第5章

函　　数

学习目标：

- 学会使用用户自定义的函数实现代码的重用和抽象。
- 学会合理选择参数传递的方式，满足实际问题的求解需要。
- 学会使用 Python 语言特有的嵌套定义，掌握不同作用域变量的使用方法。

5.1　函数的定义与调用

函数是将一段代码单独取出命名，按名称多次使用这段代码的一种机制。取出代码并命名的过程称为函数定义，按定义的名称使用函数代码的过程称为函数调用。Python 中已经预先定义好了一批函数，这些函数分别放在内置库、标准库或第三方库模块中，称为预定义函数。另一方面，设计者可以根据设计任务的需要，利用 Python 的函数定义机制定义自己需要的函数，称为自定义函数。

使用函数进行程序设计的方法称为模块化设计方法，适合较复杂问题的求解。复杂的问题直接编程比较困难，可以将问题分解成功能相对简单的子问题，如果子问题仍然比较复杂可以继续分解，直到子问题足够简单，然后直接编程求解。子问题的求解代码会被编成函数存放在模块库中，库中的函数就像积木块或零件，可以被组装起来，去解决功能更复杂的问题。函数代码会不断被重复使用，其中的错误容易被发现并去除，代码的质量较高，所以，使用函数求解问题成本低、可靠性好。

模块化的程序设计方法的重点是构建适合重复使用的函数，如何分解问题是关键。分解问题时需要从功能出发，分析子问题是否具有相对独立的功能。下面看一个示例。

组合数 $C_m^n=\frac{m!}{n!\ (m-n)!}$，求 C_m^n 可以转换为 3 个求阶乘 m!、n! 和(m－n)! 的子问题来完成。计算 k 阶乘的公式为 k! ＝ 1×2×3×…×k，这个公式以前已经介绍过计算方法，如果将 m，n，m－n 的值分别赋值给 k 就可以利用同样的代码分别求出这 3 个阶乘，然后，使用乘除运算可以求出组合数 C_m^n。由于子问题功能是相对独立的，求阶乘的子问题编成函数后，可以重复使用 3 次。如果将问题分解成分子和分母两个子问题，则无法重用。这是因为，这种子问题的分解严重依赖于原问题的具体求解步骤，离开了原问题，

子问题就没有存在的意义。将求阶乘代码设计成函数可以保证代码的重用性，即设计一次可以多次使用。求阶乘问题不只是用于组合数计算，其他问题中也经常遇到，为此，Python 在 math 标准库模块中，将它设计为了预定义函数 factorial。

5.1.1 函数的定义

自定义一个求阶乘的函数需要使用函数的定义机制。

Python 定义函数的语法规则如下：

```
def <函数名>([形式参数列表]):
    <函数体>
```

＜函数体＞是完成相对独立功能的代码段，＜函数名＞是调用代码段时使用的名称，名称需要符合标识符的取名规则。圆括号内是形式参数表，符号[]表示形参列表可以省略，这样定义的函数是无参的。多个形参间以逗号分隔，每个形参都是一个变量，用来引入函数体要加工的数据值。形参变量的值在调用函数时提供，称为实际参数，实参在调用函数时会赋值给对应位置的形参变量，这个过程称为参数传递。def 保留字和冒号不能省略，函数体的每行代码需要缩进相同的位置，函数体执行时如果遇到 return 语句或相同缩进位置的最后一行，函数会结束。函数结束时，会计算 return 后的表达式的值返回给调用者作为函数结果，执行无值的 return 语句或函数体的最后一行时，会返回 None 值作为函数结果。

【例 5-1】 自定义阶乘函数，求组合数 C_m^n。

首先编写一个求阶乘的函数：

```
#liti5-1-1.py
def factor(k):
    f=1                                #第二行
    for i in range(1,k+1):
        f=f*i
    return f
factor(6)
```

程序运行结果如下：

```
720
```

factor 是自定义的求阶乘的函数名，第 2 行开始到 return f 是函数体。函数体用来求 k 的阶乘并保存到变量 f 中，return f 语句会将变量 f 的值作为函数结果返回给调用者，形参 k 的值是由函数调用者调用函数时以实参形式传递而来的。

factor(6)这一行是程序的主控部分，相比于函数体没有向右缩进。6 是实参，执行函数体前会传递给形参 k。720 是函数执行的结果，会返回给主控部分并显示。

下面是求组合数 C_m^n 的程序，函数定义部分相同，主控部分修改如下：

```
#liti5-1-2.py
def factor(k):
    f=1
    for i in range(1,k+1):
        f=f*i
    return f
m=int(input("请输入 m 的值:"))
n=int(input("请输入 n 的值:"))
c=factor(m)//(factor(n)*factor(m-n))
print("C({},{})={}".format(n,m,c))
```

程序运行结果如下：

```
请输入 m 的值:5
请输入 n 的值:3
C(3,5)=10
```

首先执行程序中的函数定义命令 def，生成函数对象 factor，但不会执行 factor 函数体。接着执行函数体后的主控部分，在倒数第 2 行的赋值操作中，factor(m)、factor(n)和 factor(m－n)分别以实参 m、n、m－n 调用函数 factor，调用的顺序是按赋值操作右边表达式运算的执行顺序进行，得到的结果赋值给变量 c。最后调用 print 函数显示 c，显示的格式是由带 format()方法的格式字符串"C({},{})＝{}"设置，串中的花括号称为槽(slot)，用来设置 format()方法中对应位置实参值的显示位置和格式，显示时，format 的实参 n、m、c 的值会按顺序出现到格式串的 3 个槽的位置上。

5.1.2 函数的调用与返回

已定义好的函数的整个执行过程分为函数调用和返回两个阶段。已定义好的函数可以被多次调用，每次调用都会返回不同的函数结果。函数的调用格式如下：

```
<函数名>([<实际参数列表>])
```

＜实际参数列表＞的成员个数由函数定义时的形式参数的个数决定，要保证每个形参变量都能得到实际参数值。实际参数的形式可以是表达式、常量或变量，参数传递时会将实际参数计算后赋值给对应位置的形参变量。

表达式中的函数调用可以看作是表达式的一步运算，函数返回值作为运算结果会继续参与表达式的下一步运算。函数调用作为实际参数去调用函数，称为函数的复合运算，例如，print(factor(6))，先调用 factor(6)函数，得到的结果作为实参值去调用 print 函数。

函数调用的结果称为函数的返回值，由函数体中的 return 语句提供，return 语句的格式如下：

```
return [<结果表达式>]
```

return 语句会结束函数调用，将＜结果表达式＞的值作为函数的返回值。返回值可以是简单的值，如数值、字符串、逻辑值等，也可以是组合类型的值，如元组、列表等，还可以是 Python 对象，如函数对象、类型对象、类对象等。

例如：

return max 返回的是函数对象 max。

return int 返回的是类型对象 int。

return 3,4,5 返回的是元组(3,4,5)。

return 返回的是空对象 None，返回 None 值表示没有返回值。

函数体可以包含多个 return 语句，将它们处于不同的执行路径上，当执行到其中一个 return 语句时，函数调用就会结束，并返回该 return 语句的＜结果表达式＞的值。

【例 5-2】 编写函数求一元二次方程 $ax^2+bx+c=0$ 的解。

一元二次方程的解有 3 种情况：两个不相等的实数解，一个实数解和无实数解。通过判断 $\Delta=b^2-4ac$ 的值是大于零、等于零还是小于零，来决定是哪种情况。编写该函数时，系数 a、b、c 作为函数的形式参数，让调用者提供系数值，函数名定义为 rootofqe。

```
#liti5-2.py
def rootofqe(a,b,c):
    d=b*b-4*a*c
    if d>0:
        return (-b+d**0.5)/(2*a),(-b-d**0.5)/(2*a)      #返回两个不相等的实数解
    elif d==0:
        return -b/(2*a)                                  #返回一个实数解
    else:
        return None                                      #返回 None,表示无实数解
a=int(input("请输入 a 的值:"))
b=int(input("请输入 b 的值:"))
c=int(input("请输入 c 的值:"))
print(rootofqe(a,b,c))
```

程序运行结果如下：

```
请输入 a 的值:2
请输入 b 的值:5
请输入 c 的值:3
(-1.0, -1.5)
```

提供不同的实参，再次运行程序的结果如下：

```
请输入 a 的值:1
请输入 b 的值:4
请输入 c 的值:4
-2.0
```

提供无实数解的实参，第三次运行程序的结果如下：

```
请输入 a 的值:2
请输入 b 的值:4
请输入 c 的值:4
None
```

程序的主控部分调用函数 rootofqe 并显示结果，主控部分之前是 rootofqe 函数的定义部分，这种先后顺序是必须的，**任何函数必须先定义后调用**，否则会因找不到函数而调用出错。

rootofqe 的函数体中出现了 3 个 return 语句，提供了 3 种不同情况下的函数结果。当变量 d 大于零、等于零、小于零时，会选择不同的执行路径到达相应的 return 语句，分别提供两个不相等的实数解、一个实数解、无实数解 3 种情况下的求解结果。

当为 d 小于零的情况时，执行的路径是最后的"else:"后面的 return 语句，返回值是 None，表示无实数解。如果 return 后面不写 None 值，或者去除"else:"后这条执行路径，函数也能处理无实数解的情况，并返回 None 值。

5.1.3 函数嵌套调用和递归调用

1. 函数嵌套调用

在一个函数的函数体中调用函数称为函数的嵌套调用，嵌套调用可以有多层，如图 5-1 所示。

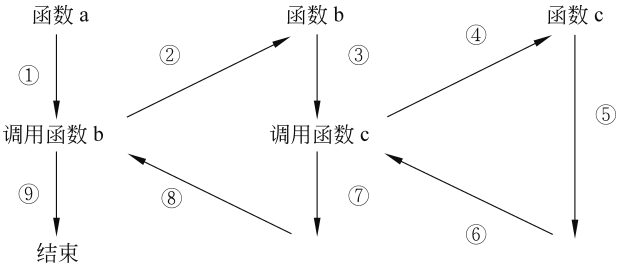

图 5-1 两层函数嵌套调用的过程

图 5-1 的函数嵌套调用分为两层：函数 a 调用了函数 b，函数 b 调用了函数 c，执行顺序如图中数字所示。函数 a 调用函数 b 时，必须等待函数 b 调用结束，才能从调用位置继续执行，函数 a 暂停执行的位置称为断点(breakpoint)，函数 a 暂停期间，运行状态会一直保持不变。当函数 b 调用函数 c 时，也必须保持状态等待，直到函数 c 调用结束后返回断点继续执行。当函数 a 调用结束时，会回到主控部分继续执行，整个过程呈现的是一种多层的、先入后出的函数调用。

【例 5-3】 函数的嵌套调用过程。

```
#liti5-3.py
def c():
    print("In c")
```

```
def b():
    for j in range(3):
        print("In B j=",j)
        c()
def a():
    for i in range(2):
        print("In A i=",i)
        b()
a()
```

程序运行结果如下：

```
In A i= 0
In B j= 0
In c
In B j= 1
In c
In B j= 2
In c
In A i= 1
In B j= 0
In c
In B j= 1
In c
In B j= 2
In c
```

函数 a 循环调用了两次函数 b，函数 b 循环调用了 3 次函数 c，函数 a 每次调用函数 b 时，函数 b 都需要调用 3 次函数 c 才能返回。整个程序执行过程中，函数 c 总共被调用了 6 次。每次调用函数 b 或函数 c 时，作为调用者的函数 a 和函数 b 中的变量 i、j 的值会在暂停执行期间保持不变，等到被调用函数 b 或函数 c 返回时，函数 a 和函数 b 会使用变量 i、j 的值继续执行。

函数中的变量不会总能保持值不变，每次函数 b 被调用时，其内部的变量 j 都会重新创建，调用结束时变量 j 会自动删除，这个过程称为变量 j 的生存期，变量处在生存期之外时，值会丢失。而且，不同函数中的变量会有不同的生存期，变量 i 的生存期就是在函数 a 被调用期间。函数中的变量还有一个特性，即使处于生存期中，在定义它的函数之外不能被引用，例如，函数 b 不能引用函数 a 中的变量 i，变量的这种引用范围的限制称为变量的作用域。Python 中的变量有不同的生存期和作用域，在 5.3 节中会详细介绍。

【例 5-4】 编写求组合数 C_m^n 的函数。

组合数 C_m^n 的计算依赖于阶乘函数 factor，而组合数的计算程序自身也适合编写为函数，以便作为可重用的部件来使用。组合数函数定义两个形式参数 m、n，函数名为 combin。

```
#liti5-4.py
```

```
def factor(k):
    f=1
    for i in range(1,k+1):
        f=f*i
    return f
def combin(m,n):
    c=factor(m)//(factor(n)*factor(m-n))
    return c
m=int(input("请输入 m 的值:"))
n=int(input("请输入 n 的值:"))
print("C({},{})={}".format(n,m,combin(m,n)))
```

程序运行结果如下：

```
请输入 m 的值:6
请输入 n 的值:4
C(4,6)=15
```

例 5-4 使用了函数嵌套调用的方法。程序中的主控部分调用函数 combin 前，已经先后定义了函数 factor 和函数 combin。调用函数 combin 时会分 3 次嵌套调用函数 factor，得到 3 个阶乘值后，再继续执行函数 combin。通过计算得到组合数后，函数 combin 结束并将结果返回到主控程序继续执行，程序按格式显示结果后结束。

函数必须先定义后调用，将例 5-4 中的函数 factor 和函数 combin 的定义位置互换，不会影响程序的执行，如果将函数 factor 或函数 combin 的定义放在主控程序之后，则程序会因找不到函数而调用出错。

2. 函数递归调用

一个函数在函数体中嵌套调用该函数自身的过程称为递归调用。无条件的递归调用时，函数会不断嵌套调用自己而无法结束。因此，递归函数要为嵌套调用的语句设置一个条件，当条件满足则递归调用，当条件不满足则函数不再嵌套调用，直接结束并返回上一层。这样，多层嵌套的递归调用就能逐层结束并返回，调用过程才能正常结束。

【例 5-5】 编写计算 $S=\sum_{i=1}^{n} i$ 的函数。

```
#liti5-5.py
def sum(n):
    if n==1:
        return 1
    else:
        return sum(n-1)+n
print("1+2+…+{}={}".format(6,sum(6)))
```

程序运行结果如下：

```
1+2+…+6=21
```

sum(6)的计算过程如图 5-2 所示。

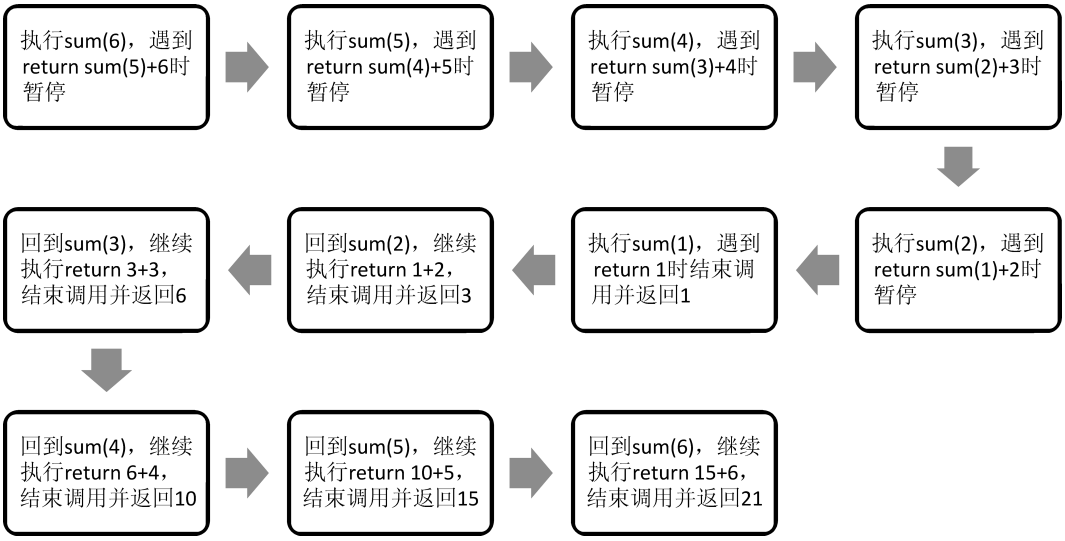

图 5-2　sum(6)的递归调用过程

函数 sum 的递归是有条件限制的，当 n 等于 1 时不递归，直接返回结果 1。主控部分调用了 sum(6)得到返回的值 21，也就是 1＋2＋3＋4＋5＋6 的结果，然后按格式显示结果后结束程序。

【例 5-6】　编写计算 S＝n! 的函数。

n!表示求 n 的阶乘，阶乘有一种递归性质：

$$n!=\begin{cases}(n-1)!\times n & n>1\\ 1 & n=1\end{cases}$$

可以根据该性质直接定义求阶乘的递归函数。

```
#liti5-6.py
def fact(n):
    if n==1:
        return 1
    else:
        return fact(n-1) * n
print("{}!={}".format(6,fact(6)))
```

程序运行结果如下：

```
6!=720
```

使用数学的递归式可以较为清晰地概括递归函数的作用，有助于理解递归函数的含义，方便使用递归函数解决实际问题。递归函数都可以表示为一个数学递归式，例如，例 5-5 的递归函数的数学递归式如下。

$$\sum_{i=1}^{n} i = \begin{cases} n + \sum_{i=1}^{n-1} i & n \neq 1 \\ 1 & n = 1 \end{cases}$$

有这个数学式可以更好地理解程序的计算方法：1 到 n 的和可以用 1 到 n－1 的和加 n 得到。

当有一个有条件限制的数学递归式时，也可以直接转换为递归函数。

【例 5-7】 编写求组合数 C_m^n 的递归函数。

在组合数的性质中，有一种可描述为数学递归式：

$$C_m^n = \begin{cases} C_{m-1}^{n-1} + C_{m-1}^{n} & m > n \text{ 且 } n > 0 \\ 1 & m = n \text{ 或 } n = 0 \end{cases}$$

可以根据这种数学递归式快速转换为等价的递归函数程序以计算组合数：

```
#liti5-7.py
def combin(m,n):
    if m==n or n==0:
        return 1
    else:
        return combin(m-1,n-1)+combin(m-1,n)
m=int(input("请输入 m 的值:"))
n=int(input("请输入 n 的值:"))
print("C({},{})={}".format(n,m,combin(m,n)))
```

程序运行结果如下：

```
请输入 m 的值:7
请输入 n 的值:3
C(3,7)=35
```

5.1.4 函数的闭包空间与装饰器

1. 闭包空间

函数体中包含另一个函数的定义称为**函数的嵌套定义**，内层嵌套定义的函数在外层函数调用时被定义，只能在外层函数中使用，不能在外层函数之外使用。外层函数可以将内层函数名作为返回值，调用外层函数时可以获得并调用内层函数，内层函数被调用时，外层函数处于一种为内层函数服务的工作状态，外层函数定义的变量都处于生存期，可以供内层函数使用，就像是航天员需要穿着航天服来保障其正常工作，这种外层函数提供的工作空间称为内层函数的**闭包空间**，简称为闭包。每次调用外层函数不仅会返回一个内层函数，还会提供执行内层函数所需要的一个专属闭包，主控程序调用内层函数都会在这个闭包空间中运行。

【例 5-8】 构建函数 b 的闭包空间并执行函数 b。

```
#liti5-8.py
def a():
    def b():
        return i
    i=0
    return b
c=a()
print(c())
```

程序的运行结果如下：

```
0
```

在例 5-8 中，外层函数 a 嵌套定义了一个内层函数 b，并且将函数名 b 作为返回值。注意，这里不是调用函数 b，b 后面不能加一对圆括号。主控程序调用外层函数 a，得到的结果是内层函数 b，赋值给变量 c 后，使 c 成为函数变量，调用外层函数 a 还构建了内层函数 b 运行所需要的闭包空间，内层函数 b 返回的变量 i 的值就是来自闭包空间。这样，主控程序调用函数 c 实际上就是调用内层函数 b，内层函数 b 的运行结果是外层闭包空间中变量 i 的值 0，最后程序显示结果 0 并结束。

变量 i 是外层函数 a 中定义的，可以在内层嵌套的函数 b 中使用。当主控程序调用函数 a 时，会返回内层函数 b 并构建闭包空间，这时，变量 i 也被定义并存在于闭包之中。只要函数变量 c 存在，闭包空间就会存在，变量 i 也就一直存在，可以随时被内层函数 b 所使用。

内层嵌套的函数不能对外层函数定义的变量进行修改，函数 b 可以使用外层的变量 i 的值，但不能对 i 赋值。Python 规定，赋值就是对变量的定义。如果函数 b 对变量 i 赋值，会重新产生一个函数 b 中的变量 i，外层函数 a 中的变量 i 仍然存在，与内层函数 b 定义的变量 i 同名，但并不是同一个变量。这时，函数 b 使用的是本地的变量 i，不再允许使用外层的变量 i。要避免函数 b 因为赋值而重新定义一个变量 i，可以在函数 b 中声明变量 i 的作用域为 nonlocal，这样，函数 b 即使赋值变量 i，也只是修改外层的变量。

2. 装饰器

装饰器(decorator)是一种修改已有函数的方法，使用的是函数的闭包技术，也称为修饰器。外层函数通过形参引入要修改的函数，在内层函数中利用黑盒的方式调用该函数，对函数返回的结果再加工后作为内层函数的结果返回，通过闭包技术将内层函数提供给外界使用，原先作为形参引入的函数被替代不再使用，这时，内层函数可以看成是对形参函数的一种修改或装饰。

【例 5-9】 编写计算方差的函数，再通过装饰改造成标准差函数。

方差的计算公式是 $D=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2$，$\bar{x}$ 是平均值，标准差是方差 D 的平方根。