[**python中numpy函**](http://blog.csdn.net/u011367448/article/details/17370393)**数应用**

[**0、如果geany不能显示中文，就在第一行加一句代码：（必须是第一行）**](#_Toc13397)

[**1、在Python中使用help帮助**](#_Toc27674)

[**2、创建数组**](#_Toc7282)

[**2.1、使用array函数创建时，参数必须是由方括号括起来的列表，而不能使用多个数值作为参数调用array。**](#_Toc31395)

[**2.2、NumPy提供一个类似arange的函数返回一个数列形式的数组:**](#_Toc28630)

[**2.3、当arange使用浮点数参数时，由于浮点数精度有限，通常无法预测获得的元素个数。因此，最好使用函数linspace去接收我们想要的元素个数来代替用range来指定步长。linespace用法如下，将在通用函数一节中详细介绍。**](#_Toc2202)

[**3、 创建二维数组 array() ：**](#_Toc7717)

[**4、 NumPy中的基本数据类型**](#_Toc9005)

[**5、自定义结构数组**](#_Toc21386)

[**6、 组合函数（函数有hstack、vstack、dstack、row\_stack、column\_stack ）**](#_Toc4807)

[**6.1、水平组合**](#_Toc30724)

[**6.2、垂直组合**](#_Toc25506)

[**6.3、深度组合**](#_Toc565)

[**6.4、行组合**](#_Toc10212)

[**6.5、列组合**](#_Toc403)

[**7、分割数组（hsplit、vsplit、dsplit和split）**](#_Toc26499)

[**7.1、水平分割**](#_Toc8999)

[**7.2、垂直分割**](#_Toc5266)

[**7.3、面向深度的分割**](#_Toc19773)

[**8、复制和镜像（View）**](#_Toc20128)

[**8.1、完全不复制**](#_Toc4541)

[8.2视图(view)和浅复制](#_Toc22087)

[8.3、切片数组返回它的一个视图：](#_Toc29686)

[8.4、深复制](#_Toc23086)

[**9、 Numpy扩充数组函数之repeat和tile用法**](#_Toc9770)

[**10、 排序（使用argsort，sort，sorted）**](#_Toc20733)

[**10.1 argsort的用法**](#_Toc17439)

[**10.2、sort, sorted, itemgetter的用法**](#_Toc28697)

[11、 python中read() readline()以及readlines()区别](#_Toc31607)

[12、 strip()函数 介绍](#_Toc22217)

[**13、详解Python中的split()函数的使用方法**](#_Toc4585)

[**14、Python使用os.listdir()函数来得目录内容（os.mkdir()函数创建目录,os.rmdir()函数删除目录）**](#_Toc21886)

[**15、 python数据持久存储：pickle模块的基本使用**](#_Toc23054)

[**16、 arange 与range的区别**](#_Toc5091)

[**python中numpy函**](http://blog.csdn.net/u011367448/article/details/17370393)**数应用**

**0、如果geany不能显示中文，就在第一行加一句代码：（必须是第一行）**

# -\*- coding: UTF-8 -\*-

**1、在Python中使用help帮助**

>>> import numpy

>>> help(numpy.argsort)

**2、创建数组**

　　先来介绍创建数组。创建数组的方法有很多。如可以使用array函数从常规的Python列表和元组创造数组。所创建的数组类型由原序列中的元素类型推导而来。

>>> from numpy import \*

>>> a = array( [2,3,4] )

>>> a

array([2, 3, 4])

对于一个一维数组，可用print（a[0:10]）输出前十项数

**2.1、使用array函数创建时，参数必须是由方括号括起来的列表，而不能使用多个数值作为参数调用array。**

>>> a = array(1,2,3,4) # 错误

>>> a = array([1,2,3,4]) # 正确

可使**用双重序列来表示二维的数组，三重序列表示三维数组**，以此类推。

>>> b = array( [ (1.5,2,3), (4,5,6) ] )

>>> b

array([[ 1.5, 2. , 3. ],

　　　 [ 4. , 5. , 6. ]])

可以在创建时显式指定数组中元素的类型

>>> c = array( [ [1,2], [3,4] ], dtype=complex)

>>> c

array([[ 1.+0.j, 2.+0.j],

　　　 [ 3.+0.j, 4.+0.j]])

通常，刚开始时数组的元素未知，而数组的大小已知。因此，NumPy提供了一些使用占位符创建数组的函数。这些函数有助于满足除了数组扩展的需要，同时降低了高昂的运算开销。

**用函数zeros可创建一个全是0的数组，用函数ones可创建一个全为1的数组，函数empty创建一个内容随机并且依赖与内存状态的数组。默认创建的数组类型(dtype)都是float64。**

**2.2、NumPy提供一个类似arange的函数返回一个数列形式的数组:**

>>> arange(10, 30, 5)

array([10, 15, 20, 25])

**以10开始，差值为5的等差数列。该函数不仅接受整数，还接受浮点参数：**

>>> arange(0,2,0.5)

array([ 0. , 0.5, 1. , 1.5])

**arrange(n)表示生成一个从0到n-1的数组,reshape表示遍历顺序**

>>> a = arange(6) # 1d array

>>> print a

[0 1 2 3 4 5]

>>> b = arange(12).reshape(4,3) # 2d array

>>> print b

[[ 0 1 2]

[ 3 4 5]

[ 6 7 8]

[ 9 10 11]]

>>> c = arange(24).reshape(2,3,4) # 3d array

>>> print c

[[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

[[12 13 14 15]

[16 17 18 19]

[20 21 22 23]]]

**2.3、当arange使用浮点数参数时，由于浮点数精度有限，通常无法预测获得的元素个数。因此，最好使用函数linspace去接收我们想要的元素个数来代替用range来指定步长。linespace用法如下，将在通用函数一节中详细介绍。**

>>> numpy.linspace(-1, 0, 5) #5 表示个数。而在arange中5表示步长

array([-1. , -0.75, -0.5 , -0.25, 0. ])

**3、 创建二维数组 array() ：**

set = array([[1., 2, ],[3., 4.],[5., 6.],[7., 9.]]) %记住是两个[[与]]。

**ndarray.ndim：数组的维数（即数组轴的个数），等于秩。最常见的为二维数组（矩阵）。**

**ndarray.shape：数组的维度。为一个表示数组在每个维度上大小的整数元组。例如二维数组中，表示数组的“行数”和“列数”。ndarray.shape返回一个元组，这个元组的长度就是维度的数目，即ndim属性。**

求 数组的 行数：

>>> set.shape[0]  
4

求 数组的列数：

>>> set.shape[1]

2

>>> set.shape  
(4, 2)

**ndarray.size：数组元素的总个数，等于shape属性中元组元素的乘积。**

**ndarray.dtype：表示数组中元素类型的对象，可使用标准的Python类型创建或指定dtype。另外也可使用前一篇文章中介绍的NumPy提供的数据类型。**

>>> set.dtype  
dtype('float64')

>>> a.dtype

dtype('int32')

>>> b = array([1.2, 3.5, 5.1])

>>> b.dtype

dtype('float64')

**ndarray.itemsize：数组中每个元素的字节大小。例如，一个元素类型为float64的数组itemsiz属性值为8(float64占用64个bits，每个字节长度为8，所以64/8，占用8个字节），又如，一个元素类型为complex32的数组item属性为4（32/8）。**

**ndarray.data：包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。**

#### 记得 >>> from numpy import \*

2  empty() 函数：

>>> a = empty([2, 2])

>>> a

array([[ 2.01269048e-313, 4.44659081e-323],

[ 5.03965339e+223, 6.48588014e-310]])

>>> b = empty([2, 2], dtype = int)

>>> b

array([[19988563, 36534944],

[ 2460004, 2460004]])

>>> c = empty([2, 2], dtype = int, order = 'C')

>>> c

array([[19988581, 36534944],

[ 2460004, 2460004]])

>>> d = empty([2, 2], dtype = int ,order = 'F')

>>> d

array([[19857521, 2460004],

[36534944, 2460004]])

>>>

最后一个参数，返回数组在内存中的存放顺序，

C代表C语言风格， row major

F代表····，column  major

3.  eye（）

>>> e1 = eye(2, 3, 0, dtype = int)

>>> e1

array([[1, 0, 0],

[0, 1, 0]])

>>> e2 = eye(3)

>>> e2

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

>>> e3 = eye(3, dtype = int)

>>> e3

array([[1, 0, 0],

[0, 1, 0],

[0, 0, 1]])

>>> e4 = eye(3, 1, dtype = int)

>>> e4

array([[1],

[0],

[0]])

>>> e5 = eye(3, k = 1, dtype = int)

>>> e5

array([[0, 1, 0],

[0, 0, 1],

[0, 0, 0]])

>>> e6 = eye(3, k = -1, dtype = int)

>>> e6

array([[0, 0, 0],

[1, 0, 0],

[0, 1, 0]])

>>>

**第一个参数N = 列数**

**第二个参数 M = 行数，省略代表M = N**

**第三个参数 k 代表对角线位置， = 0 代表主对角线， +1就向右上方偏移1， -1 就向左下角偏移1**

**第四个参数表示类型 dtype 默认为 float 类型**

4 。 创建 方阵 identity（）

>>> i1 = identity(3)

>>> i1

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

>>> i2 = identity(3, dtype = int)

>>> i2

array([[1, 0, 0],

[0, 1, 0],

[0, 0, 1]])

>>>

只有两个参数，第一个表示 行（列）数，第二个表示类型（默认为float）类型

5.  生成一个元素全为1的数组

>>> o1 = ones(3)

>>> o1

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

>>> o1.shape

(3,)

要指定完整的shape（完整的行数和列数）的话：

>>> o4 = ones( (2, 3), dtype = int)

>>> o4

array([[1, 1, 1],

[1, 1, 1]])

>>>

6. zeros（） 全是0 的矩阵

>>> from numpy import \*

>>> z1 = zeros(3)

>>> z1

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

>>> z1.shape

(3,)

**4、 NumPy中的基本数据类型**

名称 描述

bool 用一个字节存储的布尔类型（True或False）

inti 由所在平台决定其大小的整数（一般为int32或int64）

int8 一个字节大小，-128 至 127

int16 整数，-32768 至 32767

int32 整数，-2 \*\* 31 至 2 \*\* 32 -1

int64 整数，-2 \*\* 63 至 2 \*\* 63 - 1

uint8 无符号整数，0 至 255

uint16 无符号整数，0 至 65535

uint32 无符号整数，0 至 2 \*\* 32 - 1

uint64 无符号整数，0 至 2 \*\* 64 - 1

float16 半精度浮点数：16位，正负号1位，指数5位，精度10位

float32 单精度浮点数：32位，正负号1位，指数8位，精度23位

float64或float 双精度浮点数：64位，正负号1位，指数11位，精度52位

complex64 复数，分别用两个32位浮点数表示实部和虚部

complex128或complex 复数，分别用两个64位浮点数表示实部和虚部

NumPy类型转换方式如下：

>>> float64(42)

42.0

>>> int8(42.0)

42

>>> bool(42)

True

>>> bool(42.0)

True

>>> float(True)

1.0

**5、自定义结构数组**

通过NumPy也可以定义像C语言那样的结构类型。在NumPy中定义结构的方法如下：

定义结构类型名称；定义字段名称，标明字段数据类型。

student= dtype({'names':['name', 'age', 'weight'], 'formats':['S32', 'i','f']}, align = True)

这里student是自定义结构类型的名称，使用dtype函数创建，在第一个参数中，'names'和'formats'不能改变，names中列出的是结构中字段名称，formats中列出的是对应字段的数据类型。S32表示32字节长度的字符串，i表示32位的整数，f表示32位长度的浮点数。最后一个参数为True时，表示要求进行内存对齐。

在定义好结构类型之后，就可以定义以该类型为元素的数组了：

a= array([(“Zhang”, 32, 65.5), (“Wang”, 24, 55.2)], **dtype =student**)

除了在每个元素中依次列出对应字段的数据外，还需要在array函数中最后一个参数指定其所对应的数据类型。

1. **组合函数（函数有hstack、vstack、dstack、row\_stack、column\_stack ）**

这里介绍以不同的方式组合函数。首先创建两个数组：

>>> a = arange(9).reshape(3,3)

>>> a

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

>>> b = 2 \* a

>>> b

array([[ 0, 2, 4],

[ 6, 8, 10],

[12, 14, 16]])

**6.1、水平组合**

>>> hstack((a, b))

array([[ 0, 1, 2, 0, 2, 4],

[ 3, 4, 5, 6, 8, 10],

[ 6, 7, 8, 12, 14, 16]])

也可通过concatenate函数并指定相应的轴来获得这一效果：

>>> concatenate((a, b), axis=1)

array([[ 0, 1, 2, 0, 2, 4],

[ 3, 4, 5, 6, 8, 10],

[ 6, 7, 8, 12, 14, 16]])

**6.2、垂直组合**

>>> vstack((a, b))

array([[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8],

[ 0, 2, 4],

[ 6, 8, 10],

[12, 14, 16]])

同样，可通过concatenate函数，并指定相应的轴来获得这一效果。

>>> concatenate((a, b), axis=0)

array([[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8],

[ 0, 2, 4],

[ 6, 8, 10],

[12, 14, 16]])

**6.3、深度组合**

另外，还有深度方面的组合函数dstack。顾名思义，就是在数组的第三个轴（即深度）上组合。如下：

>>> dstack((a, b))

array([[[ 0, 0],

[ 1, 2],

[ 2, 4]],

[[ 3, 6],

[ 4, 8],

[ 5, 10]],

[[ 6, 12],

[ 7, 14],

[ 8, 16]]])

仔细观察，发现对应的元素都组合成一个新的列表，该列表作为新的数组的元素。

**6.4、行组合**

行组合可将多个一维数组作为新数组的每一行进行组合：

>>> one = arange(2)

>>> one

array([0, 1])

>>> two = one + 2

>>> two

array([2, 3])

>>> row\_stack((one, two))

array([[0, 1],

[2, 3]])

对于2维数组，其作用就像垂直组合一样。

**6.5、列组合**

列组合的效果应该很清楚了。如下：

>>> column\_stack((oned, twiceoned))

array([[0, 2],

[1, 3]])

对于2维数组，其作用就像水平组合一样。

**7、分割数组（hsplit、vsplit、dsplit和split）**

在NumPy中，分割数组的函数有hsplit、vsplit、dsplit和split。可将数组分割成相同大小的子数组，或指定原数组分割的位置。

**7.1、水平分割**

>>> a = arange(9).reshape(3,3)

>>> a

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

>>> hsplit(a, 3)

[array([[0],

[3],

[6]]),

array([[1],

[4],

[7]]),

array([[2],

[5],

[8]])]

也调用split函数并指定轴为1来获得这样的效果：

split(a, 3, axis=1)

**7.2、垂直分割**

垂直分割是沿着垂直的轴切分数组：

>>> vsplit(a, 3)

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

同样，也可通过split函数并指定轴为1来获得这样的效果：

>>> split(a, 3, axis=0)

**7.3、面向深度的分割**

dsplit函数使用的是面向深度的分割方式：

>>> c = arange(27).reshape(3, 3, 3)

>>> c

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],

[24, 25, 26]]])

>>> dsplit(c, 3)

[array([[[ 0],

[ 3],

[ 6]],

[[ 9],

[12],

[15]],

[[18],

[21],

[24]]]),

array([[[ 1],

[ 4],

[ 7]],

[[10],

[13],

[16]],

[[19],

[22],

[25]]]),

array([[[ 2],

[ 5],

[ 8]],

[[11],

[14],

[17]],

[[20],

[23],

[26]]])]

**8、复制和镜像（View）**

　　当运算和处理数组时，它们的数据有时被拷贝到新的数组有时不是。这通常是新手的困惑之源。这有三种情况:

**8.1、完全不复制**

　　简单的赋值，而不复制数组对象或它们的数据。

>>> a = arange(12)

>>> b = a #不创建新对象

>>> b is a # a和b是同一个数组对象的两个名字

True

>>> b.shape = 3,4 #也改变了a的形状

>>> a.shape

(3, 4)

## 8.2视图(view)和浅复制

　　　不同的数组对象分享同一个数据。视图方法创造一个新的数组对象指向同一数据。

>>> c = a.view()

>>> c is a

False

>>> c.base is a #c是a持有数据的镜像

True

>>> c.flags.owndata

False

>>>

>>> c.shape = 2,6 # a的形状没变

>>> a.shape

(3, 4)

>>> c[0,4] = 1234 #a的数据改变了

>>> a

array([[ 0, 1, 2, 3],

[1234, 5, 6, 7],

[ 8, 9, 10, 11]])

## 8.3、切片数组返回它的一个视图：

>>> s = a[ : , 1:3] # 获得每一行1，2处的元素

>>> s[:] = 10 # s[:] 是s的镜像。注意区别s=10 and s[:]=10

>>> a

array([[ 0, 10, 10, 3],

[1234, 10, 10, 7],

[ 8, 10, 10, 11]])

## 8.4、深复制

 　　这个复制方法完全复制数组和它的数据。

>>> d = a.copy() #创建了一个含有新数据的新数组对象

>>> d is a

False

>>> d.base is a #d和a现在没有任何关系

False

>>> d[0,0] = 9999

>>> a

array([[ 0, 10, 10, 3],

[1234, 10, 10, 7],

[ 8, 10, 10, 11]])

# Numpy扩充数组函数之repeat和tile用法

用repeat和tile扩充数组元素，例如

>>> import numpy as np

>>> np.arange(10)

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

>>> a = np.arange(10)

>>> a

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

>>> a.repeat(5)

array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4,

4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 9,

9, 9, 9, 9])

>>> a

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

>>> np.tile(a,2)

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

>>>

官方例子：

>>> from numpy import \*

>>> a=array([10,20])

>>> tile(a,(3,2)) #构造3X2个copy

array([[10, 20, 10, 20],

[10, 20, 10, 20],

[10, 20, 10, 20]])

>>> tile(42.0,(3,2))

array([[ 42., 42.],

[ 42., 42.],

[ 42., 42.]])

>>>

>>> from numpy import \*

>>> repeat(7.,4)

array([ 7., 7., 7., 7.])

>>> a=array([10,20])

>>> a.repeat([3,2])

array([10, 10, 10, 20, 20])

>>> repeat(a,[3,2])

array([10, 10, 10, 20, 20])

>>> a=array([[10,20],[30,40]])

>>> a.repeat([3,2],axis=0)

array([[10, 20],

[10, 20],

[10, 20],

[30, 40],

[30, 40]])

>>> a.repeat([3,2],axis=1)

array([[10, 10, 10, 20, 20],

[30, 30, 30, 40, 40]])

>>>

1. **排序（使用argsort，sort，sorted）**

**10.1 argsort的用法**

从中可以看出argsort函数返回的是数组值从小到大的索引值

One dimensional array:一维数组  
>>> x = np.array([3, 1, 2])  
>>> np.argsort(x)  
array([1, 2, 0])  
Two-dimensional array:二维数组  
>>> x = np.array([[0, 3], [2, 2]])  
>>> np.argsort(x, axis=0) #按列排序  
array([[0, 1],  
[1, 0]])  
>>> np.argsort(x, axis=1) #按行排序  
array([[0, 1],  
[0, 1]])

#######################################

例1：

>>> x = np.array([3, 1, 2])  
>>> np.argsort(x) #按升序排列  
array([1, 2, 0])  
>>> np.argsort(-x) #按降序排列  
array([0, 2, 1])

>>> x[np.argsort(x)] #通过索引值排序后的数组  
array([1, 2, 3])  
>>> x[np.argsort(-x)]  
array([3, 2, 1])

另一种方式实现按降序排序：

>>> a = x[np.argsort(x)]  
>>> a  
array([1, 2, 3])  
>>> a[::-1]  
array([3, 2, 1])

**10.2、sort, sorted, itemgetter的用法**

一、介绍

sort函数是list列表中的函数，而sorted可以对list或者iterator进行排序

二、sort和sorted的比较

1、用sort函数对列表排序时会影响列表本身，而sorted不会

举例：

>>> a = [1,2,1,4,3,5]  
>>> a.sort()  
>>> a  
[1, 1, 2, 3, 4, 5]

>>> a = [1,2,1,4,3,5]  
>>> sorted(a)  
[1, 1, 2, 3, 4, 5]  
>>> a  
[1, 2, 1, 4, 3, 5]

**2、sorted(iterable，cmp，key，reverse）**

参数：iterable可以是list或者iterator；

cmp是带两个参数的比较函数；

key是带一个参数的函数；

reverse为False或者True；

reverse：排序规则. reverse = True 或者 reverse = False，默认False（从小到大）。

举例说明

**（1）用cmp函数排序**

>>> list1 = [('david', 90), ('mary',90), ('sara',80),('lily',95)]  
>>> sorted(list1,cmp = lambda x,y: cmp(x[0],y[0]))  
[('david', 90), ('lily', 95), ('mary', 90), ('sara', 80)]  
>>> sorted(list1,cmp = lambda x,y: cmp(x[1],y[1]))  
[('sara', 80), ('david', 90), ('mary', 90), ('lily', 95)]

（2）用key函数排序

>>> list1 = [('david', 90), ('mary',90), ('sara',80),('lily',95)]  
>>> sorted(list1,key = lambda list1: list1[0])  
[('david', 90), ('lily', 95), ('mary', 90), ('sara', 80)]  
>>> sorted(list1,key = lambda list1: list1[1])  
[('sara', 80), ('david', 90), ('mary', 90), ('lily', 95)]

**（3）用reverse排序**

>>> sorted(list1,reverse = True)  
[('sara', 80), ('mary', 90), ('lily', 95), ('david', 90)]

**（4）用operator.itemgetter函数排序**

>>> from operator import itemgetter  
>>> sorted(list1, key=itemgetter(1))  
[('sara', 80), ('david', 90), ('mary', 90), ('lily', 95)]  
>>> sorted(list1, key=itemgetter(0))  
[('david', 90), ('lily', 95), ('mary', 90), ('sara', 80)]

介绍operator.itemgetter函数

>>> import operator  
>>> a = [1,2,3]  
>>> b = operator.itemgetter(0)  
>>> b(a)  
1

operator.itemgetter函数获取的不是值，而是定义了一个函数。

**（5）sorted(iterable, cmp=None, key=None, reverse=False)**

说明：内置的排序函数，对list，字典等等可以使用

iterable：是可迭代类型;  
cmp：用于比较的函数，比较什么由key决定,有默认值，迭代集合中的一项;  
key：用列表元素的某个属性和函数进行作为关键字，有默认值，迭代集合中的一项;  
reverse：排序规则. reverse = True 或者 reverse = False，默认False（从小到大）。  
返回值：是一个经过排序的可迭代类型，与iterable一样;

例如：b是一个字典

b：

{'a': 2, 'c': 1, 'b': 3}

对b进行排序：

>>c=sorted(b.iteritems(),key=operator.itemgetter(1),reverse=False)

>>print c

[('c', 1), ('a', 2), ('b', 3)]

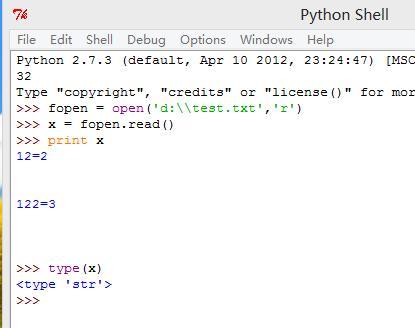
可见：返回的是一个list

**（6）多级排序(itemgetter(0,1))**

>>> sorted(list1, key=itemgetter(0,1))  
[('david', 90), ('lily', 95), ('mary', 90), ('sara', 80)]

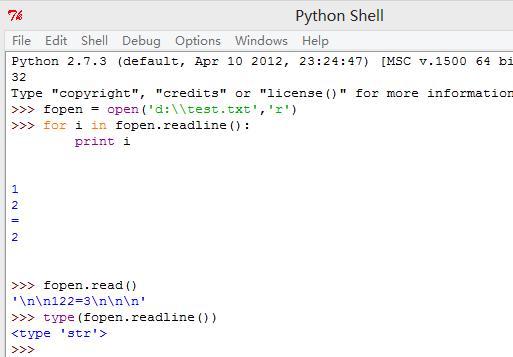
# python中read() readline()以及readlines()区别

**11.1、.read() 每次读取整个文件，它通常将读取到底文件内容放到一个字符串变量中，也就是说 .read() 生成文件内容是一个字符串类型，如下图；**

****

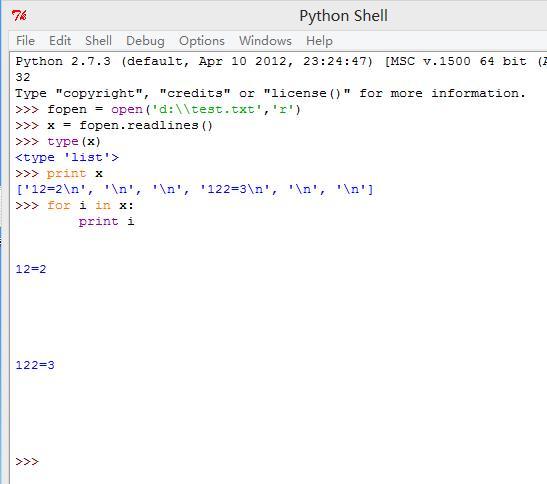
IMG_256IMG_256

**11.2、** **.readline()每只读取文件的一行，通常也是读取到的一行内容放到一个字符串变量中，返回str类型，如下图；**

****

从for循环中我们可以看出.readline()只读取了文件中的一行内容。

**11.3、 .readlines()每次按行读取整个文件内容，将读取到的内容放到一个列表中，返回list类型，如下图。**

****

# strip()函数 介绍

函数原型

声明：s为字符串，rm为要删除的字符序列

s.strip(rm) 删除s字符串中开头、结尾处，位于 rm删除序列的字符

s.lstrip(rm) 删除s字符串中开头处，位于 rm删除序列的字符

s.rstrip(rm) 删除s字符串中结尾处，位于 rm删除序列的字符

注意：

1. 当rm为空时，默认删除空白符（包括'\n', '\r', '\t', ' ')

例如：

>>> a = ' 123'  
>>> a.strip()  
'123'  
>>> a='\t\tabc'  
'abc'  
>>> a = 'sdff\r\n'  
>>> a.strip()  
'sdff'

2.这里的rm删除序列是只要边（开头或结尾）上的字符在删除序列内，就删除掉。

例如 ：

代码如下:

>>> a = '123abc'  
>>> a.strip('21')  
'3abc' 结果是一样的  
>>> a.strip('12')  
'3abc'

# ****13、详解Python中的split()函数的使用方法****

函数：split()

Python中有split()和os.path.split()两个函数，具体作用如下：  
split()：拆分字符串。通过指定分隔符对字符串进行切片，并返回分割后的字符串列表（list）  
os.path.split()：按照路径将文件名和路径分割开

**一、函数说明**1、split()函数  
语法：str.split(str="",num=string.count(str))[n]

参数说明：  
str： 表示为分隔符，默认为空格，但是不能为空('')。若字符串中没有分隔符，则把整个字符串作为列表的一个元素  
num：表示分割次数。如果存在参数num，则仅分隔成 num+1 个子字符串，并且每一个子字符串可以赋给新的变量  
[n]： 表示选取第n个分片

注意：当使用空格作为分隔符时，对于中间为空的项会自动忽略

2、os.path.split()函数  
语法：os.path.split('PATH')

参数说明：

1. PATH指一个文件的全路径作为参数：
2. 如果给出的是一个目录和文件名，则输出路径和文件名
3. 如果给出的是一个目录名，则输出路径和为空文件名

1、常用实例

1. >>> u = "www.doiido.com.cn"
3. #使用默认分隔符
4. >>> print u.split()
5. ['www.doiido.com.cn']
7. #以"."为分隔符
8. >>> print u.split('.')
9. ['www', 'doiido', 'com', 'cn']
11. #分割0次
12. >>> print u.split('.',0)
13. ['www.doiido.com.cn']
15. #分割一次
16. >>> print u.split('.',1)
17. ['www', 'doiido.com.cn']
19. #分割两次
20. >>> print u.split('.',2)
21. ['www', 'doiido', 'com.cn']
23. #分割两次，并取序列为1的项
24. >>> print u.split('.',2)[1]
25. doiido
27. #分割最多次（实际与不加num参数相同）
28. >>> print u.split('.',-1)
29. ['www', 'doiido', 'com', 'cn']
31. #分割两次，并把分割后的三个部分保存到三个文件
32. >>> u1,u2,u3 = u.split('.',2)
33. >>> print u1
34. www
35. >>> print u2
36. doiido
37. >>> print u3
38. com.cn
39. 2、去掉换行符
40. >>> c = '''say
41. hello
42. baby'''
44. >>> print c
45. say
46. hello
47. baby

>>> print c.split('\n')

1. ['say', 'hello', 'baby']
2. 3、分离文件名和路径

>>> import os

>>> print os.path.split('/dodo/soft/python/')

('/dodo/soft/python', '')

>>> print os.path.split('/dodo/soft/python')

('/dodo/soft', 'python')

1. 4、一个超级好的例子

>>> str="hello boy<[www.doiido.com]>byebye"

>>> print str.split("[")[1].split("]")[0]

www.doiido.com

>>> print str.split("[")[1].split("]")[0].split(".")

['www', 'doiido', 'com']

# ****14、Python使用os.listdir()函数来得目录内容（os.mkdir()函数创建目录,os.rmdir()函数删除目录）****

Python编程语言如何获得目录中的内容

在Python中可以使用os.listdir()函数获得指定目录中的内容。其原型如下所示。

os.listdir(path)

其参数含义如下。path 要获得内容目录的路径。以下实例获得当前目录的内容。

>>> import os

>>> os.listdir(os.getcwd())

获得当前目录中的内容

['dde.pyd', 'license.txt', 'Pythonwin.exe',  'scintilla.dll', 'win32ui.pyd', 'win32uiole.pyd', 'pywin']

如何用Python编程语言创建目录

在Python中可以使用os.mkdir()函数创建目录。其原型如下所示。

os.mkdir(path)

其参数含义为。 path 要创建目录的路径。以下的实例将在E:\book目录下创建temp目录。

>>> import os

>>> os.mkdir('E:\\book\\temp')

使用os.mkdir创建目录

删除目录

在Python中可以使用os.rmdir()函数删除目录。其原型如下所示。

os.rmdir(path)

其参数含义如下。path 要删除的目录的路径。以下实例删除E:\book\temp目录。

>>> import os

>>> os.rmdir('E:\\book\\temp')

删除目录需要说明的是，使用os.rmdir删除的目录必须为空目录，否则函数出错。以上的内容就是对Python编程语言如何获得目录中的内容，创建目录，还有如何用Python编程语言删除目录的相关内容的介绍。

# ****[python数据持久存储：pickle模块的基本使用](http://www.cnblogs.com/pzxbc/archive/2012/03/18/2404715.html)****

　python的pickle模块实现了基本的数据序列和反序列化。通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储；通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

　　基本接口：

　　pickle.dump(obj, file, [,protocol])  
　　注解：将对象obj保存到文件file中去。  
　　protocol为序列化使用的协议版本，0：ASCII协议，所序列化的对象使用可打印的ASCII码表示；1：老式的二进制协议；2：2.3版本引入的新二进制协议，较以前的更高效。其中协议0和1兼容老版本的python。protocol默认值为0。  
　　file：对象保存到的类文件对象。file必须有write()接口， file可以是一个以'w'方式打开的文件或者一个StringIO对象或者其他任何实现write()接口的对象。如果protocol>=1，文件对象需要是二进制模式打开的。

　　pickle.load(file)  
　　注解：从file中读取一个字符串，并将它重构为原来的python对象。  
　　file:类文件对象，有read()和readline()接口。

以下为代码示例：

#使用pickle模块将数据对象保存到文件  
import pickle  
data1 = {'a': [1, 2.0, 3, 4+6j],  
 'b': ('string', u'Unicode string'),  
 'c': None}  
selfref\_list = [1, 2, 3]  
selfref\_list.append(selfref\_list)  
output = open('data.pkl', 'wb')  
# Pickle dictionary using protocol 0.  
pickle.dump(data1, output)  
# Pickle the list using the highest protocol available.  
pickle.dump(selfref\_list, output, -1)  
output.close()

# ****arange 与range的区别****

a = range(5)

print(a)

print(type(a))

[0, 1, 2, 3, 4]

<type 'list'>

import numpy as np

b = np.arange(5)

print(b)

print(type(b))

[0 1 2 3 4]

<type 'numpy.ndarray'>

首先，range是python自带的函数，无需引用函数库，而arange是numpy编程库里的函数，使用时要引用numpy函数库

其次，range返回的是一个数据组成的列表list，而arange返回的是numpy函数库中的ndarray数组。区别一方面可以从数值中看出来，a=[0, 1, 2, 3, 4]，每个数值之间是带逗号的，所以是列表list，b=[0 1 2 3 4]不带逗号，是数值计算属于numpy中的数组ndarray

这里list数组可以通过np.array()转换成ndarray数组：

f = np.array(a)

print(f)

print(type(f))

[0 1 2 3 4]

<type 'numpy.ndarray'>

这里你会发现f与b的数值以及类型是一样的，因此通过nump函数库中的ndarray()可以将列表list转换成ndarray数组

当然，若果转换成矩阵来进行相应数值计算的话，不必要从list转换成ndarray，直接转换成矩阵即可。

e = np.mat(a)

print(e)

print(type(e))

[[0 1 2 3 4]]

<class 'numpy.matrixlib.defmatrix.matrix'>

c = np.mat(b)

print(c)

print(type(c))

[[0 1 2 3 4]]

<class 'numpy.matrixlib.defmatrix.matrix'>

很明显两者是一样的，但这里需对c的元素进行相应的探究

print(c)

print(type(c))

print(c[0])

print(type(c[0]))

print(c==c[0])

[[0 1 2 3 4]]

<class 'numpy.matrixlib.defmatrix.matrix'>

[[0 1 2 3 4]]

<class 'numpy.matrixlib.defmatrix.matrix'>

[[ True True True True True]]

c是矩阵，包含的元素也相应的是一个矩阵，c可看成一个一行五列的二维矩阵，c[0]是第一行包含的矩阵元素，只不过也是一个二维矩阵

print(np.shape(c))

print(np.shape(c[0]))

（1, 5）

（1, 5）

c[0,3]表示的是第一行第三列的元素，如果只有一个输入元素，表示的是行，所以c[1]超出了范围range，包含的数据类型是numpy库中的numpy,int型

print(c[0,3])

print(type(c[0,3]))

print(type(c[1]))

3

<type 'numpy.int32'>

Traceback (most recent call last):

File "type.py", line 23, in <module>

print(type(c[1]))

File "C:\Users\Administrator\AppData\Roaming\Python\Python27\site-packages\num

py\matrixlib\defmatrix.py", line 318, in \_\_getitem\_\_

out = N.ndarray.\_\_getitem\_\_(self, index)

IndexError: index 1 is out of bounds for axis 0 with size 1

# [****python numpy中nonzero()的用法****](http://blog.csdn.net/xiaolewennofollow/article/details/44985907)

当使用布尔数组直接作为下标对象或者元组下标对象中有布尔数组时，都相当于用nonzero()将布尔数组转换成一组整数数组，然后使用整数数组进行下标运算。（http://blog.csdn.net/xiaolewennofollow/article/details/44985907）

nonzeros(a)返回数组a中值不为零的元素的下标，它的返回值是一个长度为a.ndim(数组a的轴数)的元组，元组的每个元素都是一个整数数组，其值为非零元素的下标在对应轴上的值。

例如对于一维布尔数组b1，nonzero(b1)所得到的是一个长度为1的元组，它表示b1[0]和b1[2]的值不为0(False)。

>>> b1 = np.array([True, False, True, False])

>>> np.nonzero(b1)

(array([0, 2]),)

对于二维数组b2，nonzero(b2)所得到的是一个长度为2的元组。它的第0个元素是数组a中值不为0的元素的第0轴的下标，第1个元素则是第1轴的下标，因此从下面的结果可知b2[0,0]、b[0,2]和b2[1,0]的值不为0：

>>> b2 = np.array([[True, False, True], [True, False, False]])

>>> np.nonzero(b2)

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

当布尔数组直接做为下标时，相当于使用由nonzero()转换之后的元组作为下标对象：

>>> a = np.arange(3\*4\*5).reshape(3,4,5)

**>>>** a[b2]array([[ 0, 1, 2, 3, 4],

[10, 11, 12, 13, 14],

[20, 21, 22, 23, 24]])

>>> a[np.nonzero(b2)]

array([[ 0, 1, 2, 3, 4],

[10, 11, 12, 13, 14],

[20, 21, 22, 23, 24]])

当下标对象是元组，并且其中有布尔数组时，相当于将布尔数组展开为由nonzeros()转换之后的各个整数数组：

>>> a[1:3, b2]

array([[20, 22, 25],

[40, 42, 45]])

>>> a[1:3, np.nonzero(b2)[0], np.nonzero(b2)[1]]

array([[20, 22, 25],

[40, 42, 45]])

# [****python numpy中np.ndarray()和np.asarray()****](http://blog.csdn.net/xiaolewennofollow/article/details/44985907)****的区别****

# ****元组怎么使用range：****

img = cv2.imread('IMG1.jpg', 0)

for row in range(img.shape[0]):

for column in range(img.shape[1]):

print(…)

tuple使用shape时生成的是列表而不是数组，不要用shape(0 or 1), 而要用中括号[].

# ****怎么将二维数组转化成列表：****

有两种方法，但不建议使用第一种。循环太慢：

import cv2

import numpy as np

img = cv2.imread('IMG1.jpg', 0)

chessboards = np.empty((1,76800), dtype='int8')

# 第一种 for循环

for row in range(img.shape[0]):

for column in range(img.shape[1]):

chessboards[0] = np.ndarray.flatten(img[:,:])

#第二种 整体转换

print(img.reshape(76800))

z = chessboards[0] == img.reshape(76800)

if False in z:

print("!!!!!")

结果为： !!!!!