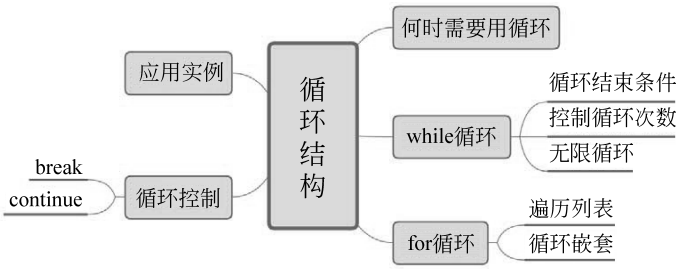
第5章

循 环 结 构

5.1 导　　学



学习目标：

- 了解循环语句的执行流程。
- 掌握 while 循环语句的基本用法。
- 熟练掌握 for 语句的单循环和多重循环方法。
- 掌握 continue 和 break 的使用。
- 熟悉循环的嵌套。
- 了解死循环的概念。

有时需要程序重复执行类似的代码很多次，如果使用之前学习过的知识来完成这个任务是吃力不讨好的。如从 1 开始到 1000 输出每个整数，只能写成：

```
print(1)
print(2)
...
print(1000)
```

可见，这是一个非常乏味并容易出错的过程。那么该如何解决这个问题呢？答案是使用循环。Python 提供的这种非常强大的功能，支持自动多次执行语句。通过使用循环，就不需要写成上面的形式，只需要写成下面这样：

```
i=0
while i <=1000:
    print(i)
    i=i+1
```

比较这两个程序，上面的语句需要写 1000 行，而现在只需要短短 4 行就解决，是不是很神奇？循环是一种控制一个语句块重复执行的控制结构，它是 3 种语句结构之一，是语言的重要基础。Python 提供了两种类型的控制语句：while 循环和 for 循环。while 循环是一个条件控制循环，由真/假条件控制。for 循环是一个由计数控制的循环，它重复指定的次数。

5.2 while 循环

5.2.1 while 循环语法

Python 编程中 while 语句用于循环执行程序，即在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。while 循环的语法是

```
while 判断条件:
执行语句
...
```

执行语句可以是单个语句或语句块。判断条件可以是任何表达式，任何非零、或非空(null)的值均为 True。当判断条件为 False 时，循环结束。执行流程图如图 5-1 所示。

5.1 节中输出从 1 到 1000 的例子使用上面的流程图如图 5-2 所示。

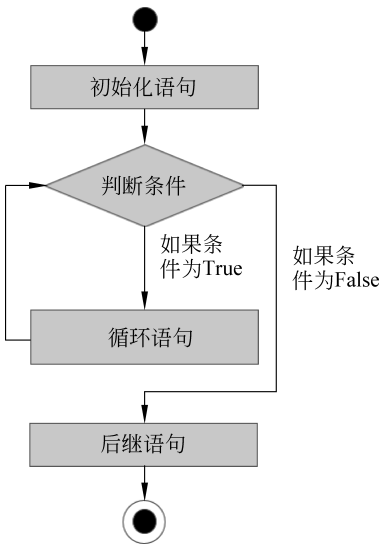


图 5-1 while 循环语句流程图

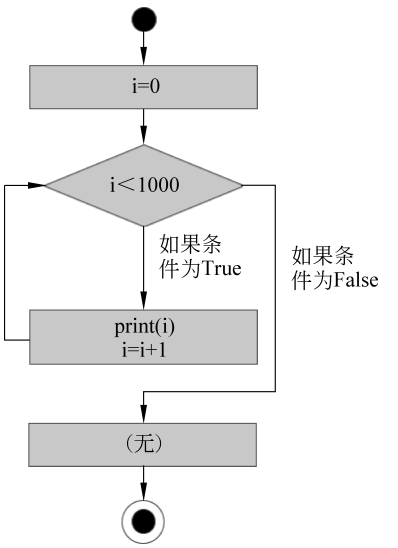


图 5-2 while 循环语句流程图示例

5.2.2 while 语句体

接下来看另一个例子，如要计算 1＋2＋3＋…＋100 的和。试着思考这个任务，这里应该使用循环，那么就需要考虑如图 5-1 所示的几个部分。

1. 初始化语句

初始化语句部分其实不属于循环的一部分，但对于循环的正确运行有着重要的作用，如果没有设置足够的循环变量并设置正确的初始值，那么很可能循环得到的结果是错误的。在这个例子里面，要考虑使用一个变量来存储每个加法项，就是上面的 1、2、3 等数字。另外，还需要考虑使用另一个变量来保存当前加法的和。两个变量确定下来之后，加法项变量的初值简单，设为 1 就可以了。和变量的初值不能设为 1，必须设为 0 才行，因为如果没开始加的时候，和为 0。根据上面的分析，就完成了初始化语句：

```
sum=0
item=1
```

2. 判断条件

判断条件判断是否应该继续循环，它其实是一个布尔表达式，如果布尔表达式结果为 True，则继续循环，否则退出循环而执行其他后继语句。那么这个例子中，什么时候该结束循环呢？根据题目要求，item 从 1 开始一直到 100，那么意味着只需要判断 item 的值是否在 1 到 100 这样一个范围内。而 item 初值为 1，所以只需要判断 item 是否小于或等于 100，判断条件如下：

```
item <=100
```

3. 循环语句

循环语句是一个循环的最重要部分，它是每次循环（有时也称为迭代）都会执行的部分。在给定初值和判断条件的情况下，循环语句比较容易得到。这里，每次要做的事情分两步。首先是把当前的 item 值累加到 sum（和）中，其次是需要把 item 的值加 1，使其成为下一代累加项。代码如下：

```
sum=sum+item
item=item+1
```

4. 后继语句

要想圆满完成一个循环任务，有时还需要后继语句做一些收尾工作。本例中 sum 的输出工作放在后继语句中执行。代码如下：

```
print(sum)
```

最后把刚才这几个部分综合在一起，代码如例 5-1 所示。

【例 5-1】 完整循环示例

```
sum=0
item=1
while (item <=100):
    sum=sum+item
    item=item+1
print(sum)
```

最后的输出结果为 5050，表明代码是正确的，任务完成。

5.2.3 简单语句组

while 循环支持类似 if 语句的语法，如果 while 循环体中只有一条语句，可以将该语句与 while 写在同一行中，如下所示：

```
while(True): print("hello")
print("Good bye!")
```

这种语句并不常见，实际上这种循环是一个死循环。

5.2.4 while 循环常见错误

初学者在学习循环时是比较容易出现各种错误的，主要有如下几种情况。

1. 死循环

死循环也称无限循环，意味着循环永远无法结束。原因在于判断条件语句总是为 True，根本原因在于忘了修改循环迭代对象的值，例 5-1 修改后如例 5-2 所示。

【例 5-2】 死循环

```
sum=0
item=1
while (item <=100):
    sum=sum+item
print(sum)
```

在尝试上面的语句时，会发现程序一直没有输出。原因是 item 的值一直保持为 1，每次迭代的时候没有变化，那么 item <= 100 这个表达式的结果永远为 True，这就陷入死循环。还有一种情况如例 5-3 所示。

【例 5-3】 循环缩进问题

```
sum=0
item=1
```

```
while (item <=100):
    sum=sum+item
item=item+1
print(sum)
```

例 5-3 中的错误很明显，虽然想要改变 item 的值，但是因为错误的缩进，导致 item 的值变化发生在循环语句之外，同样陷入了死循环，这一点请大家注意。

2. 错误的边界

错误的边界发生在判断条件中，如例 5-4 所示。

【例 5-4】 错误的边界

```
sum=0
item=1
while (item<100):
    sum=sum+item
item=item+1
print(sum)
```

可以看出，item 的终止值应该是小于或等于 100 才对，或者写成 item＜101 也可以。当需要完成比这个布尔表达式更复杂的语句时一定要注意这个问题。

3. 错误的语句顺序

错误的语句顺序带来错误的结果，而且比较难以查找错误的位置，如例 5-5 所示。

【例 5-5】 错误的语句顺序

```
sum=0
item=1
while (item<100):
    item=item+1
    sum=sum+item
print(sum)
```

运行结果：

```
5049
```

造成错误的原因是，累加项的变化语句放在了累加语句的前面，导致少加了第一项：1，多加了一项：101。

5.3 for 循环

for 循环提供了 Python 中最强大的循环结构(for 循环是一种迭代循环机制，而 while 循环是条件循环，迭代即重复相同的逻辑操作，每次操作都是基于上一次的结果而进行的)。Python 中 for 循环常用于遍历序列类型数据，如一个列表或者一个字符串等。

5.3.1 for 循环语法

for 循环的一般格式如下：

```
for  迭代变量  in  迭代序列:
执行语句
...
```

每次循环时，迭代变量可以设置为迭代序列(也可为迭代器，或是其他支持迭代的对象，这些在后面的章节中介绍)的当前元素，提供给循环语句块使用。流程图如图 5-3 所示。

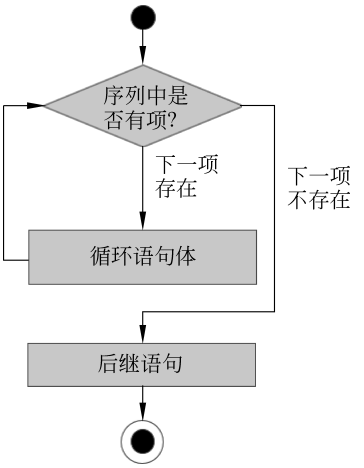


图 5-3　for 循环语句流程图

5.3.2 for 语句体

接下来将 5.2 节中从 1 累加到 100 的例子使用 for 循环做一遍。大体上，for 语句体里面的大部分和 while 语句体是一致的，也可以分为如下几部分。

1. 初始化语句

初始化语句和 while 循环是一样的，如下：

```
sum=0
item=1
```

2. 项判断语句

项判断语句主要判断当前项，是否存在于迭代序列中。这里序列为 1、2 一直到 100 的各个整数。那么如何表达一个迭代序列呢？需要用到一个函数 range()，该函数里面有两个参数，第一个参数表示为初值，第二个参数表示为终值。range(a,b)函数返回一系列连续的整数：a,a+1,a+2,…,b−2 和 b−1。本例中，项判断语句如下：

```
for item in range(1,101)
```

3. 循环语句

循环语句也和 while 循环一致。代码如下：

```
sum=sum+item
item=item+1
```

4. 后继语句

后继语句代码如下：

```
print(sum)
```

最后把刚才这几个部分综合在一起，代码如例 5-6 所示。

【例 5-6】 for 循环完整示例

```
sum=0
item=1
for item in range(1,101):
    sum=sum+item
    item=item+1
print(sum)
```

最后的输出结果为 5050，表明代码是正确的，任务完成。

5.3.3 range()函数

例 5-6 成功实现了累加 1 到 100 的例子，大家可能对 range()这个函数比较好奇。实际上的 range()函数比想象中更加强大，本节重点讨论这个函数。

range()函数的语法格式如下：

```
range([起始值, ]终止值[, 步长])
```

这里的起始值表示计数从它开始，默认是从 0 开始。例如 range(5)即 range(0,5)。终止值表示计数到它结束，但不包括它本身。例如：range(0,5) 是[0,1,2,3,4]没有 5。步长，表示每次迭代对象改变的大小，默认为 1。例如：range(0,5)等价于 range(0,5,1)。下面来看一个具体例子，这里为了输出方便，使用列表的构造方法将 range()函数的结果转换成了列表：

```
print(list(range(0, 10, 2)))
print(list(range(10)))
print(list(range(10, 2)))
print(list(range(10, 0, -3)))
```

运行结果：

```
[0, 2, 4, 6, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[]
[10, 7, 4, 1]
```

需要注意的是，因为起始值和步长都是可以省略的。当省略了一个值，那么默认省略的是步长。另外，起始值是可以大于终止值的，起始值大于终止值的情况下，步长是负数才能输出结果。还有，起始值是包含在 range()函数的范围内的，而终止值不在范围内，这点和切片相似。

接下来通过一个例子学习它，打印 0 到 20 中所有的偶数，代码如例 5-7 所示。

【例 5-7】 range()函数

```
for item in range(0,21,2):
    print(item)
```

运行结果：

```
0
2
4
6
8
10
12
14
16
18
20
```

5.4 循环控制语句

关键字 continue 和 break 提供了另一种控制循环的方式，在某些情况下，使用这两个关键字可以简化程序设计。

5.4.1 break 语句

要立即退出 while 循环，不再运行循环中余下的代码，也不管条件测试的结果如何，可使用 break 语句。break 语句用于控制程序流程，可使用它来控制哪些代码行将执行，哪些代码行不执行，从而让程序按要求执行要执行的代码。

例如，可以使用 break 语句在满足某些条件时立即退出 while 循环，如例 5-8 所示。

【例 5-8】 break 语句

```
for i in range(1,10):
    if (i==3):
        break
    print(i)
```

运行结果：

```
1
2
```

例 5-8 中，从 1 开始通过循环依次输出每一个整数。但当 i 等于 3 的时候，执行了 break 语句，循环就被终止了，print 语句不会被执行并且也不会继续下一次的循环。

5.4.2 continue 语句

要返回循环开头，并根据条件测试结果决定是否继续执行循环，可使用 continue 语句，它不像 break 语句那样不再执行余下的代码并退出整个循环。把例 5-8 修改一下，如例 5-9 所示。

【例 5-9】 continue 语句

```
for i in range(1,10):
    if (i==3):
        continue
    print(i)
```

运行结果：

```
1
2
4
5
6
7
8
9
```

例 5-9 中，通过循环从 1 开始依次输出每一个整数。但是当 i 等于 3 的时候，执行了 continue 语句，本次循环被终止，print 语句不会被执行。但与 break 语句不同的是：程序会继续下一次的循环，所以会输出数字 4～9。

需要注意的是，在有多层嵌套的情况下，无论是 continue 还是 break，都只会对语句所在的循环起作用，而不会对外层嵌套的循环起作用，如例 5-10 所示。

【例 5-10】 多层嵌套的作用域

```
for j in range(1,3):
    for i in range(1,5):
        if (i==3):
            break
        print(i)
```

运行结果：

```
1
2
1
2
```

可以看出，当 i 等于 3 的时候，只有内层循环被终止了，而外层循环不受影响。

5.5 循环嵌套

一层循环只能解决一些简单的循环问题，复杂问题如列表的冒泡排序有时需要多层循环嵌套来解决。嵌套循环是由一个外层循环和一个或多个内层循环构成。每次重复外层循环时，内层循环都被重新进入并且重新开始。

5.5.1 循环嵌套结构

for 循环嵌套语法(以两层嵌套为例)如下：

```
for 迭代变量 1 in 序列 1:
    for 迭代变量 2 in 序列 2:
        第二层循环语句块
            第一层循环语句块
```

相应地，while 循环嵌套语法如下：

```
while 布尔表达式 1:
    while 布尔表达式 2:
        第二层循环语句块
            第一层循环语句块
```

来看一个更加复杂的问题，计算 1 的阶乘加上 2 的阶乘，然后加上 3 的阶乘，直到累加到 10 的阶乘的问题。这里阶乘的相加需要使用循环来实现，那就是外层循环。而阶乘的计算实际上可以使用内层循环来实现，内层循环的结果成为外层循环的一次累加中的一项，如例 5-11 所示。

【例 5-11】 循环嵌套

```
sum=0
for i in range(1,11):
    item=1
    for j in range(1,i+1):
        item=item*j
    sum=sum+item
print(sum)
```

运行结果：

```
4037913
```

从例 5-11 可以看到，这个程序有两层嵌套。内层嵌套负责计算阶乘的结果，然后成为外层循环中的一项；外层循环则负责将每一项相加，最后得到总和。