

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

```
1  import numpy as np
2  import random as rd
3  #两个库，实现生成二维数组和随机数的作用
4
5  class Matrix:
6
7      #构造函数初始化
8      def __init__(self, A):
9          self.A = A
10
11      #矩阵的加法
12      def add(self, other):
13          if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
14              return None
15          rA = [[0 for j in range(len(self.A[0]))] for i in
16 range(len(self.A))]
17          for i in range(len(self.A)):
18              for j in range(len(self.A[0])):
19                  rA[i][j] = self.A[i][j] + other.A[i][j]
20          result = Matrix(rA)
21          return result
22
23      #矩阵的减法
24      def sub(self, other):
25          if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
26              return None
27          rA = [[0 for j in range(len(self.A[0]))] for i in
28 range(len(self.A))]
29          for i in range(len(self.A)):
30              for j in range(len(self.A[0])):
31                  rA[i][j] = self.A[i][j] - other.A[i][j]
```

```

30         result = Matrix(rA)
31         return result
32
33     #矩阵的数乘
34     def num_mul(self, p):
35         rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
36         for i in range(len(self.A)):
37             for j in range(len(self.A[0])):
38                 rA[i][j] = self.A[i][j] * p
39         result = Matrix(rA)
40         return result
41
42     #矩阵的乘法
43     def mul(self, other3):
44         if len(self.A[0]) != len(other3.A):
45             return None
46         rA = [[0 for j in range(len(other3.A[0]))] for i in
range(len(self.A))]
47         for i in range(len(self.A)):
48             for j in range(len(other3.A[0])):
49                 for k in range(len(other3.A)):
50                     rA[i][j] += self.A[i][k] * other3.A[k][j]
51         result = Matrix(rA)
52         return result
53
54     #矩阵的哈德马乘法
55     def hadamard(self, other):
56         if len(self.A) != len(other.A) & len(other.A[0]) != len(self.A[0]):
57             return None
58         rA = [[0 for j in range(len(self.A[0]))] for i in
range(len(self.A))]
59         for i in range(len(self.A)):
60             for j in range(len(self.A[0])):
61                 rA[i][j] = self.A[i][j] * other.A[i][j]
62         result = Matrix(rA)
63         return result
64
65     #矩阵转置
66     def switch(self):
67         rA = [[0 for j in range(len(self.A))] for i in
range(len(self.A[0]))]
68         for i in range(len(self.A[0])):
69             for j in range(len(self.A)):
70                 rA[i][j] = self.A[j][i]
71         result = Matrix(rA)
72         return result
73
74     #封装的矩阵打印函数
75     def print_matrix(self):
76         if self is None:
77             print("Error!")
78         else:
79             for i in range(len(self.A)):
80                 for j in range(len(self.A[0])):
81                     print('%0.1f' % self.A[i][j], end="")

```

```
82         if j != len(self.A[0]) - 1:
83             print(end=" ")
84         print(end="\n")
85     print()
86
87     #析构函数
88     def __del__(self):
89         print("Matrix is deleted!")
90
91     #主程序
92     if __name__ == '__main__':
93
94         #定义两个矩阵
95         A = Matrix(np.random.rand(4, 6))
96         B = Matrix(np.random.rand(6, 4))
97         Matrix.print_matrix(A)
98         Matrix.print_matrix(B)
99
100        #计算加法（减法同理）
101        C = Matrix(np.random.rand(4, 6))
102        add_re = A.add(C)
103        Matrix.print_matrix(add_re)
104
105        #计算乘法
106        D = A.mul(B)
107        Matrix.print_matrix(D)
108
109        #计算哈德马乘法
110        E = A.hadamard(C)
111        Matrix.print_matrix(E)
112
113        #计算矩阵转置
114        F = A.switch()
115        Matrix.print_matrix(F)
116
```

```
"C:\Users\Alan Soong\PycharmProjects\python_highercourse\
0.5 0.3 0.6 0.5 0.3 0.0
0.7 0.1 0.6 0.6 0.1 0.6
0.2 0.1 0.2 0.7 0.9 0.3
0.2 0.8 0.2 0.5 0.3 0.5

0.8 0.2 1.0 0.9
0.3 0.3 0.8 0.7
0.3 0.5 0.5 0.9
0.8 0.0 0.2 0.6
0.2 0.7 0.0 0.6
0.9 0.9 0.0 0.9

1.0 0.9 0.8 0.7 1.2 0.3
1.3 1.0 0.7 1.6 0.5 0.6
1.0 0.9 0.9 1.6 1.2 1.2
0.9 1.8 0.4 0.8 0.4 1.0

1.2 0.7 1.2 1.8
1.9 1.1 1.3 2.3
1.3 1.0 0.6 1.8
1.4 1.0 1.1 1.9

0.2 0.2 0.1 0.1 0.3 0.0
0.4 0.1 0.0 0.6 0.0 0.0
0.2 0.1 0.1 0.7 0.2 0.2
0.2 0.8 0.0 0.2 0.0 0.3

0.5 0.7 0.2 0.2
0.3 0.1 0.1 0.8
0.6 0.6 0.2 0.2
0.5 0.6 0.7 0.5
0.3 0.1 0.9 0.3
0.0 0.6 0.3 0.5

Matrix is deleted!
Matrix is deleted!
Matrix is deleted!
Matrix is deleted!
Matrix is deleted!
Matrix is deleted!
Matrix is deleted!

进程已结束，退出代码为 0
```

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：

```
1 #以矩阵加法为例
2 def add(self, other1):
3     if len(self.A) != len(other1) | len(other1[0]) != len(self.A[0]):
4         return None
5     result = self
6     for i in range(len(self.A)):
7         for j in range(len(self.A[0])):
8             result.A[i][j] = self.A[i][j] + other1[i][j]
9     return result
10
```

这样会导致self本身的值被修改，以下是修正后的代码：

```
1 #以矩阵加法为例：
2 def add(self, other1):
3     if len(self.A) != len(other1.A) & len(other1.A[0]) != len(self.A[0]):
4         return None
5     rA = [[0 for j in range(len(self.A[0]))] for i in range(len(self.A))]
6     for i in range(len(self.A)):
7         for j in range(len(self.A[0])):
8             rA[i][j] = self.A[i][j] + other1.A[i][j]
9     result = Matrix(rA)
10    return result
11
```

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：

```
1 .....  
2 F.print_matrix()  
3 Matrix.print_matrix(F)  
4 .....
```

这两种写法对于只含有self形参的函数来说，都是有效的。