第 5 章

Python 函数

复杂的问题通常采用“分而治之”的思想解决，把大任务分解为多个小的任务，解决每个小的容易的子任务，从而解决较大的复杂任务。本章介绍函数的声明和调用、返回值及函数的四种参数，以及两类特殊的函数等相关知识。

函数

5.1 函数声明与调用

5.1.1 函数声明

函数是可重复使用的，用来实现单一或相关联功能的代码段。函数能提高代码的重复利用率。Python 提供了许多内建函数，如 print()等。用户自己创建的函数称为用户自定义函数，语法格式如下。

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

说明：

- 函数使用关键字 def(define 的缩写)声明，函数名为有效的标识符和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间用于定义参数。
- 函数内容以冒号起始，并且缩进。
- 函数名下的每条语句前都要用 Tab 键缩进，没有缩进的第一行则被视为在函数体之外的语句，与函数同级的程序语句。
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的 return 相当于返回 None。

【例 5-1】 函数声明，如图 5-1 所示。

hello 是函数的名称，后面的括号里是参数，这里没有，表示不需要参数。但括号和后面的冒号都不能少。

```
>>> def hello():
...     print("Hello World!")
...
>>> hello()
Hello World!
```

图 5-1 函数声明

5.1.2 函数调用

在 Python 中，函数调用的语法格式为：

```
函数名([实际参数])
```

函数调用时传递的参数是实参，实参可以是变量、常量或表达式。当实参个数超过一个

时,用逗号分隔,实参和形参应在个数、类型和顺序上一一对应。对于无参函数,调用时实参为空,但()不能省略。

【例 5-2】 利用海伦公式,求三角形面积。

程序代码如下。

```
import math
def triarea(x,y,z):
      s =(x +y +z)/ 2
      print(math.sqrt((s -x) * (s -y) * (s -z) * s) )
triarea(3,4,5)
```

程序运行结果如下。

```
6.0
```

triarea(3,4,5) 调用 triarea(x,y,z),程序执行步骤如图 5-2 所示。

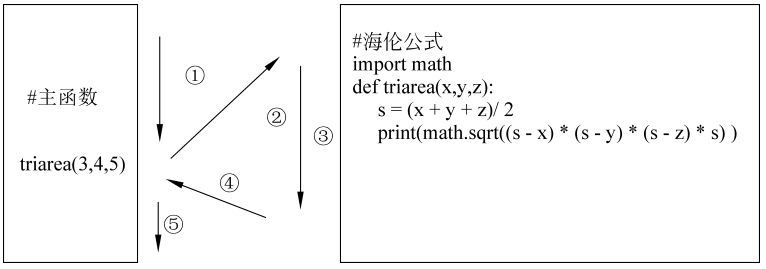

图 5-2 函数调用

函数调用步骤如下。

步骤 1:运行主函数,如图 5-2 中①箭头所示,当运行到 triarea(3,4,5)语句时,主函数中断,Python 寻找同名的 triarea()函数。如果没有找到,Python 提示语法错误。

步骤 2:找到同名函数,进行函数调用,实现将实参的值传递给形参,如图 5-2 中②箭头所示。

triarea(3,4,5)中 3,4,5 是实参的取值。

triarea(x , y, z)中 x,y,z 是形参。

在实参和形参结合时,必须遵循以下三条规则。

(1) 实参和形参个数相等。

(2) 实参和形参的数据类型依次相同。

(3) 实参给形参依次传递,实参和形参传递如表 5-1 所示。

表 5-1 函数调用时,实参和形参传递的三条规则

三条规则	实参(3,4,5)	形参(x,y,z)	运 行 结 果
参数个数	3 个	3 个	个数相等
参数类型	3 为整型 4 为整型 5 为整型	x 为整型 y 为整型 z 为整型	类型依次相同
依次传递			x 得到 3,y 得到 4,z 得到 5

步骤 3：执行海伦公式函数，如图 5-2 中③箭头所示。

步骤 4：海伦公式执行结束，程序返回到主函数的中断处，如图 5-2 中④箭头所示。

5.1.3 函数返回值

函数返回值是指函数被调用执行后，返回给主调函数的值。一个函数可以有返回值，也可以没有返回值。使用关键字 return 实现，形式如下。

```
return  表达式
```

return 语句使得程序控制从被调用函数返回到调用函数，并将返回值带回。

(1) 在函数内根据具体的 return 语句返回。

【例 5-3】 求两个数中的较大值。

```
def max(a,b):
    if a>b:
        return a
    else:
        return b
t=max(3,4)
print(t)
```

程序运行结果如下。

```
4
```

(2) 如果没有 return 语句，会自动返回 None；如果有 return 语句，但是 return 后面没有表达式，也返回 None。

【例 5-4】 没有 return 的语句。

```
def add(a,b):
    c=a+b
t=add(3,4)
print(t)
```

程序运行结果如下。

```
None
```

(3) 如果需要从函数中返回多个值时，可以使用元组作为返回值。

【例 5-5】 返回多个值。

```
def getMaxMin(a):
    max=a[0]
    min=a[0]
    for i in range(0,len(a)):
        if max<a[i]:
            max=a[i]
        if min>a[i]:
```

```
            min=a[i]
    return(max,min)

a_list=[4,8,3,0,-3,93,6]
x,y=getMaxMin(a_list)
print("")
print("最大值为",x,"最小值为",y,)
```

程序运行结果如下。

```
最大值为 93 最小值为 -3
```

5.2 参数传递

5.2.1 实参与形参

实参(实际参数)是指传递给函数的值,即在调用函数时,由调用语句传给函数的常量、变量或表达式。形参(形式参数)是在定义函数时,函数名后面括号中的变量,用逗号分隔。作为函数与主调程序交互的接口,用来接收调用该函数时传递的实参,从主调程序获得初值,或将计算结果返回给主调程序。

形参和实参具有以下特点。

(1) 函数在被调用前,形参只是代表了执行该函数所需要参数的个数、类型和位置,并没有具体的数值,形参只能是变量,不能是常量、表达式。只有当调用时,主调函数将实参的值传递给形参,形参才具有值。

(2) 形参只有在被调用时才分配内存单元,调用结束后释放内存单元,因此形参只在函数内部有效,函数调用结束返回主调用函数后则不能再使用该形参变量。

(3) 实参可以是常量、变量、表达式、函数等,无论实参是何种数据类型的变量,函数调用时必须是确定的值,以便把这些值传给形参。

(4) 实参和形参在数量、类型、顺序方面应严格一致,否则会发生类型不匹配错误。

5.2.2 传对象引用

Python 的参数传递与 C 语言不同,既不是传值(pass-by-value),也不是传引用(pass-by-reference),而是传对象引用(pass-by-object-reference),传递的是一个对象的内存地址。这种方式相当于传值和传址的一种综合。当函数收到的是可变对象(如字典或者列表)的引用,就能修改对象的原始值——相当于“传引用”。当函数收到的是不可变对象(如数字、字符或者元组)的引用,就不能直接修改原始对象——相当于“传值”。

【例 5-6】 数字和列表。

```
import sys
a=2
b=[1,2,3]
def change(x,y):
    x=3
```

```
    y[0]=4
change(a,b)
print(a,b)
```

程序运行结果如下。

```
2 [4, 2, 3]
```

数字作为一个不可变对象，a 的值没有变化，而 b 作为列表对象，是可变对象，所以 b 被改变了。

【例 5-7】 字符串和字典。

```
import sys
a="11111"
b={"a":1,"b":2,"c":3}
def change(x,y):
    x="222"
    y["a"]=4
change(a,b)
print(a,b)
```

程序运行结果如下。

```
11111 {'a': 4, 'c': 3, 'b': 2}
```

a 作为字符串是不可变对象，所以没有变化；b 作为字典是可变对象，所以被改变了。

5.3 参数分类

Python 的参数分为必备参数、默认参数、关键参数和不定长参数等。

5.3.1 必备参数

必备参数是指调用函数时，参数的个数、参数的数据类型，以及参数的输入顺序必须正确，否则会出现语法错误。

【例 5-8】 必备参数。

```
def printme(str):
    print(str)
    return
printme()
```

程序运行结果如图 5-3 所示。

5.3.2 默认参数

默认参数是指允许函数参数有默认值，如果调用函数时不给参数传值，参数将获得默认值。Python 通过在函数定义的形参名后加上赋值运算符(=)和默认值，给形参指定默认参数值。注意，默认参数值是一个不可变的参数。

```
>>> printme()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: printme() missing 1 required positional argument: 'str'
```

图 5-3　运行结果

【例 5-9】 使用默认参数值。

```
def say(message, times =1):
      print(message * times)

#调用函数
say('Hello')              #默认参数 times 为 1
say('World', 4)
```

程序运行结果如下。

```
Hello
WorldWorldWorldWorldWorld
```

5.3.3　关键参数

函数的多个参数值一般默认从左到右依次传入。但是,Python 也提供了灵活的传参顺序,引入了关键参数。关键参数又称为命名参数,用于改变指定参数的顺序。

【例 5-10】 使用关键参数。

```
def func(a, b=4, c=10):
  print('a is', a, 'and b is', b, 'and c is', c)
#调用函数
func(3, 7)
func(24, c=24)
func(c=40, a=100)
```

程序运行结果如下。

```
a is 3 and b is 7 and c is 10
a is 24 and b is 4 and c is 24
a is 100 and b is 4 and c is 40
```

5.3.4　不定长参数

不定长参数又称为可变长参数,参数以一个 * 号开头代表接收元组,以两个 * 号开头代表接收字典。

【例 5-11】 不定长参数。

```
def foo(x, * y, * * z):
    print(x)
    print(y)
```

```
    print(z)
```

程序运行结果如下。

根据输入数据的不同,分别有如下三种执行效果。

效果 1: 输入 foo(1)

程序运行结果如下。

```
1
()
{}
```

效果 2: 输入 foo(1,2,3,4)

程序运行结果如下。

```
1
(2, 3, 4)
{}
```

效果 3: 输入 foo(1,2,3,a="a",b="b")

程序运行结果如下。

```
1
(2, 3)
{'a': 'a', 'b': 'b'}
```

5.4 两类特殊函数

5.4.1 匿名函数

匿名函数是指 lambda 表达式,不使用 def 定义函数,所有使用 lambda 函数的地方都可以使用普通函数(def 声明的函数)来代替。语法如下。

```
lambda parameters:expression
```

参数如下。

- parameters: 可选,通常是以逗号分隔的变量表达式。
- expression: 条件表达式,不能包含分支或循环,也不能包含 return 函数。

【例 5-12】 lambda 函数。

```
sum =lambda arg1, arg2: arg1 +arg2
#调用 sum 函数
print ("相加后的值为: ", sum( 10, 20 ))
```

程序运行结果如下。

```
相加后的值为: 30
```

5.4.2 递归函数

【例 5-13】 计算 4 的阶乘。

两种方法如表 5-2 所示。方法一通过循环语句来计算阶乘,该方法的前提是了解阶乘的计算过程,并可用语句把计算过程模拟出来。方法二通过递推关系将原来的问题缩小成一个规模更小的同类问题,将 4 的阶乘问题转换为 3 的阶乘问题,只需找到 4 的阶乘和 3 的阶乘之间的递推关系,以此类推,直到在某一规模上(当 n 为 1 时)问题的解已知,其后,回归。这种解决问题的思想称为递归。

表 5-2 计算阶乘

方 法 一	方 法 二
循　　环	递　　归

s =1
for i in range(1,4):
　　s =s * i
print(s)
	def fac(n):
	　　if n==1:
	　　　　return 1
	　　return n * fac(n -1)

fac(4)递归求解过程如图 5-4 所示。

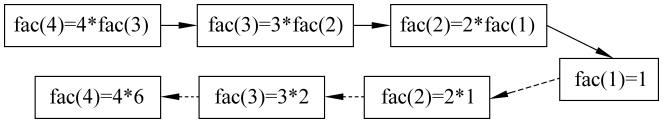

图 5-4 fac(n)=n! 递归求解过程

递归调用的过程类似于多个函数的嵌套调用,只不过这时的调用函数和被调用函数是同一个函数,即在同一个函数中进行嵌套调用。

递归是通过“栈”来实现的,按照“后调用先返回”的原则——每当函数调用时,就在栈顶分配一个存储区;每当退出函数时,就在栈顶释放该存储区。

下面以 fac(4!)来分析其如何在内存中进行数据的入栈与出栈。

第一阶段:递推阶段(入栈)。

(1) 初始调用 fac(4!)会在栈中产生第一个活跃记录,输入参数 n=4,输出参数 n=3,如图 5-5 中第 1 步所示。

(2) 由于 fac(4!)调用没有满足函数的终止条件,因此 fac()将继续以 n=3 为参数递归调用,在栈上创建另一个活跃记录,n=3 成为第一个活跃期中的输出参数,同时又是第二个活跃期中的输入参数,这是因为在第一个活跃期内调用 fact()产生了第二个活跃期,如图 5-5 中第 2 步所示。

(3) 以此类推,这个入栈过程将一直继续,直到 n 的值变为 1,此时满足终止条件,fac()将返回 1,如图 5-5 中第 3、4 步所示。

第二阶段:回归阶段(出栈)。

(1) 当 n=1 时的活跃期结束,n=2 时的递归计算结果就是 2×1=2,因而 n=2 时的活跃期也将结束,返回值为 2,如图 5-5 中第 5 步所示。

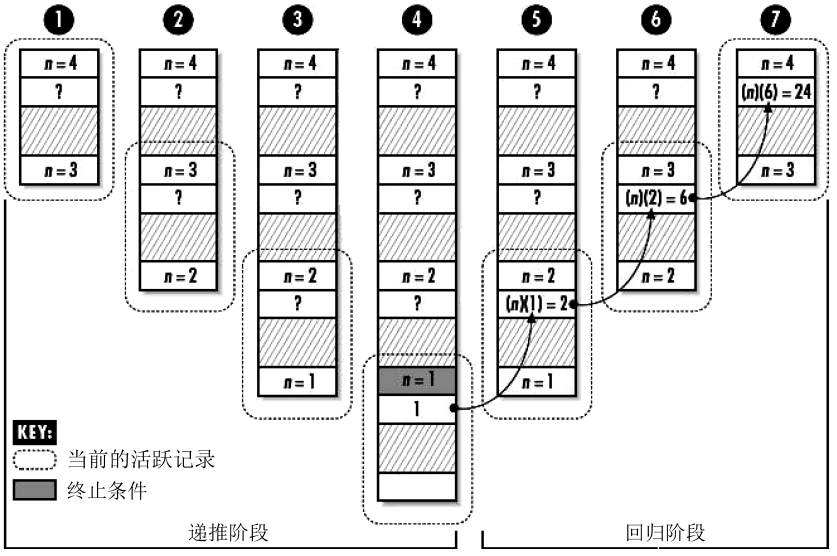

图 5-5　以 fac(4!)讲解基本递归

(2) 如此反复，$n=3$ 的递归计算结果表示为 $3\times2=6$，因此 $n=3$ 时的活跃期结束，返回值为 6，如图 5-5 中第 6 步所示。

(3) 最终，当 $n=4$ 时的递归计算结果将表示为 $6\times4=24$，$n=4$ 时的活跃期将结束，返回值为 24，如图 5-5 中第 7 步所示，递归过程结束。

递归调用的另一种形式是尾递归。尾递归是指函数中所有递归形式的调用都出现在函数的末尾，即当递归调用是整个函数体中最后执行的语句且它的返回值不属于表达式的一部分时，这个递归调用就是尾递归。由于尾递归是函数的最后一条语句，则当该语句执行结束从下一层返回至本层后立刻又返回至上一层，因此在进入下一层递归时，不需要继续保存本层所有的实参数和局部变量，即不做入栈操作而是将栈顶活动记录中的所有实参数更改为下一层的实参数，从而不需要进行任何其他操作而是连续出栈。

计算 $n!$ 的尾递归函数如下：

$$F(n,a)=\begin{cases}a, & n=1\\ F(n-1,na), & n>1\end{cases}$$

尾递归函数为 $F(n,a)$，与基本递归 fac(n)相比多了第二个参数 a，a 用于维护递归层次的深度，初始值为 1，从而避免每次需要将返回值再乘以 n。尾递归是在每次递归调用中，令 $a=na$ 并且 $n=n-1$，持续递归调用，直到满足结束条件 $n=1$，返回 a 即可。

尾递归计算 4! 的过程如图 5-6 所示。$F(4,1)$的递归过程如下。

$$F(4,1)=F(3,4\times1)\rightarrow F(2,3\times4\times1)\rightarrow F(1,2\times3\times4\times1)$$

$n!$ 的尾递归代码如下。

```
def F(n,a):
    if n==1:
        return a
    else:
        return F(n -1, n * a)
```

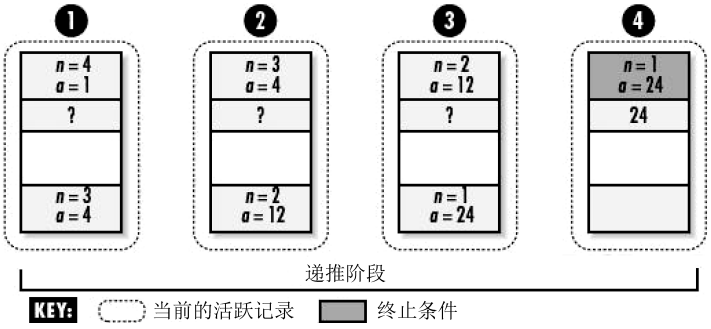

图 5-6　以 $F(4,1)$讲解尾递归

```
#调用 F(n,a)函数
print(F(4,1))
```

递归简洁、清晰、可读性强,但执行效率低。

【例 5-14】 列表元素个数的加权和。

输入一个嵌套列表,嵌套层次不限,根据层次,求列表元素的加权个数和。第一层每个元素算一个元素,第二层每个元素算两个元素,第三层每个元素算三个元素,第四层每个元素算四个元素,…,以此类推。

输入格式:

在一行中输入一个列表。

输出格式:

在一行中输出加权元素个数值。

输入样例:

在这里给出一组输入。例如:

```
[1,2,[3,4,[5,6],7],8]
```

输出样例:

在这里给出相应的输出。例如:

```
15
```

代码如下。

```
def f(a,b):
    s=0
    for i in a:
        if isinstance(i,list):
            s=s+f(i,b+1)
        else:
            s=s+b
    return s
s=eval(input(""))
r=f(s,1)
```