实验报告

1 预习内容Python面向对象编程（OOP）是一种组织代码的方式，主要通过类和对象来实现。类是对象的蓝图或模板，通过定义类，可以创建多个对象。类和对象的基本概念包括实例属性和类属性：实例属性是对象独有的，通过self引用；类属性是类本身所有的，所有对象共享。类的方法包括实例方法、类方法和静态方法，实例方法需要通过self访问，类方法使用@classmethod装饰器并通过cls访问，静态方法使用@staticmethod装饰器，不需要访问实例或类。

Python的构造函数是\_\_init\_\_方法，用于初始化对象的属性，析构函数是\_\_del\_\_方法，用于在对象销毁时执行清理操作。Python支持运算符重载，通过定义特殊方法（如\_\_add\_\_, \_\_sub\_\_等）实现自定义对象的运算符行为。这些特性使得Python的面向对象编程非常灵活和强大，能够更好地组织和管理复杂的代码。

2 程序代码

（1）class ComplexNumber:  
 def \_\_init\_\_(self, real, imag):  
 self.real = real  
 self.imag = imag  
  
 def \_\_str\_\_(self):  
 return f"{self.real} + {self.imag}i"  
  
 def \_\_add\_\_(self, other):  
 real\_part = self.real + other.real  
 imag\_part = self.imag + other.imag  
 return ComplexNumber(real\_part, imag\_part)  
  
 def \_\_sub\_\_(self, other):  
 real\_part = self.real - other.real  
 imag\_part = self.imag - other.imag  
 return ComplexNumber(real\_part, imag\_part)  
  
 def \_\_mul\_\_(self, other):  
 real\_part = self.real \* other.real - self.imag \* other.imag  
 imag\_part = self.real \* other.imag + self.imag \* other.real  
 return ComplexNumber(real\_part, imag\_part)  
  
 def \_\_truediv\_\_(self, other):  
 denominator = other.real \*\* 2 + other.imag \*\* 2  
 real\_part = (self.real \* other.real + self.imag \* other.imag) / denominator  
 imag\_part = (self.imag \* other.real - self.real \* other.imag) / denominator  
 return ComplexNumber(real\_part, imag\_part)  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 c1 = ComplexNumber(3, 4) # 3 + 4i  
 c2 = ComplexNumber(1, 2) # 1 + 2i  
  
 print(f"+: {c1} + {c2} = {c1 + c2}")  
 print(f"-: {c1} - {c2} = {c1 - c2}")  
 print(f"\*: {c1} \* {c2} = {c1 \* c2}")  
 print(f"/: {c1} / {c2} = {c1 / c2}")

（2）class Matrix:  
 def \_\_init\_\_(self, matrix):  
 self.matrix = matrix # 初始化矩阵数据  
 self.rows = len(matrix) # 记录矩阵行数  
 self.cols = len(matrix[0]) # 记录矩阵列数  
  
 def \_\_str\_\_(self):  
 result = ""  
 for row in self.matrix:  
 result += " ".join(map(str, row)) + "\n" # 将矩阵转换为字符串形式  
 return result.strip()  
  
 def \_\_add\_\_(self, other):  
 if self.rows != other.rows or self.cols != other.cols:  
 raise ValueError("矩阵维度必须相同才能相加") # 检查矩阵维度是否相同  
  
 result = []  
 for i in range(self.rows):  
 row = []  
 for j in range(self.cols):  
 row.append(self.matrix[i][j] + other.matrix[i][j])  
 result.append(row)  
  
 return Matrix(result)  
  
 def \_\_sub\_\_(self, other):  
 if self.rows != other.rows or self.cols != other.cols:  
 raise ValueError("矩阵维度必须相同才能相减") # 检查矩阵维度是否相同  
  
 result = []  
 for i in range(self.rows):  
 row = []  
 for j in range(self.cols):  
 row.append(self.matrix[i][j] - other.matrix[i][j])  
 result.append(row)  
  
 return Matrix(result)  
  
 def inverse(self):  
 if self.rows != self.cols:  
 raise ValueError("矩阵必须是方阵才能求逆") # 检查矩阵是否为方阵  
  
 # 创建单位矩阵  
 identity = [[1 if i == j else 0 for j in range(self.cols)] for i in range(self.rows)]  
 augmented\_matrix = [row[:] + identity[i][:] for i, row in enumerate(self.matrix)]  
  
 # 高斯消元法求逆矩阵  
 for col in range(self.cols):  
 # 寻找主元素所在行  
 max\_row = max(range(col, self.rows), key=lambda i: abs(augmented\_matrix[i][col]))  
 augmented\_matrix[col], augmented\_matrix[max\_row] = augmented\_matrix[max\_row], augmented\_matrix[col]  
  
 # 主元素归一化  
 主元素 = augmented\_matrix[col][col]  
 if 主元素 == 0:  
 raise ValueError("矩阵奇异，无法求逆")  
 augmented\_matrix[col] = [entry / 主元素 for entry in augmented\_matrix[col]]  
  
 # 消元操作，将当前列的其他行消成零  
 for row in range(self.rows):  
 if row != col:  
 因子 = augmented\_matrix[row][col]  
 augmented\_matrix[row] = [entry - 因子 \* augmented\_matrix[col][i] for i, entry in  
 enumerate(augmented\_matrix[row])]  
  
 # 提取逆矩阵  
 逆矩阵 = [row[self.cols:] for row in augmented\_matrix]  
 return Matrix(逆矩阵)  
  
  
# 测试  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 测试数据  
 matrix1 = Matrix([[1, 2], [3, 4]])  
 matrix2 = Matrix([[5, 6], [7, 8]])  
  
 # 测试加法  
 print("矩阵加法测试:")  
 print(matrix1 + matrix2)  
  
 # 测试减法  
 print("\n矩阵减法测试:")  
 print(matrix1 - matrix2)  
  
 # 测试求逆  
 matrix3 = Matrix([[1, 2], [3, 4]])  
 print("\n原始矩阵:")  
 print(matrix3)  
 print("\n逆矩阵:")  
 print(matrix3.inverse())

3 成果展示

+: 3 + 4i + 1 + 2i = 4 + 6i

-: 3 + 4i - 1 + 2i = 2 + 2i

\*: 3 + 4i \* 1 + 2i = -5 + 10i

/: 3 + 4i / 1 + 2i = 2.2 + -0.4i

矩阵加法测试:

6 8

10 12

矩阵减法测试:

-4 -4

-4 -4

原始矩阵:

1 2

3 4

逆矩阵:

-1.9999999999999996 0.9999999999999998

1.4999999999999998 -0.49999999999999994

4 心得体会

在本次课程设计中，我深入学习了Python面向对象编程（OOP）的核心概念和应用，通过实际编写类和对象的方法，进一步巩固了面向对象编程的理解。以下是我在这次课程设计中的一些心得体会。

首先，通过实现复数运算类（ComplexNumber），我深刻理解了类和对象的基本概念以及运算符重载的机制。在设计复数类时，通过定义\_\_init\_\_方法初始化复数的实部和虚部，利用\_\_str\_\_方法实现复数的字符串表示形式，使得复数对象的输出更加直观。此外，通过重载加法（\_\_add\_\_）、减法（\_\_sub\_\_）、乘法（\_\_mul\_\_）和除法（\_\_truediv\_\_）运算符，实现了自定义的复数运算功能，这不仅增强了代码的可读性和可维护性，还提升了编程的灵活性。

其次，在实现矩阵运算类（Matrix）时，我学习并应用了矩阵的基本运算方法，包括矩阵的加法、减法和求逆运算。在编写矩阵加法和减法方法时，通过遍历矩阵的元素并进行对应的加减运算，解决了矩阵维度不匹配的问题。特别是在求矩阵逆的过程中，我深入理解了高斯消元法的原理，并通过编程实现这一算法，成功解决了矩阵求逆的问题。这一过程不仅加深了我对线性代数知识的理解，也让我体会到算法在编程中的重要性。

最后，通过本次课程设计，我进一步体会到了面向对象编程的优势。在实际编程中，通过将相关功能封装到类中，可以提高代码的模块化和重用性，使得代码结构更加清晰，逻辑更加严谨。面向对象编程不仅帮助我更好地组织和管理复杂的代码，还培养了我抽象思维和解决实际问题的能力。

总体而言，本次课程设计让我在面向对象编程的理论知识与实践技能方面都有了显著提升。我学会了如何通过类和对象来组织代码，并通过实践巩固了对运算符重载、高斯消元法等高级编程技巧的掌握。未来的学习和工作中，我将继续深入探索Python的更多高级特性和应用，不断提高自己的编程能力和解决实际问题的能力。

# 封面格式

出于保留格式的需求，相应封面在后续页。

|  |  |
| --- | --- |
| **成 绩** |  |

****

**中 国 矿 业 大 学**

**China University of Mining and Technology**

**20 - 20 学年**

**《Python编程实践》**

**课 程 设 计 报 告**

院 系： 信息与控制工程学院

班 级：

学 号：

日 期：

姓 名：

签 名：

注：若要求交打印稿，（1）请在姓名处手写签名 ，（2）并删除本行文字

|  |  |
| --- | --- |
| **成 绩** |  |

****

**中 国 矿 业 大 学**

**China University of Mining and Technology**

**20 - 20 学年**

**《Python编程实践》**

**第 次实验报告**

院 系： 信息与控制工程学院

班 级：

学 号：

日 期：

姓 名：

签 名：

注：若要求交打印稿，（1）请在姓名处手写签名 ，（2）并删除本行文字