NumPy

NumPy 提供了两种基本的对象：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。

可以通过给array函数传递Python的序列对象创建数组，如果传递的是多层嵌套的序列，将创建多维数组

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

数组的元素类型可以通过dtype 属性获得，数组的大小可以shape属性获得，可以通过修改数组的shape属性，在保持数组元素个数不变的情况下，改变数组每个轴的长度。

dtype('int32')

c.shape

c.shape = 4,3

NumPy提供了很多专门用来创建数组的函数。

arange函数类似于python的range函数，通过指定开始值、终值和步长来创建一维数组，注意数组不包括终值。

例：np.arange(0,1,0.1)

linspace函数通过指定开始值、终值和元素个数来创建一维数组。

例：np.linspace(0, 1, 10) # 步长为1/9

logspace函数和linspace类似，创建等比数列。

例：np.logspace(0, 2, 20) #产生1(10^0)到100(10^2)、有20个元素的等比数列

zeros()、ones()、empty()可以创建指定形状和类型的数组。

例：np.zeros(4, np.float)

使用frombuffer, fromstring, fromfile，fromfunction等函数可以从字节序列、文件创建数组。

二维数组表示九九乘法表，输出的数组a中的每个元素a[i, j]都等于func2(i, j)：

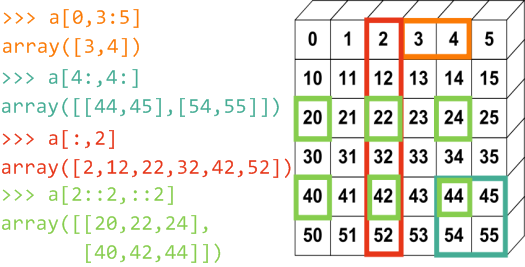
|  |
| --- |
| >>> def func2(i, j):  ... return (i+1) \* ( j+1)  ...  >>> a = np.fromfunction(func2, (9,9))  >>> a  array([[ 1., 2., 3., 4., 5., 6., 7., 8., 9.],  [ 2., 4., 6., 8., 10., 12., 14., 16., 18.],  [ 3., 6., 9., 12., 15., 18., 21., 24., 27.],  [ 4., 8., 12., 16., 20., 24., 28., 32., 36.],  [ 5., 10., 15., 20., 25., 30., 35., 40., 45.],  [ 6., 12., 18., 24., 30., 36., 42., 48., 54.],  [ 7., 14., 21., 28., 35., 42., 49., 56., 63.],  [ 8., 16., 24., 32., 40., 48., 56., 64., 72.],  [ 9., 18., 27., 36., 45., 54., 63., 72., 81.]]) |

数组元素的存取方法和Python的标准方法相同。和Python的列表序列不同，通过下标范围获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据空间。

|  |
| --- |
| >>> b = a[3:7] # 通过下标范围产生一个新的数组b，b和a共享同一块数据空间  >>> b  array([101, 4, 5, 6])  >>> b[2] = -10 # 将b的第2个元素修改为-10  >>> b  array([101, 4, -10, 6])  >>> a # a的第5个元素也被修改为10  array([ 0, 1, 100, 101, 4, -10, 6, 7, 8, 9]) |

多维数组的存取和一维数组类似，因为多维数组有多个轴，因此它的下标需要用多个值来表示，NumPy采用组元(tuple)作为数组的下标。如下图所示，a为一个6x6的数组，图中用颜色区分了各个下标以及其对应的选择区域。

组元不需要圆括号。虽然我们经常在Python中用圆括号将组元括起来，但是其实组元的语法定义只需要用逗号隔开即可，例如x,y=y,x 就是用组元交换变量值的一个例子。



from PIL import Image

im = Image.open('dog.jpg')

print(im)

print(im.format,im.size,im.mode)

im.show()

from PIL import Image

im = Image.open('dog.jpg')

print(im)

print(im.format,im.size,im.mode)

im.show() #显示图片

im.save('dog.BMP') #保存图片

###设置要拷贝的区域

##box = (100, 100, 500, 500)

### 将im表示的图片对象拷贝到region中，大小为(400\*400)像素。

### 这个region可以用来后续的操作(region其实就是一个Image对象)，

### box变量是一个四元组(左，上，右，下)。

##region = im.crop(box)

##

### 从字面上就可以看出，先把region中的Image反转180度，然后再放回到region中。

##region = region.transpose(Image.ROTATE\_90)

###粘贴box大小的region到原先的图片对象中。

##im.paste(region, box)

##im.save('dogA.jpg')

##r,g,b = im.split()#分割成三个通道

##im = Image.merge("RGB", (b, g, r)) #将b,r两个通道进行翻转。

##im.save('rgb.jpg')

#几何转变

out = im.resize( ( 128,128))

out = im.rotate(45)

out.show()

# 左右翻转

out = im.transpose(Image.FLIP\_LEFT\_RIGHT)

# 上下反向

out = im.transpose(Image.FLIP\_TOP\_BOTTOM)

out = im.transpose(Image.ROTATE\_90)

out.show()

#模式转换

# 黑白

am = im.convert('L')

am.show()

#图像增强

#1.滤镜

out = im.filter(ImageFilter.DETAIL) #detail滤镜

out.show()

#2.直接操作像素点

out = im.point(lambda i : i \*1.2) #每个像素点亮度增20

out.show()