

Python程序设计

第八讲 科学计算、数据分析与可视化 NumPy



张 华

数据的表示

从一个数据到一组数据

3.14



3.1413 3.1404
3.1398 3.1401 3.1376
3.1349

一个数据

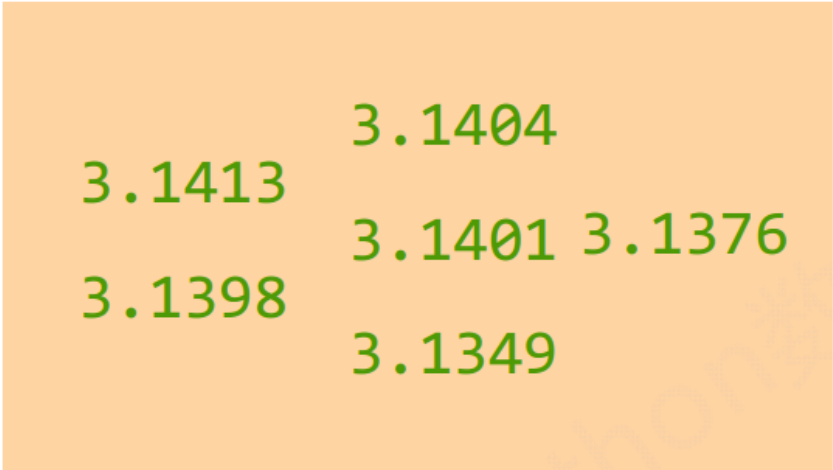
表达一个含义

一组数据

表达一个或多个含义

数据的表示

维度：一组数据的组织形式



3.1413	3.1404
	3.1401 3.1376
3.1398	3.1349

一组数据

→
3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

或

⇨
→
3.1398, 3.1349, 3.1376
↓
3.1413, 3.1404, 3.1401

数据的组织形式

数据的表示

一维数据

- 一维数据由对等关系的有序或无序数据构成，采用线性方式组织。

3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

对应列表、数组和集合等概念

区别

列表：数据类型可以不同

3.1413, 'pi', 3.1404, [3.1401, 3.1349], '3.1376'

数组：数据类型相同

3.1413, 3.1398, 3.1404, 3.1401, 3.1349, 3.1376

数据的表示

二维数据

- 二维数据由多个一维数据构成，是一维数据的组合形式。
 - 表格是典型的二维数据，其中，表头是二维数据的一部分。
- 可以用列表保存。

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分） ▾
1	清华大学	北京	94.0	100.0
2	北京大学	北京	81.2	96.1
3	浙江大学	浙江	77.8	87.2
4	上海交通大学	上海	77.5	89.4
5	复旦大学	上海	71.1	91.8
6	中国科学技术大学	安徽	65.9	91.9
7	南京大学	江苏	65.3	87.1
8	华中科技大学	湖北	63.0	80.6
9	中山大学	广东	62.7	81.1
10	哈尔滨工业大学	黑龙江	61.6	76.4

数据的表示

多维数据

- 多维数据由一维或二维数据在新维度上扩展形成。
- 可以用列表、字典来保存。

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分）
1	清华大学	北京市	95.9	100.0
2	北京大学	北京市	82.6	98.9
3	浙江大学	浙江省	80	88.8
4	上海交通大学	上海市	78.7	90.6
5	复旦大学	上海市	70.9	90.4
6	南京大学	江苏省	66.1	90.7
7	中国科学技术大学	安徽省	65.5	90.1
8	哈尔滨工业大学	黑龙江省	63.5	80.9
9	华中科技大学	湖北省	62.9	83.5
10	中山大学	广东省	62.1	81.8

时间维度

2016

排名	学校名称	省市	总分	指标得分
				生源质量（新生高考成绩得分）
1	清华大学	北京	94.0	100.0
2	北京大学	北京	81.2	96.1
3	浙江大学	浙江	77.8	87.2
4	上海交通大学	上海	77.5	89.4
5	复旦大学	上海	71.1	91.8
6	中国科学技术大学	安徽	65.9	91.9
7	南京大学	江苏	65.3	87.1
8	华中科技大学	湖北	63.0	80.6
9	中山大学	广东	62.7	81.1
10	哈尔滨工业大学	黑龙江	61.6	76.4

2017

NumPy

NumPy简介

✿ NumPy是Python中科学计算的基础软件包，包括：

- 一个强大的N维数组对象ndarray
- 广播函数
- 整合C/C++/Fortran代码的工具
- 线性代数、傅里叶变换、随机数生成等功能

✿ NumPy包的核心是ndarray对象。

✿ NumPy是SciPy、Pandas等数据处理或科学计算库的基础。

导入numpy

```
import numpy as np
```

NumPy

N维数组对象：ndarray

✿ Python已有列表类型，为什么需要一个数组对象(类型)?

➤ 例如：计算 A^2+B^3 ，其中，**A**和**B**是一维数组。

```
def pySum():  
    a = [0, 1, 2, 3, 4]  
    b = [9, 8, 7, 6, 5]  
    c = []
```

```
    for i in range(len(a)):  
        c.append(a[i]**2 + b[i]**3)
```

```
    return c
```

```
print(pySum())
```



```
import numpy as np
```

```
def npSum():  
    a = np.array([0, 1, 2, 3, 4])  
    b = np.array([9, 8, 7, 6, 5])
```

```
    c = a**2 + b**3
```

```
    return c
```

```
print(npSum())
```

- 数组对象可以去掉元素间运算所需的循环，使一维向量更像单个数据。
- 设置专门的数组对象，经过优化，可以提升这类应用的运算速度。

NumPy

N维数组对象：ndarray

- ✿ **ndarray**是一个多维数组对象，由两部分构成：
 - 实际的数据
 - 描述这些数据的元数据（数据维度、数据类型等）
- ✿ **ndarray**数组一般要求所有元素类型相同（同质），数组下标从**0**开始。
 - 科学计算中，一个维度所有数据的类型往往相同；
 - 数组对象采用相同的数据类型，有助于节省运算和存储空间。
- ✿ **Python**标准库中的**array**不支持多维数组，处理函数不丰富，不适合数值计算。

NumPy

N维数组对象：ndarray

ndarray对象的属性

属性	说明
.ndim	秩，即轴的数量或维度的数量
.shape	ndarray对象的尺度，对于矩阵，n行m列
.size	ndarray对象元素的个数，相当于.shape中n*m的值
.dtype	ndarray对象的元素类型
.itemsize	ndarray对象中每个元素的大小，以字节为单位

```
>>> a=np.array([1,2,3])
>>> type(c)
<class 'numpy.ndarray'>
>>> a.shape
(3,)
```

```
>>> b=np.array([[1,2],[3,4],[5,6]])
>>> b.shape
(3, 2)
>>> b.ndim
2
```

numpy的简单应用

创建数组

```
>>> np.array([1, 2, 3, 4, 5])           # 把列表转换为数组
array([1, 2, 3, 4, 5])

>>> np.array((1, 2, 3, 4, 5))           # 把元组转换成数组
array([1, 2, 3, 4, 5])

>>> np.array(range(5))                  # 把range对象转换成数组
array([0, 1, 2, 3, 4])

>>> np.array([[1, 2, 3], [4, 5, 6]])    # 二维数组
array([[1, 2, 3],
       [4, 5, 6]])

>>> np.arange(8)                        # 类似于内置函数range()
array([0, 1, 2, 3, 4, 5, 6, 7])

>>> np.arange(1, 10, 2)
array([1, 3, 5, 7, 9])
```

numpy的简单应用

创建数组

```
>>> np.random.randint(0, 50, 5)           # 随机数组, 5个[0,50)之间的整数
array([13, 47, 31, 26,  9])

>>> np.random.randint(0, 50, (3,5))      # 3行5列, 15个介于[0, 50)之间的整数
array([[34,  2, 33, 14, 40],
       [ 9,  5, 10, 27, 11],
       [26, 17, 10, 46, 30]])

>>> np.random.rand(10)                   # 10个小数[0, 1)
array([ 0.98139326,  0.35675498,  0.30580776,  0.30379627,
        0.19527425,  0.59159936,  0.31132305,  0.20219211,  0.20073821,
        0.02435331])

>>> np.random.standard_normal(5)         # 从标准正态分布中随机采样
array([ 2.82669067,  0.9773194 , -0.72595951, -0.11343254,
        0.74813065])
```

numpy的简单应用

创建数组

```
>>> np.linspace(0, 10, 11)           # 等差数组, 包含11个数
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

>>> np.zeros(3)                      # 全0一维数组
array([ 0.,  0.,  0.])

>>> np.ones(3)                      # 全1一维数组
array([ 1.,  1.,  1.])

>>> np.zeros((3,3))                 # 全0二维数组, 3行3列
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]

>>> np.identity(3)                  # 单位矩阵
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

numpy的简单应用

创建数组

```
>>> np.diag([1,2,3])           # 对角矩阵
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> np.diag([1,2,3,4])        # 对角矩阵
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

NumPy

N维数组对象：ndarray

ndarray的元素类型

数据类型	说明
bool	布尔类型，True或False
intc	与C语言中的int类型一致，一般是int32或int64
intp	用于索引的整数，与C语言中ssize_t一致，int32或int64
int8	字节长度的整数，取值： $[-128, 127]$
int16	16位长度的整数，取值： $[-32768, 32767]$
int32	32位长度的整数，取值： $[-2^{31}, 2^{31}-1]$
int64	64位长度的整数，取值： $[-2^{63}, 2^{63}-1]$

NumPy

N维数组对象：ndarray

ndarray的元素类型

数据类型	说明
uint8	8位无符号整数，取值： $[0, 255]$
uint16	16位无符号整数，取值： $[0, 65535]$
uint32	32位无符号整数，取值： $[0, 2^{32}-1]$
uint64	32位无符号整数，取值： $[0, 2^{64}-1]$
float16	16位半精度浮点数：1位符号位，5位指数，10位尾数
float32	32位半精度浮点数：1位符号位，8位指数，23位尾数
float64	64位半精度浮点数：1位符号位，11位指数，52位尾数

(符号)尾数*10^{指数}

NumPy

N维数组对象：ndarray

ndarray的元素类型

数据类型	说明
complex64	复数类型，实部和虚部都是32位浮点数
complex128	复数类型，实部和虚部都是64位浮点数

```
>>> x = np.array([1,2,3], dtype='float32')
>>> x
array([1., 2., 3.], dtype=float32)
```

numpy的简单应用

数组与数组的运算

```
>>> a = np.array((1, 2, 3))
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

>>> c = a * b                                     # 数组与数组相乘
>>> c                                              # a中的每个元素乘以b中的对应列元素
array([[ 1,  4,  9],
       [ 4, 10, 18],
       [ 7, 16, 27]])

>>> c / b                                         # 数组之间的除法运算
array([[ 1.,  2.,  3.],
       [ 1.,  2.,  3.],
       [ 1.,  2.,  3.]])

>>> c / a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])
```

numpy的简单应用

测试两个数组是否足够接近

```
>>> x = np.array([1, 2, 3, 4.001, 5])
>>> y = np.array([1, 1.999, 3, 4.01, 5.1])
>>> np.allclose(x, y)
False
>>> np.allclose(x, y, rtol=0.2)           # 设置相对误差参数
True
>>> np.allclose(x, y, atol=0.2)          # 设置绝对误差参数
True
```

numpy的简单应用

改变数组元素值

```
>>> x = np.arange(8)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> np.append(x, 8) # 返回新数组，增加元素
array([0, 1, 2, 3, 4, 5, 6, 7, 8])

>>> np.append(x, [9,10])
array([0, 1, 2, 3, 4, 5, 6, 7, 9, 10])
>>> x # 不影响原来的数组
array([0, 1, 2, 3, 4, 5, 6, 7])

>>> x[3] = 8 # 原地修改元素值
>>> x
array([0, 1, 2, 8, 4, 5, 6, 7])

>>> np.insert(x, 1, 8) # 返回新数组，插入元素
array([0, 8, 1, 2, 8, 4, 5, 6, 7])
```

numpy的简单应用

数组与数组的运算

```
>>> a + a  
array([2, 4, 6])
```

数组之间的加法运算

```
>>> a * a  
array([1, 4, 9])
```

数组之间的乘法运算

```
>>> a - a  
array([0, 0, 0])
```

数组之间的减法运算

```
>>> a / a  
array([ 1., 1., 1.])
```

数组之间的除法运算

numpy的简单应用

访问数组元素

```
>>> b = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b[0]                                # 第0行
array([1, 2, 3])
>>> b[0][0]                             # 第0行第0列的元素值
1
>>> b[0,2]                             # 第0行第2列的元素值
3
>>> b[[0,1]]                           # 第0行和第1行
array([[1, 2, 3],
       [4, 5, 6]])
>>> b[[0,1], [1,2]]                     #第0行第1列的元素和第1行第2列的元素
array([2, 6])
```

numpy的简单应用

改变数组大小

```
>>> a = np.arange(1, 11, 1)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> a.shape = 2, 5                                # 改为2行5列
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>> a.shape = 5, -1                                # -1表示自动计算，原地修改
>>> a
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10]])
>>> b = a.reshape(2, 5)                             # reshape()方法返回新数组
>>> b
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

numpy的简单应用

切片操作

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> a[::-1]                                # 反向切片
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

>>> a[::2]                                  # 隔一个取一个元素
array([0, 2, 4, 6, 8])

>>> a[:5]                                  # 前5个元素
array([0, 1, 2, 3, 4])
```


NumPy的应用案例

案例：图像处理

- ✱ NumPy

- ✱ PIL