第 5 章

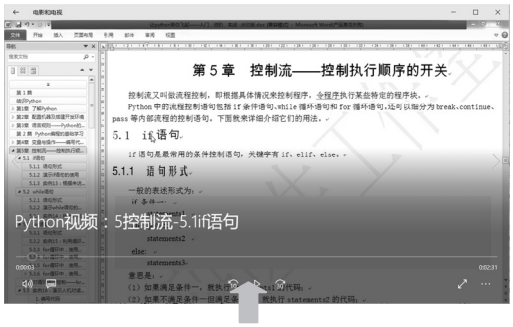
控制流——控制执行顺序的开关

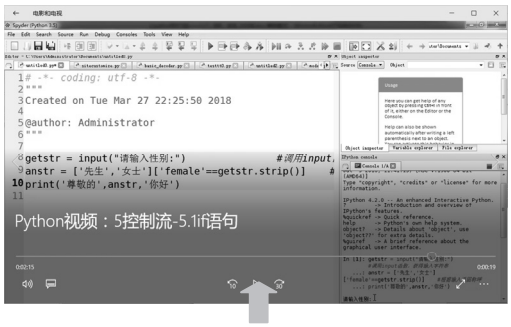
控制流又叫作流程控制，即，根据具体情况来控制程序，令程序执行某些特定的程序块。

Python 中的流程控制语句包括 if 条件语句、while 循环语句和 for 循环语句。还可以细分为 break、continue、pass 等内部流程的控制语句。

本章教学视频说明

作者按照图书的内容和结构，录制了同步对应的教学视频。





按照图书的结构，像老师一样讲解

具体代码操作演示


Python视频：5控制流-5.1if语句.mp4


Python视频：5控制流-5.2while语句.mp4


Python视频：5控制流-5.3for语句.mp4


Python视频：5控制流-5.4-5.5循环控制语句.mp4


Python视频：5控制流-5.6-5.7列表表达式.mp4

本章共有 5 段教学视频，总时长为 34 min 左右。包含如下内容：

- 讲述了 if 语句，并通过“实例 13”的演示来强化相关知识。
- 讲述了 while 语句，并演示了“实例 14”中的代码，将十进制转成二进制。
- 首先讲述了 for 语句，接着演示了“实例 15”中使用 for 循环实现冒泡排序，最后介绍了 for 循环与内置函数 range、zip、enumerate 一起使用的知识。
- 讲述了循环过程中的控制语句，包括：break、continue、pass 语句，并演示了“实例 16”的代码，实现一个人机对话中的控制流程。
- 讲述了列表表达式，并通过“实例 17”演示了列表表达式的基本操作。

5.1 if 语句

if 语句是最常用的条件控制语句，关键字有 if、elif、else。

5.1.1 语句形式

一般的表述形式为：

```
if 条件一:
   statements1
elif 条件二:
   statements2
else:
   statements3
```

意思是：

（1）如果满足条件一，则执行 statements1 代码。

（2）如果不满足条件一，但满足条件二，则执行 statements2 代码。

（3）如果条件一和条件二都不满足，则执行 statements3 代码。

这里的条件一、条件二、条件三，分别代表三个条件判断语句。当条件判断语句返回值为 True 时，则代表满足该条件。

5.1.2 演示 if 语句的使用

下面通过代码来演示 if 语句的用法。

```
a = 4          #定义变量 a 的值为 4
b = 5          #定义变量 6 的值为 5
if a>b:        #当 a 大于 b 时;
   c = a       #令 c 等于 a 的值
else:          #否则 c 等于 b 的值
   c = b
print(c)       #将 c 显示。输出：5
```

在上面代码中，先比较 a 与 b 的值，当满足 a 大于 b 的条件，则执行 c 等于 a 的语句；否则会执行 c 等于 b 的语句。因为 a 的值为 4，并不大于 b 的值（5），所以将 b 的值赋给了 c，输出的结果为 5。

注意：

上面的代码中，if 与 else 部分还有一个简化的写法，即，c = [a,b][a<b]。这是一个小技巧，可以使代码更为简洁。这种由两个中括号组成的语法，完整的意思是：

（1）如果第二个方括号里的条件值为假，则返回第一个方括号中的第一个元素。

（2）如果第二个方括号里的条件值为真，则返回第一个方括号中的第二个元素。

5.1.3　实例 13：根据来访人的性别选择合适的称呼

下面模拟一个人机对话的场景，使用本节前面的知识来控制程序，实现一个动态打招呼的功能。

实例描述

通过一个输入函数来模拟系统接口，获取来访人的性别，并根据输入的性别进行不同的处理：

（1）如果输入“female”，则显示“尊敬的女士，你好”；

（2）如果输入“male”，则显示“尊敬的先生，你好”。

使用变量 getstr 来接收 input 函数得到的键盘输入，根据输入返回来访人性别称呼的字符串，最后将字符串输出。代码如下：

代码 5-1：根据来访性别选择合适的称呼

```
getstr = input("请输入性别:")                          #调用 input 函数，获得输入字符串
anstr = ['先生','女士']['female'==getstr.strip()]      #根据输入返回称呼
print('尊敬的',anstr,'你好')
```

上面代码运行后，输出如下结果：

```
请输入性别：
```

这时，程序会挂起，等待用户输入。通过键盘输入“male”，并按 Enter 键。程序继续执行，输出如下结果：

```
请输入性别:male
尊敬的 先生 你好
```

5.2 while 语句

While 语句用来表述一个循环执行的代码流程。

5.2.1 语句形式

语句的形式为：

```
while 条件一：
   statements1
```

该语句的执行过程，可以分解成如下步骤：

（1）执行条件一判断语句，看是否返回 True。

（2）如果返回 True，则执行下面的 statements1 代码。

（3）执行完 statements1 的代码后，再回到第（1）步。

（4）如果第（1）步返回 False，则整个语句结束。

statements1 属于 while 的子代码块，每一行的开头都需要缩进。

5.2.2 演示 while 语句的使用

下面通过代码演示 if 语句的用法。

```
c=4                  #定义一个变量 c
while c>0:           #使用 while 循环。当 c 大于 0，就执行下面语句
   print(c)          #输出 c 的值
   c -=1             #c 自身减 1
```

在上面代码中，while 的循环条件是 c 大于 0。在 while 循环体里，每次都会打印出 c 的值，并将 c 自身减 1。代码运行后输出如下：

```
4
3
2
1
```

变量 c 的初始值为 4。c 经过 4 次减 1 操作将会变为 0，不符合 while 的条件，于是结束循环。

5.2.3　实例 14：将十进制数转化为二进制数

下面通过实例演示 while 的使用。

实例描述

通过输入函数获取一个十进制数值，并将其转换为二进制数并输出到屏幕。

转换的原理是：将输入的十进制数值循环地除以 2，直到返回的结果商为 0 为止。这期间，每次除 2 的余数就是转换后的二进制结果。使用变量 a 来接收 input 函数接到的键盘输入，并将其转化为整型，然后通过 while 循环来处理。代码如下：

代码 5-2：将十进制数转化成二进制数

```
a = input("请输入一个十进制数：")          #获取一个十进制数
d=int(a)                                  #将输入的字符串转为整型
s=""
while d!=0:                               #使用 while 循环，直到 d 的值为 0
    d,f=divmod(d,2)                       #除以 2，并返回除数和余数
    s=str(f)+s                            #余数作为转换后的二进制数，除数作为循环条件
print(s)                                  #将结果输出
```

上面代码运行后，输出如下结果：

```
请输入一个十进制数：
```

这时，程序会挂起，等待用户输入。通过键盘输入“6”并按 Enter 键。程序继续执行，输出如下结果：

```
110
```

结果显示为 110。正是十进制数 6 转化成二进制数的结果。

5.3 for 语句

Python 中的 for 语句与 while 语句都是用来循环执行代码块的。for 循环的执行条件，不是判断逻辑是否为真，而是遍历一个序列容器。

5.3.1 语句形式

for 的语句形式为：

```
for item in 序列数据:
 statements1
```

意思是，每次从序列容器中取一个值（item），然后执行语句 statements1。当遍历完整个序列容器，循环也就结束了。

关于 for 循环和 if 语句的使用，可以参见 5.3.4 小节的实例演示。

5.3.2 在 for 循环中，使用切片

序列容器可以是任何 Python 支持的序列数据类型。

如果在循环体内，执行语句 statements1 对 for 后面的序列数据进行了修改，这会影响到 for 语句的初始循环次数。切片的介绍见 4.3.4 小节。

避免上面问题的方法是：使用切片的方法为该序列数据做一个副本，让 for 来遍历副本中的序列数据，这样即使 statements1 语句修改了原始的序列容器中的数据，也不会影响到初始的循环次数了。例如：

```
words = ['I','love','Python']  #定义一个列表
for item in words[:]:          #for 后面没有使用 words，而是使用了切片作为 words 的副本
   words.insert(0, item)       #向 words 里插入一个元素
   print(item)                 #打印 item，三次迭代分别输出：'I'、 'love'、 'Python'
print(words)    #打印整个 words，输出：['Python', 'love', 'I', 'I', 'love', 'Python']
```

上面代码中的第 2 行，使用了 words 的切片作为副本，完成了 3 次迭代（因为有 3 个元素，各迭代一次）。

再来看一个错误的写法。假如不使用副本，将第二句代码换成：

```
for item in words:
```

这将会带来死循环。因为，每次从 words 中遍历一个数据，执行下面语句时，就会为 words 添加一个数据，这样永远也无法将 words 中的数据遍历完。注意，一定要避免这种错误的写法。

5.3.3　在 for 循环中，使用内置函数 range

在 for 循环中，还可以使用内置函数 range 来遍历一个数字序列。

1. range 介绍

range 的意思是返回一个数字区间的所有整数。

（1）输出大于零的序列

单独的 print(range(5))打印不出 0~5 之间的数字。输出的是“range(0, 5)”，代表从 0~5 的一个范围。

如果要想将其内容散列出来，可以将其转化成 list 类型再打印，例如：

```
print(list(range(5)))  #将 0~5 间的整数转化成 list 类型，并打印出来。输出：[0, 1, 2, 3, 4]
```

range 中的参数默认是从 0 开始。上例中，range 的参数 5 代表从 0~5 之间的数。这里 0~5 区间的取值仍然与切片的取值一致，即“要头，不要尾”。意思就是，0~5 区间的数包含起始值（0），但是不包含结束值（5）。

（2）输出大于零的序列

如果要将比 0 小的数传入 range，会打印不出内容。例如：

```
print(list(range(-5)))      #-5 比 0 小，range 找不到任何比 0 大的数，只能返回空。输出：[]
```

这种情况下，可以通过指定起始值来将–5~0 之间的数打印出来。例如：

```
print(list(range(-5,0)))  #将-5~0 间的整数转化成列表类型，并打印出来。输出：[-5, -4, -3, -2, -1]
```

上例中，range 里面的第一个参数–5 代表起始值，第二个参数 0 代表结束值。

（3）步长

range 函数中还可以有第三个参数——步长。即，从起始到结束，每隔“步长个数字”返回一次。例如：

```
print(list(range(-5,0,2)))  #在-5~0 间，每隔两个数取出一个，并转化成 list 类型，打印出来。输出：[-5, -3, -1]
```

2. range 与 for 结合

了解完 range 后，再来看一个 range 与 for 组合的例子：

```
for i in range(5):          #循环遍历 0~5 之间的整数
   print(i)                 #将取出的值打印出来
```

这样就实现了循环 5 次的控制。在 range 中放置一个整数（5）的方式，指定了代码的具体循环次数。上例执行后，输出：

```
0
1
2
3
4
```

5.3.4 实例 15：利用循环实现冒泡排序

在算法编程领域，循环与判断语句的应用非常频繁。下面就来实现一个算法的例子。

实例描述

使用 for 循环对一个列表进行排序。要求不使用内置的库函数。

冒泡排序是数据结构中的经典算法。手动实现冒泡排序，对初学者锻炼自己的编程逻辑有很大帮助。

冒泡排序算法的运作过程如下：

（1）比较相邻的元素。如果第一个比第二个大，就交换它们两个。

（2）从最开始的第一对到结尾的最后一对，对每一对相邻元素做步骤（1）所描述的比较工作，并将最大的元素放在后面。这样，当从最开始的第一对到结尾的最后一对都执行完后，整个序列中的最后一个元素便是最大的数。

（3）将循环缩短，除去最后一个数（因为最后一个已经是最大的了），再重复步骤（2）的操作，得到倒数第二大的数。

（4）持续做步骤（3）的操作，每次将循环缩短一位，并得到本次循环中的最大数。直到循环个数缩短为 1，即没有任何一对数字需要比较。最终便得到了一个从小到大排序的序列。

具体实现的代码如下：

代码 5-3：冒泡排序

```
01  n = [5,8,20,1]                          #定义一个列表
02  print("原数据：",n)
03
04  for i in range(len(n)-1):                #这个循环负责设置冒泡排序进行的次数
05     for j in range(len(n)-i-1):           # j 为列表索引（索引可参见 4.3.4 小节）
06        if n[j] > n[j+1]:                  #比较两个数
07           n[j], n[j+1] = n[j+1], n[j]     #交换
08
09  print("排序后：",n)                       #输出结果
```

上面代码使用了两层循环：

- 外层循环，赋值冒泡排序进行的次数，见代码第 4 行。
- 内层循环，负责将列表中相邻的两个元素进行比较，并调整顺序。将较小的数放在前面，较大的数放在后面，见代码第 6、7 行。

程序运行后，输出如下结果：

```
原数据： [5, 8, 20, 1]
排序后： [1, 5, 8, 20]
```

5.3.5　在 for 循环中，使用内置函数 zip

for 语句还可以配合内置函数 zip，以同时遍历多个序列。

1. zip 介绍

zip 函数可以将任意多的“序列”类型数据结合起来，生成新序列数据。生成的新序列数据中，每个元素都是一个元组。该元组的元素是由传入 zip 函数中的多个序列数据的元素组成。例如：

```
x=[1,2,3]                    #定义两个列表，x 与 y
y=[3,2,"hello"]
t = zip(x,y)                 #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(tuple(t))              #将 t 转成元组，并打印。输出：((1, 3), (2, 2), (3, 'hello'))
```

当然，生成的序列 t 也可以转成列表（list）类型。例如：

```
x=[1,2,3]                    #定义两个列表，x 与 y
y=[3,2,"hello"]
t = zip(x,y)                 #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(list(t))               #将 t 转成列表，并打印。输出：[(1, 3), (2, 2), (3, 'hello')]
```

需要注意的是，当 zip 对象（t）被转化为元组或列表后，就会自动销毁。如果再使用 t，将得不到具体的元素。例如：

```
x=[1,2,3]                    #定义两个列表，x 与 y
y=[3,2,"hello"]
t = zip(x,y)                 #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(list(t))               #将 t 转成列表，并打印。输出：[(1, 3), (2, 2), (3, 'hello')]
print(tuple(t))              #再次使用 t，将 t 转成元组，得到的是空元组。输出： ()
```

zip 对象还可以通过前面加个字符“*”的方式来完成 unzip 的过程，所谓 unzip，就是将 zip 生成的数据返回。例如：

```
x=[1,2,3]                #定义两个列表，x 与 y
y=[3,2,"hello"]
t = zip(x,y)             #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(*t)                #在 t 前面加一个*，完成 unzip。输出：(1, 3) (2, 2) (3, 'hello')
print(*t)                #同样，t 只能 unzip 一次，再次使用时，会返回空值。没任何输出
```

上例中，在最后两行特意调用了两次 zip 对象 t。可以看到第二次调用 t 时，得到的输出是空的。这也是值得注意的地方。

如果 zip 里面的序列长度不同，就会以最短的序列数据为主。例如：

```
x=[1,2,3,4,5,6]              #定义两个 list——x 与 y，x 的长度会更大一些
y=[3,2,"hello"]
t = zip(x,y)                 #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(list(t))               #将 t 转成列表。输出：[(1, 3), (2, 2), (3, 'hello')]
```

上例中，y 的长度最短，只包含 3 个元素。所以生成的 t 也只有 3 个元素。

传入 zip 中的类型可以不同。下面演示 zip 参数一个是列表和一个是元组的情况：

```
x=[1,2,3,4,5,6]              #定义 1 个列表 x
y=(3,2,"hello")              #定义 1 个元组 y
t = zip(x,y)                 #通过 zip 生成一个新的序列 t，这时 t 是 zip 类型
print(list(t))               #将 t 转成列表。输出：[(1, 3), (2, 2), (3, 'hello')]
```

上例中，将列表 x 和元组 y 一起传入 zip 里一样可以得出 zip 对象。这表明，zip 的参数可以是任意序列类型。

2．zip 与 for 结合

了解完 zip 后，再来看一个 zip 与 for 结合的例子：