Table of Contents

In [1]:
```
#全部行都能输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import numpy as np
```

1 Numpy 是什么

- 简单来说，Numpy 是 Python 的一个科学计算包，包含了多维数组以及多维数组的操作。
- Numpy 的核心是 ndarray 对象，这个对象封装了同质数据类型的n维数组。
- 起名 ndarray 的原因就是因为是 n-dimension-array 的简写。

2 与python原生array区别

- NumPy 数组在创建时有固定的大小，更改ndarray的大小将创建一个新的数组并删除原始数据，不同于Python列表（可以动态增长）。
- **NumPy 数组中的元素都需要具有相同的数据类型。**
- 数组的元素如果也是数组（可以是 Python 的原生 array，也可以是 ndarray）的情况下，则构成了多维数组。NumPy 数组便于对大量数据进行高级数学和其他类型的操作。通常，这样的操作比使用Python的内置序列可能更有效和更少的代码执行。
- 越来越多的科学和数学的基于Python的包使用NumPy数组，所以需要学会 Numpy 的使用。

3 构建ndarray

- Numpy 中最重要的一个对象就是 ndarray。
- ndarray中的每个元素在内存中使用相同大小的块。 ndarray中的每个元素是数据类型对象的对象(称为dtype)。
- 从Python列表创建数组

In [2]:
```
a= np.array([2, 3, 5])
a
```

Out[2]:
```
array([2, 3, 5])
```

In [3]:
```
a= np.array([[2, 3, 5], [4, 6, 8], [3, 5, 66]])
a
```

Out[3]:
```
array([[ 2,  3,  5],
       [ 4,  6,  8],
       [ 3,  5, 66]])
```

In [ ]:
```
```

4 数据类型

Numpy 中的数组比 Python 原生中的数组（只支持整数类型与浮点类型）强大的一点就是它支持更多的数据类型。

4.1 基本数据类型

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，其中部分类型对应为 Python 内置的类型。

数据类型	描述
bool_	布尔（True或False），存储为一个字节
int_	默认整数类型（与C long相同；通常为int64或int32）
intc	与C int（通常为int32或int64）相同
intp	用于索引的整数（与C ssize_t相同；通常为int32或int64）
int8	字节（-128到127）
int16	整数（-32768到32767）
int32	整数（-2147483648至2147483647）
int64	整数（-9223372036854775808至9223372036854775807）
uint8	无符号整数（0到255）
uint16	无符号整数（0到65535）
uint32	无符号整数（0至4294967295）
uint64	无符号整数（0至18446744073709551615）
float_	float64的简写。
float16	半精度浮点：符号位，5位指数，10位尾数
float32	单精度浮点：符号位，8位指数，23位尾数

数据类型	描述
float64	双精度浮点：符号位，11位指数，52位尾数
complex_	complex128的简写。
complex64	复数，由两个32位浮点（实数和虚数分量）
complex128	复数，由两个64位浮点（实数和虚数分量）

- 请记住，不同于 Python 列表，NumPy 要求数组必须包含同一类型的数据。
- **如果类型不匹配，NumPy 将会向上转换（如果可行）。**
- 字符串>浮点数>整数

这里整型被转换为浮点型：

In [4]:
```
a=np.array([3.14, 4, 2, 3])
b=np.array([3.14, "4", 2, 3])

a
b
```

Out[4]:
```
array([3.14, 4.  , 2.  , 3.  ])
```

Out[4]:
```
array(['3.14', '4', '2', '3'], dtype='<U32')
```

但我们也可以强行指定类型，将浮点型被转换成整型

In [9]:
```
a = np.array([3.14, 4, 2, 4], dtype='int32')
a
```

Out[9]:
```
array([3, 4, 2, 4])
```

In [12]:
```
a = np.array([3.14, 4, 2, 4], dtype='<U3')
a
```

Out[12]:
```
array(['3.1', '4', '2', '4'], dtype='<U3')
```

8位，32位是什么意思？

- 是指二进制的存储的长度，比如32位，能存储2的32次幂的位数
- dtype='<U3'的意思是：字符串最大字节是4

4.2 ndarray.dtype

ndarray.dtype用来显示当前ndarray对象的数据属于什么类型：

In [27]:
```
a=np.array([3, 4, 2, 3])
b=np.array([3.14,4.5, 2.8, 3,2])
c=np.array([3.14, 4, 2, 4], dtype='<U3')

a.dtype
b.dtype
c.dtype
```

Out[27]: dtype('int32')

Out[27]: dtype('float64')

Out[27]: dtype('<U3')

4.3 类型转换np.astype()

- 要转换数组的类型，请使用.astype()方法（首选）或类型本身作为函数。
- 转换类型之后，ndarray的id不会变。

In [14]:
```
a=np.array([3, 4, 2, 4], dtype="int64")
a
id(a)
```

Out[14]: array([3, 4, 2, 4], dtype=int64)

Out[14]: 1869882205488

- 第一种方式： np.astype("数据类型名")
- 第二种方式： np.astype(np.数据类型名)

In [15]:
```
a
a.astype(bool)   #这里返回的是一个新的对象，并没有改变a
```

Out[15]: array([3, 4, 2, 4], dtype=int64)

Out[15]: array([True, True, True, True])

In [16]:
```
id(a)
```

Out[16]: 1869882205488

5 常用的数组

5.1 np.arange()

用来创建一个线性序列的数组，在给定间隔内返回均匀间隔的值。

- arange([start,] stop[, step,], dtype=None)

In [18]:
```
np.array(range(20))

np.arange(20)    #不包括结束值20
np.arange(0, 20,5)
```

Out[18]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

Out[18]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

Out[18]:
```
array([ 0,  5, 10, 15])
```

5.2 np.linspace()

np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

- 在指定的间隔内返回均匀间隔的数字，用作相同间隔采样。
- start：标量,序列的起始值。
- stop：标量,除非"endpoint"设置为False，否则为序列的结束值。
- num：int，可选。要生成的样本数。默认值为50.必须为非负数。
- endpoint：bool，可选。如果为True，则“stop”是最后一个样本。否则，它不包括在内。默认为True。
- retstep：bool，可选如果为True，则返回（"samples"，"step"），其中"step"是间距样本之间。
- dtype：dtype，可选。输出数组的类型。如果没有给出"dtype"，推断数据从其他输入参数中键入。

In [19]:
```
np.linspace(0, 20)  #num默认值为50
```

Out[19]:
```
array([ 0.        ,  0.40816327,  0.81632653,  1.2244898 ,  1.63265306,
        2.04081633,  2.44897959,  2.85714286,  3.26530612,  3.67346939,
        4.08163265,  4.48979592,  4.89795918,  5.30612245,  5.71428571,
        6.12244898,  6.53061224,  6.93877551,  7.34693878,  7.75510204,
        8.16326531,  8.57142857,  8.97959184,  9.3877551 ,  9.79591837,
       10.20408163, 10.6122449 , 11.02040816, 11.42857143, 11.83673469,
       12.24489796, 12.65306122, 13.06122449, 13.46938776, 13.87755102,
       14.28571429, 14.69387755, 15.10204082, 15.51020408, 15.91836735,
       16.32653061, 16.73469388, 17.14285714, 17.55102041, 17.95918367,
       18.36734694, 18.7755102 , 19.18367347, 19.59183673, 20.        ])
```

In [28]:
```
np.linspace(0, 20, 2, endpoint=True)
np.linspace(0, 20, 2, endpoint=False)
```

Out[28]:
```
array([ 0., 20.])
```

Out[28]:
```
array([ 0., 10.])
```

In [34]:
```
np.linspace(0, 32, 4, endpoint=True)
np.linspace(0, 32, 4, endpoint=False)
```

Out[34]:
```
array([ 0.        , 10.66666667, 21.33333333, 32.        ])
```

Out[34]:
```
array([ 0.,  8., 16., 24.])
```

5.3 np.zeros()

第一个参数输入数组的形状

In [39]:
```
np.zeros(10) # 默认float型
np.zeros((3, 2))
```

Out[39]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Out[39]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

所谓的形状究竟是什么?

In [38]:
```
np.zeros((3,3,2))
```

Out[38]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

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

       [[0., 0.],
        [0., 0.],
        [0., 0.]]])
```

注意区分这三种情况:

In [41]:
```
np.zeros((3,1))
```

Out[41]:
```
array([[0.],
       [0.],
       [0.]])
```

In [45]:
```
np.zeros((1,3))
```

Out[45]:
```
array([[0., 0., 0.]])
```

思考，上面是几维数组? 下面呢?

In [46]:
```
np.zeros((3,))
```

Out[46]:
```
array([0., 0., 0.])
```

In [56]:
```
np.zeros(3)
```

Out[56]:
```
array([0., 0., 0.])
```

其实上面生成的是一维数组，和下面的写法生成的效果是一样的:

In [47]:
```
np.array([0,0,0],dtype="float64")
```

Out[47]:
```
array([0., 0., 0.])
```

5.4 np.ones()

In [31]:
```
np.ones(10) # float型
np.ones(10, dtype="int32")
np.ones((3, 3), dtype="int32")
```

Out[31]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Out[31]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Out[31]:
```
array([[[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, 1, 1]]])
```

5.5 np.eye()

返回一个二维数组，其中对角线为1，零点为零的二维数组。（单位矩阵）

In [31]:
```
np.eye(3)
```

Out[31]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [36]:
```
np.eye(3,3)
np.eye(3,2)
```

Out[36]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Out[36]:
```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```

In [37]:
```
np.eye(3,k=1, dtype=int)  #k=1时，斜角线会右移一行
```

Out[37]:
```
array([[0, 1, 0],
       [0, 0, 1],
       [0, 0, 0]])
```

5.6 np.empty()

- empty(shape, dtype=float)
- 返回给定形状和类型的新数组。
- 数组的值是内存空间中的任意值。
- np.empty()与np.zeros()不同，不会将数组值设置为零，因此可能会略微加快。

In [48]:
```
np.empty(0)
```

Out[48]:
```
array([], dtype=float64)
```

In [49]:
```
np.empty(5)
```

Out[49]:
```
array([2.12199579e-314, 2.12199579e-314, 2.12199579e-314, 2.12199579e-314,
       2.12199579e-314])
```

In [50]:
```
np.empty([3,3])
```

Out[50]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

5.7 np.diag()

- 提取对角线或构造对角线阵列。
- np.diag(v，k = 0)
  - v：array_like
    - 如果'v'是二维数组，则返回其第k个对角线的副本。
    - 如果'v'是一维数组，则在'k'-th上返回一个带有'v'的二维数组对角线。
  - k：int，可选
    - 对角线有问题。默认值为0.对角线使用“k> 0”
    - 在主对角线上方，对于主要对角线下方的对角线，“k <0”对角线。

如果操作对象是二维数组，则np.diag()操作将取出对角线数值，作为新数组返回：

In [3]:
```
A=np.random.randint(1,10,(3,3))
A

np.diag(A)
```

Out[3]:
```
array([[7, 7, 9],
       [5, 2, 3],
       [3, 6, 1]])
```

Out[3]:
```
array([7, 2, 1])
```

如果操作对象是一维数组，则np.diag()操作将会将其作为对角线，生成一个新二维数组（对角矩阵）：

In [6]:
```
A=np.random.randint(1,10,3)
A

np.diag(A)
```

Out[6]:
```
array([8, 4, 2])
```

Out[6]:
```
array([[8, 0, 0],
       [0, 4, 0],
       [0, 0, 2]])
```

5.8 np.full()

返回给定形状和类型的新数组，填充 `fill_value`

- np.full（shape，fill_value，dtype = None）
  - shape：int或int的序列新数组的形状，例如 （2,3） 或 2 。
  - fill_value：标量填充值。
  - dtype：数据类型，可选数组所需的数据类型默认值为"None"。

In [51]:
```
np.full((3, 5),3.14)
```

Out[51]:
```
array([[3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14]])
```

In [52]:
```
np.full((2, 2), [(1, 2), (3, 4)])  #第二个参数以指定嵌套兑现
```

Out[52]:
```
array([[1, 2],
       [3, 4]])
```

5.9 设置空值

np中缺失值用np.nan表示，其他ndarry对象与之运算的结果都为缺失值，运算结果数组的形状与参与运算的数组的形状 一致。

In [49]:
```
a=np.nan
a

a+np.full((3, 5),3.14)
```

Out[49]:
```
nan
```

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

5.10 随机数组

5.10.1 np.random.randint()

将随机整数从“低”（包括）返回到“高”（不包括）

- 语法：randint(low, high=None, size=None, dtype)
  - 使用方法一：将随机整数从“低”（包括）返回到“高”（不包括）——左闭右开。
  - 使用方法二：返回特定形状的随机整数。

In [50]:
```
np.random.randint(1,2)    #随机数包含1，不包括2

np.random.randint(1,2,(5,5))     #随机数包含1，不包含2
```

Out[50]:
```
1
```

Out[50]:
```
array([[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]])
```

注意，random库中的randint方法，生成随机数范围是是左闭右闭

In [51]:
```
import random
random.randint(1,2)    #随机数包含1，也包含2
```

Out[51]:
```
1
```

注意区分以下现象：

In [54]:
```
np.random.seed(10)
np.random.randint(1,10,3)
np.random.randint(1,10,(3,))
np.random.randint(1,10,(3,1))
np.random.randint(1,10,(1,3))
```

Out[54]:
```
array([5, 1, 2])
```

Out[54]:
```
array([1, 2, 9])
```

Out[54]:
```
array([[1],
       [9],
       [7]])
```

Out[54]:
```
array([[5, 4, 1]])
```

5.10.2 np.random.random()

- 语法：np.random.random(size=None)
- 返回随机浮点数，在半开区间[0.0,1.0)中。

In [52]:
```
np.random.random((2,3))     # uniform in [0.0, 1)
```

Out[52]:
```
array([[0.12345372, 0.5496263 , 0.70925739],
       [0.24490213, 0.93514463, 0.25110996]])
```

In [53]:
```
np.random.random((2))
```

Out[53]:
```
array([0.8088932 , 0.96961363])
```

5.10.3 np.random.randn()

- 语法：np.random.randn(形状)
- 从“标准正态”分布中返回一个样本（或样本）。

In [54]:
```
np.random.randn(2,2)    #均值为0，方差为1的标准正态分布
```

Out[54]:
```
array([[ 0.60266728,  2.09186513],
       [ 0.793269  , -2.96117602]])
```

5.10.4 np.random.normal()

- 语法：normal（平均值，标准偏差，形状）
- 作用：从正态分布中抽取随机样本。
- 如果平均值和标准差为0和1，或者不写这两个参数，就等同于np.random.randn()

In [56]:
```
array = np.random.normal(0, 1, (4, 5))  # 均值, 标准差, 形状
array
```

Out[56]:
```
array([[ 0.66570007,  0.85546326,  0.78599728,  0.12516354,  0.50456543],
       [-0.0474935 ,  1.116325  ,  1.21881036, -1.72530634,  0.74229336],
       [ 0.57693705,  0.69629254,  0.69727795,  0.2397621 , -0.13664166],
       [ 0.33552219,  0.25786482, -0.84018127, -0.91523743, -0.01437391]])
```

In [60]:
```
array = np.random.randn(4, 5)    #标准正态分布，作用和上面的一样
array
```

Out[60]:
```
array([[-0.13623575,  0.17637307,  0.31085074,  1.72937588, -0.24066194],
       [-1.02735202,  0.42401507,  1.40862087,  0.43999202, -0.42823439],
       [-0.31201268, -0.56888339, -1.58494101,  1.05535316, -1.92657911],
       [ 0.69858388, -0.74620143, -0.15662666, -0.19363594,  1.13912535]])
```

5.10.5 np.random.choice()

从给定的1-D阵列生成随机样本

- choice(a, size=None, replace=True, p=None)
  - a：1-D数组或int
    - 如果是ndarray，则从其元素生成随机样本。
    - 如果是int，则生成随机样本，就像a是np.arange（a）
  - size：int或int的元组，可选
    - 输出形状。如果给定的形状是例如“（m，n，k）”，那么绘制了 m * n * k 样本。默认值为None，在这种情况下为a返回单个值。
  - replace：布尔值，可选样品是否有替代品
  - p：1-D数组，可选与a中每个条目相关的概率。如果没有给出样品，则假定均匀分布一个条目。

In [58]:
```
np.random.choice((2,3,5,8,4,5))
np.random.choice((2,3,5,8,4,5),(3,3))  #随机选取数，放在特定形状的矩阵
```

Out[58]:
```
5
```

Out[58]:
```
array([[5, 8, 8],
       [8, 2, 8],
       [5, 4, 2]])
```

In [62]:
```
np.random.choice(range(8),(3,3))
```

Out[62]:
```
array([[2, 4, 6],
       [2, 0, 7],
       [5, 0, 7]])
```

In [65]:
```
np.random.choice(range(8),(3,3),replace=False)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-65-b1a972a8ef1a> in <module>
----> 1 np.random.choice(range(8),(3,3),replace=False)

mtrand.pyx in mtrand.RandomState.choice()

ValueError: Cannot take a larger sample than population when 'replace=False'
```

上面为什么报错?

In [61]:
```
np.random.choice((2,3,5,8,4,5),(3,3),replace=True,p=(0.1,0.1,0.1,0.1,0.5,0.1)) #p数组的值和为1
```

Out[61]:
```
array([[5, 4, 4],
       [3, 4, 3],
       [8, 4, 3]])
```

5.10.6 np.random.shuffle()

通过混洗其内容来就地修改序列。此功能仅沿a的第一轴洗牌。

- np.random.shuffle(x)
  - x：array_like（要洗牌的数组或列表）。

In [62]:
```
a=[7,2,3,1,5]
np.random.shuffle(a)    #shuffle有洗牌的意思
a
```

Out[62]:
```
[5, 3, 1, 7, 2]
```

In [63]:
```
arr = np.arange(9).reshape((3,3))
arr

np.random.shuffle(arr)
arr
```

Out[63]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Out[63]:
```
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

5.10.7 其他分布随机数

In [64]:
```
np.random.standard_t(10,(3,3))  #t分布（自由度，形状）
np.random.chisquare(2,(3,3))        #卡方分布(自由度，形状)
np.random.binomial(10,0.9,(10,10))  #伯努利分布（实验次数，概率，形状），生成数组中数字代表单轮10次
```

Out[64]:
```
array([[-2.06272318, -0.59788335, -0.97566785],
       [-0.21467088,  0.59084742,  0.32336346],
       [ 0.53094261,  0.3051783 ,  0.66767663]])
```

Out[64]:
```
array([[1.01208669, 0.13298307, 0.81808134],
       [5.50100056, 1.85738841, 2.00693025],
       [0.50933914, 0.32513772, 1.11475583]])
```

Out[64]:
```
array([[10,  8,  7,  7,  9,  9,  9,  9,  8,  9],
       [10, 10,  8,  8, 10, 10, 10, 10, 10,  9],
       [10,  9,  8,  9, 10, 10,  9,  8, 10, 10],
       [ 9,  8, 10,  9,  9,  7,  9,  9,  9,  7],
       [10,  9, 10, 10,  8, 10,  9, 10,  7, 10],
       [ 9,  9, 10,  9,  9, 10, 10,  9, 10,  9],
       [ 9, 10,  9,  9, 10, 10, 10,  8,  7,  7],
       [ 8,  8,  9, 10, 10,  9,  9,  8, 10,  9],
       [10, 10,  8,  9,  9,  9,  9,  9, 10,  9],
       [10, 10, 10, 10,  9,  8, 10,  9, 10, 10]])
```

6 习题

创建一个5个元素的数组，5个数均匀的分配到0到50之间，包括50：

In [100]:

Out[100]: array([0, 12, 25, 37, 50])

Out[100]: array([0. , 12.5, 25. , 37.5, 50.])