第5章 函 数

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个模块，每一个模块可以再划分成更小的模块，直至每一个小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小模块的程序，最后再把这若干个模块组织成一个完整程序。这样一种程序设计模式，可使编写出的程序结构清晰，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。

5.1 函数定义

在 Python 中，一个小模块的功能由一个函数来实现，一个 Python 程序可由若干个函数组成，函数之间通过调用关系形成一个完整的程序。在 Python 中，函数要先定义，后调用。

定义函数的语法格式如下：

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

示例：求 3 个数的平均值的函数定义如下：

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

结合上述示例，对函数定义简要说明如下：

① def 是关键字，用于定义函数。

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来使用这个函数，函数调用时要用到函数的名字。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 scoreAverage()。

③ 形式参数表用于给函数运算提供所需的数据，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，但一对圆括号不能省略。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。

④ 函数定义的第一行以冒号(:)结束。

⑤ 函数体由实现函数功能的一条或若干条语句组成。

⑥ return 语句用于把函数的结果带回调用该函数的调用函数(调用程序)，如果函数没有返回值，可以不写 return 语句。例如，如下函数：

```
def display(n):
    for i in range(n):
        print("**************")
```

的功能是输出若干行的星号串，只是完成特定的动作，没有值需要带回到调用程序，所以没有 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，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。函数调用语法格式如下：

函数名(实际参数表)

对于前面定义的函数 average()，先通过键盘为 x、y、z 输入值：

```
x=int(input("x="))
y=int(input("y="))
z=int(input("z="))
```

然后，函数调用形式如下：

```
print(average(x,y,z))
```

函数调用时的实际参数表应与函数定义时的形式参数表在数量上要一致，把每个实际参数（简称实参）的值传递给对应的形式参数（简称形参），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。

调用一个函数时，首先计算实参表中各表达式的值，然后函数调用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】 定义函数求 3 个数的平均值。

```
#P0501.py
def average(num1,num2,num3):                    #定义函数计算 3 个数的平均值
    ave=(num1+num2+num3)/3
    return ave
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)
```

上述程序包括两大部分：函数定义和函数调用。程序的执行过程如下：

① 先执行调用程序中的如下语句：

```
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 暂停调用函数的执行，转去执行 average()函数中的语句：

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

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

④ 当执行完 return ave 语句后，返回调用函数中，继续执行如下语句：

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

其中，average(score1,score2,score3)函数调用的值就是被调用函数中通过 return 语句返回的 ave 的值。

【例 5.2】 定义函数求 3 个数的中间数。

```
#P0502.py
def medium(num1,num2,num3):                    #定义函数
    if num1<num2:
        num1,num2=num2,num1                    #两者之中的大数存入 num1
    if num1<num3:
        num1,num3=num3,num1                    #两者之中的大数存入 num1
    if num2<num3:
        num2,num3=num3,num2                    #两者之中的大数存入 num2
    return num2
print("请输入 3 个数: ")
a=eval(input("a="))
b=eval(input("b="))
c=eval(input("c="))
med=medium(a,b,c)                              #通过调用函数求 3 个数中的中间数
print("3 个数的中间数={}".format(med))
```

5.3 函数的参数传递

调用函数与被调用函数之间的联系是通过参数传递来实现的。定义函数时，系统并不给函数的形参分配存储单元，函数被调用执行时，系统才为各形参分配存储空间，并把对应

的实参的值传递给形参。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

由于 Python 中的变量并不是直接存储某个值，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，实际上将实参所指向对象的地址传递给了形参（为便于理解，我们仍简单地称之为把实参值传递给了形参）。如果实参对象是不可变对象（数值、字符串、元组等），则有新值就分配新的存储空间，所以执行被调用函数时形参值的改变不会影响到实参。

【例 5.3】 不改变实参值的参数传递方式。

```
#P0503.py
def increa(x,y):
    print("x={},y={}".format(x,y))
    x=int(x*1.1)
    y=int(y*1.2)
    print("x={},y={}".format(x,y))
a=20
b=50
print("a={},b={}".format(a,b))
increa(a,b)
print("a={},b={}".format(a,b))
```

程序执行时的参数传递过程如下（程序从语句 a＝20 开始执行）。

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 increa()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，返回主函数之前，各变量的值如下：

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

返回到主程序后，各变量的值如下（被调用函数中的各变量被回收）：

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

所以，程序执行结果如下：

```
a=20,b=50           #进入函数前变量 a 和 b 的值
x=20,y=50           #刚进入函数时变量 x 和 y 的值
x=22,y=60           #退出函数前变量 x 和 y 的值
a=20,b=50           #退出函数后变量 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，就有了上述程序执行结果。

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)，操作可在原数据上进行，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 5.4】 改变实参值的参数传递方式。

```
#P0504.py
def fun(list2,n):
    print("list2=",list2)
    for i in range(n):
        list2[i]=int(list2[i]＊1.2)
    print("list2=",list2)
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 list2(实际上也是 list1) 的操作可在原值上进行。

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 5.5】 基于位置的参数传递方式。

```
#P0505.py
def disp(x,y):
    print("x={},y={}".format(x,y))
a=x=20
b=y=30
disp(a,b)          #实参 a、b 的值分别传递给形参 x、y
disp(x,y)          #实参 x、y 的值分别传递给形参 x、y
disp(y,x)          #实参 y、x 的值分别传递给形参 x、y
disp(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
x=20,y=30
x=30,y=20
x=20,y=20
```

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。

5.3.4 关键字参数

在调用函数时，也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 5.6】 基于关键字的参数传递方式。

```
#P0506.py
def totalScore(math,language,computer):
    total=math+language+computer
    average=total//3
    return total,average
print("请输入 3 门课的成绩：")
math1=int(input("数学="))
language1=int(input("语文="))
computer1=int(input("计算机="))
total,average=totalScore(math=math1,computer=computer1,language=language1)
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来传递的。如果需要，也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 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（覆盖掉默认形参值）。

【例 5.7】 有默认值的参数传递方式。

```
#P0507.py
def area(r=1.0,pi=3.14):
    return r * r * pi
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
```

程序的执行结果如下：

```
面积=3.14
面积=21.23
面积=21.24
```

第一次调用函数 area()时，由于没有实参，所以程序执行进入函数 area()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，所以进入函数 area()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，所以进入函数 area()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进行对应的，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。

如下的默认值设定是正确的。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 x，y 和 z 取默认值：

```
fun1(x,y=5,z=10)
```

如下的默认值设定是错误的。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 x(覆盖掉 x 的默认值)，而 y 得不到相应的取值，会引起错误：

```
fun2(x=5,y,z=10)
```

5.3.6 可变长度参数

在 Python 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。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(*)，把接收来的多个实参组合在一个元组内，以形参名为元组名；双星号参数是在形参名前加两个星号(**)，把接收来的多个实参组合在一个字典内，以形参名为字典名。

【例 5.8】 单星号可变长度参数。

```
#P0508.py
def sumScore(*score):                    #单星号形参用于把接收到的实参组合为元组
    sum=0
    for i in score:
        sum+=i
    ave=sum//len(score)
    return sum,ave
sum1,ave1=sumScore(78,62,81)             #第一次调用,接收 3 个实参
print("总成绩={},平均成绩={}".format(sum1,ave1))
sum2,ave2=sumScore(95,61,72,87)          #第二次调用,接收 4 个实参
print("总成绩={},平均成绩={}".format(sum2,ave2))
```

程序运行结果如下：

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

结合示例说明如下：

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组合为一个名字为 score 的元组。

② 第一次调用 sumScore()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 sumScore()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 5.9】 双星号可变长度参数。

```
#P0509.py
def student(**stu):                    #双星号形参用于把接收到的实参值组合为字典
    cnt=0
    for item in stu.values():
        if item=="河北省":
            cnt+=1
    return cnt
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))
```

结合示例说明如下：

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"”。

5.3.7 序列解包

在前面的介绍中，实参为简单类型值(整数、实数、字符串等)，形参为简单变量(整型变量、实型变量、字符串变量等)和组合类型变量(元组变量、字典变量等)。除此之外，Python 还支持实参为组合类型值的情形，如可以是元组、列表、字典、集合等。

【例 5.10】 计算几天的平均温度。

```
#P0510_1.py
def aveTemp(*temp):
    sum_temp=0
    for i in temp:
        sum_temp+=i
    ave_temp=sum_temp//len(temp)
```

```
    return ave_temp
temp1=[34,32,27,28,32]
ave1=aveTemp(temp1)
print("平均温度=",ave1)
```

执行该程序，会报告如下错误提示信息：

```
==========RESTART: C:/Users/Administrator/Desktop/P0510.py==========
Traceback (most recent call last):
  File "C:/Users/Administrator/Desktop/P0510.py", line 9, in<module>
    ave1=aveTemp(temp1)
  File "C:/Users/Administrator/Desktop/P0510.py", line 5, in aveTemp
    sum_temp+=i
TypeError: unsupported operand type(s) for+=: 'int' and 'list'
```

错误在于程序中进行了 Python 不支持的“整数与列表相加”操作，其原因在于，由于实参 temp 是一个列表，形参 temp 的元素为列表，所以程序中的语句

```
for i in temp:
    sum_temp+=i
```

是要实现一个整数(sum_temp)和一个列表(i)的加法，因此导致了错误。

函数调用部分可以改写如下：

```
temp=[34,32,27,28,32]
ave1=aveTemp(temp[0],temp[1],temp[2],temp[3],temp[4])
print("平均温度=",ave1)
```

再执行此程序会得到正确的结果。

但上述方式需要编程人员写出组合类型的元素，当元素比较多时就不大好处理，Python 提供了一种简单的书写方式：在组合类型实参的前面加一个星号(*)，将组合类型实参值自动解包为元素传递给形参。函数调用部分进一步改写如下：

```
temp1=[34,32,27,28,32]
ave1=aveTemp(*temp1)
print("平均温度=",ave1)
```

此时执行也能得到正确结果。

如下形式也能得到正确结果，形参和实参都是列表类型(比加星号还简单)：

```
#P0510_2.py
def aveTemp(temp):
    sum_temp=0
    for i in temp:
        sum_temp+=i
    ave_temp=sum_temp//len(temp)
    return ave_temp
temp1=[34,32,27,28,32]
ave1=aveTemp(temp1)
```

```
print("平均温度=",ave1)
```

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(**)，前者把接收到的实参值组合为元组，后者把接收到的实参值组合为字典。

④ 实参是组合类型，形参是简单类型，可以通过序列解包的形式把实参值传递给形参，此时需要在实参名前加写一个星号(*)，计算平均温度的程序也可改写如下：

```
#P0510_3.py
def aveTemp(t1,t2,t3,t4,t5):
    sum_temp=t1+t2+t3+t4+t5
    ave_temp=sum_temp//5
    return ave_temp
temp1=[34,32,27,28,32]
ave1=aveTemp(*temp1)
print("平均温度=",ave1)
```

5.4 函数的嵌套与递归

在 Python 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】 求 100～200 中能够被 3 整除的数之和，用函数的嵌套机制实现。

问题分析：设计两个函数，一个是 sum()，用于求若干个数之和；另一个是 fun()，用于判定一个数是否能够被 3 整除。主程序调用函数 sum()，函数 sum()再调用函数 fun()。

```
#P0511.py
def fun(num):
    if (num%3==0):
        b=True                  #如果能够被 3 整除,设标记值为 True
    else:
        b=False                 #否则设标记值为 False
    return b                    #返回标记值
def sum(m,n):
    sum=0
    for i in range(m,n+1):      #取值范围为 m～n
        if (fun(i)):            #调用函数 fun()来判断 i 是否能被 3 整除
           sum+=i
    return sum
```