[NumPy简明教程（一、简介）](http://blog.csdn.net/sunny2038/article/details/8873241)

## 1、NumPy初探

　　用Python在ArcGIS的开发中，对NumPy的接触越来越多，从现在开始就将以前的NumPy笔记整理一下，慢慢放出来。供有需要的朋友参考。

**为什么需要numpy**

　　Python中提供了list容器，可以当作数组使用。但列表中的元素可以是任何对象，因此列表中保存的是对象的指针，这样一来，为了保存一个简单的列表[1,2,3]。就需要三个指针和三个整数对象。对于数值运算来说，这种结构显然不够高效。

　　Python虽然也提供了array模块，但其只支持一维数组，不支持多维数组，也没有各种运算函数。因而不适合数值运算。

　　而NumPy的出现弥补了这些不足。（——摘自张若愚的《Python科学计算》）

**Numpy基础**

　　Numpy基础部分中，有两个主要内容，如下：

* 任意维数的数组对象（ndarray，n-dimensional array object）
* 通用函数对象（ufunc，universal function object）

　　本节首先来介绍数组对象基本概念，下一节介绍其属性以及相应函数的使用。后续文章将介绍通用函数对象。

**数组**

　　Numpy中，任意维数的数组对象（ndarray， n-dimensional array object）是最基本的内容。ndarray由两部分组成：

* 实际所持有的数据；
* 描述这些数据的元数据（metadata）

后续部分先介绍NumPy数组，接着介绍通用函数对象，最后介绍一些高级属性。

未完待续。。。

[NumPy简明教程（二、数组1）](http://blog.csdn.net/sunny2038/article/details/9002531)

**NumPy数组**

NumPy数组是一个多维数组对象，称为ndarray。其由两部分组成：

* 实际的数据
* 描述这些数据的元数据

大部分操作仅针对于元数据，而不改变底层实际的数据。

关于NumPy数组有几点必需了解的：

* NumPy数组的下标从0开始。
* 同一个NumPy数组中所有元素的类型必须是相同的。

**NumPy数组属性**

在详细介绍NumPy数组之前。先详细介绍下NumPy数组的基本属性。NumPy数组的维数称为秩（rank），一维数组的秩为1，二维数组的秩为2，以此类推。在NumPy中，每一个线性的数组称为是一个轴（axes），秩其实是描述轴的数量。比如说，二维数组相当于是两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是NumPy中的轴（axes），第一个轴相当于是底层数组，第二个轴是底层数组里的数组。而轴的数量——秩，就是数组的维数。

NumPy的数组中比较重要ndarray对象属性有：

* 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。
* 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。
* 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。

**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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> **from** numpy **import** \*
3. >>> a = array( [2,3,4] )
4. >>> a
5. array([2, 3, 4])
6. >>> a.dtype
7. dtype('int32')
8. >>> b = array([1.2, 3.5, 5.1])
9. >>> b.dtype
10. dtype('float64')

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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> a = array(1,2,3,4)    # 错误
2. >>> 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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

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

可以在创建时显式指定数组中元素的类型

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> c = array( [ [1,2], [3,4] ], dtype=complex)
2. >>> c
3. array([[ 1.+0.j,  2.+0.j],
4. [ 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。

可以哟娜特d.dtype.itemsize来查看数组中元素占用的字节数目。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> d = zeros((3,4))
2. >>> d.dtype
3. dtype('float64')
4. >>> d
5. array([[ 0.,  0.,  0.,  0.],
6. [ 0.,  0.,  0.,  0.],
7. [ 0.,  0.,  0.,  0.]])
8. >>> d.dtype.itemsize
9. 8

也可以自己制定数组中元素的类型

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> ones( (2,3,4), dtype=int16 )  #手动指定数组中元素类型
2. array([[[1, 1, 1, 1],
3. [1, 1, 1, 1],
4. [1, 1, 1, 1]],
6. [[1, 1, 1, 1],
7. [1, 1, 1, 1],
8. [1, 1, 1, 1]]], dtype=int16)
9. >>> empty((2,3))
10. array([[  2.65565858e-316,   0.00000000e+000,   0.00000000e+000],
11. [  0.00000000e+000,   0.00000000e+000,   0.00000000e+000]])

NumPy提供一个类似arange的函数返回一个数列形式的数组:

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> arange(10, 30, 5)
2. 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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> arange(0,2,0.5)
2. array([ 0. ,  0.5,  1. ,  1.5])

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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> numpy.linspace(-1, 0, 5)
2. array([-1.  , -0.75, -0.5 , -0.25,  0.  ])

数组中的元素是通过下标来访问的，可以通过方括号括起一个下标来访问数组中单一一个元素，也可以以切片的形式访问数组中多个元素。关于切片访问，将在切片一节介绍。

**知识点：NumPy中的数据类型**  
对于科学计算来说，Python中自带的整型、浮点型和复数类型远远不够，因此NumPy中添加了许多数据类型。如下：

|  |  |
| --- | --- |
| NumPy中的基本数据类型 | |
| 名称 | 描述 |
| bool | 用一个Bit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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> float64(42)
2. 42.0
3. >>> int8(42.0)
4. 42
5. >>> bool(42)
6. True
7. >>> bool(42.0)
8. True
9. >>> float(True)
10. 1.0

许多函数的参数中可以指定参数的类型，当然，这个类型参数是可选的。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> arange(7, dtype=uint16)
2. array([0, 1, 2, 3, 4, 5, 6], dtype=uint16)

**输出数组**

当输出一个数组时，NumPy以特定的布局用类似嵌套列表的形式显示：

* 第一行从左到右输出
* 每行依次自上而下输出
* 每个切片通过一个空行与下一个隔开
* 一维数组被打印成行，二维数组成矩阵，三维数组成矩阵列表。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> a = arange(6)                         # 1d array
2. >>> **print** a
3. [0 1 2 3 4 5]
5. >>> b = arange(12).reshape(4,3)           # 2d array
6. >>> **print** b
7. [[ 0  1  2]
8. [ 3  4  5]
9. [ 6  7  8]
10. [ 9 10 11]]
11. >>> c = arange(24).reshape(2,3,4)         # 3d array
12. >>> **print** c
13. [[[ 0  1  2  3]
14. [ 4  5  6  7]
15. [ 8  9 10 11]]
17. [[12 13 14 15]
18. [16 17 18 19]
19. [20 21 22 23]]]

reshape将在下一篇文章中介绍　  
 如果一个数组太长，则NumPy自动省略中间部分而只打印两端的数据：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. >>> **print** arange(10000)
2. [   0    1    2 ..., 9997 9998 9999]
4. >>> **print** arange(10000).reshape(100,100)
5. [[   0    1    2 ...,   97   98   99]
6. [ 100  101  102 ...,  197  198  199]
7. [ 200  201  202 ...,  297  298  299]
8. ...,
9. [9700 9701 9702 ..., 9797 9798 9799]
10. [9800 9801 9802 ..., 9897 9898 9899]
11. [9900 9901 9902 ..., 9997 9998 9999]]

　　可通过设置printoptions参数来禁用NumPy的这种行为并强制打印整个数组。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9002531)

1. set\_printoptions(threshold='nan')

这样，输出时数组的所有元素都会显示出来。

未完待续，如有错误，敬请指正！

参考文献：

《NumPy for Beginner》  
《Python科学计算》  
《Tentative NumPy Tutorial》

[NumPy简明教程（二、数组2）](http://blog.csdn.net/sunny2038/article/details/9023797)

**基本运算**

数组的算术运算是按元素逐个运算。数组运算后将创建包含运算结果的新数组。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.array([20,30,40,50])
2. >>> b= np.arange( 4)
3. >>> b
4. array([0, 1, 2, 3])
5. >>> c= a-b
6. >>> c
7. array([20, 29, 38, 47])
8. >>> b\*\*2
9. array([0, 1, 4, 9])
10. >>> 10\*np.sin(a)
11. array([ 9.12945251,-9.88031624, 7.4511316, -2.62374854])
12. >>> a<35
13. array([True, True, False, False], dtype=bool)

与其他矩阵语言不同，NumPy中的乘法运算符\***按元素逐个**计算，矩阵乘法可以使用dot函数或创建矩阵对象实现(后续章节会介绍)

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> A= np.array([[1,1],
2. ...[0,1]])
3. >>> B= np.array([[2,0],
4. ...[3,4]])
5. >>> A\*B # 逐个元素相乘
6. array([[2, 0],
7. [0, 4]])
8. >>> np.dot(A,B) # 矩阵相乘
9. array([[5, 4],
10. [3, 4]])

　有些操作符如+=和\*=用来更改已存在数组而不创建一个新的数组。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.ones((2,3), dtype=int)
2. >>> b= np.random.random((2,3))
3. >>> a\*= 3
4. >>> a
5. array([[3, 3, 3],
6. [3, 3, 3]])
7. >>> b+= a
8. >>> b
9. array([[ 3.69092703, 3.8324276, 3.0114541],
10. [ 3.18679111, 3.3039349, 3.37600289]])
11. >>> a+= b # b转换为整数类型
12. >>> a
13. array([[6, 6, 6],
14. [6, 6, 6]])

当数组中存储的是不同类型的元素时，数组将使用占用更多位（bit）的数据类型作为其本身的数据类型，也就是偏向更精确的数据类型(这种行为叫做upcast)。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.ones(3, dtype=np.int32)
2. >>> b= np.linspace(0,np.pi,3)
3. >>> b.dtype.name
4. 'float64'
5. >>> c= a+b
6. >>> c
7. array([ 1., 2.57079633, 4.14159265])
8. >>> c.dtype.name
9. 'float64'
10. >>> d= exp(c\*1j)
11. >>> d
12. array([ 0.54030231+0.84147098j,-0.84147098+0.54030231j,
13. -0.54030231-0.84147098j])
14. >>> d.dtype.name
15. 'complex128'

　　许多非数组运算，如计算数组所有元素之和，都作为ndarray类的方法来实现，使用时需要用ndarray类的实例来调用这些方法。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.random.random((2,3))
2. >>> a
3. array([[ 0.65806048, 0.58216761, 0.59986935],
4. [ 0.6004008, 0.41965453, 0.71487337]])
5. >>> a.sum()
6. 3.5750261436902333
7. >>> a.min()
8. 0.41965453489104032
9. >>> a.max()
10. 0.71487337095581649

这些运算将数组看作是一维线性列表。但可通过指定axis参数（即数组的行）对指定的轴做相应的运算：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> b= np.arange(12).reshape(3,4)
2. >>> b
3. array([[ 0, 1, 2, 3],
4. [ 4, 5, 6, 7],
5. [ 8, 9, 10, 11]])
6. >>> b.sum(axis=0) # 计算每一列的和，注意理解轴的含义，参考数组的第一篇文章
7. array([12, 15, 18, 21])
8. >>> b.min(axis=1) # 获取每一行的最小值
9. array([0, 4, 8])
10. >>> b.cumsum(axis=1) # 计算每一行的累积和
11. array([[ 0, 1, 3, 6],
12. [ 4, 9, 15, 22],
13. [ 8, 17, 27, 38]])

**索引，切片和迭代**

　　　和列表和其它Python序列一样，一维数组可以进行索引、切片和迭代操作。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.arange(10)\*\*3 #记住，操作符是对数组中逐元素处理的！
2. >>> a
3. array([0, 1, 8, 27, 64, 125, 216, 343, 512, 729])
4. >>> a[2]
5. 8
6. >>> a[2:5]
7. array([ 8, 27, 64])
8. >>> a[:6:2]= -1000 # 等同于a[0:6:2]= -1000，从开始到第6个位置，每隔一个元素将其赋值为-1000
9. >>> a
10. array([-1000, 1,-1000, 27,-1000, 125, 216, 343, 512, 729])
11. >>> a[: :-1] # 反转a
12. array([ 729, 512, 343, 216, 125,-1000, 27,-1000, 1,-1000])
13. >>>**for** i **in** a:
14. ...    **print** i\*\*(1/3.),
15. ...
16. nan 1.0 nan 3.0 nan 5.0 6.0 7.0 8.0 9.0

　　　多维数组可以每个轴有一个索引。这些索引由一个逗号分割的元组给出。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>>**def** f(x,y):
2. ...    **return** 10\*x+y
3. ...
4. >>> b= np.fromfunction(f,(5,4),dtype=int) #fromfunction是一个函数，下篇文章介绍。
5. >>> b
6. array([[ 0, 1, 2, 3],
7. [10, 11, 12, 13],
8. [20, 21, 22, 23],
9. [30, 31, 32, 33],
10. [40, 41, 42, 43]])
11. >>> b[2,3]
12. 23
13. >>> b[0:5, 1] # 每行的第二个元素
14. array([ 1, 11, 21, 31, 41])
15. >>> b[: ,1] # 与前面的效果相同
16. array([ 1, 11, 21, 31, 41])
17. >>> b[1:3,: ] # 每列的第二和第三个元素
18. array([[10, 11, 12, 13],
19. [20, 21, 22, 23]])

　　　当少于提供的索引数目少于轴数时，已给出的数值按秩的顺序复制，确失的索引则默认为是整个切片：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> b[-1] # 最后一行，等同于b[-1,:]，-1是第一个轴，而缺失的认为是：，相当于整个切片。
2. array([40, 41, 42, 43])

    b[i]中括号中的表达式被当作i和一系列:，来代表剩下的轴。NumPy也允许你使用“点”像b[i,...]。

    点(…)代表许多产生一个完整的索引元组必要的分号。如果x是秩为5的数组(即它有5个轴)，那么:

* x[1,2,…] 等同于 x[1,2,:,:,:],
* x[…,3] 等同于 x[:,:,:,:,3]
* x[4,…,5,:] 等同 x[4,:,:,5,:]

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> c= array( [ [[ 0, 1, 2], #三维数组（两个2维数组叠加而成）
2. ...[ 10, 12, 13]],
3. ...
4. ...[[100,101,102],
5. ...[110,112,113]]] )
6. >>> c.shape
7. (2, 2, 3)
8. >>> c[1,...] #等同于c[1,:,:]或c[1]
9. array([[100, 101, 102],
10. [110, 112, 113]])
11. >>> c[...,2] #等同于c[:,:,2]
12. array([[ 2, 13],
13. [102, 113]])

多维数组的遍历是以是第一个轴为基础的：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>>**for** row **in** b:
2. ...    **print** row
3. ...
4. [0 1 2 3]
5. [10 11 12 13]
6. [20 21 22 23]
7. [30 31 32 33]
8. [40 41 42 43]

如果想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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>>**for** element **in** b.flat:
2. ...    **print** element,
3. ...
4. 0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43

更多关于[]、…、newaxis、ndenumerate、indices、index exp的内容请参考NumPy示例

**形状（shape）操作**

**更改数组的形状**

数组的形状取决于其每个轴上的元素个数：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a= np.floor(10\*np.random.random((3,4)))
2. >>> a
3. array([[ 7., 5., 9., 3.],
4. [ 7., 2., 7., 8.],
5. [ 6., 8., 3., 2.]])
6. >>> a.shape
7. (3, 4)

可以用多种方式修改数组的形状：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a.ravel() # 平坦化数组
2. array([ 7., 5., 9., 3., 7., 2., 7., 8., 6., 8., 3., 2.])
3. >>> a.shape= (6, 2)
4. >>> a.transpose()
5. array([[ 7., 9., 7., 7., 6., 3.],
6. [ 5., 3., 2., 8., 8., 2.]])

由ravel()展平的数组元素的顺序通常是“C风格”的，就是以行为基准，最右边的索引变化得最快，所以元素a[0,0]之后是a[0,1]。如果数组改变成其它形状(reshape)，数组仍然是“C风格”的。NumPy通常创建一个以这个顺序保存数据的数组，所以ravel()通常不需要创建起调用数组的副本。但如果数组是通过切片其它数组或有不同寻常的选项时，就可能需要创建其副本。还可以同过一些可选参数函数让reshape()和ravel()构建FORTRAN风格的数组，即最左边的索引变化最快。  
  
reshape函数改变调用数组的形状并返回该数组，而resize函数改变调用数组自身。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/9023797)

1. >>> a
2. array([[ 7., 5.],
3. [ 9., 3.],
4. [ 7., 2.],
5. [ 7., 8.],
6. [ 6., 8.],
7. [ 3., 2.]])
8. >>> a.resize((2,6))
9. >>> a
10. array([[ 7., 5., 9., 3., 7., 2.],
11. [ 7., 8., 6., 8., 3., 2.]])

如果在reshape操作中指定一个维度为-1，那么其准确维度将根据实际情况计算得到

更多关于shape、reshape、resize和ravel的内容请参考NumPy示例

参考文献：

《NumPy for Beginner》  
《Tentative NumPy Tutorial》

[NumPy简明教程（二、数组3）](http://blog.csdn.net/sunny2038/article/details/8907736)

前两篇文章对NumPy数组做了基本的介绍，本篇文章对NumPy数组进行较深入的探讨。首先介绍自定义类型的数组，接着数组的组合，最后介绍数组复制方面的问题。

**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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. 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。

字段中使用NumPy的字符编码来表示数据类型。更详细的数据类型见下表。

|  |  |
| --- | --- |
| 数据类型 | 字符编码 |
| 整数 | i |
| 无符号整数 | u |
| 单精度浮点数 | f |
| 双精度浮点数 | d |
| 布尔值 | b |
| 复数 | D |
| 字符串 | S |
| Unicode | U |
| Void | V |

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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. 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。

注：例子来源于张若愚的Python科学计算艺术的29页。更多关于dtype的内容请参考《NumPy for Beginner》一书的第二章。

**组合函数**

这里介绍以不同的方式组合函数。首先创建两个数组：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> a = arange(9).reshape(3,3)
2. >>> a
3. array([[0, 1, 2],
4. [3, 4, 5],
5. [6, 7, 8]])
6. >>> b = 2 \* a
7. >>> b
8. array([[ 0, 2, 4],
9. [ 6, 8, 10],
10. [12, 14, 16]])

**水平组合**

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> hstack((a, b))
2. array([[ 0, 1, 2, 0, 2, 4],
3. [ 3, 4, 5, 6, 8, 10],
4. [ 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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> concatenate((a, b), axis=1)
2. array([[ 0, 1, 2, 0, 2, 4],
3. [ 3, 4, 5, 6, 8, 10],
4. [ 6, 7, 8, 12, 14, 16]])

**垂直组合**

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> vstack((a, b))
2. array([[ 0, 1, 2],
3. [ 3, 4, 5],
4. [ 6, 7, 8],
5. [ 0, 2, 4],
6. [ 6, 8, 10],
7. [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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> concatenate((a, b), axis=0)
2. array([[ 0, 1, 2],
3. [ 3, 4, 5],
4. [ 6, 7, 8],
5. [ 0, 2, 4],
6. [ 6, 8, 10],
7. [12, 14, 16]])

**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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> dstack((a, b))
2. array([[[ 0, 0],
3. [ 1, 2],
4. [ 2, 4]],
6. [[ 3, 6],
7. [ 4, 8],
8. [ 5, 10]],
10. [[ 6, 12],
11. [ 7, 14],
12. [ 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。

**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：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> one = arange(2)
2. >>> one
3. array([0, 1])
4. >>> two = one + 2
5. >>> two
6. array([2, 3])
7. >>> row\_stack((one, two))
8. array([[0, 1],
9. [2, 3]])

对于2维数组，其作用就像垂直组合一样。

**列组合**

列组合的效果应该很清楚了。如下：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> column\_stack((oned, twiceoned))
2. array([[0, 2],
3. [1, 3]])

对于2维数组，其作用就像水平组合一样。

**分割数组**

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。

**水平分割**

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> a = arange(9).reshape(3,3)
2. >>> a
3. array([[0, 1, 2],
4. [3, 4, 5],
5. [6, 7, 8]])
6. >>> hsplit(a, 3)
7. [array([[0],
8. [3],
9. [6]]),
10. array([[1],
11. [4],
12. [7]]),
13. array([[2],
14. [5],
15. [8]])]

也调用split函数并指定轴为1来获得这样的效果：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. split(a, 3, axis=1)

**垂直分割**

垂直分割是沿着垂直的轴切分数组：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

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

同样，也可通过solit函数并指定轴为1来获得这样的效果：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> split(a, 3, axis=0)

**面向深度的分割**

dsplit函数使用的是面向深度的分割方式：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> c = arange(27).reshape(3, 3, 3)
2. >>> c
3. array([[[ 0,  1,  2],
4. [ 3,  4,  5],
5. [ 6,  7,  8]],
7. [[ 9, 10, 11],
8. [12, 13, 14],
9. [15, 16, 17]],
11. [[18, 19, 20],
12. [21, 22, 23],
13. [24, 25, 26]]])
14. >>> dsplit(c, 3)
15. [array([[[ 0],
16. [ 3],
17. [ 6]],
19. [[ 9],
20. [12],
21. [15]],
23. [[18],
24. [21],
25. [24]]]),
26. array([[[ 1],
27. [ 4],
28. [ 7]],
30. [[10],
31. [13],
32. [16]],
34. [[19],
35. [22],
36. [25]]]),
37. array([[[ 2],
38. [ 5],
39. [ 8]],
41. [[11],
42. [14],
43. [17]],
45. [[20],
46. [23],
47. [26]]])]

**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:

**完全不复制**

　　　简单的赋值，而不复制数组对象或它们的数据。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> a = arange(12)
2. >>> b = a      #不创建新对象
3. >>> b **is** a           # a和b是同一个数组对象的两个名字
4. True
5. >>> b.shape = 3,4    #也改变了a的形状
6. >>> a.shape
7. (3, 4)

 　　　Python 传递不定对象作为参考4，所以函数调用不拷贝数组。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> **def** f(x):
2. ...     **print** id(x)
3. ...
4. >>> id(a)       #id是一个对象的唯一标识
5. 148293216
6. >>> f(a)
7. 148293216

**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。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> c = a.view()
2. >>> c **is** a
3. False
4. >>> c.base **is** a      #c是a持有数据的镜像
5. True
6. >>> c.flags.owndata
7. False
8. >>>
9. >>> c.shape = 2,6    # a的形状没变
10. >>> a.shape
11. (3, 4)
12. >>> c[0,4] = 1234        #a的数据改变了
13. >>> a
14. array([[   0,    1,    2,    3],
15. [1234,    5,    6,    7],
16. [   8,    9,   10,   11]])

切片数组返回它的一个视图：

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> s = a[ : , 1:3]     # 获得每一行1，2处的元素
2. >>> s[:] = 10           # s[:] 是s的镜像。注意区别s=10 and s[:]=10
3. >>> a
4. array([[   0,   10,   10,    3],
5. [1234,   10,   10,    7],
6. [   8,   10,   10,   11]])

**深复制**

 　　这个复制方法完全复制数组和它的数据。

**[python]** [view plaincopy](http://blog.csdn.net/sunny2038/article/details/8907736)

1. >>> d = a.copy()       #创建了一个含有新数据的新数组对象
2. >>> d **is** a
3. False
4. >>> d.base **is** a        #d和a现在没有任何关系
5. False
6. >>> d[0,0] = 9999
7. >>> a
8. array([[   0,   10,   10,    3],
9. [1234,   10,   10,    7],
10. [   8,   10,   10,   11]])

参考文献：

《Python科学计算》   
《Tentative NumPy Tutorial》  
《NumPy for Beginner》

# 《Python核心编程 》笔记分

春节终于over了，回归充实的学习研究生活。打开久违的CSDN博客，看到官方推送的 [『博客Markdown编辑器上线啦』](http://blog.csdn.net/csdnproduct/article/details/43561659)，让我顿时有了写作的欲望，真是程序员的福利。之前阅读各种文章书籍，都是用MarkDownPad做的笔记，喜欢以及习惯于MarkDown简洁的语法。

总之各种方便。为了试试效果，将以前阅读《Python核心编程》的手记整理发上来，也当温习一遍。

## 第二章 快速入门

* print语句中使用字符串格式操作符，实现字符替换功能。

print "%s is %d" %("one",1)

* + 1

%s、%d、%f分别用字符串、整型、浮点类型数据来替换。

* print语句重定向
* logfile = open('/tmp/mylog.txt','a')
* print &gt;&gt; logfile,'something....'

logfile.close()

* + 1
  + 2
  + 3

符号>>是用来重定向的，上面的代码将输出添加到日志文件mylog.txt中。

* raw\_input内建函数，读取键盘输入，返回值类型是字符串。

s = raw\_input('some tips：')

* + 1
* 操作符单斜杆 / 做传统的除法，双斜杠 //用作浮点除法，四舍五入。
* python中的变量名命名规则与其他大多数高级语言一样，都是受到C语言影响。（事实上python本身就是C语言写成的）
* python是动态型语言，不需要预先声明变量的类型。变量的类型在赋值的那一刻确定。
* python是动态型语言，不需要预先声明变量的类型。变量的类型在赋值的那一刻确定。
* python的长整型所能表达的范围远远超过C语言的长整型，仅受限于计算机的虚拟内存总数，不用担心溢出。
* something interesting：1.1无法用二进制精确表示，2.2、3.3、、、很多数字都无法用二进制精确表示。但python中提供了十进制浮点型decimal模块，可以精确表示。
* 元组：可看成是只读的列表，不可修改。
* 字典
* aDict = {"one":1,"two":2}
* aDict.keys() #输出['one','two']
* for key in aDict:

print key,aDict[key]

* + 1
  + 2
  + 3
  + 4
* python中的for循环与传统的for循环（计数器循环）不太一样，它更像shell里的foreach迭代。python中的for接受可迭代对象作为其参数，每次迭代其中的一个元素。在for语句中，经常还会用到range、len函数。
* for i in range(len(A)):

print A[i]

* + 1
  + 2
* 列表解析，这个非常实用

spdEvens = [x\*\*2 for x in range(8) if not x%2]

* + 1
* 文件和内建函数open()、file()

handle = open(file\_name,access\_mode = 'rb')

* + 1

r表示可读模式，b表示二进制访问，还有w表示可写模式（覆盖掉原来的），a表示添加模式（不覆盖），+表示读写模式。

open()返回文件的句柄handle，后续的操作如readlines()、close()可通过这个句柄进行：如handle.close()….

* try-except

try之后的代码组，即你打算管理的代码。except之后的代码组，则是处理错误的代码。

* 函数在调用前必须先定义（被调用函数要在调用处上方）。
* 如果函数没有return语句，会自动返回None对象。
* 所有名字开始和结束都有两个下划线的方法都是特殊方法。如**name**、**init**
* 模块，可以包含执行代码、函数、类。.py文件即一个模块。
* 常用的函数：int()、str()、range()、len()、raw\_input()、ttype()、dir([obj])、help([obj])

## 第三章 Python基础

### 1、语句和语法

* 注释

Python中注释用符号“#”，也可以用三引号：”’ 注释内容 ”’

* 继续

过长的语句，可以用反斜杠,将一行分解为几行：

if a==1 and \

b==0 :

* + 1
  + 2
* Python缩进风格

Pyhton使用缩进来分隔代码组，缩进可以为空格、制表符Tab（另：推荐使用4个空格来缩进，并且尽量不要用tab键，因为不同平台tab键的代表的空白宽度不一样。）

* 多个语句写在同一行上

把多个语句写在同一行上是不推荐的，因为可读性会大大降低。如果非要这么做，也是允许的。

import sys; x = 'foo'; sys.stdout.write(x)

* + 1
* 模块

通过import导入

### 2、变量赋值

* 多元赋值（multuple）

采用这种方式赋值，等号两边的对象其实都是元组：

x,y,z = 1,2,3

* + 1

本质上为：

( x,y,z ) = (1,2,3)

* + 1

这种多元赋值方式可以实现无需中间变量交换两个变量的值：

x,y = 1,2

x,y = y,x

* + 1
  + 2

执行代码，x和y的值将互换。

### 3、标志符

* 专用下划线标志符

\_XXX，不用from module import \*导入

\_ XXX \_，系统定义名字

\_XXX,类中的私有变量名

下划线对解析器具有特许的意义，而且是内建标志符使用的符号，所以建议避免用下划线作为变量名的开始。

### 4、Python基本风格

注释、缩进、标志符风格以上述及，此外Python还提供了一个机制“文档”：可以通过\_ doc \_变量，动态获得文档字符串：

obj.\_\_doc\_\_

* 1

### 5、内存管理

* 变量定义

大多数编译型语言，变量在使用前必须先声明，而Python作为解析型语言，无须显示声明变量，变量在第一次被赋值时自动声明。

* 动态类型

Python不但无须事先声明变量，而且也无需类型声明。对象的类型和内存占用都是在运行时确定的。

* 内存分配

Python承担了内存管理的复杂任务，程序员无需关心内存管理。

* 引用计数

Python使用引用计数这个技术来追踪内存中的对象。

**增加引用计数：**

当对象被创建时，就生成了一个引用，引用计数被设置为1。当同一个对象又被赋值给其他变量时，或作为参数传递给函数、方法、类实例时，或被赋值为一个窗口对象的成员时，该对象一个新的引用就被创建，引用计数自动加1。

**减少引用计数：**

当对象的引用被销毁时，引用计数会减少。比如：   
(1)函数运行结束，局部变量被自动销毁，对象的引用计数也会随之减少。   
(2)变量被重新赋值（值改变），引用计数减1。   
(3)del obj，obj引用计数减1。

* 垃圾收集

Python的垃圾收集器实际上是一个引用计数器和一个循环垃圾收集器。

## 第4章 Python对象

### 1、Python对象

Python使用对象模型来存储数据，构造任何类型的值都是一个对象。所有的对象都有三个特性：

* 身份，可通过内建函数id()查看，这个值即该对象的内存地址。
* 类型，可通过内建函数type()查看。
* 值，对象表示的数据项。

>>> p = 12

>>> id(p)

31108092

>>> type(p)

<type 'int'>

>>> p

12

>>>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

### 2、标准类型

整型Integer，长整型Long integer，浮点型float，复数型complex number，布尔型bool，字符串string，列表list，元组tuple，字典dictionary。

### 3、其他内建类型

* type类型对象

type类型本身就是一个对象，它的类型为‘type’。

>>> type(1)

<type 'int'>

>>> type(type(1))

<type 'type'>

* 1
* 2
* 3
* 4
* None——Python的Null对象

Python有一个特殊的类型，称作Null对象或者NoneType，它只有一个值：None，None的布尔值为False。

* 文件
* 集合
* 函数/方法
* 模块
* 类

### 4、内部类型

内部类型我们一般不会过多关注与使用，了解一下即可。

* 代码对象

代码对象是编译过的Python源代码片段，可执行。可以通过内建函数compile()得到代码对象。代码对象可以被exec命令或者内建函数eval()执行。

* 帧
* 跟踪记录对象

程序出现异常退出时，一个包含针对异常的栈跟踪信息的跟踪记录对象被创建：

>>> pri

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'pri' is not defined

* 1
* 2
* 3
* 4
* 切片对象

当使用Python的切片语法时，就会创建切片对象。

* 省略对象

用于切片语法中，起记号作用。 如 str[::2]之类。

* Xrange对象

调用内建函数xrange()会生成一个XRange对象，xrange是range的兄弟版本，用于需要节省内存 或 range无法完成的超大数据集场合。

### 5、标准类型操作符

* 对象值的比较
* 对象身份比较

这个涉及到Python“引用计数”的相关知识，上一篇文章中有总结。Python还提供了is和is not操作符来测试两个变量是否指向同一个对象。

>>> a = 1

>>> b =a

>>> c = 2

>>> a == b

True

>>> a is c

False

>>> a is not c

True

>>> a is b

True

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

我们通过id也可以直接判断它们是否指向同一对象：

>>> id(a),id(b),id(c)

(31108224, 31108224, 31108212)

* 1
* 2

### 6、标准类型内建函数

* cmp(obj1,obj2)

obj1大于obj2则返回1，小于返回-1，等于返回0。

>>> a,b,c,d = 1,2,3,2

>>> cmp(b,a),cmp(b,c),cmp(b,d)

(1, -1, 0)

* 1
* 2
* 3
* type(obj)
* str()、repr()、”操作符

str()、repr()、”操作符都能够以字符串的方式获取对象的内容。str()获得的字符串可读性好，repr()获取的字符串通常可以用来重新获得该对象。

>>> str([1,2,3])

'[1, 2, 3]'

>>> repr([1,2,3])

'[1, 2, 3]'

>>> '[1,2,3]'

'[1,2,3]'

* 1
* 2
* 3
* 4
* 5
* 6

### 7、类型工厂函数

Python2.2统一了类和类型。原来所谓的内建类型转换函数像int()、type()、list()等都成了工厂函数，也就是说他们看上去像函数，实质上是类，当调用它们时，实际上生成了该类型的一个实例，像工厂生成货物一样。

### 8、标准类型的分类

存储模型，更新模型，访问模型。

### 9、Python不支持的类型

* char或byte
* 指针
* 整型

Python中没有像C语言那样的int、short、long。当你使用一个整型数值超出范围时，python会自动返回一个长整型给你，python的长整型表示的数值范围很大。

* float vs double

Python的浮点类型float实际上是C的double。浮点型总是不精确的，故Python还提供了Decimals模块，它有任意精度，在处理金钱这类确定的值时，Decimals模块很有用。

## [数字类型](http://blog.csdn.net/u012162613/article/details/44286655)

## 1、数字类型简介

* Python中数字类型包括：整型、长整型、布尔型、双精度浮点型、十进制浮点型、复数。这些数字类型都是不可变类型，也就是说，改变了数字的值会生成新的对象。
* 在Python中删除数字对象，可以用语句：del aInt,aLong,aFloat,aComplex

## 2、整型

* 布尔型

取值范围只有两个值，True和False。它们在数学运算中对应1和0.

对于值为0的任何数字或空集（空列表、空元组、空字典等）在Python中的布尔值都是False。

* 长整型

在其他编程语言中，比如C++中，int占用了32位，long int则取决于机器字长，long long int为64位。而在Python中，标准整型一般也是占用32位，但是如果在64位机器上用64位的编译器编译Python，则整型占用64位。

Python中长整型可以支持的范围很大，仅仅取决机器支持的虚拟内存大小。长整型的声明：直接在数字后加L：12345L

Python中整型与长整型逐渐统一为一种，所以使用时不用去关心整型还是长整型。

## 3、双精度浮点型

Python中的浮点型类似C中的double型，是双精度浮点型，占用64位。

## 4、复数

Python中的复数类型规定：

* 复数由实数部分和虚数部分构成：real + imagj
* 虚数部分不能单独存在，必须与0.0的实数部分一起构成复数
* 实数部分real和虚数部分imag都是浮点类型。
* 复数类型有三个内建属性： real、imag、conjugate（返回共轭复数）

>>> aComplex = 1.0 + 2.0j

>>> aComplex.real

1.0

>>> aComplex.imag

2.0

>>> aComplex.conjugate()

(1-2j)

* 1
* 2
* 3
* 4
* 5
* 6
* 7

## 5、操作符

**自动类型转换**

当对两个不同类型的数进行操作时，Python会自动进行类型转换。基本的原则就是：整型转换为浮点型，非复数转换为复数。

**真正的除法 vs 地板除**

现在Python的版本中调用除号“/”，执行的都是真正的除法：

>>> 1/2

0.5

>>> 1.0/2.0

0.5

* 1
* 2
* 3
* 4

另外还有一种除法叫做地板除，它不管操作数为何种类型，总是舍去小数部分，可以通过符合“//”实现：

>>> 1.0//2.0

0.0

* 1
* 2

**冪运算**

通过操作符 \*\* 实现，注意各种符号的优先级，或者直接通过括号()消除优先级带来的歧义：

>>> -2\*\*4

-16

>>> (-2)\*\*4

16

* 1
* 2
* 3
* 4

**位操作符**

Python整型支持标准位运算：取反 ~ 、按位与 & 、按位或 | 、按位异或^ 、左移<< 、右移>>

## 6、工厂函数

* 标准类型函数如cmp()、str()、type()
* 数字类型函数   
  + 转换工厂函数：int()、long()、float、conplex()
  + 功能函数：   
    - abs(num)，返回绝对值
    - corece(num1,num2)将num1、num2转换为同类型，然后以元组形式返回
    - divmod(num1,num2)，返回(num1/num2,num1%num2)
    - pow(num1,num2)，实现num1\*\*num2
    - round(flt,ndig=0),对浮点数flt四舍五入，保存ndig位小数，默认为0。
* 仅用于整型的函数   
  + 进制转换函数，oct()、hex()
  + ASCII转换函数   
    - chr()接受一个单字节整型值，返回对应的字符
    - ord()接受一个字符，返回对应的整型值

>>> chr(97)

'a'

>>> ord('a')

97

* 1
* 2
* 3
* 4

## 7、相关模块

* decimal 十进制浮点运算类 Decimal
* array 高效数值数组
* math/cmath 标准C库数学运算函数，常规数学运算在math中，复数运算在cmath中
* operator 数学操作符的函数实现，比如 operator.sub(2,1)相当于2-1
* random 多种伪随机数生成器

>>> import operator

>>> operator.sub(2,1)

1

>>> import random

>>> random.randint(0,10)

3

>>> random.randint(0,10)

7

[【NumPy基础】100道numpy练习——初学与入门篇](http://blog.csdn.net/u012162613/article/details/42784403)

@author：wepon

@blog：<http://blog.csdn.net/u012162613/article/details/42784403>

今天在deeplearning.net上看theano tutorial，发现一个numpy-100-exercise，介绍numpy一些基本用法的，不过不是很具体，我利用闲暇时间照着敲了一些，权且当作翻译吧，增加函数的原型和详细介绍。持续更新。

**一、初学者10道**

1、在python环境中导入numpy包，并命名为np

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> **import** numpy as np</span>

2、查看numpy版本和配置信息

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> **print** np.\_\_version\_\_
2. >>> np.\_\_config\_\_.show()</span>

3、创建零向量，zeros函数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.zeros((2,3))
2. >>> **print** z
3. [[ 0.  0.  0.]
4. [ 0.  0.  0.]]</span>

4、将上面的零向量的第二行第三列元素置为1。注意python中行列下班是从0开始。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>>z[1,2]=1
2. >>> **print** z
3. [[ 0.  0.  0.]
4. [ 0.  0.  1.]]</span>

5、arange函数，创建一个在给定范围的向量。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.arange(1,101)    %1～100范围，注意不包括101
2. >>> **print** z</span>

6、reshape函数，将array变形为矩阵

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.arange(9).reshape(3,3)
2. >>> **print** Z
3. [[0 1 2]
4. [3 4 5]
5. [6 7 8]]</span>

7、nonzero函数，寻找非0元素的下标

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> nz=np.nonzero([1,2,3,0,0,4,0])
2. >>> nz
3. (array([0, 1, 2, 5]),)</span>

8、eye函数，生成单位向量

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.eye(3)
2. >>> **print** z
3. [[ 1.  0.  0.]
4. [ 0.  1.  0.]
5. [ 0.  0.  1.]]</span>

9、diag函数，diagonal对角线。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.diag([1,2,3,4],k=0)   %k=0，以[1,2,3,4]为对角线
2. >>> **print** z
3. [[1 0 0 0]
4. [0 2 0 0]
5. [0 0 3 0]
6. [0 0 0 4]]
8. >>> z=np.diag([1,2,3,4],k=1)   %k=1，[1,2,3,4]在对角线上一行
9. >>> **print** z
10. [[0 1 0 0 0]
11. [0 0 2 0 0]
12. [0 0 0 3 0]
13. [0 0 0 0 4]
14. [0 0 0 0 0]]
16. >>> z=np.diag([1,2,3,4],k=-1)  %k=-1，[1,2,3,4]在对角线下一行
17. >>> **print** z
18. [[0 0 0 0 0]
19. [1 0 0 0 0]
20. [0 2 0 0 0]
21. [0 0 3 0 0]
22. [0 0 0 4 0]]</span>

10、random模块的random函数，生成随机数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.random.random((3,3))
2. >>> **print** Z
3. [[ 0.95171484  0.61394126  0.38864802]
4. [ 0.41943918  0.9398714   0.31608202]
5. [ 0.9993507   0.91717093  0.73002723]]</span>

**二、入门级10道**

1、创建一个8\*8的“棋盘”矩阵。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.zeros((8,8),dtype=int)
2. >>> z[1::2,::2]=1     %1、3、5、7行&&0、2、4、6列的元素置为1
3. >>> **print** z
4. [[0 0 0 0 0 0 0 0]
5. [1 0 1 0 1 0 1 0]
6. [0 0 0 0 0 0 0 0]
7. [1 0 1 0 1 0 1 0]
8. [0 0 0 0 0 0 0 0]
9. [1 0 1 0 1 0 1 0]
10. [0 0 0 0 0 0 0 0]
11. [1 0 1 0 1 0 1 0]]
12. >>> z[::2,1::2]=1
13. >>> **print** z
14. [[0 1 0 1 0 1 0 1]
15. [1 0 1 0 1 0 1 0]
16. [0 1 0 1 0 1 0 1]
17. [1 0 1 0 1 0 1 0]
18. [0 1 0 1 0 1 0 1]
19. [1 0 1 0 1 0 1 0]
20. [0 1 0 1 0 1 0 1]
21. [1 0 1 0 1 0 1 0]]</span>

2、min()、max()函数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.random.random((10,10))
2. >>> zmin,zmax=z.min(),z.max()
3. >>> **print** zmin,zmax
4. 0.014230501632 0.99548760299</span>

3、函数tile(A,reps),reps即重复的次数，不仅可以是数字，还可以是array。比如构造棋盘矩阵：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.tile(np.array([[0,1],[0,1]]),(4,4))
2. >>> **print** z
3. [[0 1 0 1 0 1 0 1]
4. [0 1 0 1 0 1 0 1]
5. [0 1 0 1 0 1 0 1]
6. [0 1 0 1 0 1 0 1]
7. [0 1 0 1 0 1 0 1]
8. [0 1 0 1 0 1 0 1]
9. [0 1 0 1 0 1 0 1]
10. [0 1 0 1 0 1 0 1]]</span>

4、归一化，将矩阵规格化到0～1，即最小的变成0，最大的变成1，最小与最大之间的等比缩放。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.random.random((5,5))
2. >>> Zmax,Zmin = Z.max(), Z.min()
3. >>> Z = (Z - Zmin)/(Zmax - Zmin)
4. >>> **print** Z
5. [[ 0.          0.32173291  0.17607851  0.6270374
7. 0.95000808]
8. [ 0.49153473  0.70465605  0.61930085  0.00303294  1.
10. ]
11. [ 0.4680561   0.88742782  0.29899683  0.80704789
13. 0.12300414]
14. [ 0.05094248  0.23065875  0.82776775  0.07873239
16. 0.50644422]
17. [ 0.27417053  0.78679222  0.517819    0.5649124   0.4716856
19. ]]</span>

5、矩阵点乘

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> z=np.dot(np.ones((5,3)),np.ones((3,2)))
2. >>> **print** z
3. [[ 3.  3.]
4. [ 3.  3.]
5. [ 3.  3.]
6. [ 3.  3.]
7. [ 3.  3.]]</span>

6、矩阵相加，5\*5矩阵+1\*5的向量，相当于每一行都加上1\*5矩阵

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.zeros((5,5))
2. >>> Z += np.arange(5)
3. >>> **print** Z
4. [[ 0.  1.  2.  3.  4.]
5. [ 0.  1.  2.  3.  4.]
6. [ 0.  1.  2.  3.  4.]
7. [ 0.  1.  2.  3.  4.]
8. [ 0.  1.  2.  3.  4.]]</span>

7、linspace函数，在给定区间中生成均匀分布的给定个数。  
函数原型 linspace(start, stop, num=50, endpoint=True, retstep=False)

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.linspace(0,10,11,endpoint=True, retstep=False)
2. >>> **print** Z
3. [  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]</span>

生成0~10之间均匀分布的11个数，包括0和1。  
若endpoint=False,则10不包括在里面。  
若retstep=False，会同时返回均匀区间中每两个数的间隔。

8、sort函数。调用random模块中的random函数生成10个随机数，然后sort排序。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.random.random(10)
2. >>> Z.sort()
3. >>> **print** Z
4. [ 0.15978787  0.28050494  0.35865916  0.40047826  0.45141311
6. 0.4828367
7. 0.66133575  0.66775779  0.69278544  0.98095989]</span>

9、allclose函数，判断两个array在误差范围内是否相等  
函数原型allclose(a, b, rtol=1e-05, atol=1e-08)，若absolute(a - b) <= (atol + rtol \* absolute(b))则相等。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">A = np.random.randint(0,2,5)
2. B = np.random.randint(0,2,5)
3. equal = np.allclose(A,B)
4. **print** equal</span>

10、mean函数，求平均值

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42784403)

1. <span style="font-size:18px;">>>> Z = np.random.random(30)
2. >>> m = Z.mean()
3. >>> **print** m
4. 0.362299527973
6. >>> A = np.random.randint(0,2,5)
7. >>> B = np.random.randint(0,2,5)
8. >>> equal = np.allclose(A,B)
9. >>> **print** equal
10. False
11. </span>

注：randint(min,max,num)生成大小为num的array，数值范围min～max

[【NumPy基础】100道numpy练习——Apprentice篇](http://blog.csdn.net/u012162613/article/details/42811297)

**@author：wepon**

**@blog：http://blog.csdn.net/u012162613/article/details/42811297**

今天又用半小时扫了一下Apprentice篇里的10道exercise，不知道怎么翻译Apprentice（学徒～～）这个词，就直接以Apprentice篇作为题目了。numpy语法直白如水啊，花这些时间exercise有点浪费了.......Anyway，为了后面更熟练地用一些ML的包，利用闲暇时间扫一扫基本语法。原地址：<https://github.com/rougier/numpy-100>

1、将array属性设置为只读（read-only）

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.zeros(10)
2. >>> Z.flags.writeable = False
3. >>> Z[0] = 1
4. Traceback (most recent call last):
5. File "<stdin>", line 1, **in** <module>
6. ValueError: assignment destination **is** read-only

2、将笛卡尔坐标系下的点(x,y)转化为极坐标系的(R,T)

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.random.random((10,2))
2. >>> **print** Z
3. [[ 0.49094922 0.58585236]
4. [ 0.32265092 0.14445935]
5. [ 0.8078448 0.70284832]
6. [ 0.35341989 0.76888775]
7. [ 0.01212107 0.43668101]
8. [ 0.36924292 0.64075512]
9. [ 0.22349212 0.4561141 ]
10. [ 0.44836271 0.19591462]
11. [ 0.6639701 0.16595659]
12. [ 0.25559111 0.01741761]]
13. >>> X,Y = Z[:,0], Z[:,1]
14. >>> R = np.sqrt(X\*\*2+Y\*\*2)
15. >>> T = np.arctan2(Y,X) %求反正切值
16. >>> **print** R
17. [ 0.76436518 0.35351396 1.07079829 0.84622337 0.4368492
18. 0.73953192
19. 0.50792598 0.48929711 0.684396 0.2561839 ]
20. >>> **print** T
21. [ 0.87330531 0.42096163 0.71600756 1.13994582 1.5430462
22. 1.04801403
23. 1.11518737 0.41195346 0.24492773 0.06804118]

3、获取数组中最大值/最小值对应的下标，argmax()、argmin()函数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.random.random(10)
2. >>> **print** Z
3. [ 0.90315764 0.06574957 0.86297424 0.46519603 0.01174512
4. 0.60977188
5. 0.52107975 0.6328012 0.12344019 0.05034712]
6. >>> Z.argmax()
7. 0
8. >>> Z.argmin()
9. 4

4、meshgrid()产生格点矩阵，用于三维曲面的分格线座标

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.zeros((10,10), [('x',float),('y',float)])
2. >>> Z['x'], Z['y'] = np.meshgrid(np.linspace(0,1,10),
3. np.linspace(0,1,10))
4. >>> **print** Z

5、各种数据类型的最大值/最小值，主要有8位、32位、64位整型，32位、64位浮点型

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> **for** dtype **in** [np.int8, np.int32, np.int64]:
2. ... **print** np.iinfo(dtype).min
3. ... **print** np.iinfo(dtype).max
4. ...
5. -128
6. 127
7. -2147483648
8. 2147483647
9. -9223372036854775808
10. 9223372036854775807
11. >>> **for** dtype **in** [np.float32, np.float64]:
12. ... **print** np.finfo(dtype).min
13. ... **print** np.finfo(dtype).max
14. ... **print** np.finfo(dtype).eps
15. ...
16. -3.40282e+38
17. 3.40282e+38
18. 1.19209e-07
19. -1.79769313486e+308
20. 1.79769313486e+308
21. 2.22044604925e-16

6、生成一个矩阵，矩阵中每个元素由坐标(x,y)以及颜色(r,g,b)组成

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.zeros(10, [ ('position', [ ('x', float, 1),
2. ... ('y', float, 1)]),
3. ... ('color', [ ('r', float, 1),
4. ... ('g', float, 1),
5. ... ('b', float, 1)])])
6. >>> **print** Z
7. [((0.0, 0.0), (0.0, 0.0, 0.0)) ((0.0, 0.0), (0.0, 0.0, 0.0))
8. ((0.0, 0.0), (0.0, 0.0, 0.0)) ((0.0, 0.0), (0.0, 0.0, 0.0))
9. ((0.0, 0.0), (0.0, 0.0, 0.0)) ((0.0, 0.0), (0.0, 0.0, 0.0))
10. ((0.0, 0.0), (0.0, 0.0, 0.0)) ((0.0, 0.0), (0.0, 0.0, 0.0))
11. ((0.0, 0.0), (0.0, 0.0, 0.0)) ((0.0, 0.0), (0.0, 0.0, 0.0))]

7、生成10个坐标(x,y)，计算两两之间的距离

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.random.random((10,2))
2. >>> X,Y = np.atleast\_2d(Z[:,0]), np.atleast\_2d(Z[:,1])
3. >>> D = np.sqrt( (X-X.T)\*\*2 + (Y-Y.T)\*\*2)
4. >>> **print** D %D是10\*10,对角线的值都是0

也可以直接用scipy里的cdist函数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. # Much faster with scipy
2. **import** scipy
3. Z = np.random.random((10,2))
4. D = scipy.spatial.distance.cdist(Z,Z)
5. **print** D

8、生成二维的高斯矩阵

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> X, Y = np.meshgrid(np.linspace(-1,1,10), np.linspace(-
2. 1,1,10))
3. >>> D = np.sqrt(X\*X+Y\*Y)
4. >>> sigma, mu = 1.0, 0.0 %方差=1,均值=0
5. >>> G = np.exp(-( (D-mu)\*\*2 / ( 2.0 \* sigma\*\*2 ) ) )
6. >>> **print** G
7. [[ 0.36787944 0.44822088 0.51979489 0.57375342 0.60279818
8. 0.60279818
9. 0.57375342 0.51979489 0.44822088 0.36787944]
10. [ 0.44822088 0.54610814 0.63331324 0.69905581 0.73444367
11. 0.73444367
12. 0.69905581 0.63331324 0.54610814 0.44822088]
13. [ 0.51979489 0.63331324 0.73444367 0.81068432 0.85172308
14. 0.85172308
15. 0.81068432 0.73444367 0.63331324 0.51979489]
16. [ 0.57375342 0.69905581 0.81068432 0.89483932 0.9401382
17. 0.9401382
18. 0.89483932 0.81068432 0.69905581 0.57375342]
19. [ 0.60279818 0.73444367 0.85172308 0.9401382 0.98773022
20. 0.98773022
21. 0.9401382 0.85172308 0.73444367 0.60279818]
22. [ 0.60279818 0.73444367 0.85172308 0.9401382 0.98773022
23. 0.98773022
24. 0.9401382 0.85172308 0.73444367 0.60279818]
25. [ 0.57375342 0.69905581 0.81068432 0.89483932 0.9401382
26. 0.9401382
27. 0.89483932 0.81068432 0.69905581 0.57375342]
28. [ 0.51979489 0.63331324 0.73444367 0.81068432 0.85172308
29. 0.85172308
30. 0.81068432 0.73444367 0.63331324 0.51979489]
31. [ 0.44822088 0.54610814 0.63331324 0.69905581 0.73444367
32. 0.73444367
33. 0.69905581 0.63331324 0.54610814 0.44822088]
34. [ 0.36787944 0.44822088 0.51979489 0.57375342 0.60279818
35. 0.60279818
36. 0.57375342 0.51979489 0.44822088 0.36787944]]

9、any函数，判断矩阵中的元素是否满足给定的条件，是的话返回True，否则返回False。例子：判断二维矩阵中有没有一整列数为0？

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z = np.random.randint(0,3,(2,10))
2. >>> **print** Z
3. [[1 0 1 2 2 0 1 0 1 0]
4. [0 0 1 2 2 0 1 0 0 1]]
5. >>> **print** Z.any(axis=0) %按列判断，全0则为False
6. [ True False True True True False True False True True]
7. >>> **print** (~Z.any(axis=0)).any()
8. True

10、找出数组中与给定值最接近的数

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z=array([[0,1,2,3],[4,5,6,7]])
2. >>> **print** Z
3. [[0 1 2 3]
4. [4 5 6 7]]
5. >>> z=5.1
6. >>> np.abs(Z - z).argmin()
7. 5
8. >>> **print** Z.flat[np.abs(Z - z).argmin()]
9. 5

补充介绍：  
flat是一个一维迭代器，将Z当成一维向量去遍历，Z[5]将超出原来矩阵的边界，Z.flat[5]才是正确的用法。  
另外，不得不提flatten，它将矩阵摊平为一维向量:

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42811297)

1. >>> Z.flatten()

[【NumPy基础】100道numpy练习——进阶篇](http://blog.csdn.net/u012162613/article/details/42847011)

@author：wepon

@blog：<http://blog.csdn.net/u012162613/article/details/42846777>

选自[numpy-100](https://github.com/rougier/numpy-100)，当作熟悉NumPy的练习。NumPy只是一个数值计算的工具包，在实际的算法实现中来熟悉NumPy才是有效的，因此后面不打算继续写了，到此文为止，基本的语法已经够用了，之后在实践中总结可能效果更好。而且遇到问题参考NumPy官网文档即可。

之前两篇：

[【NumPy基础】100道numpy练习——初学与入门篇](http://blog.csdn.net/u012162613/article/details/42784403)

[【NumPy基础】100道numpy练习——Apprentice篇](http://blog.csdn.net/u012162613/article/details/42811297)

1、读入文件中的数据，例如一个example.txt，里面的数据如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. 1,2,3,4
2. 5,,,6
3. 7,8,,9

读入该文件的代码：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> Z = np.genfromtxt("example.txt", delimiter=",")
2. >>> **print** Z
3. [[ 1. 2. 3. 4.]
4. [ 5. nan nan 6.]
5. [ 7. 8. nan 9.]]

note：np.genfromtxt()函数第一个参数表示文件路径名，delimiter是分隔符，在我们的exampl.txt中分隔符是逗号“,”，故设置为逗号。顺便提一下，很多数据文件都是以csv后缀格式给出的，csv就是逗号分隔符文件。

2、numpy的高级特性：生成器。利用生成器函数创建数组。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> **def** generate():
2. ... **for** x **in** xrange(10):
3. ... **yield** x
4. ...
5. >>> Z = np.fromiter(generate(),dtype=float,count=-1)
6. >>> **print** Z
7. [ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]

note：其中的yield关键字使得generate()变成生成器，这是numpy的高级特性。

3、bincount()函数，输出数组中每个数出现的次数。output的下标代表原数组中的数，output的值就是该数出现的次数，举例：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> Z=[0,2,4,8,8]
2. >>> np.bincount(Z)
3. array([1, 0, 1, 0, 1, 0, 0, 0, 2])

可以看到8出现了两次。

4、 bincount(x, weights=None, minlength=None)，其中weights表示权重，用法如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> X = [1,2,3,4,5,6]
2. >>> I = [1,3,9,3,4,1]
3. >>> F = np.bincount(I,X)
4. >>> **print** F
5. [ 0. 7. 0. 6. 5. 0. 0. 0. 0. 3.]

note：以I为输入，X是权重。分析一下：F[0]=0，因为I中没有出现过0，F[1]=7，因为I中1出现了两次，并且它们的权重为1、6，故F[1]=1\*1+1\*6

5、Considering a (w,h,3) image of (dtype=ubyte), compute the number of unique colors

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. # Author: Nadav Horesh
2. w,h = 16,16
3. I = np.random.randint(0,2,(h,w,3)).astype(np.ubyte)
4. F = I[...,0]\*256\*256 + I[...,1]\*256 +I[...,2]
5. n = len(np.unique(F))
6. **print** np.unique(I)

6、一个四维的数组，以后两维度为单位，计算它们的和，比如一个1\*2\*3\*4的数组，则以后面3\*4维度为单位，输出1\*2的sum，举例：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> A = np.random.randint(0,10,(1,2,3,4))
2. >>> **print** A
3. [[[[2 7 9 7]
4. [6 6 8 2]
5. [0 0 9 3]]
6. [[5 4 1 4]
7. [5 7 9 7]
8. [8 4 1 4]]]]
9. >>> A.reshape(A.shape[:-2] + (-1,))
10. array([[[2, 7, 9, 7, 6, 6, 8, 2, 0, 0, 9, 3],
11. [5, 4, 1, 4, 5, 7, 9, 7, 8, 4, 1, 4]]])
12. >>> sum = A.reshape(A.shape[:-2] + (-1,)).sum(axis=-1)
13. >>> **print** sum
14. [[59 59]]

7、Considering a one-dimensional vector D, how to compute  means of subsets of D using a vector S of same size describing subset indices ?

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> D = np.random.uniform(0,1,100)
2. >>> S = np.random.randint(0,10,100)
3. >>> D\_sums = np.bincount(S, weights=D)
4. >>> D\_counts = np.bincount(S)
5. >>> **print** D\_sums
6. [ 3.37619132 6.13452126 3.84121952 5.27577033 4.45979323
7. 7.26807049
8. 6.00231146 6.72050881 4.1281527 6.55666661]
9. >>> **print** D\_counts
10. [ 6 12 10 8 7 13 13 12 7 12]
11. >>> D\_means = D\_sums / D\_counts
12. >>> **print** D\_means
13. [ 0.56269855 0.5112101 0.38412195 0.65947129 0.63711332
14. 0.55908235
15. 0.46171627 0.5600424 0.5897361 0.54638888]

8、在数组[1, 2, 3, 4, 5]中相邻两个数字中间插入两个0

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> Z = np.array([1,2,3,4,5])
2. >>> nz = 3
3. >>> Z0 = np.zeros(len(Z) + (len(Z)-1)\*(nz))
4. >>> Z0[::nz+1] = Z
5. >>> **print** Z0
6. [ 1. 0. 0. 0. 2. 0. 0. 0. 3. 0. 0. 0. 4. 0. 0.
7. 0. 5.]

9、二维矩阵与三维矩阵如何相乘？

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> A = np.ones((3,3,2))
2. >>> B = 2\*np.ones((3,3))
3. >>> **print** A \* B[:,:,None]
4. [[[ 2. 2.]
5. [ 2. 2.]
6. [ 2. 2.]]
7. [[ 2. 2.]
8. [ 2. 2.]
9. [ 2. 2.]]
10. [[ 2. 2.]
11. [ 2. 2.]
12. [ 2. 2.]]]
13. >>> B[:,:,None]
14. array([[[ 2.],
15. [ 2.],
16. [ 2.]],
17. [[ 2.],
18. [ 2.],
19. [ 2.]],
20. [[ 2.],
21. [ 2.],
22. [ 2.]]])

10、怎么交换矩阵的其中两行？比如交换第一二行

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42847011)

1. >>> A = np.arange(25).reshape(5,5)
2. >>> **print** A
3. [[ 0 1 2 3 4]
4. [ 5 6 7 8 9]
5. [10 11 12 13 14]
6. [15 16 17 18 19]
7. [20 21 22 23 24]]
8. >>> A[[0,1]]
9. array([[0, 1, 2, 3, 4],
10. [5, 6, 7, 8, 9]])
11. >>> A[[1,0]]
12. array([[5, 6, 7, 8, 9],
13. [0, 1, 2, 3, 4]])
14. >>> A[[0,1]] = A[[1,0]]
15. >>> **print** A
16. [[ 5 6 7 8 9]
17. [ 0 1 2 3 4]
18. [10 11 12 13 14]
19. [15 16 17 18 19]
20. [20 21 22 23 24]]