

第 5 章

循环结构

在结构化程序设计方法中,循环结构是最复杂的结构。通过循环结构与顺序结构、选择结构相结合,可以解决所有的计算机编程问题。本章从循环结构的基本概念入手,读者需要掌握 while 和 for-in 结构,理解嵌套式循环结构的使用方法,掌握控制循环结构的 break 和 continue 语句的使用,以及两者之间的区别;了解 Python 语言中异常捕捉的使用方法;掌握 turtle 库的常用函数使用方法,绘制简单图形。

5.1 循环结构概述

利用选择结构可以根据条件确定执行的流程,但如果要重复执行一些操作,如计算连续多个整数的和,例如求 1~1000 的累加和,选择结构就无法实现。这种情况就需要用程序设计语言中的循环结构。

循环结构是在满足一个指定的条件下,重复执行一条或多条语句的过程。Python 根据循环体触发条件的不同,分为无限循环 while 结构和遍历循环 for-in 结构。循环体执行与否,以及执行的次数由循环条件来决定。使用循环结构,可以减少源程序重复编写的工作量,降低程序的复杂度。

相同或不同的循环结构之间可以相互嵌套,也可以与选择结构嵌套使用,用来编写更为复杂的程序。为了优化程序以获得更高的效率和运行速度,在编写循环结构时,应尽量减少循环体内部不必要的计算,将与循环变量无关的代码尽可能放到循环体外。对于多重循环,尽可能减少内循环的计算。

5.2 while 循环结构

5.2.1 while 的基本结构

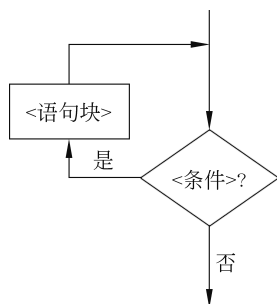
while 语句是一种“当型”循环结构,只有当条件满足时才执行循环体。while 循环语句的语法格式为:

```
while 表达式:  
    语句/语句块
```

执行时先计算表达式(条件)的值,当给定的条件成立,即表达式的结果为 True 时,执行语句/语句块(循环体);当到达循环语句的结束点时,转到 while 处继续判断表达式

的值是否为 True,若是,则继续执行循环体,如此周而复始,直到表达式的值为 False 退出

while 循环,转到循环语句的后继语句执行。其流程图如图 5-1 所示。



这里有几点要注意:

(1) while 语句是先判断再执行,所以循环体有可能一次也不执行。

(2) 循环体中必须包含能改变循环条件的语句,使循环趋于结束,否则,若表达式的结果始终是 True,会造成死循环。

(3) 要注意语句序列的对齐,while 语句只执行其后的一条或一组同一层次的语句。

图 5.1 while 循环的流程图

5.2.2 while 的使用示例

【例 5-1】 计算 $1+2+\cdots+100$ 的累加和。

编写程序如下:

```

1  sum=0
2  i=1
3  while i<=100:
4      sum+=i
5      i+=1
6  print("1+2+...+100={:d}".format(sum))
7  程序运行结果为:
8  1+2+...+100=5050
  
```

本例的 while 循环体包括两条语句 $\text{sum}+=i$ 和 $i+=1$,如果漏写 $i+=1$,则循环变量 i 始终等于 1,条件表达式 $i\leq 100$ 始终满足,将造成死循环。另外,如果 $i+=1$ 无缩进而与 while 对齐,则表示它不在 while 语句的循环体中,这种情况同样会造成死循环。因此,在使用 while 语句时要注意条件表达式是否发生变化。

【例 5-2】 求两个正整数的最大公约数和最小公倍数。

分析: 计算两个正整数的最大公约数可以用辗转相除法。辗转相除法的具体描述如下:

① 判断 x 除以 y 的余数 r 是否为 0。若 r 为 0,则 y 是 x 、 y 的最大公约数,继续执行后续操作;否则 $y\rightarrow x, r\rightarrow y$,重复执行第①步。

② 输出(或返回) y 。

③ 两数乘积除以最大公约数即可得到最小公倍数。

编写程序如下:

```

1  x=eval(input('输入第一个数:'))
2  y=eval(input('输入第二个数:'))
3  z=x*y
  
```

```
4  if x<y:
5      x,y=y,x
6  while x%y!=0:
7      r=x%y
8      x=y
9      y=r
10 print('最大公约数=',y)
11 print('最小公倍数=',z//y)
```

运行结果：

输入第一个数:14

输入第二个数:6

最大公约数=2

最小公倍数=42

【例 5-3】 利用如下公式计算 π 。当通项的绝对值小于或等于 10^{-8} 时停止计算。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

编写程序如下：

```
1  import math
2  x,s=1,0
3  sign=1
4  k=1
5  while math.fabs(x)>1e-8:
6      s+=x
7      k+=2
8      sign*=-1
9      x=sign/k
10 s*=4
11 print('pi={:.15f}'.format(s))
```

运行结果：

pi=3.141592633590251

math 标准库中的 pi 值等于 3.141592653589793, 可以看到, 本例中利用公式计算的结果在误差的要求范围内, 与标准库中的标准值基本一致。

5.3 for-in 循环结构

Python 中的 for-in 循环一般适用于已知循环次数的场合, 尤其适用于枚举、遍历序列或迭代对象中的元素。Python 中的 for-in 循环语句功能更加强大, 编程时一般优先考虑使用 for-in 循环。

5.3.1 for-in 的基本结构

for-in 循环语句用于遍历可迭代对象集合中的元素,并对集合中的每个元素执行一次相关的循环语句块,当集合中所有元素完成迭代后,结束循环体,继续执行后面的语句。在 Python 中 for-in 语句语法形式为:

```
for <循环变量> in 序列或可迭代对象:
    语句/语句块
[else:
    else 子句代码块]
```

例如:

```
for num in ['11','22','33']:
    print(num)
```

注意: 如果循环是因为 break 结束的,就不执行 else 中的代码,具体请参考 5.5.3 节。

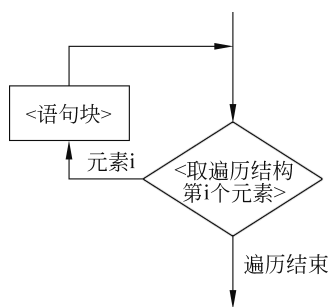


图 5.2 for-in 语句流程图

for-in 循环结构的流程图如图 5.2 所示。

可迭代对象指可以按次序迭代(循环)的对象,一次返回一个元素,适用于循环。Python 包括以下几种可迭代对象。

- (1) 序列: 字符串(str)、列表(list)、元组(tuple)。
- (2) 迭代器对象(iterator)。
- (3) 生成器函数(generator)。
- (4) enumerate()函数产生字典的键和文件的行等。

迭代器是一个对象,表示可迭代的数据集合,包括方法__iter__()和__next__(),可实现迭代功能。

生成器是一个函数,使用 yield 语句,每次产生一个值,也可以用于循环迭代。执行时变量取可迭代对象中的一个值,执行语句序列,再取下一个值,继续执行语句序列。

例如:

```
for i in [1, 2, 3]:
    print(i)
```

运行结果如下:

```
1
2
3
```

又如:

```
>>>for item in enumerate(['a','b','c']):
    print(item)
```

运行结果如下：

```
(0, 'a')
(1, 'b')
(2, 'c')
```

其中,前一示例中的 `[1,2,3]` 是一个列表,产生的是一个包含 3 个元素的序列,后一示例中的 `enumerate(['a','b','c'])` 产生的是一个迭代器,对于每一个产生的元素执行一次 `print()` 函数。序列中使用最多的是列表,迭代时为什么不直接使用列表(或其他序列对象)而要用迭代器呢? 那是因为若使用列表,在每次取值时会一次性获取所有值,如果值较多,会占用较多的内存;迭代器则是一个接一个地计算值,在计算一个值时只获取一个值,占内存少。

使用迭代器还有很多其他优点,如使代码更通用简单,更为优雅,有兴趣的读者可以继续深入挖掘。`for-in` 语句主要用来迭代,但迭代的方式有多种,可以用序列迭代,也可以用序列索引迭代,还可以用迭代器迭代。另外,如字典的键和文件的行等可迭代对象也有自己的迭代方式。

`range()` 函数是一个可迭代对象,产生不可变的整数序列,常常用在 `for` 循环语句中,为 `for` 循环提供所需的数字容器。`range()` 函数返回的结果是一个整数序列的对象,而不是列表。

例如：

```
>>>ls=range(0,24,5)
>>>list(ls)
[0, 5, 10, 15, 20]
>>>ls1=range(0,-10,-2)
>>>list(ls1)
[0, -2, -4, -6, -8]
>>>for i in range(1,5):
    print(i*i)
1
4
9
16
```

例如,求 $1+2+3+\cdots+100$ 累加和的代码如下：

```
sum=0
for i in range(0,101,1):
    sum=sum+x
    print('累加和是:', sum)
```

5.3.2 for-in 的使用示例

【例 5-4】 求斐波那契(Fibonacci)数列的前 20 项。斐波那契数列定义如下：

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_i = F_{i-1} + F_{i-2} \end{cases}$$

分析：斐波那契数列的前两项分别是 0 和 1,从第 3 项开始,后项都是前两项之和。

斐波那契数列的每一项可以用简单的整型变量表示、迭代并逐步输出,也可以用列表将数列中的所有值都保存下来,最后统一输出该列表。这里我们使用列表来求解。

编写程序如下：

```
1  f = [0] * 20                                #生成一个元素是 20 个 0 的列表
2  f[0], f[1] = 0, 1
3  for i in range(2, 20):
4      f[i] = f[i-1] + f[i-2]
5  print(f)
```

运行结果：

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
```

再以一个简单的例子,来对比一下同一个问题怎样用序列索引迭代和序列迭代来求解。

【例 5-5】 竖着打印"Hello World!"字符串。

编写程序如下：

方法一

```
1  x = 'Hello World!'
2  for i in x:
3      print(i)
```

方法二

```
1  x = 'Hello World!'
2  for i in range(len(x)):
3      print(x[i])
```

【例 5-6】 编程求 n! 的值。

分析：此题目为经典的累计求和变形问题。n 从键盘输入,t 用于存放累乘结果,t=1*2*3*4*...*n,t 的初值为 1,控制循环的变量 i 的初值为 1,终值为 n,当 i=n+1 时结束循环。

编写程序如下：

```
1  n = int(input("请输入要求阶乘的 n 值:"))
2  t = 1
3  for i in range(1, n+1):
4      t = t * i
5  print(t)
```

运行结果：

请输入要求阶乘的 n 值:5

120

拓展：如果把此例改为求 $1!+2!+3!+\cdots+n!$ ，如何实现？

参考实现代码如下：

```
n=int(input("请输入 n 的值:"))
t=1          # 累乘初值
s=0          # 累加和初值
for i in range(1,n+1):
    t=t*i
    s=s+t
print(t,s)
```

【例 5-7】 使用 for 循环语句编写程序，打印由★、■符号组成的一座房子。程序运行效果如图 5.3 所示。

分析：需要 2 个循环语句分别打印房子的顶部和底部。使用语句 $i * "★"$ 打印 i 个★，语句 $i * "■"$ 打印 i 个■，使用方法 $\text{center}(20-i)$ 控制打印位置。

编写程序如下：

```
1  for i in range(1,5):
2      print((i * "★").center(20-i))
3  for j in range(1,5):
4      print((4 * "■").center(20))
```



图 5.3 例 5-7 运行效果

【例 5-8】 输出“水仙花数”。所谓水仙花数是指一个 3 位的十进制数，其各位数字的立方和等于该数本身。例如，153 是水仙花数，因为 $1^3+5^3+3^3=153$ 。

编写程序如下：

方法一，序列解包法。

```
1  for i in range(100, 1000):
2      bai, shi, ge = map(int, str(i))
3      if ge**3 + shi**3 + bai**3 == i:
4          print(i)
```

方法二，函数式编程。

```
1  for num in range(100, 1000):
2      r = map(lambda x:int(x)**3, str(num))
3      if sum(r) == num:
4          print(num)
```

运行结果：

```
153
370
371
407
```

5.4 嵌套循环

一个循环结构可以包含一个或多个循环结构,这种一个循环结构的循环体内又包含一个或多个循环结构的情况,称为嵌套循环,也称为多重循环,其嵌套层数视问题复杂程度而定。while 语句和 for-in 语句可以嵌套自身语句结构,也可以相互嵌套,可以呈现各种复杂的形式。下面来看一个经典的嵌套循环示例。

【例 5-9】 编写程序,统计一元人民币换成一分、两分和五分的所有兑换方案个数。

分析：这个问题可以用三重循环直接解决,也可以用两重循环,且两重循环效率更高。

编写程序如下：

方法一,利用 for-in 结构。

```
1  i, j, k = 0, 0, 0          # i, j, k 分别代表五分、两分和一分的数量
2  count = 0
3  for i in range(21):
4      for j in range(51):
5          k = 100 - 5 * i - 2 * j
6          if k >= 0:
7              count += 1
8  print('count={:d}'.format(count))
```

方法二,利用 while 结构。

```
1  i, j, k = 0, 0, 0          # i, j, k 分别代表五分、两分和一分的数量
2  count = 0
3  while i <= 20:
4      j = 0
5      while j <= 50:
6          k = 100 - 5 * i - 2 * j
7          if k >= 0:
8              count += 1
9              j += 1
10     i += 1
11 print('count={:d}'.format(count))
```

运行结果：

```
count=541
```


【例 5-10】 从两个列表中分别选出一个元素,组成一个元组放到一个新列表中,要求新列表中包含所有的组合。

编写程序如下:

```
1 result=[]
2 pdlList=['C++','Java','Python']
3 creditList=[2,3,4]
4 for pdl in pdlList:
5     for credit in creditList:
6         result.append((pdl,credit))
7 print(result)
```

运行结果:

```
[('C++', 2), ('C++', 3), ('C++', 4), ('Java', 2), ('Java', 3), ('Java', 4),
 ('Python', 2), ('Python', 3), ('Python', 4)]
```

5.5 break 与 continue 语句

正常来说,执行到条件为假,或迭代取不到值时,循环将结束,但有时需要提前终止循环或提前结束本轮循环的执行(并非终止循环语句的执行),这就需要用到 break 语句和 continue 语句。

5.5.1 break 语句

break 语句用来终止当前循环,转而执行循环之后的语句。例如,对于如下程序:

```
s=0
i=1
while i<10:
    s+=i
    if s>10:
        break
    i+=1
print('i={0:d},sum={1:d}'.format(i,s))
```

程序运行结果为:

```
i =5, sum=15
```

从程序运行结果可以看到,s 是 $1+2+3+\cdots$ 不断累加的和,当 i 等于 5 时,s 第一次大于 10,执行 break 语句跳出 while 循环语句,继续执行 print 函数调用语句,此时 s 的值为 15,i 的值为 5。

上例中语句“while i<10:”意义不大,这种情况常常会将此条语句替换成另一种在 Python 中常与 break 语句一起使用的循环控制语句 while True。因此,上面程序中的核

心代码可替换成如下形式：

```
s=0
i=1
while True:
    s+=i
    if s>10:
        break
    i+=1
print('i={0:d},sum={1:d}'.format(i,s))
```

break 语句用来跳出其所在的循环。如果是一个双重循环,那么在内层循环有一个 break 语句时将如何执行? 是否会跳出两重循环? 来看一个简单的例子。

```
for i in range(11):
    for j in range(11):
        if i * j >= 50:
            break
print(i,j)
```

上述代码的运行结果是：

```
10 5
```

当 i 和 j 分别等于 5 和 10 时, i 和 j 的乘积第一次等于 50, 如果 break 语句可以跳出双重循环, 则输出结果应该是“5 10”, 而程序的实际输出结果是“10 5”, 这表明 break 只能跳出紧包层, 即 break 所在层次的循环。在使用嵌套循环时要注意这类问题。

【例 5-11】 输出 2~100 的素数, 每行显示 5 个。

分析：如果一个正整数 n 只能被 1 和 n 自身整除, 则称 n 为素数(prime)。例如, 13 是素数, 而 6 不是素数。因此, 可以将素数判断算法设计为: 若 n 不能被 2~n-1 的任一个整数整除, 则 n 是素数, 否则 n 不是素数; 如果发现能被某个整数整除了, 可立即停止判断 n 是否能被范围内其他整数整除。进一步分析, 如果确定 n 不能被 2~n/2 的所有整数整除, 就可以断定 n 是素数; 接着还可以证明, 如果 n 不能被 $2\sim\sqrt{n}$ 的所有整数整除, 则 n 是素数。

编写程序如下：

```
1  from math import sqrt
2  j=2
3  count=0
4  while j<=100:
5      i=2
6      k=int(sqrt(j))                #sqrt 函数用来求平方根
7      while i<=k:
8          if j%i==0:
9              break
```