第5章

用函数实现模块化程序设计

前面已经介绍了数值、赋值语句、输入/输出、选择结构、循环结构以及列表等内容，这些编程方法只适合于解决简单的问题。在解决比较复杂的问题时，经常需要在程序中多次执行同一项任务。显然，反复编写或复制多份完成同一任务的代码并不是好的方法，这会使程序变得冗长且难以维护。各种高级语言都提供了代码复用的机制，Python 可以使用函数和类实现代码复用，本章学习函数。

5.1 函数调用

函数是一个“子程序”，也就是程序中的一段代码。函数实现代码复用的基本思想是给一个语句序列(代码块)取一个名称(函数名)，然后在程序的任何位置可以通过引用函数名称来执行这些语句。也可以说，函数是有名称的代码块。

Python 提供了很多内置函数，如前面程序中一直在使用的 print()、input()、len()等，也允许用户自己定义函数。通过函数名执行内置函数或自定义函数的语句序列称为“调用”函数。

5.1.1 函数调用格式

函数调用的一般格式为

函数名(实参列表)

例如，内置函数 pow(x,y)的功能是计算 x^y，即 x 的 y 次方。调用 pow()函数如下。

```
>>> pow(2,5)
32
```

其中，pow 是函数名，x 和 y 是形参(形式参数)，2 和 5 是传递给 pow 的实参(实际参数)。各个形参或实参间用逗号分隔。32 是函数计算以后得到的结果，称为返回值，因此说 pow(2,5)返回 32，如图 5.1 所示。

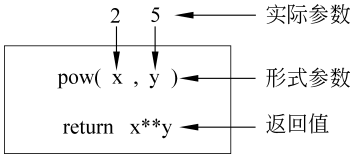

图 5.1 形参、实参与返回值

如果在表达式中调用函数时 Python 将函数调用替换为其返回值，例如表达式 pow(2,5)+8 与 32+8 等价，则结果为 40。

即使函数不接受任何参数，也要在函数名后使用()。()让 Python 调用指定的函数，如果不使用()，将输出错误。例如，只输入 print 时，Python 并不执行函数和输出空行，而是提示 print 指向一个函数。

```
>>> print
<built-in function print>
```

有些函数的部分形参会使用默认值。在函数调用时，这些参数既可以传实参，也可以不传实参而使用默认值。前面章节使用 print()函数中的 end 参数既设置了默认参数值，end 表示结束字符串，默认值为换行符'\n'。在输出时，print()函数将要输出的字符串(或其他类型对象转换的字符串)与 end 参数连接在一起，然后输出。例如：print('hello')实际输出的是'hello'+end，也就是输出'hello\n'；print('hello',end='\t')实际输出的是'hello\t'。

有默认值的形参在传实参时一般要使用参数名，并且应该放在没有默认值的参数后面，如 print('hello',end='\t')。

函数参数表可以包含任意数量的参数，参数数量与函数定义有关，例如

```
>>> abs(-4)                    #abs()函数用于计算一个数的绝对值
4
>>> max(6,-2)                  #max()函数和 min()函数分别用于求一组数的最大值和最小值
6
>>> min(2,-4,6,26.5)
-4
```

5.1.2 不返回值的函数

有些函数无返回值(如 print())，Python 在执行无返回值的函数时返回一个特殊值 None，表示无返回值。

```
>>> print('hello')
hello
>>> x=print('hello')
hello
>>> print(x)
None
```

None 既不是字符串，也不是数字，不能用它进行任何有意义的计算。

5.2 定 义 函 数

在 Python 中，定义一个函数时，需要指定函数的名称和语句序列。

5.2.1 函数定义的一般形式

在 Python 中，函数定义的一般形式为

```
def 函数名([形式参数 1,形式参数 2,……]):
    语句块
```

例 5.1 定义 maxVal()函数求两个数的最大值。

```
def maxVal(x,y):
  if x>y:
    return x
  else:
    return y
```

函数定义以 def 关键词开头，后接函数名称和圆括号()。括号里面是函数的参数，参数可以是零个或多个，各个参数间用逗号分隔，冒号后面对应缩进的代码块是函数体。函数体可以是任何一段 Python 代码。函数如果需要有返回的计算结果，可利用 return 关键字进行返回。return 关键字后面可以是数值或其他类型的数据(如字符串、列表、元组等)，也可以是变量或表达式。在执行到 return 语句时会结束对函数的调用，一个函数可能会有多个 return 语句。

函数定义时并不会执行，函数定义中的语句只有在被调用时才会执行。例如下面这个函数调用。

```
maxVal(3, 4)
```

执行时，将 3 传给 x，4 传给 y。在执行 if 语句时，条件为假，转 else 部分，返回 4，函数结束。

当函数被调用时，会执行如下过程。

① 求实参表达式的值，将求值结果传给函数的形参。例如，调用 maxVal(3+4,z)会在解释器求值后将 7 传给形参 x，将变量 z 的值传给形参 y。

② 执行函数体的第一条语句。

③ 执行函数体中的代码，直至遇到 return 语句。return 后面的表达式的值就是函数调用的值；如果没有语句可以继续执行，函数返回的值为 None；如果 return 后面没有表达式，函数返回的值也为 None。

例 5.2 定义函数，根据半径计算圆面积。

```
def calCircleArea(r):
  return 3.14159*r*r
r=float(input("please input radius:"))
area=calCircleArea(r)
print("area=",area)
```

由于函数定义时并不会执行，因此程序执行的第一条语句是输入语句，获得半径 r。然后将 r 作为参数调用函数 calCircleArea()，函数执行完成后，返回值赋值给 area 变量，最后输出。

例 5.3 定义函数，对列表求和。

```
def add_list(list):
  sum=0
  for i in list:
    sum=sum+i
  return sum
array=[1,2,3,4,5,6]
s=add_list(array)
print("sum=",s)
```

函数 add_list()的参数是列表。在函数体中，利用 for 语句遍历列表元素求和，最后将和作为返回值。add_list()函数的功能与 sum()相同。

如果实参不是列表，在执行 for 语句时会产生错误。为避免产生这样的错误，可以在函数中添加参数类型判断。使用 type()函数求列表变量的类型为＜class 'list'＞。使用 str()函数将其转换为字符串后，如果字符串中包含'list'，则为列表，否则直接进行 return。修改后的函数 add_list()在参数为非列表类型时，返回 None。

```
def add_list(list):
  if not 'list' in str(type(list)):                    #list 不是列表时为真
    return                                             #结束函数调用,返回 None
  sum=0
  for i in list:
    sum=sum+i
  return sum
```

5.2.2 参数传递方式

Python 处理参数的方式要比其他语言更加灵活，有多种参数传递方式。

1. 按位置传参数

最常用的参数传递方法是按位置传参数，传入参数的值是按照顺序依次复制的。例如，函数定义为 maxVal(x,y)，调用时 maxVal(3, 4)的第 1 个实参 3 传给第 1 个形参 x，第 2 个实参 4 传给第 2 个形参 y。

按位置传参数必须熟记每个位置参数的含义，这在参数个数很多时并不容易。

2. 指定参数名称

为避免按位置传参数带来的难以记忆问题，调用函数时可以指定对应形式参数的名

称，甚至可以采用与函数定义不同的顺序传递参数。

例 5.4 使用函数将十进制数转换成其他进制数。

将十进制转换为其他进制时使用除基取余法，流程如图 5.2 所示。第一个余数为最低位。

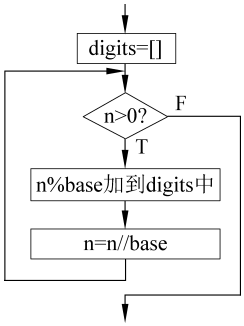

图 5.2 除基取余流程

```
def convert(n,base):
  digits=[]
  while n>0:
    digits.append(n%base)
    n=n//base
  return digits
def printConvert(digits):
  p=len(digits)-1                                    #逆序输出
  while p>=0:
    print(digits[p],end=" ")
    p=p-1
printConvert(convert(base=2,n=24))
```

3. 指定默认参数值

定义函数时，可以为参数指定默认的参数值。调用函数时，如果参数表中没有该参数，那么该参数使用默认值。

例 5.5 使用默认参数值。

```
Def greet(name,greeting='hello',end="!"):
  print(greeting,name+end)
greet("wangming")
greet("wangming","Good morning")
greet("wangming",greeting="Hi")
```

程序运行结果为

```
hello wangming!
Good morning wangming!
Hi wangming!
```

函数可以根据需要使用任意数量的默认参数，但是带默认值的参数不能位于没有默认值的参数的前面。

4. 可变数量参数

函数形参变量名前加 * 表示不限定参数数量，实参以元组形式传递给形参。

例 5.6 按位置传不限数量参数。

```
def greet(*name,greeting='hello',end="!"):
  print(name)
```

```
    names=""
    for n in name:                              #将 name(元组形式)元素连接成一个字符串
      names=names+n+" "
    print(greeting,names+end)
greet("wangming")                               #name=('wangming',)
greet("wangming","liping","Good morning")
                                    #name=('wangming', 'liping', 'Good morning')
greet("wangming","liping","zhangling",greeting="Hi")
                        #name=('wangming', 'liping', 'zhangling'),greeting="Hi"
```

程序运行结果为

```
('wangming',)
hello wangming !
('wangming', 'liping', 'Good morning')
hello wangming liping Good morning !
('wangming', 'liping', 'zhangling')
Hi wangming liping zhangling !
```

函数形参变量名前加**表示不限定参数数量，参数名称和参数值以字典形式传递给形参。参数的名称是字典的键，对应参数的值是字典的值。

例 5.7 编写函数，求矩形、圆和梯形面积。

```
def calArea(type,**args):
    print(args)                                 #args 是一个字典,包含参数数量不定
    if type=="rectangle":                       #矩形
        return args["width"]*args["height"]
    elif type=="circle":                        #圆
        return 3.14159*args["radius"]*args["radius"]
    elif type=="trapezoid":                     #梯形
        return (args["top"]+args["bottom"])*args["height"]/2
print("rectangle area=",calArea("rectangle",width=2,height=3))
print("circle area=",calArea("circle",radius=2))
print("trapezoid area=",calArea("trapezoid",top=2,bottom=5,height=3))
```

程序运行结果为

```
{'width': 2, 'height': 3}
rectangle area= 6
{'radius': 2}
circle area= 12.56636
{'top': 2, 'bottom': 5, 'height': 3}
trapezoid area= 10.5
```

5.2.3 参数类型

从变量指向地址的内容是否可以变化的角度看，数值、字符串、元组是不可变类型，而

列表和字典则是可变类型(如图 5.3 所示)。

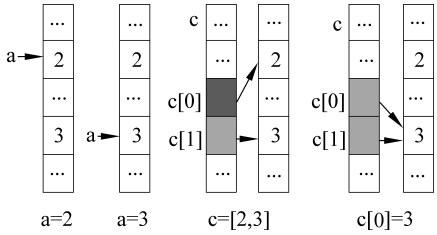

图 5.3　不可变类型与可变类型

在数值、字符串、元组变量作为函数参数时,如果在函数中修改形参变量的值,不会影响实参变量本身。

在列表、字典变量作为函数参数时,如果在函数中修改列表或字典内容,函数调用结束后的实参值也会发生变化。

例 5.8　交换函数对不同类型参数的影响。

```
def swap(x,y):
  x,y=y,x                                  #修改形式参数 x、y,将 x 和 y 的值互换
def swaplist(list):
  list[0],list[1]=list[1],list[0]          #修改形式参数列表的值,将列表第 0 项元素和第
                                           #1 项元素的值互换
a=2
b=3
c=[2,3]
swap(a,b)
swaplist(c)
print("after call swap:","a=",a,"b=",b)
print("after call swaplist:","c=",c)
```

程序运行结果为

```
after call swap: a= 2 b= 3
after call swaplist: c= [3, 2]
```

程序执行过程如图 5.4 所示。数值对象 2 和 3 存储在内存中,变量 a 是存储 2 的地址的名称,b 是存储 3 的地址的名称,或者说 a 指向 2,b 指向 3。在调用函数 swap()时,实

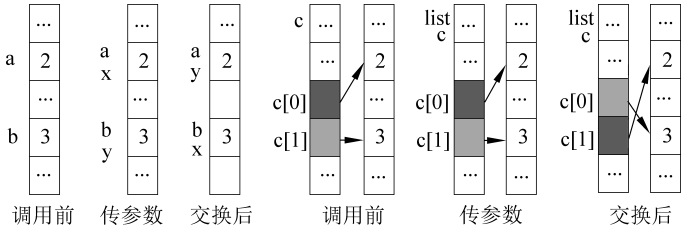

图 5.4　交换过程

际参数 a 传递给 x，变量 x 取值为 2，也就是存储 2 的地址又有了一个名称 x，或者说 x 也指向 2。x 和 y 的值交换后，x 指向 3，y 指向 2，a 和 b 的值不变。列表 c 的 c[0]位置存储的是 2 的地址，访问 c[0]时可以通过地址找到 2。调用函数 swaplist()时，实际参数 c 传递给形式参数 list，list 是列表 c 的另一个名称，在交换列表 list[0]和 list[1]时，也是修改 c[0]和 c[1]的值。因此，在列表 c 作为参数时，在函数中修改 list 就是修改 c。函数调用结束后，c 的内容也发生了变化。

例 5.9 输入一串字符，编写一个函数，统计字符串中小写字母的个数。

```
def staLower(inputStr):
  count=0
  for ch in inputStr:
    if ch>='a' and ch<='z':            #小写字符大于或等于'a',并且小于或等于'z'
      count+=1
  return count
inputStr=input("please input a string:")
print("count=",staLower(inputStr))
```

程序运行后，输入一串字符，输出小写字母个数。例如

```
please input a string:abcdEFG12340xy;*
count= 6
```

5.2.4 lambda()函数

lambda()函数是一种匿名函数，可接受任意数量的参数，但只能有一个表达式。定义 lambda()函数的一般格式为

lambda 参数表 : 表达式

其功能是执行表达式并返回结果。

```
>>> x=lambda a:a+10
>>> print(x(5))
15
```

等效于

```
def x(a):
  return a+10
>>> p = lambda x,y:x+y
>>> print(p(4,6))
10
```

5.2.5 pass 语句

在 Python 中，def 语句、if 语句、for 语句和 while 循环语句必须有一个语句体，即非

空缩进代码块。如果遗漏了代码块，则解析程序时会发生语法错误。有些情况下，块中的代码实际上不需要做任何事情，这时仍然需要在语句体中添加代码。出于这个原因，Python 提供了 pass 语句，它不执行任何操作，但仍然是一个有效的语句。例如

```
if n%2==0:
  pass                                    #n 为偶数时，不执行任何操作
else:
  print(n)                                #n 为奇数时，输出 n
```

在编写函数时，当代码体还没有实现时，可以用 pass 语句作为代码体。

5.3 变量的作用域

使用函数涉及的一个重要问题是变量的作用域。变量的作用域是指变量在程序的哪些地方可以访问。在语句中如果使用在该位置无效的变量，运行时将会产生错误。

例 5.10 在程序中使用函数中使用的变量。

```
def calRectangle(w,h):
  area=w*h
  return area
print("Rectangle area=",calRectangle(2,3))
print(area)
```

程序的执行结果如图 5.5 所示。

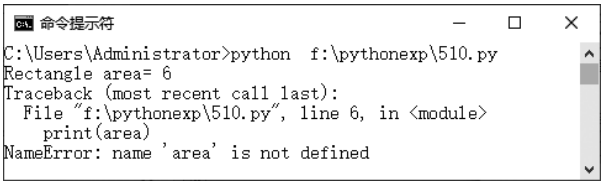

图 5.5 变量未定义错误

虽然在执行 print(area)语句之前在函数中已经使用了 area 变量，但是在 print 语句位置仍然找不到 area 变量，说明 area 变量在此处已经失效。

5.3.1 局部变量

在一个函数内部使用的变量是内部变量，它只在本函数范围内有效，也就是说只有在本函数内才能使用它们，在此函数之外是不能使用这些变量的。这称为局部变量。

图 5.6 中的 area 是在函数 calRectangle()函数中使用的变量，只在函数中有效，在主流程中是无效的，因此在执行 print(area)语句时找不到该变量。

在程序中添加一条赋值语句 area=0，运行时就不会出现错误了。程序中使用的各个

变量的作用域如图 5.6 所示。

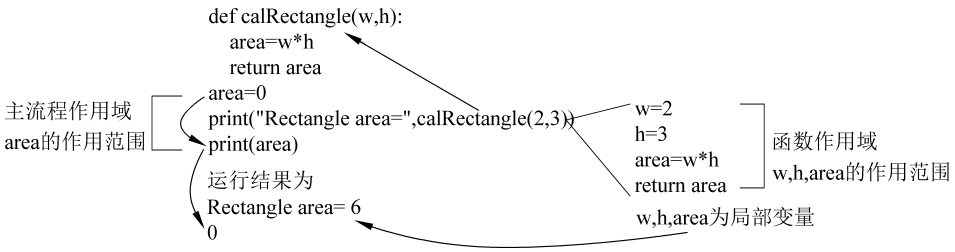

图 5.6　局部变量作用域

形式参数也是局部变量，例如形参 w 和 h 也只在函数中有效。

不同作用域中可以使用相同名称的变量，它们代表不同的对象，互不干扰。例如，在主流程作用域中使用的 area 和在 calRectangle()函数中使用的 area 是同名变量。print(area)语句使用的 area 的值为 0，return area 语句使用的 area 的值为 6，这两个 area 是不同的变量。这表明在设计函数时对变量的命名可以是任意的，不会对其他程序产生影响。同样，在调用函数时，也不需要考虑函数中使用了哪些变量名。

5.3.2　全局变量

在函数外面声明的变量称为全局变量，程序中的任何函数或代码都可以读取全局变量。然而，在函数中给全局变量赋值时需要特别小心。

例 5.11　在函数中使用程序中声明的变量。

```
def say_hello():
  print('hello',name+'!')
def change_name(new_name):
  name=new_name
  print("function name:",name)
name="wangming"
say_hello()
change_name("wangmingli")
print(name)
```

程序运行结果为

```
hello wangming!
function name: wangmingli
wangming
```

程序中的 name 变量是一个全局变量，因为它是在函数外面声明的。在调用 say_hello()函数时，name 变量已经存在，say_hello()读取 name 的值并输出。在调用 change_name()时，将 name 修改为"wangmingli"，然而全局变量 name 的值并没有改变，在输出时仍然是"wangming"。实际上 Python 将 change_name()函数中的 name 变量看成局部