第 5 章

循环结构程序设计

结构化程序由顺序结构、选择结构和循环结构组成。前面已经介绍了顺序结构和选择结构程序设计，本章主要介绍循环结构的程序设计。

循环结构是一种重复执行的程序结构。在许多实际问题中，需要对问题的一部分通过若干次、有规律的重复计算来实现。例如，求大量数据之和、迭代求根、递推法求解等，都要用到循环结构的程序设计。循环是计算机解题的一个重要特征，计算机运算速度快，最善于进行重复性的工作。

Python 语言提供了 while 语句和 for 语句来实现循环结构。

5.1 while 语句结构

5.1.1 while 语句

1. while 语句的一般格式

while 语句是当型循环，一般格式为：

```
while 条件表达式:
    循环体
```

功能：条件表达式描述循环的条件，循环体语句描述要反复执行的操作，称为循环体。while 语句执行时，先计算条件表达式的值，当条件表达式的值为真(非 0) 时，循环条件成立，执行循环体；当条件表达式的值为假(0) 时，循环条件不成立，退出循环，执行循环语句之后的下一条语句。其执行流程如图 5.1 所示。

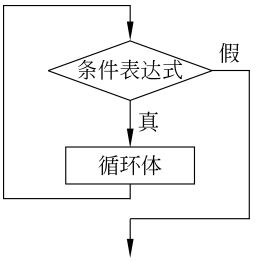

图 5.1 while 循环流程图

注意：

(1) 当循环体由多条语句构成时，必须用缩进对齐的方式组成一个语句块来分隔子句，否则会产生错误。

(2) 与 if 语句的语法类似，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。

(3) while 语句的条件表达式不需要用括号括起来，表达式后面必须有冒号。

(4) 如果表达式永远为真，循环将会无限地执行下去。因此，在循环体内必须要有修改表达式值的语句，使其值趋向 False，让循环趋于结束，避免出现无限循环。

2. 在 while 语句中使用 else 子句

while 语句中使用 else 子句的一般格式：

```
while 条件表达式;
    循环体
else:
    语句
```

Python 与其他大多数语言不同，可以在循环语句中使用 else 子句，即构成了 while…else 循环结构，else 中的语句会在循环正常执行完的情况下执行（不管是否执行循环体）。例如：

```
count=int(input())
while count<5:
    print(count,"is less han 5")
    count=count+1
else:
    print(count,"is not less than 5")
```

程序的一次运行结果如下：

```
8↙
8 is not less than 5
```

在该程序中，当输入 8 时，循环体一次都没有执行，退出循环时，执行 else 子句。

5.1.2 while 语句应用

【例 5.1】 求 $\sum_{n=1}^{100} n$。

分析：本例实际是求若干个数之和的累加问题。定义 sum 用于存放累加和，用 n 表示加数，用循环结构来求解，每循环一次累加一个整数值，整数的取值范围为 1～100。

程序如下：

```
sum,n=1,0
while n<=100:
    sum=sum+n
    n=n+1
print("1+2+3+…+100=",sum)
```

程序运行结果：

```
1+2+3+…+100=5050
```

说明：程序中变量 n 有两个作用，其一是作为循环计数变量，其二是作为每次被累加的整数值。循环体有两条语句，sum＝ sum ＋ n 实现累加；n＝n＋1 使加数 n 每次增 1，这是改变循环条件的语句，否则循环不能终止，成为“死循环”。循环条件是当 n 小于或等于 100 时，执行循环体，否则跳出循环，执行循环语句之后的下一条语句（print 语句）以输出计算结果。

思考：如果将循环体语句“s＝s＋n”和“n＝n＋1”互换位置，程序的运行结果如何？

对于 while 语句的用法，还需要注意以下几点。

（1）如果 while 后面表达式的值一开始就为假，则循环体一次也不会执行。例如：

```
a=0
b=0
while a>0:
   b=b+1
```

（2）循环体中的语句可以是任意类型的合法语句。

（3）遇到下列情况，退出 while 循环：

- 表达式不成立；
- 循环体内遇到 break、return 等语句。

【例 5.2】 从键盘上输入若干个数，求所有正数之和。当输入 0 或负数时，程序结束。

程序如下：

```
sum=0
x=int(input("请输入一个正整数(输入 0 或者负数时结束):"))
while x>=0:
   sum=sum+x
   x=int(input("请输入一个正整数(输入 0 或者负数时结束):"))
print("sum=",sum)
```

程序运行结果：

```
请输入一个正整数(输入 0 或者负数时结束): 13
请输入一个正整数(输入 0 或者负数时结束): 21
请输入一个正整数(输入 0 或者负数时结束): 5
请输入一个正整数(输入 0 或者负数时结束): 54
请输入一个正整数(输入 0 或者负数时结束): 0
sum = 92
```

【例 5.3】 输入一个正整数 x，如果 x 满足 0＜x＜99999，则输出 x 是几位数并输出 x 个位上的数字。

程序如下：

```
x=int(input("Please input x: "))
if x>=0 and x<99999:
   i=x
   n=0
   while i>0:
        i=i//10
        n=n+1
   a=x%10
   print("%d 是%d 位数,它的个位上数字是%d"%(x,n,a))
else:
     print("输入错误!")
```

程序运行结果：

```
Please input x: 12345
12345 是 5 位数,它的个位上数字是 5
```

再次运行程序，结果如下：

```
Please input x: -1
输入错误!
```

5.2 for 语句结构

5.2.1 for 语句

1. for 语句的一般格式

for 语句是循环控制结构中使用较广泛的一种循环控制语句，特别适合于循环次数确定的情况。其一般格式为：

```
for 目标变量 in 序列对象:
    循环体
```

for 语句的首行定义了目标变量和遍历的序列对象，后面是需要重复执行的语句块。语句块中的语句要向右缩进，且缩进量要一致。

注意：

(1) for 语句是通过遍历任意序列的元素来建立循环的，针对序列的每一个元素执行一次循环体。列表、字符串、元组都是序列，可以利用它们来创建循环。

(2) for 语句也支持一个可选的 else 块，其功能就像在 while 循环中一样，如果循环离开时没有遇到 break 语句，就会执行 else 块。也就是序列所有元素都被访问过了之后，执行 else 块。其一般格式为：

```
for 目标变量 in 序列对象:
    语句块
else:
      语句
```

2. rang 对象在 for 循环中的应用

在 Python 3.x 中，range()函数返回的是可迭代对象。Python 专门为 for 语句设计了迭代器的处理方法。range()函数的一般格式为：

```
range([start,]end[,step])
```

range()函数共有 3 个参数，start 和 step 是可选的，start 表示开始，默认值为 0，end 表示结束，step 表示每次跳跃的间距，默认值为 1。该函数功能是生成一个从 start 参数的值开始，到 end 参数的值结束(但不包括 end)的数字序列。

例如，传递一个参数的 range()函数：

```
>>> for i in range(5):
    print(i)
0
1
2
3
4
```

传递两个参数的 range()函数：

```
>>> for i in range(2,4):
    print(i)
2
3
```

传递三个参数的 range()函数：

```
>>> for i in range(2,20,3):
    print(i)
2
5
8
11
14
17
```

执行过程中首先对关键字 in 后的对象调用 iter()函数获得迭代器，然后调用 next()函数获得迭代器的元素，直到抛出 StopIteration 异常。

range()函数的工作方式类似于分片。它包含下限（上例中为 2），但不包含上限（上例中为 20）。如果希望下限为 0，则只可以提供上限，例如：

```
>>>range(10)
[0,1,2,3,4,5,6,7,8,9]
```

【例 5.4】 用 for 循环语句实现例 5.1。

程序如下：

```
sum=0
for i in range(101):
    sum=sum+i
print("1+2+3+…+100=",sum)
```

该例中采用 range()函数得到一个 0～100 的序列，变量 i 依次从序列中取值累加到 sum 变量中。

5.2.2 for 语句应用

【例 5.5】 判断 m 是否为素数。

一个自然数，若除了 1 和它本身外不能被其他整数整除，则称为素数。例如，2、3、5、7 等。根据定义，只需检测 m 能否被 2，3，4，…，m－1 整除，只要能被其中一个数整除，则 m 不是素数，否则就是素数。程序中设置标志量 flag，若 flag 为 0 时，则 m 不是素数；若 flag 为 1 时，则 m 是素数。

程序如下：

```
m=int(input("请输入要判断的正整数 m: "))
flag=1
for i in range(2,m):
   if  m%i==0:
      flag=0
      i=m                    #令 i 为 m,使 i<m 不成立,在不是素数时退出循环
if  flag==1:
    print("%d 是素数"%m)
else:
    print("%d 不是素数"%m)
```

程序运行结果：

```
请输入要判断的正整数 m: 11
11 是素数
```

再次运行程序，结果如下：

```
请输入要判断的正整数 m: 20
20 不是素数
```

【例 5.6】 已知四位数 3025 具有特殊性质：它的前两位数字 30 与后两位数字 25 之和是 55，而 55 的平方正好等于 3025。编写程序，列举出所有具有这种性质的四位数。

分析：采用列举的方法。将给定的四位数按前两位数、后两位数分别进行分离，验证分离后的两个两位数之和的平方是否等于分离前的那个四位数，若等于即打印输出。

程序如下：

```
print("满足条件的四位数分别是: ")
for i in range(1000,10000):
       a=i//100
       b=i%100
       if  (a+b)**2==i:
              print(i)
```

程序运行结果：

```
满足条件的四位数分别是:
2025
3025
9801
```

【例 5.7】 求 1～100 中能被 7 或 11 整除、但不能同时被 7 和 11 整除的所有整数并将它们输出。每行输出 10 个。

分析：列举出 1～100 的所有数据，根据题目中的条件对这些数据进行筛选。要控制每行输出 10 个，则应使用 count 变量，用于计数，每当有一个满足条件的数输出时，count 加 1，当 count 能整除 10 时，则换行。

程序如下：

```
print("满足条件的数分别是: ")
count=0
for i in range(1,100):
    if i%7==0 and i%11!=0 or i%11==0 and i%7!=0:
       print(i,end="   ")
       count=count+1
       if count%10==0:
          print("")
```

程序运行结果：

```
满足条件的数分别是:
7   11   14   21   22   28   33   35   42   44
49   55   56   63   66   70   84   88   91   98
99
```

5.3 循 环 嵌 套

如果一个循环结构的循环体又包括了另一个循环结构，就称为循环的嵌套。这种嵌套过程可以有很多种。一个循环外面仅包含一层循环称为两重循环；一个循环外面包围两层循环称为三重循环；一个循环外面包围多层循环称为多重循环。

循环语句 while 和 for 可以相互嵌套。在使用循环嵌套时，应注意以下几个问题：

(1) 外层循环和内层循环控制变量不能同名，以免造成混乱。

(2) 循环嵌套的缩进在逻辑上一定要注意，以保证逻辑上的正确性。

(3) 循环嵌套不能交叉，即在一个循环体内必须完整地包含另一个循环，如图 5.2 所示的循环嵌套都是合法的嵌套形式。

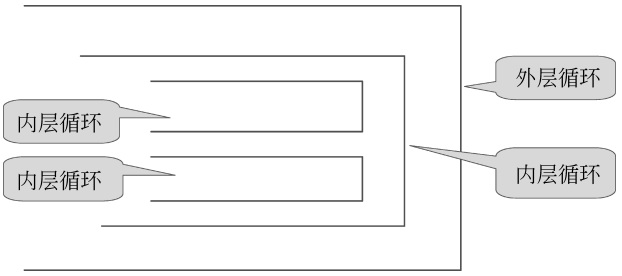

图 5.2 合法的循环嵌套形式

嵌套循环执行时，先由外层循环进入内层循环，并在内层循环终止后接着执行外层循环，再由外层循环进入内层循环中，当内层循环终止时，程序结束。

【例 5.8】 输出九九乘法表，格式如下。

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6   3*3=9
1*4=4  2*4=8   3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

程序如下：

```
for i in range(1, 10, 1):                #控制行
    for j in range(1, i+1, 1):           #控制列
        print("%d*%d=%2d  " %(j,i,i*j),end=" ")
    print("")                           #每行末尾的换行
```

【例 5.9】 找出所有的三位数，要求其各位数字的立方和正好等于这个三位数。例如，$153=1^3+5^3+3^3$ 就是这样的数。

分析：假设所求的三位数百位数字是 i，十位数字是 j，个位数字是 k，根据题目描述，应满足 $i^3+j^3+k^3=i*100+j*10+k$。

程序如下：

```
for i in range(1,10):
    for j in range(0,10):
        for k in range(0,10):
            if i**3+j**3+k**3==i*100+j*10+k:
                print("%d%d%d"%(i,j,k))
```

程序运行结果：

```
153
370
371
407
```

从程序中可以看出，三个 for 语句循环嵌套在一起，第二个 for 语句是前一个 for 语句的循环体，第三个 for 语句是第二个 for 语句的循环体，第三个 for 语句的循环体是 if 语句。

【例 5.10】 求 100～200 中的全部素数。

在例 5.5 中判断了给定的整数 m 是否为素数。本例要求 100～200 中的所有素数，可在外层加一层循环，用于提供要考查的整数 m=100，101，…，200。

程序如下：

```
print("100~200中的素数有:")
for m in range(100,201):
    flag=1
    for i in range(2,m):
        if  m%i==0:
            flag=0
            break
    if flag==1:
        print(m,end=" ")
```

程序运行结果：

```
100~200 中的素数有：
101  103  107  109  113  127  131  137  139  149  151  157  163  167  173
179  181  191  193  197  199
```

5.4 循环控制语句

有时需要在循环体中提前跳出循环，或者在某种条件满足时，不执行循环体中的某些语句而立即从头开始新一轮循环，这时就要用到循环控制语句 break、continue 和 pass 语句。

5.4.1 break 语句

break 语句用在循环体内，使所在循环立即中止，即跳出所在循环体，继续执行循环结构之后的语句。break 语句对循环执行过程的影响如图 5.3 所示。

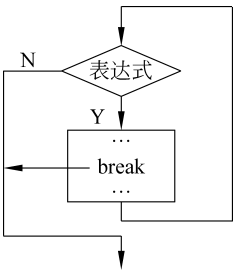

图 5.3 break 语句对循环执行过程的影响示意图

【例 5.11】 求两个整数 a 与 b 的最大公约数。

分析：找出 a 与 b 中较小的一个，则最大公约数必在 1 与这个较小整数的范围内。使用 for 语句，循环变量 i 从较小整数变化到 1。一旦循环控制变量 i 同时能被 a 与 b 整除，则 i 就是最大公约数，然后使用 break 语句强制退出循环。

程序如下：

```
m,n=eval(input("请输入两个整数："))
if m<n:
      min=m
else:
      min=n
for i in range(min,1,-1):
   if m%i==0 and n%i==0:
         print("最大公约数是：",i)
         break
```

程序运行结果：

```
请输入两个整数：156,18
最大公约数是：6
```

注意：

(1) break 语句只能用于由 while 和 for 语句构成的循环结构中。

(2) 在循环嵌套的情况下，break 语句只能终止并跳出包含它的最近一层循环体。