第5章 函数与模块

锲而舍之，朽木不折；锲而不舍，金石可镂。

——《荀子·劝学》

在软件开发过程中，经常会在不同的代码位置多次执行相似或完全相同的代码块，这时就可以将需要反复执行的代码封装为函数或模块，并在需要执行该代码功能的地方进行函数调用或模块的导入，从而实现代码的复用。利用函数或模块这个特点，在软件开发过程中就可以把大任务拆分成多个函数或模块，这也是分治法的经典应用，即使复杂问题简单化，使软件开发像搭积木一样简单。

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。函数能提高软件的模块化和代码的重复利用率。

模块是方法的集合，用于有逻辑地组织 Python 代码段。把相关的代码分配到一个模块里能让代码更好用、更易懂。简单地说，模块就是一个保存了 Python 代码的文件。模块能定义函数、类和变量，模块里也能包含可执行的代码。

Python 的默认安装仅包含部分基本或核心模块，启动时也仅加载了基本模块，在需要时再显式地加载（有些模块可能需要先安装）其他模块，这样可以减少程序运行的压力，且具有很好的扩展性。

Python 可重用的第三方程序代码包括库、函数、模块、类、程序包等，这些可重用代码统称为库。在 Python 中使用到的库可以分为以下三类。

（1）Python 的内置函数。Python 的内置函数是默认安装和加载的基本模块，这些函数不需要引用库，直接使用即可。Python 共提供了 68 个内置函数。比如，在前面几章介绍和使用过的 print()、内置的数值运算函数、内置的字符串运算函数等。

（2）Python 标准库和第三方库（或称扩展库）。Python 之所以得到各行业领域工程师、策划师以及管理人员的青睐，与其庞大的第三方库有很大关系，而

且这些库每天都在以迅猛的速度在增加，大幅度地提高了各行各业软件的开发速度。这些大量的标准库和第三方库不是 Python 默认安装和加载的模块，需要导入之后才能使用其中的对象，模块的文件类型是.py。

标准库内置在 Python 安装包中，不需要安装，只需导入。受限于安装包的设定大小，标准库数量不太多，270 个左右，安装在 Python 安装目录的 Lib 目录下。

第三方库由全球各行业专家、工程师和爱好者开发。第三方库需要先正确安装才能导入。

（3）自定义函数。自定义函数是指程序员在编程过程中，发现某些代码需要重复编写，而 Python 内置函数、标准库和第三方库中又没有此类函数，因此需要自己定义的函数。

本章主要介绍后两类的使用。

5.1 模块的导入和使用

Python 标准库和第三方库都需要先导入才能使用。

Python 中导入模块的方法主要有 import 语句和 from…import 语句两种形式。

1. import 语句

导入模块的语法格式如下：

```
import 模块 1[, 模块 2[,… 模块 N]
```

使用模块的语法格式如下：

```
模块名.函数名(参数)
```

例如：

```
>>> import math
>>>math.sqrt(9)
3.0
>>>math.sin(2)
0.9092974268256817
```

2. from…import 语句

语法格式一：

```
from 模块名 import 函数名或变量名 1[,函数名或变量名 2[, ... 函数名或变量名 N]]
```

语法格式二：

```
from 模块名 import *
```

说明：

语法格式一是从模块中导入指定的模块成员。

语法格式二是把模块的所有内容全都导入到当前的命名空间。这种导入方式可以减少查询次数，提高访问速度，同时也减少了程序员需要输入的代码量，而不需要使用模块名作为前缀。这种导入模块方式虽然写起来比较省事，可以直接使用模块中的所有函数和对象而不需要再使用模块名作为前缀，但一般并不推荐使用。因为如果多个模块中有同名的对象，这种方式将会导致只有最后一个导入的模块中的同名对象是有效的，而之前导入的模块中的该同名对象无法访问。

使用模块的语法格式如下：

```
函数名(参数)
```

例如：

```
>>> from math import sqrt,sin
>>>sqrt(9)
3.0
>>>sin(2)
0.9092974268256817
```

【例 5-1】 导入 calendar 模块，用户输入想要查询的年和月后，显示该月的日历。

【程序 5-1.py】

```
1  from calendar import month
2  yy = int(input("输入年份: "))
3  mm = int(input("输入月份: "))
4  print(month(yy,mm))
```

【运行结果】

```
输入年份: 2019
输入月份: 7
```

```
     July 2019
Mo  Tu  We  Th  Fr  Sa  Su
 1   2   3   4   5   6   7
 8   9  10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31
```

另外，如果想使用已有的 Python 文件，也只需在另一个程序文件中执行导入语句即可。

【例 5-2】 已知 eg1.py 中有一个名为 f 的函数，如下所示。编写程序，在另外一个程序中调用 eg1.py 中的 f 函数。

【eg1.py】

```
1  def f():
2      print('eg1.py is OK!!!')
```

【程序 5-2.py】

```
1  from eg1 import *
2  print("调用 eg1 中的 f 函数")
3  f()
```

【运行结果】

```
调用 eg1 中的 f 函数
eg1.py is OK!!!
```

说明：在用 import 语句导入模块时，最好按照这样的顺序依次导入，即 Python 标准库模块→Python 扩展库模块→自定义函数。

5.2 Python 标准库

Python 标准库是 Python 安装的时候默认自带的标准库。Python 标准库无须安装。Python 标准库中有众多的库，这里介绍其中常用的三个库。

5.2.1 random 库

random 库是使用随机数的 Python 标准库。

Python 中随机数的生成基于随机数种子，根据输入的种子，利用算法生成一系列的随机数。random 库的常用函数有 9 个，见表 5-1 所示。

表 5-1 random 库的常用函数

函　数	描　述
seed(a=None)	初始化随机数种子，默认值为当前系统时间，例如： >>>random.seed(10)#产生种子 10 对应的序列
random()	生成一个[0.0,1.0)的随机小数，例如： >>>random.random() 0.5714025946899135
randint(a,b)	生成一个[a,b]的整数，例如： >>>random.randint(10,100) 84
getrandbits(k)	生成 1kb 长的随机整数，例如： >>>random.getrandbits(16) 37885
randrange(m,n[,k])	生成一个[m,n)的以 k 为步长的随机整数，例如： >>>random.randrange(10,100,10) 90
uniform(a,b)	生成一个[a,b]的随机小数，例如： >>>random.uniform(10,100) 16.848041210321334
choice(seq)	从序列类型(字符串、列表、元组)中随机选择一个元素，例如： >>>random.choice([1，2，3，4，5，6，7，8，9]) 8
shuffle(seq)	将序列类型(字符串、列表、元组)中元素随机排列，返回打乱顺序后的序列，例如： >>>s=[1，2，3，4，5，6，7，8，9] >>>random.shuffle(s) >>>s [9，4，6，3，5，2，8，7，1]

续表

函　数	描　述
sample(pop,k)	从组合类型(字符串、列表、元组、集合)中随机选取 k 个元素,以列表类型返回,例如: >>>random.sample("1234567",3) ['1', '4', '7'] >>>random.sample({1,2,3,4,5,6},2) [1, 4]

【例 5-3】 随机生成一个四位验证码。四位验证码中的每位元素可以是 3 种情况:数字 0～9、大写字母 A～Z 或小写字母 a～z。

【程序 5-3.py】

```
import random
checkcode=''
for i in range(4):                          #每循环一次产生一位验证码元素
    n=random.randrange(0, 3)                #生成随机数 0~2(因为有三种情况)
    if n==0:
        tmp=chr(random.randrange(65, 91))   #65~90 对应 A~Z
    elif n==1:
        tmp=chr(random.randrange(97, 123))  #97~122 对应 a~z
    else:
        tmp=random.randrange(0, 10)  #生成随机数字 0~9
    checkcode+=str(tmp)
print(checkcode)
```

【运行结果】

```
8ZjL
```

5.2.2 time 库

time 库是 Python 中处理时间的标准库,主要用于获取系统时间,提供系统级精确计时功能,以便进行程序性能分析。

1. 时间处理

时间处理主要包括如下 4 个函数。

(1) 使用 time.time()获取当前时间戳。

时间戳是指格林尼治时间 1970 年 01 月 01 日 00 分 00 秒(北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒)起至现在的总秒数。

Python 获取时间的常用方法是,先得到时间戳,再将其转换成想要的时间格式。例如:

```
>>> import time
>>>time.time()
1564450927.1466367
```

(2) 使用 time.gmtime([secs])把一个时间戳(按秒计算的浮点数)转换为 struct_time 对象。

日期、时间是包含许多变量的,所以在 Python 中定义了一个元组对象 struct_time 将所有这些变量组合在一起,包括 4 位数年、月、日、小时、分钟、秒等,其元素构成见表 5-2 所示。

表 5-2 struct_time 对象的元素构成

下标	属性	值
0	tm_year	年,4 位整数
1	tm_mon	月,1～12
2	tm_mday	日,1～31
3	tm_hour	小时,0～23
4	tm_min	分,0～59
5	tm_sec	秒,0～61(60 或 61 是闰秒)
6	tm_wday	星期,0～6 (0 是周一)
7	tm_yday	该年的第几天,1～366
8	tm_isdst	是否夏令时,0 为否、1 为是、－1 为未知

例如:

```
>>>time.gmtime()
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=30, tm_hour=2, tm_
min=12, tm_sec=57, tm_wday=1, tm_yday=211, tm_isdst=0)
```

(3) 使用 time.localtime([secs]把一个时间戳(按秒计算的浮点数)转换为本地时间的 struct_time 对象。例如:

```
>>>time.localtime()
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=30, tm_hour=9, tm_
min=53, tm_sec=4, tm_wday=1, tm_yday=211, tm_isdst=0)
```

(4) 使用 time.ctime(secs) 把一个时间戳(按秒计算的浮点数)转换为对应的易读字符串表示。例如:

```
>>>time.ctime()
'Tue Jul 30 10:13:18 2019'
```

2. 时间格式化

时间格式化主要包括如下 3 个函数。

(1) 使用 time.mktime(t)将 struct_time 对象 t 转换为时间戳,注意 t 为当地时间。该函数相当于 time()的逆函数。例如:

```
>>>time.mktime(time.localtime())
1564452132.0
```

(2) 使用 time.strftime(format[,t])将对象 t 格式化输出。strftime()的格式化控制符见表 5-3 所示。

表 5-3 strftime()的格式化控制符

格式化字符串	含 义	值
%Y	年份	0001～9999,例如 1970
%m	月份	01～12,例如 10
%B	月名	January～December,例如 April
%b	月名缩写	Jan～Dec,例如 Apr
%d	日期	01～31,例如 23
%A	星期	Monday～Sunday,例如 Wednesday
%a	星期缩写	Mon～Sun,例如 Wed
%H	小时(24 小时制)	00～23,例如 10
%I	小时(12 小时制)	01～12,例如 5

续表

格式化字符串	含　义	值
%p	上午/下午	AM,PM,例如 PM
%M	分钟	00～59,例如 22
%S	秒	00～59,例如 16

例如：

```
>>>time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())
'Tue, 30 Jul 2019 07:27:56 +0000'
```

(3) 使用 time.strptime(string[,format])提取字符串中的时间来生成 struct_time 对象,与 strftime()完全相反。例如：

```
>>>time.strptime('2019-7-30 10:30:35',"%Y-%m-%d %H:%M:%S")
time.struct_time(tm_year=2019, tm_mon=7, tm_mday=30, tm_hour=10, tm
_min=30, tm_sec=35, tm_wday=1, tm_yday=211, tm_isdst=-1)
```

3. 计时

计时主要包括如下两个函数。

(1) 使用 time.sleep(t)函数推迟调用线程的运行,t 是指推迟执行的秒数。例如 time.sleep(5)表示推迟 5 秒钟调用线程。

(2) 使用 time.perf_counter()返回系统运行的精准时间。

【例 5-4】 测试一下当前计算机运行某程序的速度。

【程序 5-4.py】

```
1  from time import *
2  d1=perf_counter()  #返回计时器的精准时间(系统的运行时间)
3  for i in range(1,10000000+1):
4      pass
5  d2=perf_counter()
6  print("程序运行时间是:{}秒".format(d2-d1))
```

【运行结果】

```
程序运行时间是:0.40451200419390826 秒
```

5.2.3 datetime 库

datetime 模块在支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。例如：

```
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2019, 7, 31)
>>>now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'07-31-19. 31 Jul 2019 is a Wednesday on the 31 day of July.'
```

5.2.4 tkinter 库

tkinter 是 Python 中可用于构建 GUI(Graphical User Interface，图形化用户界面，也称为图形用户接口)的众多工具集之一。

1. tkinter 编程

由于 tkinter 是内置到 Python 的安装包中，只要安装好 Python 之后，就能导入 tkinter 库，而且 IDLE 也是用 tkinter 编写而成，对于简单的图形界面 tkinter 能应付自如。例如：

```
>>> from tkinter import *
>>> window = Tk()
>>>window.mainloop()
```

以上代码可以显示一个空白窗口。可以将其看成是应用程序的最外层容器，创建其他插件的时候就需要用到它。如果关闭屏幕上的窗口，则相应的窗口对象就会被销毁。所有的应用程序都只有一个主窗口；此外，还可以通过 TopLevel 小插件来创建其他的窗口。tkinter 的小插件包括 Button、Canvas、Checkbutton、Entry、Frame、Label、Listbox、Menu、Message、Menubutton、Text、TopLevel 等。

2. tkinter 组件

tkinter 的提供各种控件，如按钮，标签和文本框等，见表 5-4 所示。