第5章

函　　数

学习目标

- 学会使用函数实现程序代码的分解与重用。
- 学会使用形式参数的函数设计方法，理解函数的参数传递过程，学会返回命令的使用方法。
- 学会使用 lambda 函数和递归函数解决实际问题。

5.1　函数的定义与调用

Python 使用模块和函数来支持分而治之的结构化设计方法。函数是单独定义的一个程序段或表达式，主程序可以通过调用的方式执行该程序段或表达式。函数具有功能抽象表示、代码块可重复使用的特点。在前面的章节中，已经学习了内置函数，例如 input()、print()、eval()等函数，提供了操控输入和输出设备处理数据的能力；也学习了一些标准模块中的函数，例如 math 模块中的 log()、sin()函数，random 模块中的 randint()、sample()函数等，为 Python 表达式提供了更加全面的计算能力；还有通过 pip 下载的各种第三方模块中的函数，支持复杂处理任务的编程，使 Python 具有了大数据分析、动画制作、网络设计等各种应用领域的问题处理能力。这些模块和函数可以构成一个编程平台，程序设计人员可以在不了解函数的代码段如何解决问题的情况下导入模块、调用函数处理问题，具有良好的抽象表示能力。

5.1.1　函数的定义

程序设计人员可以将自己编写的程序中的程序段或表达式定义为函数，称为自定义函数。自定义函数时需要为程序段命名，该名称用于调用定义好的函数，调用函数遵循先定义函数后调用的规则。自定义函数可以存放到自定义模块中，编程时只需要导入自定义模块就可以使用自定义函数，方便函数的重复使用。而且，模块中的函数就像积木块或零件，可以组装起来构建更复杂的模块，解决更复杂的问题。模块和函数就这样不断被重复使用，其中隐藏的错误也会在使用中被发现并去除，从而保证模块和函数的质量越来越高。

要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单一性出

发，分析子问题是否功能相对独立、与其他程序代码联系较少，这样方便重用。分解子问题通常会采用功能分解法，即按照问题的处理要求分解为多个子任务，每个子任务的处理功能要相对完整，子任务间的联系要像工业流水线一样简单，例如，IPO 程序结构就是将程序分解为输入 I、处理 P、输出 O 这 3 个子任务，分解后的子任务功能单一，处理 P 子任务中没有输入、输出的需求，程序容易理解，降低编程难度。同时，子任务间的逻辑关系比较简单，只需要按顺序执行就可以解决问题。

下面看一个示例：

组合数的计算公式为：$C_m^n=\dfrac{m!}{n!\ (m-n)!}$。根据 IPO 分解，可以将变量 m、n 的输入，结果变量 C 的显示，与公式的计算分解为 3 个子任务。输入、输出子任务直接可以使用内置函数 input 和 print 完成，不需要另外自定义函数。将处理子任务自定义为一个函数，功能具有单一性、重用性，但函数代码中需要重复 3 次计算阶乘，仍然不够简洁。如果将计算阶乘作为函数，则可以有效减少程序中的重复代码。

假设有自定义函数：y=f(x)用于求阶乘，自变量 x 称为形式参数(简称形参)，因变量 y 称为返回值，调用函数 f()可以使 y=x!。但在调用函数 f()之前，形式参数 x 必须获得要计算的数据值，这些数值称为实际参数(简称实参)。例如，将实际参数(m－n)的值提供给形参 x，再调用函数 f()可以使 y=(m－n)!，将实参 m 或 n 提供给形参 x，分别调用函数 f()可以分别使 y=m! 或 y=n!。所以，调用函数时，必须先将实参的值提供给函数的形参变量，这个过程称为参数传递。

定义一个函数需要使用保留字为 def 的函数定义命令。函数的定义包含函数头和函数体两部分，函数头以 def 开始，后面依次提供定义的函数名称、以圆括号包住的零个或多个以逗号分隔的形参变量，以冒号结尾，函数体则是缩进表示的程序段。

Python 定义函数的语法规则如下：

```
def <函数名>([<形参列表>]):
    <函数体>
```

<函数名>要符合标识符的命名规则，不能是保留字。<形参列表>是以逗号分隔的多个变量的列表，包含在圆括号中，语法符号[]表示<形参列表>可以省略，没有形参列表则表示函数为零个参数，冒号(:)是函数头部的结束标志。

调用函数时必须为每一个形式参数变量提供数据值，然后才可以执行<函数体>。<函数体>是完成函数功能的程序段，执行完<函数体>的最后一行语句会自动结束函数的调用并会以 None 作为函数调用的返回值。<函数体>中的 return 语句是结束函数调用并返回结果的语句，return 后面的剩余语句不再执行。return 后面可以提供一个表达式，表达式的值是函数调用结束的返回值，如果没有表达式则将 None 作为返回值。

【例 5-1】 将计算阶乘的程序改编为函数程序。

计算 x 的阶乘的程序代码如下：

```
x=int(input())              #输入一个整数给变量 x
y=1
for i in range(1,x+1):
```

```
    y=y*i
print(y)                    #输出 y 中的 x!值
```

根据 IPO 结构分解出处理部分，改编得到的函数定义如下：

```
#liti5-1.py
def f(x):                   #def 保留字后定义了函数头部，f 是函数名，x 是形参变量
    y=1
    for i in range(1,x+1):
        y=y*i
    return y
```

改编后的函数中不包含输入、输出语句，需要处理的数据在主程序中输入，调用函数时会作为实参传递给形参变量 x，得到计算结果 y 后使用 return 语句返回主程序，在主程序中显示返回的结果。

由于缺少主程序部分，执行上面的程序只会得到一个函数对象，并进入 IDLE Shell 中，不会调用函数并执行函数体，但可以在 IDLE Shell 中以互动方式通过函数名 f 调用已定义的函数对象。函数体中的形参变量 x 和结果变量 y 只能局限于函数中使用，无法在主程序中使用。执行程序后操作如下：

```
============ RESTART: D:/liti5-1.py ============
>>> f(5)                    #调用函数 f()时，先将实参值 5 传递给形参 x
    120
>>> print(y)                #f 的函数体中的变量无法在主程序中使用
    NameError: name 'y' is not defined
```

下面在主程序部分加入程序的输入、输出部分，构成的完整程序如下：

```
#liti5-1.py
def f(x):                   #def 保留字后定义了函数头部，f 是函数名，x 是形参变量
    y=1
    for i in range(1,x+1):
        y=y*i
    return y
x=int(input())              #主程序中变量 x 与函数 f()的形参变量 x 是同名的两个不同变量
y=f(x)                      #调用函数 f()时，将主程序中变量 x 的值传递给函数 f()的形参 x
                            #调用函数 f()的结果值返回给主程序中的变量 y
print(y)
```

执行该程序的过程是：先执行函数头部，定义函数 f()，再跳过函数体部分，执行主程序中变量 x 的输入语句，接着调用函数 f()，将主程序中变量 x 的值作为实参传递给函数 f()中的形参变量 x，并执行函数体，最后执行主程序中变量 y 的输出语句。

函数的调用要遵循先定义后调用的顺序，如果将函数的定义调整到主程序变量 x 的输入语句之后，不会影响程序的正常执行。但是，如果将函数的定义调整到函数的调用之后，则会引发函数 f 未定义的错误。

5.1.2 函数的调用与返回

已定义的函数通过函数调用执行，调用时必须为所有形参变量提供实参值，实参值以圆括号包住跟在函数名之后。调用函数时必须保证每一个形参变量都有数据值，否则函数无法调用。

函数的调用格式如下：

```
<函数名>([<实参列表>])
```

＜函数名＞对应的函数对象必须是之前已定义的，＜实参列表＞的成员个数与定义函数时的形参个数一致，以逗号分隔的实参会按顺序赋值给对应位置的形参变量。如果函数的形参个数为零，则为无参调用，不用＜实参列表＞但圆括号要保留。实参可以是表达式、变量或常量，调用时会先计算得到表达式的值再赋值给形参变量。

函数调用可以作为实参表达式的一部分调用其他函数，称为函数的复合调用。复合调用先内后外，内层的函数调用会先执行，得到内层函数值后再计算出实参表达式值，然后调用外层函数。例如，math.sin(math.radians(30))先调用 radians()函数再调用 sin()函数。

【例 5-2】 编写计算 1 到 x 的累加和的函数 s(x)。输入整数 n，使用函数 s()的复合调用计算 n 的累加和的累加和。

程序代码如下：

```
#liti5-2.py
def s(x):                    #函数 s()计算 1+2+…+x 的和
    y=sum(range(1,x+1))      #外层函数 sum()的调用会在内层函数 range()调用之后执行
    return y
n=int(input("请输入 n 的值:"))
r=s(s(n))                #复合调用时外层的函数 s()的调用会在内层的函数 s()调用之后执行
print(r)
```

程序运行结果如下：

```
请输入 n 的值:3
21
```

函数体中可以使用 return 语句返回结果值给主程序的调用者，同时结束函数的调用。return 语句后可以不跟结果值，这时函数也会返回一个 None 常量值作为函数结果。

函数返回语句 return 的使用格式如下：

```
return [<结果表达式>]
```

＜结果表达式＞可以省略，这时返回 None 值，否则会将＜结果表达式＞的计算结果作为函数的返回值。返回值可以是组合数据类型的值，元组数据可以不写圆括号，直接提供以逗号分隔的元组元素值。例如，return 3,4,5，返回的是元组(3,4,5)。

函数体中可以包含多个 return 语句，但需要使用 if 语句将它们分到不同的分支，当执

行到其中一个分支的 return 语句时，函数调用就会结束并返回＜结果表达式＞的值。

【例 5-3】 编写函数 root(a,b,c)计算一元二次方程 $ax^2+bx^1+c=0$ 的实数解。输入 3 个整数给变量 a、b、c，调用函数 root()计算以 a、b、c 为系数的一元二次方程的解。

程序代码如下：

```
#liti5-3.py
def root(a,b,c):
    d=b*b-4*a*c                                  #计算 Δ 值到变量 d 中
    if d>0:                                      #根据 Δ 值得到 3 种不同的解并返回
        x1= (-b+d**0.5)/(2*a)
        x2= (-b-d**0.5)/(2*a)
        return x1,x2                             #返回两个实数解 x1 和 x2 构成的元组
    elif d==0:
        return -b/(2*a)                          #返回一个表达式结果的实数解
    else:
        return                                   #返回 None 值,表示无实数解
a,b,c= eval(input("请输入三个系数 a,b,c 的值:"))
y= root(a,b,c)
print(y)
```

有两个实数解时，程序运行结果如下：

```
请输入三个系数 a,b,c 的值:2,5,3
(-1.0, -1.5)
```

有一个实数解时，程序运行结果如下：

```
请输入三个系数 a,b,c 的值:1,4,4
-2.0
```

无实数解时，程序运行结果如下：

```
请输入三个系数 a,b,c 的值:2,4,4
None
```

一个程序有多个函数时，可以按先后顺序依次定义，后面定义的函数可以调用前面定义的函数，反之则不行。在一个函数的函数体中调用已定义的另一个函数的过程，称为函数的嵌套调用。例如，组合数 C_m^n 的计算可以编写一个函数 c(n,m)，在函数 c()的函数体中可以调用前面定义的阶乘函数 f()完成组合数的计算。

【例 5-4】 编写阶乘函数 f(x)，再编写计算组合数 C_m^n 的函数 c(n,m)。编写主程序，输入两个整数 n,m，如果 n＜＝m 则调用函数 c()计算组合数 C_m^n。

程序代码如下：

```
#liti5-4.py
def f(x):
    y=1
    for i in range(1,x+1):
        y=y*i
    return y
```

```
def c(n,m):
    y=f(m)//(f(n)*f(m-n))      #函数 c()的函数体中先后嵌套调用了 3 次已定义的函数 f
    return y                   #两个函数中的同名变量 y 不是同一个变量,不会相互影响
n,m= eval(input("请输入 n,m 的值:"))
if n<=m:                       #只有输入的值满足 n<=m,函数 c()才能得到正确的结果
    y=c(n,m)
    print(y)
```

程序运行结果如下：

```
请输入 n,m 的值:3,5
10
```

5.1.3 函数的嵌套调用和递归调用

1. 函数的嵌套调用

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。

【例 5-5】 下面程序中有 3 个函数，函数之间呈现两层嵌套。请分析程序的运行过程和运行结果。

程序代码如下：

```
#liti5-5.py
def c(x):
    y=2*x
    return y
def b(x):
    y=2*c(x)                          #函数 b()嵌套调用函数 c()
    return y
def a(x):
    y=2*b(x)                          #函数 a()嵌套调用函数 b()
    return y
y=a(1)
print(y)
```

程序中函数的嵌套结构如图 5-1 所示：

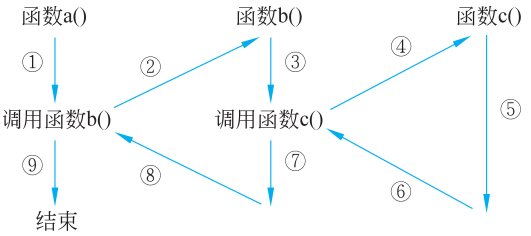

图 5-1 两层函数嵌套调用的过程

程序从主程序调用函数 a()开始执行，函数 a()嵌套调用函数 b()时必须在函数 b()得到结果并返回后，函数 a()才能得到变量 y 的值并返回，即要执行图中的⑨必须先执行图中

的⑧。执行函数 b()时又嵌套调用了函数 c(),所以,函数 b()必须在函数 c()得到结果并返回后,才能继续计算变量 y 的值并返回,即要执行图中的⑦必须先执行图中的⑥。执行函数 c()时没有嵌套调用,可以直接计算得到变量 y 的值并返回,即图中的⑤可以直接执行。

程序的执行过程和结果是:主程序将实参值 1 提供给函数 a()的形参 x,函数 a()嵌套调用函数 b(),即②,将 x 中的值 1 提供给函数 b()的形参 x。函数 b()嵌套调用函数 c(),即④,将 x 中的值 1 提供给函数 c()的形参 x。函数 c()直接计算 2 * 1 得到结果值 2,赋值给变量 y 并返回到函数 b(),即⑥。函数 b()得到返回值 2,计算 2 * 2 的结果值 4,赋值给变量 y 并返回函数 a(),即⑧。函数 a()得到返回值 4,计算 2 * 4 的结果值 8,赋值给变量 y 并返回主程序。所以,执行程序显示的结果是 8。

虽然程序中的 3 个函数形参变量名称都是 x,返回值变量名称都是 y,但不同函数中使用的变量都是独立存在的,整个调用过程中总共定义的变量有 6 个,同名变量的使用也不会相互影响。

【例 5-6】 编写阶乘函数 f(x),再编写求和函数 s(n),能够计算$\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!}$的结果。输入变量 n,调用函数 s 求 1..n 范围内阶乘倒数的和,结果保留 4 位小数。

程序代码如下:

```
#liti5-6.py
def f(x):
    y=1
    for i in range(1,x+1):
        y=y*i
    return y
def s(n):
    y=0
    for i in range(1,n+1):
        y=y+1/f(i)                          #嵌套调用函数 f()求 i 的阶乘
    return y
n=int(input("请输入 n 的值:"))
y=s(n)
print("{:.4f}".format(y))                   #格式槽{:.4f}设置结果保留 4 位小数
```

程序运行结果如下:

```
请输入 n 的值:5
1.7167
```

2. 函数的递归调用

当一个函数在函数体中嵌套调用该函数自身时,这种嵌套调用过程称为递归调用,递归调用是一种特殊的嵌套调用。递归调用自身时必须放置在一个条件分支中,否则会造成无穷的递归调用而无法结束。所以,递归函数要为递归调用设置一个条件,当条件满足进行递归调用,条件不满足时函数直接得到结果并结束。

【例 5-7】 下面程序中的函数 a(n),当 n>1 时会递归调用 a(n-1),当 n=1 时直接计

算结果并返回。请分析程序的运行过程和运行结果。

程序代码如下：

```
#liti5-7.py
def a(n):
    if n==1:
        y=2
        return y                          #当 n=1 时,函数 a(1)不递归调用,直接返回结果 2
    else:
        y=2*a(n-1)                        #当 n!=1 时,函数 a(n)递归调用 a(n-1)
        return y
n=int(input("请输入 n 的值:"))
y=a(n)
print(y)
```

当 n 的值为 3 时，函数 a(3)会两层嵌套地递归调用自身，调用结构如图 5-2 所示。

程序的运行过程和结果是：主程序使用 n 中实参值 3 第一次调用函数 a()，即 a(3)。函数 a(3)会递归调用自身，即②中嵌套调用 a(2)。函数 a(2)会递归调用自身，即④中嵌套调用函数 a(1)。函数 a(1)不递归直接返回结果值 2，即⑥。函数 a(2)得到返回值 2 后会继续执行，将返回值 * 2 的结果值 4 返回，即⑧。函数 a(3)得到返回值 4 后会继续执行，将返回值 * 2 的结果值 8 返回主程序。所以，主程序执行函数 a(3)的结果是 8。

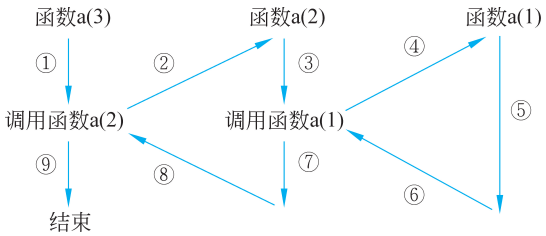

图 5-2　例 5-7 递归调用的过程

递归函数的执行过程比较复杂，但它的正确性可以通过问题所基于的数学递归求解公式的证明来保证。所以，编写递归函数前，可以先分析问题的计算过程是否可以表示成数学递归求解公式。

【例 5-8】　分析 $S=n!$ 问题的递归求解公式，并编写基于递归公式的递归函数。

$n!$ 是求 n 的阶乘，数学中表示阶乘时会有下述递归公式表示求解法。

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

根据该递归公式可以编写出求阶乘的递归函数。

程序代码如下：

```
#liti5-8.py
def f(n):
    if n==1:                              #当 n=1 时,直接返回结果 1
        return 1
    else:
        return f(n-1)*n                   #当 n>1 时,递归调用自身
```

```
n=int(input("请输入 n 的值:"))
print(f(n))
```

程序运行结果如下：

```
请输入 n 的值:6
720
```

与此问题类似的是求 1..n 的累加和问题，该问题的数学递归公式如下所示。

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

有了这个数学递归求解公式，可以得到递归程序，而且该数学模型可以保证递归程序的正确性。

【例 5-9】 编写求组合数 C_m^n 的递归函数。

组合数的数学性质中，有一种为数学递归公式，如下所示。

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

根据该数学递归公式可以编写出计算组合数的递归程序。

程序代码如下：

```
#liti5-9.py
def combin(n,m):
    if m==n or n==0:                                  #当 m=n 或 n=0 时,返回结果 1
        return 1
    else:
        return combin(n-1,m-1)+combin(n,m-1)          #当 m>n 且 n>0 时,递归调用自身
n,m=eval(input("请输入 n,m 的值:"))
y=combin(n,m)
print(y)
```

程序运行结果如下：

```
请输入 n,m 的值:3,7
35
```

5.2 参数传递

函数的参数传递是调用函数时首先要做的工作。参数传递是将调用函数时的实参按照位置顺序依次赋值给形参变量，如果每一个形参变量都有数据值，则开始执行函数体。调用函数时提供的实参个数不能少于函数定义时设计的形参个数，否则会造成部分函数形参没有数据值；而实参个数多于形参个数时，调用过程同样无法进行。

例 5-9 中定义的函数 combin(n,m)需要两个实参值,提供的实参只有一个时,会提示有一个形参 m 没有值:

```
>>> combin(3)
    TypeError: combin() missing 1 required positional argument: 'm'
```

提供的实参达到 3 个时,会提示实参个数不对:

```
>>> combin(3,7,9)
    TypeError: combin() takes 2 positional arguments but 3 were given
```

然而也有特殊的情况,如果形参变量是带默认值的形参,则在不为该形参提供实参值时,也能正常调用函数。

5.2.1 函数形参

函数定义时形参有以下 4 种形态。

- 位置形参(positional parameter)。这种形参必须提供实参值,否则调用一定会失败,是必填形参。
- 默认值形参(parameter with a default)。这种形参不能写在位置形参之前,每一个默认值形参名后面一定会通过等号带上一个默认值表达式。这种形参如果没有提供实参值,会使用默认值作为实参,所以,函数调用时默认值形参的实参值可填可不填,是选填形参。
- 可变长位置形参。这种形参定义时至多只能有一个,形参名前面会带上一个星号(*)。可变长位置形参只能按位置顺序提供实参值,它将其他形参分成前后两部分,它前面的所有形参优先得到位置实参,然后它会将所有剩余的位置实参打包成元组接收,这样位于它后面的所有形参就无法得到位置实参值了。
- 可变长关键字形参。这种形参定义时至多只能有一个,形参名前面会带上连续两个星号(**),它必须处于所有形参中的最后位置。可变长关键字形参只能得到关键字实参值,它会将调用函数时所有的关键字实参中多出来的、找不到对应形参名进行赋值的关键字实参打包成字典接收。由于实参只有位置实参和关键字实参两种,如果在函数定义中同时有了这两种可变长形参,超过需要的实参都会被打包带走,从而避免实参过多的错误。这两种可变长形参作为复杂的形参后面会单独作为一节介绍。

下面通过一些示例介绍位置形参和默认值形参的定义方法。

```
>>> def f(a,b=2,c,d=4):   #定义错误:位置形参 c 放在了默认值形参 b 之后
    SyntaxError: parameter without a default follows parameter with a default
>>> def f(a,b,c=3,d=4):   #两个位置形参 a、b 必须放在两个默认值形参 c、d 之前
        return a+b+c+d
>>> f(10,20,30,40)        #提供了 4 个按位置赋值的实参给 a、b、c、d
    100
>>> f(10,20,30)           #提供了 3 个按位置赋值的实参给 a、b、c、d 用默认值 4
    64
```