NumPy 简介

NumPy 简介

NumPy(Numerical Python) 是Python语言的一个扩展程序库，支持大量的多维度数组与矩阵运算，主要用来计算、处理一维或多维数组, 此外也针对数组运算提供大量的数学函数库。 NumPy的前身Numeric最早是由Jim Hugunin与其它协作者共同开发，2005年，Travis Oliphant在Numeric中结合了另一个同性质的程序库Numarray的特色，并加入了其它扩展而开发了NumPy。 NumPy为开放源代码并且由许多协作者共同维护开发。

NumPy 是什么

NumPy是使用Python进行科学计算的基础软件包。 除其他外，它包括：

- 功能强大的N维数组对象。
- 集成 C/C+和Fortran 代码的工具。
- 强大的线性代数、傅立叶变换和随机数功能。

NumPy 特点：

ndarray

NumPy 最重要的一个特点是其 N 维数组对象 ndarray，它是一系列**同类型**数据的集合，以0下标为开始进行集合中元素的索引。 ndarray 对象是用于存放同类型元素的**多维数组**。 ndarray 中的每个元素在内存中都有**相同**存储大小的区域。

切片和索引

ndarray对象的内容可以通过索引或切片来访问和修改，与Python中list的切片操作一样。 ndarray数组可以基于0 - n的下标进行索引，切片对象可以通过内置的slice函数，并设置start,stop及step参数进行，从原数组中切割出一个新数组。

NumPy 功能：

- 创建n维数组(矩阵)
- 对数组进行函数运算，使用函数计算十分快速，节省了大量的时间，且不需要编写循环，十分方便
- 数值积分、线性代数运算、傅里叶变换
- ndarray快速、节省空间的多维数组，提供数组化的算术运算和高级的广播功能。

NumPy 对象

- NumPy中的核心对象是ndarray
- ndarray可以看成数组，存放同类元素
- NumPy里面所有的函数都是围绕ndarray展开的

ndarray 内部由以下内容组成：

- 一个指向数据(内存或内存映射文件中的一块数据)的指针。
- 数据类型(dtype)，描述在数组中的固定大小值的格子。
- 一个表示数组形状(shape)的元组，表示各维度大小的元组。形状为(row×col)

NumPy 数据类型

numpy 支持的数据类型比 Python 内置的类型要多很多，基本上可以和C语言的数据类型对应上主要包括 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32、float64

NumPy应用场景：

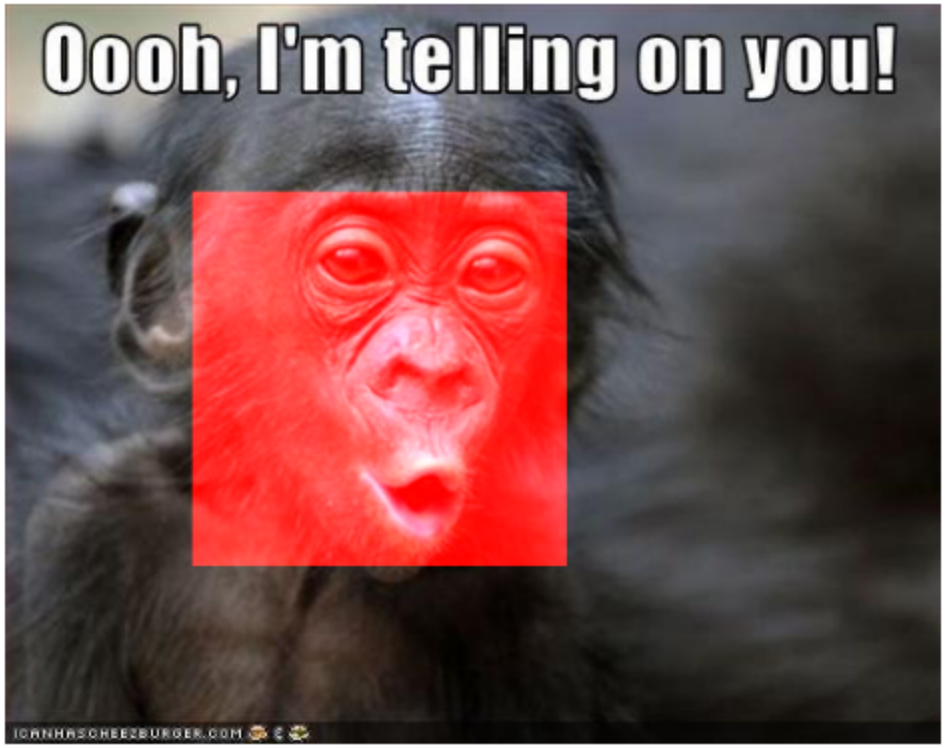
NumPy 通常与 SciPy（Python科学计算库）和 Matplotlib（Python绘图库）等软件包组合使用，这种组合方式被用来广泛地代替 MatLab 的使用。

MatLab 是一款强大的数学计算软件，广泛应用在数据分析、电子通信、深度学习、图像处理、机器视觉、量化金融等领域，但近些年随着 Python 语言的迅猛发展，Python 被看作是一种更适合代替 MatLab 的编程语言。用户可以使用 NumPy、SciPy 与 Matplotlib 等 Python 工具包搭建科学计算环境，比如 Anaconda 就是是一个开源的 Python 发行版本，它包含了 Python 、NumPy 等 180 多个科学包及其依赖项。

https://docs.scipy.org/doc/scipy/tutorial/index.html#user-guide

In [7]:
```
# Import required modules
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
# Load image
image = Image.open('image/funny.jpg')
#print(type(image))
# Convert image to array
image_arr = np.array(image)
#print(type(image_arr))
#print(image_arr.shape)
# crop:
#image_arr = image_arr[100:300, 100:300, :]
# mask:
image_arr[100:300, 100:300, 0] = 255

fig, ax = plt.subplots(num="funny image")
ax.imshow(image_arr, cmap='gray')
ax.axis('off')
plt.show()
```



In [14]:
```
image = Image.open('image/test2.jpg')
# Convert image to array
image_arr = np.array(image)
image_arr_crop = image_arr[550:750, 1220:1350, :]
fig, ax = plt.subplots(num="funny image")
ax.imshow(image_arr_crop)
ax.axis('off')
plt.show()
```

In [19]:
```
import cv2

# Convert to graycsale
img_gray = cv2.cvtColor(image_arr, cv2.COLOR_BGR2GRAY)
img_gray = img_gray[::3, ::3]
# Blur the image for better edge detection
```

```
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)

sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5) # Sobel Edge De
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5) # Sobel Edge De
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5) # Combined X a
# Display Sobel Edge Detection Images
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)
cv2.imshow('Sobel Y', sobely)
cv2.waitKey(0)
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
cv2.destroyAllWindows()

sobelx[sobelx<=128] = 0
sobelx[sobelx>128] = 1

sobely[sobely<=128] = 0
sobely[sobely>128] = 1

sobelxy[sobelxy<=128] = 0
sobelxy[sobelxy>128] = 1
```

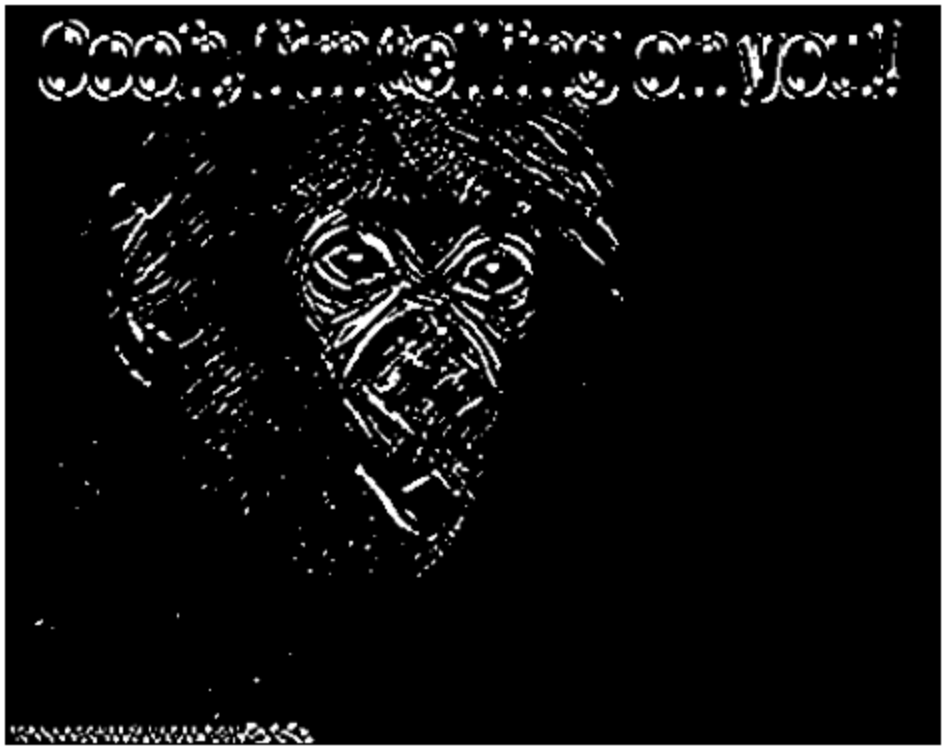
In [20]:

```
fig, ax = plt.subplots(num="funny image")
ax.imshow(sobelx, cmap='gray')
#ax.imshow(image_arr_crop)
ax.axis('off')
plt.show()
```

In [60]:

```
fig, ax = plt.subplots(num="funny image")
ax.imshow(sobely, cmap='gray')
#ax.imshow(image_arr_crop)
ax.axis('off')
plt.show()
```

In [62]:
```
fig, ax = plt.subplots(num="funny image")
ax.imshow(sobelxy, cmap='gray')
#ax.imshow(image_arr_crop)
ax.axis('off')
plt.show()
```



In [63]:
```
# Canny Edge Detection
edges = cv2.Canny(image=img_blur, threshold1=50, threshold2=128)

fig, ax = plt.subplots(num="funny image")
ax.imshow(edges, cmap='gray')
#ax.imshow(image_arr_crop)
ax.axis('off')
plt.show()
```

NumPy 创建数组

可以使用底层 ndarray 构造器来创建外

In [11]:
```
import numpy as np

x = np.array([1,2,3], dtype='int32')
print(x)
print(x.ndim)      # 数据的维度
print(x.shape)      # 数据的形状
print(x.itemsize)   # 每个元素占用的内存大小（bytes）
```

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

In [8]:
```
x2 = np.array([[9.0,8.0,7.0],[6.0,5.0,4.0]])
print(x2)

#查看 numpy 维度
print('维度：',x2.ndim)
#查看 numpy 形状
print('形状：',x2.shape)
#查看 numpy 数据类型
print('数据类型：',x2.dtype)
#查看 numpy 数据个数
print('元素大小：',x2.itemsize)
#查看 numpy 数据规模
print('数据规模：',x2.size)
#查看 numpy 使用空间
print('使用空间：',x2.nbytes)
```

```
[[9. 8. 7.]
 [6. 5. 4.]]
维度： 2
形状： (2, 3)
数据类型： float64
元素大小： 8
```

```
数据规模：  6
使用空间：  48
```

获取数据

In [22]:
```
a = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

In [14]:
```
# 获取特定的元素 [r, c]
a[1, 5]
```

Out[14]:
```
13
```

In [15]:
```
# 获取特定的行
a[0, :]
```

Out[15]:
```
array([1, 2, 3, 4, 5, 6, 7])
```

In [16]:
```
# 获取特定的列
a[:, 2]
```

Out[16]:
```
array([ 3, 10])
```

In [12]:
```
a[0,1:]

# 更有趣的元素获取 [起始索引:结束索引:步数]
a[0,1:-1:2]
```

Out[12]:
```
array([2, 4, 6])
```

In [23]:
```
print(a)
# 改变元素的值
a[1,5] = 20

print(a)
# 改变一些元素的值
a[:,2] = [1,2]
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 20 14]]
[[ 1  2  1  4  5  6  7]
 [ 8  9  2 11 12 20 14]]
```

3-d example

In [24]:
```
b = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(b.shape)
```

```
(2, 2, 2)
```

In [16]:
```
# 获取特定元素 (从外至内)
b[0,1,1]
```

Out[16]:
```
4
```

In [18]:
```
# 替换元素
b[:,1,:] = [[8,8],[9,9]]
print(b)
```

```
[[[1 2]
  [8 8]]

 [[5 6]
  [9 9]]]
```

创建ndarray

In [40]:
```
# 创建指定大小的数组，数组元素以 0 来填充：
np.zeros((2,3))
```

Out[40]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [42]:
```
# 创建指定大小的数组，数组元素以 1 来填充：
np.ones((2,2,2), dtype='int32')
```

Out[42]:
```
array([[[1, 1],
        [1, 1]],

       [[1, 1],
        [1, 1]]])
```

In [43]:
```
# Any other number
np.full((2,2), 99)
```

Out[43]:
```
array([[99, 99],
       [99, 99]])
```

In [49]:
```
# Any other number (full_like)
np.full_like(a, 4)
```

Out[49]:
```
array([[4, 4, 4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4, 4, 4]])
```

In [56]:
```
# Random decimal numbers
np.random.rand(4,2)
```

Out[56]:
```
array([[0.07805642, 0.53385716],
       [0.02494273, 0.99955252],
       [0.48588042, 0.91247437],
       [0.27779213, 0.16597751]])
```

In [73]:
```
# Random Integer values
np.random.randint(-4,8, size=(3,3))
```

Out[73]:
```
array([[-2, -4, -4],
       [ 6,  6,  3],
       [ 3,  2,  2]])
```

In [76]:
```
# The identity matrix
np.identity(5)
```

Out[76]:
```
array([[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.]])
```

In [82]:
```
# Repeat an array
arr = np.array([[1,2,3]])
r1 = np.repeat(arr,3, axis=0)
print(r1)
```

```
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

In [44]:
```
output = np.ones((5,5))
print(output)

z = np.zeros((3,3))
z[1,1] = 9
print(z)

output[1:-1,1:-1] = z
print(output)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[0. 0. 0.]
 [0. 9. 0.]
 [0. 0. 0.]]
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 9. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

复制**numpy array!!!**

In [26]:
```
a = np.array([1,2,3])
b = a.copy()
b[0] = 100

print(a)
print(b)
```

```
[1 2 3]
[100   2   3]
```

数学计算

In [56]:
```
a = np.array([1,2,3,4])
print(a)

print(a + 2)
print(a - 2)
print(a * 2)
print(a / 2)
print(np.cos(a))
```

```
[1 2 3 4]
[3 4 5 6]
[-1  0  1  2]
[2 4 6 8]
[0.5 1.  1.5 2. ]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362]
```

In [55]:
```
a = np.array([1,2,3,4])
b = np.array([1,0,1,0])
a + b
```

Out[55]:
```
array([2, 2, 4, 4])
```

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。

In [57]:
```
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print (c)
```

```
[ 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：

In [58]:
```
import numpy as np

a = np.array([[ 0, 0, 0],
           [10,10,10],
           [20,20,20],
           [30,30,30]])
b = np.array([0,1,2])
print(a + b)
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

下面的图片展示了数组 b 如何通过广播来与数组 a 兼容。

广播的规则:

- 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。
- 输出数组的形状是输入数组形状的各个维度上的最大值。
- 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。
- 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。

In [117…
```
# For a lot more (https://docs.scipy.org/doc/numpy/reference/routines.math.html)
```

NumPy 迭代数组:

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。

In [88]:
```
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, order='C'):
    print (x, end=", " )
print ('\n')
```

```
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）。

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.T, order='C'):C order`，即是行序优先；

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）。

In [92]:
```
import numpy as np

a = np.arange(0,60,5)
a = a.reshape(3,4)
print  ('第一个数组为：')
print (a)
print  ('\n')
print ('第二个数组为：')
b = np.array([1,  2,  3,  4], dtype =  int)
print (b)
print ('\n')
print ('修改后的数组为：')
for x,y in np.nditer([a,b]):
    print ("%d:%d"  %  (x,y), end=", " )
```

```
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,
```

修改数组形状

- reshape, 不改变数据的条件下修改形状

- flat, 数组元素迭代器
- 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
- ravel, 返回展开数组

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。

In [27]:
```
import numpy as np

a = np.arange(8)
print ('原始数组：')
print (a)
print ('\n')

b = a.reshape(4,2)
print ('修改后的数组：')
print (b)
```

```
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]]
```

numpy.ndarray.flat

numpy.ndarray.flat 是一个数组元素迭代器，实例如下:

In [96]:
```
import numpy as np

a = np.arange(9).reshape(3,3)
print ('原始数组：')
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:
    print (element)
```

```
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
<built-in method flatten of numpy.ndarray object at 0x0000029B19B60DB0>
```

numpy.ndarray.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：

In [98]:
```
import numpy as np

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

print ('原数组：')
print (a)
print ('\n')
# 默认按行

print ('展开的数组：')
print (a.flatten())
print ('\n')

print ('以 F 风格顺序展开的数组：')
print (a.flatten(order = 'F'))
```

```
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

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。

该函数接收两个参数：

In [100…
```
import numpy as np

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

print ('原数组：')
print (a)
print ('\n')

print ('调用 ravel 函数之后：')
print (a.ravel())
print ('\n')

print ('以 F 风格顺序调用 ravel 函数之后：')
print (a.ravel(order = 'F'))
```

```
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]
```

连接数组

- concatenate，连接沿现有轴的数组序列
- stack，沿着新的轴加入一系列数组。
- hstack，水平堆叠序列中的数组（列方向）
- vstack，竖直堆叠序列中的数组（行方向）

In [139…

```
import numpy as np
a = np.array([[1,2],[3,4]])
print ('第一个数组：')
print (a)

b = np.array([[5,6],[7,8]])
print ('第二个数组：')
print (b)

# 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))
```

```
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：

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

In [140…

```
import numpy as np

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

print ('第一个数组：')
print (a)

b = np.array([[5,6],[7,8]])
print ('第二个数组：')
print (b)
```

```
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))
```

```
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。

In [105…

```
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')
```

```
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 数学函数

In [28]:
```
import numpy as np

a = np.array([0,30,45,60,90])
print ('不同角度的正弦值：')
# 通过乘 pi/180 转化为弧度
print (np.sin(a*np.pi/180))
print ('\n')
print ('数组中角度的余弦值：')
print (np.cos(a*np.pi/180))
print ('\n')
print ('数组中角度的正切值：')
print (np.tan(a*np.pi/180))
```

```
不同角度的正弦值：
[0.         0.5        0.70710678 0.8660254  1.        ]


数组中角度的余弦值：
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]


数组中角度的正切值：
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
```

舍入函数

In [109…
```
import numpy as np

a = np.array([1.0,5.55,  123,  0.567,  25.532])
print  ('原数组：')
print (a)
print ('\n')
print ('舍入后：')
print (np.around(a))
print (np.around(a, decimals =  1))
print (np.around(a, decimals =  -1))

# numpy.floor()  numpy.floor() 返回小于或者等于指定表达式的最大整数，即向下取整。
# numpy.ceil() 返回大于或者等于指定表达式的最小整数，即向上取整。
```

```
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 统计函数

In [29]:
```
import numpy as np

a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print ('我们的数组是：')
print (a)

print ('调用 amin() 函数：')
print (np.amin(a,1))

print ('再次调用 amin() 函数：')
print (np.amin(a,0))
```

```
print ('调用 amax() 函数：')
print (np.amax(a))

print ('再次调用 amax() 函数：')
print (np.amax(a, axis=0))
```

```
我们的数组是：
[[3 7 5]
 [8 4 3]
 [2 4 9]]
调用 amin() 函数：
[3 3 2]
再次调用 amin() 函数：
[2 4 3]
调用 amax() 函数：
9
再次调用 amax() 函数：
[8 7 9]
```

统计

In [34]:
```
stats = [[1,2],[3,4]]
print(np.max(stats))
print(np.max(stats, axis=1))
print(np.mean(stats, axis=1))
print(np.sum(stats, axis=1))
```

```
4
[2 4]
[1.5 3.5]
[3 7]
```

NumPy线性代数

numpy.dot()

按照矩阵的乘法规则，计算两个矩阵的点积运算结果。当输入一维数组时返回一个结果值，若输入的多维数组则同样返回一个多维数组结果。

In [121…
```
import numpy as np
A=[1,2,3]
B=[4,5,6]
print(np.dot(A,B))
```

```
32
```

In [2]:
```
import numpy as np
a = np.array([[100,200],
              [23,12]])
b = np.array([[10,20],
             [12,21]])
result = np.dot(a,b)
print(result)

# [[100*10+200*12,100*20+200*21]
# [23*10+12*12,23*20+12*21]]
```

```
[[3400 6200]
 [ 374  712]]
```

numpy.inner()

inner() 方法用于计算数组之间的内积。当计算的数组是一维数组时，它与 dot() 函数相同，若输入的是多维数组则两者存在不同。

inner() 函数的计算过程是 A 数组的每一行与 B 数组的每一行相乘再相加，如下所示：

In [125…
```
import numpy as np
A=[[1 ,10],
    [100,1000]]
B=[[1,2],
    [3,4]]
#inner函数
print(np.inner(A,B))
#dot函数
print(np.dot(A,B))
```

```
[[  21   43]
 [2100 4300]]
[[  31   42]
 [3100 4200]]
```

numpy.matmul()

该函数返回两个矩阵的乘积，假如两个矩阵的维度不一致，就会产生错误。

In [126…
```
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
b = np.array([[23,23,12],[2,1,2],[7,8,9]])
mul = np.matmul(a,b)
print(mul)
```

```
[[ 48  49  43]
 [144 145 112]
 [240 241 181]]
```

numpy.linalg.solve()

该函数用于求解线性矩阵方程组，并以矩阵的形式表示线性方程的解，如下所示：

```
3X  +  2 Y + Z =  10
 X + Y + Z = 6
 X + 2Y - Z = 2
```

首先将上述方程式转换为矩阵的表达形式：

方程系数矩阵 m：
```
3   2   1
1   1   1
1   2  -1
```
方程变量矩阵 x：
```
X
Y
Z
```
方程结果矩阵 n：
```
10
6
2
```

In [130…
```
import numpy as np
m = np.array([[3,2,1],[1,1,1],[1,2,-1]])
print ('数组 m：')
print (m)
print ('矩阵 n：')
n = np.array([[10],[6],[2]])
print (n)
print ('计算：')
x = np.linalg.solve(m,n)
print(x)
```

```
数组 m：
[[ 3  2  1]
 [ 1  1  1]
 [ 1  2 -1]]
矩阵 n：
[[10]
 [ 6]
 [ 2]]
计算：
[[1.]
 [2.]
 [3.]]
```

numpy.linalg.inv()

该函数用于计算矩阵的逆矩阵，逆矩阵与原矩阵相乘得到单位矩阵。示例如下：

In [133…
```
import numpy as np
a = np.array([[1,2],[3,4]])
print("原数组:",a)
b = np.linalg.inv(a)
print("求逆:",b)
```

```
原数组: [[1 2]
 [3 4]]
求逆: [[-2.   1. ]
 [ 1.5 -0.5]]
```

Load Data from File

In [5]:
```
filedata = np.genfromtxt('data.txt', delimiter=',')
filedata = filedata.astype('int32')
print(filedata)
```

```
[[  1  13  21  11 196  75   4   3  34   6   7   8   0   1   2   3   4   5]
 [  3  42  12  33 766  75   4  55   6   4   3   4   5   6   7   0  11  12]
 [  1  22  33  11 999  11   2   1  78   0   1   2   9   8   7   1  76  88]]
```

Boolean Masking and Advanced Indexing

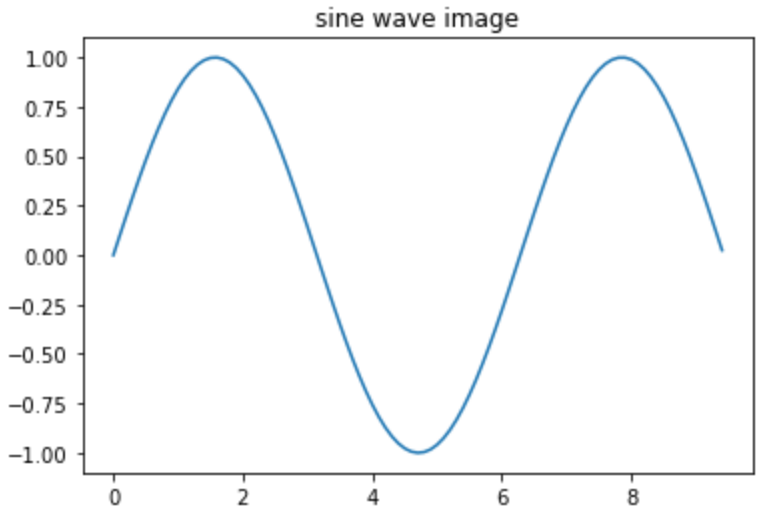
In [196…
```
(~((filedata > 50) & (filedata < 100)))
```

Out[196]:
```
array([[ True,  True,  True,  True,  True, False,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True, False,  True, False,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True, False,
         True,  True,  True,  True,  True,  True,  True, False, False]])
```

结合 Matplotlib 绘制正弦波图

```
import numpy as np
```

In [137…
```
import matplotlib.pyplot as plt
# 计算正弦曲线上的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave image")
# 使用matplotlib制图
plt.plot(x, y)
plt.show()
```



参考资料：

https://www.numpy.org.cn/user/quickstart.html

http://c.biancheng.net/numpy/what-is-numpy.html

In [ ]:
```
# the end
```