程序设计基础
——NumPy

林大 经管学院 瞿华

NumPy

- Python本身的运算速度很慢，不适合用于需要大量数学计算的场合。为了解决这个问题，NumPy库应运而生。
- NumPy提供了：
  - 高效的多维数组对象实现
  - 在多维数组基础上，进行各种数学计算的工具函数。

目录

一、NumPy与一维数组

- 在对大量数据进行存储运算时，我们最基本的数据结构是列表。
- 在第四章中我们介绍了Map、Filter和Reduce运算。有了这三个基本运算，我们可以不需要写循环，就完成对列表中的数据的迭代访问。
- NumPy中提供了数组（ndarray），可以将其用作功能增强的列表：
  - 一系列以数组为基本参数，对数组中元素进行迭代访问运算的函数。
  - numpy本身用C实现，因此运算速度比python的for循环和map函数都快的多。
- 因此，我们首先可以把numpy看成是对列表和map、filter和reduce的进一步封装。

1.1 numpy数组创建

- 有多种方法可以创建一个ndarray对象。

```
import numpy as np

#基于列表创建ndarray
a1=np.array([1,2,3,4,5,6])
print(a1)

#创建一个包含5个1的ndarray
a2=np.ones(5)
print(a2)
```

注意需要先导入numpy包

如果numpy包未安装，需要先用pip安装numpy包

```
#创建一个包含5个0的ndarray
a3=np.zeros(5)
print(a3)

#创建一个包含20个从1一直到5，均匀分隔的数字的ndarray
a4=np.linspace(1,5,20)
print(a4)

#类似于python的range函数
a5=np.arange(5)
print(a5)

a6=np.arange(1,5)
print(a6)
```

1.2.1 基本数学运算

- numpy重载了pythong中所有常用的算术和比较运算符。

```
import numpy as np

a = np.array([20,30,40,50])
b = np.arange([0,1,2,3])
c=5
```

注意需要先导入numpy包

两个ndarray运算，结果是各自对应的元素逐一做相应运算

一个ndarray和数字运算，结果是数组中各元素逐一和该数字做相应运算

```
a-b
[20 29 38 47]
a+c
[25 35 45 55]
a*b
[  0  30  80 150]
a//c
[ 4  6  8 10]
a>b
[ True  True  True  True]
b%2==0
[ True False  True False]
```

1.2.2 逻辑运算

- 由于python语言不提供逻辑运算符（not/and/or)的运算符重载功能，ndarray无法通过逻辑运算符来进行对应元素的逐一逻辑运算。
- 因此，numpy中采用了变通的做法，当两个ndarray中的元素都是bool型时，可以位运算符（bitwise operation)来进行ndarray间对应元素之间逻辑运算
  - ~相当于not
  - &相当于and
  - |相当于or

1.2.2 逻辑运算(2)

```
import numpy as np

a = np.array([True,False,True,False])
b = np.arange([True,True,False,False])
```

```
~a
[False  True False  True]
a | b
[True  True  True False]
a & b
[True False False False]
True & a
[True False  True False]
a | True
[ True  True  True  True]
```

1.3 Map与Reduce

- numpy提供了多种以ndarray为参数的函数，包括常用的数学和逻辑运算函数。
- Map类型：
  - 各种数学和逻辑运算本身就是map操作
  - sin、cos、exp、log等常用数学函数
- Reduce类型：
  - max、min、argmax、argmin等
  - mean、cov、std、var、sum、prod等统计函数
  - all、any
  - searchsorted（二分查找）
- 排序：sort、argsort

1.4.1 Filter

- 在numpy中，可以直接用一个bool型ndarray做另外一个ndarray的下标，从而选取对应的元素：

```
import numpy as np

a=np.arange(10)
[0 1 2 3 4 5 6 7 8 9]
b=(a%2==0)
[ True False  True False  True False  True False  True False]

c=a[b]
[0 2 4 6 8]
```

找出0-9中所有的偶数

以b作为a的下标，从a中选取相应的元素。

例如，b[0]是True，那么a[0]就会被选中；b[1]是False，a[1]就不会被选中。

返回的结果是一个新的ndarray

1.4.2 Filter（二）

- 在numpy中，也可以使用另外一个整数型列表（或者数组）作为下标，来选取对应的元素；

```
import numpy as np

a=np.arange(10)
a=a*2
b=[2,5,3]

print(a)
print(a[b])
```

选取a中下标为2，5和3的元素

返回的结果是一个新的ndarray

1.5 应用举例

- 在numpy中，ndarray是最基本的程序元素，程序设计围绕它来进行。
- 我们以第2章循环中学过的两个统计模拟的例子为例，来看看如何用numpy编写程序。
- 例1-5-1.用投针法求圆周率。（原理分析和用循环实现）
- 程序分析：
  - 输入：实验次数n（在程序开头处定义的常量，不需要用户输入）
  - 输出：求得的圆周率pi
  - 处理：进行n次重复射击实验

1.5.1 蒙特卡罗法求圆周率(2)

- numpy分析：
  1. 产生包含n个随机落点横坐标的数组x
  2. 产生包含n个随机落点横坐标的数组y
  3. 根据x，y，生成bool数组in_circle
  4. 利用in_circle数组，filter数组x得到所有在圆内的点的横坐标数组x_in_circle。
  5. 用len函数计算x_in_circle中元素个数，即在圆内的点的个数count
  6. 知道n和count，就可以求出圆周率了
- 注意：1-5每一步都是在对数组进行迭代运算，相当于n个实验在同时进行!

1.5.1 蒙特卡罗法求圆周率(3)

```
import numpy as np
from numpy import random

n=10000000
r=0.5
cx,cy=0,0
x=random.uniform(-0.5,0.5,n)
y=random.uniform(-0.5,0.5,n)
in_circle = np.hypot(x-cx,y-cy)<= r
x_in_circle = x[in_circle]
count = len(x_in_circle)

area = count / n
pi = area / r / r
print(f"pi = {pi : .4f}")
```

1-5-1.投针法求圆周率.py

注意，我们使用numpy的random模块来生成随机数

1.5.2 产品盈利估计

- 例1-5-2：已知某新产品一年后上市，其盈利同时受产品销量、市场价格、利率影响。已知：
  - 项目总成本为200,000元。
  - 产品销量服从均值为30000，标准差为10000的正态分布；
  - 市场价格服从均值为6元，标准差为1元正态分布；
  - 利率服从均值为10%元，标准差为2%的正态分布；
  - 求该项目亏损的可能性有多大?
- 模拟分析：
  - 可以使用蒙特卡洛法，每次随机生成销量、价格和利率，用下式计算出项目盈利：盈利=销量*价格/(1+利率) -成本
  - 重复实验多次，统计其中亏损的次数占总次数的比例即可

1.5.2 产品盈利估计(2)

- 程序分析：
  - 输入：无
  - 初始条件（常量）：
    - 项目总成本cost，产品销量、市场价格和利率的随机参数（均值和标准差），实验次数n
  - 输出：
    - 亏损的次数占总次数的比例
  - 处理步骤：
    - 使用numpy进行模拟

1.5.2 产品盈利估计(3)

- numpy处理步骤分析：
  1. 随机产生n次实验销量数组sale
  2. 随机产生n次实验价格数组price
  3. 随机产生n次实验利率数组rate
  4. 计算n次实验利润数组profit
  5. 利用下标filter和len函数，计算亏损实验次数count
- 注意：1-5每一步都是在对数组进行迭代运算，相当于n个实验在同时进行!

1.5.2 产品盈利估计(4)

```
from numpy import random
import numpy as np

def get_normal(mean,dev,n):
    """
    产生n个非负，服从指定正态分布的随机数

    :param mean: 正态分布的均值
    :param dev: 正态分布的标准差
    :return: 产生的随机数数组
    """
    if mean<=0 or dev<=0 :
        raise  ValueError("Mean and dev must be positive! ")
    result = random.normal(mean,dev,n)
    return np.where(result>0,result,0)
```

1-5-2.盈利分析.py

numpy函数的where函数，作用是如果第一个参数数组的元素为True，则返回第二个参数数组中对应元素，否则返回第三个参数数组中对应元素

1.5.2 产品盈利估计(5)

```
cost = 200000
sale_mean, sale_dev = 30000,10000
price_mean, price_dev= 6,1
rate_mean, rate_dev = 0.1,0.02
n=5000000
sale = get_normal(sale_mean,sale_dev,n)
price = get_normal(price_mean,price_dev,n)
rate = get_normal(rate_mean,rate_dev,n)
profit = sale * price / (1+rate) - cost
count = len(profit[profit<=0])
print(f"亏损的比例为{count/n *100: .2f}%")
```

1-5-2.盈利分析.py（续）

二、NumPy与矩阵

- 矩阵及线性方程组，是我们在现实中解决涉及多个变量之间关系的数学文件的基础和核心工具。
- NumPy真正的核心功能，其实是对矩阵（以及更高维度的多维数组）和相关数学运算的支持。

2.1 多维数组的创建

- 在NumPy中，可以通过下列方法来创建矩阵（或者多维数组）
  - 使用多维列表初始化
  - 通过ndarray对象的reshape方法，将一维数组转换为多维数组
  - 使用zeros、ones或者identity函数直接创建多维数组。

2.1 多维数组的创建（2）

- 在NumPy中，可以通过下列方法来创建矩阵（或者多维数组）

```
lst=[[1,2,3],[4,5,6],[7,8,9]]
a1=np.array(lst)
print("a1")
print(a1)
```

使用多维列表初始化

```
a1=np.arange(1,10).reshape((3,3))
print("a1")
print(a1)
```

使用reshape转化

```
a1=np.zeros( (5,3) )
print("a1")
print(a1)
```

创建全零数组

```
a1=np.ones( (5,3) )
print("a1")
print(a1)
```

创建全一数组

```
a1=np.identity(5)
print("a1")
print(a1)
```

创建单位矩阵

2.2 矩阵计算应用

- 例2-2-1.投资环境评价。已知专家对A、B、C、D四个城市的投资环境评价如下图所示，请求各城市的评价总分，并进行比较。

投资环境评价打分表

投资环境	权重	A市	B市	C市	D市
消费状况	0.3	90	95	85	80
经济状况	0.4	95	100	85	80
自然环境	0.1	85	95	90	80
交通状况	0.2	80	85	90	95
总分					

2.2.1 应用一（2）

- 每个市的总分，等于其各单项的分数，乘以对应权重后，加在一起的结果。
- 例如，A市的总分等于90*03+95*0.4+85*0.1+80*.02=89.5
- 用矩阵A表示各市的单项分数，向量w表示权重，即

$$A = \begin{pmatrix} 90 & 95 & 85 & 80 \\ 95 & 100 & 85 & 80 \\ 85 & 95 & 90 & 80 \\ 80 & 85 & 90 & 95 \end{pmatrix} \qquad w = \begin{pmatrix} 0.3 \\ 0.4 \\ 0.1 \\ 0.2 \end{pmatrix}$$

- 则$A^T W$就是我们要求的结果

2.2.1 应用一（3）

```
import numpy as np
A=np.array([
    [90,95,85,80],
    [95,100,85,80],
    [85,95,90,80],
    [80,85,90,95]
])
W=np.array([0.3,0.4,0.1,0.2])
T=np.dot(A.T,W)
print(T)
```

使用np.dot计算两个矩阵的乘积

使用ndarray对象的.T属性获取矩阵的转置

2.2.2 矩阵计算应用二

- 例2-2-2.已知某厂生产A、B、C三种产品，每种产品的成本及每季度生产数量如下表所示。请求出每季度的分类成本和总成本。

单件产品的分类成本

成本（元）	产品A	产品B	产品C
原材料成本	0.1	0.3	0.15
劳动力成本	0.3	0.4	0.25
企业管理成本	0.3	0.2	0.15

各季度产品生产数量

	一季度	二季度	三季度	四季度
产品A	4000	4500	4500	4000
产品B	2000	2800	2400	2200
产品C	5800	6200	6000	6000

2.2.2 矩阵应用二（2）

- 使用矩阵，可以将两个表分别表示为两个矩阵C和P

$$C = \begin{pmatrix} 0.1 & 0.3 & 0.15 \\ 0.3 & 0.4 & 0.25 \\ 0.3 & 0.2 & 0.15 \end{pmatrix} \qquad P = \begin{pmatrix} 4000 & 4500 & 4500 & 4000 \\ 2000 & 2800 & 2400 & 2200 \\ 5800 & 6200 & 6000 & 6000 \end{pmatrix}$$

以二季度的劳动力成本为例，应等于产品A的劳动力成本0.3*4500+产品B的劳动力成本0.4*2800+产品C的原材料成本0.25*6200，即C的第二行和P的第三列对应相乘后相加

而这恰恰正是矩阵相乘的定义。因此，直接使用矩阵乘法即可计算出各季度的分类成本

2.2.2 矩阵应用二（3）

```
import numpy as np
C=np.array([
    [0.1,0.3,0.15],
    [0.3,0.4,0.25],
    [0.3,0.2,0.15]
])
P=np.array([
    [4000,4500,4500,4000],
    [2000,2800,2400,2200],
    [5800,6200,6000,6000]
])
T=np.dot(C,P)
print(T)
print(T.sum(axis = 0))
print(T.sum(axis = 1))
```

使用np.dot计算两个矩阵的乘积

使用sum函数，分别计算各行和各列的总和：

- 各行上的和，即各成本分类四季度成本的总和
- 各列上的和，即各季度的成本总和

2.3 线性方程组求解

- 矩阵最初和基本的用途是用于表示线性方程组。
- 例2-3-1：某位同学A准备通过节食和运动来减肥。在减肥期间，A决定每天只吃脱脂奶粉，大豆面粉和乳清三种食物，并通过一定量的慢跑消耗掉多余的营养。已知三种食品的营养含量，慢跑消耗的营养量，以及人体每日所需的营养量如下标所示。A准备每天慢跑20分钟，请计算A应怎样安排饮食营养要求（营养量必须正好，不能少，也不能多）？

营养	每100克食物所含营养（克）			每慢跑1分钟消耗营养量（g）	每日所需营养量（g）
	脱脂牛奶	大豆面粉	乳清		
蛋白质	36	51	13	0.2	33
碳水化合物	52	34	74	0.4	45
脂肪	10	7	1	0.3	3

2.3.1 应用1（二）

- 分析：令x_1、x_2、x_3分别为脱脂牛奶、大豆面粉、乳清的摄入量（单位为100克），y为慢跑的量（单位为1分钟），则应该满足下列线性方程组：

$$\begin{cases} 36x_1 + 51x_2 + 13x_3 - 0.2y = 33 \\ 52x_1 + 34x_2 + 74x_3 - 0.4y = 45 \\ 10x_1 + 7x_2 + x_3 - 0.3y = 3 \end{cases}$$

- 带入y=20（分钟），得到

$$\begin{cases} 36x_1 + 51x_2 + 13x_3 = 37 \\ 52x_1 + 34x_2 + 74x_3 = 53 \\ 10x_1 + 7x_2 + x_3 = 9 \end{cases}$$

2.3.1 应用1（三）

- 方程组的矩阵形式为$AX = b$，其中：

$$A = \begin{pmatrix} 36 & 51 & 13 \\ 52 & 34 & 74 \\ 10 & 7 & 1 \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad b = \begin{pmatrix} 37 \\ 53 \\ 9 \end{pmatrix}$$

```
import numpy as np
A = np.array([[36, 51, 13],
              [52, 34, 74],
              [10, 7, 1]])
b = np.array([37, 53, 9])
X = np.linalg.solve(A, b)

print(X)
print(np.dot(A, X))
```

使用NumPy求解，得到
x_1=79（克）
x_2=14.4（克）
x_3=9.5（克）

2.3.2 应用2（一）

- 例2-3-2：某公园在湖的周围设有甲、乙、丙三个游船出租点，游客可以在任何一处租船，也可以在任何一处还船。统计表明租船和还船的情况如下表所示：

		各处还船比例		
		甲	乙	丙
借船比例	甲	0.8	0.2	0
	乙	0.2	0	0.8
	丙	0.2	0.2	0.6

- 为了游客的安全，公园要建立一个检修站，请问检修站建立在哪个点最好?

2.3.2 应用2（二）

- 显然，游船检修站应该建在拥有船只最多的那个点。因此，我们实际需要回答，经过长时间的运营后，公园哪个点的船只最多？
- 由于船只会在各个点之间移动，不管最初甲乙丙各有多少船，我们可以假设，经过长时间的运营后，甲乙丙三地的船只数量达到稳态（即对于任意一个地点，单位时间内借出的船只数量和归还的船只数量相等）。
- 因此，假设x_1、x_2、x_3分别为各点单位时间内借出的船只数，M为公园全部船只总数，我们可以列出下面的方程组：

$$\begin{cases} 0.8x_1 + 0.2x_2 + 0.2x_3 = x_1 \\ 0.2x_1 + 0.2x_3 = x_2 \\ 0.8x_2 + 0.6x_3 = x_3 \\ x_1 + x_2 + x_3 = M \end{cases}$$

所有公式两边同时除以M，并且令$p_1 = \frac{x_1}{M}$，

2.3.2 应用2（三）

所有公式两边同时除以M，得到

$$\begin{cases} 0.8\dfrac{x_1}{M}+0.2\dfrac{x_2}{M}+0.2\dfrac{x_3}{M}=\dfrac{x_1}{M} \\ 0.2\dfrac{x_1}{M}+0.2\dfrac{x_3}{M}=\dfrac{x_2}{M} \\ 0.8\dfrac{x_2}{M}+0.6\dfrac{x_3}{M}=\dfrac{x_3}{M} \\ \dfrac{x_1}{M}+\dfrac{x_2}{M}+\dfrac{x_3}{M}=1 \end{cases}$$

令$p_1=\frac{x_1}{M}$，$p_2=\frac{x_2}{M}$，$p_3=\frac{x_3}{M}$，并合并同类项，得到

$$\begin{cases} -0.2p_1+0.2p_2+0.2p_3=0 \\ 0.2p_1-p_2+0.2p_3=0 \\ 0.8p_2-0.4p_3=0 \\ p_1+p_2+p_3=1 \end{cases}$$

2.3.2 应用2（四）

- 方程组的矩阵形式为$AX = b$，其中：

$$A = \begin{pmatrix} -0.2 & 0.2 & 0.2 \\ 0.2 & -1 & 0.2 \\ 0 & 0.8 & -0.4 \\ 1 & 1 & 1 \end{pmatrix} \quad X = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

系数矩阵A的秩为3，与未知数个数相同，因此有唯一解
但由于NumPy的solve函数要求A为方阵，因此无法使用solve求解

```
import numpy as np
A = np.array([[-0.2,0.2,0.2],
              [0.2, -1, 0.2],
              [0, 0.8, -0.4],
              [1,1,1]])
print(np.linalg.matrix_rank(A))
b = np.array([0,0,0,1])
X = np.linalg.lstsq(A, b,rcond=None)[0]
print(X)
```

改用最小二乘法lstsq函数求解，得到：
p_1=0.5
p_2=0.167
p_3=0.333
即甲点的船只数量最多，为全部船只数量的一半
因此应该在甲地建检修点

注：这是一个马尔可夫链。

三、Pandas

- Numpy主要用于数学和逻辑计算，同一个ndarray对象中一般各元素是同一类型的数据（如，都是数字类型，都是bool类型等）
- Pandas更进一步，提供了两个重要的数据结构：Series和DataFrame

3.1 Series

- Series可以看成是能当成ndarray使用的类或者字典
- 我们知道，类把多个相关的数据项（属性）组成一个数据实体。例如直接学习的学生考试信息类（Score类）。每个属性都有自己的名字，如Score类中的name、id等。
- Series和类类似，也包含若干个数据项，每个数据项都有自己的名字。可以用.语法来访问Series中的数据项。
- 比类更强的是：
  - Series和ndarray类似，可以用0、1、2……下标来访问各数据项。（可以当ndarray和列表使用）
  - Series和字典类似，可以用数据项的名称作为关键字来访问各数据项。（可以当字典使用）

3.1.1 创建Series对象

```
import numpy as np
import pandas as pd

labels=['a','b','c','d','e']
a=np.arange(5)

#使用数据数组和标签列表建立Series对象
p1 = pd.Series(a,index=labels)
print("p1: ",p1)

#使用字典来建立Series对象
p2=pd.Series({'c':1,'b':2,'a':3,'z':100})
print("p2:",p2)
```

3.1.2 访问数据项和基本运算

```
#分别用数字下标、数据项名称下标、数据项（属性）名访问数据项:
print("p1[0]",p1[0])
print("p1['a']",p1['a'])
print("p1.a",p1.a)
```

```
# 和ndarray做基本运算，相当于两个ndarray做基本运算
print("a+p1",a+p1)
# 和另一个Series做运算，注意是同名项相互运算
print("p1+p2",p1+p2)
# 将series对象当ndarray做numpy中的函数的参数
print(np.sin(p1))
```

3.2 DataFrame

- 一个Series可以看成是一个复杂数据对象，如一个学生的成绩信息，一名病人的体检结果，一条交易记录等等。
- DataFrame相当于一个二维的表格，可以看成：
  - 包含多个Series对象的ndarray（一行为一个Series对象）
- 也可以看成：
  - 包含多个ndarray的Series对象（一列为ndarray）
- 在数据分析处理中，我们通常将DataFrame中的每一列作为一个ndarray进行处理，以实现对数据的批量处理
- Pandas提供了读取csv文件，并将其转换为DataFrame对象；以及将DataFrame对象保存到csv文件中的方法

3.2.1 DataFrame的使用

- DataFrame对象和Series一样，可以当作ndarray对象使用（ndarray可以是多维的）
- 可以使用列名作为下标，从DataFrame中选取一列
- 除了numpy中的一系列方法外，DataFrame本身对象提供了一系列方法，以进行按行或者按列的Filter和Reduce：
  - 按行Filter：和ndarray类似，使用bool数组做下标
  - Reduce：DataFrame.apply方法；sum、mode、mean、max等
    - 如果需要按行操作，使用参数axis=1；如果按列操作，使用参数axis=0

3.2.2 应用举例

- 例3-2：文件3-2.销售信息.csv中保存着某超市某时间段内的所有销售信息（包括商品名、单价和数量）。请为每条销售记录加入销售额项（销售额=单价*数量），并将所有销售额大于100的记录写入result.csv中。

```
import numpy as np
import pandas as pd
from easygraphics import dialog as dlg

with open(filename,mode="r",encoding="GBK") as file:
    df=pd.read_csv(file)
df["销售额"]=df["单价"]*df["数量"] #map运算
index=df["销售额"]>100 #生成筛选bool数组（map运算）
new_df=df[index] #使用bool数组筛选行（filter运算）
df.to_csv("result.csv")
```

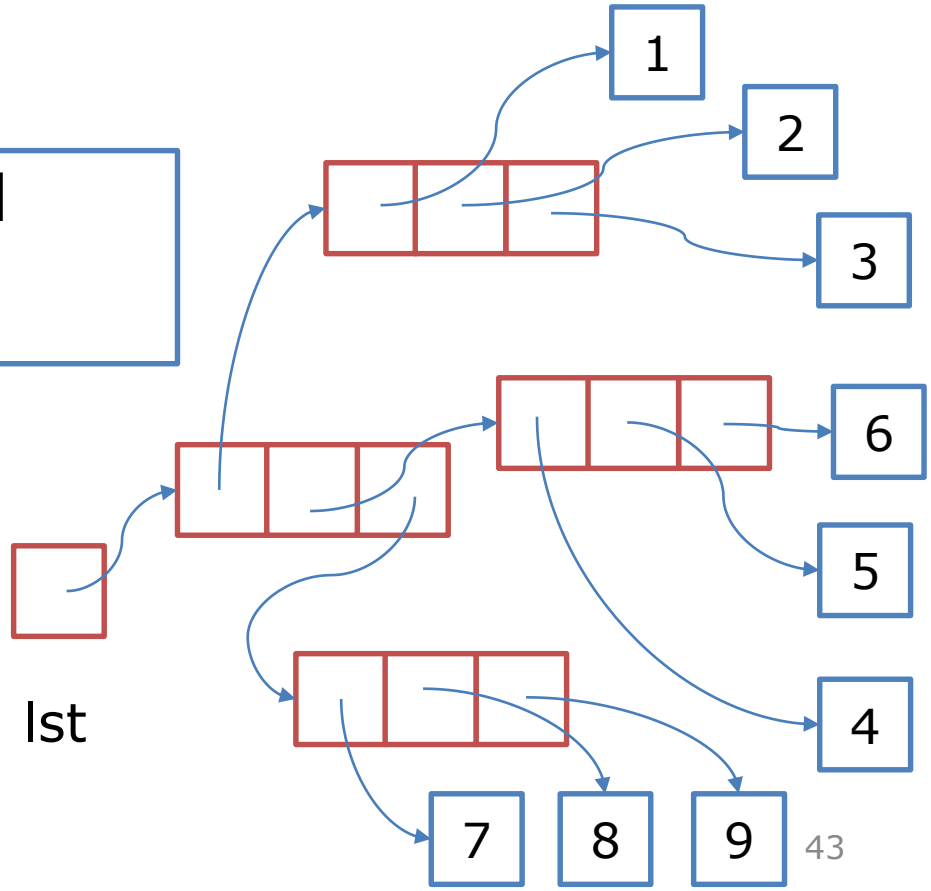
3-2.pandas应用.py

附A：Python多维列表

- python的列表中可以保存任意的对象，当然也可以保存其他列表对象。这样就构成了多维列表。

```
lst=[[1,2,3],[4,5,6],[7,8,9]]
print(lst[1])
print(lst[1][1])
```

创建一个二维列表
使用下标访问其中的元素



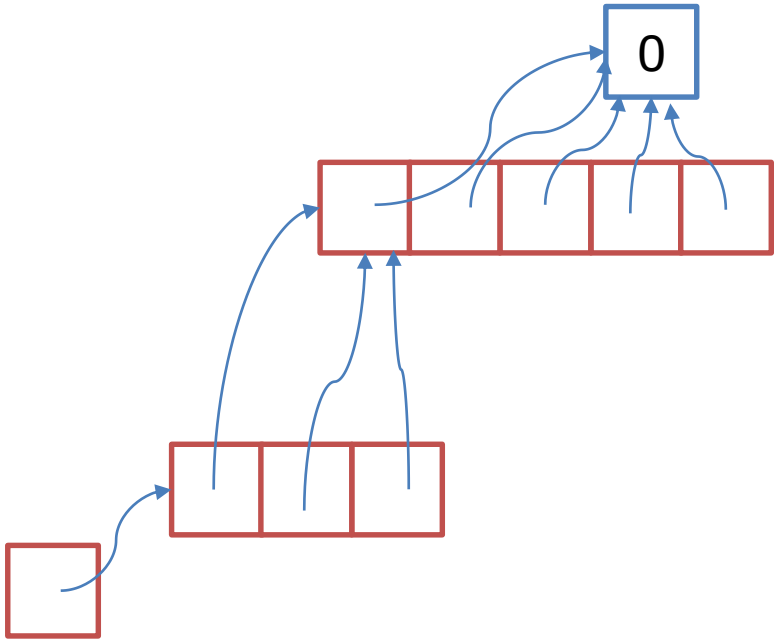
A Python多维列表（二）

- 但是，在多维列表中存在一个重大的陷阱。
- 例如，我想创建一个5行3列的二维列表，其元素都为零：

```
lst=[[0]*5]*3
print(lst)
lst[0][1]=3
print(lst)
```

错误的方法

lst[0]、lst[1]和lst[2]是同一个一维列表



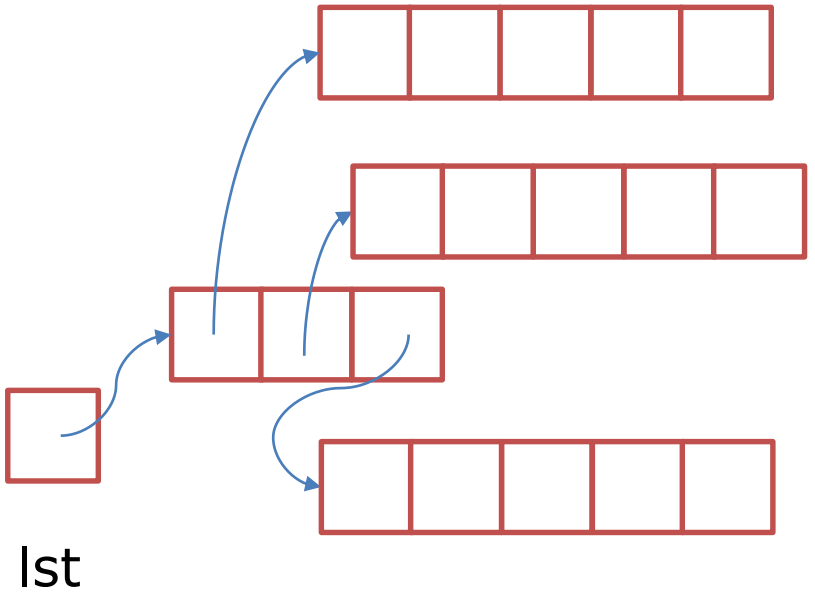
A Python多维列表（三）

- 但是，在多维列表中存在一个重大的陷阱。
- 例如，我想创建一个5行3列的二维列表，其元素都为零：

```
lst = [ [0]*5 for x in range(3)]
print(lst)
lst[0][1]=3
print(lst)
```

正确的方法

lst[0]、lst[1]和lst[2]是不同的一维列表



本章重点

- ndarray的使用方法
- Series的使用方法
- DataFrame的使用方法
- 使用numpy和pandas，基于map、reduce和filter思想解决问题的方法
- 使用矩阵来表示问题的方法

本章练习

- 见练习《第六章 Numpy》