
np.save('outfile.npy', a)

# 保存到 outfile2.npy 文件上,
# 如果文件路径末尾没有扩展名 .npy, 该扩展名会被自动加上
np.save('outfile2', a)
```

```
import numpy as np

b = np.load('outfile.npy')
print (b)
```

[1 2 3 4 5]

```
import numpy as np

c = np.load('outfile2.npy')
print (c)
```

[1 2 3 4 5]

Numpy的文件输入与输出

- ▶ savez() 把多个数组存储到文件file中
 - savez(file, *args, **kwds)

```
import numpy as np
```

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

```
b = np.arange(0, 1.0, 0.1)
```

```
c = np.sin(b)
```

```
# c 使用了关键字参数 sin_array
```

```
np.savez("abc.npz", a, b, sin_array = c)
```

```
r = np.load("abc.npz")
```

```
print(r.files) # 查看各个数组名称
```

```
print(r["arr_0"]) # 数组 a
```

```
print(r["arr_1"]) # 数组 b
```

```
print(r["sin_array"]) # 数组 c
```

```
['sin_array', 'arr_0', 'arr_1']
```

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

```
 [4 5 6]]
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

```
[0.          0.09983342 0.19866933 0.29552021 0.38941834 0.47942554
```

```
 0.56464247 0.64421769 0.71735609 0.78332691]
```


▶ savetxt() 函数

- 以简单的文本文件格式存储数据，对应的使用 loadtxt() 函数来获取数据。

```
: import numpy as np

a = np.array([1, 2, 3, 4, 5])
np.savetxt('out.txt', a)
b = np.loadtxt('out.txt')

print(b)
```

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

谢谢！