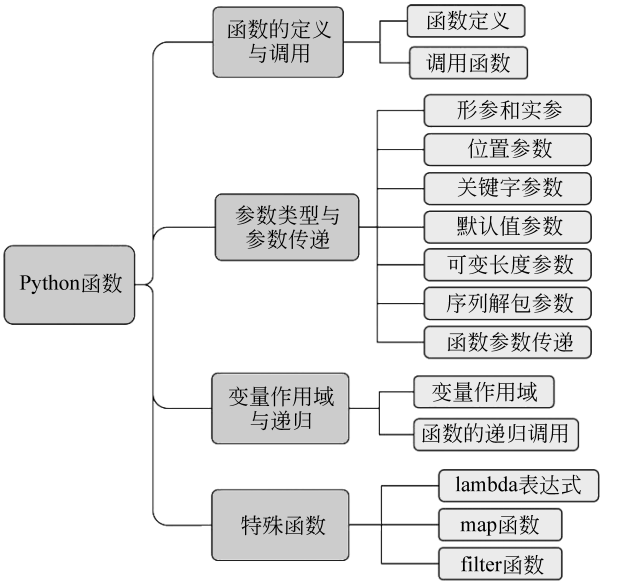
第5章

函　　数

本章要点

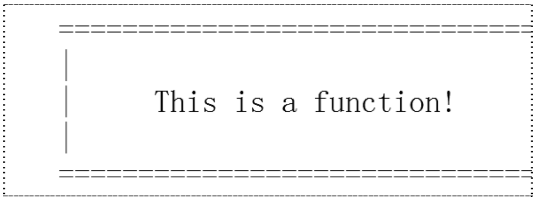
- 形参和实参
- 位置参数
- 关键字参数
- 默认值参数
- 可变长度参数
- 序列解包参数
- 函数的递归调用
- lambda 表达式
- map 函数
- filter 函数

本章知识结构图



本章示例

定义函数，将指定内容居中打印。



函数有时也称为方法，对于函数，实际上我们并不陌生，在前面一些章节的学习中，已经多次使用到 Python 的标准函数。例如，int()函数可以将用户输入的数字字符串转换成整型，type()函数可以获取某一个对象的数据类型，list()函数可以将一个可迭代的对象转换成列表，range()函数可以生成一个整型的数列等。所谓的函数实际上就是具有特定功能的程序代码块。通过函数可以快速调用某些功能，从而不需要从零开始构建程序。借助于函数还可以很方便地在前人的基础上进行功能扩展，从而大大提升工作效率。所以，函数的学习和使用是实际项目开发过程中必不可少的一部分。

本章的主要内容涉及以下 4 个方面：函数的定义与调用，例如，在 Python 中函数是如何定义的，包含哪些部分，各部分在使用时有哪些注意事项，函数定义之后如何多次调用函数等；参数类型与参数传递，函数可以包含多种类型的参数，例如位置参数、关键字参数、带有默认值的参数、可变参数等，不同类型的参数在使用上有不同的规则，在参数传递的过程中又涉及可变类型的参数传递和不可变类型的参数传递；变量作用域与递归函数，根据变量定义的位置不同，例如是定义在函数内还是在函数外，可以将变量划分为局部变量和全局变量，不同类型的变量在使用上有所不同。而递归的思想在复杂程序经常会用到，通过递归可以大幅降低代码量，学习递归的关键是如何进行划分、如何退出递归；还介绍了一些特殊函数，如 lambda 表达式、map 函数、filter 函数等。


视频讲解

5.1 函数的定义与调用

本节将主要介绍如何将自己设计的具有特定功能的程序代码封装成函数，从而方便他人或者自己在其他地方进行调用。

5.1.1 函数的概念

函数是具有特定功能的代码块，其意义主要是为了方便代码的重复使用。2.1 节中介绍了打印菱形的程序，用户输入一个菱形的行数，就能打印出对应的菱形。如果想一次性打印多个菱形，用户可能需要多次编写打印菱形的业务逻辑代码，从而存在代码的重复。这时可以把打印菱形的业务逻辑代码定义为一个函数，然后把菱形的行数作为一个参数进行传递，需要打印几次菱形，就调用几次函数。

从使用者角度看，函数就像一个“黑盒子”，如图 5-1 所示，用户传入 0 个或多个参数，该“黑盒子”经过一定操作即可返回 0 个或多个值。例如打印菱形，对于用户而言，只需要知道传递参数 7，就打印一个 7 行的菱形；传递参数 6，就打印一个 6 行的菱形，至于菱形是怎么打印的，用户可以不管。从用户角度看，只要知道传递了什么参数，就能得到对应的结果。

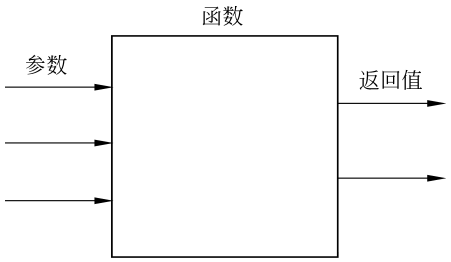


图 5-1　用户角度的函数调用流程图

但是从函数设计者（实现函数的编程人员）角度看，在设计函数时，一般需要考虑以下几个问题：

（1）函数中哪些部分是动态变化的，即哪些部分应该被定义为参数；

（2）函数要实现什么功能，最终给用户返回什么结果；

（3）函数如何实现这些功能，即函数体。

5.1.2　定义函数

Python 定义函数的语法如下：

```
def 函数名([形参列表]):
    函数体
    [return [返回值]]
```

- def：用来定义函数的关键词；
- 函数名：从语法角度来看，函数名只要是一个合法的标识符即可；从可读性角度来看，函数名应见名知意，由小写字母组成，多个单词间用下画线隔开；
- 形参列表：函数可包含 0 个或多个参数，Python 中形参不用指定类型，多个参数间用逗号隔开，调用时对其传递参数值；
- 函数体：由一条或多条语句组成的代码块；
- return：返回函数结果，可选。函数可以有返回值，也可以没有返回值，还可以有多个返回值，多个返回值以元组形式返回。

需要注意的是：def 和函数名之间有一个空格；函数可以没有参数，但小括号不能少；函数体要注意缩进，通过缩进控制层次关系。

【示例 5.1】 定义一个无返回值的函数。将指定内容居中打印，一共显示 5 行，内容前后都有 5 个空格。

```
def center_print(content):
    width = len(content) +5 * 2              #计算中间部分长度
    print("="*(width+2))                     #打印第一行
    print("|", " "*width, "|", sep="")       #打印第二行
    print("|", content.center(width), "|", sep="")  #打印中间内容
    print("|", " "*width, "|", sep="")       #打印第四行
    print("="*(width+2))                     #打印最后一行
```

【示例 5.2】 定义一个有返回值的函数：判断是否为闰年，若是则返回 True，否则返回 False。

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

5.1.3 调用函数

函数调用的方式是函数名(实参列表)，实参列表中的参数个数要与形参个数相同，参数类型也要一致，否则会抛出 TypeError 错误。

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。当存在多个同名函数时，调用的是最近一次定义的函数。

根据函数是否有返回值，调用函数有两种方式：

(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。

【示例 5.3】 调用示例 5.2 的 is_leap_year()函数。

```
result = is_leap_year(2020)
print(result)
```

程序运行结果：

```
True
```

(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。

【示例 5.4】 调用示例 5.1 的 center_print()函数。

```
center_print("This is a function!")
```

程序运行结果：

```
==============================
|                            |
|     This is a function!    |
|                            |
==============================
```

思考与练习

5.1　判断题：定义函数时，可以没有参数，但必须有一对小括号。

5.2　判断题：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。

```
1  def func(x):
2      return x +2
3      return x +4
4
5  print(func(3))
```

5.2　参数类型与参数传递


视频讲解

5.1 节中简单介绍了 Python 的函数定义以及函数调用，在这个过程中都涉及参数的使用。接下来将详细介绍函数学习中最灵活的一部分，也就是参数。本节主要介绍函数参数的类型以及函数参数传递的特点。

5.2.1　形参和实参

首先，明确两个概念：形参和实参。形参是在函数定义时写在小括号中的变量。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。而实参是在函数调用时实际传递的值。

例如，在 5.1.2 节定义的 is_leap_year()函数中，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。而在 5.1.3 节的 is_leap_year()函数调用中，is_leap_year(2020)中的 2020 就是实参，有实际值。形参和实参的名称(变量名)可以相同也可以不同，但是它们表示不同的含义。

在 Python 中，定义函数时不需要指定形参的类型，形参的类型由调用者传递的实参的类型决定。实参与形参通常个数相同，当形参是可变长度参数时，实参可以有多个，此

时实参和形参的个数不相同。

根据函数参数赋值的特点、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

位置参数指函数调用时，根据函数定义时形参的位置顺序依次将实参的值赋值给形参。位置参数要求实参和形参的个数必须一致，而且实参和形参必须一一对应。位置参数是 Python 中最简单、最常见的参数。

【示例 5.5】 定义一个函数，调用该函数时通过位置参数赋值。

```
1  def my_func(a, b, c):
2      print("a =", a)
3      print("b =", b)
4      print("c =", c)
5
6  my_func(20, 30, 40)          #按顺序依次将实参的值传给形参
```

程序运行结果：

```
a = 20
b = 30
c = 40
```

5.2.3 关键字参数

关键字参数指函数调用时，以“键-值”形式指定实参，此时将会根据形参变量名对该变量进行赋值。实参顺序和形参顺序可以不一致，但是要求传递的关键字一定要在形参列表中，否则将报错。关键字参数灵活、方便，调用者不用关注参数顺序和位置。

【示例 5.6】 定义一个函数，调用该函数时通过关键字参数赋值。

```
1  def my_func(a, b, c):
2      print("a = ", a)
3      print("b = ", b)
4      print("c = ", c)
5
6  my_func(b=5, a=10, c=15)     #根据形参变量名对该形参变量进行赋值
```

程序运行结果：

```
a = 10
b = 5
c = 15
```

5.2.4　默认值参数

默认值参数指函数定义时，在形参列表中直接为参数赋值来指定该参数的默认值。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。默认值参数方便函数调用，可减少参数传递。

在前面章节的函数中已经多次使用默认值参数，例如，print()函数中的 sep 默认是通过空格对输出的多个内容进行分割，range()函数中的 step 默认为 1 等。

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

【示例 5.7】 定义一个带有默认值参数的函数，然后调用该函数。

```
1   def my_func(a, b, c=5):
2       print("a =", a)
3       print("b =", b)
4       print("c =", c)
5
6   my_func(20, 30)                    #未传值时，参数 c 采用默认值
7   my_func(20, 30, 50)                #传值时，用新的值替换默认值
```

程序运行结果：

```
a = 20
b = 30
c = 5
a = 20
b = 30
c = 50
```

5.2.5　可变长度参数


视频讲解

可变长度参数指函数定义时无法确定参数的个数。例如系统中的 print()函数，如果不知道用户需要打印多少个对象，则可将需要打印的内容定义为可变长度参数，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：“* 参数名”和“** 参数名”。

* 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，主要针对以位置传值的实参。

【示例 5.8】 定义一个带有可变长度参数的函数，可变长度参数的形式为“* 参数名”，然后调用该函数。

```
1  def func_1(a, *b) :                    #该参数是一个元组类型
2      print("a =", a)
3      print("b =", b)
4
5  func_1(20, 30, 40)
```

程序运行结果：

```
a = 20
b = (30, 40)
```

**参数名：表示该参数是一个字典类型，可接受多个实参，并将传递的键值对存放到字典中，主要针对以关键字传值的实参。

【示例 5.9】 定义一个带有可变长度参数的函数，可变长度参数的形式为“**参数名”，然后调用该函数。

```
1  def func_2(a, **b) :                   #第二个参数是一个字典类型
2      print("a =", a)
3      print("b =", b)
4
5  func_2(20, b=10, c=20, d=30)
```

程序运行结果：

```
a = 20
b = {'b': 10, 'c': 20, 'd': 30}
```

5.2.6 序列解包参数（进阶）

序列解包参数：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。主要有两种形式：“*序列对象”和“**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。当实参为序列对象时，会将其看成一个整体赋值给某个形参。

【示例 5.10】 定义一个函数，调用该函数时通过序列解包参数赋值，序列解包参数的形式为“*序列对象”。

```
1  def func_1(a, *b):
2      print("a =", a)
3      print("b =", b)
4
5  func_1([20, 15, 30], 40, 50)           #实参为序列对象
6  func_1(*[20, 15, 30], 40, 50)          #实参为"*序列对象"
```

程序运行结果：

```
a = [20, 15, 30]
b = (40, 50)
a = 20
b = (15, 30, 40, 50)
```

当传递的实参为“**字典对象”时，将会根据关键字来匹配相应的形参，如果没有匹配到则报错；如果匹配到，则将相应的值赋值给该形参。当实参为字典对象时，会将其看成一个整体赋值给某个形参。

【示例 5.11】 定义一个函数，调用该函数时通过序列解包参数赋值，序列解包参数的形式为“**字典对象”。

```
1  def func_2(a, **b):
2      print("a = ", a)
3      print("b = ", b)
4
5  func_2({"a": 5, "b": 0}, c=20, d=30)   #实参为字典对象
6  func_2(**{"a": 5, "b": 0}, c=20, d=30) #实参为"**字典对象"
```

程序运行结果：

```
a = {'a': 5, 'b': 0}
b = {'c': 20, 'd': 30}
a = 5
b = {'b': 0, 'c': 20, 'd': 30}
```

5.2.7　多种类型参数混用（进阶）

多种类型参数混用时应注意：

（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，一般建议可变参数放在形参的最后。

（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 * 参数和**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，可以理解为可变参数的默认值为空。

【示例 5.12】 定义一个函数 student_info()，首先采用不同方式传递参数，然后调整参数位置，观察输出结果。

```
def student_info(name, age, *contacts, gender="男", **others):
    print("="*10, "学生基本信息", "="*10)  #标题
    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", 20, "手机-13823458765", "QQ-876534567", 身高=175, 籍贯="江西")
student_info(*["张静", 19, "QQ-875534597"], gender="女", **{"职务": "班长", "学号": "006"})
```

程序运行结果：

```
==========学生基本信息 ==========
姓名: 张三
年龄: 20
性别: 男
联系方式:手机-13823458765      QQ-876534567
身高 : 175
籍贯 : 江西
==================================
==========学生基本信息 ==========
姓名: 张静
年龄: 19
性别: 女
联系方式:QQ-875534597
职务 : 班长
学号 : 006
==================================
```

(1) 调换 gender 与 contacts 位置：

```
def student_info(name, age, gender="男", *contacts, **others):
```