第九周问答作业

朱士杭 231300027

2024 年 4 月 24 日

1 问题一：科里化及其应用场景

1.1 什么是科里化

科里化（Currying）是一种在函数式编程中常用的技巧，它将一个接受多个参数的函数转换为一系列使用一个参数的函数。

更具体地说，科里化是将一个 n 元函数转换成 n 个一元函数的过程。每个一元函数返回一个新的一元函数，直到所有参数都被应用。在科里化过程中，我们不是一次性传递所有参数，而是每次传递一个参数，并且返回一个新的函数，这个新函数等待下一个参数。这个过程一直重复，直到所有参数都传递完毕。

1.2 科里化使用场景

参数复用：当我们需要多次调用某个函数，并且这个函数的大部分参数在每次调用中都是相同的，可以使用科里化来避免重复传递这些参数。

函数组合：在函数式编程中，科里化使得函数组合变得更加容易，因为科里化后的函数总是返回一个新函数，这与其他函数的输出可以很容易地结合起来。

动态函数创建：科里化允许我们创建新的函数，这些函数预设了一些参数，这在需要根据特定条件动态生成函数时非常有用。

延迟计算：当我们不想立即执行一个函数，而是想要在未来的某个时间点执行时，科里化可以帮助我们“记住”已经提供的参数，并在需要时应用剩余的参数。

优化代码：在某些情况下，科里化可以用来优化代码的结构，使其更加模块化和可维护。

2 问题二：map 函数的功能及其用途

2.1 map 函数功能

它用于将一个函数应用于一个序列（如列表、元组或字符串）中的每个元素，并返回一个迭代器，其中包含应用函数后的结果。

简单来说，map 用于对序列中的每个元素执行同一个操作，就跟 map 的名字一样，本身就是一种映射关系，甚至于说是一种双射。

2.2 map 函数用途

批量处理数据：当你需要对一个数据集合中的每个元素执行相同的操作时，map 非常有用。例如，如果你有一个数字列表，并想要将每个数字乘以 2，可以使用 map 来实现。

函数式编程：在函数式编程中，map 是一个基本的概念，它允许你将函数作为第一个类对象（first-class citizen）来处理，即将函数作为参数传递给其他函数。

代码简洁：使用 map 可以使得代码更加简洁，尤其是当你需要对多个序列进行相同的操作时。map 可以接受多个序列作为输入，只要函数接受多个参数即可。

性能优化：在某些情况下，使用 map 可以比传统的循环方法更高效，尤其是在处理大规模数据时（好像是因为分布式处理的缘故吧）

3 复合函数概念及其用途

3.1 复合函数概念

python 中的函数复合概念和数学中的函数复合相当类似，如果将函数视为一种运算的话，由于运算具有顺序，函数结合也是由顺序的，但是将不同函数作用视为一个整体的话记为函数复合，将一个函数的输出作为另一个函数的输入，从而创建一个新的函数。

3.2 复合函数用途

代码重用和模块化：通过组合现有的函数，你可以创建新的函数而无需编写全新的代码。这有助于代码重用并促进模块化设计。

简化复杂操作：函数复合允许你将复杂的操作分解为更小、更简单的函数。然后，你可以将这些简单函数组合起来，以执行更复杂的任务。并且在函数式编程当中，它允许你以声明式的方式描述数据处理流程，而不是以命令式的方式。

提高代码可读性：通过组合函数，你可以创建一个清晰的操作链，这有助于提高代码的可读性和可维护性。

4 抽象与封装

关于抽象其实分为过程抽象与数据抽象，过程抽象主要针对于函数式编程，而数据抽象主要针对面向对象编程

抽象允许我们创建模型，这些模型只包含对解决问题至关重要的特征，而忽略不必要的细节，仅用一个名字就完成了一个过程或者一个对象

封装是指将数据（属性）和操作数据的方法（函数）捆绑在一起，形成一个统一的单元，即对象。

封装的目的是隐藏对象的内部状态和实现细节，只暴露必要的接口与外部通信。这样，对象的内部表示就不能被外部直接访问和修改，必须通过定义好的接口进行。

封装实际上类似于一种黑盒的存在，在程序中将过程与数据打包并且对外部不可见，从而实现保护数据隐私以及过程隐蔽性

在 python 当中抽象和封装主要通过类来实现，通过创建一个类对应的实例即对象来生成一块独立的个体，在这个个体里面可以定义成员函数来实现过程抽象，可以设置对象专属的数据来实现数据抽象与封装

5 面向对象与面向过程程序设计

面向对象程序设计是一种以对象为基础的编程范式，它将数据和操作数据的方法封装在一起，形成对象。

OOP（Object-Oriented-Program）的核心理念是将现实世界的实体抽象为程序中的对象，并通过对象之间的交互来完成任务。

基于过程的程序设计是一种以过程（函数或方法）为中心的编程范式。

在过程式编程中，程序被划分为一系列的函数，这些函数按照一定的顺序执行，以完成特定的任务。过程式编程关注的是函数和程序流程，而不是数据和对象

面向对象程序设计有哪些基本特征嘞：抽象、封装、继承、多态（这个在上个学期的 C++ 里面已经讲过啦太经典的问题啦哈哈哈）

6 make_filter 高阶函数

```python
# make_filter高阶函数
1个用法
def make_filter(condition):
    def filter_list(lst):
        # 实现过滤逻辑
        # 核心思路：如果满足条件（True），则保留，不满足则删除
        return [i for i in lst if condition(i)]  # 删除这一行，并添加适当的代码
    return filter_list
1个用法
def is_even(number):
    return number % 2 == 0
# 使用 make_filter 创建过滤器
even_filter = make_filter(is_even)
# 测试 even_filter
test_list = [1, 2, 3, 4, 5, 6]
filtered_list = even_filter(test_list)
print(filtered_list)  # 应该输出 [2, 4, 6]
```

图 1: 使用高阶函数创建过滤器

在这里核心思路就是创建高阶函数 make_filter，并且在内部定义一个过滤函数并将其返回，到时候只需要判断是否满足条件 condition 即可，这里 return 很简单，使用一下最基本的列表解析即可如果满足则包含在列表当中，不满足则丢掉即可

7 Book 类模拟图书借阅过程

```python
#Book类模拟图书借阅过程
3个用法
class Book:
    is_borrowed=False
    def __init__(self,title,author):
        self.title=title
        self.author=author
        self.borrower_history = []
    5个用法
    def borrow_book(self,borrow_name):
        '''借书的成员函数/方法'''
        if self.is_borrowed==True:
            len_his=len(self.borrower_history)
            print(f"{self.title}这本书已经被{self.borrower_history[len_his-1]}借走啦")
        else:
            self.is_borrowed=True
            self.borrower_history.append(borrow_name)
    2个用法
    def return_book(self):...
    2个用法
    def get_borrow_history(self):
        if self.borrower_history!=[]:
            print(f"所有借阅过{self.title}这本书的有：",self.borrower_history)
        else:
            print(f"{self.title}这本书至今仍未被借出去过")
    2个用法
    def print_is_borrowed(self):
        if self.is_borrowed:
            print(f"{self.title}这本书已经被借出去啦")
        else:
            print(f"{self.title}这本书还未被借出去")
#创建三本书的实例
book_list=[]
book_list.append(Book( title: "《必有人重写爱情》", author: "北岛"))
book_list.append(Book( title: "《人工智能：一种现代方法（第四版）》", author: "斯图尔特·罗素"))
book_list.append(Book( title: "《黄金时代》", author: "王小波"))
#一本书被同一个人多次借阅和归还
print("****接下来测试一本书被同一个人多次借阅和归还的场景****")
book_list[2].borrow_book("王二")
book_list[2].print_is_borrowed()
book_list[2].return_book()
book_list[2].print_is_borrowed()
book_list[2].borrow_book("王二")
book_list[2].borrow_book("王二")
book_list[2].return_book()
book_list[2].get_borrow_history()
#尝试借已经被借出的书
print("****接下来测试尝试借已经被借出的书的场景****")
book_list[0].borrow_book("朱士杭")
book_list[0].borrow_book("Edison")
#打印借阅历史记录
print("****接下来测试打印借阅历史记录的场景****")
for i in book_list:
    i.get_borrow_history()
```

图 2: 模拟图书借阅过程

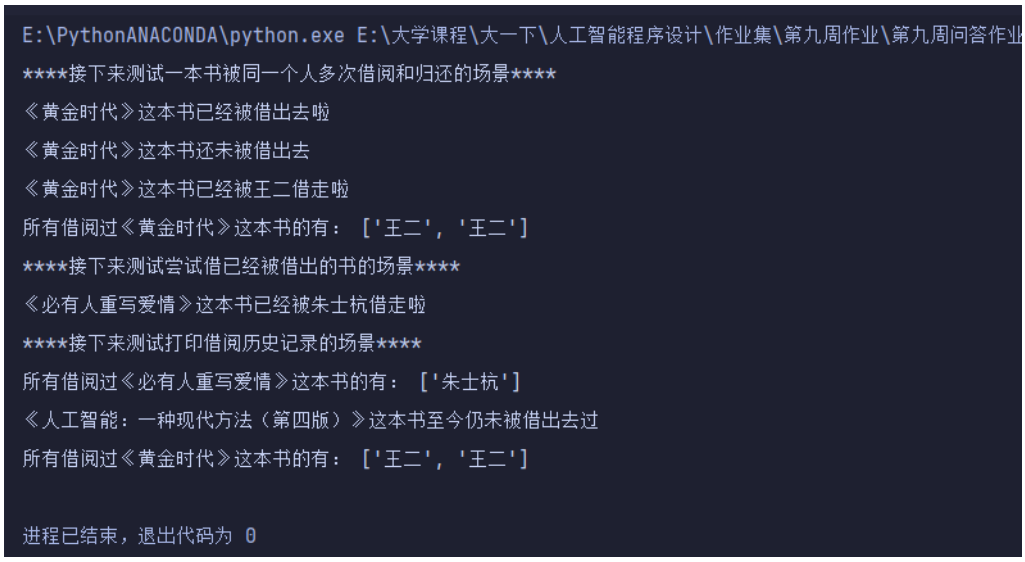
E:\PythonANACONDA\python.exe E:\大学课程\大一下\人工智能程序设计\作业集\第九周作业\第九周问答作业
****接下来测试一本书被同一个人多次借阅和归还的场景****
《黄金时代》这本书已经被借出去啦
《黄金时代》这本书还未被借出去
《黄金时代》这本书已经被王二借走啦
所有借阅过《黄金时代》这本书的有： ['王二', '王二']
****接下来测试尝试借已经被借出的书的场景****
《必有人重写爱情》这本书已经被朱士杭借走啦
****接下来测试打印借阅历史记录的场景****
所有借阅过《必有人重写爱情》这本书的有： ['朱士杭']
《人工智能：一种现代方法（第四版）》这本书至今仍未被借出去过
所有借阅过《黄金时代》这本书的有： ['王二', '王二']
进程已结束，退出代码为 0

图 3: 代码运行结果