import numpy as np

数组 np.array()

获取np的版本：

np.\_\_vision\_\_

获取np的配置：

np.show\_config()

浮点数和整数比较、浮点数之间比较，用isclose、allclose函数。

## 部分通用参数

### 1.order

* order：'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'K' -- 元素在内存中的出现顺序。

## 生成数组：

### 方一：

lis=[1,2,3]

a=np.array(lis)

### 方二：

a=array([1,2,3])

### linespace方法：

a=np.linespace(0,1,4)

a.shape=2,2 #指定为2\*2矩阵。不加括号，默认为元组。

返回：

[[0. 0.33333333]

[0.66666667 1. ]]

即：从0到1的闭区间，平均分配4个数（包含0,1）

### 生成等差数组：

a=np.arange(0,80,10) arange（start,end,step）

a：array（[0,10,20,30,40,50,60,70]）

### 随机数组

**from** numpy.random **import** rand  
data = rand(100)

### 使用fill方法设定初始值

a.fill(-4.8)

输出：[-4 -4 -4 -4 -4 -4 -4 -4 -4 -4]

数组a的类型确定的情况下（即a.dtype），不能更改，填充的数值会自动转为该类型。此处，-4.8将会取整，为-4。即a中有10个-4。不管a是几维，都会被填满。

### frombuffer

numpy.frombuffer 用于实现动态数组。

输入字符串，生成单个字母数组。

numpy.frombuffer 接受 buffer 输入参数，以流的形式读入转化成 ndarray 对象。

numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)

**注意：**buffer 是字符串的时候，Python3 默认 str 是 Unicode 类型，所以要转成 bytestring 在原 str 前加上 b。

参数：

buffer 可以是任意对象，会以流的形式读入。返回数组的数据类型，可选。

count 读取的数据数量，默认为-1，读取所有数据。

offset 读取的起始位置，默认为0。

import numpy as np

s = b'Hello World'

a = np.frombuffer(s, dtype = 'S1')

print (a)

输出结果为：

[b'H' b'e' b'l' b'l' b'o' b' ' b'W' b'o' b'r' b'l' b'd']

### fromiter

从迭代对象中生成数组:

numpy.fromiter

numpy.fromiter 方法从可迭代对象中建立 ndarray 对象，返回一维数组。

numpy.fromiter(iterable, dtype, count=-1)

参数：

iterable 可迭代对象

dtype 返回数组的数据类型

count 读取的数据数量，默认为-1，读取所有数据

实例

import numpy as np # 使用 range 函数创建列表对象

list=range(5)

it=iter(list) # 使用迭代器创建 ndarray

x=np.fromiter(it, dtype=float)

print(x)

输出结果为：

[0. 1. 2. 3. 4.]

## 存储与读入数组：

如下表所示：

文件格式|使用的包|函数

格式 使用的包 方法

txt | numpy | loadtxt, genfromtxt, fromfile, savetxt, tofile

csv | csv | reader, writer

Matlab | scipy.io | loadmat, savemat

hdf | pytables, h5py |

NetCDF | netCDF4, scipy.io.netcdf | netCDF4.Dataset, scipy.io.netcdf.netcdf\_file

\*\*文件格式\*\*|\*\*使用的包\*\* |\*\*备注\*\*

wav | scipy.io.wavfile | 音频文件

jpeg,png,... | PIL, scipy.misc.pilutil | 图像文件

fits | pyfits | 天文图像

此外， pandas ——一个用来处理时间序列的包中包含处理各种文件的方法，具体可参见它的文档：

<http://pandas.pydata.org/pandas-docs/stable/io.html>

### 1.loadtxt

loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)

comments：指定注释符号。默认为’#’，指定后，在遇到这个符号，则默认该行后面为注释。

converters：定义的转换方法，为一个字典或者func。见示例。

skiprows 参数表示忽略开头的行数，可以用来读写含有标题的文本

delimiter表示分隔符。默认为空格。

usecols：指定使用那几列数据。元组形式。如果有多列，只想要其中几列数据，可以使用本参数指定。第一列为0。

如：对于以空格和换行形式存入文本文件的数据：

2.1 2.3 3.2 1.3 3.1

6.1 3.1 4.2 2.3 1.8

data = np.loadtxt('myfile.txt')

array([[ 2.1, 2.3, 3.2, 1.3, 3.1],

[ 6.1, 3.1, 4.2, 2.3, 1.8]])

对于以逗号和换行存入文本的文件：

2.1, 2.3, 3.2, 1.3, 3.1

6.1, 3.1, 4.2, 2.3, 1.8

data = np.loadtxt('myfile.txt', delimiter=',')

array([[ 2.1, 2.3, 3.2, 1.3, 3.1],

[ 6.1, 3.1, 4.2, 2.3, 1.8]])

自定义转换方法：

文本文件：

2010-01-01 2.3 3.2

2011-01-01 6.1 3.1

import datetime

def date\_converter(s):

return datetime.datetime.strptime(s, "%Y-%m-%d")

data = np.loadtxt('myfile.txt',

dtype=np.object, #数据类型为对象

converters={0:date\_converter, #第一列使用自定义转换方法

1:float, #第二第三使用浮点数转换

2:float})

data：

array([[datetime.datetime(2010, 1, 1, 0, 0), 2.3, 3.2],

[datetime.datetime(2011, 1, 1, 0, 0), 6.1, 3.1]], dtype=object)

### 2.genfromtxt

此外，有一个功能更为全面的 genfromtxt 函数，能处理更多的情况，但相应的速度和效率会慢一些。

genfromtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None,

skiprows=0, skip\_header=0, skip\_footer=0, converters=None,

missing='', missing\_values=None, filling\_values=None, usecols=None,

names=None, excludelist=None, deletechars=None, replace\_space='\_',

autostrip=False, case\_sensitive=True, defaultfmt='f%i', unpack=None,

usemask=False, loose=True, invalid\_raise=True)

### 3.savetxt

savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ')

默认分割符为空格，也可以指定分隔符。

savetxt 可以将数组写入文件txt，默认使用科学计数法的形式保存：

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

np.savetxt('out.txt', data)

在文本中保存为：科学计数法形式

1.000000000000000000e+00 2.000000000000000000e+00

3.000000000000000000e+00 4.000000000000000000e+00

也可以使用类似\*\*C\*\*语言中 printf 的方式指定输出的格式：

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

np.savetxt('out.txt', data, fmt="%d") #保存为整数

再次读入时，就会显示为整数。

如：%.2f为保留两位小数；等等

复数值默认会加上括号：

data = np.array([[1+1j,2], [3,4]])

np.savetxt('out.txt', data, fmt="%.2f", delimiter=',') #保存为2位小数的浮点数，用逗号分隔

在文本中保存为：

(1.00+1.00j), (2.00+0.00j)

(3.00+0.00j), (4.00+0.00j)

### 4. 二进制格式

数组可以储存成二进制格式，单个的数组保存为 .npy 格式，多个数组保存为多个 .npy 文件组成的 .npz 格式，每个 .npy 文件包含一个数组。

与文本格式不同，二进制格式保存了数组的 shape, dtype 信息，以便完全重构出保存的数组。

二进制文件大概是文本文件的三分之一

保存的方法：

- save(file, arr) 保存单个数组， .npy 格式

- savez(file, \*args, \*\*kwds) 保存多个数组，无压缩的 .npz 格式

- savez\_compressed(file, \*args, \*\*kwds) 保存多个数组，有压缩的 .npz 格式

读取的方法：

- load(file, mmap\_mode=None) 对于 .npy ，返回保存的数组，对于 .npz ，返回一个名称-数组对组成的字典。

#### 保存多个数组

a = np.array([[1.0,2.0],

[3.0,4.0]])

b = np.arange(1000)

np.savez('data.npz', a=a, b=b)

注意，参数中前面a= 和b= 是保存入文件里面的变量名，也是重新读出时，字典中保存的key值，不能重复。

载入数据：

data = np.load('data.npz')

载入后可以像字典一样进行操作：

data.keys()

data['a']

输出：['a', 'b']

array([[ 1., 2.],

[ 3., 4.]])

注意：删除文件时，需要先删除data，不然会报错。

#### 压缩文件

当数组比较整齐时，压缩率较高，否则比较低。

### 将数组存入dat二进制文件：

a = array([102,111,212],   
 dtype=np.uint8)  
 a.tofile(**'foo.dat'**)

### 从二进制文件中读入数组：

a=np.fromfile(‘foo.dat’,dtype=np.uint8)

读入时，如果不指定类型，或者指定类型不对，将不能读入。

## 数组属性

ndim/shape/size/dtype/itemsize

### 1.类型：

type(a): numpy.ndarray

### 2.查看数组中的数据类型

a.dtype

### 3.查看每个元素所占的字节

a.itemsize

### 4. 查看形状

会返回一个元组，每个元素代表这一维的元素数目：

a.shape 返回一个元组。如（3,3,3）代表一个3\*3\*3的3维数组

或者 np.shape(a)

shape 的使用历史要比 a.shape 久，而且还可以作用于别的类型：

如用于列表。

### 5. 查看元素数目：

np.size(a)

返回一个整数

### 6. 查看所有元素所占的空间：

a.nbytes

但事实上，数组所占的存储空间要比这个数字大，因为要用一个header来保存shape，dtype这样的信息。

### 7. 查看数组维数：

a.ndim

返回一个整数

二维数组是数组里面包含多个一维数组，且里面所有一维数组都是一样长的。三维数组是数组里面包含多个二维数组，且里面所有二维数组都是一样长的。

### 8. 使用fill方法设定初始值

a.fill(-4.8)

输出：[-4 -4 -4 -4 -4 -4 -4 -4 -4 -4]

数组a的类型确定的情况下（即a.dtype），不能更改，填充的数值会自动转为该类型。此处，-4.8将会取整，为-4。即a中有10个-4

### 9.索引和切片

和列表一样

## 多维数组及其属性

### 1.生成：

a = array([[ 0, 1, 2, 3],  
 [10,11,12,13]])

或者

lis=[[ 0, 1, 2, 3], [10,11,12,13]]

a=np.array(lis)

类似可以生成三维、n维数组。

### 查看数组形状

a.shap:返回元组，第一个为行数，第二个为列数。

如：(2L, 4L) 2行4列

### a.size

### a.ndim

### 多维数组索引

#### 二维数组：

对于二维数组，可以传入两个数字来索引：行索引从0开始，列索引也是从0开始。

a[1,3]：代表2行4列 a24

a[1]：代表选取第2行

#### 多维数组切片：

a = array([[ 0, 1, 2, 3, 4, 5],  
 [10,11,12,13,14,15],  
 [20,21,22,23,24,25],  
 [30,31,32,33,34,35],  
 [40,41,42,43,44,45],  
 [50,51,52,53,54,55]])

a[0, 3:5] ：得到第1行的4、5两个元素。

a[4:, 4:] ：得到第5行至最后一行的每行的第5个至最后的元素。

a[2::2, ::2]：初始索引、步长、结束索引

切片在内存中使用的是引用机制。引用机制意味着，\*\*Python\*\*并没有为 b 分配新的空间来存储它的值，而是让 b 指向了 a 所分配的内存空间，因此，改变 b 会改变 a 的值。而列表则不会。

### 使数组只读

data.flags.writeable=False

## 花式索引：

与切片不同，花式索引返回的是原对象的一个复制而不是引用。

### 生成等差数组：

a=np.arange(0,80,10) arange（start,end,step）

a：array（[0,10,20,30,40,50,60,70]）

### 一维数组：

花式索引需要指定索引位置：

indices = [1, 2, -3]  
y = a[indices] ：选取第2、3、倒数第3个元素

通过布尔数组来花式索引：

mask = array([0,1,1,0,0,1,0,0],dtype=bool) 注意，mask也为数组。

y=a[mask]

或者用布尔表达式生成 mask ，选出了所有大于0.5的值：

a=array([0.88742125, 0.06523344 ,0.49448717, 0.50531947, 0.23019701 ,0.53303566,

0.22763991 ,0.64319931 ,0.16774729 ,0.14242495])

mask = a > 0.5

a[mask]

mask 必须是和a同样大小的布尔数组。

### 二维数组

a = array([[ 0, 1, 2, 3, 4, 5],  
 [10,11,12,13,14,15],  
 [20,21,22,23,24,25],  
 [30,31,32,33,34,35],  
 [40,41,42,43,44,45],  
 [50,51,52,53,54,55]])

对于二维花式索引，我们需要给定 row 和 col 的值：

a[(0,1,2,3,4), (1,2,3,4,5)]

两个元组中的数组依次对应，分别指定行和列。返回（0,1）（1,2）（2,3）..次对角线上的数值。

a[3:, [0,2,5]]

返回从第四行开始，每行的列索引为0，2，5的数值。

也可以使用mask进行索引：

mask = array([1,0,1,0,0,1],  
 dtype=bool)  
a[mask, 2]

mask不仅可以指定行，也可以指定列。

“不完全”索引：

只给定行索引的时候，返回整行：

y = a[:3]

返回前三行

也可以使用花式索引取出第2，3，5行：

condition = array([0,1,1,0,1],  
 dtype=bool)  
a[condition]

### 三维花式索引

#### 生成三维数组：

a = arange(64)  
a.shape = 4,4,4

花式索引：和二维数组类似，只不过多了一维

y = a[:,:,[2, -1]]

取每行的第3个和最后一个元素

## 数组类型与转换

整数型、布尔型、浮点数型、复数型

### 复数数组

#### 生成：

a = array([1 + 1j, 2, 3, 4])

\*\*Python\*\*会自动判断数组的类型：

a.dtype 输出：dtype('complex128')

#### 查看实部和虚部

a.real 返回实部：array([ 1., 2., 3., 4.])

a.imag 返回虚部：array([ 1., 0., 0., 0.])

#### 设置实部或虚部的值：

a.imag = [1,2,3,4]

a 为 array([ 1.+1.j, 2.+2.j, 3.+3.j, 4.+4.j])

#### 共轭

a.conj()：返回一个共轭数组 array([ 1.-1.j, 2.-2.j, 3.-3.j, 4.-4.j])

上面所有的方法都可以应用在整型和浮点型，看成是虚部为0的数组。对于整型和浮点型，a.img 返回虚部都为0，这个虚部是只读的，不能修改和设置。

### 指定数组类型

之前已经知道，构建数组的时候，数组会根据传入的内容自动判断类型：

对于浮点型，默认为双精度，8字节。float64

a = array([0,1.0,2,3])

当然，我们也可以在构建的时候指定类型：

a = array([0,1.0,2,3],dtype =np.float32) :float32为单精度，4字节

指定无符号数组：

a = array([0,1,2,3],dtype=np.uint8)

uint8：单字节数，0-255

### 类型转换

#### asarray

np.asarray(a, np.dtype=float64)

返回一个新数组，不会修改原数组的值。但是要转换的类型与元类型相同时，会返回原数组。asarray还可以将非数组转换为数组。

asarray([1,2,3,4]) 将列表转换为数组。

#### astype 方法

a.astype(np.float64):返回一个新数组

#### view 方法

view会按照位数来进行转换数组。如为4byte，转换为uint8为1byte，就一个数转换为4个数。

a = array((1,2,3,4), dtype=int32)

b = a.view(uint8)

b

array([1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 4, 0, 0, 0], dtype=uint8)

由于是view视图，所以是b对a是引用关系，修改b会修改a

### 将数组存入dat二进制文件：

a = array([102,111,212],   
 dtype=np.uint8)  
 a.tofile(**'foo.dat'**)

### 从二进制文件中读入数组：

a=np.fromfile(‘foo.dat’,dtype=np.uint8)

读入时，如果不指定类型，或者指定类型不对，将不能读入。

### 清理数据文件：

**import** os  
os.remove(**'foo.dat'**)

0-255 的数字可以表示ASCⅡ码，我们可以用 ord 函数来查看字符的ASCⅡ码值：

ord(‘f’) : 102

### 类型总结

bool\_ 布尔型数据类型（True 或者 False）

int\_ 默认的整数类型（类似于 C 语言中的 long，int32 或 int64）

intc 与 C 的 int 类型一样，一般是 int32 或 int 64

intp 用于索引的整数类型（类似于 C 的 ssize\_t，一般情况下仍然是 int32 或 int64）

int8 字节（-128 to 127）

int16 整数（-32768 to 32767）

int32 整数（-2147483648 to 2147483647）

int64 整数（-9223372036854775808 to 9223372036854775807）

uint8 无符号整数（0 to 255）

uint16 无符号整数（0 to 65535）

uint32 无符号整数（0 to 4294967295）

uint64 无符号整数（0 to 18446744073709551615）

float\_ float64 类型的简写

float16 半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位

float32 单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位

float64 双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位

complex\_ complex128 类型的简写，即 128 位复数

complex64 复数，表示双 32 位浮点数（实数部分和虚数部分）

complex128 复数，表示双 64 位浮点数（实数部分和虚数部分）

bool\_ 布尔型数据类型（True 或者 False）

intp 用于索引的整数类型（类似于 C 的 ssize\_t，一般情况下仍然是 int32 或 int64）

## 结构化数组

### 定义结构化数组

希望定义一个一维数组，每个元素有三个属性 name, age, wgt ，此时我们需要使用结构化数组。

a = np.array([1.0,2.0,3.0,4.0], np.float32)

my\_dtype = np.dtype([('mass', 'float32'), ('vol', 'float32')])

a.view(my\_dtype)

array([(1.0, 2.0), (3.0, 4.0)],

dtype=[('mass', '<f4'), ('vol', '<f4')])

这里，我们使用 dtype 创造了自定义的结构类型，然后用自定义的结构来解释数组 a 所占的内存。

对于自定义的类型，可以通过它的 names 属性查看它有哪些域：

('mass', 'vol')

这里 f4 表示四字节浮点数， < 表示小字节序。

利用这个自定义的结构类型，我们可以这样初始化结构化数组：

my\_data = np.array([(1,1), (1,2), (2,1), (1,3)], my\_dtype)

print my\_data

[(1.0, 1.0) (1.0, 2.0) (2.0, 1.0) (1.0, 3.0)]

第一个元素：

my\_data[0]

(1.0, 1.0)

得到第一个元素的速度信息，可以使用域的名称来索引：

my\_data[0]['vol']

1.0

得到所有的质量信息：

my\_data['mass']

array([ 1., 1., 2., 1.], dtype=float32)

自定义排序规则，先按速度，再按质量：

my\_data.sort(order=('vol', 'mass'))

print my\_data

[(1.0, 1.0) (2.0, 1.0) (1.0, 2.0) (1.0, 3.0)]

定义一个人的结构类型：

person\_dtype = np.dtype([('name', 'S10'), ('age', 'int'), ('weight', 'float')])

产生一个 3 x 4 共12人的空结构体数组：

people = np.empty((3,4), person\_dtype)

分别赋值：

people['name'] = [['Brad', 'Jane', 'John', 'Fred'],

['Henry', 'George', 'Brain', 'Amy'],

['Ron', 'Susan', 'Jennife', 'Jill']]

people['age'] = [[33, 25, 47, 54],

[29, 61, 32, 27],

[19, 33, 18, 54]]

people['weight'] = [[135., 105., 255., 140.],

[154., 202., 137., 187.],

[188., 135., 88., 145.]]

print(people)

[[('Brad', 33, 135.0) ('Jane', 25, 105.0) ('John', 47, 255.0)

('Fred', 54, 140.0)]

[('Henry', 29, 154.0) ('George', 61, 202.0) ('Brain', 32, 137.0)

('Amy', 27, 187.0)]

[('Ron', 19, 188.0) ('Susan', 33, 135.0) ('Jennife', 18, 88.0)

('Jill', 54, 145.0)]]

### 从文本中读取结构化数组

person\_dtype = np.dtype([('name', 'S10'), ('age', 'int'), ('weight', 'float')])

people = np.loadtxt('people.txt',

skiprows=1,

dtype=person\_dtype)

people

### 带标题的csv文件

文件.csv：

item,material,number

100,oak,33

110,maple,14

120,oak,7

145,birch,3

定义转换函数处理材料属性，使之对应一个整数：

tree\_to\_int = dict(oak = 1,

maple=2,

birch=3)

def convert(s):

return tree\_to\_int.get(s, 0)

使用 genfromtxt 载入数据，可以自动从第一行读入属性名称：

data = np.genfromtxt('wood.csv',

delimiter=',', # 逗号分隔

dtype=np.int, # 数据类型

names=True, # 从第一行读入域名

converters={1:convert}

)

data：

array([(100, 1, 33), (110, 2, 14), (120, 1, 7), (145, 3, 3)],

dtype=[('item', '<i4'), ('material', '<i4'), ('number', '<i4')])

### 嵌套类型

有时候，结构数组中的域可能包含嵌套的结构，例如，在我们希望在二维平面上纪录一个质点的位置和质量：

particle\_dtype = np.dtype([('position', [('x', 'float'), 'y', 'float')]), ('mass', 'float')])（里面一个有三个元素，(x,y),mass）

有如下文件：

2.0 3.0 42.0

2.1 4.3 32.5

1.2 4.6 32.3

4.5 -6.4 23.3

data = np.loadtxt('data.txt', dtype=particle\_dtype)

data：

array([((2.0, 3.0), 42.0), ((2.1, 4.3), 32.5), ((1.2, 4.6), 32.3), ((4.5, -6.4), 23.3)],

dtype=[('position', [('x', '<f8'), ('y', '<f8')]), ('mass', '<f8')])

查看位置的 x 轴：

data['position']['x']

## 记录数组record array

与结构数组类似

质点类型：

partical\_dtype = np.dtype([('mass', 'float'),

('velocity', 'float')])

生成记录数组要使用 numpy.rec 里的 fromrecords 方法：

from numpy import rec

particals\_rec = rec.fromrecords([(1,1), (1,2), (2,1), (1,3)],

dtype = partical\_dtype)

particals\_rec：

rec.array([(1.0, 1.0), (1.0, 2.0), (2.0, 1.0), (1.0, 3.0)],

dtype=[('mass', '<f8'), ('velocity', '<f8')])

在记录数组中，域可以通过属性来获得：

particals\_rec.mass：

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

也可以通过域来查询：

particals\_rec['mass']：

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

不过，记录数组的运行效率要比结构化数组要慢一些。

particals = np.array([(1,1), (1,2), (2,1), (1,3)], （生成结构化数组）

dtype = partical\_dtype)

使用 view 方法看成 recarray ：

particals\_rec = particals.view(np.recarray)

particals\_rec.mass：

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

对于自定义的类型，可以通过它的 names 属性查看它有哪些域：

('mass', 'velocity')

## 数组位运算

<http://www.runoob.com/numpy/numpy-binary-operators.html>

np字符串操作

## 字符串函数

以下函数用于对 dtype 为 numpy.string\_ 或 numpy.unicode\_ 的数组执行向量化字符串操作。 它们基于 Python 内置库中的标准字符串函数。

这些函数在字符数组类（numpy.char）中定义。

add() 对两个数组的逐个字符串元素进行连接

multiply() 返回按元素多重连接后的字符串

center() 居中字符串

capitalize() 将字符串第一个字母转换为大写

title() 将字符串的每个单词的第一个字母转换为大写

lower() 数组元素转换为小写

upper() 数组元素转换为大写

split() 指定分隔符对字符串进行分割，并返回数组列表

splitlines() 返回元素中的行列表，以换行符分割

strip() 移除元素开头或者结尾处的特定字符

join() 通过指定分隔符来连接数组中的元素

replace() 使用新字符串替换字符串中的所有子字符串

decode() 数组元素依次调用str.decode

encode() 数组元素依次调用str.encode

### numpy.char.add()

numpy.char.add() 函数依次对两个数组的元素进行字符串连接。

实例

import numpy as np

print (np.char.add(['hello'],[' xyz']))

print (np.char.add(['hello', 'hi'],[' abc', ' xyz']))

输出结果为：

连接两个字符串：

['hello xyz']

连接示例：

['hello abc' 'hi xyz']

### numpy.char.multiply()

numpy.char.multiply() 函数执行多重连接。

实例

import numpy as np

print (np.char.multiply('Runoob ',3))

输出结果为：

Runoob Runoob Runoob

### numpy.char.center()

numpy.char.center() 函数用于将字符串居中，并使用指定字符在左侧和右侧进行填充。

实例

import numpy as np

print (np.char.center('Runoob', 20,fillchar = '\*'))

输出结果为：加上ruboob一共是20个字符，如果字符串本身大于所规定的字符长度，则返回字符串

\*\*\*\*\*\*\*Runoob\*\*\*\*\*\*\*

### numpy.char.capitalize()

numpy.char.capitalize() 函数将字符串的第一个字母转换为大写：

实例

import numpy as np print (np.char.capitalize('runoob'))

输出结果为：

Runoob

### numpy.char.title()

numpy.char.title() 函数将字符串的每个单词的第一个字母转换为大写：

实例

import numpy as np print (np.char.title('i like runoob'))

输出结果为：

I Like Runoob

### numpy.char.lower()

numpy.char.lower() 函数对数组的每个元素转换为小写。它对每个元素调用 str.lower。

实例

import numpy as np #操作数组 print (np.char.lower(['RUNOOB','GOOGLE'])) # 操作字符串 print (np.char.lower('RUNOOB'))

输出结果为：

['runoob' 'google']

runoob

### numpy.char.upper()

numpy.char.upper() 函数对数组的每个元素转换为大写。它对每个元素调用 str.upper。

实例

import numpy as np #操作数组 print (np.char.upper(['runoob','google'])) # 操作字符串 print (np.char.upper('runoob'))

输出结果为：

['RUNOOB' 'GOOGLE']

RUNOOB

### numpy.char.split()

numpy.char.split() 通过指定分隔符对字符串进行分割，并返回数组。默认情况下，分隔符为空格。

实例

import numpy as np # 分隔符默认为空格 print (np.char.split ('i like runoob?')) # 分隔符为 . print (np.char.split ('www.runoob.com', sep = '.'))

输出结果为：

['i', 'like', 'runoob?']

['www', 'runoob', 'com']

### numpy.char.splitlines()

numpy.char.splitlines() 函数以换行符作为分隔符来分割字符串，并返回数组。

实例

import numpy as np # 换行符 \n print (np.char.splitlines('i\nlike runoob?')) print (np.char.splitlines('i\rlike runoob?'))

输出结果为：

['i', 'like runoob?']

['i', 'like runoob?']

**\n**，**\r**，**\r\n** 都可用作换行符。

### numpy.char.strip()

numpy.char.strip() 函数用于移除开头或结尾处的特定字符。

实例

import numpy as np # 移除字符串头尾的 a 字符 print (np.char.strip('ashok arunooba','a')) # 移除数组元素头尾的 a 字符 print (np.char.strip(['arunooba','admin','java'],'a'))

输出结果为：

shok arunoob

['runoob' 'dmin' 'jav']

### numpy.char.join()

numpy.char.join() 函数通过指定分隔符来连接数组中的元素或字符串

实例

import numpy as np # 操作字符串 print (np.char.join(':','runoob')) # 指定多个分隔符操作数组元素 print (np.char.join([':','-'],['runoob','google']))

输出结果为：

r:u:n:o:o:b

['r:u:n:o:o:b' 'g-o-o-g-l-e']

### numpy.char.replace()

numpy.char.replace() 函数使用新字符串替换字符串中的所有子字符串。

实例

import numpy as np print (np.char.replace ('i like runoob', 'oo', 'cc'))

输出结果为：

i like runccb

### numpy.char.encode()

numpy.char.encode() 函数对数组中的每个元素调用 str.encode 函数。 默认编码是 utf-8，可以使用标准 Python 库中的编解码器。

实例

import numpy as np a = np.char.encode('runoob', 'cp500') print (a)

输出结果为：

b'\x99\xa4\x95\x96\x96\x82'

### numpy.char.decode()

numpy.char.decode() 函数对编码的元素进行 str.decode() 解码。

实例

import numpy as np a = np.char.encode('runoob', 'cp500') print (a) print (np.char.decode(a,'cp500'))

输出结果为：

b'\x99\xa4\x95\x96\x96\x82'

runoob

## 对角线

### 1.diagonal

array([[11, 21, 31],

[12, 22, 32],

[13, 23, 33]])

a.diagonal()

array([11, 22, 33])

可以使用偏移来查看它的次对角线，正数表示右移，负数表示左移：

a.diagonal(offset=1)

array([21, 32])

a.diagonal(offset=-1)

array([12, 23])

### 2.利用花式索引

i = [0,1,2]  
a[i, i]

输出：array([11, 22, 33])

更新对角线的值：

a[i, i] = 2  
输出：

array([[ 2, 21, 31],

[12, 2, 32],

[13, 23, 2]])

或者：修改次对角线值

i = np.array([0,1])  
a[i, i + 1] = 1  
输出：array([[ 2, 1, 31],

[12, 2, 1],

[13, 23, 2]])

## 数组与字符串转换

### 1.tostring

a = np.array([[1,2],  
 [3,4]],   
 dtype = np.uint8)

转换为字符串:

a.tostring()

输出：

'\x01\x02\x03\x04'

a[0].tostring()

我们可以使用不同的顺序来转换字符串：

a.tostring(order=**'F'**)

这里使用了\*\*Fortran\*\*的格式，按照列来读数据。

输出：'\x01\x03\x02\x04'

### 2. fromstring 函数

s = a.tostring()  
b = np.fromstring(s,   
 dtype=np.uint8)  
b

输出：

array([1, 2, 3, 4], dtype=uint8)

此时，返回的数组是一维的，需要重新设定维度：

b.shape=2,2

### 对于文本文件，推荐使用

- loadtxt

- genfromtxt

- savetxt

### 对于二进制文本文件，推荐使用

- save

- load

- savez

## 数组相关操作

### 1.修改数组形状

#### numpy.reshape

numpy.reshape 函数可以在不改变数据的条件下修改形状，格式如下： numpy.reshape(arr, newshape, order='C')

* arr：要修改形状的数组
* newshape：整数或者整数数组，新的形状应当兼容原有形状
* order：'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'k' -- 元素在内存中的出现顺序。

实例

a = np.arange(8)

print (a)

b = a.reshape(4,2)

print (b)

输出结果如下：

原始数组：

[0 1 2 3 4 5 6 7]

修改后的数组：

[[0 1]

[2 3]

[4 5]

[6 7]]

#### 数组.flat/numpy.ndarray.flat

numpy.ndarray.flat 是一个数组元素迭代器，实例如下:

实例

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

for row in a:

print (row)

#对数组中每个元素都进行处理，可以使用flat属性，该属性是一个数组元素迭代器：

print ('迭代后的数组：')

for element in a.flat: 注意，不带括号

print (element)

输出结果如下：

原始数组：

[0 1 2]

[3 4 5]

[6 7 8]

迭代后的数组：

0

1

2

3

4

5

6

7

8

#### numpy.ndarray.flatten/数组.flatten()

numpy.ndarray.flatten 返回一份数组拷贝，对拷贝所做的修改不会影响原始数组，格式如下：

ndarray.flatten(order='C')

参数说明：

* order：'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'K' -- 元素在内存中的出现顺序。

实例

a = np.arange(8).reshape(2,4)

# 默认按行

print (a.flatten())

print ('以 F 风格顺序展开的数组：')

print (a.flatten(order = 'F'))

输出结果如下：

原数组：

[[0 1 2 3]

[4 5 6 7]]

展开的数组：

[0 1 2 3 4 5 6 7]

以 F 风格顺序展开的数组：

[0 4 1 5 2 6 3 7]

#### numpy.ravel（高效）

numpy.ravel() 展平的数组元素，顺序通常是"C风格"，返回的是数组视图（view，有点类似 C/C++引用reference的意味），修改会影响原始数组，引用关系。

该函数接收两个参数：

numpy.ravel(a, order='C')

参数说明：

* order：'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'K' -- 元素在内存中的出现顺序。

实例

a = np.arange(8).reshape(2,4)

print (a.ravel())

print ('以 F 风格顺序调用 ravel 函数之后：')

print (a.ravel(order = 'F'))

输出结果如下：

原数组：

[[0 1 2 3]

[4 5 6 7]]

调用 ravel 函数之后：

[0 1 2 3 4 5 6 7]

以 F 风格顺序调用 ravel 函数之后：

[0 4 1 5 2 6 3 7]

特殊情况：

a = array([[0,1],[ 2,3]])

aa = a.transpose()

b = aa.ravel()

在这种情况下，修改b不会修改到a。原因是我们用来 ravel 的对象 aa 本身是 a 的一个view。

#### apply\_over\_axes（func,a,axes）

#### apply\_along\_axis 对数组应用func

numpy.apply\_along\_axis(func, axis, arr, \*args, \*\*kwargs)：

必选参数：func,axis,arr。其中func是我们自定义的一个函数，函数func(arr)中的arr是一个数组，函数的主要功能就是对数组里的每一个元素进行变换，得到目标的结果。

其中axis表示函数func对数组arr作用的轴。

可选参数：\*args, \*\*kwargs。都是func()函数额外的参数。

返回值：numpy.apply\_along\_axis()函数返回的是一个根据func()函数以及维度axis运算后得到的的数组.

#### np.newaxis

添加新的维度

a为一维数组，

a[:,np.newaxis]：将a变为二维列向量

a[np.newaxis，：]:将a变为二维行向量

newaxis放到哪个位置，哪里增加一维，

如一维向量shape为（5，）

a[np.newaxis,:] 的shape为1\*5

a[:,np.newaxis]的shape为5\*1

如二维向量shape为（2,5）

则a[np.newaxis,:,:]的shape为1\*2\*5

a[:,np.newaxis,:]的shape为2\*1\*5

a[:,:,np.newaxis]的为2\*5\*1

### 2. 翻转数组

#### numpy.transpose/a.T

注意一维数组翻转后还是其本身。

numpy.transpose 函数用于对换数组的维度，格式如下：

numpy.transpose(arr, axes)

参数说明:

* arr：要操作的数组
* axes：整数列表，对应维度，通常所有维度都会对换。

实例

import numpy as np a = np.arange(12).reshape(3,4) print ('原数组：') print (a ) print ('\n') print ('对换数组：') print (np.transpose(a))

输出结果如下：

原数组：

[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

对换数组：

[[ 0 4 8]

[ 1 5 9]

[ 2 6 10]

[ 3 7 11]]

numpy.ndarray.T 类似 numpy.transpose：

实例

import numpy as np a = np.arange(12).reshape(3,4) print ('原数组：') print (a) print ('\n') print ('转置数组：') print (a.T)

输出结果如下：

原数组：

[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

转置数组：

[[ 0 4 8]

[ 1 5 9]

[ 2 6 10]

[ 3 7 11]]

#### numpy.rollaxis

numpy.rollaxis 函数向后滚动特定的轴到一个特定位置，格式如下：

numpy.rollaxis(arr, axis, start)

参数说明：

* arr：数组
* axis：要向后滚动的轴，其它轴的相对位置不会改变
* start：默认为零，表示完整的滚动。会滚动到特定位置。

实例

a = np.arange(8).reshape(2,2,2)

# 将轴 2 滚动到轴 0（宽度到深度） 2轴🡪0轴，0轴🡪1轴，1轴🡪2轴

print (np.rollaxis(a,2))

# 将轴 0 滚动到轴 1：（宽度到高度）

print (np.rollaxis(a,2,1))

输出结果如下：

原数组：

[[[0 1]

[2 3]]

[[4 5]

[6 7]]]

调用 rollaxis 函数：

[[[0 2]

[4 6]]

[[1 3]

[5 7]]]

调用 rollaxis 函数：

[[[0 2]

[1 3]]

[[4 6]

[5 7]]]

#### numpy.swapaxes

numpy.swapaxes 函数用于交换数组的两个轴，格式如下：

numpy.swapaxes(arr, axis1, axis2)

* arr：输入的数组
* axis1：对应第一个轴的整数
* axis2：对应第二个轴的整数

实例

a = np.arange(8).reshape(2,2,2)

# 现在交换轴 0（深度方向）到轴 2（宽度方向）

print (np.swapaxes(a, 2, 0))

输出结果如下：

原数组：

[[[0 1]

[2 3]]

[[4 5]

[6 7]]]

调用 swapaxes 函数后的数组：

[[[0 4]

[2 6]]

[[1 5]

[3 7]]]

### 3.修改维度

#### numpy.broadcast

numpy.broadcast 用于模仿广播的对象，它返回一个对象，该对象封装了将一个数组广播到另一个数组的结果。

该函数使用两个数组作为输入参数，如下实例：

实例

x = np.array([[1], [2], [3]])

y = np.array([4, 5, 6])

# 对 y 广播 x

b = np.broadcast(x,y)

# 它拥有 iterator 属性，基于自身组件的迭代器元组

r,c = b.iters

# Python3.x 为 next(context) ，Python2.x 为 context.next()

print (next(r), next(c))

print (next(r), next(c))

# shape 属性返回广播对象的形状

print (b.shape)

# 手动使用 broadcast 将 x 与 y 相加

b = np.broadcast(x,y)

c = np.empty(b.shape)

print (c.shape)

c.flat = [u + v for (u,v) in b]

print (c)

# 获得了和 NumPy 内建的广播支持相同的结果

print (x + y)

输出结果为：

对 y 广播 x：

1 4

1 5

广播对象的形状：

(3, 3)

手动使用 broadcast 将 x 与 y 相加：

(3, 3)

调用 flat 函数：

[[5. 6. 7.]

[6. 7. 8.]

[7. 8. 9.]]

x 与 y 的和：

[[5 6 7]

[6 7 8]

[7 8 9]]

#### numpy.broadcast\_to

numpy.broadcast\_to 函数将数组广播到新形状。它在原始数组上返回只读视图。 它通常不连续。 如果新形状不符合 NumPy 的广播规则，该函数可能会抛出ValueError。

numpy.broadcast\_to(array, shape, subok)

实例

import numpy as np a = np.arange(4).reshape(1,4) print ('原数组：') print (a) print ('\n') print ('调用 broadcast\_to 函数之后：') print (np.broadcast\_to(a,(4,4)))

输出结果为：

原数组：

[[0 1 2 3]]

调用 broadcast\_to 函数之后：

[[0 1 2 3]

[0 1 2 3]

[0 1 2 3]

[0 1 2 3]]

#### numpy.expand\_dims

numpy.expand\_dims 函数通过在指定位置插入新的轴来扩展数组形状，函数格式如下:

numpy.expand\_dims(arr, axis)

参数说明：

* arr：输入数组
* axis：新轴插入的位置

实例

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

y = np.expand\_dims(x, axis = 0)

print (x.shape, y.shape)

# 在位置 1 插入轴

y = np.expand\_dims(x, axis = 1)

print ('在位置 1 插入轴之后的数组 y：')

print (y)

print (x.ndim,y.ndim)

print (x.shape, y.shape)

输出结果为：

数组 x：

[[1 2]

[3 4]]

数组 y：

[[[1 2]

[3 4]]]

数组 x 和 y 的形状：

(2, 2) (1, 2, 2)

在位置 1 插入轴之后的数组 y：

[[[1 2]]

[[3 4]]]

x.ndim 和 y.ndim：

2 3

x.shape 和 y.shape：

(2, 2) (2, 1, 2)

#### numpy.squeeze

numpy.squeeze 函数从给定数组的形状中删除一维的条目，函数格式如下：

numpy.squeeze(arr, axis)

参数说明：

* arr：输入数组
* axis：整数或整数元组，用于选择形状中一维条目的子集

实例

import numpy as np x = np.arange(9).reshape(1,3,3) print ('数组 x：') print (x) print ('\n') y = np.squeeze(x) print ('数组 y：') print (y) print ('\n') print ('数组 x 和 y 的形状：') print (x.shape, y.shape)

输出结果为：

数组 x：

[[[0 1 2]

[3 4 5]

[6 7 8]]]

数组 y：

[[0 1 2]

[3 4 5]

[6 7 8]]

数组 x 和 y 的形状：

(1, 3, 3) (3, 3)

#### atleast\_1d/ atleast\_2d/ atleast\_3d ：

数组最少为1,2,3维。满足条件则不变动，若不满足，则添加新的维数。

### 4.拼接连接数组

#### numpy.concatenate

numpy.concatenate 函数用于沿指定轴连接相同形状的两个或多个数组，格式如下：

指定轴后，其他非指定轴需要shape一样。

numpy.concatenate((a1, a2, ...), axis)

参数说明：

* a1, a2, ...：相同类型的数组
* axis：沿着它连接数组的轴，默认为 0

实例

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

b = np.array([[5,6],[7,8]])

# 两个数组的维度相同

print ('沿轴 0 连接两个数组：')

print (np.concatenate((a,b)))

print ('沿轴 1 连接两个数组：')

print (np.concatenate((a,b),axis = 1))

输出结果为：

第一个数组：

[[1 2]

[3 4]]

第二个数组：

[[5 6]

[7 8]]

沿轴 0 连接两个数组：

[[1 2]

[3 4]

[5 6]

[7 8]]

沿轴 1 连接两个数组：

[[1 2 5 6]

[3 4 7 8]]

#### numpy.stack

numpy.stack 函数用于沿新轴连接数组序列，格式如下：

numpy.stack(arrays, axis)

参数说明：

* arrays相同形状的数组序列
* axis：返回数组中的轴，输入数组沿着它来堆叠

实例

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

b = np.array([[5,6],[7,8]])

print ('沿轴 0 堆叠两个数组：')

print (np.stack((a,b),0))

print ('沿轴 1 堆叠两个数组：')

print (np.stack((a,b),1))

输出结果如下：

第一个数组：

[[1 2]

[3 4]]

第二个数组：

[[5 6]

[7 8]]

沿轴 0 堆叠两个数组：

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

沿轴 1 堆叠两个数组：

[[[1 2]

[5 6]]

[[3 4]

[7 8]]]

#### numpy.hstack

numpy.hstack 是 numpy.stack 函数的变体，它通过水平堆叠来生成数组。

实例

import numpy as np a = np.array([[1,2],[3,4]]) print ('第一个数组：') print (a) print ('\n') b = np.array([[5,6],[7,8]]) print ('第二个数组：') print (b) print ('\n') print ('水平堆叠：') c = np.hstack((a,b)) print (c) print ('\n')

输出结果如下：

第一个数组：

[[1 2]

[3 4]]

第二个数组：

[[5 6]

[7 8]]

水平堆叠：

[[1 2 5 6]

[3 4 7 8]]

#### numpy.vstack

numpy.vstack 是 numpy.stack 函数的变体，它通过垂直堆叠来生成数组。

实例

import numpy as np a = np.array([[1,2],[3,4]]) print ('第一个数组：') print (a) print ('\n') b = np.array([[5,6],[7,8]]) print ('第二个数组：') print (b) print ('\n') print ('竖直堆叠：') c = np.vstack((a,b)) print (c)

输出结果为：

第一个数组：

[[1 2]

[3 4]]

第二个数组：

[[5 6]

[7 8]]

竖直堆叠：

[[1 2]

[3 4]

[5 6]

[7 8]]

#### column\_stack

### 5.分割数组

#### ①numpy.split

numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：

hsplit 将一个数组水平分割为多个子数组（按列）

vsplit 将一个数组垂直分割为多个子数组（按行）

numpy.split(ary, indices\_or\_sections, axis)

参数说明：

ary：被分割的数组

indices\_or\_sections：如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置（左开右闭）

axis：沿着哪个维度进行切向，默认为0，横向切分。为1时，纵向切分

实例

import numpy as np

a = np.arange(9)

print (a)

print ('将数组分为三个大小相等的子数组：')

b = np.split(a,3)

print (b)

print ('将数组在一维数组中表明的位置分割：')

b = np.split(a,[4,7])

print (b)

输出结果为：

第一个数组：

[0 1 2 3 4 5 6 7 8]

将数组分为三个大小相等的子数组：

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

将数组在一维数组中表明的位置分割：

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

#### ②numpy.hsplit

numpy.hsplit 函数用于水平分割数组，通过指定要返回的相同形状的数组数量来拆分原数组。

实例

import numpy as np

harr = np.floor(10 \* np.random.random((2, 6)))

print(harr)

print(np.hsplit(harr, 3))

输出结果为：

原array：

[[4. 7. 6. 3. 2. 6.]

[6. 3. 6. 7. 9. 7.]]

拆分后：

[array([[4., 7.],

[6., 3.]]),

array([[6., 3.],

[6., 7.]]),

array([[2., 6.],

[9., 7.]])]

#### ④numpy.vsplit

numpy.vsplit 沿着垂直轴分割，其分割方式与hsplit用法相同。

实例

import numpy as np

a = np.arange(16).reshape(4,4)

print (a)

b = np.vsplit(a,2)

print (b)

输出结果为：

第一个数组：

[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]

[12 13 14 15]]

竖直分割：

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

[4, 5, 6, 7]]),

array([[ 8, 9, 10, 11],

[12, 13, 14, 15]])]

### 6.数组元素的添加与删除

insert 沿指定轴将值插入到指定下标之前

delete 删掉某个轴的子数组，并返回删除后的新数组

resize 返回指定形状的新数组

append 将值添加到数组末尾

unique 查找数组内的唯一元素

#### ①numpy.resize

numpy.resize 函数返回指定大小的新数组。

如果新数组大小大于原始大小，则包含原始数组中的元素的副本。

其实就是将原数组一维排开，然后按照指定的shape 重新排列。不够的，按照一维数组顺序进行重复补充。

numpy.resize(arr, shape)

参数说明：

arr：要修改大小的数组

shape：返回数组的新形状

实例

import numpy as np

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

print (a)

print (a.shape)

b = np.resize(a, (3,2))

print (b)

print (b.shape)

#要注意 a 的第一行在 b 中重复出现，因为尺寸变大了

b = np.resize(a,(3,3))

print (b)

输出结果为：

第一个数组：

[[1 2 3]

[4 5 6]]

第一个数组的形状：

(2, 3)

第二个数组：

[[1 2]

[3 4]

[5 6]]

第二个数组的形状：

(3, 2)

修改第二个数组的大小：

[[1 2 3]

[4 5 6]

[1 2 3]]

#### ②numpy.append

numpy.append 函数在数组的末尾添加值。 追加操作会分配整个数组，并把原来的数组复制到新数组中。 此外，输入数组的维度必须匹配否则将生成ValueError。

append 函数不指定axis情况下，返回的始终是一个一维数组。

numpy.append(arr, values, axis=None)

参数说明：

arr：输入数组

values：要向arr添加的值，需要和arr形状相同（除了要添加的轴）

axis：默认为 None。当axis无定义时，是横向加成，返回总是为一维数组！当axis有定义的时候，分别为0和1的时候。当axis为0的时候，数组加下面（列数要相同）。当axis为1时，数组是加在右边（行数要相同）。

实例

import numpy as np

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

print (a) print ('\n')

print (np.append(a, [7,8,9]))

print ('沿轴 0 添加元素：')

print (np.append(a, [[7,8,9]],axis = 0))

print ('沿轴 1 添加元素：')

print (np.append(a, [[5,5,5],[7,8,9]],axis = 1))

输出结果为：

第一个数组：

[[1 2 3]

[4 5 6]]

向数组添加元素：

[1 2 3 4 5 6 7 8 9]

沿轴 0 添加元素：

[[1 2 3]

[4 5 6]

[7 8 9]]

沿轴 1 添加元素：

[[1 2 3 5 5 5]

[4 5 6 7 8 9]]

#### ③numpy.insert

numpy.insert 函数在给定索引之前，沿给定轴在输入数组中插入值。

如果值的类型转换为要插入，则它与输入数组不同。 插入没有原地的，函数会返回一个新数组。 此外，如果未提供轴，则输入数组会被展开。

numpy.insert(arr, obj, values, axis)

参数说明：

arr：输入数组

obj：在其之前插入值的索引

values：要插入的值

axis：沿着它插入的轴，如果未提供，则输入数组会被展开

如果按照某维插入，而value为给够，则会重复使用value。

实例

import numpy as np

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

print (a)

print ('未传递 Axis 参数。 在插入之前输入数组会被展开。')

print (np.insert(a,3,[11,12]))

print ('传递了 Axis 参数。 会广播值数组来配输入数组。')

print ('沿轴 0 广播：')

print (np.insert(a,1,[11],axis = 0))

print ('沿轴 1 广播：')

print (np.insert(a,1,11,axis = 1))

输出结果如下：

第一个数组：

[[1 2]

[3 4]

[5 6]]

未传递 Axis 参数。 在插入之前输入数组会被展开。

[ 1 2 3 11 12 4 5 6]

传递了 Axis 参数。 会广播值数组来配输入数组。

沿轴 0 广播：

[[ 1 2]

[11 11]

[ 3 4]

[ 5 6]]

沿轴 1 广播：

[[ 1 11 2]

[ 3 11 4]

[ 5 11 6]]

#### ④numpy.delete

numpy.delete 函数返回从输入数组中删除指定子数组的新数组。 与 insert() 函数的情况一样，如果未提供轴参数，则输入数组将展开。

Numpy.delete(arr, obj, axis)

参数说明：

arr：输入数组

obj：可以被切片，整数或者整数数组，表明要从输入数组删除的子数组

axis：沿着它删除给定子数组的轴，如果未提供，则输入数组会被展开

实例

import numpy as np

a = np.arange(12).reshape(3,4)

print (a)

print ('未传递 Axis 参数。 在插入之前输入数组会被展开。')

print (np.delete(a,5))

print ('删除第二列：')

print (np.delete(a,1,axis = 1))

print ('包含从数组中删除的替代值的切片：')

a = np.array([1,2,3,4,5,6,7,8,9,10])

print (np.delete(a, np.s\_[::2]))

输出结果为：

第一个数组：

[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

未传递 Axis 参数。 在插入之前输入数组会被展开。

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

删除第二列：

[[ 0 2 3]

[ 4 6 7]

[ 8 10 11]]

包含从数组中删除的替代值的切片：

[ 2 4 6 8 10]

#### ⑤numpy.unique

numpy.unique 函数用于去除数组中的重复元素。

numpy.unique(arr, return\_index, return\_inverse, return\_counts)

arr：输入数组，如果不是一维数组则会展开

return\_index：如果为true，返回新列表元素在旧列表中的位置（下标），并以列表形式储

return\_inverse：如果为true，返回旧列表元素在新列表中的位置（下标），并以列表形式储

return\_counts：如果为true，返回去重数组中的元素在原数组中的出现次数

实例

import numpy as np

a = np.array([5,2,6,2,7,5,6,8,2,9])

print (a)

u = np.unique(a)

print (u)

u,indices = np.unique(a, return\_index = True)

print (indices)

print ('我们可以看到每个和原数组下标对应的数值：')

print (a)

print ('去重数组的下标：')

u,indices = np.unique(a,return\_inverse = True)

print (u)

print (indices) print ('\n')

print ('使用下标重构原数组：')

print (u[indices])

print ('返回去重元素的重复数量：')

u,indices = np.unique(a,return\_counts = True)

print (u)

print (indices)

输出结果为：

第一个数组：

[5 2 6 2 7 5 6 8 2 9]

第一个数组的去重值：

[2 5 6 7 8 9]

去重数组的索引数组：

[1 0 2 4 7 9]

我们可以看到每个和原数组下标对应的数值：

[5 2 6 2 7 5 6 8 2 9]

去重数组的下标：

[2 5 6 7 8 9]

下标为：

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

使用下标重构原数组：

[5 2 6 2 7 5 6 8 2 9]

返回去重元素的重复数量：

[2 5 6 7 8 9]

[3 2 2 1 1 1]

### 7. tile：重复某个数组

(类似于Matlab中的remat)

>>> data = np.array([[1,2],[3,4]])

>>> np.tile(data,2)

array([[1, 2, 1, 2],

[3, 4, 3, 4]])

>>> np.tile(data,[2,3])

array([[1, 2, 1, 2, 1, 2],

[3, 4, 3, 4, 3, 4],

[1, 2, 1, 2, 1, 2],

[3, 4, 3, 4, 3, 4]])

### 8.获取多维数组第n个元素的索引

np.unravel\_index(100,(6,7,8))

获取一个6\*7\*8数组的第100个元素的索引。

### 9.边缘填充pad

np. pad（array，pad\_width，mode，\*\*kwars）

mode取值：

constant/edge/linear\_ramp’/maxmium/minmium/mean/median/reflect/‘symmetric/wrap

<https://blog.csdn.net/qq_29592167/article/details/81043640>

还有一个参数 constant\_values=

当mode为constant时，指定填充的值。可以为一个数，则所有填充都一样。如果是个元组（2,3），则为前面填充2，后面填充3。对于多维array，则可以使用嵌套元组。

对于多维情况下，按照每个维度前增多少，后增多少来

如：对于一个2\*3的数组

pad\_width=((2,3),(4,5))

代表第一维度前增加2，后增加3，由以前的2变为2+2+3=7

第二维度前增加4，后增加5，由以前的3变为4+3+5=12

所以数组变为7\*12。

如果pad\_width只是一个整数，则说明所有维度前后增加的都一样。

### 索引转换

np.unravel\_index(num,(dim))

将一维索引转换为指定维数的数组的索引。即查询该数组第n个元素的索引。

比如查询6\*7\*8数组的第一百个元素的索引是多少

np.unravel\_index(100,(6,78))

返回一个元组。

## 数组属性方法总结：

||作用|

|-|-|

### |1|\*\*基本属性\*\*

| a.dtype |数组元素类型 float32,uint8,...

| a.shape |数组形状 (m,n,o,...)

| a.size |数组元素数

| a.itemsize |每个元素占字节数

| a.nbytes |所有元素占的字节

| a.ndim |数组维度

### |2|\*\*形状相关\*\*

| a.flat |所有元素的迭代器

| a.flatten() |返回一个1维数组的复制

| a.ravel() |返回一个1维数组，高效

| a.resize(new\_size) |改变形状

| a.swapaxes(axis1, axis2) |交换两个维度的位置

| a.transpose(\*axex) |交换所有维度的位置

| a.T |转置， a.transpose()

| a.squeeze() | 去除所有长度为1的维度

### |3|\*\*填充复制\*\*fill copy

| a.copy() | 返回数组的一个复制

| a.fill(value) | 将数组的元组设置为特定值

### |4|\*\*转化\*\*

| a.tolist() |将数组转化为列表

| a.tostring() |转换为字符串

| a.astype(dtype) |转化为指定类型

| a.byteswap(False) |转换大小字节序

| a.view(type\_or\_dtype) |生成一个使用相同内存，但使用不同的表示方法的数组

### |5|\*\*复数\*\*

| a.imag |虚部

| a.real |实部

| a.conjugate() |复共轭

| a.conj() |复共轭（缩写）

### |6|\*\*保存\*\*

| a.dump(file) |将二进制数据存在file中

| a.dump() |将二进制数据表示成字符串

| a.tofile(fid, sep="",format="%s") |格式化ASCⅡ码写入文件

### |7|\*\*查找排序\*\*

| a.nonzero() |返回所有非零元素的索引

| a.sort(axis=-1) |沿某个轴排序

| a.argsort(axis=-1) |沿某个轴，返回按排序的索引

| a.searchsorted(b) |返回将b中元素插入a后能保持有序的索引值，在sorted参数省略的情况下，a必须是有序数组。如果a为无序，可以添加参数sorted=np.argsort(a),排序。

### |8|\*\*元素数学操作\*\*

| a.clip(low, high) |将数值限制在一定范围内

| a.round(decimals=0) |近似到指定精度

| a.cumsum(axis=None) |累加和

| a.cumprod(axis=None) |累乘积

### |9|\*\*约简操作\*\*

| a.sum(axis=None) |求和

| a.prod(axis=None) |求积

| a.min(axis=None) |最小值

| a.max(axis=None) |最大值

| a.argmin(axis=None) |最小值索引

| a.argmax(axis=None) |最大值索引

| a.ptp(axis=None) |最大值减最小值

| a.mean(axis=None) |平均值

| a.std(axis=None) |标准差

| a.var(axis=None) |方差

| a.any(axis=None) |只要有一个不为0，返回真，逻辑或

| a.all(axis=None) |所有都不为0，返回真，逻辑与

## 利用函数生成/连接数组

### 1.arange

arange 类似于\*\*Python\*\*中的 range 函数，只不过返回的不是列表，而是数组：

np.arange(start=0, stop=None, step=1, dtype=None)

产生一个在区间 [start, stop) 之间，以 step 为间隔的数组，包含start，但是不包含stop。但是对于单精度float，由于精度问题，有时可能包含stop，使用时要注意。

如果只输入一个参数， 则默认从 0 开始，并以这个值为结束：

np.arange(4)

out: array([0, 1, 2, 3])

与 range 不同， arange 允许非整数值输入，产生一个非整型的数组：

np.arange(0, 2 \* np.pi, np.pi / 4)

输出：array([0. , 0.78539816, 1.57079633, 2.35619449, 3.14159265,3.92699082, 4.71238898, 5.49778714])

可以不指定dtype来指定数组类型，自行判定。

也可以通过dtype来指定数组类型。

### 2. linspace(start, stop, N)

产生 N 个等距分布在 [start, stop] 间的元素组成的数组，包括 start, stop 。

np.linspace(0, 1, 5)

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

### 3. logspace

logspace(start, stop, N)

包含初始值和结束值

先从start到stop中平均找出N个数，然后取10幂。

产生 N 个对数等距分布的数组，默认以10为底：

np.logspace(0, 1, 5)

输出：array([ 1. , 1.77827941, 3.16227766, 5.62341325, 10. ])

产生的值为[100,100.25,100.5,100.75,101]

### 4. meshgrid 行向量、列向量

有时候需要在二维平面中生成一个网格，这时候可以使用 meshgrid 来完成这样的工作：

x\_ticks = np.linspace(-1, 1, 5)

y\_ticks = np.linspace(-1, 1, 5)

x, y = np.meshgrid(x\_ticks, y\_ticks)

x

array([[-1. , -0.5, 0. , 0.5, 1. ],

[-1. , -0.5, 0. , 0.5, 1. ],

[-1. , -0.5, 0. , 0.5, 1. ],

[-1. , -0.5, 0. , 0.5, 1. ],

[-1. , -0.5, 0. , 0.5, 1. ]])

y

array([[-1. , -1. , -1. , -1. , -1. ],

[-0.5, -0.5, -0.5, -0.5, -0.5],

[ 0. , 0. , 0. , 0. , 0. ],

[ 0.5, 0.5, 0.5, 0.5, 0.5],

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

x 对应网格的第一维， y 对应网格的第二维。

x为[-1 , -0.5, 0, 0.5, 1]

y=[-1 , -0.5, 0, 0.5, 1]T

x每一个对应y 5个，同样y每一个也对应x 5个，所以x有25个，y有25个。

事实上， x, y 中有很多冗余的元素，这里提供了一个 sparse 的选项：

x\_ticks = np.linspace(-1, 1, 5)

y\_ticks = np.linspace(-1, 1, 5)

x, y = np.meshgrid(x\_ticks, y\_ticks, sparse=True)

x

输出：array([[-1. , -0.5, 0. , 0.5, 1. ]])

y：

输出：array([[-1. ],

[-0.5],

[ 0. ],

[ 0.5],

[ 1. ]])

注意，x、y都是二维数组。

在这个选项下，x, y 变成了单一的行向量和列向量。

但这并不影响结果。

meshgrid 可以设置轴排列的先后顺序：

- 默认为 indexing='xy' 即笛卡尔坐标，对于2维数组，返回行向量 x 和列向量 y

- 或者使用 indexing='ij' 即矩阵坐标，对于2维数组，返回列向量 x 和行向量 y 。

### 5. ogrid , mgrid 行向量、列向量

ogrid 与 mgrid 的区别在于：

- ogrid 相当于 meshgrid(indexing='ij', sparse=True)

- mgrid 相当于 meshgrid(indexing='ij', sparse=False)

x, y = np.ogrid[-1:1:.5, -1:1:.5]

x

array([[-1. ],

[-0.5],

[ 0. ],

[ 0.5]])

y

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

注意：

- 这里使用的是中括号

\*\*Numpy\*\* 使用的是 start:end:step 的表示

- 这里的结果不包括 end 的值

为了包含 end 的值，我们可以使用这样的技巧：在end后面加j

x, y = np.ogrid[-1:1:5j, -1:1:5j]

x, y

(array([[-1. ],

[-0.5],

[ 0. ],

[ 0.5],

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

### 6.r \_ , c \_ 行向量、列向量

我们可以使用 r\_ / c\_ 来产生行向量或者列向量。

#### ①使用切片产生：

类似arange，不含end值。

np.r\_[0:1:.1]

输出：array([ 0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])

#### ②复数步长指定数组长度：

复数指定数组的长度，类似linespace。包含start、end

np.r\_[0:1:5j]

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

#### ③连接多个序列，产生数组：

np.r\_[(3,22,11), 4.0, [15, 6]]

array([ 3., 22., 11., 4., 15., 6.])

#### 列向量使用方法是一样，只不过产生的是列向量。

### 7. ones , zeros

ones(shape, dtype=float64)

zeros(shape, dtype=float64)

两维以上的shape是list类型。

产生一个制定形状的全 0 或全 1 的数组，还可以制定数组类型：

np.zeros(3)

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

np.ones([2,3], dtype=np.float32)

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

[ 1., 1., 1.]], dtype=float32)

产生一个全是 5 的数组：

np.ones([2,3]) \* 5

array([[ 5., 5., 5.],

[ 5., 5., 5.]])

### 8.empty

empty(shape, dtype=float64, order='C')

也可以使用 empty 方法产生一个制定大小的数组（数组所指向的内存未被初始化，所以值随机），再用 fill 方法填充：

a = np.empty(2)

a

array([-0.03412165, 0.05516321])

a.fill(5)

a

array([ 5., 5.])

另一种替代方法使用索引，不过速度会稍微慢一些：

a[:] = 5

a

array([ 5., 5.])

### 9. empty \_ like, ones \_ like, zeros \_ like

empty\_like(a)

ones\_like(a)

zeros\_like(a)

产生一个跟 a 大小一样，类型一样的对应数组。

### 10. identity/eye 单位矩阵

indentity(n, dtype=float64)

产生一个 n 乘 n 的单位矩阵：

np.identity(3)

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

[ 0., 1., 0.],

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

### 11.concatenate

concatenate((a0,a1,...,aN), axis=0)

默认维度为0，也就是第一维

注意，这些数组要用 () 包括到一个元组中去。

除了给定的轴外，这些数组其他轴的长度必须是一样的。

### 12.diag(filter,k)

创建对角矩阵或者次对角矩阵

filter为可迭代的指定的对角线或者次对角线元素

k省略，表示是主对角线，k=-1是对角线下方次对角线，k=-2是主对角线下方次次对角线。k=1是主对角线上方次对角线。

diag会根据所给的对角元素的个数和指定的对角线，自动创建对角矩阵，其余元素都填0。

练习16：创建一个5\*5矩阵，其下次对角线值为1,2,3,4

a16 = np.diag(1+np.arange(4),k=-1)

array([[0, 0, 0, 0, 0],

[1, 0, 0, 0, 0],

[0, 2, 0, 0, 0],

[0, 0, 3, 0, 0],

[0, 0, 0, 4, 0]])

## NumPy 广播(Broadcast)

广播(Broadcast)是 numpy 对不同形状(shape)的数组进行数值计算的方式， 对数组的算术运算通常在相应的元素上进行。

如果两个数组 a 和 b 形状相同，即满足 **a.shape == b.shape**，那么 a\*b 的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同。

详细介绍：<https://www.cnblogs.com/yangmang/p/7125458.html>

实例

import numpy as np a = np.array([1,2,3,4]) b = np.array([10,20,30,40]) c = a \* b print (c)

输出结果为：

[ 10 40 90 160]

当运算中的 2 个数组的形状不同时，numpy 将自动触发广播机制。如：

实例

import numpy as np

a = np.array([[ 0, 0, 0], [10,10,10], [20,20,20], [30,30,30]])

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

print(a + b)

输出结果为：

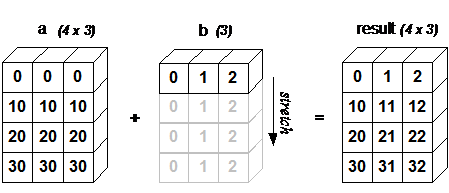
[[ 1 2 3]

[11 12 13]

[21 22 23]

[31 32 33]]

下面的图片展示了数组 b 如何通过广播来与数组 a 兼容。



4x3 的二维数组与长为 3 的一维数组相加，等效于把数组 b 在二维上重复 4 次再运算：

实例

import numpy as np

a = np.array([[ 0, 0, 0], [10,10,10], [20,20,20], [30,30,30]])

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

print(a + bb)

输出结果为：

[[ 1 2 3]

[11 12 13]

[21 22 23]

[31 32 33]]

### ****广播的规则:****

1.让所有输入数组都向其中形状最长的数组看齐，形状中不足的部分都通过在前面加 1 补齐。

2.输出数组的形状是输入数组形状的各个维度上的最大值。

3.如果输入数组的某个维度和输出数组的对应维度的长度相同或者其长度为 1 时，这个数组能够用来计算，否则出错。

4.当输入数组的某个维度的长度为 1 时，沿着此维度运算时都用此维度上的第一组值。

**简单理解：**对两个数组，分别比较他们的每一个维度（若其中一个数组没有当前维度则忽略），满足：

数组拥有相同形状。

当前维度的值相等。

当前维度的值有一个是 1。

若条件不满足，抛出 **"ValueError: frames are not aligned"** 异常。

## 视图和副本

副本是一个数据的完整的拷贝，如果我们对副本进行修改，它不会影响到原始数据，物理内存不在同一位置。

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，但原有数据不会产生拷贝。如果我们对视图进行修改，它会影响到原始数据，物理内存在同一位置。

**视图一般发生在：**

* 1、numpy 的切片操作返回原数据的视图。
* 2、调用 ndarray 的 view() 函数产生一个视图。

**副本一般发生在：**

* Python 序列的切片操作，调用deepCopy()函数。
* 调用 ndarray 的 copy() 函数产生一个副本。

### 无复制

简单的赋值不会创建数组对象的副本。 相反，它使用原始数组的相同id()来访问它。 id()返回 Python 对象的通用标识符，类似于 C 中的指针。

如对于b=a，b是a的引用，会随着a的变化一起变化。

此外，一个数组的任何变化都反映在另一个数组上。 例如，一个数组的形状改变也会改变另一个数组的形状。

实例

import numpy as np a = np.arange(6) print ('我们的数组是：') print (a) print ('调用 id() 函数：') print (id(a)) print ('a 赋值给 b：') b = a print (b) print ('b 拥有相同 id()：') print (id(b)) print ('修改 b 的形状：') b.shape = 3,2 print (b) print ('a 的形状也修改了：') print (a)

输出结果为：

我们的数组是：

[0 1 2 3 4 5]

调用 id() 函数：

4349302224

a 赋值给 b：

[0 1 2 3 4 5]

b 拥有相同 id()：

4349302224

修改 b 的形状：

[[0 1]

[2 3]

[4 5]]

a 的形状也修改了：

[[0 1]

[2 3]

[4 5]]

### 视图或浅拷贝

ndarray.view() 方会创建一个新的数组对象，该方法创建的新数组的维数更改不会更改原始数据的维数。但是使用切片创建视图修改数据会影响到原始数组：

实例

import numpy as np # 最开始 a 是个 3X2 的数组

a = np.arange(6).reshape(3,2)

print ('创建 a 的视图：')

b = a.view()

print (b)

print ('两个数组的 id() 不同：')

print ('a 的 id()：') print (id(a)) print ('b 的 id()：' ) print (id(b)) # 修改 b 的形状，并不会修改 a b.shape = 2,3 print ('b 的形状：') print (b) print ('a 的形状：') print (a)

输出结果为：

数组 a：

[[0 1]

[2 3]

[4 5]]

创建 a 的视图：

[[0 1]

[2 3]

[4 5]]

两个数组的 id() 不同：

a 的 id()：

4314786992

b 的 id()：

4315171296

b 的形状：

[[0 1 2]

[3 4 5]]

a 的形状：

[[0 1]

[2 3]

[4 5]]

使用切片创建视图修改数据会影响到原始数组：

实例

import numpy as np arr = np.arange(12) print ('我们的数组：') print (arr) print ('创建切片：') a=arr[3:] b=arr[3:] a[1]=123 b[2]=234 print(arr) print(id(a),id(b),id(arr[3:]))

输出结果为：

我们的数组：

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

创建切片：

[ 0 1 2 3 123 234 6 7 8 9 10 11]

4545878416 4545878496 4545878576

变量 a,b 都是 arr 的一部分视图，对视图的修改会直接反映到原数据中。但是我们观察 a,b 的 id，他们是不同的，也就是说，视图虽然指向原数据，但是他们和赋值引用还是有区别的。

### 副本或深拷贝

ndarray.copy() 函数创建一个副本。 对副本数据进行修改，不会影响到原始数据，它们物理内存不在同一位置。

实例

import numpy as np a = np.array([[10,10], [2,3], [4,5]]) print ('数组 a：') print (a) print ('创建 a 的深层副本：') b = a.copy() print ('数组 b：') print (b) # b 与 a 不共享任何内容 print ('我们能够写入 b 来写入 a 吗？') print (b is a) print ('修改 b 的内容：') b[0,0] = 100 print ('修改后的数组 b：') print (b) print ('a 保持不变：') print (a)

输出结果为：

数组 a：

[[10 10]

[ 2 3]

[ 4 5]]

创建 a 的深层副本：

数组 b：

[[10 10]

[ 2 3]

[ 4 5]]

我们能够写入 b 来写入 a 吗？

False

修改 b 的内容：

修改后的数组 b：

[[100 10]

[ 2 3]

[ 4 5]]

a 保持不变：

[[10 10]

[ 2 3]

[ 4 5]]

### 更多相关文章

[Python 直接赋值、浅拷贝和深度拷贝解析](http://www.runoob.com/w3cnote/python-append-deepcopy.html)

## NumPy 迭代数组

### nditer

NumPy 迭代器对象 numpy.nditer 提供了一种灵活访问一个或者多个数组元素的方式。

迭代器最基本的任务的可以完成对数组元素的访问。

接下来我们使用 arange() 函数创建一个 2X3 数组，并使用 nditer 对它进行迭代。

实例

注意在生成数组时，可以指定参数order=’F’or’C’,来指定是以行优先还是列优先。

默认为任意方向，’A’

import numpy as np

a = np.arange(6).reshape(2,3)

print ('原始数组是：')

print (a)

print ('\n')

print ('迭代输出元素：')

for x in np.nditer(a):

print (x, end=", " )

print ('\n')

输出结果为：

原始数组是：

[[0 1 2]

[3 4 5]]

迭代输出元素：

0, 1, 2, 3, 4, 5,

以上实例不是使用标准 C 或者 Fortran 顺序，选择的顺序是和数组内存布局一致的，这样做是为了提升访问的效率，默认是行序优先（row-major order，或者说是 C-order）。

这反映了默认情况下只需访问每个元素，而无需考虑其特定顺序。我们可以通过迭代上述数组的转置来看到这一点，并与以 C 顺序访问数组转置的 copy 方式做对比，如下实例：

实例

import numpy as np

a = np.arange(6).reshape(2,3)

for x in np.nditer(a.T):

print (x, end=", " )

print ('\n')

for x in np.nditer(a.T.copy(order='C')):

print (x, end=", " )

print ('\n')

输出结果为：

0, 1, 2, 3, 4, 5,

0, 3, 1, 4, 2, 5,

从上述例子可以看出，a 和 a.T 的遍历顺序是一样的，也就是他们在内存中的存储顺序也是一样的，但是 a.T.copy(order = 'C') 的遍历结果是不同的，那是因为它和前两种的存储方式是不一样的，默认是按行访问。

#### 控制遍历顺序

* for x in np.nditer(a, order='F'):Fortran order，即是列序优先；
* for x in np.nditer(a, order='C'):C order，即是行序优先；

实例

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print ('原始数组是：')

print (a) print ('\n')

print ('原始数组的转置是：')

b = a.T

print (b)

print ('\n')

print ('以 C 风格顺序排序：')

c = b.copy(order='C')

print (c)

for x in np.nditer(c):

print (x, end=", " )

print ('\n')

print ('以 F 风格顺序排序：')

c = b.copy(order='F')

print (c)

for x in np.nditer(c):

print (x, end=", " )

输出结果为：

原始数组是：

[[ 0 5 10 15]

[20 25 30 35]

[40 45 50 55]]

原始数组的转置是：

[[ 0 20 40]

[ 5 25 45]

[10 30 50]

[15 35 55]]

以 C 风格顺序排序：

[[ 0 20 40]

[ 5 25 45]

[10 30 50]

[15 35 55]]

0, 20, 40, 5, 25, 45, 10, 30, 50, 15, 35, 55,

以 F 风格顺序排序：

[[ 0 20 40]

[ 5 25 45]

[10 30 50]

[15 35 55]]

0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,

可以通过显式设置，来强制 nditer 对象使用某种顺序：

实例

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print ('原始数组是：')

print (a)

print ('\n')

print ('以 C 风格顺序排序：')

for x in np.nditer(a, order = 'C'):

print (x, end=", " )

print ('\n')

print ('以 F 风格顺序排序：')

for x in np.nditer(a, order = 'F'):

print (x, end=", " )

输出结果为：

原始数组是：

[[ 0 5 10 15]

[20 25 30 35]

[40 45 50 55]]

以 C 风格顺序排序：

0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,

以 F 风格顺序排序：

0, 20, 40, 5, 25, 45, 10, 30, 50, 15, 35, 55,

#### 修改数组中元素的值

nditer 对象有另一个可选参数 op\_flags。 默认情况下，nditer 将视待迭代遍历的数组为只读对象（read-only），为了在遍历数组的同时，实现对数组元素值得修改，必须指定 read-write 或者 write-only 的模式。

实例

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print ('原始数组是：')

print (a)

print ('\n')

for x in np.nditer(a, op\_flags=['readwrite']):

x[...]=2\*x #注意x[…]是语法，不是省略。

print ('修改后的数组是：') print (a)

输出结果为：

原始数组是：

[[ 0 5 10 15]

[20 25 30 35]

[40 45 50 55]]

修改后的数组是：

[[ 0 10 20 30]

[ 40 50 60 70]

[ 80 90 100 110]]

### 使用外部循环

nditer类的构造器拥有flags参数，它可以接受下列值：

|  |  |
| --- | --- |
| **参数** | **描述** |

|  |  |
| --- | --- |
| c\_index | 可以跟踪 C 顺序的索引 |

|  |  |
| --- | --- |
| f\_index | 可以跟踪 Fortran 顺序的索引 |

|  |  |
| --- | --- |
| multi-index | 每次迭代可以跟踪一种索引类型 |

|  |  |
| --- | --- |
| external\_loop | 给出的值是具有多个值的一维数组，而不是零维数组 |

在下面的实例中，迭代器遍历对应于每列，并组合为一维数组。因为指定了order=’F’。

实例

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print (a)

for x in np.nditer(a, flags = ['external\_loop'], order = 'F'):

print (x, end=", " )

输出结果为：

原始数组是：

[[ 0 5 10 15]

[20 25 30 35]

[40 45 50 55]]

修改后的数组是：

[ 0 20 40], [ 5 25 45], [10 30 50], [15 35 55],

### 广播迭代

如果两个数组是可广播的，nditer 组合对象能够同时迭代它们。 假设数组 a 的维度为 3X4，数组 b 的维度为 1X4 ，则使用以下迭代器（数组 b 被广播到 a 的大小）。

实例

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print (a)

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

print (b)

for x,y in np.nditer([a,b]):

print ("%d:%d" % (x,y), end=", " )

输出结果为：

第一个数组为：

[[ 0 5 10 15]

[20 25 30 35]

[40 45 50 55]]

第二个数组为：

[1 2 3 4]

修改后的数组为：

0:1, 5:2, 10:3, 15:4, 20:1, 25:2, 30:3, 35:4, 40:1, 45:2, 50:3, 55:4,

## 生成矩阵

NumPy 中包含了一个矩阵库 numpy.matlib，该模块中的函数返回的是一个矩阵，而不是 ndarray 对象。

### 1. 使用 mat 方法将 2 维数组转化为矩阵：

import numpy as np

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

[2,5,3],

[7,8,9]])

A = np.mat(a)

A

matrix([[1, 2, 4],

[2, 5, 3],

[7, 8, 9]])

### 2.传入字符串生成矩阵

\*\*Matlab\*\* 的语法传入一个字符串来生成矩阵：

A = np.mat('1,2,4;2,5,3;7,8,9') #不同的行用分号隔开

A

matrix([[1, 2, 4],

[2, 5, 3],

[7, 8, 9]])

### 3. 利用分块创造新的矩阵：bmat

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

b = np.array([[10,20], [30,40]])

np.bmat('a,b;b,a')

输出：

matrix([[ 1, 2, 10, 20],

[ 3, 4, 30, 40],

[10, 20, 1, 2],

[30, 40, 3, 4]])

### 4.矩阵与向量的乘积：

A = np.mat('1,2,4;2,5,3;7,8,9')

x = np.array([[1], [2], [3]])

A \* x

matrix([[17],

[21],

[50]])

注意：矩阵与向量的乘积返回的仍然是矩阵

### 5.逆矩阵

A.I 大写的i，表示A矩阵的逆矩阵。

### 6.矩阵指数幂

A\*\*4 = A\*A\*A\*A

A\*\*-1 为A的逆矩阵

### 7. matlib.empty()

matlib.empty() 函数返回一个新的矩阵，语法格式为：

numpy.matlib.empty(shape, dtype, order)

**参数说明：**

**shape**: 定义新矩阵形状的整数或整数元组

**Dtype**: 可选，数据类型

**order**: C（行序优先） 或者 F（列序优先）

实例

import numpy.matlib

import numpy as np

print (np.matlib.empty((2,2))) # 填充为随机数据

输出结果为：

[[-1.49166815e-154 -1.49166815e-154]

[ 2.17371491e-313 2.52720790e-212]]

### 8. numpy.matlib.zeros()

numpy.matlib.zeros() 函数创建一个以 0 填充的矩阵。

实例

import numpy.matlib

import numpy as np

print (np.matlib.zeros((2,2)))

输出结果为：

[[0. 0.]

[0. 0.]]

### 9. numpy.matlib.ones()

numpy.matlib.ones()函数创建一个以 1 填充的矩阵。

实例

import numpy.matlib import

numpy as np

print (np.matlib.ones((2,2)))

输出结果为：

[[1. 1.]

[1. 1.]]

### 10. numpy.matlib.eye()

numpy.matlib.eye() 函数返回一个矩阵，对角线元素为 1，其他位置为零。可以不是方阵。

numpy.matlib.eye(n, M,k, dtype)

**参数说明：**

**n**: 返回矩阵的行数

**M**: 返回矩阵的列数，默认为 n

**k**: 对角线的索引

**dtype**: 数据类型

实例

import numpy.matlib

import numpy as np

print (np.matlib.eye(n = 3, M = 4, k = 0, dtype = float))

输出结果为：

[[1. 0. 0. 0.]

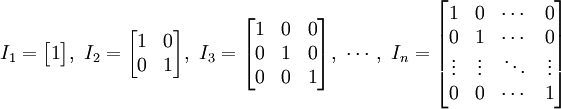
[0. 1. 0. 0.]

[0. 0. 1. 0.]]

### 11.numpy.matlib.identity()

numpy.matlib.identity() 函数返回给定大小的单位矩阵。

单位矩阵是个方阵，从左上角到右下角的对角线（称为主对角线）上的元素均为 1，除此以外全都为 0。



实例

import numpy.matlib

import numpy as np # 大小为 5，类型位浮点型

print (np.matlib.identity(5, dtype = float))

输出结果为：

[[ 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.]]

### 12.numpy.matlib.rand()

numpy.matlib.rand() 函数创建一个给定大小的矩阵，数据是随机填充的。

实例

import numpy.matlib

import numpy as np

print (np.matlib.rand(3,3))

输出结果为：

[[0.23966718 0.16147628 0.14162 ]

[0.28379085 0.59934741 0.62985825]

[0.99527238 0.11137883 0.41105367]]

### 13.np.full(shape,num)

创建一个形状为shape的全是num的数组。

### 13.矩阵与数组的转化asarray、asmatrix、mat

矩阵总是二维的，而 ndarray 是一个 n 维数组。 两个对象都是可互换的。

实例

import numpy.matlib

import numpy as np

i = np.matrix('1,2;3,4')

print (i)

输出结果为：

[[1 2]

[3 4]]

实例

import numpy.matlib import numpy as np

j = np.asarray(i)

print (j)

输出结果为：

[[1 2]

[3 4]]

实例

import numpy.matlib import numpy as np

k = np.asmatrix (j)

print (k)

输出结果为：

[[1 2]

[3 4]]

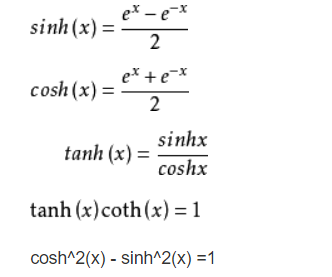
## 函数

### 1.三角函数

x为弧度。

通过 numpy.degrees()转换弧度为角度

sin(x)、cos(x)、tan(x)

sinh(x) 双曲正弦函数 conh(x) tanh(x)

arccos(x)、 arctan(x)、 arcsin(x)、 arccosh(x)

arctanh(x) 、 arcsinh(x)

arctan2(x,y)

arctan2(x,y) 返回 arctan(x/y) 。

### 2.舍入函数

#### numpy.around()

函数返回指定数字的四舍五入值。

注意，around对5是朝向远离0的位置进行舍入。如1.5🡪2.0 -1.5🡪-2.0

numpy.around(a,decimals)

参数说明：

* a: 数组
* decimals: 舍入的小数位数。 默认值为0。 如果为负，整数将四舍五入到小数点左侧的位置。如-1是对个位四舍五入，-2是对十位四舍五入。

实例

import numpy as np

a = np.array([1.0,5.55, 123, 0.567, 25.532])

print (a)

print ('舍入后：')

print (np.around(a))

print (np.around(a, decimals = 1))

print (np.around(a, decimals = -1))

输出结果为：

原数组：

[ 1. 5.55 123. 0.567 25.532]

舍入后：

[ 1. 6. 123. 1. 26.]

[ 1. 5.6 123. 0.6 25.5]

[ 0. 10. 120. 0. 30.]

#### numpy.fix()

也是四舍五入，但是对.5的处理时靠近0的方向。如1.5🡪1 -1.5🡪-1

#### numpy.floor()

numpy.floor() 返回数字的下舍整数。

向下取整。

实例

import numpy as np

a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print ('提供的数组：')

print (a)

print ('修改后的数组：')

print (np.floor(a))

输出结果为：

提供的数组：

[-1.7 1.5 -0.2 0.6 10. ]

修改后的数组：

[-2. 1. -1. 0. 10.]

#### numpy.ceil()

numpy.ceil() 返回数字的上入整数。

向上取整

实例

import numpy as np

a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print (a)

print ('修改后的数组：')

print (np.ceil(a))

输出结果为：

提供的数组：

[-1.7 1.5 -0.2 0.6 10. ]

修改后的数组：

[-1. 2. -0. 1. 10.]

#### trunc

根据指定位数进行直接截断。默认为整数位。直接扔掉小数位，保留整数位。

### 3.NumPy 统计函数

#### 1.numpy.amin() 和 numpy.amax()

numpy.amin() 用于计算数组中的元素沿指定轴的最小值。

numpy.amax() 用于计算数组中的元素沿指定轴的最大值。

print (np.amin(a,0))

print (np.amax(a,0))

#### 2.np.max()、np.min

| a.min(axis=None) |最小值

| a.max(axis=None) |最大值

默认为整个数组，指定维度，则返回该维度的最大最小值。

#### 3. a.ptp(axis=None) / np.ptp()

最大值减最小值

默认为全局，可以指定维度

#### 4. numpy.percentile()

百分位数是统计中使用的度量，表示小于这个值的观察值的百分比。 函数numpy.percentile()接受以下参数。

numpy.percentile(a, q, axis)

参数说明：

* a: 输入数组
* q: 要计算的百分位数，在 0 ~ 100 之间
* axis: 沿着它计算百分位数的轴

**首先明确百分位数：**

第 p 个百分位数是这样一个值，它使得至少有 p% 的数据项小于或等于这个值，且至少有 (100-p)% 的数据项大于或等于这个值。

举个例子：高等院校的入学考试成绩经常以百分位数的形式报告。比如，假设某个考生在入学考试中的语文部分的原始分数为 54 分。相对于参加同一考试的其他学生来说，他的成绩如何并不容易知道。但是如果原始分数54分恰好对应的是第70百分位数，我们就能知道大约70%的学生的考分比他低，而约30%的学生考分比他高。

这里的 p = 70。

实例

注意下列实例：在全局范围中p=50%，即大于50%的最后一个数为4，小于50%的第一个数为3，所以返回值为3.5。取的平均值。

import numpy as np

a = np.array([[10, 7, 4], [3, 2, 1]])

# 50% 的分位数，就是 a 里排序之后的中位数

print (np.percentile(a, 50)) # axis 为 0，在纵列上求

print (np.percentile(a, 50, axis=0)) # axis 为 1，在横行上求

print (np.percentile(a, 50, axis=1)) # 保持维度不变

print (np.percentile(a, 50, axis=1, keepdims=True))

输出结果为：

我们的数组是：

[[10 7 4]

[ 3 2 1]]

调用 percentile() 函数：

3.5

[6.5 4.5 2.5]

[7. 2.]

[[7.]

[2.]]

#### 5.numpy.median()

numpy.median() 函数用于计算数组 a 中元素的中位数（中值）

实例

import numpy as np

a = np.array([[30,65,70],[80,95,10],[50,90,60]])

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 ('沿轴 1 调用 median() 函数：')

print (np.median(a, axis = 1))

输出结果为：

我们的数组是：

[[30 65 70]

[80 95 10]

[50 90 60]]

调用 median() 函数：

65.0

沿轴 0 调用 median() 函数：

[50. 90. 60.]

沿轴 1 调用 median() 函数：

[65. 80. 60.]

#### 6. numpy.mean()

numpy.mean() 函数返回数组中元素的算术平均值。 如果提供了轴，则沿其计算。

算术平均值是沿轴的元素的总和除以元素的数量。

实例

import numpy as np

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

print (a)

print (np.mean(a))

print ('沿轴 0 调用 mean() 函数：')

print (np.mean(a, axis = 0))

print ('沿轴 1 调用 mean() 函数：')

print (np.mean(a, axis = 1))

输出结果为：

我们的数组是：

[[1 2 3]

[3 4 5]

[4 5 6]]

调用 mean() 函数：

3.6666666666666665

沿轴 0 调用 mean() 函数：

[2.66666667 3.66666667 4.66666667]

沿轴 1 调用 mean() 函数：

[2. 4. 5.]

#### 7. numpy.average()

numpy.average() 函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值。

该函数可以接受一个轴参数。 如果没有指定轴，则数组会被展开。

加权平均值即将各数值乘以相应的权数，然后加总求和得到总体值，再除以总的单位数。

考虑数组[1,2,3,4]和相应的权重[4,3,2,1]，通过将相应元素的乘积相加，并将和除以权重的和，来计算加权平均值。

加权平均值 = (1\*4+2\*3+3\*2+4\*1)/(4+3+2+1)

实例

import numpy as np

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

print ('调用 average() 函数：')

print (np.average(a)) # 不指定权重时相当于 mean 函数

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

print ('再次调用 average() 函数：')

print (np.average(a,weights = wts))

# 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 = [4,3,2,1], returned = True))

输出结果为：

我们的数组是：

[1 2 3 4]

调用 average() 函数：

2.5

再次调用 average() 函数：

2.0

权重的和：

(2.0, 10.0)

在多维数组中，可以指定用于计算的轴。

实例

import numpy as np

a = np.arange(6).reshape(3,2)

wt = np.array([3,5])

print (np.average(a, axis = 1, weights = wt))

print ('修改后的数组：')

print (np.average(a, axis = 1, weights = wt, returned = True))

输出结果为：

我们的数组是：

[[0 1]

[2 3]

[4 5]]

修改后的数组：

[0.625 2.625 4.625]

修改后的数组：

(array([0.625, 2.625, 4.625]), array([8., 8., 8.]))

#### 8.标准差std

标准差是一组数据平均值分散程度的一种度量。

标准差是方差的算术平方根。

标准差公式如下：

std = sqrt(mean((x - x.mean())\*\*2))

如果数组是 [1，2，3，4]，则其平均值为 2.5。 因此，差的平方是 [2.25,0.25,0.25,2.25]，并且其平均值的平方根除以 4，即 sqrt(5/4) ，结果为 1.1180339887498949。

实例

import numpy as np

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

输出结果为：

1.1180339887498949

#### 9.方差var

统计中的方差（样本方差）是每个样本值与全体样本值的平均数之差的平方值的平均数，即 mean((x - x.mean())\*\* 2)。

换句话说，标准差是方差的平方根。

实例

import numpy as np

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

输出结果为：

1.25

#### 10.corrcoef 求相关系数

#### 11.cov 求协方差

### 4.排序

**3.4排序**  
像Python的内建列表一样，NumPy数组也可以使用 **sort** 方法就地排序

Numpy有一些基本的针对一维ndarrays的集合操作。最常使用的一个可能是 **np.unique** ，它返回一个数组的经过排序的 *unique* 值：

| 数组集合操作 | |
| --- | --- |
| **unique(x)** | **计算x单一的元素，并对结果排序** |
| intersect1d(x, y) | 计算x和y相同的元素，并对结果排序 |
| union1d | 结合x和y的元素，并对结果排序 |
| in1d(x, y) | 得到一个布尔数组指示x中的每个元素是否在y中 |
| setdiff1d(x, y) | 差集，在x中但不再y中的集合 |
| setxor1d(x, y) | 对称差集，不同时在两个数组中的元素 |

NumPy 提供了多种排序的方法。 这些排序函数实现不同的排序算法，每个排序算法的特征在于执行速度，最坏情况性能，所需的工作空间和算法的稳定性。 下表显示了三种排序算法的比较。

种类 速度 最坏情况 工作空间 稳定性

'quicksort'（快速排序） 1 O(n^2) 0 否

'mergesort'（归并排序） 2 O(n\*log(n)) ~n/2 是

'heapsort'（堆排序） 3 O(n\*log(n)) 0 否

#### ①numpy.sort()

numpy.sort() 函数返回输入数组的排序副本。函数格式如下：

numpy.sort(a, axis, kind, order)

默认沿最后一轴进行排序，如果想全局排序，可以先把a转为1维，用flatten。

参数说明：

* a: 要排序的数组
* axis: 沿着它排序数组的轴，如果没有数组会被展开，沿着最后的轴排序， axis=0 按列排序，axis=1 按行排序
* kind: 默认为'quicksort'（快速排序）
* order: 如果数组包含字段，则是要排序的字段

实例

import numpy as np

a = np.array([[3,7],[9,1]])

print ('调用 sort() 函数：')

print (np.sort(a))

print ('按列排序：')

print (np.sort(a, axis = 0))

# 在 sort 函数中排序字段

dt = np.dtype([('name', 'S10'),('age', int)])

a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)

print ('按 name 排序：')

print (np.sort(a, order = 'name'))

输出结果为：

我们的数组是：

[[3 7]

[9 1]]

调用 sort() 函数：

[[3 7]

[1 9]]

按列排序：

[[3 1]

[9 7]]

我们的数组是：

[(b'raju', 21) (b'anil', 25) (b'ravi', 17) (b'amar', 27)]

按 name 排序：

[(b'amar', 27) (b'anil', 25) (b'raju', 21) (b'ravi', 17)]

#### ②numpy.argsort()

numpy.argsort() 函数返回的是数组值从小到大的索引值。

实例

import numpy as np

x = np.array([3, 1, 2])

print ('对 x 调用 argsort() 函数：')

y = np.argsort(x)

print (y)

print (x[y])（一维的才可以这样用，二维的就不能这样，可以这样print(a[0,x[0,:]])）

print ('使用循环重构原数组：')

for i in y: print (x[i], end=" ")

输出结果为：

我们的数组是：

[3 1 2]

对 x 调用 argsort() 函数：

[1 2 0]

以排序后的顺序重构原数组：

[1 2 3]

使用循环重构原数组

1 2 3

#### ③numpy.lexsort()

numpy.lexsort() 用于对多个序列进行排序。把它想象成对电子表格进行排序，每一列代表一个序列，排序时优先照顾靠后的列。

numpy.lexsort(a1,a2,..)

先按a1排序，如果遇到相同的值，在相同值中按照a2排序。。。。。

这里举一个应用场景：小升初考试，重点班录取学生按照总成绩录取。在总成绩相同时，数学成绩高的优先录取，在总成绩和数学成绩都相同时，按照英语成绩录取…… 这里，总成绩排在电子表格的最后一列，数学成绩在倒数第二列，英语成绩在倒数第三列。

实例

nm = ('raju','anil','ravi','amar')

dv = ('f.y.', 's.y.', 's.y.', 'f.y.')

ind = np.lexsort((dv,nm))

print (ind)

print ('使用这个索引来获取排序后的数据：')

print ([nm[i] + ", " + dv[i] for i in ind])

输出结果为：

调用 lexsort() 函数：

[3 1 0 2]

使用这个索引来获取排序后的数据：

['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']

上面传入 np.lexsort 的是一个tuple，排序时首先排 nm，顺序为：amar、anil、raju、ravi 。综上排序结果为 [3 1 0 2]。

#### ④msort、sort\_complex、partition、argpartition

msort(a) 数组按第一个轴排序，返回排序后的数组副本。np.msort(a) 相等于 np.sort(a, axis=0)。

sort\_complex(a) 对复数按照先实部后虚部的顺序进行排序。

partition(a, kth[, axis, kind, order]) 指定一个数，对数组进行分区，返回该数组前几小的值。比如返回a数组前3小的值，则返回一个新数组，左边三个为a数组中前三小的值，后面为其余元素。这两个分割区内的元素都是任意排列的。

argpartition(a, kth[, axis, kind, order]) 可以通过关键字 kind 指定算法沿着指定轴对数组进行分区

复数排序：

>>> import numpy as np

>>> np.sort\_complex([5, 3, 6, 2, 1])

array([ 1.+0.j, 2.+0.j, 3.+0.j, 5.+0.j, 6.+0.j])

>>>

>>> np.sort\_complex([1 + 2j, 2 - 1j, 3 - 2j, 3 - 3j, 3 + 5j])

array([ 1.+2.j, 2.-1.j, 3.-3.j, 3.-2.j, 3.+5.j])

partition() 分区排序：

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

>>> np.partition(a, 3) # 将数组 a 中所有元素（包括重复元素）从小到大排列，比第3小的放在前面，大的放在后面

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

>>>

>>> np.partition(a, (1, 3)) # 小于 1 的在前面，大于 3 的在后面，1和3之间的在中间

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

找到数组的第 3 小（index=2）的值和第 2 大（index=-2）的值

>>> arr = np.array([46, 57, 23, 39, 1, 10, 0, 120])

>>> arr[np.argpartition(arr, 2)[2]]

10

>>> arr[np.argpartition(arr, -2)[-2]]

57

同时找到第 3 和第 4 小的值。注意这里，用 [2,3] 同时将第 3 和第 4 小的排序好，然后可以分别通过下标 [2] 和 [3] 取得。

>>> arr[np.argpartition(arr, [2,3])[2]]

10

>>> arr[np.argpartition(arr, [2,3])[3]]

23

#### ⑤numpy.argmax() 和 numpy.argmin()

numpy.argmax() 和 numpy.argmin()函数分别沿给定轴返回最大和最小元素的索引。

#### ⑥numpy.nonzero()

numpy.nonzero() 函数返回输入数组中非零元素的索引。

#### ⑦numpy.where()

numpy.where() 函数返回输入数组中满足给定条件的元素的索引。

#### ⑧numpy.extract()

numpy.extract() 函数根据某个条件从数组中抽取元素，返回满足条件的元素。和where类似，不过是返回的元素值。

实例

np x = np.arange(9.).reshape(3, 3)

# 定义条件, 选择偶数元素

condition = np.mod(x,2) == 0

print (condition)

print ('使用条件提取元素：')

print (np.extract(condition, x))

输出结果为：

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.]

### 5.向量矩阵数组操作

#### ①dot(x,y) ：或者x.dot(y)

dot对于二维以上的数组就相当于矩阵乘法。

对一维数组就是对应相乘然后相加。类似于求内积。

而一维数组没有方向，其转置就等于其本身。

对于三维以上的数组，可以看做是分块矩阵。

如3\*4\*2的维数组：

可以看做是三个4\*2的矩阵组成的列向量。

A

B

C

其中A，B,C是三个4\*2的矩阵。

而2\*3\*4\*2则是：一共分成2\*3列的分块矩阵，每个矩阵是4\*2的小矩阵

A

B

C

D

E

F

A,B,C,D,E,F都是4\*2的小矩阵

以此类推。

对于高维的数组dot乘法的匹配：

如3维A：3\*4\*2

需匹配的数组为B：3\*2\*n

理解原因：A可以看做为一个一维数组里有三个元素，而三个元素分别为三个4\*2矩阵。由于一维数组的转置等于其本身，所以对应的B第三维也应该是3，后两维进行按照矩阵乘法，4\*2矩阵需要2\*n的矩阵来匹配。所以B大小为3\*2\*n

同理：A：3\*2\*4\*5 则B为2\*m\*5\*n

A为3\*3\*2\*4\*5 则B为3\*2\*m\*5\*n

规律：矩阵和一维数组相乘，一维数组为列向量。

2.如果x，y都是1维的数组：则为点乘，即相应位置乘以相应位置的数，然后相加

c=np.array([1,2,3])

d=np.array([1,2,3])

c.dot(d)

输出：14

3.x为多维数组，y为1维数组（注意先后顺序）： x的各维与y点乘。

d=np.array([1,2,3])

e=np.arange(1,7,1)

e.shape=2,3

e.dot(d)

输出：

array([14, 32])

f=np.arange(1,28,1)

f.shape=3,3,3

f

array([[[ 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]]])

f.dot(d)

array([[ 14, 32, 50],

[ 68, 86, 104],

[122, 140, 158]])

返回的数组比原数组少一维。

4.即是矩阵乘法。

#### ②x\*y :

矩阵\*矩阵：矩阵乘法

如果是矩阵\*（数组、列表、元组），则把后面的转换为行向量，返回dot

**if** isinstance**(**other**,** **(**N**.**ndarray**,** list**,** tuple**))** **:**

**# This promotes 1-D vectors to row vectors**

**--> 215 return** N**.**dot**(**self**,** asmatrix**(**other**))**

如果x，y有一个为矩阵，则为矩阵乘法，

如果x，y都为数组，则为逐元素相乘，如果x、y大小不一样，若符合广播规则，则广播，否则报错。

#### ③inner(x,y)

numpy.inner() 函数返回一维数组的向量内积。对于更高的维度，它返回最后一个轴上的和的乘积。

对于矩阵：

np.inner(a,b) =a \* b.T

自动将第二个矩阵转置，然后求矩阵乘积。

对于np.inner(矩阵，数组)也是这样，将数组转置。

对于数组和数组：

和dot乘积一样，对于两个一维数组，计算的是这两个数组对应下标元素的乘积和；

对于多维数组，和dot不一样，它计算的结果数组中的每个元素都是：数组a和b的最后一维的内积，因此数组a和b的最后一维的长度必须相同：

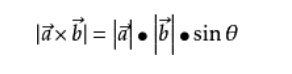
inner(a, b)[i,j] = sum(a[i,j,:]\*b[i,j,:])

#### ④outer(x,y)

将数组或者矩阵展开成1维，按照前面为列向量，后面为行向量的顺序，进行矩阵乘法，获得一个数组

#### ⑤cross（x,y）

返回向量的叉积。获得一个新的向量，与原两个向量垂直。



#### ⑥vdot(x,y)

numpy.vdot() 函数是两个向量的点积。 如果第一个参数是复数，那么它的共轭复数会用于计算。 如果参数是多维数组，它会被展开。

实例

import numpy as np

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

b = np.array([[11,12],[13,14]]) # vdot 将数组展开计算内积

print (np.vdot(a,b))

输出结果为：

130

计算式为：

1\*11 + 2\*12 + 3\*13 + 4\*14 = 130

#### ⑦numpy.matmul

numpy.matmul 函数返回两个数组的矩阵乘积。 虽然它返回二维数组的正常乘积，但如果任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。

另一方面，如果任一参数是一维数组，则通过在其维度上附加 1 来将其提升为矩阵，并在乘法之后被去除。

对于二维数组，它就是矩阵乘法：

实例

import numpy.matlib

import numpy as np

a = [[1,0],[0,1]]

b = [[4,1],[2,2]]

print (np.matmul(a,b))

输出结果为：

[[4 1]

[2 2]]

二维和一维运算：

实例

import numpy.matlib

import numpy as np

a = [[1,0],[0,1]]

b = [1,2]

print (np.matmul(a,b))

print (np.matmul(b,a))

输出结果为：

[1 2]

[1 2]

维度大于二的数组 ：

实例

import numpy.matlib

import numpy as np

a = np.arange(8).reshape(2,2,2)

b = np.arange(4).reshape(2,2)

print (np.matmul(a,b))

输出结果为：

[[[ 2 3]

[ 6 11]]

[[10 19]

[14 27]]]

#### ⑧numpy.linalg.det() 求行列式

numpy.linalg.det() 函数计算输入矩阵的行列式。

行列式在线性代数中是非常有用的值。 它从方阵的对角元素计算。 对于 2×2 矩阵，它是左上和右下元素的乘积与其他两个的乘积的差。

换句话说，对于矩阵[[a，b]，[c，d]]，行列式计算为 ad-bc。 较大的方阵被认为是 2×2 矩阵的组合。

实例

import numpy as np

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

print (np.linalg.det(a))

输出结果为：

-2.0

实例

import numpy as np

b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])

print (b) print (np.linalg.det(b))

print (6\*(-2\*7 - 5\*8) - 1\*(4\*7 - 5\*2) + 1\*(4\*8 - -2\*2))

输出结果为：

[[ 6 1 1]

[ 4 -2 5]

[ 2 8 7]]

-306.0

-306

#### ⑨numpy.linalg.solve()

numpy.linalg.solve(a,b) 函数给出了矩阵形式的线性方程的解。

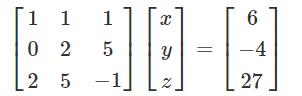
考虑以下线性方程：

x + y + z = 6

2y + 5z = -4

2x + 5y - z = 27

可以使用矩阵表示为：



如果矩阵成为A、X和B，方程变为：

AX = B

或

X = A^(-1)B

#### ⑩numpy.linalg.inv()求逆

numpy.linalg.inv() 函数计算矩阵的乘法逆矩阵。

**逆矩阵（inverse matrix）**：设A是数域上的一个n阶矩阵，若在相同数域上存在另一个n阶矩阵B，使得： AB=BA=E ，则我们称B是A的逆矩阵，而A则被称为可逆矩阵。注：E为单位矩阵。

实例

import numpy as np x = np.array([[1,2],[3,4]]) y = np.linalg.inv(x) print (x) print (y) print (np.dot(x,y))

输出结果为：

[[1 2]

[3 4]]

[[-2. 1. ]

[ 1.5 -0.5]]

[[1.0000000e+00 0.0000000e+00]

[8.8817842e-16 1.0000000e+00]]

现在创建一个矩阵A的逆矩阵:

实例

import numpy as np a = np.array([[1,1,1],[0,2,5],[2,5,-1]]) print ('数组 a：') print (a) ainv = np.linalg.inv(a) print ('a 的逆：') print (ainv) print ('矩阵 b：') b = np.array([[6],[-4],[27]]) print (b) print ('计算：A^(-1)B：') x = np.linalg.solve(a,b) print (x) # 这就是线性方向 x = 5, y = 3, z = -2 的解

输出结果为：

数组 a：

[[ 1 1 1]

[ 0 2 5]

[ 2 5 -1]]

a 的逆：

[[ 1.28571429 -0.28571429 -0.14285714]

[-0.47619048 0.14285714 0.23809524]

[ 0.19047619 0.14285714 -0.0952381 ]]

矩阵 b：

[[ 6]

[-4]

[27]]

计算：A^(-1)B：

[[ 5.]

[ 3.]

[-2.]]

结果也可以使用以下函数获取：

x = np.dot(ainv,b)

#### (11)kron(x,y)

后面矩阵和前面矩阵每个元素相乘。

A=[1 2; 3 4]; B=[1 3 2; 2 4 6]; C=kron(A,B)

C =

1 3 2 2 6 4

B

2B

3B

4B

2 4 6 4 8 12

3 9 6 4 12 8

6 12 18 8 16 24

### 6.其他常用矩阵操作

numpy 中还包含一些代数运算的方法，尤其是涉及到矩阵的计算方法，求解特征值、特征向量、逆矩阵等，非常方便。

numpy.linalg.cholesky(a)：Cholesky 分解。是把一个对称正定的矩阵表示成一个下三角矩阵L和其转置的乘积的分解 A=LLT

numpy.linalg.qr(a ,mode)：计算矩阵的 QR 因式分解。

numpy.linalg.svd(a ,full\_matrices,compute\_uv)：奇异值分解。

numpy.linalg.eig(a)：计算正方形数组的特征值和右特征向量。

numpy.linalg.eigh(a, UPLO)：返回 Hermitian（共轭矩阵）或对称矩阵的特征值和特征向量。

numpy.linalg.eigvals(a)：计算矩阵的特征值。

numpy.linalg.eigvalsh(a, UPLO)：计算 Hermitian（共轭矩阵） 或真实对称矩阵的特征值。

numpy.linalg.norm(x ,ord,axis,keepdims)：计算矩阵或向量范数。

numpy.linalg.cond(x ,p)：计算矩阵的条件数。

numpy.linalg.det(a)：计算数组的行列式。

numpy.linalg.matrix\_rank(M ,tol)：使用奇异值分解方法返回秩。

numpy.linalg.slogdet(a)：计算数组的行列式的符号和自然对数。

numpy.trace(a ,offset,axis1,axis2,dtype,out)：沿数组的对角线返回总和。

numpy.linalg.solve(a,b)：求解线性矩阵方程或线性标量方程组。

numpy.linalg.tensorsolve(a,b ,axes)：为 x 解出张量方程a x = b

numpy.linalg.lstsq(a,b ,rcond)：将最小二乘解返回到线性矩阵方程。

numpy.linalg.inv(a)：计算逆矩阵。

numpy.linalg.pinv(a ,rcond)：计算矩阵的（Moore-Penrose）伪逆。

diag 返回矩阵对角线元素

numpy.linalg.tensorinv(a ,ind)：计算N维数组的逆。

matrix.T transpose： 返回矩阵的转置矩阵  
matrix.I inverse：返回矩阵的逆矩阵  
matrix.H hermitian (conjugate) transpose：返回复数矩阵的共轭元素矩阵  
matrix.A base array：返回矩阵基于的数组

### 7.多项式

poly 给定一组根，寻找多项式系数  
roots 给顶一组系数，寻找根  
polyint 求积分  
polyder 求导数  
polyval 求函数在某一点的值  
polyfit 使用最小二乘法做曲线拟合  
polyadd, polysub, polymul, 和 polydiv 函数处理多项式系统的加，减，乘除

### 8.类型处理及判断：

iscomplexobj：判断数组类型是否为复数。

iscomplex：返回一个bool数组，对应位置为复数，则为True

isrealobj

isreal

imag

real

real\_if\_close，如果虚部接近0，则返回一个实数。

np.real\_if\_close([2.1 + 4e-14j], tol=1000)

array([ 2.1] tol为类似于定义精度。

isscalar 是否为标量

isneginf 是否为负无穷

isposinf 是否为正无穷

isinf 是否为无穷

isfinite

isnan：判断是否为nan

nan\_to\_num

common\_type

typename

np.inf：正无穷

-np.inf :负无穷

np.nan：非法值（Not a number），对于0/0的计算，并不会报错，而是返回一个nan。而对于x/0 （x不等于0），则会返回无穷。

nan 与任何数进行比较都是 False ：包括和他本身比。判断是否为nan，使用isnan。

### select：

select(条件列表，执行列表，default=0)

满足条件列表中的条件，则对数组执行执行列表中的操作，可以有多个条件，对应多个操作。相当于if/elif/…

<https://www.jianshu.com/p/f014e1e608c1>

### choose

使用choose进行选择

choose()函数的定义

def choose(a, choices, out=None, mode='raise'):

参数 a ：它必须是一个 int 型的 数组，并且 a 中的元素，必须是0~n-1之间的数，这里的n表示的就是数组choices数组最外层的维度数。

choices：表示的是要操作的数组，要注意的是choices的数组的维度是一定要和a进行匹配的，如果匹配不了，会出现错误。

参数out：接收运算结果的数组，它的维度一定要和 a 是一样的，是可选参数。

参数mode：默认的是raise，表示的是a数组中的元素不能超过 n ，她还有两个可选值，clip：将 a 中的 元素 如果小于0，则将其变为0，如果大于n-1，则变为n-1，

wrap：将a中的值 value变为value mod n，即值除以n的余数。

##### 用法一：生成一个数组

control = np.array([[1,0,1], [2,1,0], [1,2,2]])

np.choose(control, [10, 11, 12])

输出：

array([[11, 10, 11],

[12, 11, 10],

[11, 12, 12]])

choose 将 0,1,2 对应的值映射为了 10, 11, 12 ，这里的 0,1,2 表示对应的下标。

也就是a指定了数组的维数，返回的数组按照a提供的顺序，choice提供的内容进行排列。

##### 用法二：

事实上， choose 不仅仅能接受下标参数，还可以接受下标所在的位置：

i0 = np.array([[0,1,2],

[3,4,5],

[6,7,8]])

i2 = np.array([[20,21,22],

[23,24,25],

[26,27,28]])

control = np.array([[1,0,1],

[2,1,0],

[1,2,2]])

np.choose(control, [i0, 10, i2])

输出：

array([[10, 1, 10],

[23, 10, 5],

[10, 27, 28]])

这里， control 传入第一个 1 对应的是 10，传入的第一个 0 对应于 i0 相应位置的值即 1 ，剩下的以此类推。第一个0的位置是 第一行第二个，则选择io第一行第二个数进行填充。

##### 用法三：

**（3）当a的维数比choices的维数少的时候——a为1维，choices为2维**

cc=np.choose([4,2,1,3,0],[[11,22,33,32,31],[44,55,66,65,64],[77,88,99,98,97],[111,222,333,332,331],[444,555,666,665,664]])

返回：[444  88  66 332  31]

a中各数值代表choice中的第一维，而数值所在的索引代表第二维。[4,2,1,3,0]

4的索引是0，则代表选择choice中索引是40的元素。即第五行第一个444。

##### 用法四：结合条件数组

将数组中所有小于 10 的值变成了 10 ：

b= np.array([[ 0, 1, 2],

[10,11,12],

[20,21,22]]

np.choose(a < 10, (b, 10))

输出：array([[10, 10, 10],

[10, 11, 12],

[20, 21, 22]])

分析：a<10其实就是个bool数组：

array([[ True, True, True],

[False, False, False],

[False, False, False]], dtype=bool)

这个数组相当于输入a，True就是1，Flase就是0。

choice就是（b,10）,参见用法二。如果是1（True），就选择choice相应位置的10。如果是0（Flase），就要结合a数组自身位置，选择b数组相应位置的数值进行填充。结果下来，就是对<10的补为10，大于10的不变。

例二：

下面的例子将数组中所有小于 10 的值变成了 10，大于 15 的值变成了 15。

a = np.array([[ 0, 1, 2],

[10,11,12],

[20,21,22]])

lt = a < 10

gt = a > 15

choice = lt + 2 \* gt

choice

c=np.choose(choice, (a, 10, 15))

输出：choice：array([[1, 1, 1],

[0, 0, 0],

[2, 2, 2]])

c: array([[10, 10, 10],

[10, 11, 12],

[15, 15, 15]])

和上述例子一样，先构造一个数组，然后进行选择填充。

### where语句

where的两种用法：

#### 1. np.where(condition, x, y)

满足条件(condition)，输出x，不满足输出y。  
  
  
如果是一维数组，相当于[xv if c else yv for (c,xv,yv) in zip(condition,x,y)]

>>> aa = np.arange(10)

>>> np.where(aa,1,-1)

array([-1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) # 0为False，所以第一个输出-1

>>> np.where(aa > 5,1,-1)

array([-1, -1, -1, -1, -1, -1, 1, 1, 1, 1])

>>> np.where([[True,False], [True,True]], # 官网上的例子

[[1,2], [3,4]],

[[9,8], [7,6]])

array([[1, 8],

[3, 4]])

上面这个例子的条件为[[True,False], [True,False]]，分别对应最后输出结果的四个值。第一个值从[1,9]中选，因为条件为True，所以是选1。第二个值从[2,8]中选，因为条件为False，所以选8，后面以此类推。类似的问题可以再看个例子：

>>> a = 10

>>> np.where([[a > 5,a < 5], [a == 10,a == 7]],

[["chosen","not chosen"], ["chosen","not chosen"]],

[["not chosen","chosen"], ["not chosen","chosen"]])

array([['chosen', 'chosen'],

['chosen', 'chosen']], dtype='<U10')

#### 2. np.where(condition)

只有条件 (condition)，没有x和y，则输出满足条件 (即非0) 元素的坐标 (等价于[numpy.nonzero](https://docs.scipy.org/doc/numpy/reference/generated/numpy.nonzero.html#numpy.nonzero))。这里的坐标以tuple的形式给出，通常原数组有多少维，输出的tuple中就包含几个数组，分别对应符合条件元素的各维坐标。

>>> a = np.array([2,4,6,8,10])

>>> np.where(a > 5) # 返回索引

(array([2, 3, 4]),)

>>> a[np.where(a > 5)] # 等价于 a[a>5]

array([ 6, 8, 10])

>>> np.where([[0, 1], [1, 0]])

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

上面这个例子条件中[[0,1],[1,0]]的真值为两个1，各自的第一维坐标为[0,1]，第二维坐标为[1,0] 。

where(array)：返回所有非零元素的索引。

#### 一维数组：

a = array([0, 12, 5, 20])

判断数组中的元素是不是都大于10：

y = a > 10

y是一个和a维数相同的布尔类型的array

返回数组中a>10的索引：

where(a>10)

返回值：(array([1, 3], dtype=int64),)

注意到 where 的返回值是一个元组。

使用元组是由于 where 可以对多维数组使用，此时返回值就是多维的。

取返回的索引位置可以：

indice= where(a>10)[0]

可以直接用where的返回值进行索引

a[where(a>10)]

返回了一个新数组，与原数组独立。

#### 二维、多维数组

a = array([[0, 12, 5, 20],  
 [1, 2, 11, 15]])  
loc = where(a > 10)

返回一个二维元组，元组的每一个值代表一维的索引值

返回:

(array([0, 0, 1, 1], dtype=int64), array([1, 3, 2, 3], dtype=int64))

也可以直接用来索引a：

a[loc]

返回：array([12, 20, 11, 15]) 注意是个一维数组，并且与原数组是相互独立的。

或者进行解包

rows, cols = where(a>10)

a[rows, cols]

### any：

数组有一个不为0，返回True

### all：

数组全不为0，返回True

### disp

### nansum：

忽略nan的累加

### nanargmax

### nanargmin

### nanmin

### nanmax

nan 开头的函数会进行相应的操作，但是忽略 nan 值。

## 向量化函数

对于自定义的作用于单个值的函数，可以使用 numpy 的 vectorize 将函数向量化，产生一个新的函数：

vfunc = np.vectorize(func)

函数cfunc就可以作用于向量，其实就是向量中的每个值都会调用该函数一次。

因为这样的用法涉及大量的函数调用，因此，向量化函数的效率并不高。

## 其他有用运算

#### mod：求模

#### sum

#### cumsum

返回相应位置元素与前面元素相加之和。

[1,2,3,4]🡪[1,2,6,10]

#### prod

元素乘积

#### cumprod

#### diff

#### angle

#### unwrap

#### sort\_complex

#### trim\_zeros

#### fliplr

#### flipud

#### rot90

#### log、Iog10、log2、log1p：

分别为自然对数（底数为e)、底数为10的log、底数为2的log、log(1+X)

#### square：计算平方

#### sign：

计算各元素的正负号：1(正数）、0(零）、一1(负数）

## 常用一元运算

类型 说明

abs、fabs 计算整数、浮点数或复数的绝对值。对于非复数值，可以使用更快的fabs

sqrt 计算各元素的平方根。相当于arr\*\*0.5

square 计算各元素的平方。相当于arr\*\*2

exp 计算各元素的指数ex

log、Iog10、log2、loglp 分别为自然对数（底数为e)、底数为10的log、底数为2的log、log(1+X)

sign 计算各元素的正负号：1(正数）、0(零）、一1(负数）

ceil 计算各元素的“天花板”值，即大于等于所有元素的最小整数

floor 计算各元素的“地板”值，即小于等于所有元素的最大整数

rint 将各元素值四舍五入到最接近的整数，保留dtype

modf 将数组的小数和整数部分以两个独立数组的形式返回

isnan 返回一个表示“哪些值是NaN(这不是一个数字）”的布尔型数组

isfinite、isinf 分别返回一个表示“哪些元素是有穷的（非inf,非NaN)”或“哪些元素是无穷的”的布尔型数组

cos,cosh,sin,sinh,tan,tanh 普通型和双曲型三角函数

## 二元运算

### 1.四则运算

运算|函数

--- | ---

a + b | add(a,b)

-a np.negative 加负号

a - b | subtract(a,b)

a \* b | multiply(a,b)

a / b | divide(a,b)

a \*\* b | power(a,b)

a % b | remainder(a,b) mod(a,b)

a//b 取商 np.floor\_divide(a,b)

np.absolute(x) 或np.abs(s) 绝对值

以乘法为例，数组与标量相乘，相当于数组的每个元素乘以这个标量

类型 说明

add 将数组中对应的元素相加

subtract 从第一个数组中减去第二个数组中的元素

multiply 数组元素相乘

divide、floor\_divide 除法或向下圆整除法（丢弃余数）

power 对第一个数组中的元素A，根据第二个数组中的相应元素B，计算A的B次方

maximum、fmax 对应元素级的最大值计算。fmax将忽略NaN空值

minimum、fmin 对应元素级的最小值计算。fmin将忽略NaN空值

mod 元素级的求模计算（除法的余数）

### 2.逻辑运算

运算|函数<

== | equal

!= | not\_equal

> | greater

>= | greater\_equal

< | less

<= | less\_equal

logical\_and 逻辑与

logical\_or

logical\_xor 异或

logical\_not

& bitwise\_and 按位与 将数值转化为二进制，按位与，对应位相与，对于数组也一样。如果两个数值长度不一样，补0。要注意的是 & 的运算优先于比较运算如 > 等。

bitwise\_or

^ bitwise\_xor 按位异或。

~ invert 按位取反

invert() 函数对数组中整数进行位取反运算，即 0 变成 1，1 变成 0。

对于有符号整数，取该二进制数的补码，然后 +1。二进制数，最高位为0表示正数，最高位为 1 表示负数。

看看 ~1 的计算步骤：

* 将**1**(这里叫：原码)转二进制 ＝ **00000001**
* 按位取反 ＝**11111110**
* 发现符号位(即最高位)为**1**(表示负数)，将除符号位之外的其他数字取反 ＝ **10000001**
* 末位加1取其补码 ＝ **10000010**
* 转换回十进制 ＝ **-2**

>> right\_shift 右移 将数值转为2进制，向右移n位，其实就是除以2n，再转回10进制。如[8,16]>>3 变为[1,2]

a=[8,16]

<< left\_shift 左移

等于操作也是逐元素比较的：

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

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

a == b

返回：array([[ True, True, False, True],

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

这意味着，如果我们在条件中要判断两个数组是否一样时，不能直接使用

if a == b:

而要使用：

if all(a==b):

对于浮点数，由于存在精度问题，使用函数 allclose 会更好：

if allclose(a,b):

logical\_and 也是逐元素的 and 操作

## ufunc 对象函/函数对象

\*\*Numpy\*\* 有两种基本对象： ndarray (N-dimensional array object) 和 ufunc (universal function object) 。 ndarray 是存储单一数据类型的多维数组，而 ufunc 则是能够对数组进行处理的函数。

例如，我们之前所接触到的二元操作符对应的 \*\*Numpy\*\* 函数，如 add ，就是一种 ufunc 对象，它可以作用于数组的每个元素。

### 二元操作符所支持方法

#### reduce 方法

将 op 沿着某个轴应用，使得数组 a 的维数降低一维。

add 作用到一维数组上相当于求和：

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

np.add.reduce(a)

输出：10

多维数组默认只按照第一维进行运算：即axis=0

指定维度：

np.add.reduce(a, 1)

array([ 6, 15])

作用于字符串：

a = np.array(['ab', 'cd', 'ef'], np.object)

np.add.reduce(a)

'abcdef'

逻辑运算：虽然是按位与，但是输出确实总体与。

a = np.array([1,1,0,1])

np.logical\_and.reduce(a)

False

#### accumulate 方法

accumulate 可以看成保存 reduce 每一步的结果所形成的数组。

类似于对于sum就像当于cumsum，保存每一步的计算结果。

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

np.add.accumulate(a)

array([ 1, 3, 6, 10])

a = np.array(['ab', 'cd', 'ef'], np.object)

np.add.accumulate(a)

array(['ab', 'abcd', 'abcdef'], dtype=object)

a = np.array([1,1,0,1])

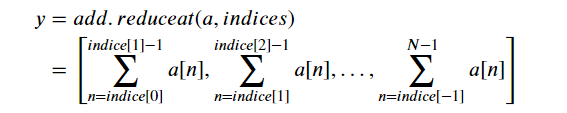
np.logical\_and.accumulate(a)

array([ True, True, False, False], dtype=bool)

#### reduceat 方法

op.reduceat(a, indices)

reduceat 方法将操作符运用到指定的下标上，返回一个与 indices 大小相同的数组：



a = np.array([0, 10, 20, 30, 40, 50])

indices = np.array([1,4])

np.add.reduceat(a, indices)

array([60, 90])

注意：indices=[1,4] 即返回a[1]至a[3]、a[4]至a[-1]的累加和。

这里， indices 为 [1, 4] ，所以 60 表示从下标1（包括）加到下标4（不包括）的结果， 90 表示从下标4（包括）加到结尾的结果。

#### outer 方法

op.outer(a, b)

对于 a 中每个元素，将 op 运用到它和 b 的每一个元素上所得到的结果：

类似于kron的操作。

a = np.array([0,1])

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

np.add.outer(a, b)

array([[1, 2, 3],

[2, 3, 4]])

注意有顺序的区别：

np.add.outer(b, a)

array([[1, 2],

[2, 3],

[3, 4]])

## np.random.模块

from numpy.random import \*

### uniform(low=0.0, high=1.0, size=None)

 生出size个符合均分布的浮点数，取值范围为[low, high)，默认取值范围为[0, 1.0)

[复制代码](javascript:void(0);)

>>> random.uniform()

0.3999807403689315

>>> random.uniform(size=1)

array([0.55950578])

>>> random.uniform(5, 6)

5.293682668235986

>>> random.uniform(5, 6, size=(2,3))

array([[5.82416021, 5.68916836, 5.89708586],

[5.63843125, 5.22963754, 5.4319899 ]])

### rand(d0, d1, ..., dn)

 生成一个(d0, d1, ..., dn)维的数组，数组的元素取自[0, 1)上的均分布，若没有参数输入，则生成一个数

[复制代码](javascript:void(0);)

>>> random.rand()

0.4378166124207712

>>> random.rand(1)

array([0.69845956])

>>> random.rand(3,2)

array([[0.15725424, 0.45786148],

[0.63133098, 0.81789056],

[0.40032941, 0.19108526]])

>>> random.rand(3,2,1)

array([[[0.00404447],

[0.3837963 ]],

[[0.32518355],

[0.82482599]],

[[0.79603205],

[0.19087375]]])

[复制代码](javascript:void(0);)

### randint(low, high=None, size=None, dtype='I')

生成size个整数，取值区间为[low, high)，若没有输入参数high则取值区间为[0, low)

>>> random.randint(8)

5

>>> random.randint(8, size=1)

array([1])

>>> random.randint(8, size=(2,2,3))

array([[[4, 7, 0],

[1, 4, 1]],

[[2, 2, 5],

[7, 6, 4]]])

>>> random.randint(8, size=(2,2,3), dtype='int64')

array([[[5, 5, 6],

[2, 7, 2]],

[[2, 7, 6],

[4, 7, 7]]], dtype=int64)

[复制代码](javascript:void(0);)

### random\_integers(low, high=None, size=None)

 生成size个整数，取值区间为[low, high], 若没有输入参数high则取值区间为[1, low]，注意这里左右都是闭区间

[复制代码](javascript:void(0);)

>>> random.random\_integers(5)

1

>>> random.random\_integers(5, size=1)

array([2])

>>> random.random\_integers(4, 5, size=(2,2))

array([[5, 4],

[4, 4]])

[复制代码](javascript:void(0);)

### random(size=None)

 产生[0.0, 1.0)之间的浮点数

>>> random.random(5)

array([0.94128141, 0.98725499, 0.48435957, 0.90948135, 0.40570882])

>>> random.random()

0.49761416226728084

### random相同用法：

* numpy.random.random\_sample
* numpy.random.ranf
* numpy.random.sample (抽取不重复)

### bytes(length)

 生成随机字节

>>> random.bytes(1)

b'%'

>>> random.bytes(2)

b'\xd0\xc3'

### choice(a, size=None, replace=True, p=None)

 从a（数组）中选取size（维度）大小的随机数，replace=True表示可重复抽取，p是a中每个数出现的概率

 若a是整数，则a代表的数组是arange(a)

[复制代码](javascript:void(0);)

>>> random.choice(5)

3

>>> random.choice([0.2, 0.4])

0.2

>>> random.choice([0.2, 0.4], p=[1, 0])

0.2

>>> random.choice([0.2, 0.4], p=[0, 1])

0.4

>>> random.choice(5, 5)

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

>>> random.choice(5, 5, False)

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

>>> random.choice(100, (2, 3, 5), False)

array([[[43, 81, 48, 2, 8],

[33, 79, 30, 24, 83],

[ 3, 82, 97, 49, 98]],

[[32, 12, 15, 0, 96],

[19, 61, 6, 42, 60],

[ 7, 93, 20, 18, 58]]])

[复制代码](javascript:void(0);)

### permutation(x)

 随机打乱x中的元素。若x是整数，则打乱arange(x)，若x是一个数组，则将copy(x)的第一位索引打乱，只打乱第一位索引，并不是所有都打乱。，意思是先复制x，对副本进行打乱处理，打乱只针对数组的第一维

[复制代码](javascript:void(0);)

>>> random.permutation(5)

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

>>> random.permutation(5)

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

>>> random.permutation([[1,2,3],[4,5,6]])

array([[1, 2, 3],

[4, 5, 6]])

>>> random.permutation([[1,2,3],[4,5,6]])

array([[4, 5, 6],

[1, 2, 3]])

[复制代码](javascript:void(0);)

### shuffle(x)

 与permutation类似，随机打乱x中的元素。若x是整数，则打乱arange(x). 但是shuffle会对x进行修改

[复制代码](javascript:void(0);)

>>> a = arange(5)

>>> a

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

>>> random.permutation(a)

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

>>> a

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

>>> random.shuffle(a)

>>> a

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

[复制代码](javascript:void(0);)

### numpy.random.seed(seed=None)

 设置随机生成算法的初始值

相同的seed，生成的随机数是一样的。在生成随机数random.random前使用。

### 其它符合函数分布的随机数函数

含链接。

* [numpy.random.beta](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.beta.html?highlight=random)
* [numpy.random.binomial](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.binomial.html?highlight=random)
* [numpy.random.chisquare](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.chisquare.html?highlight=random)
* [numpy.random.dirichlet](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.dirichlet.html?highlight=random)
* [numpy.random.exponential](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.exponential.html?highlight=random)
* [numpy.random.f](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.f.html?highlight=random)
* [numpy.random.gamma](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.gamma.html?highlight=random)
* [numpy.random.geometric](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.geometric.html?highlight=random)
* [numpy.random.gumbel](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.gumbel.html?highlight=random)
* [numpy.random.hypergeometric](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.hypergeometric.html?highlight=random)
* [numpy.random.laplace](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.laplace.html?highlight=random)
* [numpy.random.logistic](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.logistic.html?highlight=random)
* [numpy.random.lognormal](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.lognormal.html?highlight=random)
* [numpy.random.logseries](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.logseries.html?highlight=random)
* [numpy.random.multinomial](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.multinomial.html?highlight=random)
* [numpy.random.multivariate\_normal](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.multivariate_normal.html?highlight=random)
* [numpy.random.negative\_binomial](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.negative_binomial.html?highlight=random)
* [numpy.random.noncentral\_chisquare](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.noncentral_chisquare.html?highlight=random)
* [numpy.random.noncentral\_f](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.noncentral_f.html?highlight=random)
* [numpy.random.normal](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.normal.html?highlight=random) 正态分布
* [numpy.random.pareto](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.pareto.html?highlight=random)
* [numpy.random.poisson](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.poisson.html?highlight=random)
* [numpy.random.power](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.power.html?highlight=random)
* [numpy.random.randn](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html?highlight=random) 标准正态分布
* [numpy.random.rayleigh](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.rayleigh.html?highlight=random)
* [numpy.random.standard\_cauchy](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_cauchy.html?highlight=random)
* [numpy.random.standard\_exponential](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_exponential.html?highlight=random)
* [numpy.random.standard\_gamma](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_gamma.html?highlight=random)
* [numpy.random.standard\_normal](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_normal.html?highlight=random)
* [numpy.random.standard\_t](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_t.html?highlight=random)
* [numpy.random.triangular](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.triangular.html?highlight=random)
* [numpy.random.vonmises](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.vonmises.html?highlight=random)
* [numpy.random.wald](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.wald.html?highlight=random)
* [numpy.random.weibull](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.weibull.html?highlight=random)
* [numpy.random.zipf](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.zipf.html?highlight=random)

## 内存映射

\*\*Numpy\*\* 有对内存映射的支持。

内存映射也是一种处理文件的方法，主要的函数有：

- memmap

- frombuffer

- ndarray constructor

内存映射文件与虚拟内存有些类似，通过内存映射文件可以保留一个地址空间的区域，同时将物理存储器提交给此区域，内存文件映射的物理存储器来自一个已经存在于磁盘上的文件，而且在对该文件进行操作之前必须首先对文件进行映射。

使用内存映射文件处理存储于磁盘上的文件时，将不必再对文件执行I/O操作，使得内存映射文件在处理大数据量的文件时能起到相当重要的作用。

### memmap

memmap(filename,

dtype=uint8,

mode='r+'

offset=0

shape=None

order=0)

mode 表示文件被打开的类型：

- r 只读

- c 复制+写，但是不改变源文件

- r+ 读写，使用 flush 方法会将更改的内容写入文件

- w+ 写，如果存在则将数据覆盖

offset 表示从第几个位置开始。

## 从MATLAB到numpy

\*\* Numpy \*\* 和 \*\* Matlab \*\* 有很多相似的地方，但 \*\* Numpy \*\* 并非 \*\* Matlab \*\* 的克隆，它们之间存在很多差异，例如：



## 数组和矩阵

Numpy 中不仅提供了 array 这个基本类型，还提供了支持矩阵操作的类 matrix ，但是一般推荐使用 array ：

### 区别

- 很多 numpy 函数返回的是 array ，不是 matrix

- 在 array 中，逐元素操作和矩阵操作有着明显的不同

- 向量可以不被视为矩阵

具体说来：

#### \*， dot(), multiply()

- array ： \* -逐元素乘法， dot() -矩阵乘法

- matrix ： \* -矩阵乘法， multiply() -逐元素乘法

#### - 处理向量

- array ：形状为 1xN, Nx1, N 的向量的意义是不同的，类似于 A[:,1] 的操作返回的是一维数组，形状为 N ，一维数组的转置仍是自己本身

- matrix ：形状为 1xN, Nx1 ， A[:,1] 返回的是二维 Nx1 矩阵

#### - 高维数组

- array ：支持大于2的维度

- matrix ：维度只能为2

#### - 属性

- array ： .T 表示转置

- matrix ： .H 表示复共轭转置， .I 表示逆， .A 表示转化为 array 类型

#### - 构造函数

- array ： array 函数接受一个（嵌套）序列作为参数—— array([[1,2,3],[4,5,6]])

- matrix ： matrix 函数额外支持字符串参数—— matrix("[1 2 3; 4 5 6]")

其优缺点各自如下：

### - array

- [GOOD] 一维数组既可以看成列向量，也可以看成行向量。 v 在 dot(A,v) 被看成列向量，在 dot(v,A) 中被看成行向量，这样省去了转置的麻烦

- [BAD!] 矩阵乘法需要使用 dot() 函数，如： dot(dot(A,B),C) vs A\*B\*C

- [GOOD] 逐元素乘法很简单： A\*B

- [GOOD] 作为基本类型，是很多基于 numpy 的第三方库函数的返回类型

- [GOOD] 所有的操作 \*,/,+,\*\*,... 都是逐元素的

- [GOOD] 可以处理任意维度的数据

- [GOOD] 张量运算

### - \*\* matrix \*\*

- [GOOD] 类似与 \*\* MATLAB \*\* 的操作

- [BAD!] 最高维度为2

- [BAD!] 最低维度也为2

- [BAD!] 很多函数返回的是 array ，即使传入的参数是 matrix

- [GOOD] A\*B 是矩阵乘法

- [BAD!] 逐元素乘法需要调用 multiply 函数

- [BAD!] / 是逐元素操作

当然在实际使用中，二者的使用取决于具体情况。

### 互相转化：

- asarray ：返回数组

- asmatrix （或者 mat ） ：返回矩阵

- asanyarray ：返回数组或者数组的子类，注意到矩阵是数组的一个子类，所以输入是矩阵的时候返回的也是矩阵

## MATLAB与numpy常用操作参照

注意：\*\*`MATLAB`\*\* 与 \*\*`Numpy`\*\* 下标之间有这样几处不同：

MATLAB第一维为1，而np为0

MATLAB用(),np用[]

- `MATLAB`：`beg(:step):end`，包含结束值 `end`

- `Numpy`：`beg:end(:step)`，不包含结束值 `end`：如果想要包含结束值，在结束值后跟j。

|  |  |  |
| --- | --- | --- |
| MATLAB | Numpy | 注释 |
| help func | info(func) ， help(func) ， func? (IPython) | 查看函数帮助 |
| which func |  | 查看函数在什么地方定义 |
| type func | source(func) ， func?？ (IPython) | 查看函数源代码 |
| a && b | a and b | 逻辑 AND |
| 1\*i, 1\*j, 1i, 1j | 1j | 复数 |
| eps | spacing(1) | 1 与最近浮点数的距离 |
| ndims(a) | ndim(a), a.ndim | a 的维数 |
| numel(a) | size(a), a.size | a 的元素个数 |
| size(a) | shape(a), a.shape | a 的形状 |
| size(a,n) | a.shape[n-1] | 第 n 维的大小 |
| a(2,5) | a[1,4] | 第 2 行第 5 列元素 |
| a(2,:) | a[1], a[1,:] | 第 2 行 |
| a(1:5,:) | a[0:5] | 第 1 至 5 行 |
|  |  |  |
| a(end-4:end,:) | a[-5:] | 后 5 行 |
| a(1:3,5:9) | a[0:3][:,4:9] | 特定行列（1~3 行，5~9 列） |
| a([2,4,5],[1,3]) | a[ix\_([1,3,4],[0,2])] | 特定行列（2,4,5 行的 1,3 列） |
| a(3:2:21,:) | a[2:21:2,:] | 特定行列（3,5,...,21 行） |
| a(1:2:end,:) | a[ ::2,:] | 奇数行 |
| a([1:end 1],:) | a[r\_[:len(a),0]] | 将第一行添加到末尾 |
| a.' | a.T | 转置 |
| a ./ b | a/b | 逐元素除法 |
| (a>0.5) | (a>0.5) | 各个元素是否大于 0.5 |
| find(a>0.5) | nonzero(a>0.5) | 大于 0.5 的位置 |
| a(a<0.5)=0 | a[a<0.5]=0 | 小于 0.5 的设为 0 |
| a(:) = 3 | a[:] = 3 | 所有元素设为 3 |
| y=x | y=x.copy() | 将 y 设为 x |
| y=x(2,:) | y=x[1,:].copy() | 注意值传递和引用传递的区别 |
| y=x(:) | y=x.flatten(1) | 将矩阵变为一个向量，这里 1 表示沿着列进行转化 |
| max(max(a)) | a.max() | 最大值 |
| max(a) | a.max(0) | 每一列的最大值 |
| max(a,[],2) | a.max(1) | 每一行的最大值 |
| max(a,b) | maximum(a,b) | 逐元素比较，取较大的值 |
| a & b | logical\_and(a, b) | 逻辑 AND |
| bitand(a, b) | a & b | 逐比特 AND |
| inv(a) | linalg.inv(a) | a 的逆 |
| pinv(a) | linalg.inv(a) | 伪逆 |
| rank(a) | linalg.matrix\_rank(a) | 秩 |
| a\b | linalg.solve(a,b)(如果a是方阵),linalg.lstsq(a,b) | 解 a x = b |
| b/a | 求解 a.T x.T = b.T | 解 x a = b |
| [U,S,V]=svd(a) | U, S, Vh = linalg.svd(a), V = Vh.T | 奇异值分解 |
| chol(a) | linalg.cholesky(a).T | Cholesky 分解 |
| [V,D]=eig(a) | D,V = linalg.eig(a) | 特征值分解 |
| [V,D]=eig(a,b) | V,D = scipy.linalg.eig(a,b) |  |
| [V,D]=eigs(a,k) |  | 前 k 大特征值对应的特征向量 |

|  |  |  |  |
| --- | --- | --- | --- |
| MATLAB | numpy.array | numpy.matrix | 注释 |
| [1,2,3;4,5,6] | array([[1.,2.,3.],[4.,5.,6.]]) | mat([[1.,2.,3.],[4.,5.,6.]]), mat('1,2,3;4,5,6') | 2x3 矩阵、数组 |
| [a b;c d] | vstack([hstack([a,b]), hsatck([c,d])]]) | bmat('a b;c d') | 分块矩阵构造 |
| a(end) | a[-1] | a[:,-1][0,0] | 最后一个元素 |
| a' | a.conj().T | a.H | 复共轭转置 |
| a \* b | dot(a,b) | a \* b | 矩阵乘法 |
| a .\* b | a \* b | multiply(a,b) | 逐元素乘法 |
| a.^3 | a\*\*3 | power(a,3) | 逐元素立方 |
| a(:,find(v>0.5)) | a[:,nonzero(v>0.5)[0]] | a[:,nonzero(v.A>0.5)[0]] | 找出行向量 v>0.5 对应的 a 中的列 |
| a(:,find(v>0.5)) | a[:,v.T>0.5] | a[:,v.T>0.5)] | 找出列向量 v>0.5 对应的 a 中的列 |
| a .\* (a>0.5) | a \* (a>0.5) | mat(a.A \* (a>0.5).A) | 将所有小于 0.5 的元素设为 0 |
| 1:10 | arange(1.,11.), r\_[1.:11.], r\_[1:10:10j] | mat(arange(1.,11.)), r\_[1.:11., 'r'] | 这里 1. 是为了将其转化为浮点数组 |
| 0:9 | arange(10.),r\_[:10.], r\_[:9:10j] | mat(arange(10.)), r\_[:10., 'r'] |  |
| [1:10]' | arange(1.,11.)[:,newaxis] | r\_[1.:11.,'c'] | 列向量 |
| zeros, ones, eye, diag, linspace | zeros, ones, eye, diag, linspace | mat(...) |  |
| rand(3,4) | random. rand(3,4) | mat(...) | 0~1 随机数 |
| [x,y]=meshgrid(0:8,0:5) | mgrid[0:9.,0:6.], meshgrid(r\_[0:9.],r\_[0:6.]) | mat(...) | 网格 |
|  | ogrid[0:9.,0:6.], ix\_(r\_[0:9.],r\_[0:6.]) | mat() | 建议在 Numpy 中使用 |
| [x,y]=meshgrid([1,2,4],[2,4,5]) | meshgrid([1,2,4],[2,4,5]) | mat(...) |  |
|  | ix\_([1,2,4],[2,4,5]) | mat(...) |  |
| repmat(a, m, n) | tile(a, (m,n)) | mat(...) | 产生 m x n 个 a |
| [a b] | c\_[a,b] | concatenate((a,b),1) | 列对齐连接 |
| [a; b] | r\_[a,b] | concatenate((a,b)) | 行对齐连接 |
| norm(v) | sqrt(dot(v,v)), linalg.norm(v) | sqrt(dot(v.A,v.A)), linalg.norm(v) | 模 |
| [Q,R,P]=qr(a,0) | Q,R = scipy.linalg.qr(a) | mat(...) | QR 分解 |
| [L,U,P]=lu(a) | L,U = Sci.linalg.lu(a) | mat(...) | LU 分解 |
| fft(a) | fft(a) | mat(...) | FFT |
| ifft(a) | ifft(a) | mat(...) | IFFT |
| sort(a) | sort(a),a.sort | mat(...) | 排序 |

参考：<http://wiki.scipy.org/NumPy_for_Matlab_Users#whichNotes>