第5章 函数与模块

按照结构化程序设计思想，对于一个规模比较大的程序，先按功能划分成若干模块，每一个模块可以再划分成更小的模块，直至每一个模块完成一个比较单一的功能，然后分别编写对应于每一个模块的程序，最后再把这些模块组织成一个完整的程序。这样一种程序设计模式可使程序结构清晰，易于阅读和理解，易于修改和维护，也便于多人合作编写程序，从而保证程序的质量和开发效率。

在 Python 语言中，一个模块的功能可由一个或多个函数实现。一个 Python 源程序可由若干函数组成，函数之间通过调用关系形成一个完整的程序。

5.1 函数定义

在 Python 语言中，函数要先定义后调用。函数定义的语法格式如下：

```
def 函数名(形式参数表):
    函数体
    return 返回值
```

示例：定义一个函数，求 3 个数的平均值。

```
def average(num1,num2,num3):
    ave=(num1+num2+num3)/3
    return ave
```

结合该示例，对函数定义解释如下：

(1) def 是用于定义函数的关键字。

(2) 函数名用于标识函数。函数定义后，一般要通过函数调用的方式使用这个函数，函数调用时要用到函数名。函数名要符合标识符的命名规则。当函数名由多个单词组成时，本书采用非首单词首字母大写的形式，如 averageScore。

(3) 形式参数表用于给函数运算提供需要的数据。如果有多个形式参数，形式参数之间用逗号分开。参数名也要符合标识符的命名规则。函数也可以没有形式参数，此时形式参数表为空，但一对小括号不能省略。上面示例函数的功能是计算 3 个数的平均值，形式参数 num1、num2、num3 用于提供参与计算的 3 个数值。

(4) 函数定义的第一行以冒号(:)结束。该行称为函数首部。

(5) 函数体由实现函数功能的一条或多条语句组成。相对于函数首部，函数体要有缩进。

(6) 如果函数的功能是计算出一个或多个结果值，则需要用 return 语句把函数的结果带回调用该函数的调用函数(调用程序)。return 后面可以跟一个值，也可以跟由逗号分开

的多个值。如果函数的功能只是完成一些动作,没有值需要带回调用程序,可以不写 return 语句。

5.2 函数调用

如果只有函数定义,并不能发挥实际的作用。一个函数只有被其他函数(或程序)调用才能被执行,才能实现其定义的功能。调用其他函数的函数(或程序)称为调用函数(或调用程序),被其他函数调用的函数称为被调用函数。

函数调用的语法格式如下:

```
函数名(实际参数表)
```

函数调用时的实际参数(简称实参)个数应与函数定义时的形式参数(简称形参)个数一致,把每个实参的值传递给对应的形参。每个实参是一个表达式,但函数的各个形参必须是变量,接收来自实参的值。实参表达式可以是简单表达式,如一个常量或一个变量等;也可以是比较复杂的表达式,如由常量和变量组成的表达式。

函数调用可以以一个语句的形式出现,这时函数的执行结果不是得到一个或多个返回值,而是实现特定的功能,如交换两个变量的取值、对一组数据排序、显示多行字符串等。函数调用也可以出现在表达式中,作为运算对象出现,这时的函数必须有返回值。实际编程时,定义和调用有返回值的函数的情况比较多。

调用一个函数时,首先计算实参表中各表达式的值,然后调用函数所在的程序暂停执行,转去执行被调用函数,被调用函数中各形参的初值就是调用函数中各对应实参的值。被调用函数执行完函数体语句后,返回调用函数继续执行函数调用所在语句后面的语句。

【例 5.1】 通过函数计算 3 个成绩的平均值。

```
#P0501.py
def average(num1,num2,num3):                    #定义计算平均值的函数
    ave=(num1+num2+num3)//3
    return ave
#主程序,程序执行时从主程序开始
print("请输入 3 个成绩:")
score1=int(input("score1="))
score2=int(input("score2="))
score3=int(input("score3="))
ave_score=average(score1,score2,score3)     #调用函数计算平均成绩
print("平均成绩=",ave_score)
```

该程序包括两部分:函数定义和函数调用。程序的执行过程如下:

(1) 程序的执行从主程序开始,先执行主程序中的如下语句:

```
print("请输入 3 个成绩:")
score1=int(input("score1="))
score2=int(input("score2="))
score3=int(input("score3="))
```

(2) 执行函数调用所在的语句:

```
ave_score=average(score1,score2,score3)
```

(3) 暂停主程序的执行,转去执行被调用函数 average()中的函数体:

```
ave=(num1+num2+num3)//3
return ave
```

其中,3 个形参 num1、num2、num3 的初值分别来自调用函数中实参 score1、score2、score3 的值。

(4) 当执行完 return ave 语句后,返回主程序,继续执行如下语句:

```
ave_score=average(score1,score2,score3)
print("平均成绩=",ave_score)
```

其中,被调用函数 average(score1,score2,score3)的值就是通过 return 语句返回的 ave 的值。

【例 5.2】 定义函数将 3 个数按由小到大的顺序排序。

```
#P0502.py
def reorder(num1,num2,num3):      #定义函数
    if num1>num2:
        num1,num2=num2,num1       #交换变量 num1 和 num2 中的数据
    if num1>num3:
        num1,num3=num3,num1       #交换变量 num1 和 num3 中的数据
    if num2>num3:
        num2,num3=num3,num2       #交换变量 num2 和 num3 中的数据
    return num1,num2,num3         #返回排序后的 3 个值
#主程序
print("请输入 3 个数: ")
a=eval(input("a="))
b=eval(input("b="))
c=eval(input("c="))
x,y,z=reorder(a,b,c)              #调用函数对 3 个数排序
print("排序前 3 个数的顺序是:",a,b,c)
print("排序后 3 个数的顺序是:",x,y,z)
```

该程序调用的 reorder()函数有 3 个返回值,在 x,y,z=reorder(a,b,c)的调用模式中,把 3 个返回值直接依次赋给变量 x、y、z;也可以用 t=reorder(a,b,c)模式调用,由于赋值号(=)左边只有一个变量名,所以要把 3 个返回值组织成一个元组赋给变量 t,此时的变量 t 被看作元组变量。

说明:

(1) 上面的程序代码由两部分组成,前面是函数定义,后面是主程序。程序的执行从主程序开始,主程序调用函数时,转去执行函数,函数执行结束返回主程序继续执行。

(2) 在由主程序和被调用函数组成的程序结构中,数据的输入与输出一般在主程序中完成,被调用函数通过实参和形参接收来自主程序的数据并进行处理,然后用 return 语句将处理结果返回给主程序。有返回值的被调用函数中尽量不要出现输入与输出语句。

【例 5.3】 定义函数输出指定行数的星号串。

```
#P0503.py
def display(n):                          #定义函数
    for i in range(n):
        print("************ * ")          #函数没有返回值
#主程序
m=int(input("请输入要输出星号串的行数="))
display(m)
```

由于定义函数 display()时没有 return 语句,即该函数没有返回值,只是完成输出指定行数的星号串的动作,所以对该函数的调用要以语句形式出现,即 display(m)的形式。

5.3 函数的参数传递

调用函数(调用程序)与被调用函数之间的联系是通过参数传递实现的。定义函数时,系统并不给函数的形参分配内存单元,函数被调用执行时,系统才为各形参分配内存单元,并把对应的实参的值传递给形参。在 Python 语言中,实参值传递给形参有两种方式:一是不改变实参值的传递方式,二是改变实参值的传递方式。

5.3.1 不改变实参值的参数传递

Python 语言中的变量(对应的内存单元)并不直接存储某个值,而是存储了值所在内存单元的地址,这也是在同一个程序中变量类型可以改变的原因,详细介绍见 2.5 节。在调用函数时,实参值传递给形参,实际上是将实参对象的地址传递给形参(为便于表述,这里仍简单地称之为把实参值传递给形参)。如果实参对象是不可变对象(如数值、字符串、元组等),则有新值就分配新的内存单元,所以执行被调用函数时形参值的改变不会影响实参。

【例 5.4】 不改变实参值的参数传递方式。

```
#P0504.py
def fun(x,y):          #定义函数
    print("x={},y={}".format(x,y))
    x=int(x*1.1)
    y=int(y*1.2)
    print("x={},y={}".format(x,y))
#主程序
a=20
b=50
print("a={},b={}".format(a,b))
fun(a,b)
print("a={},b={}".format(a,b))
```

程序执行时的参数传递过程如下。

执行程序时,首先为 a 和 b 两个变量赋初值:

```
a=20        b=50
```

调用执行 fun()函数后,把 a 和 b 的值分别传递给 x、y:

```
x=20        y=50
```

被调用函数执行完，返回主程序之前，各变量的值如下：

```
x=22        y=60
a=20        b=50
```

返回主程序后，各变量的值如下(被调用函数中的变量被撤销)：

```
a=20        b=50
```

所以，程序执行结果如下：

```
a=20,b=50          #进入被调用函数前主程序中变量 a 和 b 的值
x=20,y=50          #刚进入被调用函数时变量 x 和 y 的值
x=22,y=60          #退出被调用函数前变量 x 和 y 的值
a=20,b=50          #返回主程序后变量 a 和 b 的值
```

说明：由于数值是不可变对象，函数中的 x=int(x * 1.1)语句产生了新值 22，所以 x 指向新的对象 22，同样 y 指向新的对象 60，因此就有了上面的程序执行结果。

5.3.2　改变实参值的参数传递

如果实参对象是可变对象(如列表、字典、集合等)，运算可在原数据上进行，所以在被调用函数执行后，形参值的改变会影响到对应的实参值。

【例 5.5】 改变实参值的参数传递方式。

```
#P0505.py
def fun(list2,n):                  #定义函数
    print("list2=",list2)
    for i in range(n):
        list2[i]=int(list2[i] * 1.2)
    print("list2=",list2)
#主程序
list1=[10,20,30,40,50]
print("list1=",list1)
fun(list1,5)
print("list1=",list1)
```

程序执行结果如下：

```
list1=[10, 20, 30, 40, 50]
list2=[10, 20, 30, 40, 50]
list2=[12, 24, 36, 48, 60]
list1=[12, 24, 36, 48, 60]
```

说明：由于列表是可变对象，所以函数中对列表 list2 的运算可在原值上进行，也就是在列表 list1 上进行。

5.3.3　位置参数

默认情况下，调用函数时实参的个数、位置要与定义函数时形参的个数、位置一致，即实

参是按出现的位置与形参对应的,与参数的名称无关,此时的参数称为位置参数。

【例 5.6】 基于位置的参数传递方式。

```
#P0506.py
def fun(x,y):          #定义函数
    print("x={},y={}".format(x,y))
#主程序
a=x=20
b=y=30
fun(a,b)               #实参 a、b 的值分别传递给形参 x、y
fun(x,y)               #实参 x、y 的值分别传递给形参 x、y
fun(y,x)               #实参 y、x 的值分别传递给形参 x、y
fun(x,x)               #实参 x、x 的值分别传递给形参 x、y
```

程序执行结果如下:

```
x=20,y=30              #fun(a,b)的结果
x=20,y=30              #fun(x,y)的结果
x=30,y=20              #fun(y,x)的结果
x=20,y=20              #fun(x,x)的结果
```

从程序执行结果可以看出,实参到形参的对应只是依据参数的位置而定,第一个实参对应第一个形参,第二个实参对应第二个形参,与参数的名字无关。在示例中,不管实参用的是与形参不同名的 a、b 还是与形参同名的 x、y 或交叉对应的 y、x 以及两个实参均为 x,都是把第一个实参的值传递给第一个形参,把第二个实参的值传递给第二个形参。

说明:实参和形参可以同名,也可以不同名,即使同名也分别对应不同的内存单元。

5.3.4 关键字参数

在调用函数时,可以明确指定把某个实参值传递给某个形参,此时的参数称为关键字参数,关键字参数不再按位置进行对应。

【例 5.7】 基于关键字的参数传递方式。

```
#P0507.py
def totalScore(math,lang,puter):          #定义函数
    total=math+lang+puter
    average=total//3
    return total,average
#主程序
print("请输入三门课的成绩:")
math1=int(input("数学="))
lang1=int(input("语文="))
puter1=int(input("计算机="))
total,average=totalScore(math=math1,puter=puter1,lang=lang1)
print("总成绩={},平均成绩={}".format(total,average))
```

关键字参数的优点是:不需要记住形参的顺序,只需指定哪个实参传递给哪个形参即可,而且指定顺序也可以和定义函数时的形参顺序不一致,这对于形参个数较多的情形是很

方便的(形参较多时,不容易准确记住形参的顺序),而且能够更好地保证参数传递正确。

5.3.5　默认值参数

一般来说,函数中形参的值是通过实参传递的。如果需要,则可以在定义函数时直接对形参赋值,此时的参数称为带默认值的参数。在 Python 语言中,对于带默认值的形参,在函数调用时,如果没有对应的实参,就使用该默认值;如果有对应的实参,则仍用实参值,覆盖形参默认值。

【例 5.8】　有默认值的参数传递方式。

```
#P0508.py
def area(r=1.0,pi=3.14):                              #定义函数
    return r * r * pi
#主程序
print("面积={:.2f}".format(area()))                   #两个参数都用默认值
print("面积={:.2f}".format(area(2.6)))                #一个参数用默认值
print("面积={:.2f}".format(area(2.6,3.1415926)))      #两个参数都不用默认值
```

程序执行结果如下:

```
面积=3.14
面积=21.23
面积=21.24
```

第一次调用 area()函数时,由于没有实参,所以程序执行进入 area()函数后,两个形参都取默认值(分别为 1.0 和 3.14);第二次调用时,有一个实参,所以进入 area()函数后,形参 r 取实参值(2.6),形参 pi 取默认值(3.14);第三次调用时,由于有两个实参的值,所以进入 area()函数后,两个形参都取实参传过来的值(分别为 2.6 和 3.1415926)。

说明:

(1) 形参的默认值在定义函数时设定。

(2) 由于函数调用时实参和形参是按照从左至右的顺序对应的,所以设定默认值的形参必须出现在形参表的右端,即在有默认值的形参右面不能有无默认值的形参出现。

如下定义函数时的默认值设定是正确的:

```
def fun1(x,y=5,z=10):
    …
```

调用函数时,如果只提供一个实参,如 fun1(2),将会把实参值 2 传递给形参 x,形参 y 和 z 取默认值。

如下定义函数时的默认值设定是错误的:

```
def fun2(x=3,y,z=10):
    …
```

调用函数时,如果只提供一个实参,如 fun2(6),将会把实参值 6 传递给形参 x(覆盖 x 的默认值),而形参 y 得不到相应的取值,会引起错误。

5.3.6 可变长度参数

在 Python 语言中,除了可以定义固定长度参数(参数个数固定)的函数外,还可以定义可变长度参数的函数。调用此类函数时,可以提供不同个数的参数以满足实际需要,进一步增强了函数的通用性和灵活性。在定义函数时,可变长度参数主要有两种形式:单星号参数和双星号参数。单星号参数是在形参名前加一个星号(*),把接收的多个实参值组合在一个元组内,以形参名为元组名;双星号参数是在形参名前加两个星号(**),把接收的多个实参值组合在一个字典内,以形参名为字典名。

【例 5.9】 单星号可变长度参数。

```
#P0509.py
def totalScore(*score):      #定义函数,单星号形参用于把接收的实参值组合为元组
    total=0
    for i in score:
        total+=i
    ave=total//len(score)
    return total,ave
#主程序
total1,ave1=totalScore(78,62,81)          #第一次调用,接收 3 个实参
print("总成绩={},平均成绩={}".format(total1,ave1))
total2,ave2=totalScore(95,61,72,87)       #第二次调用,接收 4 个实参
print("总成绩={},平均成绩={}".format(total2,ave2))
```

程序执行结果如下:

```
总成绩=221,平均成绩=73
总成绩=315,平均成绩=78
```

说明:

(1) 参数 score 前面有一个星号(*),Python 解释器会把形参 score 看作可变长度参数,可以接收多个实参,并把接收的多个实参值组合为一个名字为 score 的元组。

(2) 第一次调用 totalScore()函数时,将 3 个数值传递给可变长度形参 score,并组合为包含 3 个元素、名字为 score 的元组。

(3) 第二次调用 totalScore()函数时,将 4 个数值传递给可变长度形参 score,并组合为包含 4 个元素、名字为 score 的元组。

【例 5.10】 双星号可变长度参数。

```
#P05010.py
def student(**stu):                    #定义函数,双星号形参用于把接收的实参值组合为字典
    num=0
    for item in stu.values():
        if item=="河北省":
            num+=1
    return num
#主程序
count=student(小明="河北省",小亮="北京市",小莲="河北省")
print("有{}个学生来自河北省".format(count))
```

说明：

（1）参数 stu 前面有两个星号(**)，Python 解释器会把形参 stu 看作可变长度参数，可以接收多个实参，并把接收的多个实参值组合为一个名字为 stu 的字典。

（2）调用 student()函数时，将 3 个实参值传递给可变长度形参 stu，并组合为包含 3 个元素、名字为 stu 的字典。

（3）每个实参值应以“键＝值”的形式提供，键不需要加引号，值若是字符串则需要加引号，如示例中的“小明＝"河北省"”。

在定义函数与调用函数时，实参与形参可以如下对应关系出现：

（1）实参与形参都是对应的简单类型，如整型、浮点型、字符串等。

（2）实参和形参都是对应的组合类型，如列表、元组、字典、集合等。

（3）实参是简单类型，形参是组合类型，此时需在形参名前加单星号(*)或双星号(**)，前者把接收的实参值组合为元组，后者把接收的实参值组合为字典，此时对实参值有格式要求。

5.4　函数的嵌套与递归

在 Python 语言中，函数 f1 可以调用函数 f2，函数 f2 还可以再调用函数 f3，如此下去，便可形成函数的多级调用。函数的多级调用有两种形式：一是嵌套调用，二是递归调用。

5.4.1　函数嵌套

在函数的多级调用中，如果函数 f1、f2…fn 各不相同，则称为嵌套调用。

【例 5.11】 计算 1000～2000 的素数之和，用函数的嵌套调用实现。

分析： 要实现题目要求的功能，可以设计两个函数。一个是 total()，用于求若干个素数之和；另一个是 prime()，用于判定一个数是否为素数。主程序调用 total()函数，total()函数再调用 prime()函数，这样使整个程序的结构更为清晰，也降低了每个函数的编写难度。

```
#P0511.py
def prime(n):                          #定义函数,判断 n 是否是素数
    for i in range(2,n):
        if n%i==0:
            return False               #如果不是素数,则返回 False
    else:
        return True                    #如果是素数,则返回 True
def total(n1,n2):                      #定义函数,计算 n1～n2 的素数之和
    total=0
    for i in range(n1,n2+1):           #i 的取值范围为 n1～n2
        if (prime(i)):                 #调用函数 prime()判断 i 是否是素数
            total+=i                   #如果是素数,则进行累加
    return total                       #返回累加和
#主程序
print(total(1000,2000))                #调用函数计算 1000～2000 的素数之和
```

程序执行结果如下：

```
200923
```

说明：

(1) 定义 prime()函数和 total()函数时都用到了变量 i，这是允许的，两个变量 i 在各自的函数内起作用，互不影响。

(2) 定义 total()函数时，函数名和其中的变量都用到了标识符 total，这也是允许的。

5.4.2 函数递归

所谓递归，就是将一个较大的问题归约为一个或多个子问题的求解方法。这些子问题比原问题简单，且在结构上与原问题相同。递归在 Python 语言中的含义是：在函数的多级调用中，如果函数 f1、f2…fn 中有相同的函数，即存在某个函数直接或间接地调用自己，则称为递归调用。递归调用可以看作嵌套调用的特例。

递归过程包括递推和回归两部分。

例如，求 5 的阶乘就可以用递归方法，过程如下：

递推过程：5!＝5×4!→4!＝4×3!→3!＝3×2!→2!＝2×1! →1!＝1。

回归过程：1!＝1→2!＝2×1＝2→3!＝3×2＝6→4!＝4×6＝24→5!＝5×24＝120。

一个问题能够用递归方法解决的关键在于递推有结束点，如阶乘问题的 1!＝1。如果递推没有结束点，就无法回归，问题将得不到有效的解决，表现在程序中就是程序的执行永远也不能结束，这可以称为无效递归。

【例 5.12】 用递归方法求 n!。

```
#P0512.py
def fact(n):                    #定义递归函数
    if (n==0 or n==1):
        fac=1
    else:
        fac=n*fact(n-1)     #调用 fact()函数
    return fac
#主程序
m=int(input("请输入求阶乘的数 m="))
print("{}!={}".format(m,fact(m))
```

【例 5.13】 用递归方法求解汉诺塔问题。

传说在古代印度的贝拿勒斯神庙里安放了一个黄铜座，座上竖有三根宝石柱子。在第一根柱子上，自上而下按照从小到大的顺序放有 64 个直径不同的金盘子，形成一座金塔，如图 5.1 所示，即所谓的汉诺塔(Hanoi)，又称梵天塔。天神让庙里的僧侣们将第一根柱子上的 64 个盘子借助第二根柱子全部移到第三根柱子上，即将整个金塔搬迁，同时定下如下 3 条规则：

(1) 每次只能移动一个盘子。

(2) 盘子只能在 3 根柱子上来回移动，不能放在别处。