Python程序设计——第四次实验报告

2311095 宋卓伦　计算机科学与技术 2024.10.01

一、实验目的

利用类和对象编写程序，掌握面向对象程序设计方法，理解实例属性和类属性在应用场景上的区别，理解普通方法和内置方法在应用场景上的区别，理解实例方法、类方法和静态方法在应用场景上的区别。

二、实验描述

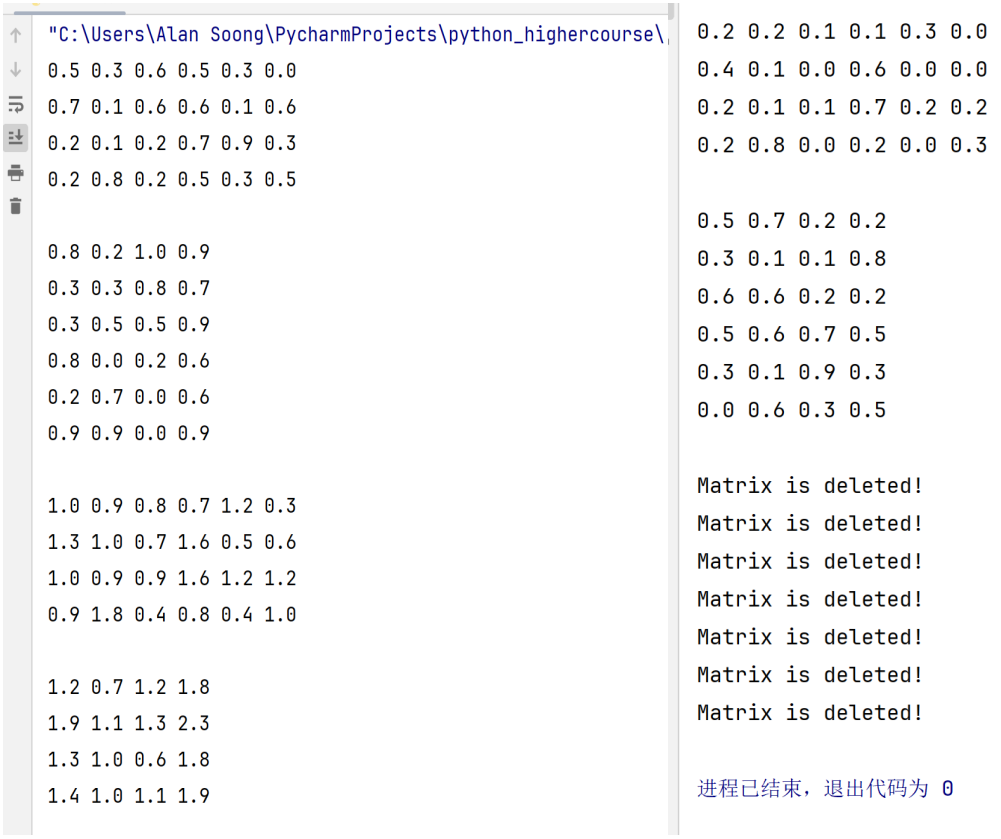
对实验三的程序采用面向对象的方法进行改写。编写矩阵类(保存在一个单独的脚本文件中如mymatrix.py)，至少支持矩阵的乘积、哈达玛积和转置运算(注意需要能够判断传入参数矩阵是否符合计算规则要求)。编写测试程序(如main.py)，对矩阵运类中的方法进行测试，要求矩阵中的元素可根据指定的矩阵形状随机生成、不需要人工输入。

三、实验代码

```
import numpy as np
import random as rd
#两个库，实现生成二维数组和随机数的作用

class Matrix:

    #构造函数初始化
    def __init__(self, A):
        self.A = A

    #矩阵的加法
    def add(self, other):
        if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
            return None
        rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
        for i in range(len(self.A)):
            for j in range(len(self.A[0])):
                rA[i][j] = self.A[i][j] + other.A[i][j]
        result = Matrix(rA)
        return result

    #矩阵的减法
    def sub(self, other):
        if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
            return None
        rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
        for i in range(len(self.A)):
            for j in range(len(self.A[0])):
                rA[i][j] = self.A[i][j] - other.A[i][j]
```

```
        result = Matrix(rA)
        return result

    #矩阵的数乘
    def num_mul(self, p):
        rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
        for i in range(len(self.A)):
            for j in range(len(self.A[0])):
                rA[i][j] = self.A[i][j] * p
        result = Matrix(rA)
        return result

    #矩阵的乘法
    def mul(self, other3):
        if len(self.A[0]) != len(other3.A):
            return None
        rA = [[0 for j in range(len(other3.A[0]))] for i in
range(len(self.A))]
        for i in range(len(self.A)):
            for j in range(len(other3.A[0])):
                for k in range(len(other3.A)):
                    rA[i][j] += self.A[i][k] * other3.A[k][j]
        result = Matrix(rA)
        return result

    #矩阵的哈德马乘法
    def hadamard(self, other):
        if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
            return None
        rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
        for i in range(len(self.A)):
            for j in range(len(self.A[0])):
                rA[i][j] = self.A[i][j] * other.A[i][j]
        result = Matrix(rA)
        return result

    #矩阵转置
    def switch(self):
        rA = [[0 for j in range(len(self.A))] for i in
range(len(self.A[0]))]
        for i in range(len(self.A[0])):
            for j in range(len(self.A)):
                rA[i][j] = self.A[j][i]
        result = Matrix(rA)
        return result

    #封装的矩阵打印函数
    def print_matrix(self):
        if self is None:
            print("Error!")
        else:
            for i in range(len(self.A)):
                for j in range(len(self.A[0])):
                    print('%.1f' % self.A[i][j], end="")
```

```
                    if j != len(self.A[0]) - 1:
                        print(end=" ")
                print(end="\n")
        print()

    #析构函数
    def __del__(self):
        print("Matrix is deleted!")

#主程序
if __name__ == '__main__':

    #定义两个矩阵
    A = Matrix(np.random.rand(4, 6))
    B = Matrix(np.random.rand(6, 4))
    Matrix.print_matrix(A)
    Matrix.print_matrix(B)

    #计算加法（减法同理）
    C = Matrix(np.random.rand(4, 6))
    add_re = A.add(C)
    Matrix.print_matrix(add_re)

    #计算乘法
    D = A.mul(B)
    Matrix.print_matrix(D)

    #计算哈德马乘法
    E = A.hadamard(C)
    Matrix.print_matrix(E)

    #计算矩阵转置
    F = A.switch()
    Matrix.print_matrix(F)

```



图中打印了矩阵的运算结果，注意到函数构造了7次，所以析构了7次。

四、实验问题

1、计算时形参和引用的问题没有搞清楚，导致计算的时候改变了原来矩阵的值

修正前的代码：

```
#以矩阵加法为例
def add(self, other1):
    if len(self.A) != len(other1) | len(other1[0]) != len(self.A[0]):
        return None
    result = self
    for i in range(len(self.A)):
        for j in range(len(self.A[0])):
            result.A[i][j] = self.A[i][j] + other1[i][j]
    return result

```

这样会导致self本身的值被修改，以下是修正后的代码：

```
#以矩阵加法为例：
def add(self, other1):
    if len(self.A) != len(other1.A) & len(other1.A[0]) != len(self.A[0]):
        return None
    rA = [[0 for j in range(len(self.A[0]))] for i in range(len(self.A))]
    for i in range(len(self.A)):
        for j in range(len(self.A[0])):
            rA[i][j] = self.A[i][j] + other1.A[i][j]
    result = Matrix(rA)
    return result

```

采用提前设定一个空矩阵的值来存储计算的结果，从而不会改变self矩阵自带的值。

2、代码中仍然存在大量重复的内容，这一点我目前还没有很好的办法，只能是封装相关操作。

3、注意线性代数的相关知识，矩阵的运算；同时防止计算的越界。

4、注意矩阵的返回值和形参，一般保证形参和实例的类型是一致的，同时self类似于C++中的this指针。

5、发现了两种写法：

```
……
F.print_matrix()
Matrix.print_matrix(F)
……
```

这两种写法对于只含有self形参的函数来说，都是有效的。