第 5 章

函数与函数式编程

函数是一段封装好的、实现特定功能的代码段，用户不需要关心程序的实现细节，可以通过函数名及其参数直接调用。

Python 内置了丰富的函数资源，用户可以在程序中直接调用这些内置函数。程序开发人员也可以根据实际应用需要定义函数，从而方便随时调用，并能提高应用程序的模块性和代码的复用率，增强了代码的重用性和可读性。

本章将对如何定义和调用函数，以及函数的参数、变量的作用域、函数的递归与调用、函数式编程等进行详细介绍。

5.1 内 置 函 数

Python 中内置了丰富的函数资源，可以用来进行数据类型转换与类型判断、统计计算、输入/输出等操作。比如，使用 dir(__builtins__)可以查看内置函数。前面章节中使用的 input()和 print()就是内置函数。

内置函数可以在程序中直接调用，其语法格式如下。

```
函数名(参数 1,参数 2,参数 3,…)
```

内置函数说明如下。

(1) 调用函数时，函数名后面必须加一对圆括号。

(2) 函数通常都有一个返回值，表示调用的结果。

(3) 不同函数的参数个数不同，有的是必选的，有的是可选的。

(4) 函数的参数值必须符合要求的数据类型。

(5) 函数可以嵌套调用，即一个函数可以作为另一个函数的参数。

5.2 自定义函数与调用

Python 除了可以直接使用标准函数外，还支持自定义函数，即通过将一段实现单一功能或相关联功能的代码定义为函数，来达到“一次编写，多次调用”的目的。使用函数可以提高代码的重用率。

5.2.1 函数的定义

函数定义的语法格式如下：

```
def function_name(arguments):
    function_block
    return[expression]
```

函数定义的说明如下。

(1) 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号。

(2) function_name 是用户自定义的函数名称。

(3) arguments 是零个或多个参数，且任何传入参数必须放在圆括号内。如果有多个参数，则参数之间必须用英文逗号分隔。即使没有任何参数，也必须保留一对空的圆括号。括号后边的冒号表示缩进的开始。

(4) 最后必须跟一个冒号(:)，函数体从冒号开始，并且缩进。

(5) function_block 是实现函数功能的语句块。

(6) 在函数体中，可以使用 return 语句返回函数代码的执行结果，返回值可以有一个或多个。如果没有 return 语句，则默认返回 None(空对象)。

(7) 如果想定义一个空函数，可以使用 pass 语句作为占位符。

5.2.2 函数的调用

调用函数也就是执行函数。如果把创建的函数理解为一个具有某种用途的工具，那么调用函数就相当于使用该工具。定义一个函数，但不调用，那么这个函数中的代码就不会运行。调用函数的语法格式如下：

```
function_name (arguments)
```

参数说明如下。

- function_name：函数名称，即要调用的函数名称，必须是已经创建好的。
- arguments：可选参数，用于指定各个参数的值。如果需要传递多个参数值，则各个参数值之间使用逗号分隔；如果该函数没有参数，则直接写一对圆括号即可。

5.2.3 函数的返回值

在 Python 中，可以在函数体内使用 return 语句为函数指定返回值，从而将函数的处理结果返回给调用它的程序。该返回值可以是任意类型，若函数没有返回值，可以省略 return 语句。return 语句是函数的结束标志，无论 return 语句出现在函数的什么位置，只要得到执行，就会直接结束函数的执行。

return 语句的语法格式如下：

```
return [value]
```

参数说明如下。

value：可选参数，用于指定要返回的值，可以返回一个或多个值。如果返回一个值，那么 value 中保存的值可以为任意类型。如果返回多个值，那么在 value 中保存的是一个元组。

当函数中没有 return 语句时，或者省略了 return 语句的参数时，将返回 None，即返回空值。

【例 5-1】 自定义函数名称为 fun_area()的函数，用于计算矩形的面积，该函数包括两个参数，分别为矩形的长和宽，返回值为矩形的面积。

实现代码如下。

```
#计算矩形面积的函数
def fun_area(width,height):
    if str(width).isdigit() and str(height).isdigit():  #验证数据是否合法
        area = width * height                           #计算矩形的面积
    else:
        area = 0
    return area                                         #返回矩形的面积

w = 20                                                  #矩形的长
h = 15                                                  #矩形的宽
area = fun_area(w,h)                                    #调用函数
print(area)
```

运行结果如下。

```
300
```

5.3　函数参数的传递

在使用函数时，经常会用到形式参数和实际参数，两者之间的区别如下。

(1) 形式参数简称为形参，在使用 def 定义函数时，函数名后面的括号里的变量称为形参。

(2) 在调用函数时提供的值或者变量称为实际参数，实际参数简称实参。

(3) 函数的参数传递是指将实参传递给形参的过程。

(4) 定义函数时不需要声明形参的数据类型，Python 解释器会根据实参的类型自动推断形参的类型。

(5) 形参与实参的关系：两者是在调用的时候进行结合的，通常实参会将取值传递

给形参之后进行函数过程运算，然后可能将某些值经过参数或函数符号返回给调用者。

函数既可以传递参数，也可以不传递参数。同样，函数既可以有返回值，也可以没有返回值。

根据实参的类型不同，可以分为将实参的值传递给形参，以及将实参的引用传递给形参两种情况。其中，当实参为不可变对象时，进行的是值传递；当实参为可变对象时，进行的是引用传递。实际上，值传递和引用传递的基本区别就是，进行值传递后，改变形参的值，实参的值不变；而在进行引用传递后，改变形参的值，实参的值也一同改变。

5.3.1 固定参数传递

直接将实参赋给形参，根据位置做匹配，即严格要求实参的数量与形参的数量以及位置均相同。也就是第 1 个实参传递给第 1 个形参，第 2 个实参传递给第 2 个形参，以此类推。

【例 5-2】 定义一个名称为 fun_bmi()函数，根据身高、体重计算 BMI 指数。要求：fun_bmi()函数包括 3 个参数，分别用于指定姓名、身高和体重，再根据公式：BMI＝体重/(身高×身高)，计算 BMI 指数，并输出结果。

实现代码如下。

```
def fun_bmi(person,height,weight):
    print(person+"的身高为:"+str(height)+"m 体重:"+str(weight)+"kg")
    bmi = weight/(height * height)
    print(person+"的 BMI 指数为:"+str(bmi))
    #判断身材是否正常
    if bmi < 18.5:
        print("你的体重过轻!")
    if bmi >= 18.5 and bmi < 24.9:
        print("正常范围,注意保持。")
    if bmi >= 24.9 and bmi < 29.9:
        print("你的体重过重!")
    if bmi >= 29.9:
        print("肥胖!")
#函数定义
fun_bmi("吴怡晴" , 1.78 , 75)
```

运行结果如下。

```
吴怡晴的身高为:1.78m 体重:75kg
吴怡晴的 BMI 指数为:23.671253629592222
正常范围,注意保持。
```

5.3.2 默认参数传递

Python 支持默认值参数，即在定义函数时可以为形参设置默认值。调用带有默认值

参数的函数时，可以通过传递实参值来替换默认值。如果没有给设置默认值的形参传值，则函数会直接使用默认值。

需要注意的是，定义函数时，默认值参数必须出现在形参表的最后，即任何一个默认值参数的右边都不能再出现没有默认值的普通位置参数，否则会提示语法错误。

默认值参数的定义格式如下：

```
def 函数名(…,形参名=默认值):
    函数体
```

【例 5-3】 定义函数 user_info()，设置参数默认值，调用时验证其功能。

实现代码如下。

```
#定义函数
def user_info(name,age,gender='女'):
    print("您的名字是{name},年龄是{age},性别是{gender}")
#调用函数
user_info('Tom',20)
user_info('Jack',18,'男')
```

运行结果如下。

```
您的名字是 Tom,年龄是 20,性别是女
您的名字是 Jack,年龄是 18,性别是男
```

定义函数时，为形参设置默认值需注意默认参数必须指向不可变对象。若使用可变对象作为函数参数的默认值时，多次调用可能会导致意料之外的情况。

5.3.3　未知参数个数传递

如果函数在定义时无法确定参数的具体数目，定义函数时可以在形参前添加星号“*”或“**”。

通过 *arg 和**kwargs 这两个特殊语法可以实现可变长参数。

arg 表示元组变长参数(参数名的前面有一个星号“”)，可以以元组形式接收不定长度的实参。

**kwargs 表示字典变长参数(参数名的前面有两个星号“*”)，可以以字典形式接收不定长度的键值对。

【例 5-4】 利用可变长参数定义函数 get_score()，可以根据姓名同时查询多人的成绩。

实现代码如下。

```
def get_score(**names):
    result = []
```

```
    for name in names:
        score = std_sc.get(name, -1)
        result.append((name, score))
    return result

std_sc = {'Merry': 95, 'Jack': 76, 'Rose': 88, 'Xinyi': 65}
print(get_score('Merry'))
print(get_score('Jack', 'Rose'))
print(get_score('Merry', 'Xinyi', 'Jack'))
```

运行结果如下。

```
[('Merry', 95)]
[('Jack', 76), ('Rose', 88)]
[('Merry', 95), ('Xinyi', 65), ('Jack', 76)]
```

如果要使用一个已有列表作为函数的可变参数，可以在列表的名称前加一个星号“*”。比如：

```
schoolname=['清华大学','北京大学','中国农业大学']
printschool(*schoolname)
```

使用可变参数需要考虑形参位置的问题。如果在函数中，既有普通参数，又有可变参数，通常可变参数会放在最后。若可变参数放在函数参数的中间或者最前面，普通参数需用关键字参数形式来传递参数，如可变参数后面的普通参数传值时不想使用关键字参数，那么就必须为这些普通参数指定默认值。

如果要使用一个已有字典作为函数的可变参数，可以在字典的名称前加两个星号“**”。

总之，在传递参数时，字典和列表（元组）的主要区别是字典前面需要加两个星号（定义函数与调用函数都需要加两个星号），而列表（元组）前面只需要加一个星号。

5.3.4 关键字参数传递

调用函数时，可以通过“形参名＝值”的形式来传递参数，称之为关键字参数传递。与位置参数相比，关键字参数传递可以通过参数名明确指定为哪个参数传值，因此参数的顺序可以与函数定义中的不一致。这样可以避免用户需要牢记参数位置的麻烦，使得函数的调用和参数传递更加灵活方便。

使用关键字参数传递时，必须正确引用函数定义中的形参名称。

【例 5-5】 定义一个函数 user_info()，可以通过关键字参数传递实参。

实现代码如下。

```
def user_info(name,age,gender):
```

```
    print("您的名字是{name},年龄是{age},性别是{gender}")
#函数调用
user_info('Tom',age=20,gender='女')
user_info('Jack',gender='男',age=18)
```

运行结果如下。

```
您的名字是 Tom,年龄是 20,性别是女
您的名字是 Jack,年龄是 18,性别是男
```

当位置参数与关键字参数混用时,位置参数必须在关键字参数的前面,关键字参数之间则可以不区分先后顺序。

5.4　变量的作用域

变量的作用域是指程序代码能够访问该变量的区域,如果超出变量的有效范围,访问时就会出现错误。在程序中,一般是根据变量的有效范围将变量分为局部变量和全局变量。

5.4.1　局部变量

局部变量是指在函数内部定义并使用的变量,它只在函数内部有效。即函数内部的变量只在函数运行时才会创建,在函数运行完毕之后,局部变量将无法进行访问。所以,如果在函数外部使用函数内部定义的变量,会显示 NameError 异常。

【例 5-6】 局部变量的使用示例。

实现代码如下。

```
def my_add(x, y):
    print("x=" + str(x) + ";" + "y=" + str(y))
    s = x + y
    return s
```

在函数内部定义的变量一般为局部变量,其作用范围限定在这个函数内,当函数执行结束后,局部变量会自动删除,不可以再访问。

5.4.2　全局变量

与局部变量对应,全局变量是能够作用于函数内外的变量。在函数外部定义的变量称为全局变量,其作用范围是整个程序。全局变量可以在当前程序及其所有函数中引用。

全局变量可通过以下两种方式定义。

(1) 如果一个变量,在函数外定义,那么不仅在函数外部可以访问它,在函数内部也可以访问。在函数体以外定义的变量是全局变量。

(2) 在函数体内定义,使用 global 关键字修饰后,该变量也就变为全局变量。在函数体外也可以访问到该变量,并且在函数体内还可以对其进行修改。

【例 5-7】 全局变量的使用示例,其中 global_num 是全局变量,my_add()函数内部定义的 local_num 是局部变量。

实现代码如下。

```
def my_add():
    local_num = 3                          #局部变量
    return global_num + local_num

global_num = 5                             #全局变量
print(my_add())                            #结果:8
print(global_num)                          #结果:5
print(local_num)                           #报错,local_num 变量没有定义
```

当函数内的局部变量和全局变量重名时,该局部变量会在自己的作用域内暂时隐藏同名的全局变量,即只有局部变量起作用。

【例 5-8】 global 关键字使用示例,在函数内使用 global 关键字声明了全局变量 x 及其操作,将其值修改为 3。

实现代码如下。

```
def my_add():
    global x                               #声明全局变量
    print(x)                               #结果:5
    x = 3                                  #修改变量值
    return x + x

x = 5
print(my_add())                            #结果:6
print(x)                                   #结果:3
```

通过 global 关键字可以在函数内定义或者使用全局变量。如果要在函数内修改一个定义在函数外部的变量值,则必须使用 global 关键字将该变量声明为全局变量,否则会自动创建新的局部变量。

5.5 函数的递归与嵌套

5.5.1 函数的递归函数

程序调用自身的编程方法称为递归。函数的递归调用是函数调用的一种特殊情况。函数反复地自己调用自己,直到某个条件满足时就不再调用了,然后一层一层地返回,直

到该函数第一次调用的位置。

递归作为一种算法在程序设计语言中被广泛应用。通常可以把一个大型复杂的问题层层转化为一个与原问题相似的、规模较小的问题进行求解。递归策略只需要少量的程序就可以描述出解题过程所需要的多次重复计算，大幅减少了程序的代码量。

递归函数在定义时必须有边界条件，即递归停止的条件，否则函数将永远无法跳出递归，陷入死循环。

【例 5-9】 利用递归函数计算 5!。

实现代码如下。

```
def fn(num):
    if num==1:
        result=1
    else:
        result=fn(num-1) * num
    return  result
n=int(input("请输入一个正整数:"))
print("%d! ="%n, fn(n))
```

运行结果如下。

```
请输入一个正整数:5
5! = 120
```

接下来，通过图来描述阶乘 5! 算法的执行原理，如图 5-1 所示。

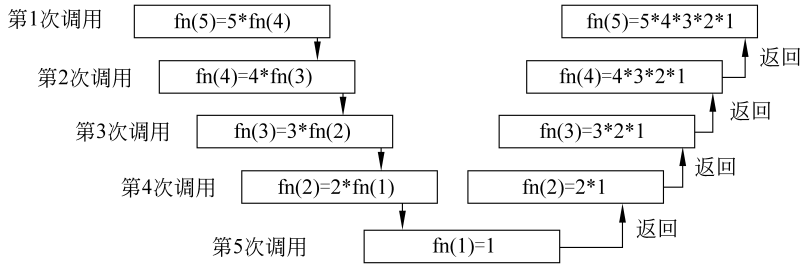

图 5-1　计算阶乘 5—1! 的执行过程

由上述例子可以看出，递归函数具有如下特征。

(1) 递归函数必须有一个明确的结束条件。

(2) 递归的递推(调用)和回归(返回)过程，与入栈和出栈类似。这是因为在计算机中，函数的调用其实就是通过栈这种数据结构实现的。每调用一次函数，就会执行一次入栈操作，每当函数返回，就执行一次出栈操作。

递归结构往往消耗内存较大，因此能用迭代解决的问题尽量不要用递归。

5.5.2 函数的嵌套

函数的嵌套是指在函数中调用另外的函数。这是函数式编程的重要结构，也是我们在编程中常用的一种程序结构。

【例 5-10】 函数的嵌套调用示例。

实现代码如下。

```
#计算 3 个数之和
def sum_num(a, b, c):
    return a + b + c

#求 3 个数平均值
def average_num(a, b, c):
    sumResult = sum_num(a, b, c)
    return sumResult / 3

result = average_num(1, 2, 3)
print(result)
```

运行结果如下。

```
2.0
```

5.6 函数式编程

函数式编程(Functional Programming)是一种抽象程度较高的编程范式，它的一个重要特点是编写的函数中没有变量。这就解决了在函数中定义、使用变量导致的输出不确定等问题。

函数式编程的另一个特点是可以把函数作为参数传入另一个函数。但 Python 的函数式编程中允许使用变量，因此，Python 不是纯函数式编程语言，它对函数式编程提供部分支持。

函数式编程编写的代码将数据、操作、返回值等都放在一起，使代码更加简洁。

5.6.1 lambda 匿名函数

匿名函数是指不需要显示指定函数名的函数，调用一次或几次后就不再需要的函数。Python 中用 lambda 关键字通过表达式的形式来定义匿名函数。lambda 表达式的首要用途是指定短小的回调函数。

lambda 表达式用来声明没有函数名称、临时使用的匿名函数，尤其适用于将一个函数作为另一个函数的参数的情形。