山东大学网络空间安全学院

Python高级程序设计课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100460065 | 姓名：李昕 | | 班级：密码21 二班 |
| 实验题目：实验四 函数的定义与使用2 | | | |
| 实验学时：2 | | 实验日期：2022/10/18 | |
| 实验目的：熟悉yield的使用，熟悉递归函数的定义与使用，熟悉函数嵌套。 | | | |
| 硬件环境：  AMD Ryzen 7 5800H with Radeon Graphics     3.20 GHz  机带 RAM 16.0 GB (13.9 GB 可用) | | | |
| 软件环境：IDLE | | | |
| 实验步骤与内容：  本次实验包括四个问题：   1. 编写递归代码解答汉诺塔问题：据说古代有一个梵塔，塔内有三个底座A、B、C，A 座上有64 个盘子，盘子大小不等，大的在下，小的在上。有一个和尚想把这64 个盘子从A 座移到C 座，但每次只能允许移动一个盘子。在移动盘子的过程中可以利用B 座，但任何时刻3 个座上的盘子都必须始终保持大盘在下、小盘在上的顺序。如果只有一个盘子，则不需要利用B 座，直接将盘子从A 移动到C 即可。编写函数，接收一个表示盘子数量的参数和分别表示源、目标、临时底座的参数，然后输出详细移动步骤和每次移动后三个底座上的盘子分布情况。   特别的，函数定义如下：num表示多少个盘子, src表示原底座‘A’，dst表示目标底座’C’，temp表示中间的临时底座’B’。底座上的盘子可用列表表示，num个大小不一的盘子可用0~(num-1)这些数字表示，则初始状态下’A’底座的列表为[num-1,num-2,…,2,1,0],’B’底座的列表为空，‘C’底座的列表为空。def hannoi(num, src, dst, temp=None):  2. 编写函数，求解八皇后问题。（使用yield和不使用yield）  3. 编写程序，实现一个计算输入数值的平均数的计算器。该计算器可以根据需要接受输入的值，并计算当前所有输入值的平均值。然后还可以继续接受输入值，继续计算平均值。直到外部主动结束不再输入数值。（使用yield from实现）  4. 编写一个装饰器函数，可以实现为函数增加计算函数执行时间的功能。支持带输入参数和返回值的函数。  【汉诺塔问题】  1. 汉诺塔问题可以简化为三个盘子的情况，具体移动方法为：我们可以把n个盘子想成两个部分：第n个盘子和前n-1个盘子。要想把这n个盘子从1柱移动到3柱，首先需要把前n-1个盘子先移动到2柱上，接着将第n个盘子从1柱移动到三柱上。（注意：我们这里把前n-1个盘子看成一个整体。）  2. 分析上述过程，发现可以把利用递归将问题化成3,4,5…n个盘子的问题，从而构造出递归函数，其接受一个表示盘子数量的参数和分别表示源，目标，临时底座的参数。  **代码实现：**   1. class Pillar: 2. def \_\_init\_\_(self,str,num): 3. self.key=str 4. self.value=num 5. def Hanoi(num,a,b,c): 6. if num==1: 7. a.value-=1 8. c.value+=1 9. print(a.key,"->",c.key) 10. print("盘子个数 A:{},B:{},C:{}".format(a.value,b.value,c.value)) 11. else: 12. Hanoi(num-1,a,c,b) 13. Hanoi(1,a,b,c) 14. Hanoi(num-1,b,a,c) 15. def main(): 16. num=eval(input("A上的盘子有：")) 17. a=Pillar('A',num) 18. b=Pillar('B',0) 19. c=Pillar('C',0) 20. Hanoi(num,a,b,c) 21. main()   运行结果：    【八皇后问题yield和不用yield】  1.问题分析：（1）要使两个棋子不能在同一行，不妨设置一个队列，队列里的每个元素代表一行；（2）任意两个棋子不能在同一列即设置的队列里每个元素的数值代表着每行棋子的列号。（3）要使两个棋子不能在同一斜线上，可以把整个棋盘当作是一个XOY平面，原点在棋盘的左上角，斜线的斜率为1或者-1，可以推导出Y-X=n或者Y+X=n。也就是说在同一斜线上的两个棋子行号与列号之和或者之差不相同。  2.编写包含yield的代码时，使用conflict函数接受（用状态元组表示的）既有皇后的位置，并确定下一个皇后的位置是否会导致冲突。如果下一个皇后与当前皇后的水平距离为0或与他们的垂直距离相等（位于一条对角线上），则表达式**abs(newx-newy)in(len(state)-i,0)**为真  **代码实现1：**   1. *#包含yeild* 2. def conflict(state, newx): 3. for i, newy in enumerate(state): 4. if abs(newx-newy) in (len(state)-i, 0):   *# 判断表达式* 5. return False 6. return True 8. def queens(num, state=list()): 9. for m in range(num): 10. if conflict(state, m): 11. if len(state) == num-1: 12. yield [m] 13. else: 14. for k in queens(num,state+[m]): 15. yield [m] + k 16. def prettyprint(i): 17. def line(pos,length=len(i)): 18. return  '.'\*(pos)+'X'+'.'\*(length-pos-1) 19. for pos in i: 20. print(line(pos)) 21. n=int(input("请输入皇后的个数：")) 22. for i in queens(n): 23. print(i) 24. for i in queens(n): 25. prettyprint(i) 26. print(' ') 27. print('--------------------------') 28. print(' ')   **代码实现2：**  **#不包含yeild的八皇后代码**   1. def queen(state,a): *#a为当前行* 2. m=len(state) 3. if a == m: 4. print(state) 5. return 0 6. for k in range(m): 7. state[a], flag = k, True 8. for row in range(a): 9. if (state[row] == k or abs(k - state[row]) == a - row): 10. flag = False 11. break 12. if flag: 13. queen(state, a+1) 14. n=int(input("请输入皇后的个数：")) 15. queen([None]\*n,0)   执行结果：      【计算输入数值的平均数的计算器】  python中yield的用法很像return，都是提供一个返回值，但是yield和return的最大区别在于，return一旦返回，则代码段执行结束，但是yield在返回值以后，会交出CUP的使用权，代码段并没有直接结束，而是在此处中断，当调用send()或者next()方法之后，yield可以从之前中断的地方继续执行。  yield from 实际上起到的是一个信息通道、异常捕获的作用，一方面可以精简代码，不需要手动实现嵌套传值，另一方面还可以避免异常和捕获异常的情况。  代码实现：   1. *#内部生成器* 2. def ave1(): 3. i = [] 4. s,l=0,0 5. i=yield [] 6. while len(i)!=0: 7. s+=sum(i) 8. l+=len(i) 9. i = yield s/l 10. *#外部生成器* 11. def ave2(): 12. yield from ave1() 13. *#实例化* 14. gen=ave2() 15. next(gen) 16. while(True): 17. numlist=list(input('请进行下一轮输入或者回车结束进程：').split(' ')) 18. *#判断结束条件* 19. if numlist==['']: 20. break 21. *#类型转换* 22. for i in range(len(numlist)): 23. numlist[i]=int(numlist[i]) 24. print(gen.send(numlist))   运行结果：    【装饰器函数】  通过预习，我了解到装饰器就是一个闭包，装饰器是闭包的一种应用。，python装饰器就是用于拓展原来函数功能的一种函数，这个函数的特殊之处在于它的返回值也是一个函数，使用python装饰器的好处就是在不用更改原函数的代码前提下给函数增加新的功能。使用时，再需要的函数前加上@demo（本次实验为@timer）即可.  代码实现：   1. import time 2. def timer(func): *#装饰器函数* 3. def inner(\*args,\*\*kwargs): 4. start=time.time() 5. re=func(\*args,\*\*kwargs) 6. print(time.time()-start) 7. return re 8. return inner 9. @timer 10. def func1(\*args,\*\*kwargs): 11. sum=0 12. for i in range(50000): 13. sum=sum+i 14. func1()   运行结果： | | | |
| 结论分析与体会：  通过本次练习，我熟悉了解了Python的循环与条件控制语句，基本掌握了yield和yield from的语法规则和用法，练习了汉诺塔问题和八皇后问题的解决，最后，预习了装饰器函数的用法。 | | | |