







# Python

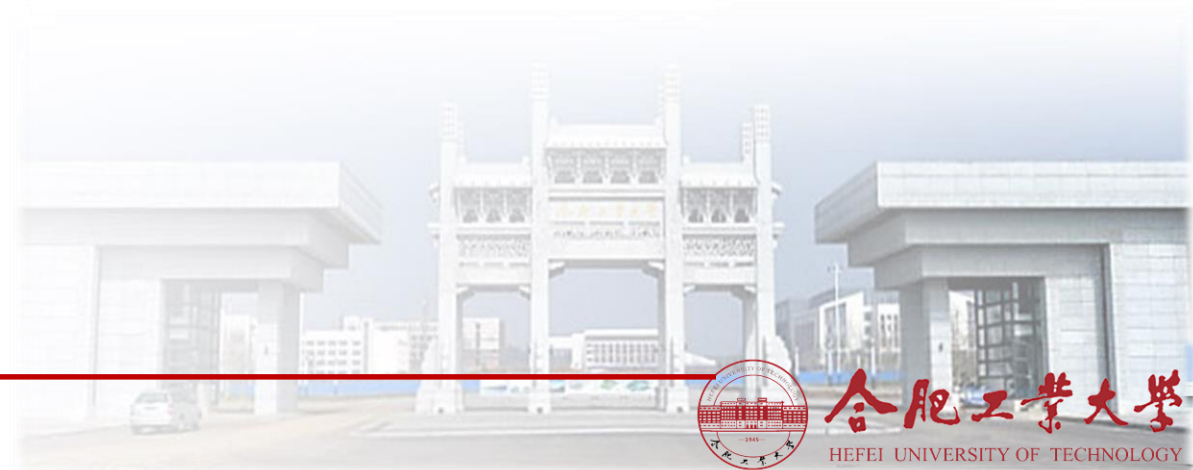
## 数据分析及可视化-8

数据分析库



# 本章学习目标

-  了解numpy库，掌握数组、矩阵的使用
-  掌握Pandas的两种数据结构Series、DataFrame
-  掌握Pandas的常见操作
-  掌握Pandas读写数据的方式







# 问题概述

## 要点：科学计算需要采用矩阵运算库numpy

开展基本的科学计算需要两个步骤：**组织数据**和**展示数据**；组织数据是运算的基础，也是将客观世界数字化的必要手段；展示数据是体现运算结果的重要方式，也是展示结论的有力武器。

## 矩阵：

数学的矩阵（Matrix）是一个按照长方阵列排列的复数或实数集合，最早来自于方程组的系数及常数所构成的方阵

科学计算领域最著名的计算平台Matlab采用矩阵作为最基础的变量类型。矩阵有维度概念，**一维矩阵是线性的**，类似于**列表**，**二维矩阵是表格状的**，这是常用的数据表示形式。





# numpy 库的使用

## Numpy概述

 numpy 是用于处理含有**同种元素**的多维数组运算的第三方库。

 数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从0开始。

 ndarray 类型的维度(dimensions)叫做轴(axes)，轴的个数叫做秩(rank)。一维数组的秩为1，二维数组的秩为2，二维数组相当于由两个一维数组构成。

由于numpy库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用numpy库：

```
>>>import numpy as np
```

其中，**as** 保留字与import一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。在程序的后续部分中，**np**代替**numpy**。





# numpy 库的使用

## 创建数组函数

函数	描述
<code>np.array([x,y,z],dtype=int)</code>	从Python列表或元组创建数组
<code>np.arange(x,y,s)</code>	创建一个由x到y，以s为步长的数组
<code>np.ones((m,n),dtype)</code>	创建一个m行n列全1的数组
<code>np.zeros((m,n),dtype)</code>	创建一个m行n列全0的数组
<code>np.random.rand(m,n)</code>	创建一个m行n列的[0, 1]之间随机二维数组
<code>np.random.randint(m,n)</code>	从给定上、下限范围内随机选取整数
<code>np.random.normal(a)</code>	以a为中心的正态分布样本值，
<code>np.random.uniform(a,b)</code>	产生在[a, b]中的均匀分布样本值
<code>np.random.beta()</code>	产生beta分布的样本值





# numpy 库的使用

## ndarray 类的常用属性

属性	描述
<code>ndarray.ndim</code>	数组轴的个数，也被称作秩
<code>ndarray.shape</code>	数组在每个维度上大小的整数元组
<code>ndarray.size</code>	数组元素的总个数
<code>ndarray.dtype</code>	数组元素的数据类型， <code>dtype</code> 类型可以用于创建数组中
<code>ndarray.itemsize</code>	数组中每个元素的字节大小
<code>ndarray.data</code>	包含实际数组元素的缓冲区地址
<code>ndarray.flat</code>	数组元素的迭代器







# ndarray 库的使用

## ndarray 类的操作方法

方法	描述
<code>ndarray.reshape(n,m)</code>	不改变数组 <code>ndarray</code> ，返回一个维度为(n,m)的数组
<code>ndarray.resize(new_shape)</code>	与 <code>reshape()</code> 作用相同，直接修改数组 <code>ndarray</code>
<code>ndarray.swapaxes(ax1, ax2)</code>	将数组 <code>n</code> 个维度中任意两个维度进行调换
<code>ndarray.flatten()</code>	对数组进行降维，返回一个折叠后的一维数组
<code>ndarray.ravel()</code>	作用同 <code>np.flatten()</code> ，但是返回数组的一个视图

数组在numpy中被当作对象，可以采用`<a>.<b>()`方式调用一些方法。这里给出了改变数组基础形态的操作方法，例如改变和调换数组维度等。其中，`np.flatten()`函数用于数组降维，相当于平铺数组中数据，该功能在矩阵运算及图像处理中用处很大。





# ndarray 库的使用

## ndarray 类的索引和切片方法

方法	描述
<code>x[i]</code>	索引第 $i$ 个元素
<code>x[-i]</code>	从后向前索引第 $i$ 个元素
<code>x[n:m]</code>	默认步长为 1，从前往后索引，不包含 $m$
<code>x[-m:-n]</code>	默认步长为 1，从后往前索引，结束位置为 $n$
<code>x[n,m,i]</code>	指定 $i$ 步长的由 $n$ 到 $m$ 的索引

数组切片得到的是原始数组的视图，**所有修改都会直接反映到源数组**。如果需要得到的 ndarray 切片的一份副本，需要进行复制操作，比如 `arange[5:8].copy()`







# numpy 库的使用

## ndarray 类的常用属性

```
>>> import numpy as np
>>> data=np.arange(12).reshape(3,4)
>>> data
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> type(data)
<class 'numpy.ndarray'>
>>> data.ndim
2
>>> data.shape
(3, 4)
>>> data.dtype
dtype('int32')
>>> data.astype(float)
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

```
>>> d1=np.ones((4,3))
```

```
>>> d1
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
>>> d1.T                                     #矩阵转置
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

```
>>> d1=np.zeros((3,4))
```

```
>>> d1
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
>>> d1.reshape(2,6)                         #重设矩阵的行、列
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```





# numpy 库的使用

## ndarray数组的切片

```
>>> ar1=np.arange(10)
>>> ar1
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> ar1[3]
3
>>> ar1[2:6]
array([2, 3, 4, 5])
>>> ar1[[0,3,8]]
array([0, 3, 8])
>>> ar2=np.arange(20).reshape(4,5)
>>> ar2
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

```
>>> ar2[1]
array([5, 6, 7, 8, 9])
>>> ar2[:,2]
array([ 2,  7, 12, 17])
>>> ar2[2,2]
12
>>> ar2[:,1:3]
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17]])
>>> ar2[1:3,2:4]
array([[ 7,  8],
       [12, 13]])
>>> ar2[[1,3],[2,4]]
array([ 7, 19])
>>> ar2[[0,2,3],[0,1,4]]
array([ 0, 11, 19])
>>> ar2[-4:-1:2]
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14]])
```





# numpy 库的算术运算函数

## numpy 库的算术运算函数

函数	描述
<code>np.add(x1, x2 [, y])</code>	$y = x1 + x2$
<code>np.subtract(x1, x2 [, y])</code>	$y = x1 - x2$
<code>np.multiply(x1, x2 [, y])</code>	$y = x1 * x2$
<code>np.divide(x1, x2 [, y])</code>	$y = x1 / x2$
<code>np.floor_divide(x1, x2 [, y])</code>	$y = x1 // x2$ , 返回值取整
<code>np.negative(x [,y])</code>	$y = -x$
<code>np.power(x1, x2 [, y])</code>	$y = x1^{**}x2$
<code>np.remainder(x1, x2 [, y])</code>	$y = x1 \% x2$

这些函数中，输出参数 $y$  可选，如果没有指定，将创建并返回一个新的数组保存计算结果；如果指定参数，则将结果保存到参数中。例如，两个数组相加可以简单地写为 $a+b$ ，而`np.add(a, b, a)`则表示 $a+=b$ 。







# numpy 库的比较运算函数

## numpy 库的比较运算函数

函数	描述
<code>np.equal(x1,x2[,y])</code>	$y=x1==x2$
<code>np.not_equal(x1,x2[,y])</code>	$y=x1!=x2$
<code>np.less(x1,x2[,y])</code>	$y=x1<x2$
<code>np.less_equal(x1,x2[,y])</code>	$y=x1\leq x2$
<code>np.greater(x1,x2[,y])</code>	$y=x1>x2$
<code>np.greater_equal(x1,x2[,y])</code>	$y=x1\geq x2$
<code>np.where(condition[x,y])</code>	根据给出的条件，判断输出为x，还是y

 其将返回一个布尔数组，它包含两个数组中对应元素值的比较结果。

```
>>> np.less([1,2],[2,2])  
array([ True, False])
```

 `where()` 函数是三元表达式 `x if condition else y` 的矢量版本。





# numpy 库的其他运算函数

## numpy 库的其他运算函数

函数	描述
<code>np.abs(x)</code>	计算基于元素的整形，浮点或复数的绝对值。
<code>np.sqrt(x)</code>	计算每个元素的平方根
<code>np.squire(x)</code>	计算每个元素的平方
<code>np.sign(x)</code>	计算每个元素的符号：1(+), 0, -1(-)
<code>np.ceil(x)</code>	计算大于或等于每个元素的最小值
<code>np.floor(x)</code>	计算小于或等于每个元素的最大值
<code>np rint (x[, out])</code>	圆整,取每个元素为最近的整数,保留数据类型
<code>np.exp(x[, out])</code>	计算每个元素指数值
<code>np.log(x), np.log10(x), np.log2(x)</code>	计算自然对数(e),基于 10,2 的对数, $\log(1 + x)$

numpy 库还包括三角运算函数、傅里叶变换、随机和概率分布、基本数值统计、位运算、矩阵运算等非常丰富的功能，读者在使用时可以到官方网站查询。





# numpy 库实例

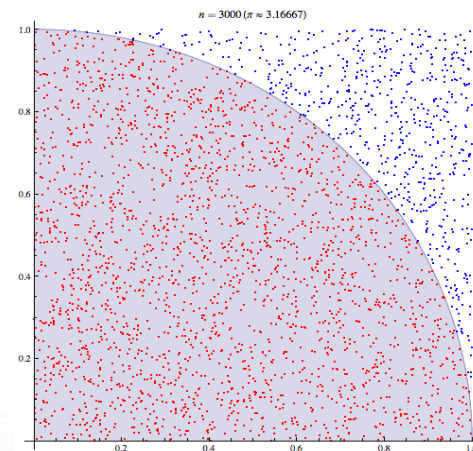
## $\pi$ 的求解

### 应用蒙特卡洛方法求解 $\pi$

```
>>> n=1000
>>> for i in range(5):
    start=time.perf_counter
    x=np.random.rand(n)
    y=np.random.rand(n)
    d=np.sqrt(np.squre(x)+np.squre(y))
    pi=4*len(d[d<1])/n
    end=time.perf_counter()
    print("n=%-10i  $\pi$  的值:%f 耗时:%f"%(n, pi, end-start))
    n*=10
```

n=1000	$\pi$ 的值:3.104000	耗时:0.044515
n=10000	$\pi$ 的值:3.165600	耗时:0.001067
n=100000	$\pi$ 的值:3.142760	耗时:0.009244
n=1000000	$\pi$ 的值:3.143232	耗时:0.078801
n=10000000	$\pi$ 的值:3.141261	耗时:0.729611

使用数组随机函数，通过向量运算，统计圆内的点数，求解  $\pi$ 。



计算 $\pi$ 使用的1/4区域和抛点过程







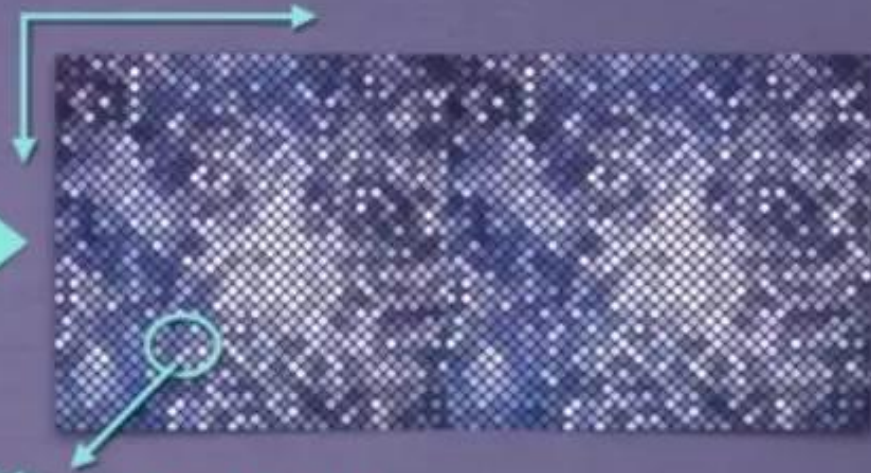
# 实例 图像的手绘效果



## 图像的数组表示

图像是有规则的二维数据，可以用numpy库将图像转换成数组对象。

图像的数组表示



RGB值: (R, G, B)

图像是一个由像素组成的二维矩阵，每个元素是一个RGB值。









# 实例 图像的手绘效果

## 图像的数组表示

需要用到的方法：

-  `Image.open()`：打开图片
-  `np.array()`：将图像转化为数组
-  `convert("L")`：将图片转换成二维灰度图片
-  `Image.fromarray()`：将数组还原成图像uint8格式

```
>>> from PIL import Image
>>> import numpy as np
>>> im=np.array(Image.open("d:\\p_train\\img\\boy.jpg"))
>>> print(im.shape,im.dtype)
(492, 500, 3) uint8
```

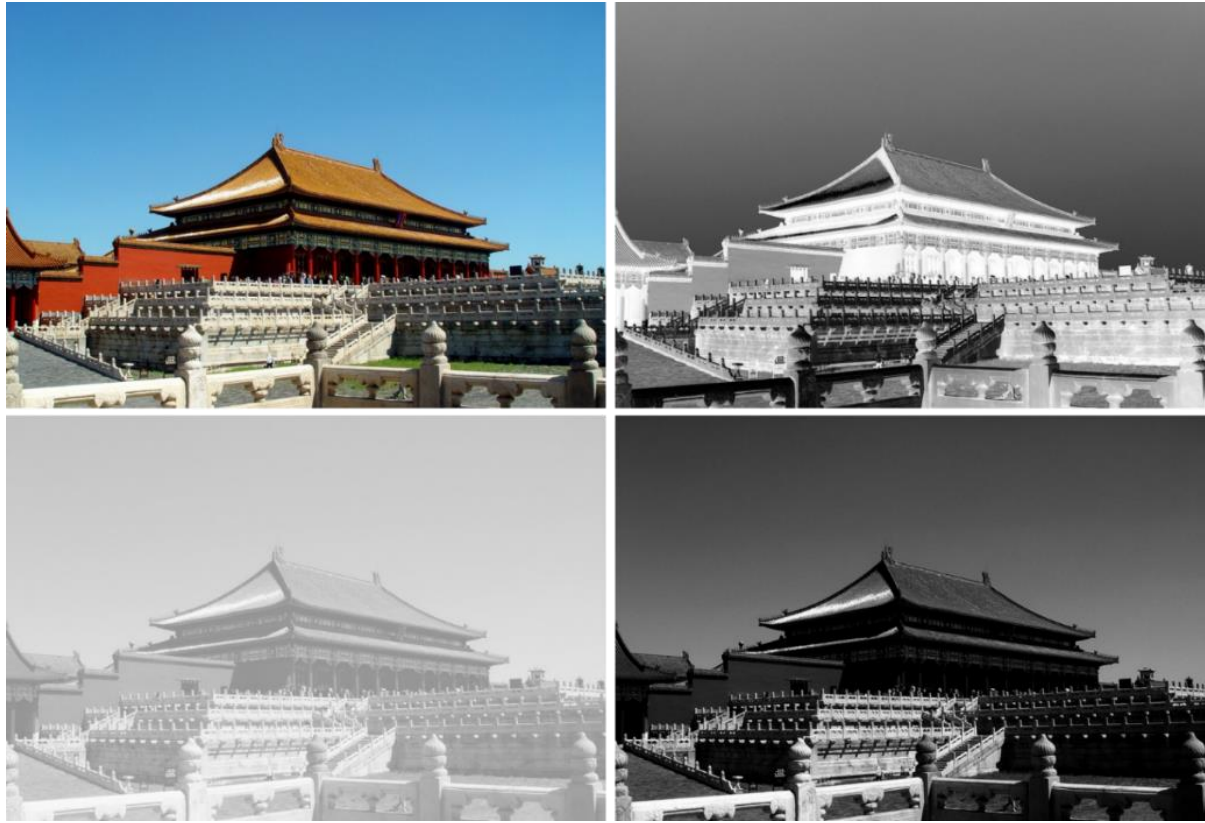




# 实例 图像的手绘效果



## 图像的像素处理



图像从彩色变为带有灰度的黑白色。转换后，图像的ndarray 类型变为二维数据，每个像素点色彩只由一个整数表示。

将图像读入ndarray 数组对象后，可以通过任意数学操作来获取相应的图像变换。以灰度变换为例，分别对灰度变化后的图像进行反变换、区间变化和像素值平方处理。

灰度值指黑白图像中点的颜色深度，范围从0到255，黑白图像主要用于构建非可见光图像，例如医学中超声波形成的图像等。RGB 彩色图片可以通过如下公式转换成灰度值：

$$\text{Gray} = R * 0.3 + G * 0.59 + B * 0.11$$

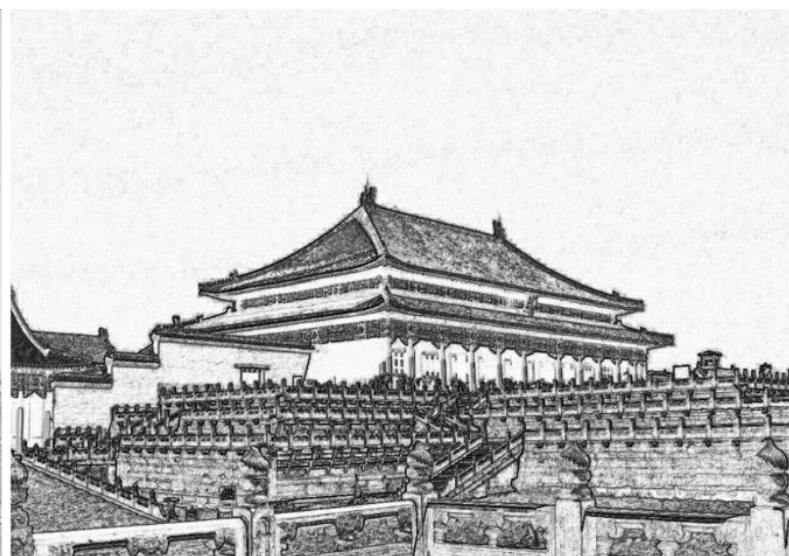
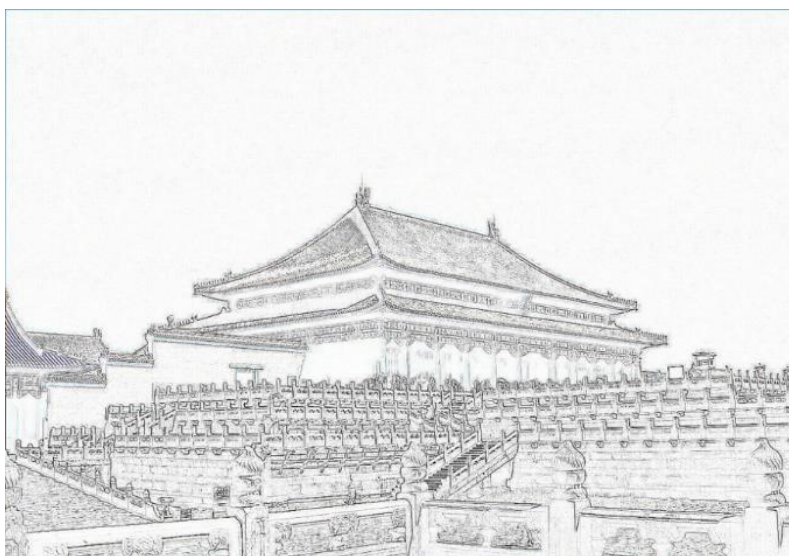






# 图像的对比

## 图像轮廓和手绘效果



实现手绘风格，即黑白轮廓描绘，首先需要读取原图像的明暗变化，即灰度值。从直观视觉感受上定义，图像灰度值显著变化的地方就是梯度，它描述了图像灰度变化的强度。

通常可以使用梯度计算来提取图像轮廓，numpy中提供了直接获取灰度图像梯度的函数 `gradient()`，传入图像数组表示即可返回代表x 和y 各自方向上梯度变化的二维元组



## 图像的手绘效果处理

实现思路步骤：

 梯度的重构

`np.gradient(a)`： 计算数组a中元素的梯度，f为多维时，返回每个维度的梯度。

当为二维数组时，`np.gradient(a)` 得出两个数组，第一个数组对应最外层维度的梯度，第二个数组对应第二层维度的梯度。

代码如下：

```
grad=np.gradient(a)
grad_x,grad_y=grad
grad_x = grad_x * depth / 100.
grad_y = grad_y * depth / 100.
```

#对grad\_x值进行归一化  
#对grad\_y值进行归一化





# 像素处理

## 图像的手绘效果处理

### 构造光源效果

设计一个位于图像斜上方的虚拟光源, 光源相对于图像的视角为Elevation, 方位角为Azimuth。

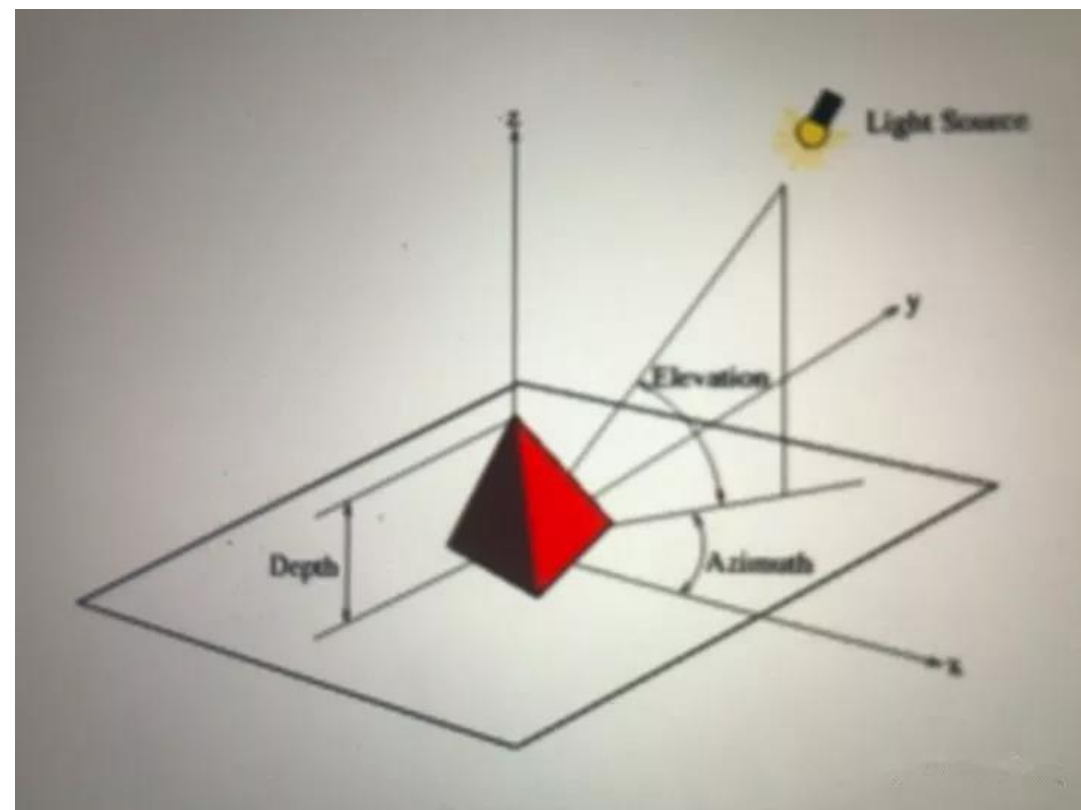
建立光源对各点梯度值的影响函数, 计算出各点的新像素值, 其中:

$\text{np.cos}(\text{evc.el})$ : 单位光线在地平面上的投射长度。

$dx, dy, dz$ : 光源对 $x, y, z$ 三方向的影响程度。

### 梯度归一化

构造 $x$ 和 $y$ 轴梯度的三维归一化单位坐标系; 梯度与光源相互作用, 将梯度转化为灰度。

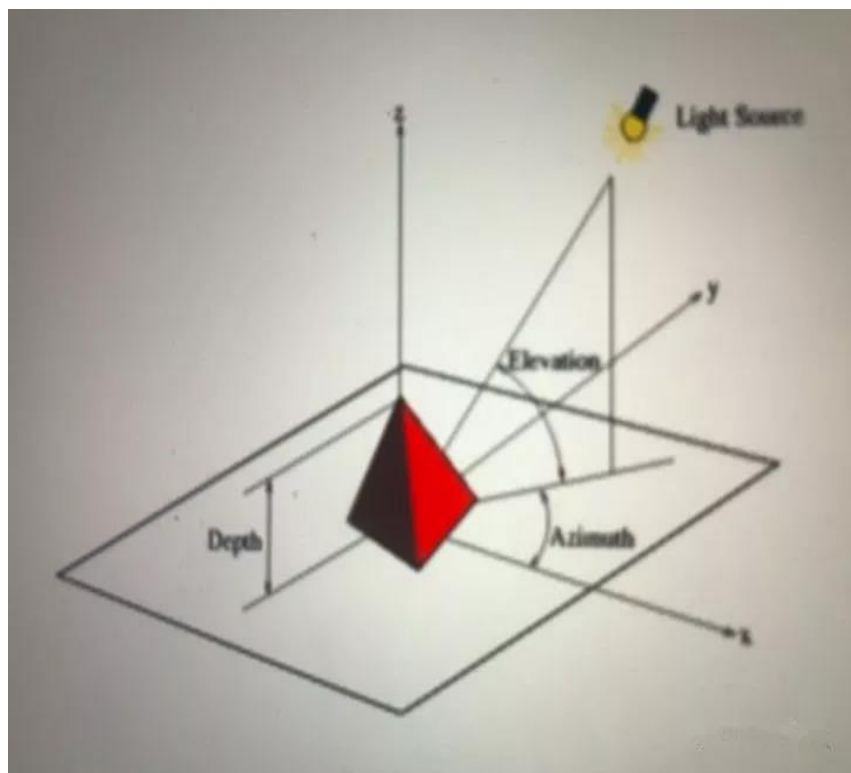






## 图像的手绘效果处理

图像生成



```
from PIL import Image
import numpy as np
import os, join, time
def image(sta, end, depths=10):
    a = np.array(Image.open(sta).convert('L')).astype('float')
    depth = depths # 深度的取值范围(0-100), 标准取10
    grad = np.gradient(a) # 取图像灰度的梯度值
    grad_x, grad_y = grad # 分别取横纵图像梯度值
    grad_x = grad_x * depth / 100. # 对grad_x值进行归一化
    grad_y = grad_y * depth / 100. # 对grad_y值进行归一化
    A = np.sqrt(grad_x ** 2 + grad_y ** 2 + 1.)
    uni_x = grad_x / A
    uni_y = grad_y / A
    uni_z = 1. / A
    vec_el = np.pi / 2.2 # 光源的俯视角, 弧度值
    vec_az = np.pi / 4. # 光源的方位角, 弧度值
    dx = np.cos(vec_el) * np.cos(vec_az) # 光源对x轴的影响
    dy = np.cos(vec_el) * np.sin(vec_az) # 光源对y轴的影响
    dz = np.sin(vec_el) # 光源对z轴的影响
    b = 255 * (dx * uni_x + dy * uni_y + dz * uni_z) # 光源归一化
    b = b.clip(0, 255)
    im = Image.fromarray(b.astype('uint8')) # 重构图像
    im.save(end)
```



# 数据分析工具pandas

## pandas概述

pandas是一个基于Numpy的Python库，专门为解决数据分析任务而创建的。

pandas中有两个主要的数据结构：**Series**和**DataFrame**。

其中，Series是**一维**的数据结构，DataFrame是**二维**的表格型数据结构。

由于pandas 库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用pandas 库：

```
>>>import pandas as pd
```

其中，as 保留字与import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。在程序的后续部分中，**pd**代替pandas。



# 数据分析工具pandas

## Series

Series是一个类似于一维数组的对象，它能够保存任何类型的数据，主要由一组数据和与之相关的索引两部分构成，分别为values和index。

Pandas的Series类对象可以使用以下构造方法创建：

**class : pd.Series(data,index)**

data: 传入的数据，可以是ndarray,list等；

index: 索引，必须是唯一的，且与数据的长度相同，如果没有传入索引，则默认会自动创建一个从0-n的整数索引，可以切片操作。

Series	
index	element
0	1
1	2
2	3
3	4
4	5

```
>>> ds=pd.Series([1,2,3,4,5])
```

```
>>> ds
```

```
0    1
```

```
1    2
```

```
2    3
```

```
3    4
```

```
4    5
```

```
dtype: int64
```

```
>>> ds[2]
```

```
3
```

```
>>> ds=pd.Series(data=[1,2,3,4,5],index=['a','b','c','d','e'])
```

```
>>> ds
```

```
a    1
```

```
b    2
```

```
c    3
```

```
d    4
```

```
e    5
```

```
dtype: int64
```

```
>>> ds['b']
```

```
2
```

```
#将二个序列数据匹配转成一维数组
```

```
>>> dsdc=pd.Series({"语文":90,"数学":92,"英语":88})
```

```
>>> dsdc #将字典转成一维数组
```

```
语文    90
```

```
数学    92
```

```
英语    88
```

```
dtype: int64
```





# 数据分析工具pandas

## Series

```
>>> ds.values  
array([1, 2, 3, 4, 5], dtype=int64)  
>>> ds.index  
Index(['a', 'b', 'c', 'd', 'e'],  
      dtype='object')
```

```
>>> ds["b"]=4  
>>> ds  
a    1  
b    4  
c    3  
d    4  
e    5  
dtype: int64
```

```
>>> ds.index=['h', 'i', 'j', 'k', 'l']  
>>> ds  
h    1  
i    4  
j    3  
k    4  
l    5  
dtype: int64
```

```
>>> ds.sort_index(ascending=False)  
l    5  
k    4  
j    3  
i    4  
h    1  
dtype: int64
```

```
>>> ds.sort_values(ascending=True)  
h    1  
j    3  
i    4  
k    4  
l    5  
dtype: int64
```







# 数据分析工具pandas

## Series

```
>>> ds.reset_index() #创建重新索引的新对象
```

	index	0
0	h	1
1	i	4
2	j	3
3	k	4
4	l	5

```
>>> ds.reset_index(drop=True)
```

0	1
1	4
2	3
3	4
4	5

dtype: int64

```
>>> ds.where(ds>4) #满足条件的显示
```

h	NaN
i	NaN
j	NaN
k	NaN
l	5.0

dtype: float64

```
>>> ds.where(ds>4, 0) #不大于4的用0替换
```

h	0
i	0
j	0
k	0
l	5

dtype: int64

```
>>> dsc=ds.value_counts() #统计出现次数
```

```
>>> dsc
```

4	2
5	1
3	1
1	1

dtype: int64





# 数据分析工具pandas

## Series

#apply()方法对Series对象的值进行函数运算

```
>>> ds.apply(lambda x:x*3)
```

```
b      3
```

```
r     12
```

```
d      9
```

```
f     12
```

```
s     15
```

```
dtype: int64
```

```
>>> ds.sum()
```

#求和

```
17
```

```
>>> ds.mean()
```

#平均值

```
3.4
```

```
>>> ds.max()
```

#求最大值

```
5
```

```
>>> ds.std()
```

#求标准差

```
1.5165750888103102
```

```
>>> ds.var()
```

#求无偏方差

```
2.3000000000000003
```

```
>>> ds.sem()
```

#求无偏标准差

```
0.6782329983125268
```

```
>>> dl=ds.to_list()
```

```
>>> dl
```

```
[1, 4, 3, 4, 5]
```

```
>>> dar=ds.to_numpy()
```

```
>>> dar
```

```
array([1, 4, 3, 4, 5], dtype=int64)
```

```
>>> ddc=ds.to_dict()
```

```
>>> ddc
```

```
{ 'h': 1, 'i': 4, 'j': 3, 'k': 4, 'l': 5 }
```





# 数据分析工具pandas

## 时间序列函数date\_range()

该函数主要用于生成一个固定频率的时间索引：

`pd.date_range(start=None, end=None, periods=None, freq='D')`

`start`: 开始时间; `end`: 结束时间; `periods`: 固定时期, 取值为整数或None

`freq`: 日期偏移量(D, M, Y, H, T, S, W等), 取值为string或DateOffset, 默认为'D'

```
>>> pd.date_range(start='20190101', end='20191231', freq='M')
```

```
DatetimeIndex(['2019-01-31', '2019-02-28', '2019-03-31', '2019-04-30',  
              '2019-05-31', '2019-06-30', '2019-07-31', '2019-08-31',  
              '2019-09-30', '2019-10-31', '2019-11-30', '2019-12-31'],  
              dtype='datetime64[ns]', freq='M')
```

```
>>> pd.date_range(start='20190101', periods=12, freq='MS')
```

```
DatetimeIndex(['2019-01-01', '2019-02-01', '2019-03-01', '2019-04-01',  
              '2019-05-01', '2019-06-01', '2019-07-01', '2019-08-01',  
              '2019-09-01', '2019-10-01', '2019-11-01', '2019-12-01'],  
              dtype='datetime64[ns]', freq='MS')
```

```
>>> pd.Series(data=np.random.uniform(15,20,24),index=pd.date_range(start='20190101',periods=24,freq='H'))
```

#以时间序列作索引, 生成一维序列Series对象。



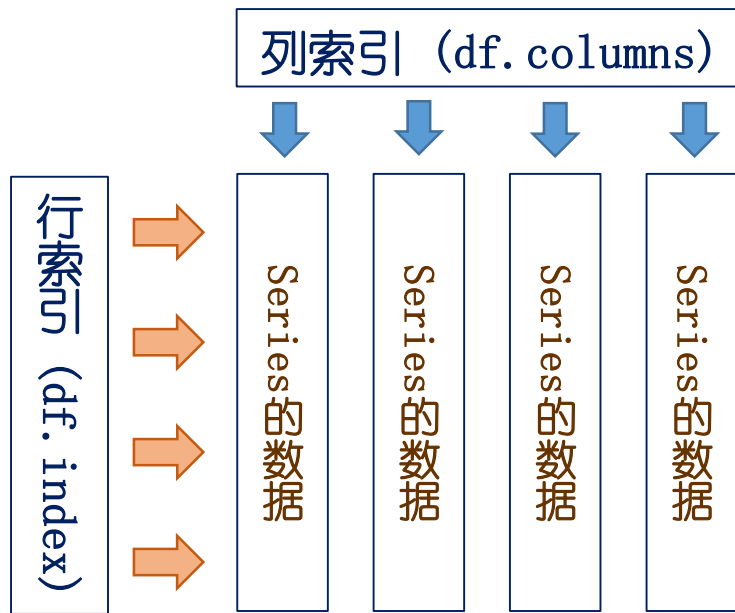




# 数据分析工具pandas

## DataFrame

DataFrame是一个类似于二维数组或表格(如Excel)的对象，它每列的数据可以是不同的数据类型，与Series结构相似，也是由索引和数据两部分构成，但DataFrame不仅有行索引，还有列索引，分别为index和columns，其值为values。



index	columns		
	a	b	c
0	1	3	5
1	2	4	6
2	3	5	7
3	4	6	8
4	5	7	9

DataFrame



# 数据分析工具pandas



## DataFrame

Pandas的DataFrame类对象可以使用以下构造方法创建：

class: **pd.DataFrame(data,index,columns)**

data: 传入的数据，可以是二维数组值；

index: 行索引，如果没有传入索引，则默认会自动创建一个从0-n的整数索引。

columns: 列索引，如果没有传入索引，则默认会自动创建一个从0-n的整数索引。





# 数据分析工具pandas

## DataFrame创建

用二维数组创建DF数据对象:

```
>>> ar=np.array([[1, 7, 4, 2, 9],  
                  [2, 6, 5, 3, 8],  
                  [3, 5, 6, 4, 7]])
```

```
>>> ar=ar.T
```

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

```
>>> pd.DataFrame(ar, columns=['a', 'b', 'c'])
```

	a	b	c
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7

用字典创建DF数据对象:

```
>>> 11=[82, 86, 95, 83, 88]
```

```
>>> 12=[73, 85, 86, 94, 97]
```

```
>>> pd.DataFrame({'数学':11, '计算机':12})
```

	数学	计算机
0	82	73
1	86	85
2	95	86
3	83	94
4	88	97

```
>>> pd.DataFrame([11, 12], index=['数学', '计算机'])
```

	0	1	2	3	4
数学	82	86	95	83	88
计算机	73	85	86	94	97







# 数据分析工具pandas

## DataFrame切片操作

DataFrame中每列的数据都是一个Series对象，  
可以使用列索引名进行获取。

```
>>> df
   a  b  c
0  1  2  3
1  7  6  5
2  4  5  6
3  2  3  4
4  9  8  7
```

```
>>> df['b']
0    2
1    6
2    5
3    3
4    8
Name: b, dtype: int32
```

```
>>> df.b
0    2
1    6
2    5
3    3
4    8
Name: b, dtype: int32
```

```
>>> df[['a', 'c']]
   a  c
0  1  3
1  7  5
2  4  6
3  2  4
4  9  7
```

DataFrame行索引名切片

```
>>> df[:3]
   a  b  c
0  1  2  3
1  7  6  5
2  4  5  6
```

```
>>> df[3:4]
   a  b  c
3  2  3  4
```

分别用行、列索引切片


```
>>> df[1:4][['a', 'c']]
   a  c
1  7  5
2  4  6
3  2  4
>>> df[:,2]['a']
0    1
2    4
4    9
Name: a, dtype: int32
```






# 数据分析工具pandas

## DataFrame索引方法操作

 **loc**: 基于**标签索引**(索引名称, 如a, b等), 用于按**标签选取数据**, 当执行切片操作时, 既包含起始索引, 也包含结束索引, 但名称如为字符, 则不能切片。

 **iloc**: 基于**位置索引**(整数索引, 从0到length-1), 用于按**位置选取数据**, 当执行切片操作时, 包含起始索引, 不包含结束索引。

```
>>> ar=np.arange(20).reshape(4, 5)
```

```
>>> df=pd.DataFrame(ar)
```

```
>>> df
```

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

```
>>> df.loc[1:3, 2:4]
```

	2	3	4
1	7	8	9
2	12	13	14
3	17	18	19

```
>>> df.iloc[1:3, 2:4]
```

	2	3
1	7	8
2	12	13





# 数据分析工具pandas

## DataFrame索引方法操作

```
>>> df=pd.DataFrame(ar,columns=['a','b','c','d','e'])
```

```
>>> df
```

	a	b	c	d	e
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19

```
>>> df.loc[:,['a','c','d']]
```

	a	c	d
0	0	2	3
1	5	7	8
2	10	12	13
3	15	17	18

```
>>> df.iloc[:,[0,2,3]]
```

```
>>> df.loc[1:2,['b','c']]
```

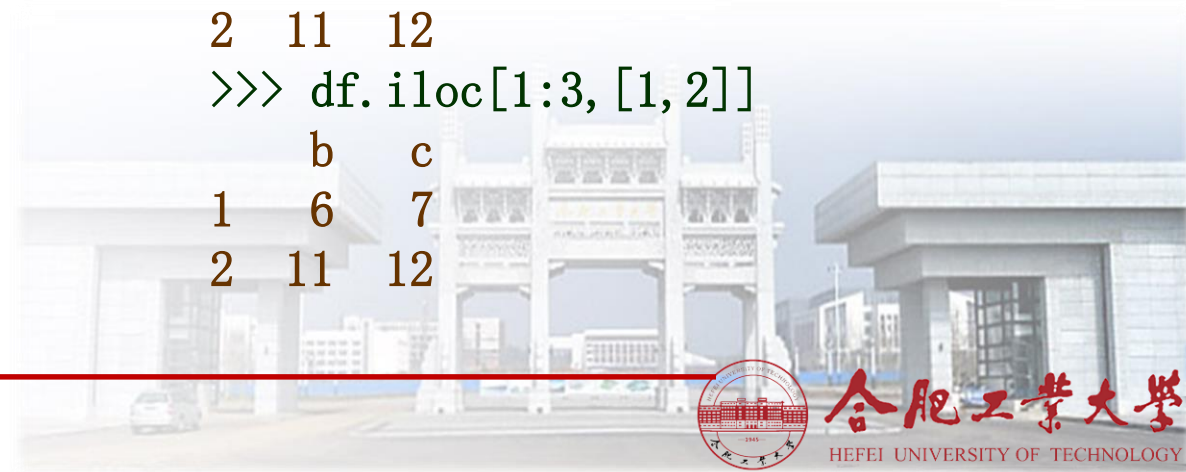
	b	c
1	6	7
2	11	12

```
>>> df.iloc[1:3,1:3]
```

	b	c
1	6	7
2	11	12

```
>>> df.iloc[1:3,[1,2]]
```

	b	c
1	6	7
2	11	12







# 数据分析工具pandas

## DataFrame修改操作

```
>>> df
```

	a	b	c
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7

#整体修改二维表对象的行索引和列索引

```
>>> df1.index=['E','C','A','D','B']
```

```
>>> df1.columns=['y','x','z']
```

```
>>> df1
```

	y	x	z
E	1	2	3
C	7	6	5
A	4	5	6
D	2	3	4
B	9	8	7

#个别修改行索引

```
>>> df.rename(index={'A':'P'})
```

	y	x	z
E	1	2	3
C	7	6	5
P	4	5	6
D	2	3	4
B	9	8	7

#个别修改列索引

```
>>> df.rename(columns={'z':'t','y':'s'},inplace=True)
```

```
>>> df
```

	s	x	t
E	1	2	3
C	7	6	5
A	4	5	6
D	2	3	4
B	9	8	7

#重建行索引

```
>>> df.reset_index(drop=True)
```

	s	x	t
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7





# 数据分析工具pandas

## DataFrame修改操作

```
>>> df
   a  b  c
0  1  2  3
1  7  6  5
2  4  5  6
3  2  3  4
4  9  8  7
```

#将1行b列数据更新为0

```
>>> df.loc[1, 'b'] = 0
```

```
      # df.iloc[1, 1] = 0
```

```
>>> df
   a  b  c
0  1  2  3
1  7  0  5
2  4  5  6
3  2  3  4
4  9  8  7
```

#将0列0、2、4三行的数据更改为NaN

```
>>> df.iloc[::2, 0] = np.NaN
```

```
>>> df
   a  b  c
0 NaN 2  3
1  7.0 6  5
2 NaN 5  6
3  2.0 3  4
4 NaN 8  7
```

#将1行b、c两列的数据更改为10

```
>>> df.loc[1, ['b', 'c']] = 10
```

```
>>> df
   a  b  c
0  1  2  3
1  7 10 10
2  4  5  6
3  2  3  4
4  9  8  7
```





# 数据分析工具pandas

## DataFrame修改操作

```
>>> df
```

	a	b	c
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7

#筛选出a列中所有大于5的数据

```
>>> df['a'][df['a']>5]
```

1	7
4	9

#筛选出4行中所有大于7的数据

```
>>> df.iloc[4][df.iloc[4]>7]
```

a	9
b	8

Name: 4, dtype: int32

#筛选出所有大于5的数据

```
>>> df[df>5]
```

	a	b	c
0	NaN	NaN	NaN
1	7.0	6.0	NaN
2	NaN	NaN	6.0
3	NaN	NaN	NaN
4	9.0	8.0	7.0

```
>>> df.where(df>5)
```

	a	b	c
0	NaN	NaN	NaN
1	7.0	6.0	NaN
2	NaN	NaN	6.0
3	NaN	NaN	NaN
4	9.0	8.0	7.0

#将所有大于5的数据用20替换

```
>>> df.where(df<=5, 20)
```

	a	b	c
0	1	2	3
1	20	20	5
2	4	5	20
3	2	3	4
4	20	20	20





# 数据分析工具pandas

## DataFrame删除数据

**df\_obj.drop(labels=None,axis=0,inplace=False)**

**labels:** 表示列名或行名。

**axis:** 取0, 为对行操作, 取1则对列操作。

**inplace:** 取False, 创建新对象。

```
>>> df
```

	a	b	c
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7

#删除索引1行的数据

```
>>> df.drop(labels=1,axis=0)
```

	a	b	c
0	1	2	3
2	4	5	6
3	2	3	4
4	9	8	7

#删除索引为b列的数据

```
>>> df.drop(labels='b',axis=1)
```

	a	c
0	1	3
1	7	5
2	4	6
3	2	4
4	9	7

```
>>>del df['b']
```







# 数据分析工具pandas

## DataFrame删除数据

**df\_obj.drop(df\_obj[条件].index)** #删除满足条件的行

```
>>> df
```

	a	b	c
0	1	2	3
1	7	6	5
2	4	5	6
3	2	3	4
4	9	8	7

#删除满足a列数据中值为7的行

```
>>> df.drop(df[df['a']==7].index)
```

	a	b	c
0	1	2	3
2	4	5	6
3	2	3	4
4	9	8	7

#删除满足b列数据中为偶数的所有行

```
>>> df.drop(df[df['b']%2==0].index)
```

	a	b	c
2	4	5	6
3	2	3	4





# 数据分析工具pandas

## DataFrame数据按索引排序

**df\_obj.sort\_index(axis=0,ascending=True,inplace=False)**

**axis:** 轴索引, 0表示index(按行), 1表示columns(按列)。

**ascending:** 是否升序排列, 默认为True, 表示升序。

**inplace:** 默认为False, 表示对数据表进行排序, 但不创建新实例。

```
>>> df
```

	y	x	z
E	1	2	3
C	7	6	5
A	4	5	6
D	2	3	4
B	9	8	7

```
>>> df.sort_index(axis=0)
```

	y	x	z
A	4	5	6
B	9	8	7
C	7	6	5
D	2	3	4
E	1	2	3

```
>>> df.sort_index(axis=1,ascending=False)
```

	z	y	x
E	3	1	2
C	5	7	6
A	6	4	5
D	4	2	3
B	7	9	8





# 数据分析工具pandas

## DataFrame数据按值排序

**df\_obj.sort\_values(by,axis=0,ascending=True,inplace=False,  
na\_position='last')**

**by:** 表示排序的列。

**axis:** 为0, 取列索引, 将每行数据按该列中数据的升序或降序调整;  
为1, 取行索引, 将每列数据按该行中数据的升序或降序调整。

**na\_position:** 若为first, 则会将NaN值放在开头, 若为last, 则会将NaN值放在最后。

```
>>> df
```

	y	x	z
E	1	2	3
C	7	6	5
A	4	5	6
D	2	3	4
B	9	8	7

```
>>> df.sort_values(by='x',ascending=False)
```

	y	<b>x</b>	z
B	9	<b>8</b>	7
C	7	<b>6</b>	5
A	4	<b>5</b>	6
D	2	<b>3</b>	4
E	1	<b>2</b>	3

```
>>> df.T.sort_values(by='B').T
```

	z	x	y
E	3	2	1
C	5	6	7
A	6	5	4
D	4	3	2
<b>B</b>	<b>7</b>	<b>8</b>	<b>9</b>

```
>>> df.sort_values(by='B',axis=1)
```





# 数据分析工具pandas

## Pandas常用统计计算

函数名称	说明	函数名称	说明
sum	计算和	std	样本值的标准差
mean	计算平均值	skew	样本值的偏度
median	获取中位数	kurt	样本值的峰度
max, min	获取最大值和最小值	cumsum	样本值的累加和
idxmax idxmin	获取最大和最小索引值	cummin cummax	样本值累积最小值和最大值
count	计算非NaN值的个数	cumprod	样本值的累乘积
head	获取前N个值	describe	对Series和DataFrame列计算汇总统计
var	样本值的方差		

```
>>> df.describe()
           y         x         z
count  5.000000  5.000000  5.000000
mean   4.600000  4.800000  5.000000
std    3.361547  2.387467  1.581139
min     1.000000  2.000000  3.000000
25%     2.000000  3.000000  4.000000
50%     4.000000  5.000000  5.000000
75%     7.000000  6.000000  6.000000
max     9.000000  8.000000  7.000000
```







# 数据分析工具pandas

## Pandas读写文件数据

### 读入文件数据:

```
pd.read_csv(pathfile, header, index_col)
pd.read_excel(pathfile, header, index_col, sheet_name='sheet1')
pd.read_html(htmlfile) #读入网页表格数据
pd.read_sql(sql, con, index_col, coerce_float, parame) #读入数据库
```

### 数据写入文件:

```
df_obj.to_csv(pathfile, index=True, sep=',', mode='w', encoding=None)
df_obj.to_excel(pathfile, sheet_name=None)
df_obj.to_sql(name, con, schema)
```



# 数据分析工具pandas

## 案例

 读入一数据文件(stcj1.xlsx)，如右图。

 分别求出每门课的最高分、最低分、平均分、标准差。

```
>>> df=pd.read_excel('stcj1.xlsx',index_col=0)
>>> df.head()
>>> tj=df.describe()
>>> tj
>>> print(tj.loc['max'],tj.loc['min'],tj.loc['mean'],tj.loc['std'],tj="\n")
```

 求出每人的总分，并按总分降序排序。

```
>>> df['总分']=df['写作']+df['英语']+df['逻辑']+df['计算机']
>>> df.head()
>>> df1=df.sort_values(by='总分',ascending=False)
# df.sort_values(by='总分',ascending=False,inplace=True)
```

	A	B	C	D	E	F	G
1	学号	姓名	性别	写作	英语	逻辑	计算机
2	1001	马红丽	女	89	89	92	95
3	1002	刘绪	女	79	81	95	80
4	1003	付艳丽	女	86	93	94	91
5	1004	张建立	男	82	88	92	77
6	1005	魏翠香	女	83	97	94	93
7	1006	赵晓娜	女	75	84	84	85
8	1007	刘宝英	女	82	90	92	84
9	1008	郑会锋	女	85	77	96	91
10	1009	申永琴	女	72	96	88	91
11	1010	许宏伟	女	90	90	89	86





# 祝学习进步!

