CHAPTER 第5章
分而治之——函数程序设计

学习目标：

- 理解函数程序设计的基本思想。
- 掌握函数的定义和函数调用的方法。
- 理解函数调用的过程和函数参数传递机制。
- 学会使用标准库中的函数。

迄今为止，我们已经学习了三种控制程序的结构：顺序结构、选择结构和循环结构。如果能把这三种控制结构灵活运用，使用堆叠嵌套技术，毫不夸张地说已经可以解决绝大多数问题了。前几章解决的问题都相对比较简单，问题的求解算法都比较明显，写出的程序一般是十几行，最多不过几十行，大多还不到一页纸的长度，都保存在一个 Python 源文件中，都是直接执行的。但是很多问题往往求解算法比较复杂，内容很多，实现的代码可能多达几百行，几千行，甚者几万行。这种规模的代码还能像前几章那样都保存在一个源文件中吗？在一个文件中那么多代码还是从头到尾都写在一起吗？回答应该是能！但是，设想一下成千上万行代码都写在一起，该有多么难以编辑控制。

一个规模比较大的问题，往往也不是一个人能独立完成的。即使能独立完成也要把它分解成若干个相对比较小的子问题，每个子问题还可以再分成若干个更小的子问题，这些小问题都解决了，整个问题就解决了。这种把大问题按层分解成子问题的过程是分而治之的过程，可以采用自顶向下、逐步求精的方法去分解。自顶向下、逐步求精的过程为函数程序设计提供了方法。每个子问题都可以对应一个独立的功能函数。一个问题经过函数化之后，所得到的函数并不是彼此孤立的，而是具有层次关系的一个整体，顶层的函数可以称为主函数，用 main 表示，main 调用它下一层的子函数，子函数再调用更下一层的子函数，这样有机地结合在一起。实际上，三种程序控制结构与函数相结合才是真正的结构化程序设计。

本章详细讨论如何定义**函数**，如何使用函数，函数之间是如何联系在一起的，如何定义**接口**、管理众多的函数、建立**函数库，或者说建立一个多个函数组成的模块（文件）**，比较系统地研究一下函数程序设计的基本方法。

本章要解决的问题：

- 再次讨论猜数游戏模拟问题；
- 是非判断问题；
- 递归问题；

- 简单的计算机绘图问题；
- 学生成绩管理系统的初步。

5.1 再次讨论猜数游戏模拟问题

问题描述：

问题同 4.7 节的问题描述，这里略。输入输出样例也请参考 4.7 节。

问题分析：

在 4.7 节已经采用自顶向下、逐步求精的方法研究了这个问题。已经认识到要解决一个比较复杂的问题，不可能一开始就能确定一个十分精细的解决方案，一般要经历一个从抽象到具体，逐步明晰的过程。开始先勾画出求解方案的一个比较粗糙的轮廓，确定一个比较抽象的概念，然后再逐步细化，把抽象的东西逐渐细化到可以实现的具体步骤。自顶向下、逐步求精的过程是一层层分解的过程。如果设猜数游戏模拟的顶层是问题的原始抽象描述，经过第一次分解问题变成了两个子问题，这样就有：

猜数游戏算法

① 计算机“想”一个数。

② 玩家“猜数”。

③ 问是否继续玩，回答 y/Y，返回到①，否则程序结束。

其中①和②的每一步都是比较抽象的子问题，分别命名为 make_magic 和 guess_number，make_magic 的功能是实现计算机“想”一个数，guess_number 的功能就是模拟一次猜数游戏的全过程。可以先认为它们都已经实现了(**先设一个桩，占一个位**)。因此猜数游戏就是顺序使用 make_magic 和 guess_number，其流程图如图 5.1 所示，这个比较粗糙/抽象的算法可以称为主算法，对应的流程可以称为**主流程**。

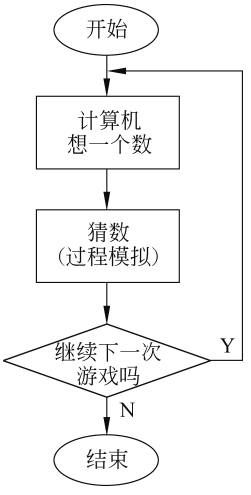

图 5.1 猜数游戏的主流程

如果用 main 表示主流程，把它认为是第一层，两个子问题对应的 make_magic 和 guess_number 是第二层，这样它们之间就形成了一个层次结构，如图 5.2 所示。这个层次结构表明了主算法和子算法之间的层次调用关系，图中单向的箭头线表示**调用**。因此可以说猜数游戏 main 调用 make_magic 和 guess_number。注意，实际 main 是不能主动执行的，Python 解释器调用 main 才能驱动整个游戏程序，即在 main 上面还有一层。

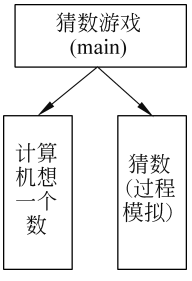

图 5.2 猜数游戏程序的层次结构

主流程和主算法和子算法的层次结构清楚之后，接下来就可以进一步研究每个子问题的解决方法了，即考虑 make_magic 怎么想一个数，guess_number 怎么猜的具体细节。如果子问题还很复杂，还很抽象，那就还要把每个子问题再次进行分解，每个分解出来的子子问题又对应一个更小的子问题，它们合起来构成本子问题的解决方案。如果经过第一次划分之后的子问题很容易求解了，就可以直接设计这个子问题的详细算法了。对于猜数游戏模拟问题，第

一次分解求精得到的子问题“让计算机想一个数”已经比较简单了，可以直接给出它的算法如下：

make_magic 算法

1.1　使用随机函数 randint( ) 产生一个 0～1000 的数 magic

对于“猜数”来说，第 2 步判断是否猜中，又加细求精了一次，所以其算法可以确定如下：

guess_number 算法

1.2.1　接受用户猜的数 guess

1.2.2.1　如果 guess＞magic 提示 too high 返回到 1.2.1

1.2.2.2　如果 guess＜magic 提示 too low 返回到 1.2.1

1.2.2.3　如果猜中，转到 1.2.3

1.2.3　输出祝贺信息

Python 中把这样的层次结构中的算法，包括主算法和子问题对应的算法都用函数表达，因此就有 main 函数，make_magic 函数和 guess_number 函数。有了这些函数之后，在程序中就可以按照给定的层次关系调用它们了。完整的程序见程序清单 5.1。

程序清单 5.1

```
#@File: guessNumFuncs.py
#@Software: PyCharm

from random import *

def main():
    """ The top level of the Game """
    print("Welcome to GuessNumber Game!\nWould you want beginning")
    c = input("Y/N or y/n? ")
    while c == 'y' or c == 'Y':
        magic=makeMagic()            #call makeMagic 函数
        guessNumber(magic)           #call guessNumber 函数
        c = input("Next time ? Y/N or y/n? ")

def makeMagic():
    """ Get a random number """
    return randint(1,1000)

def guessNumber(magic):
    """ Guessing """
    guess=int(input("Please guess a number between 1 and 999:"))
    while guess !=magic:
        if guess <magic:
            print("Too low!")
        if guess >magic:
            print("Too high!")
        guess=int(input("Please guess a number between 1 and 999:"))
```

```
28      print("Congratulation! you are right!")
29
30  if __name__ == "__main__":              #程序从这里开始执行
31      main()
32
```

其中在 main 里两个关键步骤 11 行和 12 行分别调用了函数 make_magic( )和 guess_number(magic),

下面将分节系统地讨论关于函数的一些细节。

5.1.1 模块化思想

现在总结一下刚才的讨论。在自顶向下、逐步求精的过程中,最初是比较粗糙的算法,把比较抽象的猜数游戏模拟归结为两个子问题,每个子问题对应一个函数。如果每个子问题能够很容易地得到解决,整个问题便得到了解决,如果子问题还比较抽象,就继续拆分成更小的子问题,又产生很多更小的子问题求解的函数,依此类推,直到很具体地能够解决为止。这个从抽象到具体的过程**蕴藏着一种层次结构**,所有的函数可以把它们组成一个解决这类问题的库,或者叫函数构成的模块,这种程序设计方法体现的就是模块化程序设计的基本思想,它是解决复杂问题或大规模问题的一种行之有效的策略——"分而治之"策略。

模块化程序设计使得一个比较大的问题转化为了若干个相对独立的、规模较小的子问题,这给软件开发带来了很多方便和好处。

- 整个库模块的开发可以由一个团队合作完成,每个成员只需完成其中的一部分;
- 开发一个或几个模块中的函数,不必知道其他函数内部结构和编程细节,只需知道它所需要的那些函数的接口,每个函数可以独立开发;
- 函数之间通过特别的消息传递机制(函数调用,参数传递)有机地结合在一起;
- 问题求解模块化之后,其函数之间具有层次结构,降低了复杂性,因此具有易读性,容易阅读和理解;
- 模块化还具有可修改性,对整个求解系统的修改只涉及少数部分函数;
- 模块化具有易验证性,每个功能函数可以独立测试验证,而且由于功能单一、规模较小,所以容易验证。
- 模块具有可重用性,每个功能函数可以反复使用,这一特征也称可复用。

注意:模块化程序设计要求每个功能函数的规模不要过大,函数的功能应该单一,即遵循函数功能的**高内聚**基本原则。函数的接口(名字和参数)应该尽可能简明,不同函数之间尽可能少地有关联,即遵循**低耦合**的基本原则。

5.1.2 函数定义

Python 结构化程序设计中把实现某种功能的一段代码封装在一起用**函数**表示,这个函数从功能上来看有点像数学函数,但表现形式和实现方法都与数学截然不同。在前面几章的学习中我们就已经接触到一些函数了,用来输出信息的 print 函数,用来输入信息的 input 函数,eval 函数等。不管是哪个函数,它们都对应一段代码。**函数定义**(**简称函数**)最基本的形式如下:

```
def  函数名  (参数列表):     #函数头
    """函数注释
    """
    函数体
    return
```

从整体上来看，**函数定义**是一段**有名字、有注释的、可执行的程序代码**，它由两部分组成：**函数头**和**缩进排列**的**函数体**。

1. 函数头

一个函数定义的头由三部分构成。

(1) **def**，这是一个系统的关键字，是 define 的简写。

(2) **函数名**，它是函数的标识符，其命名规则同变量名的命名规则一样。其作用类似变量名的作用，也可以称其为变量。变量名引用一个数据对象，**函数名则引用一个函数对象**(封装在一起的函数定义代码)，**因此函数定义就像一个赋值语句那样，它是一个可执行语句**(**executable statement**)。

(3) **参数列表**，它是逗号隔开的一些列表项，每一项有一个参数名。这组参数是将来函数被调用时函数能够接收的参数。声明格式为：

```
(参数名称,参数名称,…)
```

注意这个参数列表必须放在一对小括号之内，参数的个数大于或等于 0。一对小括号是函数的重要特征，只有看到小括号，才能确定其前面的名字是函数名。函数定义的参数列表可以为空，但小括号不能没有。例如：

```
max2 ( a, b)
```

的参数列表是(a, b)，说明当使用函数 max2 时它可以接受两个参数。但它只是在形式上给出了有什么参数，它叫**形参**。一般来说，函数在未被使用时，形参并不是什么变量，也不知道它引用什么类型的数据，只有在函数被使用的时候，形参才作为传递来的对象的变量自动产生，才有引用的对象(**注意：当参数具有默认值时例外**)。既然形参是形式上的参数，因此用什么名字是无关紧要的。

Python 函数的形参形式是比较丰富的，最基本的用法是**简单的形参名字**和**严格的参数顺序**，如函数 def print_rect(h, w)的功能是打印一个 h 行 w 列的字符图案，第一个参数的意义是行数，而第二个参数的意义是列数，两个参数如果交换一下顺序，意义就不同了。

注意：Python 的参数虽然没有限定数据类型和返回类型，但是也可以给用户一些提示，提示函数代码所期望的参数对象类型，如下面的函数定义对参数的数据类型和返回值的数据类型给出了提示，均为整型。

```
>>>def max2num (a: int, b: int) ->int:
```

但这仅仅是为用户服务的，机器内部是不理会的。

(4) **函数头结尾的冒号**：同分支循环结构的冒号一样，函数头结尾的冒号不能丢掉。

2. 函数体

函数体是函数定义的主体。

按照 Python 编码规范 PEP8，函数体的第一行，是可选的**函数注释**，一般是用三双引号括起来的文档字符串(DocString)，注释的内容要求用英文书写，包括函数的功能、函数的参数的意义、函数的返回值信息，且在函数功能描述之后有一空行。大家都知道，注释部分是被解释器忽略的，它仅仅为了方便阅读而存在，但它是比较重要的一部分。作为一个函数的设计者，不仅要能够设计出符合上述格式的、好用的函数，还要考虑代码的可读性，也就是不仅要熟悉前面几章学过的基本程序结构，写出正确的函数体代码，还要写好注释。

接下来是实现函数功能的代码。前面学过的各种语句都可以出现在函数代码中，如赋值语句、输入输出语句、函数调用语句、判断选择语句、循环语句等。也就是必须熟练掌握前面已经学过的输入输出语句，三种控制程序结构的语句等。在 Python 中函数体的代码中允许包含另一个函数的定义，**即函数的定义允许嵌套**。

一个函数体可以包含

返回语句：return [返回值列表]

返回语句是可选的。如果需要把一些值返回给将来使用它的语句——函数调用语句，可以用 return 返回。**实际上没有 return 语句时系统也会返回一个 None，这时**相当于有一个没有返回值列表的 return 或直接是 return None 的效果。因此，**可以说 Python 函数都是有返回值的**。只不过有的有显式的 return，有的没有。return 语句常常位于函数代码的后面，但有时也在函数体的内部，允许有多个 return 语句，当然它们肯定是有条件的返回，不可能多个 return 同时进行。在一个 return 的返回值列表中，可能包含多个值，即**允许返回多个值**。

函数头和函数体一起构成了解释器识别的函数定义。从函数的定义可以看出，一个函数实际上就是一组语句被封装在了一起，用一个名字来表示，这组语句在被执行之前会通过参数带进一些信息，在被执行之后将完成一个特定的任务，其结果通过 return 语句或其他形式返回。因此，可以说一个函数就是具有某种特别功能的一种工具。一般来说，使用函数的人比较关心的是要提供给函数什么样的参数它才能执行，函数执行后其结果会是什么，并不太关心函数内部到底是怎么工作的、如何实现的。因此，人们常常把函数看成是一个黑盒子，如图 5.3 所示。

图 5.3　函数是一个黑盒子

下面看几个函数定义的例子。

【例 5.1】 定义一个函数，求两个数的最大值。

定义一个函数要从函数头出发，即要确定函数名称，函数的参数列表。此函数的功能是对任意给定的两个数，求它的最大值并返回。参数的类型在定义时不确定。函数的命名应该尽量有意义，这里命名为 max2num，意思是两个数的最大值。完整的函数定义如下：

```
1   def max2num ( a, b):
2       """ The Sum of two numbers
3       :param a: one number, int type
```

```
4      :param b: another number, int type
5      :return : maximum number
6      """
7      if a >b:
8         result = a
9      else:
10        result = b
11     return result
```

或者把第 7～10 行的代码用 if-else 表达式简单地写成

```
return a if a>b else b
```

【例 5.2】 定义一个函数，打印 h 行 w 列的矩形图案。

这个函数要打印一个 h 行 w 列的图案，显然函数的参数列表应该有两个参数，第一个参数表示行数或者是高度，第二个参数表示列数，也可以认为是宽度。它的功能是打印图案，没有结果要返回，函数取名为 print_rect，完整的函数定义如下：

```
1   def print_rect(h, w):
2       """ Display a Rectangle
3       :param h: height
4       :param w: width
5       :return: none
6       """
7       for i in range(h):
8          for j in range(w):
9              print('*', end = '')
10         print()
```

【例 5.3】 定义一个函数，显示一个菜单界面。

当一个问题比较复杂时，经过分解会有很多子问题或者具有很多功能函数，这时可以给用户一个菜单界面提示，供用户选择，用户选择不同的功能就去执行不同的功能函数。而且这个界面是始终要显示在用户面前的，需要多次调用才能达到这样的效果。因此，有必要定义一个显示菜单界面的函数，函数命名为 menu，函数定义的完整实现如下：

```
1   def menu():
2      """ Display a Menu """
3       print("Welcome! Please give me your choice:")
4       print("=====================================")
5       print("      1 --append record")
6       print("      3 --display record")
7       print("      4 --modify record")
8       print("      5 --find  record")
9       print("      0 --append record")
```

【例 5.4】 定义猜数游戏模拟的 make_magic 函数。

它的功能产生一个随机“想”的数，不需要参数，返回一个数，见本节开始的函数定义。

【例 5.5】 定义猜数游戏模拟的 guess_number 函数。

它的功能是模拟猜数过程,需要知道计算机想的数是什么,因此要有一个参数,没有无返回值。见本节开始的函数定义。

【例 5.6】 定义一个函数 sort2num,对两个数排序。

```
1   def sort2num(a,b):
2       """Sorting a and b """
3       if a <b :
4           return a,b
5       else:
6           return b,a
```

注意:这个函数同时返回两个值,因此,调用返回时候需要两个接收变量,即

```
x, y = sort2num(4,2)
```

实际 return 还是返回一个值,内部有一个组合和拆分的过程,见 7.1.8 节。

一个函数的规模一般不要过大,控制在 50 行以内为宜。一个函数的功能也应尽可能单一,不要让一个函数的负担过重。这里需声明,**在本书的章节中,限于篇幅,大部分函数的注释部分都省略了或采用简化的方式**。

最后再次强调一下,**在 Python 中**函数定义 **def 是一个可执行语句,函数定义时相当于定义了一条赋值语句,函数名作为函数对象的引用**,因此 def 是**隐性的赋值运算**,但当函数被调用时,才执行函数体。

5.1.3 函数调用

函数是自顶向下、逐步求精的求解过程中分而治之的结果,每个子问题均可以定义为一个函数,函数之间呈现一种层次结构,最顶层的是主函数 main。一般来说,下一层的函数是为上一层的函数服务的,也就是说,上一层的函数要使用下一层的函数,当然如果需要的话,下一层的函数也可以使用上一层的函数,同层的函数也可以互相使用。**注意 main 函数是自定义的**。在一个函数中使用另一个函数称为**函数调用(Invoke)**,使用者称为**主调函数(或调用函数)**,被使用者称**被调用函数**。函数调用的一般形式是:

```
>>>函数名(实参列表)
```

其中函数名后面的一对小括号是必需的,**实参**列表是逗号隔开的,**一般来说**,它与函数定义中的形参列表一一对应。常量、变量或表达式均可以作为实参提供给形参需要的"值"。函数调用可以独立使用,形成一个函数调用语句,如调用 print_rect 函数打印一个 5 行 10 列矩形图案:

```
>>>print_rect(5,10)
```

也可以作为其他语句或表达式的一部分,如函数 make_magic 调用的结果赋给变量 magic:

```
>>>magic = make_magic()
```

注意:当使用某个函数时,必须从下面三个方面着手:首先要知道那个**函数的名字**是

什么，函数的功能是什么，然后看看它**有无参数**，如果有参数，还要进一步确认：

① 有几个参数；

② 参数的先后顺序如何。

③ 函数的**返回值**是什么样子的。

【例 5.7】 写一个程序，测试函数 max2int 和函数 print_rect。

我们以 max2int 函数和 print_rect 函数为例，看看它们是如何被调用的。首先注意这两个函数都是有参数的，函数 max2int 有一个返回值，而函数 print_rect 无返回值。

Python 语言规定，一个函数在被使用或调用之前**必须先定义**，不然解释器在执行的时候就会出现警告和错误，例如假设你要使用一个函数 func()

```
>>>func()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'func' is not defined
```

怎么让 Python 解释器知道已经定义某个函数了呢？如果在 shell 窗口解释执行，当然要先输入函数的定义了。**如果在程序脚本里定义函数，就没有先后之说了**，只要在同一个脚本文件中即可，至于在这个文件的哪个位置，都不受影响。如果要在 main 函数中使用 max2int 和 print_rect 函数，把 max2int 和 print_rect 两个函数的定义放在 main 函数定义之前还是之后都是一样的，**但是执行 main 调用的语句一定要在 main 函数定义之后**，见程序清单 5.2。

程序清单 5.2

```
1   #@File: myfuncstest.py
2   #@Software: PyCharm
3
4   """
5   Test how a function is invoked after defining
6   First one  is a function max2num, it have tow integer numbers parameters,
    return one max value.
7   Second one is a function print_rect to display rectangle with star sign by h
    rows and w columns
8   """
9   def main():
10      #常量作为实参
11      max = max2num(3,7)
12      print(max)
13      print(max2num(5, 10))
14      print_rect(5,10)
15
16      #变量作为实参
17      x = 5
18      y = 10
19      print(max2num(x, y))
```

```
20      print_rect(x, y)
21
22      #表达式作为实参
23      print(max2num(x+y, y))
24      print_rect(x, y+5)
25
26  def max2num (a: int, b: int) ->int:
27      """Get the maximum one from two numbers
28      :rtype: int
29      :param a: one number
30      :param b: another number
31      :return: a max one
32      """
33      if a >b:
34          result = a
35      else:
36          result = b
37      return result
38      #return ( a if a>b else b)
39
40  def print_rect(h: int, w: int) ->int:
41      """
42      :rtype: int
43      :param h: height
44      :param w: width
45      :return: none
46      """
47
48      for i in range(h):
49          for j in range(w):
50              print('*', end = '')
51          print()
52
53  if __name__ == '__main__':
54      main()
```

这个程序同前几章的程序相比大不相同,除了有 main 函数之外,在它前面还有两个自定义的函数。第 52 行前是 3 个函数的定义,它们仅仅是一些工具,如果不调用它们,它们是没有用的。这样的程序是怎么运行呢?整个程序的顶层只有一句话,就是第 53 行的判断结构,当条件为真时调用 main 函数,这是程序的入口。虽然在 main 的前面有那么多行代码,但运行这个程序时,从第 54 行跳到主函数,然后依次执行 main 函数里的从第 9 行到第 24 行的代码。其中有多条语句是进一步调用 max2num 和 print_rect 函数。以第 13 行的调用语句为例,看看计算机是怎么执行的。调用 max2num 函数要提供两个实参,第 13 行提供的是 5 和 10,即实参是 5 和 10,接下来会发生什么呢?接下来会暂时离开第 13 行,跳到 max2int 函数的定义,即跳到第 26 行,然后把实参的值 5 和 10 会传给形参 a 和 b,这时形参