```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  8 15:27:41 2017

@author: zhxing
"""

#关于numpy的一些学习
import numpy as np
#1--数组
a=np.array([1,2,3,4,5])
print(type(a))
print(a.shape)
print(a[0],a[1])
a[0]=5
print(a)

b=np.array([[1,2,3],[4 ,5, 6]])
 #python不能用分号来分开矩阵，要用中括号,数字之间也必须有逗号
print(b)
#2--多元数组创建
a2=np.zeros((2,2))
b2=np.ones((3,5))
c2=np.full((3,3),7)
d2=np.eye(5)        #单位阵只接受一个参数
e2=np.random.random((3,3))
print(a2,b2,c2,d2,e2)
#3--访问数组
a3=np.array([[1,22,3],[3,3,4],[3,3,4]])
print(a3[:2,:])       #坐标是从零开始的，且是左闭右开区间，和matlab有点像但是不一样
print(a3[0,1])        #先行后列
b3=a3[:2,:2]         #这种访问和matlab相似但不同
print(a3)
#============================================================================
# a3:
# [[ 1 22  3]
#  [ 3  3  4]
#  [ 3  3  4]]
#
#============================================================================
c3=np.array([2,1,0])
print(a3[np.arange(3),c3])
#============================================================================
#
# [3 3 3]
#============================================================================
a3[np.arange(3),c3] +=5
print(a3)
#============================================================================
# [[ 1 22  8]
#  [ 3  8  4]
#  [ 8  3  4]]    #这里是给所有的对应元素加上一个值而不是取出了，如上
#============================================================================
#4--布尔型访问数组
#通常都是用于选取某种符合条件的元素
print(a3[a3>5])
#============================================================================
# [22  8  8  8]
#============================================================================
bool_index=(a3>5)
```

```
print(bool_index)
#==========================================================================================
# [[False  True  True]
#  [False  True False]
#  [ True False False]]
#==========================================================================================
#4--数据类型
a4=np.array([2,3,3,4])
print(a4.dtype)
a4=np.array([2,3.0,3,4])
print(a4.dtype)
#==========================================================================================
# int32
# float64
#==========================================================================================
#5--数组计算
a5 = np.array([[1,2],[3,4]], dtype=np.float64)
b5 = np.array([[5,6],[7,8]], dtype=np.float64)
print(a5+b5)
print(np.add(a5,b5))    #结果是一样的

#=====对应元素相加==========================================================================
# [[  6.   8.]
#  [ 10.  12.]]
#==========================================================================================
print(a5-b5)
print(np.subtract(a5,b5))    #结果是一样的
#==========================================================================================
# [[-4. -4.]
#  [-4. -4.]]
# [[-4. -4.]
#  [-4. -4.]]
#==========================================================================================
print(a5*b5)
print(np.multiply(a5,b5))    #结果是一样的
#==========================================================================================
# [[  5.  12.]
#  [ 21.  32.]]
# [[  5.  12.]
#  [ 21.  32.]]
#==========================================================================================
print(a5/b5)
print(np.divide(a5,b5))    #结果是一样的
#==========================================================================================
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
#==========================================================================================
#这里和matlab不同，*表示的是元素点乘
print(a5.dot(b5))
print(np.dot(a5,b5))
#===============这个是矩阵乘法=============================================================
# [[ 19.  22.]
#  [ 43.  50.]]
# [[ 19.  22.]
#  [ 43.  50.]]
#==============一维的话会自动转为内积=======================================================
c5=np.array([1,2,3])
d5=np.array([3,4,5])
```

```
print(c5.dot(d5))

print(np.sum(a5,axis=0))
print(np.sum(a5,axis=1))
#=============分别是列和和行和==========================================================
# [ 4.  6.]
# [ 3.  7.]
#================================================================================
print(a5.T)
#============转置==========================================================
# [[ 1.  3.]
#  [ 2.  4.]]
#================================================================================
#6---广播-broadcasting
#广播是一种非常有用的机制，可以让我们把不同大小的矩阵在一起运算，实际上是封装了一些循环进去，这是非常有用
#比如我们想把一个向量加到矩阵的每一行,我们当然可以用for循环的方式，但是数据量如果很大的话就太慢了
a6= np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
b6= np.array([1, 0, 1])

c6=np.tile(b6,(4,1))
print(c6)
#================================================================================
# [[1 0 1]
#  [1 0 1]
#  [1 0 1]
#  [1 0 1]]
#================================================================================
d6=a6+c6
print(d6)
#================================================================================
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
#===这种方式就比循环好多了，但numpy可以让我们不用做这种操作，=========================================
e6=a6+b6      #这样是可以直接相加的
print(e6)
#================================================================================
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
#================================================================================
#================================================================================
# 对两个数组使用广播机制要遵守下列规则：
#
# 如果数组的秩不同，使用1来将秩较小的数组进行扩展，直到两个数组的尺寸的长度都一样。
# 如果两个数组在某个维度上的长度是一样的，或者其中一个数组在该维度上长度为1，那么我们就说这两个数组在该维
# 如果两个数组在所有维度上都是相容的，他们就能使用广播。
# 如果两个输入数组的尺寸不同，那么注意其中较大的那个尺寸。因为广播之后，两个数组的尺寸将和那个较大的尺寸一
# 在任何一个维度上，如果一个数组的长度为1，另一个数组长度大于1，那么在该维度上，就好像是对第一个数组进行了
#=============下面是一点例子=======================================================
# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:
# [[ 4  5]
```

```
#  [ 8 10]
#  [12 15]]
print (np.reshape(v, (3, 1)) * w)

# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3),
# giving the following matrix:
# [[2 4 6]
#  [5 7 9]]
print (x + v)

# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column. Gives the following matrix:
# [[ 5  6  7]
#  [ 9 10 11]]
print ((x.T + w).T)

# Another solution is to reshape w to be a row vector of shape (2, 1);
# we can then broadcast it directly against x to produce the same
# output.
print (x + np.reshape(w, (2, 1)))

# Multiply a matrix by a constant:
# x has shape (2, 3). Numpy treats scalars as arrays of shape ();
# these can be broadcast together to shape (2, 3), producing the
# following array:
# [[ 2  4  6]
#  [ 8 10 12]]
print (x * 2)
#这里是numpy的文档：https://docs.scipy.org/doc/numpy/reference/
#7--图像操作
from scipy.misc import imread,imsave,imresize
img=imread('zx.jpg')
print(img.dtype,img.shape)
img_tine=img*[1,0.5,0.5]
img_tine=imresize(img_tine,(300,300))
imsave('zx_test.jpg',img_tine)

#8--可计算点间距离

from scipy.spatial.distance import pdist, squareform
a8=np.array([[1,2,3],[3,3,4],[44,4,5]])
# Compute the Euclidean distance between all rows of x.
# d[i, j] is the Euclidean distance between x[i, :] and x[j, :],
# and d is the following array:
#所以这样搞出来的肯定是对称的矩阵而且对角线为0
b8=squareform(pdist(a8,'euclidean'))
print(b8)

#9--matplotlib----是一个画图库
import matplotlib.pyplot as plt
a9=np.arange(0,3*np.pi,0.1)
b9_sin=np.sin(a9)
b9_cos=np.cos(a9)
plt.subplot(2,1,1)
```

```
plt.title('cos and sin')
plt.plot(a9,b9_sin)
plt.xlabel('x')
plt.ylabel('y')
plt.subplot(2,1,2)
plt.plot(a9,b9_cos)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
#10--图像相关
image=imread('zx.jpg')
plt.imshow(image)
```