任务五

函数和代码复用——基于控制台的购物系统

5.1 任务说明

5.1.1 任务描述

本任务要实现模拟超市控制台的购物系统功能，首先在控制台显示购物系统主菜单，显示效果如图 5-1 所示。

然后根据主菜单选项，提示用户输入对应的数字，如果用户输入“1”，就会显示超市商品信息，效果如图 5-2 所示。

```
==============================
购物系统主菜单
1.显示商品信息
2.购物并付款
0.退出系统
==============================
请输入功能对应的数字：
```

图 5-1 购物系统主菜单

```
==============================
请输入功能对应的数字：1
蒙牛纯奶 ： 2.5元
科迪牛奶 ： 2.5元
伊利纯奶 ： 3元
光明纯奶 ： 2.8元
特化苏牛奶 ： 4.5元
金典牛奶 ： 6元
光明有机奶 ： 3元
德亚全脂牛奶 ： 2.5元
==============================
```

图 5-2 超市商品信息

如果用户输入数字“2”，则提示用户输入购买商品的名称和数量，并且可以多次购买，直到用户输入“q”，则完成购买并计算出需要支付的金额，并退回到主菜单，效果如图 5-3 所示。

如果用户输入数字“0”，首先提示：亲，真的要退出么？（Yes or No），输入 yes(Yes)则显示：谢谢使用！并退出购物系统。输入 no(No)，则返回主菜单，效图如图 5-4 所示。

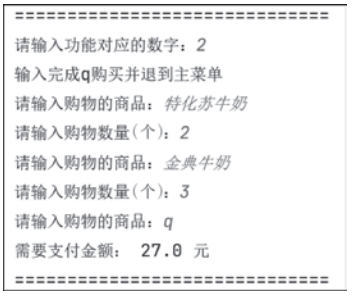

图 5-3　需要支付的金额信息

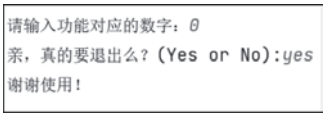

图 5-4　退出功能

5.1.2　任务目标

知识目标

1. 理解什么是自定义函数
2. 掌握函数的定义与调用
3. 掌握函数参数的传递
4. 掌握变量的作用域
5. 了解两种特殊形式的函数

能力目标

1. 学会分析“超市商品购物系统”任务实现的逻辑思路
2. 能够在程序中进行自定义函数的代码编写
3. 能够在程序中调用自定义函数
4. 能够独立完成“超市商品购物系统”程序的代码编写

素养目标

1. 培养学生对自定义函数的理解和应用能力
2. 培养学生程序的思维

5.2　任务相关知识

初识函数

5.2.1　初识函数

函数是一段代码的表示，是具有特定功能的、可重用的语句组。在前面任务的学习中我们已经接触过一些函数，例如，可以打印语句的 print()函数，可以接收用户输入的 input()函数等。Python 内部已有很多功能强大的函数供我们使用，但是 Python 自身提供的函数仅能满足开发过程中的基本需求，很多时候，还需要自定义函数来实现。

为了帮助大家更直观地理解使用函数的好处，下面分别以非函数和函数 2 种形式计算 N!。

例 5-1(a)：以非函数形式计算 N!，代码如下。

```
n = int(input('请输入 n 的值'))
s = 1
for i in range(1,n + 1):
s * = i
print(n,"!= {}".format(s))
```

这段代码只能根据输入 n 的值计算一次阶乘，如果要计算不同数据的阶乘，就需多次运算该程序。

例 5-1(b)：以函数形式计算 N!，代码如下。

```
def fact(n):
      s = 1
for i in range(1,n + 1):
      s * = i
      return s
print("5!= ",fact(5))
print("6!= ",fact(6))
print("7!= ",fact(7))
```

这段代码中首先自定义函数 fact(n)，它的功能是计算 n 的阶乘，如果想分别计算 5!、6! 和 7! 只需调用 3 次 fact(n)函数，并分别传递 5、6 和 7 即可，相比较 5-1(a)而言，使用函数来编程可使程序模块化，既可减少冗余代码又能让程序结构更为清晰；既能提高开发人员编程效率，又方便后期的维护和扩展。

5.2.2　函数的定义与调用

1) 定义函数

Python 中使用 def 关键字来定义函数，其语法格如下。

```
def 函数名称(参数):
    函数体代码
    return 返回值
```

说明如下。

函数要使用 def 关键字标识。

函数名称要遵循标识符的定义规则，不能以数字开头。

函数的参数是可选项，如果设置多个参数，需用逗号隔开。

如果没有参数，可以将其设置为空。

冒号表示函数体的开始标志

函数体代码前要有 Tab 缩进。

如果函数有返回值，使用 return 加返回值，如果没有返回值可以省略 return。

例 5-2：定义函数，实现打印个人信息功能，编程实现该功能。

分析：定义自定义函数 user_info()，其功能是输出个人的 name，age 和 gender。然后调用 user_info()函数，输出个人信息。代码如下。

```
#自定函数输出个人信息
def user_info():
    print("name:Smith")
    print("age:17")
    print("gender:male ")
#调用 user_info()函数
print(user_info())
```

运行结果如下。

```
name:Smith
age:17
gender:male
```

例 5-3：定义函数，实现求两个数的乘积，编程实现该功能。

分析：因为要求两个数的乘积，所以在自定义函数 product(x,y)中需要有两个参数 x,y 接传递的值，计算结果存放在变量 s 中，最后要用 returns 把结果返回，实现代码如下。

```
#自定义函数实现两个数的乘积
def product(x,y):
    s = x * y
    returns
#调用函数并传递参数
print("4 * 5 = ",product(4,5))
```

运行结果如下。

```
4 * 5 = 20
```


调用函数

2) 调用函数

函数在定义后不会立刻执行，直到程序调用时才会执行。调用函数的格式如下。

```
函数名([参数列表])
```

例 5-2 中 user_info()，例 5-3 中的 product(4,5)都是调用函数的语句。实际上，程序在执行“product(4,5)”时经历了 4 个步骤。

① 程序在调用函数的位置暂停执行。

② 将数据 4、5 传递给函数参数。

③ 执行函数体中的语句。

④ 程序回到暂停处继续执行。

说明：函数在定义时可以在其内部嵌套定义另外一个函数，此时嵌套的函数称为外层函数，被嵌套的函数称为内层函数，函数外部无法直接调用内层函数，只能在外层函数中调用内层函数。

例 5-4：定义外层函数 add(x,y)，在 add(x,y)函数中调用内层函数 test()，实现代码如下。

```
#定义外层函数
def add(x,y):
    s = x + y
    print(s)
    #定义内层函数
    def test():
        print('这是一个内层函数')
    #调用内层函数
    test()
#调用外层函数
add(4,5)
```

运行结果如下。

```
9
这是一个内层函数
```

5.2.3　函数参数的传递


函数参数的传递

通常，将定义函数时设置的参数称为形式参数（简称为形参），将调用函数时传入的参数称为实际参数（简称实参）。函数的参数传递是指将实际参数传递给形式参数的过程。函数参数的传递分为 4 种形式。

① 位置参数传递。

② 关键字参数。

③ 默认参数。

④ 参数的打包与解包。

1）位置的参数传递


位置的参数传递

函数在被调用时会将实际参数按照相应的位置依次传递给形式参数，即第 1 个实参传给第 1 个形参，将第 2 个实参传给第 2 个形参，以此类推。

例 5-5：定义一个可以获取两个数之间较小值的函数 get_min(x,y)，并调用 get_min() 函数，代码如下。

```
#定义一个可以获取两个数之间较小值的函数
def get_min(x,y):
    if x < y:
        print(x,'是较小值')
    else:
        print(y,'是较小值')
#调用 get_min()函数传递实参
get_min(5,6)
```

分析：以上函数执行后会将第 1 个实参 5 传给第 1 个形参 x，第 2 个实参 6 传给第 2 个形参 y。

运行结果如下。

```
5 是较小值
```


关键字的参数传递

2) 关键字的参数传递

若函数的参数数量比较多,开发者很难记住每个参数的作用,可以通过"形参=实参"的形式加以指定,可以让函数更加清晰,容易使用,同时也清除了参数的顺序要求。

例 5-6:定义一个打印个人信息的函数 print_user(),调用 print_user()函数,按照"形参=实参"的方式传递,代码如下。

```
#定义一个打印个人信息的函数
def print_user(name,gender,age):
    print('姓名:{}'.format(name))
    print('性别:{}'.format(gender))
    print('年龄:{}'.format(age))
#调用自定义函数并传递实参值
print_user(name = '张小明',gender = '男',age = 16)
```

分析:以上代码执行后,会将"张小明"传给形参 name,将"男"传给形参 gender,将 16 传给形参 age。

运行结果如下。

```
姓名:张小明
性别:男
年龄:16
```

3) 默认参数传递

函数在定义时可以指定形式参数的默认值,所以在调用函数时可以选择是否给带用默认值的形参传递值,若没有给带有默认值的形参传递值,则直接使用该形参的默认值,若给带有默认值的形参传递新值,则使用传递的新值。

例 5-7:定义一个打印个人信息的函数 print_user(),有两个形参带有默认值,调用 print_user()函数时只给其中一个带默认值的形参传值,另一个使用默认值,代码如下。

```
#定义一个打印个人信息的函数
def print_user(name,gender = '男',age = 16):
    print('姓名:{}'.format(name))
    print('性别:{}'.format(gender))
    print('年龄:{}'.format(age))
#调用自定义函数并传递实参值
print_user(name = '李丽',gender = '女')
```

分析:调用 print_user()函数时,将"李丽"传给形参 name,将"女"传给形参 gender,age 使用默认值 16。

运行结果如下。

```
姓名:李丽
性别:女
年龄:16
```

4) 不定长参数

有时需要一个函数能处理有不定个数、不定类型的参数,这些参数为不定长参数,不定

长参数在声明时不会设置所有的参数名称，也不会设置参数的个数。

不定长参数有两种定义方式：一种是不定长参数名称前有一个 * 表示把接收到的参数封装到一个元组中；另一种是不定长参数名称前有两个 * 表示接收键值对的参数值，并将接收到的键-值对添加到一个字典中。

（1）带有一个 * 的不定长参数。

例 5-8：定义一个可以计算任意多个数字和的函数，并输出参数的值和参数的类型，代码如下。

```
#计算任意多个数字的和
def sum_any(*args):
    s=0
    if len(args)>0:
        for i in args:
            s+=i
        print('总和:{}'.format(s))
    print('args 参数值:',args)
    print('args 参数类型:',type(args))
#调用函数
sum_any(10,20)
sum_any(10,20,30)
sum_any(10,20,30,40,50)
```

分析：不定长参数 *args 可以接收任意个数的数字，并将函数调用传入的数字封装到元组中，通过循环遍历，获取每个参数值，然后累加求和。不定长参数不仅可以接收数字类型的参数值，还可接收其他不同类型的参数值。在定义函数时，在不定长参数之前还可以设置其他参数。

运行结果如下。

```
总和:30
args 参数值: (10, 20)
args 参数类型: <class 'tuple'>
总和:60
args 参数值: (10, 20, 30)
args 参数类型: <class 'tuple'>
总和:150
args 参数值: (10, 20, 30, 40, 50)
args 参数类型: <class 'tuple'>
```

（2）带有两个 * 的不定长参数。

例 5-9：每月在发工资之前，公司的财务要计算每个人的实发工资，这里我们使用一种比较简单的工资计算方法：实发工资＝基本工资-个人所得税-个人应缴社保费，下面按照实发工资计算方法编写一个工资计算器函数，代码如下。

```
#计算工资的函数
def pay(basic, **kvargs):
    print('kvargs 参数值:',kvargs)
    print('kvargs 参数类型',type(kvargs))
    #扣除个税金额
```

```
        tax = kvargs.get('tax')
        #扣除社保金额
        social = kvargs.get('social')
        pay = basic - tax - social
        print('实发工资(元):{}'.format(pay))
    #函数调用
    pay(7000,tax = 400,social = 1000)
```

分析: 不定长参数 ** kvargs 接收了函数调用时传入的键值对形式的实参,通过运行结果打印出的 kvargs 参数值和参数类型'dict'可以知道 kvargs 参数将接收到的参数值封装到字典中。

运行结果如下。

```
kvargs 参数值: {'tax': 400, 'social': 1000}
kvargs 参数类型 <class 'dict'>
实发工资(元):5600
```

(3) 拆包。

当一个函数中设置了不定长参数,如果想把已存在的元组、列表、字典传入到函数中,并且能够被不定长参数识别,需要使用拆包的方法。

例 5-10: 定义一个可以计算任意多个数字和的函数,已经存在一个列表 num_list,要求计算出列表中各个数字之和,代码如下。

```
#计算任意多个数字的和
def sum_any( * args):
    s = 0
    if len(args)> 0:
        for i in args:
            s += i
        print('总和:{}'.format(s))
    print('args 参数值:',args)
    print('args 参数类型:',type(args))
#数字列表
num_list = [10,20,30,40,50]
#由于函数设置了不定长参数 * args,所以在传入列表时需要拆包 * num_list
sum_any( * num_list)
```

分析: 拆包的作用是将已定义好的列表从元素中拆分出来,然后传入函数,这样才能被函数中的不定长参数 * args 接收。

运行结果如下。

```
总和:150
args 参数值: (10, 20, 30, 40, 50)
args 参数类型: <class 'tuple'>
```

例 5-11: 定义一个工资计算器函数,实发工资=基本工资-个人所得税-个人应缴社保费,已经存在一个字典,包含个人所得税 tax 和个人应缴社保费 social 的值,要求传入字典,计算出实发工资,代码如下。

```
#计算工资的函数
def pay(basic, **kvargs):
    print('kvargs 参数值:',kvargs)
    print('kvargs 参数类型',type(kvargs))
    #扣除个税金额
    tax = kvargs.get('tax')
    #扣除社保金额
    social = kvargs.get('social')
    pay = basic - tax - social
    print('实发工资(元):{}'.format(pay))
#发工资之前需要扣除的费用
fee = {'tax':400,'social':1000}
#由于 pay 函数设置了不定长参数 **kvargs,所以在传入字典时需要拆包 **fee
pay(7000, **fee)
```

分析：字典类型的拆包方法是在变量名前加 **，拆包之后的参数只能由“**kvargs”这种前边带有两个 * 的不定长参数接收。运行结果如下。

```
kvargs 参数值: {'tax': 400, 'social': 1000}
kvargs 参数类型 <class 'dict'>
实发工资(元):5600
```

5）混合传递

前面介绍的参数传递的方式在定义函数或调用函数时可以混合使用，但是需要遵循一定的优先级规则，这些方式优先组从高到低依次为按位置参数传递、按关键字参数传递、按默认参数传递，按不定长参数传递。

在定义函数时，带有默认值的参数必须位于普通参数（不带默认值或标识的参数）之后，带有“*”标识的参数必须位于带有默认值的参数之后，带有“**”标识的参数必须位于带有“*”标识的参数之后。

例 5-12：定义一个混合了多种形式的参数的函数，代码如下。

```
#定义一个带有多种形式参数的函数
def test(a,b,c = 33, *args, **kvargs):
    print(a,b,c,args,kvargs)
#调用函数
test(1,2)
test(1,2,3)
test(1,2,3,4)
test(1,2,3,4,e = 5)
```

分析：test()函数共有 5 个参数，以上代码多次调用 test()函数并传入不同数量的参数。

执行 test(1,2)函数时，该函数接收到实参 1 和实参 2，这两个实参被 a 和 b 接收；其余 3 个参数 c，*args，**kvargs 没有接收到实参，都使用默认值（33,(),{}）。

执行 test(1,2,3)函数时，该函数接收到实参 1～实参 3，前 3 个实参被普通参数 a,b 及带默认值的参数 c 接收；其余 2 个形参，都使用默认值 ()和{}。

执行 test(1,2,3,4)函数时，该函数接收到实参 1～实参 4，4 个实参被普通参数 a,b,c

和 * args 接收；形参 ** kvargs 没有接到实参，故使用默认值{}。

执行 test(1,2,3,4,e=5)函数时，该函数接收到实参 1～实参 4 和形参 e 关联的实参 5，所有的实参被相应的形参接收。

运行结果如下。

```
1 2 33 () {}
1 2 3 () {}
1 2 3 (4,) {}
1 2 3 (4,) {'e': 5}
```


函数的返回值

5.2.4 函数的返回值

函数中的 return 语句会在函数结束时将数据返回程序，同时让程序回到函数被调用的位置继续执行。

例 5-13：请用程序实现用函数判断三个数字作为边长能否构成三角形，并将判断结果返回，返回值说明如下。

三角形三边长必须大于零，不满足则返回数字-1，表示数据不合法。

任意两边之和必须大于第三边。

不满足则返回数字 0，表示不能组成三角形。

满足则返回数字 1，表示能组成三角形。

分析：定义 is_triangle()函数要有三个形参来接收三个数字表示三条边长，首先判断三条边长如果都大于 0，并且满足任意两边之和大于第三边则返回 1；如果三条边长都大于 0，但是不能满足任意两边之和大于第三边则返回 0；如果三条边长不满足都大于 0，则返回－1，实现代码如下。

```
#定义函数判断三个数字作为边长是否能组成角形
def is_triangle(a, b, c):
    if a > 0 and b > 0 and c > 0:
        if (a + b > c) and (a + c > b) and (b + c > a):
            return 1
        else:
            return 0
    else:
        return - 1
#调用函数并传递三个数
print(is_triangle(3,4,5))
print(is_triangle(2,1,1))
print(is_triangle( - 1,1,1))
```

运行结果如下。

```
1
0
- 1
```

以上定义的函数只返回了一个值，如果函数使用 return 语句返回了多个值，那么这些

值将被保存到元组中。

例 5-14：定义一个控制游戏角色位置的函数，该函数使用 return 语句返回游戏角色目前所处的位置的 x 坐标和 y 坐标，实现代码如下。

```
#定义 move()函数,包含三个形参分别用来接收 x 坐标、y 坐标和移动距离
def move(x,y,step):
    nx = x + step
    ny = y - step
    return nx,ny
result = move(100,100,60)
print('result 的值',result)
print('result 的类型',type(result))
```

分析：定义 move()函数，包含三个参数，分别接收 x 坐标、y 坐标和移动的距离，最后返回移动后的 x 坐标和 y 坐标。返回的两个值将被保存在元组中。

运行结果如下。

```
result 的值 (160, 40)
result 的类型 <class 'tuple'>
```

5.2.5　变量作用域


局部变量和全局变量

变量作用域指的是定义的变量在代码中可以使用的范围，根据变量的使用范围可以把变量分为局部变量和全局变量两种。从这两种变量的名称中可以看出，局部变量只能在某个特定的范围内使用，而全局变量可以在全局范围内使用。

1）局部变量

局部变是指在函数内部定义的变量，它只能在函数内部使用，函数执行结束之后局部变量会被释放，此时无法进行访问。例如，在 a 函数中定义的变量，在 b 函数中不能使用。同理，在 b 函数中定义的变量，在 a 函数中也不能使用。

例 5-15：在 test()函数中定义了一个局部变量 number，分别在函数内和函数外使用 number，实现代码如下。

```
def test():
    number = 5
    print(number)
#调用 test()函数
test()
#在函数外变量访问 number
print(number)
```

分析：程序在定义了 test()函数后调用该函数（给局部变量 number 赋值 5，并且打印 number 的值），结果成功访问了 number；接下来程序在 test()函数外直接用 print()函数访问局部变量 number，出现异常信息，说明函数外部无法访问局部变量。

运行结果如下。

```
5
Traceback (most recent call last):
File "D:/书稿/任务五源代码/5 - 15.py", line 5, in <module>
     print(number)
NameError: name 'number' is not defined
```

不同函数内的局部变量可以定义相同的名字,它们相互独立,互不影响。

例 5-16:定义打印两个人的信息的函数 print_user1()和 print_user2(),调用它们输出不同的个人信息,实现代码如下。

```
#定义 print_user1()函数
def print_user1():
    #局部变量
    name = 'Smith'
    age = 16
    print('name:{},age:{}'.format(name,age))
#定义 print_user2()函数
def print_user2():
    name = 'Michelle'
    age = 16
    print('name:{},age:{}'.format(name, age))
#调用 print_user1()函数
print_user1()
#调用 print_user2()函数
print_user2()
```

分析:print_user1()打印了 Smith 的个人信息,在函数中定义了两个局部变量 name 和 age。print_user2()函数打印了 Michelle 的个人信息,在函数中定义了两个与 print_user1()函数相同的局部变量 name 和 age。当分别调用两个函数时,都能够正确打印出函数内的变量值,说明局部变量在对应的函数内有效,不同函数内的局部变量可以同名并且互不影响。

运行结果如下。

```
name:Smith,age:16
name:Michelle,age:16
```

2) 全局变量

函数外定义的变量称为全局变量。建议定义全局变量时,在全局变量的名称前添加“g_”前缀,这样可以避免在函数内部局部变量与全局变量重名,能够清晰地分辨出全局变量和局部变量,作用范围:可以在不同函数中使用同一个全局变量。

例 5-17:在程序中定义全局变量 n,在自字义函数 test_1()和自定义函数 test_2()中分别访问全局变量,实现代码如下。

```
#定义全局变量 n
n = 10
#定义函数 test_1()
def test_1(x):
    s = n + x
    return s
```

```
#定义函数 test_2()
def test_2(x):
    s = n * x
    return s
#调用 test_1()函数
print(test_1(2))
#调用 test_2()函数
print(test_2(2))
```

分析：在函数外定义了一个全局变量 n，分别在 test_1()和 test_2()函数的计算过程中使用了全局变量 n 的值，说明全局变量是一个公共的共享变量，在不同函数内部都可以使用。

运行结果如下。

```
12
20
```

3）带 global 关键字的变量

函数只能访问全局变量，而无法直接修改全局变量，如果在函数内修改全局变量的值，需要在函数内使用 global 关键字声明全局变量。

格式：global 变量名。

例 5-18：在程序中定义全局变量 n，并赋值 10，在函数 test_1()修改变量 n 的值为 20，然后在 test_2()中访问变量 n，实现代码如下。

```
#定义全局变量 n
n = 10
#定义函数 test_1()
def test_1(x):
    global n
    n = 20
    s = n + x
    return s
#定义函数 test_2()
def test_2(x):
    s = n * x
    return s
print('n = ',n)
print(test_1(2))
print('n = ',n)
print(test_2(2))
```

分析：在调用 test_1()之前先打印了没有经过修改的全局变量 n 的值为 10，然后调用 test_1()函数，在 test_1()函数中通过 global 关键字声明了全局变量 n，然后修改了 n 的值为 20，在 test_1()函数中使用 n＝20 的值进行了计算。test_1()函数执行结束后，又打印了变量 n 的值，依然是修改之后的值 20，最后调用 test_2()函数，在 test_2()函数中 n 的值为 20 参与计算。说明在函数内修改全局变量的值会使全局变量的值被彻底更改。

运行结果如下。

```
n = 10
22
n = 20
40
```

4) 带 nonlocal 关键字的变量

使用 nonlocal 关键字可以在局部作用域中修改嵌套作用域中声明的变量。

格式：nonlocal 变量。

例 5-19：在函数 test()中定义局部变量 number=10,在嵌套内层函数 test_in()中使用 nonlocal 关键字修饰了 number,并修改 number 的值为 20,实现代码如下。

```
#定义函数 test()
def test():
    number = 10
    #定义内层嵌套函数 test_in()
    def test_in():
        #用 nonlocal 声明变量 number,并修改它的值
        nonlocal number
        number = 20
    test_in()
    print('number = ',number)
#调用 test()函数
test()
```

分析：程序执行 test_in()函数时成功地修改了变量 number 的值,并且打印了修改后的 number 的值。

运行结果如下。

```
number = 20
```

5.2.6 两个特殊形式的函数

除了前面按标准定义的函数外,Python 还提供了两种具有特殊形式的函数：递归函数和匿名函数。


递归函数

1) 递归函数

函数在定义时可以直接或间接地调用其他函数。若函数内部调用了自身,则这个函数被称为递归函数。递归函数常用于解决结构相似的问题,它采用递归公式,将一个复杂的大型问题转化为与原问题结构相似的、规模较小的若干子问题,之后对最小化的子问题求解,从而得到原问题的解。

递归函数在定义时需要满足 2 个基本条件,一个是递归公式,另一个是边界条件。其中,递归公式用来求解原问题或相似的子问题的结构；边界条件是最小化的子问题,也是递归终止的条件。

递归函数的执行可以分为以下 2 个阶段。

递推：基于上一次的运算结果执行本次的递归。

回溯：遇到终止条件时,则沿着递推结果返向一级一级地得到结果。

递归函数的一般定义格式如下。

```
def 函数名([参数列表]):
  if 边界条件:
      return 结果
  else:
      return 递归公式
```

例 5-20：递归最经典的应用是求阶乘，求正整数 n!（n 的阶乘）问题根据 n 的取值可以分为以下 2 种情况。

（1）当 n＝1 时，所得结果为 1。

（2）当 n＝2 时，所得结果为 n＊(n－1)!。

实现代码如下。

```
#定义计算阶乘的函数
def func(num):
    if num == 1:
        return 1
    else:
        return num * func(num - 1)
num = 10
#调用 func()函数返回阶乘值
result = func(num)
print('10!= {}'.format(result))
```

分析：利用递归求阶乘时，n＝1 是边界条件，n＊(n－1)! 是递归公式。

运行结果如下。

```
10! = 3628800
```

例 5-21：编程定义一个函数，传入任意字符串，可以实现翻转字符串的功能，实现代码如下。

```
#定义递归函数
def rvs(s):
    if s  == "":
        return s
    else:
        return rvs(s[1:]) + s[0]
#输入任意字符串
s = input('请你入字符串:')
print('原字符串为:',s)
print('翻转后字符串为',rvs(s))
```

分析：s 为空串即 s＝""是边界条件，rvs(s[1:])＋s[0]是递归公式。

运行结果如下。

```
请你入字符串:abcd
原字符串为: abcd
翻转后字符串为 dcba
```


匿名函数

2) 匿名函数

匿名函数是一类无须定义标识符的函数,它与普通函数一样可以在程序的任何位置使用。Python 中使用 lambda 关键字定义匿名函数,它的格式如下。

```
lambda <形式参数列表>:<表达式>
```

结合以上的语法格式可知,匿名函数与普通函数主要有以下区别。

普通函数在定义时有名称,而匿名函数没有名称。

普通函数的函数体中包含多条语句,而匿名函数的函数体只能是一个表达式。

普通函数可以实现比较复杂的功能,而匿名函数可实现的功能比较简单。

普通函数能被其他程序使用,而匿名函数不能被其他程序使用。

匿名函数会将表达式的结果自动返回,不需要使用 return 关键字。

定义好的匿名函数不能直接使用,最好使用一个变量保存它,以便后期可以随时使用这个函数。

例 5-22:利用匿名函数计算两个数字的和,实现代码如下。

```
#定义匿名函数求两个数字的和
sum = lambda x,y:x + y
#调用匿名函数
print(sum(10,20))
```

分析:将计算两个变量的和的匿名函数赋值给一个变量,通过变量调用匿名函数,使用起来非常灵活方便。

运行结果如下。

```
30
```

例 5-23:编程实现根据不同的匿名函数对两个数进行求值计算,实现代码如下。

```
#定义函数
def x_y_js(x,y,s):   #s 是匿名函数
    print('x = ',x)
    print('y = ',y)
    result = s(x,y)  #对 x 和 y 两个参数使用匿名函数进行计算
    print('result = ',result)
#传入的匿名函数用于求两个数的和
x_y_js(3,5,lambda x,y:x + y)
#传入的匿名函数用于求两个数的积
x_y_js(3,5,lambda x,y:x * y)
```

分析:定义函数 x_y_js 时,在函数内并没有定义计算逻辑,而是在函数参数中设置匿名函数作为参数,在函数体内采用匿名函数的计算逻辑。

运行结果如下。

```
x =  3
y =  5
result =  8
```

```
x = 3
y = 5
result = 15
```

5.3　任务设计思路

根据购物系统要完成的功能首先设计自定义函数 main()，main()函数的功能是：显示购系统主菜单，提示用户输入不同选项，根据用户输入的数字，执行不同的自定义函数完成相应功能。如果用户输入数字 1，则执行用来显示商品信息的自定义函数 show_goods()；如果用户输入数字 2，则执行购物功能的自定义函数 gouwu()；如果用户输入数字 0，则会再次确认用户是否要真的退出该购物系统，为了使程序界面更友好一些，无论用户输入大小写的 yes 都会退出该系统，无论输入大小写 no 都可以重新返回主菜单。

要显示商品信息先在自定义函数 all_goods()中定义所有的商品信息，然后通过 show_goods()函数去调用 all_goods()函数。

gouwu()函数的功能是提示用户输入购买商品的名称和购买数量，根据用户输入的商品名称和数量计算出用户应付金额，如果用户输入的商品名称不正确或者输入的数量不是数字，系统会有错误提示。用户可以多次输入商品名称和数量，直到输入“q”后完成本次交易并返回主菜单界面。

5.4　任务实施

步骤一：首先编写主程序，用来调用 main()函数，实现代码如下。

```
if __name__ == '__main__':
    main()
```

步骤二：编写 main()函数，用来实现显示购物系统主菜单，提示用户输入不同选项，根据用户输入的数字，执行不同的自定义函数完成相应功能，实现代码如下。

```
def main():
     while True:
        print_menu()                              # 打印主菜单界面
        key = input("请输入功能对应的数字:")      # 获取用户输入的序号
        if key == '1':                            # 显示商品信息
            show_goods()
       elif key == '2':                           # 执行购物功能
            gougw()
       elif key == '0':
          quit_confirm = input('亲,真的要退出么?(Yes or No):').lower()
          if quit_confirm == 'yes':
               print("谢谢使用!")
               break # 跳出循环
            elif quit_confirm == 'no':
```

```
                continue
            else:
                print('输入有误!')
```

步骤三：定义主菜单界面，实现代码如下。

```
def print_menu():
    print('=' * 30)
    print('购物系统主菜单')
    print('1.显示商品信息')
    print('2.购物并付款')
    print('0.退出系统')
    print('=' * 30)
```

步骤四：定义 all_goods()函数用来保存商品信息，定义 show_goods()函数用来显示在 all_goods()函数中保存的所有商品，实现代码如下。

```
#定义商品信息
def all_goods():
    goods = {"蒙牛纯奶": 2.5, "科迪牛奶": 2.5, "伊利纯奶": 3, "光明纯奶": 2.8, "特仑苏牛
奶": 4.5,"金典牛奶": 6, "光明有机奶": 3, "德亚全脂牛奶": 2.5}
    return goods

#显示商品信息
def show_goods():
    for x, y in all_goods().items():
        print(x, ":", str(y) + "元")
```

步骤五：定义函数购物函数 gouwu()，用来提示用户输入购物商品名称和数量，并调用 total()函数，传递购物商品的单价和数量计算出相应金额。用户可以多次购买，直到输入"q"完成交易，返回主菜单，实现代码如下。

```
#输入购买商品信息并计算付款金额
def gougw():
    goods_dict = {}
    print("输入完成 q 购买并退到主菜单")
    while True:
        goods_name = input("请输入购物的商品:")
        if goods_name == 'q':
            break
        if goods_name in [g_name for g_name in all_goods().keys()]:
            goods_num = input("请输入购物数量:")
            if goods_num.isdigit():
                goods_dict[goods_name] = float(goods_num)
            else:
                print('商品数量不合法')
        else:
            print('请输入正确的商品名称')
    total(goods_dict)
#计算应付金额
```

```
def total(goods_dict):
    count = 0
    for name, num in goods_dict.items():
        total_money = all_goods()[name] * num
        # 总金额
        count += total_money
    print("需要支付金额:", count, "元")
```

5.5　任务小结

本任务实现了超市购物系统的简单功能，用户根据主菜单提示输入不同的选项，从而执行不同的自定义函数来完成相应的功能。通过本任务的学习可以进一步地掌握自定义函数的调用、自定义函数参数的传递、变量的作用域以及自定义函数返回值的应用等，为以后的学习打下坚实的基础。

5.6　技能训练

一、单选题

1. 以下关于函数优点的描述中，错误的是(　　)。

A. 函数便于阅读　　B. 函数可以使程序更加模块化

C. 函数可以减少代码重复　　D. 函数可以表现程序复杂度

2. Python 中定义函数的关键字是(　　)。

A. def　　B. define　　C. function　　D. defun

3. 关于函数定义，以下形式错误的是(　　)。

A. deffoo(a,b)　　B. deffoo(a,b=10)

C. deffoo(a, * b)　　D. deffoo(* a,b)

4. 以下关于 Python 函数说法错误的是(　　)。

```
def func(a, b):
    c = a ** 2 + b
    b = a
    return c

a = 10
b = 100
c = func(a, b) + a
```

A. 执行该函数后，变量 b 的值为 100　　B. 执行该函数后，变量 a 的值为 10

C. 该函数名称为 func　　D. 执行该函数后，变量 c 的值为 200

5. 关于 Python 的 lambda 函数，以下选项中描述错误的是(　　)。

A. lambda 函数将函数名作为函数结果返回

B. f=lambda x,y: x+y 执行后，f 的类型为数字类型

C. lambda 用于定义简单的、能够在一行内表示的函数

D. 可以使用 lambda 函数定义列表的排序原则

6. 关于递归函数基例的说明,以下选项中错误的是(　　)。

A. 递归函数的基例决定递归的深度　　B. 每个递归函数都只能有一个基例

C. 递归函数必须有基例　　D. 递归函数的基例不再进行递归

7. 下列程序的输出结果为(　　)。

```
def f(a, b):
    a = 4
    return a + b

def main():
    a = 5
    b = 6
    print(f(a, b), a + b)

main()
```

A. 10 10　　B. 11 10　　C. 11 11　　D. 10 11

8. 关于以下代码的描述中,错误的是(　　)。

```
def fact(n):
    s = 1
    for i in range(1,n + 1):
        s * = i
return s
```

A. 代码中 n 是可选参数　　B. fact(n) 函数功能为求 n 的阶乘

C. s 是局部变量　　D. range() 函数是 Python 内置函数

9. 以下不属于 Python 的内置函数的选项是(　　)。

A. abs()　　B. sum()　　C. input()　　D. get()

10. 以下代码的输出结果是(　　)。

```
def fibRate(n):
    if n <= 0:
        return - 1
    elif n == 1:
        return - 1
    elif n == 2:
        return 1
    else:
        L = [1, 5]
        for i in range(2,n):
            L.append(L[ - 1] + L[ - 2])
        return L[ - 2] % L[ - 1]
print(fibRate(7))
```

A. 0.6　　B. 28　　C. −1　　D. 1

二、实操题

1. 按下面的要求，输出两个正整数的最大公约数。

（1）定义函数 common_divisor(num1, num2)，该函数返回两个正整数的最大公约数；函数有两个参数，分别对应两个正整数；有 1 个返回值，代表求得的最大公约数。

（2）接收两个输入值，调用函数 common_divisor，输出最大公约数。

2. 按下面的要求，计算 n 个自然数的立方和。

（1）定义函数 sumOfSeries。

（2）有 1 个参数 n 表示自然数的个数。

（3）有 1 个返回值，返回自然数的立方和。

（4）调用函数，输出 5 个自然数的立方和。

3. 判断以下长度的三条线段能否构成一个三角形。需要满足两条规则：

（1）三角形的三条边长必须大于零。

（2）任意两边之和必须大于第三边。

编程实现：用函数判断三个数字能否构成三角形，并将判断结果返回。

4. 密码是账户的重要安全保障，涉及安全问题，太简单的密码容易被猜到或破解。

请用函数实现：一个校验密码强度的函数，用于提醒用户在注册时，密码是否足够安全。以下为密码强度校验规则：

（1）密码长度在 6 位及以上，强度＋1，在 8 位及以上，强度＋2，12 位及以上，强度＋4

（2）有大写字母，强度＋2

（3）除字母外，还包含数字，强度＋2

（4）有除字母、数字以外字符强度＋2

5. 古人经常使用藏头诗，隐晦地表达自己的想说的话，既有诗意，又能传递信息，比如下面这两首诗：

芦花丛中一扁舟，
俊杰俄从此地游.
义士若能知此理，
反躬逃难可无忧.

我画蓝江水，
爱晚亭上枫.
秋月溶溶照，
香烟袅袅绕.

请用函数实现：将其隐含的意思找出来。