第5章

函　　数

函数也可称为方法，是具有特定功能的程序代码块。前几章中已经多次使用了Python自带的标准函数，如int()、type()、range()、print()等。通过使用函数，开发者可以快速调用某些功能，从而不需要从零开始构建程序，大大提升程序的开发效率。函数是编程语言学习过程中非常重要的一部分内容。

本章将对函数的定义、调用、函数参数类型、参数传递、变量作用域和递归函数等内容进行讲解，还将对一些重要的特殊函数，如lambda函数、map函数、filter函数等进行介绍。

5.1　函数的定义与调用


视频讲解

5.1.1　函数概念

函数，是具有特定功能的代码块，其目的是代码可以重复使用。比如第3章示例3.21的菱形打印小程序，用户只要输入菱形的行数，程序就可以打印出对应的菱形。

如果项目组成员都需要调用这个程序，那么让每个成员都编写一个打印菱形的程序，显然是较为浪费人力和物力资源的。最好的方式就是将其编写为一个函数，共享给项目组成员使用。

【示例5.1】　将示例3.21改为函数形式。

```
def print_lx(rows):                                    #将菱形打印程序转换为函数形式
    rows = int(rows)
    half = rows // 2                                   #整除,分为上下两部分
    if rows % 2 == 0:                                  #进行奇偶判断
        up = half
    else:
        up = half +1
    for i in range(1,up+1):
        print('  '*(up-i),'* '*(2*i-1))
    for i in range(half, 0, -1):                       #反向遍历
        print('  '*(up-i), '* '*(2*i-1))#打印下半部分的结果

rows = input("请输入菱形的行数:")
print_lx(rows)
```

程序运行结果：

```
请输入菱形的行数：7
    *
   * * *
  * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

示例 5.1 把菱形行数作为函数的参数进行传递，需要打印几次菱形，就调用几次函数。

从函数使用者角度看，函数就如图 5.1 所示的一个黑盒子，用户传入零个或多个参数，该黑盒子经过一定的处理后，就会返回零个或多个值。使用者只要知道传递什么参数，能得到什么结果即可，而不需要了解函数的内部处理过程。

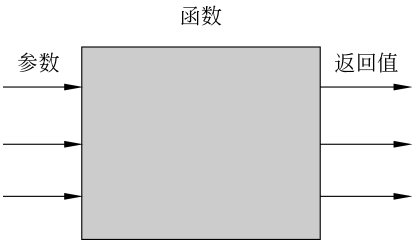

图 5.1　使用者角度的函数调用流程图

从函数设计者(实现函数的编程人员)的角度看，在设计函数时，一般要思考以下几个问题：

(1) 函数中哪些内容是动态变化的，即哪些内容应该被定义为函数参数，如程序示例 5.1 的菱形行数；

(2) 函数要实现什么功能，函数最终给使用者返回什么结果；

(3) 函数如何实现这些功能，即函数体。

5.1.2　函数定义及调用

Python 定义函数的语法结构如下。

```
def 函数名([形参列表]):
    """函数说明文档"""
    函数体
    [return [返回值]]
```

语法释义如下。

(1) def：用来定义函数的关键词。

(2) 函数名：必须是一个合法的标识符，函数名应见名知义，建议由小写字母组成，多个单词间用下画线隔开。

(3) 形参列表：函数可包含零个或多个参数，形参不用指定数据类型，多个参数间用逗号隔开，调用函数时再对形参传递参数值。

(4) 函数说明文档：一般用于说明函数的功能及调用方式，函数说明文档可以用三双引号或三单引号对括起，可选。

(5) 函数体：由一条或多条语句组成的代码块，用于实现函数的功能。

(6) return 语句：返回函数结果，可选。函数可以没有返回值，也可以有一个或多个

返回值，多个返回值会以元组的形式返回。

注意：函数可以没有参数，但函数名后的圆括号不能少；函数体、函数说明文档及 return 语句都要保持缩进。

【示例 5.2】　定义一个无返回值的函数。

```
def greet_user(name):
    """向用户问候的函数"""
    name = name.title()
    print(f"Hello, {name}!")          #打印欢迎语句

name = "alice"
greet_user(name)                      #函数调用
greet_user("bob")                     #函数调用
```

程序运行结果：

```
Hello, Alice!
Hello, Bob!
```


视频讲解

【示例 5.3】　定义一个有返回值的函数。

```
def is_leap_year(year):                                    #判断一个年份是否为闰年
    """判断 year 是否为闰年的函数"""
    if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
        return True
    else:
        return False

year = int(input("Input year:"))
is_or_not = "is" if is_leap_year(year) else "is not"   #使用简化 if 语句返回结果
print(f"{year} {is_or_not} a leap year.")
```

程序运行结果：

```
Input year:2000
2000 is a leap year.
```

通过上述两个程序示例可以看出，函数调用的方式是：函数名(实参列表)。一般来讲，实参列表中的参数数量要与函数定义时的形参数量相同，参数类型也要一致，否则将会抛出 TypeError 错误。

由于 Python 是解释性语言，因此函数的调用一定要放在函数定义之后，否则解释器将找不到函数，会抛出 NameError 错误。当定义了多个同名函数时，解释器调用的是最近一次定义的函数。

根据函数是否有返回值，函数调用有以下两种方式。

(1) 带有返回值的函数调用，通常将函数的调用结果作为一个值处理。

(2) 没有返回值的函数调用，通常将函数调用作为一条语句来处理。

注意：不论 return 语句出现在函数体的什么位置，一旦得到执行，将直接结束函数的运行。

思考与练习

5.1 判断题：定义函数时，可以没有参数，但必须要有一对圆括号。

5.2 判断题：在 Python 中，函数的使用包括函数定义和函数调用，并且函数调用一定要放在函数定义之后。

5.3 什么是函数？函数的作用是什么？

5.4 Python 中用来定义函数的关键字是什么？函数必须包含 return 语句吗？

5.5 阅读下面的代码，写出其执行结果。

```
def func(x):
    return x + 2
    return x + 4

print(func(3))
```


视频讲解

5.2 参数类型与参数传递

5.2.1 形参和实参

在介绍函数参数传递前，首先要明确形式参数和实际参数两个概念。形式参数通常简称为形参，是指在函数定义时，写在圆括号内的变量。形参不代表任何具体的值，只是作为一种占位符参与函数体的业务逻辑。实际参数通常简称为实参，是指在函数调用时，实际传递给形参的值。

程序示例 5.3 的第 1 行代码，函数定义 is_leap_year(year)中的 year 就是形参，没有具体的值，只是作为一种占位符参与函数体的业务逻辑。而程序示例 5.3 的第 9 行代码，则是对 is_leap_year()的函数调用，其中的 year 就是实参，由用户输入的具体值确定。一般来讲，形参和实参的名称(变量名)可以相同也可以不同，但建议设置为相同的名称。

在 Python 程序中，定义函数时不需要指定形参的类型，形参类型由函数调用时传递的实参类型决定。实参与形参通常数量一致，当形参为可变长度参数时，实参可以有多个，此时实参和形参的数量可以不相同。

根据函数参数传递的特点和形式，可大致将函数参数分为位置参数、关键字参数、默认值参数、可变长度参数、序列解包参数等。

5.2.2 位置参数

位置参数是指函数调用时，根据函数定义的形参位置，依次将实参的值赋值给形参。位置参数传递要求实参和形参的数量必须一致，顺序一一对应。位置参数是最常见、最简单的函数参数传递方式。

【示例 5.4】 函数的位置参数传递。

```
def describe_pet(pet_name, pet_type):
    """描述宠物信息的函数"""
```

```
    print(f"I have a {pet_type}.")
    print(f"It's {pet_name.title()}.")

pet_name = "Tom"
pet_type = "cat"
describe_pet(pet_name, pet_type)     #按顺序依次将实参的值传给形参
describe_pet("Jerry", "mouse")       #按顺序依次将实参的值传给形参
```

程序运行结果：

```
I have a cat.
It's Tom.
I have a mouse.
It's Jerry.
```

5.2.3　关键字参数

关键字参数是函数调用时，以“形参名＝实参值”的形式指定形参的实参值。此时形参出现的顺序可以和函数定义时不一致。关键字参数灵活、方便，调用者不用关注形参定义时的顺序和位置。

【示例 5.5】　函数的关键字参数传递。

```
def describe_pet(pet_name, pet_type):                    #该函数和示例 5.4 一致
    """描述宠物信息的函数"""
    print(f"I have a {pet_type}.")
    print(f"It's {pet_name.title()}.")

pet_name = "bath"
pet_type = "dog"
describe_pet(pet_type=pet_type, pet_name=pet_name) #指定形参对应的实参值
describe_pet(pet_type="mouse", pet_name="jerry")   #指定形参对应的实参值
```

程序运行结果：

```
I have a dog.
It's Bath.
I have a mouse.
It's Jerry.
```

5.2.4　默认值参数

默认值参数是函数定义时，在形参列表中，直接为形参指定默认值。函数调用时，对于有默认值的参数，可传值也可不传值。未传值时，将采用默认值；传值时，将用新的值替换默认值。默认值参数可方便函数调用，减少参数传递数量。

注意：定义带有默认值参数的函数时，默认值参数要出现在函数形参列表的右端，任何一个默认值参数右边都不能再出现非默认值参数。

【示例 5.6】 定义一个带有默认值参数的函数。

```
def describe_pet(pet_name, pet_type="dog"):     #定义函数,并使用默认值参数
    """描述宠物信息的函数"""
    print(f"I have a {pet_type}.")
    print(f"It's {pet_name.title()}.")

pet_name = "Bath"
describe_pet(pet_name=pet_name)                          #默认值参数不传值
describe_pet(pet_name="Jerry", pet_type="mouse")    #默认值参数传值
```

程序运行结果:

```
I have a dog.
It's Bath.
I have a mouse.
It's Jerry.
```

对于函数的参数默认值,可使用“函数名.__defaults__”查看函数所有参数的默认值,其返回值为一个元组,其中的元素依次为每个参数的默认值。

【示例 5.7】 查看参数默认值。

```
def describe_pet(pet_name="bath", pet_type="dog"):  #定义函数,并使用默认值参数
    """描述宠物信息的函数"""
    print(f"I have a {pet_type}.")
    print(f"It's {pet_name.title()}.")

>>>describe_pet.__defaults__                          #运行代码需在命令行中运行
```

程序运行结果:

```
('Bath', 'dog')
```

5.2.5 可变长度参数


视频讲解

可变长度参数是函数定义时,无法确定具体参数的数量,此时可将函数的形参设为可变长度参数,然后根据调用者传递的实际参数数量来确定参数的长度。

可变长度参数有两种形式:*形参名和**形参名。

*参数名:表示该参数是一个元组类型,可接受多个实参,并将传递的实参依次存放到元组中,其主要针对以位置传值的实参。

例如披萨店会根据用户的口味来提供不同的配料,那么定义函数时,便可将配料设为可变长度参数。

【示例 5.8】 定义一个制作披萨的函数。

```
def make_pizza(*toppings):          #定义函数,并使用可变长度参数
    """制作披萨的函数"""
    print("Your pizza has following toppings:")
    for topping in toppings:
```

```
    print("-->", topping)

make_pizza("pepper")
make_pizza("rushroom", "pepper", "raddish")
```

程序运行结果：

```
Your pizza has following toppings:
--> pepper
Your pizza has following toppings:
--> rushroom
--> pepper
--> raddish
```

在有多种类型参数的情况下，Python 解释器会先匹配位置实参，再匹配任意数量实参，所以要将可变长度参数放在函数定义的位置参数右边。

【示例 5.9】　定义一个制作披萨的函数，并使用多种类型参数。

```
def make_pizza(size, *toppings):    #定义函数，并使用可变长度参数
    """制作披萨的函数"""
    print(f"Your pizza is {size} inches.")
    print("It has following toppings:")
    for topping in toppings:
        print("-->", topping)

make_pizza(12, "pepper")            #先传 12 给 size 形参，其余给 toppings
make_pizza(12, "rushroom", "pepper", "raddish")
```

程序运行结果：

```
Your pizza is 12 inches.
It has following toppings:
--> pepper
Your pizza is 8 inches.
It has following toppings:
--> rushroom
--> pepper
--> raddish
```

**形参名：表示该形参是一个字典类型，可接受多个实参，主要针对以关键字传值的实参，并将传递的键值对保存到字典中。

【示例 5.10】　定义一个水果描述函数。


视频讲解

```
def fruit_info(name, grade, **other_info):    #定义函数，并使用可变长度参数
    """定义水果描述函数"""
    profile = {}
    profile["name"] = name
    profile["grade"] = grade
    for key, value in other_info.items():
```

```
        profile[key] = value
    return profile

fruit = fruit_info("apple", "A++",#通过关键字参数给 other_info 传值
location="shandong", color="red")#other_info 为字典
print(fruit)
```

程序运行结果:

```
{'name': 'apple', 'grade': 'A++', 'location': 'shandong', 'color': 'red'}
```


视频讲解

5.2.6 多种类型参数混用*

Python 支持定义函数时几种不同形式的参数混用,但初学者要谨慎使用,因为使用不当将导致代码混乱,且降低可读性,使得程序查错困难。

在多种类型参数混用时,要注意以下几点原则。

(1) 实参传值时,既可通过位置传值,也可通过关键字传值,但 * 可变参数后面的参数只能通过关键字传值。

(2) 函数定义时,不能同时包含多个相同类型的可变参数,即多个 * 参数或多个**参数,但可同时包含一个 * 参数和一个**参数,且 * 参数要放在**参数前面。

(3) 当既有可变参数又有普通参数时,会先给普通参数赋值,最后将多余的值存放在可变参数中,可变参数可以不进行赋值,此时可变参数为空。

(4) 带有默认值的参数后面不能包含没有默认值的参数,但可以包含可变参数。

【示例 5.11】 定义一个学生信息函数,并使用多种参数混用。

```
def student_info(name, age, *contacts, gender="男", **others):
    ""定义学生信息函数"""
    print("="*10, "学生基本信息", "="*10)
    print("姓名: ", name)
    print("年龄: ", age)
    print("性别: ", gender)
    print("联系方式: ", end="")
    if len(contacts) == 0:              #判断联系方式是否为空
        print("无")
    else:
        for contact in contacts:        #将所有联系方式放在一行打印
            print(contact, end="\t")
        print()
    for key, value in others.items():
        print(key, ":", value)
    print("="*34)                       #结束分割线

student_info("张三", 19, "手机-138****2222", "QQ-876***567", 身高=175,
籍贯="江西")
student_info("张丽", 18, "QQ-358***121", gender="女",职务="班长",学号="006")
student_info("张小东", 18, 专业="软件工程",学号="008")
```

程序运行结果：

```
========== 学生基本信息 ==========
姓名：张三
年龄：19
性别：男
联系方式：手机-138****2222    QQ-876***567
身高：175
籍贯：江西
==================================
========== 学生基本信息 ==========
姓名：张丽
年龄：18
性别：女
联系方式：QQ-358***121
职务：班长
学号：006
========== 学生基本信息 ==========
姓名：张小东
年龄：18
性别：男
联系方式：无
专业：软件工程
学号：008
==================================
```

【示例 5.12】　对示例 5.11 进行修改。

```
1    def student_info(name, age, gender="男", *contacts, **others):
2        #调换 gender 与 *contacts 位置,后续代码同示例 5.11 一致,不再列出
```

程序运行结果：

```
========== 学生基本信息 ==========
姓名：张三
年龄：19
性别：手机-138****2222
联系方式：QQ-876***567
身高：175
籍贯：江西
==================================
Traceback (most recent call last):
  File "D:\Python\Basic_Python \chap05\ch05_12.py", line 25, in <module>
    student_info("张丽", 18, "QQ-358* * *121", gender="女",职务="班长",学号=
    "006")
TypeError: student_info() got multiple values for argument 'gender'
```

当 gender="男"放在 *contacts 之前时，Python 解释器在解释执行第 19 行代码时，会先按照位置参数进行传值，gender 首先得到值"QQ-358***121"，后又继续赋值 gender="女"，因此程序报错。

【示例 5.13】 继续对示例 5.11 进行修改。

```
def student_info(name, *contacts, age, gender="男", **others):
    #调换 name 之后的参数顺序,后续代码同示例 5.11 一致,不再列出
```

程序运行结果:

```
Traceback (most recent call last):
  File "D:\Python\Basic_Python \chap05\ch05_13.py", line 24, in <module>
    student_info("张三", 19, "手机-138****2222", "QQ-876***567", 身高=175, 籍贯="江西")
TypeError: student_info() missing 1 required keyword-only argument: 'age'
```

此时第 18 行代码中函数参数传递时,contacts 得到的值为(19,"手机-138****2222","QQ-876***567")。因此,age 没有得到赋值,程序报错。实参传值时,可变参数后面的参数只能通过关键字传值。

【示例 5.14】 继续对示例 5.11 进行修改。

```
def student_info(name, age, *contacts, **others, gender="男"):     #语法错误
    #将 gender 置于**others 后,后续代码同示例 5.11 一致,不再列出
```

程序运行结果:

```
  File "D:\Python\Basic_Python \chap05\ch05_14.py", line 8
    def student_info(name, age, *contacts, **others, gender="男"):
                                           ^
SyntaxError: arguments cannot follow var-keyword argument
```

带有**的可变长度参数必须放在函数定义的最后,因此这里程序报错。


视频讲解

5.2.7 参数传递的序列解包*

参数传递的序列解包:实参为序列对象,传值时将序列中的元素依次取出,然后按照一定规则赋值给相应变量。其主要有两种形式:*序列对象、**字典对象。

当传递的实参为*序列对象时,将会取出序列中的每个元素,然后按照位置顺序依次赋值给每一个形参。

【示例 5.15】 使用*序列实参。

```
def demo(a, b, c):
    print(f"a = {a}, b = {b}, c = {c}")

seq = [1, 2, 3]
demo(*seq)          #实参为*序列对象
```

程序运行结果:

```
a = 1, b = 2, c = 3
```