第5章　函　　数

学习导读

在第4章中，学习了使用顺序结构、选择结构和循环结构来编写代码，以解决一些简单的问题。然而，随着问题复杂度的增加，代码会变得越来越长、难以阅读和维护。你是否遇到过这样的困扰：一段相同的代码在程序中多次出现，一旦需要修改，就必须找出所有地方逐一更改？或者，一个程序文件成百上千行，是否连自己都很难厘清逻辑？

本章将学习Python程序设计中一个至关重要，乃至影响整个编程思维的概念——函数。函数是将一段具有特定功能的代码封装成一个独立的、可重复使用的工具。它如同建筑中的预制模块、音乐中的重复乐章、文章中的经典段落，是构建清晰、高效、可靠程序的基石。掌握函数，意味着编程能力将从“写简单脚本”跃升到“设计结构化程序”，是迈向设计结构化程序的关键一步。

内容导学

（1）函数的作用。

（2）函数的基本使用。

（3）函数的参数。

（4）多函数程序。

（5）递归函数。

教学目标

1. 知识目标

（1）理解函数在程序设计中的作用。

（2）掌握函数的具体使用方法。

（3）掌握函数的参数作用和使用方法。

（4）掌握多函数程序的设计方法。

2. 能力目标

（1）能够将复杂问题分解为若干功能模块，并运用函数进行实现和组合。

（2）能够运用递归思想，将问题转化为自相似的子问题，以简洁的代码解决特定类型的复杂问题。

育人目标

像一名工匠那样思考。当精心设计一个函数，思考它的名字是否清晰、功能是否单一、接口是否易用时，你就是在打磨自己的“作品”。这种对代码优美性、健壮性和可维护性的追求，正是“工匠精神”在数字世界的体现。请记住，你写下的每行代码，都是专业素养的签名。

建立系统的大局观。程序如同一个社会系统，函数则是其中的个体或组织。你需要学会让每个函数“各司其职”，通过清晰的规则（参数与返回值）进行“沟通协作”，最终完成复杂的整体目标。这种模块化设计的训练，能帮助你理解在任何一个集体项目中，个人的卓越必须融入并服务于整体的架构，这是一种至关重要的系统工程思维。

理解合作与共享的价值。在编程世界里，没有人是一座孤岛。当使用 import 引入一个强大的外部库时，你正在享受全球开发者社区的智慧结晶。函数让你能够封装自己的成果，供他人和未来的你重复使用。这深刻揭示了一个道理：现代社会的进步，源于有效的分工协作与知识的开放共享。学会成为这个良性生态的贡献者而非单纯的索取者，将是伴随你职业生涯的宝贵财富。

5.1 函数概述

Python 语言中的函数大致可以分为以下 4 类，如表 5.1 所示。

表 5.1 Python 语言中的函数类别及其特点

类别	内置函数	标准库函数	第三方库函数	用户定义函数
来源	Python 解释器	Python 标准库	PyPI 社区	开发者自己
导入	无须导入	import 语句	pip 安装后通过 import 语句	直接定义
数量	约 70 个	数百个模块	数十万个包	无限制
文档	官方标准	官方标准	社区提供	自己维护
更新	Python 版本更新	Python 版本更新	独立更新	随时修改

（1）内置函数，是 Python 语言核心组成部分，由 Python 解释器直接提供，无须导入语句即可全局使用。这类函数构成了 Python 编程的基础工具集。

（2）标准库函数，包含在 Python 标准发行版中，通过导入特定模块后使用。它们是官方维护的功能集合，随 Python 一起安装。

（3）第三方库函数，由 Python 包托管（PyPI）社区开发者创建和维护，需要通过包管理工具额外安装。这类函数极大地扩展了 Python 的应用领域。

（4）用户定义函数，是程序员根据具体需求自行创建的函数，是代码封装和复用的基本单位。

本章所学函数指用户定义函数，在开始学习如何定义和调用函数之前，首先深入地理解：函数到底有什么作用？为什么它被认为是程序设计的核心？

想象一下，正在编写一个程序来计算一个班级学生的平均分、最高分和最低分。如果没有函数，代码可能会像一长串连续不断的指令，所有的计算逻辑都混杂在一起。当需要计算另一个班级的成绩时，只能把代码再复制粘贴一遍，稍作修改。一旦计算逻辑需要调整（例如，排除缺考学生），就必须在多个地方进行相同的修改，这不仅效率低下，而且极易出错。

函数正是为了解决这类问题而生的。在 Python 中，函数的作用可以概括为以下 3 个核心方面，它们共同将代码从“一锅乱炖”升级为“模块化大餐”。

（1）代码复用。这是函数最直接、最显著的作用。将一段需要重复使用的语句块定义

为函数后，就可以在程序的任何地方通过简单的函数名来“调用”它，无须重写。

（2）模块化设计。“分而治之”是解决复杂问题的黄金法则。一个大型程序可以被分解为若干逻辑清晰、功能独立的模块（函数），每个函数只负责一项具体的任务。这使得程序结构像一本书的目录一样清晰，易于理解和设计。

（3）提高可读性与可维护性。一个具有描述性名称的函数（如 calculate_average）本身就是一个绝佳的注释，当你阅读函数时，立刻能明白函数内所封装代码的意图，而无须关心其底层复杂的实现细节。

当程序出现漏洞或需求变更时，只需修改相关的函数即可，所有调用该函数的地方都会自动生效，极大地降低了维护成本。

5.2 函数的基本使用

函数就是将一段具有独立功能的语句块整合到一个整体并命名，在需要的位置调用这个名称即可完成对应的需求。因此，函数是组织好的、可重复使用的、用来实现单一或相关联功能的代码段。在 Python 中，函数能够提高应用的模块性和代码的重复利用率。函数的使用步骤：先定义函数，再调用函数。

5.2.1 定义函数

定义函数的语法格式如下：

```
def 函数名(参数 1,参数 2,…):
    函数体
```

Python 使用关键字 def 定义一个函数，所定义的函数包括函数头和函数体，如图 5.1 所示，其中，函数头包括函数名和形参。

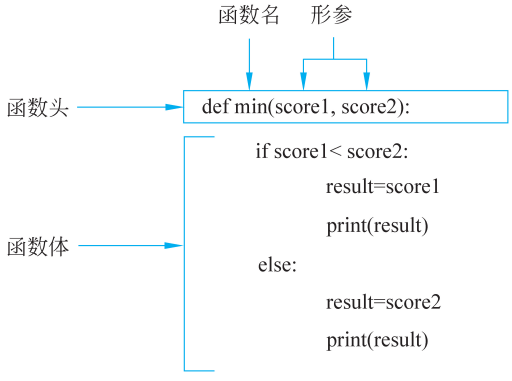

图 5.1 所定义函数的组成

定义一个函数时，应注意以下 3 点。

（1）函数必须使用关键字 def 定义，函数名须自定义，但必须符合 Python 标识符命名规则。

（2）形参一般是变量，是函数形式上的参数，调用函数时传入实际数据。形参的个数可

以为 0 个，也可以为多个，个数不受限制，根据实际情况而定，形参与形参之间用逗号(,)分隔。即使函数不接收任何参数，也必须保留一对圆括号和冒号。有关参数的内容后续将进一步介绍。

(3) Python 语言严格遵循缩进机制，Python 语言中冒号与缩进是同步出现的，有冒号必定有缩进，所有的函数体都必须缩进。

5.2.2 调用函数

调用函数实际上就是执行函数。定义好函数后，为了使用函数，就必须调用这个函数。如果把定义函数比作创造工具，那么调用函数就是使用工具的过程。调用函数的基本语法格式如下：

```
函数名(实参 1,实参 2,…)
```

程序调用一个函数需要执行以下 4 个步骤。

(1) 当执行到调用程序处时，程序暂停执行。

(2) 在调用时将实参“传递”给定义函数处函数的形参。

(3) 执行函数体语句。

(4) 函数体语句执行完毕后，程序回到调用前的暂停处继续向后执行程序。

对图 5.1 的实例进一步进行分析，该段代码是在定义函数，不会直接执行，只有在函数被调用时才会执行。图 5.2 中的程序，最先执行的是第 8 行，当执行到第 8 行时，由于调用了函数，当前程序暂停执行，程序将实参“传递”至形参，形参被赋值为实参值，相当于执行了如下语句：

```
score1 = 15 ; score2 = 25
```

然后，将实参代入并执行函数体内的语句，函数体执行完毕后，重新回到第 8 行，继续向后执行余下的语句，如图 5.2 所示。

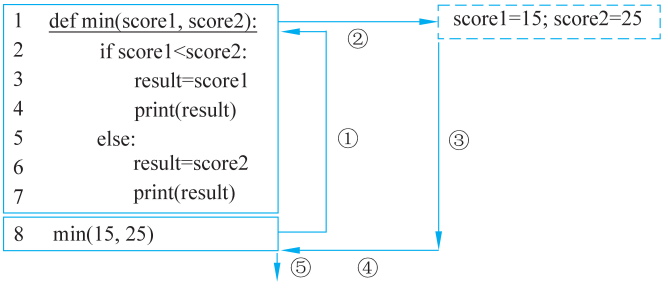

图 5.2 调用函数时程序执行流程

值得注意的是，函数调用的形式可以是函数表达式、函数参数、赋值语句，要根据具体情况确定函数调用方式。实参是调用时必须传递给函数的实际数值，实参可以是各种数据类型、表达式，多个实参之间用逗号(,)分隔。一般情况下，实参个数与形参个数需要相等，形成一一对应的原则；但也可以不相等，后续章节会做具体介绍。

【例 5.1】 计算任意半径的圆面积，要求半径由用户输入。

```
import  math
#定义计算圆面积的函数
def  circle_area(radius)
    s = math.pi  *  radius ** 2
    return s
def  main():
     #获取用户输入
    r = float(input('请输入圆的半径:'))
    #调用 circle_area()函数计算圆面积
    area = circle_area(r)
    #输出结果
    print(f'半径为{r}的圆面积:{area:.2f}')
main()
```

本例定义了两个函数：第一个函数 circle_area()带一个形参，并返回计算结果；第二个函数 main()既没有形参，也没有返回值，该函数的作用主要是将输入输出和函数调用等代码封装在一起，从而使得整个程序段看起来更为清晰、简洁。

从本例可以看出，通过函数的定义和调用，可以将所需解决的问题变得十分便捷，减少循环嵌套的复杂逻辑，这就是模块化程序设计的主旨。

5.2.3 函数参数的作用

参数是函数实现功能灵活性和代码复用性的核心机制。通过合理使用参数，可以将函数从固定、僵化的语句块转变为通用、可配置的功能单元。函数通过参数摆脱对具体数据的依赖，转变为可处理一类问题的通用解决方案。这种从“具体”到“通用”的转变，是参数最根本的价值体现。本节将深入探讨参数如何赋予函数强大的适应能力和灵活性。

接下来，请思考：假如需要通过一个函数完成两个数 1 和 2 的加法运算，如何书写程序？根据前面所学，很快就能写出如下程序：

```
#定义函数
def  add_result():
    result = 1+2
    print(result)
#调用函数
add_result()
```

进一步思考，上述 add_result()函数只能完成数字 1 和 2 的加法运算，如果想要这个函数变得更灵活，可以计算任何用户指定的两个数字的和，应该如何书写程序？

实际上，这个问题的解决，也就是用户要在调用函数时指定具体数值，那么，在定义函数时就需要接收用户指定的数字。函数调用时指定的数值和定义函数时接收的数值即是函数的参数，分别称为实参和形参。程序如下：

```
#定义函数时同时定义用于接收用户数据的参数 num1 和 num2,其为形参
def  add_result(num1,num2):
    result = num1 + num2
```

```
4       print(result)
5   #调用函数时传入真实的数据 100 和 200,其为实参
6   add_result(100,200)
```

这个简单的例子揭示了参数的核心价值：将数据处理逻辑与具体数据分离。函数 add_result()不再关心具体处理什么，只关心“如何处理”，这种分离使得函数具备了处理一类问题的能力，而不仅仅是解决单个具体问题。

参数是函数从“死”代码变为“活”工具的关键机制。通过参数，函数获得了数据通用性、可复用性、可扩展性、可定制性等方面的能力。理解参数的作用不仅是掌握 Python 编程技术，更是培养抽象思维和模块化设计能力的重要途径。在后续章节中，将看到参数机制如何在更复杂的编程范式中发挥核心作用。

5.2.4 函数的返回值

函数的返回值是函数与调用环境之间进行数据交换的关键机制，它使得函数不仅能够执行特定操作，还能将处理结果反馈给调用者，实现数据的流动和功能的组合。在 Python 中，理解返回值的机制对于编写高效、模块化的代码至关重要。

返回值是函数执行完成后返回给调用者的数据结果。在 Python 中，使用 return 语句指定函数的返回值。

```
1   #定义函数,计算 num1 与 num2 的和
2   def  add_result1(num1,num2):
3       result = num1 + num2
4       return result                                    #返回计算结果
5   #调用函数接收到返回值后,将返回值代入计算
6   s = add_result1(10,20) * 30
7   print(s)
```

这是一个带返回值函数的例子，函数 add_result1()返回计算结果 result，它可以像调用一个数字一样使用。实际上，函数不是一定都有返回值，此时的函数被称为无返回值函数。例如，

```
1   #定义函数,计算 num1 与 num2 的和
2   def  add_result2(num1,num2):
3       result = num1 + num2
4       print(result)                                    #输出计算结果
5   #调用函数接收到返回值后,将返回值代入计算
6   s = add_result2(10,20)
7   print(s)                                             #输出 None
```

上面的程序执行后的结果如下：

```
300
None
```

第 6 行调用 add_result2()函数，add_result2()函数不返回任何值，所以，此处被当作一

个调用语句。

实际上，无论是否使用 return，所有 Python 的函数都将返回一个值。如果某个函数没有返回值，默认情况下，它返回一个特殊值 None。因此，无返回值函数也称 None 函数。None 函数可以赋值给一个变量来表明这个变量不指向任何对象，正如上面的例子。在使用 return 语句来获得函数返回值时，要注意以下 4 点。

(1) return 后面可以跟数值、变量或表达式，当 return 后面只有一个值时，它可以是任意数据类型。

(2) return 后面也可以跟多个值(变量或表达式)，当有多个值时，每个值之间用逗号分隔，返回结果以元组的方式返回(元组内包含所有返回值)。

(3) 若没有 return 语句，默认在函数的最后一行返回 None。

(4) return 具有返回函数值的作用，同时，也具有退出当前函数的特点，函数体内 return 语句下方的代码不会执行。因此，函数中可以出现多个 return 语句，但执行完任意一个 return 语句后就会直接退出当前函数，返回至函数调用处。

5.2.5 函数的说明文档

在 Python 中，函数的说明文档是一个位于函数定义下方的多行字符串，用于描述函数的功能、参数、返回值等信息。Python 官方建议使用三重双引号来定义说明文档。定义函数说明文档的语法如下：

```
def  函数名(参数):
    """说明文档的位置"""
    代码
    …
```

在定义好函数的说明文档后，后续需要访问该函数的说明文档时，可以使用 help()函数查看函数的说明文档。

```
help(函数名)
```

【例 5.2】 定义并访问函数的说明文档。

```
def  sum_num(a,b):
    """求和函数"""
    return a + b
help(sum_num)
```

程序运行结果如图 5.3 所示。

```
Help on function sum_num in module __main__:

sum_num(a, b)
    求和函数
```

图 5.3 程序运行结果

函数的说明文档是函数定义中不可或缺的部分，它不仅是对函数功能的文字说明，更是

代码可读性、可维护性的重要保障。良好的说明文档能够让其他开发者(以及未来的自己)快速理解函数的作用、用法和注意事项。许多工具(如 Sphinx)可以从说明文档自动生成 API 文档,IDE 和编辑器也可以利用说明文档提供智能提示。

5.3 函数的参数

Python 中所有的数据都是对象,所以对象的变量通常都是指向对象的引用。当调用一个带参数的函数时,每个实参的引用值就被传递给形参。函数的作用就在于它处理参数的能力,Python 语言在参数设计上极具特色,提供了多种参数传递机制,使得函数既能保持简洁明了的调用方式,又能应对复杂多变的应用场景。本节将系统阐述 Python 中各类参数的定义、特点和使用方法。

5.3.1 位置参数

位置参数是函数定义中最基本、最常见的参数类型。这类参数的特点是调用时必须按照函数定义时参数的顺序依次传递对应的值,参数之间通过位置一一对应,故而得名“位置参数”。

位置参数体现了函数调用的一种直观对应关系:调用者提供的第一个实参与函数定义的第一个形参匹配,第二个实参与第二个形参匹配,以此类推。这种一一对应的关系使得代码阅读和理解相对直观,但也要求调用者必须清楚每个位置参数的具体含义。

在实际编程中,位置参数通常用于那些参数数量固定、含义明确且顺序自然的场景。例如,计算两点距离的函数 distance(x1, y1, x2, y2),其参数顺序就是逻辑上的自然顺序:第一个点的 x 坐标、第一个点的 y 坐标、第二个点的 x 坐标、第二个点的 y 坐标。

【例 5.3】 计算二维平面上任意两点之间的欧几里得距离。

```
import math

def  distance(x1, y1, x2, y2):
    """
    计算二维平面上任意两点之间的欧几里得距离
    参数:
    x1 : float      第一个点的 x 坐标
    y1 : float      第一个点的 y 坐标
    x2 : float      第二个点的 x 坐标
    y2 : float      第二个点的 y 坐标
    返回:
      float 两点之间的直线距离
      计算公式:
      distance =√((x2 - x1)²+(y2 - y1)²)
      """
      dx = x2 - x1                                  #计算 x 轴方向的距离差
      dy = y2 - y1                                  #计算 y 轴方向的距离差
      dist = math.sqrt(dx**2 + dy**2)               #应用勾股定理计算距离
      return dist

#计算点(0,0)到点(3,4)的距离
```

```
dist1 = distance(0, 0, 3, 4)
print(f"点(0,0)到点(3,4)的距离: {dist1}")          #输出: 5.0

#计算点(1,2)到点(4,6)的距离
dist2 = distance(1, 2, 4, 6)
print(f"点(1,2)到点(4,6)的距离: {dist2:.2f}")      #输出: 5.00
```

【例 5.4】 计算身体质量指数。

```
def  calculate_bmi(weight, height):
    """
    计算身体质量指数
    参数说明:
    weight - 体重(千克),必须按第一个位置传递
    height - 身高(米),必须按第二个位置传递
    """
    return weight / (height ** 2)

#正确调用:70 对应参数 weight,1.75 对应参数 height
bmi = calculate_bmi(70, 1.75)

#错误调用:参数顺序颠倒会导致逻辑错误
#错误:将身高当作体重,体重当作身高
bmi_wrong = calculate_bmi(1.75, 70)
```

使用位置参数时需特别注意参数顺序的准确性,一旦顺序错误,程序可能不会报语法错误,但会产生逻辑错误,这类错误往往难以察觉。因此,当函数参数较多或含义不够直观时,建议使用时结合关键字参数,以提高代码的可读性和可靠性。

5.3.2 关键字参数

关键字参数是通过参数名来指定传递值的参数类型。与位置参数不同,关键字参数不依赖参数的顺序,而是通过明确的参数名称来建立实参与形参之间的对应关系。这种机制大大提高了代码的可读性和可维护性。

关键字参数的核心优势在于其自文档化特性。当读者看到 draw_circle(radius=10, color='red', filled=True)这样的调用时,即使不了解函数的具体定义,也能从参数名直观理解每个参数的含义。相比之下,相同的位置参数调用 draw_circle(10, 'red', True)就显得晦涩难懂。

在函数定义中,关键字参数的使用方式与位置参数相同,但在调用时可以指定参数名。Python 允许在函数调用时混合使用位置参数和关键字参数,但有一个重要规则:位置参数必须出现在关键字参数之前,但关键字参数之间不存在先后关系。这种设计既保证了灵活性,又避免了潜在的歧义。

【例 5.5】 创建学生记录。

```
def  create_student_record(name, age, major, grade='一年级', scholarship=
False):
```

```
    """
    创建学生记录
    前三个参数建议使用位置参数,后两个可选参数建议使用关键字参数
    """
    record = {
        'name': name,
        'age': age,
        'major': major,
        'grade': grade,
        'scholarship': scholarship
    }

    return record

#使用关键字参数调用,顺序无关紧要
student1 = create_student_record(name='张三', age=20, major='计算机科学',
                                 scholarship=True, grade='二年级')
#混合使用位置参数和关键字参数
student2 = create_student_record('李四', 21, '软件工程', scholarship=True)
#错误:关键字参数出现在位置参数之前
#student3 = create_student_record(name='王五', 22, major='人工智能')
                                                                #语法错误
```

关键字参数特别适用于以下场景：函数参数较多(超过 3 个)、参数含义需要明确说明、部分参数有默认值,以及需要跳过某些参数只提供特定参数的情况。在实际开发中,良好的习惯是对那些含义不直观或需要特别说明的参数使用关键字参数调用。

5.3.3 默认参数

默认参数是在函数定义时为其指定了默认值的参数。这类参数的最大特点是调用时可以不提供对应的实参,此时函数将使用定义时指定的默认值。默认参数机制极大地增强了函数的灵活性和易用性。

默认参数的设计哲学是“合理的默认值”。在许多应用场景中,函数的某些参数在大多数情况下都取相同的值,只有少数特殊情况需要改变。如果将这些参数设计为必需参数,即使这个值总是相同的,每次调用也都需要显式提供值。默认参数解决了这个问题：它为这些常用值提供预设,同时允许在需要时被覆盖。

在函数定义中,默认参数必须放在非默认参数(即必需参数)之后,旨在保证函数调用的清晰性。如果允许默认参数出现在非默认参数之前,那么调用时的参数绑定就会产生歧义。

【例 5.6】 格式化文档。

```
def format_document(title, content, font_size=12, font_family='宋体',
                    line_spacing=1.5, margin=2.0):
    """
    格式化文档
    title 和 content 是必需参数,其余都有合理的默认值
    """
```