第3章 chapter 3

程序流程控制

Python 程序中的语句默认是按照书写顺序依次执行的，语句之间的此种结构称为顺序结构。在顺序结构中，各语句是按自上而下的顺序执行的，执行完上一条语句就自动执行下一条语句，语句之间的执行是不做任何判断的，无条件的。但仅有顺序结构还不够，因为有时需要根据特定的情况，有选择地执行某些语句，这时就需要一种选择结构的语句。另外，有时还需要在给定条件下重复执行某些语句，这时称这些语句是循环结构的。有了顺序、选择和循环这 3 种基本的结构，就能构建任意复杂的程序。


布尔表达式

3.1 布尔表达式

选择结构和循环结构都会使用布尔表达式作为选择和循环的条件。布尔表达式是由关系运算符和逻辑运算符按一定语法规则组成的式子。关系运算符有＜(小于)、＜＝(小于或等于)、＝＝(等于)、＞(大于)、＞＝(大于或等于)、!＝(不等于)。逻辑运算符有 and、or、not。

布尔表达式的值只有两个：True 和 False。在 Python 中，False、None、0、" "、()、[]、{}作为布尔表达式时，会被解释器看作假(False)。换句话说，也就是特殊值 False 和 None、所有类型的数字 0(包括浮点型、长整型和其他类型)、空序列(例如空字符串、元组和列表)以及空的字典都被解释为假。其他的一切都被解释为真，包括特殊值 True。

True 和 False 属于布尔数据类型(bool)，它们都是保留字，不能在程序中被当作标识符。一个布尔变量可以代表 True 或 False 中的一个。bool()函数可以用来转换其他值(与 list、str 以及 tuple 一样)。

```
>>>type(True)
<class 'bool'>
>>>bool('Practice makes perfect.')        #转换为布尔值
True
>>>bool(101)                              #转换为布尔值
True
>>>bool('')                               #转换为布尔值
False
```

```
>>>print(bool(4))
True
```


选择结构

3.2 选择结构

选择结构通过判断某些特定条件是否满足来决定下一步执行哪些语句。Python 有多种选择语句类型：单向 if 语句、双向 if-else 语句、嵌套 if 语句、多向 if-elif-else 语句以及条件表达式。

3.2.1 单向 if 语句

if 语句用来判定给出的条件是否满足，然后根据判断的结果（即真或假）决定是否执行给定的操作。if 语句是一种单选结构，它选择的是做与不做。它由 3 部分组成：关键字 if 本身、测试条件真假的布尔表达式和表达式结果为真时要执行的代码。if 语句的语法格式如下：

```
if 布尔表达式:
  语句块
```

if 语句的流程图如图 3-1 所示。

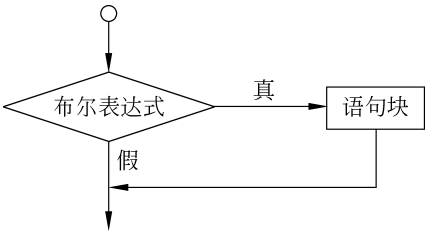

图 3-1 if 语句的流程图

注意：单向 if 语句的语句块只有当表达式的值为真（即非零）时，才会执行；否则，程序就会直接跳过这个语句块，去执行紧跟在这个语句块之后的语句。这里的语句块，既可以包含多条语句，也可以只有一条语句。当语句块由多条语句组成时，要有统一的缩进形式，相对于 if 向右至少缩进一个空格，否则就会出现逻辑错误，即语法检查没错，但是结果却非预期。

【例 3-1】 输入一个整数，如果这个整数是 5 的倍数，输出“输入的整数是 5 的倍数”，如果这个数是 2 的倍数，输出“输入的整数是 2 的倍数”。（3-1.py）

说明：求解例 3-1 的程序文件将被命名为 3-1.py，后面章节会多次使用这种表示方式，不再一一赘述。

3-1.py 程序文件：

```
num=eval(input('输入一个整数: '))   #eval(str)将字符串 str 当成有效的表达式来求值
if num%5==0:
    print('输入的整数%d 是 5 的倍数'%num)
if num%2==0:
    print('输入的整数%d 是 2 的倍数'%num)
```

3-1.py 在 IDLE 中运行的结果如图 3-2 所示。

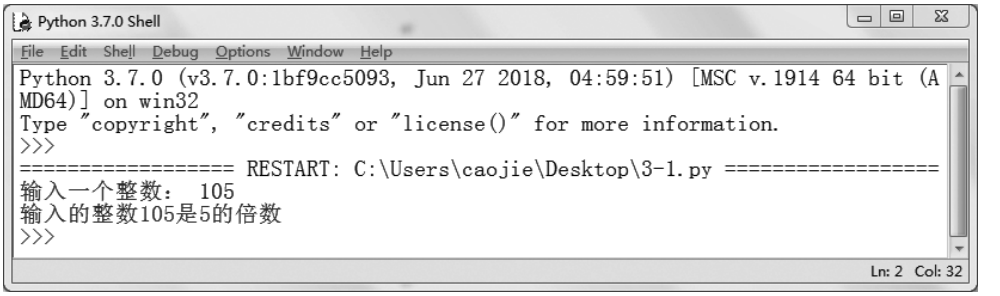

图 3-2 运行 3-1.py 的结果

3.2.2 双向 if-else 语句

3.2.1 节中的 if 语句是一种单选结构，如果表达式为真（即表达式的值非零），就执行指定的操作；否则就会跳过该操作。所以，if 语句选择的是做与不做的问题。if-else 语句是一种双选结构，根据表达式是真还是假来决定执行哪些语句，它选择的不是做与不做的问题，而是在两种备选操作中选择哪一个操作的问题。if-else 语句由 5 部分组成：关键字 if、测试条件真假的布尔表达式、表达式结果为真时要执行的语句块 1，以及关键字 else 和表达式结果为假时要执行的语句块 2。if-else 语句的语法格式如下：

```
if 布尔表达式:
    语句块 1
else:
    语句块 2
```

if-else 语句的流程示意图如图 3-3 所示。

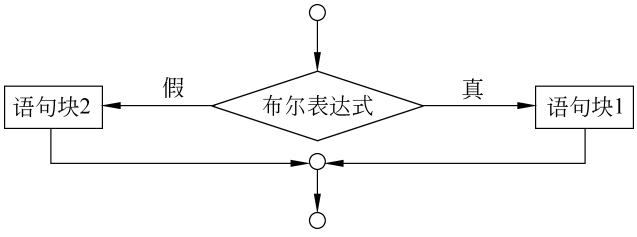

图 3-3 if-else 语句的流程示意图

从 if-else 语句的流程示意图中可以看出：当表达式为真（即表达式的值非零）时，执行语句块 1；当表达式为假（即表达式的值为零）时，执行语句块 2。if-else 语句不论表达式取何值，它总要在两个语句块中选择一个语句块执行，双向结构的称谓由此而来。

【例 3-2】 编写一个两位数减法的程序，程序随机产生两个两位数，然后向学生提问这两个数相减的结果是什么，在回答问题之后，程序会显示一条信息表明答案是否正确。(3-2.py)

```
import random
num1 =random.randint(10, 100)
num2 =random.randint(10, 100)
if num1 <num2:
    num1, num2 =num2, num1
answer=int(input(str(num1) +'-' +str(num2) +'=' +' ? '))
if num1-num2==answer:
    print('你是正确的!')
else:
    print('你的答案是错误的.')
    print(str(num1), '-' , str(num2), '=' , str(num1-num2))
```

在 cmd 命令窗口运行 3-2.py 的结果如图 3-4 所示。

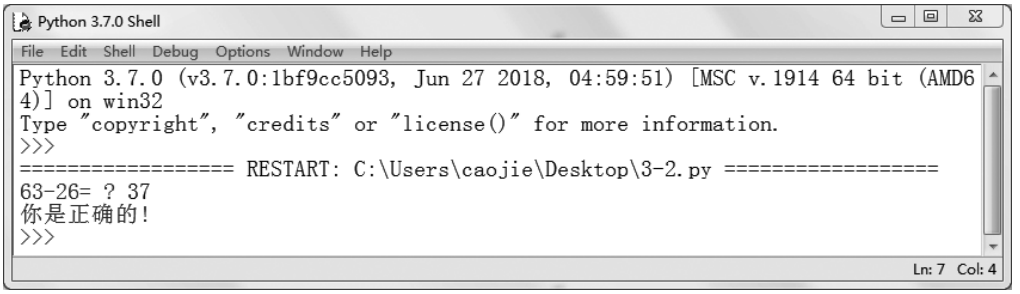

图 3-4 运行 3-2.py 的结果

注意：

(1) 每个条件后面要使用冒号(:)，表示接下来是满足条件后要执行的语句块。

(2) 使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。

3.2.3 嵌套 if 语句和多向 if-elif-else 语句

将一个 if 语句放在另一个 if 语句中就形成了一个嵌套 if 语句。

有时人们需要在多组操作中选择一组执行，这时就会用到多选结构，对于 Python 语言来说就是 if-elif-else 语句。该语句可以利用一系列布尔表达式进行检查，并在某个表达式为真的情况下执行相应的代码。需要注意的是，虽然 if-elif-else 语句的备选操作较多，但是有且只有一组操作被执行，if-elif-else 语句的语法格式如下：

```
if 布尔表达式 1:
    语句块 1
elif 布尔表达式 2:
```

```
    语句块 2
⋮
elif 布尔表达式 m:
    语句块 m
else:
    语句块 n
```

其中,关键字 elif 是 else if 的缩写。

【例 3-3】 利用多分支选择结构将成绩从百分制变换到等级制。(score_degree.py)

```
score=float(input('请输入一个分数:'))
if score>=90.0:
      grade='A'
elif score>=80.0:
      grade='B'
elif score>=70.0:
      grade='C'
elif score>=60.0:
      grade='D'
else:
      grade='F'
print(grade)
```

在 cmd 命令窗口运行 score_degree.py 的结果如图 3-5 所示。

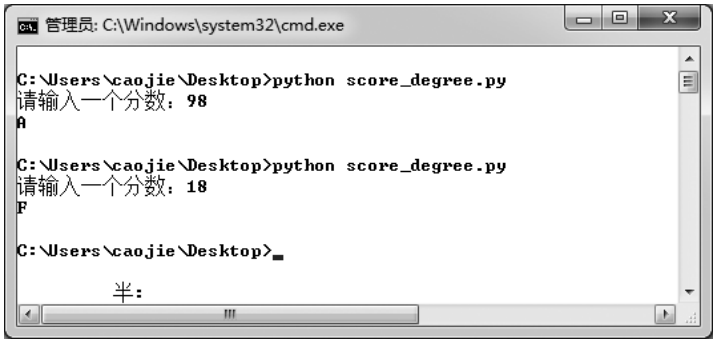

图 3-5 运行 score_degree.py 的结果

例 3-3 中 if-elif-else 语句的执行过程如图 3-6 所示。首先测试第一个条件(score>=90.0),如果表达式的值为 True,那么 grade='A'。如果表达式的值为 False,就测试第二个条件(score>=80.0),若表达式的值为 True,那么 grade='B'。以此类推,如果所有的条件的值都是 False,那么 grade='F'。

注意:一个条件只有在这个条件之前的所有条件都变成 False 之后才会被测试。

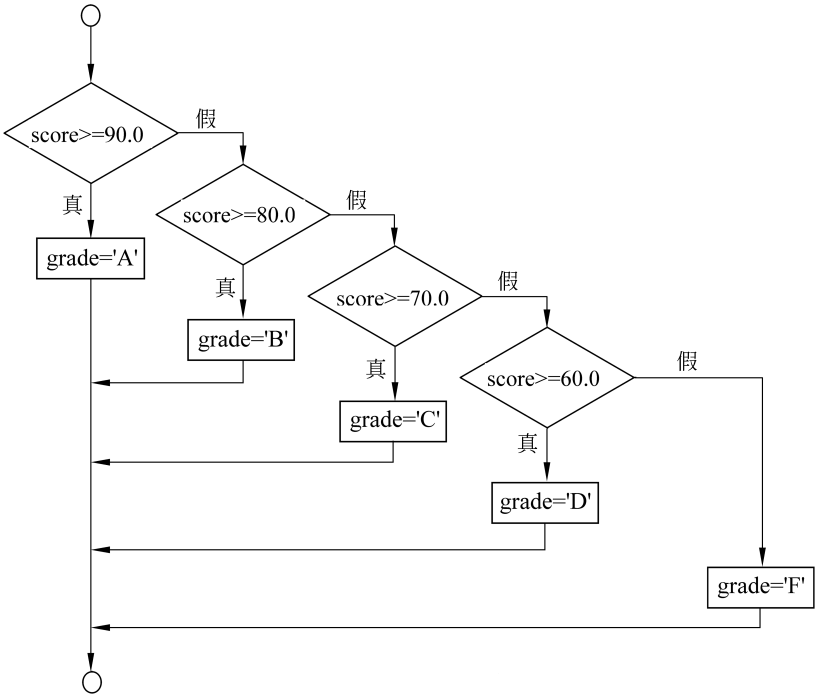

图 3-6　例 3-3 if-elif-else 语句的执行过程

3.3　条件表达式

有时人们可能想给一个变量赋值，但又受一些条件的限制。例如，下面的语句在 x 大于 0 时将 1 赋给 y，在 x 小于或等于 0 时将－1 赋给 y。

```
>>>x=2
>>>if x>0:
    y=1
else:
    y=-1
>>>print(y)
1
```

在 Python 中，还可以使用条件表达式 y＝1 if x＞0 else －1 来获取同样的结果。

```
>>>x=2
>>>y=1 if x>0 else -1
>>>print(y)
1
```

显然，对于上述问题使用条件表达式更简洁，用一行代码就可以完成所有选择的赋值操作。

条件表达式的语法结构如下：

```
表达式 1 if 布尔表达式 else 表达式 2
```

如果布尔表达式为真，那么这个条件表达式的结果就是表达式 1；否则，这个结果就是表达式 2。

若想将变量 number1 和 number2 中较大的值赋给 max，可以使用下面的条件表达式简洁地完成。

```
max=number1 if number1>number2 else number2
```

判断一个数 number 是偶数还是奇数，并在是偶数时输出“number 这个数是偶数”，是奇数时输出“number 这个数是奇数”，可用一个条件表达式简单地编写一条语句来实现。

```
print(' number 这个数是偶数' if number%2==0 else ' number 这个数是奇数')
```


while 循环结构

3.4 while 循环结构

while 语句用于在某条件下循环执行某段程序，以处理需要重复处理的任务。while 语句的语法格式如下：

```
while 循环继续条件:
    循环体
```

while 循环流程图如图 3-7 所示。循环体可以是一个单一的语句或一组具有统一缩进的语句。每个循环都包含一个循环继续条件，即控制循环执行的布尔表达式，每次都计算该布尔表达式的值，如果它的计算结果为真，则执行循环体；否则，终止整个循环并将程序控制权转移到 while 循环后的语句。while 循环是一种条件控制循环，它根据一个条件的真假来控制循环。使用 while 语句通常会遇到两种类型的问题：一种是循环次数事先确定的问题；另一种是循环次数事先不确定的问题。

显示“Python is very fun!”100 次的 while 循环的流程图如图 3-8 所示，循环继续条件是 count＜100，该循环的循环体包含两条语句：

```
print('Python is very fun!')
count=count+1
```

【例 3-4】 计算 $1+2+3+\cdots+100$，即 $\sum_{i=1}^{100} i$。 (3-4.py)

问题分析如下。

(1) 这是一个累计求和的问题，需要先后将 1～100 这 100 个数相加，需要重复进行 100 次加法运算，这可使用 while 循环语句来实现，重复执行循环体 100 次，每次加一个数。

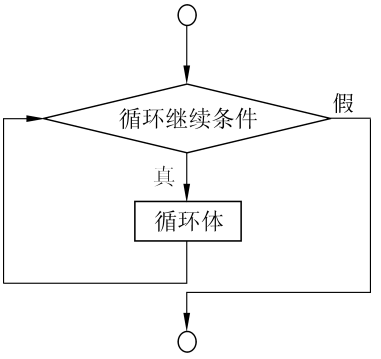

图 3-7　while 循环流程图

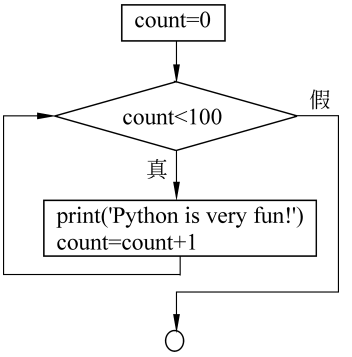

图 3-8　while 循环显示“Python is very fun!”100 次

（2）可以发现每次累加的数是有规律的，后一个加数比前一个加数多 1，这样在加完上一个加数 i 后，使 i 加 1 就可以得到下一个数。

```
n=100
sum =0          #定义变量 sum 的初始值为 0
i =1            #定义变量 i 的初始值为 1
while i <=n:
    sum =sum +i
    i =i+1
print("1～%d 之和为%d" %(n,sum))
```

在 IDLE 中，运行 3-4.py 的结果如下。

```
1～100 之和为 5050
```

循环体易被错误地写成如下：

```
n=100
sum =0
i =1
while i <=n:
    sum =sum +i
i =i +1
print("1～%d 之和为%d" %(n, sum))
```

注意整个循环体必须被内缩进到循环内部，这里的语句 i＝i＋1 不在循环体里，这是一个无限循环，因为 i 一直是 1，而 i＜＝n 总是为真。

注意：确保**循环继续条件**最终变成 False 以便结束循环。编写循环程序时，常见的程序设计错误是循环继续条件总是为 True，循环变成无限循环。如果一个程序运行后，经过相当长的时间也没有结束，那么它可能就是一个无限循环。如果是通过命令行运行这个程序，可按 Ctrl＋C 键来停止它。无限循环在服务器上响应客户端的实时请求时非常有用。

【例 3-5】 求 1～100 能被 5 整除，但不能同时被 3 整除的所有整数。（3-5.py）

问题分析如下。

(1) 本题需要对 1～100 范围内的所有数一一进行判断。

(2) 本题的循环次数是 100 次。

(3) 在每次循环过程中需要用 if 语句进行条件判断。

本整除问题的框图如图 3-9 所示。

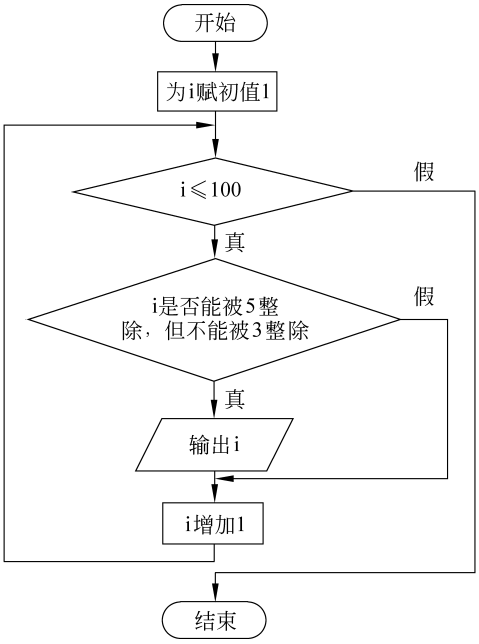

图 3-9 整除问题的框图

```
i=1                                                    #i 既是循环变量,又是被判断的数
print("1～100 能被 5 整除,但不能同时被 3 整除的所有数是")
while i<=100:
    if i%5==0 and i%3!=0:                              #判断本次的 i 是否满足条件
        print(i,end=' ')                               #输出满足条件的 i
    i=i+1                                              #每次循环 i 加 1
```

3-5.py 在 IDLE 中运行的结果如下：

```
1～100 能被 5 整除,但不能同时被 3 整除的所有数是
5 10 20 25 35 40 50 55 65 70 80 85 95 100
```

【例 3-6】 打印出所有的“水仙花数”。“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 是一个“水仙花数”,因为 $153=1^3+5^3+3^3$。(3-6.py)

问题分析如下。

(1) “水仙花数”是一个 3 位的十进制数,因而本题需要对 100～999 范围内的每个数判断是否是“水仙花数”。

（2）每次需要判断的数是有规律的，后一个数比前一个数多 1，这样在判断完上一个数 i 后，使 i 加 1 就可以得到下一个数，因而变量 i 既是循环变量，又是被判断的数。

```
i =100                                          #给变量 i 赋初始值
print('所有的水仙花数是', end='')
while i <=999:                                  #循环继续的条件
   c =i%10                                      #获得个位数
   b =i//10%10                                  #获得十位数
   a =i//100                                    #获得百位数
   if a* *3+b* *3+c* *3==i:                     #判断是否“水仙花数”
      print(i,end=' ')                          #输出水仙花数
   i =i+1                                       #变量 i 增加 1
```

在 IDLE 中，3-6.py 运行的结果如下：

```
所有的水仙花数是 153 370 371 407
```

【例 3-7】 将一个列表中的数进行奇、偶分类，并分别输出所有的奇数和偶数。（3-7.py）

```
numbers=[1,2,4,6,7,8,9,10,13,14,17,21,26,29]
even_number=[]
odd_number=[]
while len(numbers) >0:
    number=numbers.pop()
    if(number %2 ==0):
        even_number.append(number)
    else:
        odd_number.append(number)
print('列表中的偶数有', even_number)
print('列表中的奇数有', odd_number)
```

在 IDLE 中，3-7.py 运行的结果如下：

```
列表中的偶数有 [26, 14, 10, 8, 6, 4, 2]
列表中的奇数有 [29, 21, 17, 13, 9, 7, 1]
```

【例 3-8】 猜数字。随机生成一个 0～100 的数字，编写程序提示用户输入数字，直到输入的数和随机生成的数相同，对于用户输入的每个数字，程序显示输入的数字是过大、过小还是正确。（guess_number.py）

```
import random
number=random.randint(0,100)
print('请猜一个 0～100 的数字')
guess=-1
while guess!=number:
    guess=int(input('输入你猜测的数字:'))
    if guess==number:
```

```
        print('恭喜你!你猜对了,这个数字就是', number)
    elif guess>number:
        print('你猜的数字大了!')
    elif guess<number:
        print('你猜的数字小了!')
```

guess_number.py 在 IDLE 中运行的结果如下：

```
请猜一个 0～100 的数字
输入你猜测的数字:34
你猜的数字大了!
输入你猜测的数字:25
你猜的数字大了!
输入你猜测的数字:15
你猜的数字大了!
输入你猜测的数字:10
你猜的数字小了!
输入你猜测的数字:12
你猜的数字小了!
输入你猜测的数字:14
你猜的数字大了!
输入你猜测的数字:13
恭喜你!你猜对了,这个数字就是 13
```

3.5 循环控制策略

要想编写一个能够正确工作的 while 循环,需要考虑以下 3 步。

第 1 步：确认需要循环的循环体语句,即确定重复执行的语句序列。

第 2 步：把循环体语句放在循环内。

第 3 步：编写循环继续条件,并添加合适的语句以控制循环能在有限步内结束,即能使循环继续条件的值变成 False。

3.5.1 交互式循环

交互式循环是无限循环的一种,允许用户通过交互的方式重复循环体的执行,直到用户输入特定的值结束循环。

【例 3-9】 编写 100 以内加法训练程序,在学生结束测验后能报告正确答案的个数和测验所用的时间,能让用户自己决定随时结束测验。(3-9.py)

```
import random
import time
correctCount=0                                    #记录正确答对数
count=0                                           #记录回答的问题数
```

```
continueLoop='y'                                    #让用户来决定是否继续答题
startTime=time.time()                               #记录开始时间
while continueLoop=='y':
    number1=random.randint(0,50)
    number2=random.randint(0,50)
    answer=eval(input(str(number1)+'+'+str(number2)+'='+'?'))
    if number1+number2==answer:
        print('你的回答是正确的!')
        correctCount+=1
    else:
        print('你的回答是错误的。')
        print(number1,'+',number2,'=',number1+number2)
    count+=1
    continueLoop=input('输入 y 继续答题,输入 n 退出答题:')
endTime=time.time()                                 #记录结束时间
testTime=int(endTime-startTime)
print("  正确率:%.2f%%\n 测验用时:%d 秒" %((correctCount/count) * 100, testTime))
```

在 IDLE 中，3-9.py 运行的结果如下：

```
2+36=?38
你的回答是正确的!
输入 y 继续答题,输入 n 退出答题:y
8+28=?38
你的回答是错误的。
输入 y 继续答题,输入 n 退出答题:n
```

3.5.2 哨兵式循环

另一个控制循环结束的技术是指派一个特殊的输入值，这个值称为哨兵值，它表明输入的结束。哨兵式循环是指执行循环语句直到遇到哨兵值，循环体语句才终止执行的循环结构设计方法。

哨兵式循环是求平均数的较好方案，思路如下。

(1) 设定一个哨兵值作为循环终止的标志。

(2) 任何值都可以作为哨兵，但要与实际数据有所区别。

【例 3-10】 计算不确定人数班级的平均成绩。(StatisticalMeanValue.py)

```
total =0
gradeCounter =0                                     #记录输入的成绩个数
grade =int(input("输入一个成绩,若输入-1 结束成绩输入:"))
while grade !=-1:
    total =total +grade
    gradeCounter =gradeCounter +1
    grade =int(input("输入一个成绩,若输入-1 结束成绩输入:"))
```

```
if gradeCounter !=0:
    average =total / gradeCounter
    print("平均分是:%.2f"%(average))
else:
    print('没有录入学生成绩')
```

在 IDLE 中,StatisticalMeanValue.py 运行的结果如下:

```
输入一个成绩,若输入-1 结束成绩输入:86
输入一个成绩,若输入-1 结束成绩输入:88
输入一个成绩,若输入-1 结束成绩输入:-1
平均分是: 87.00
```

3.5.3 文件式循环

例 3-10 中,如果要输入的数据很多,那么从键盘输入所有数据将是一件非常麻烦的事。可以事先将数据录入文件中,然后将这个文件作为程序的输入,避免人工输入的麻烦,也便于编辑修改。面向文件的方法是数据处理的典型应用。例如,可以把数据存储在一个文本文件(例如,命名为 input.txt)里,并使用下面的命令来运行这个程序:

```
python StatisticalMeanValue.py <input.txt
```

这个命令称为输入重定向。用户不再需要程序运行时从键盘录入数据,而是从文件 input.txt 中获取输入数据。同样地,输出重定向是把程序运行结果输出到一个文件里而不是输出到屏幕上。输出重定向的命令为

```
python StatisticalMeanValue.py >output.txt
```

同一条命令里可以同时使用输入重定向与输出重定向。例如,下面这个命令从 input.txt 中读取输入数据,然后把输出数据写入文件 output.txt 中。

```
python StatisticalMeanValue.py <input.txt >output.txt
```

假设 input.txt 这个文件包含下面的数字,每行一个:

```
45
80
90
98
68
-1
```

在命令行窗口中,StatisticalMeanValue.py 从文件 input.txt 中获取输入数据执行的结果,如图 3-10 所示。

【例 3-11】 例 3-10 的程序实现可改写为更简洁的文件读取的方式来实现,改写后的程序代码如下。(StatisticalMeanValue2.py)

图 3-10　从文件 input.txt 中获取输入数据执行的结果

```
FileName=input('输入数据所在的文件的文件名:')
infile=open(FileName,'r')                          #打开文件
sum=0
count=0
line=infile.readline()                             #按行读取数据
while line!='-1':
    sum=sum+eval(line)
    count=count+1
    line=infile.readline()
if count!=0:
   average =float(sum) / count
   print("平均分是", average)
else:
   print('没有录入学生成绩')
infile.close()                                     #关闭文件
```

StatisticalMeanValue2.py 在命令行窗口中执行的结果如图 3-11 所示。

图 3-11　StatisticalMeanValue2.py 在命令行窗口中执行的结果


for 循环结构

3.6　for 循环结构

3.6.1　for 循环的基本用法

循环结构在 Python 语言中有两种表现形式：一种是前面的 while 循环，另一种是 for 循环。for 循环是一种遍历型的循环，因为它会依次对某个序列中全体元素进行遍

历，遍历完所有元素之后便终止循环。列表、元组、字符串都是序列，序列类型有相同的访问模式：它的每一个元素可以通过指定一个偏移量的方式得到，而多个元素可以通过切片操作的方式得到。

for 循环的语法格式如下：

```
for 控制变量 in 可遍历序列:
    循环体
```

这里的关键字 in 是 for 循环的组成部分，而非运算符 in。“可遍历序列”里保存了多个元素，且这些元素按照一个接一个的方式存储。“可遍历序列”被遍历处理，每次循环时，都会将“控制变量”设置为“可遍历序列”的当前元素，然后执行循环体。当“可遍历序列”中的元素被遍历一遍后，退出循环。for 循环的流程如图 3-12 所示。

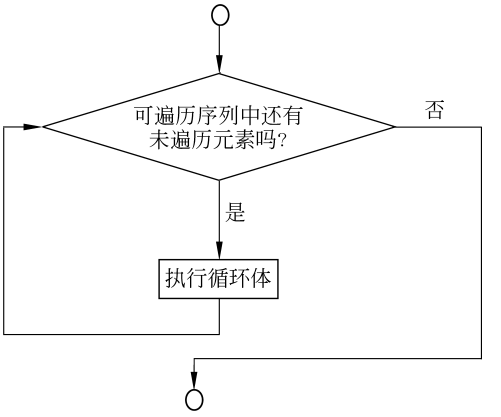

图 3-12 for 循环的流程

3.6.2 for 循环适用的对象

for 循环可用于迭代容器对象中的元素，这些对象可以是列表、元组、字典、集合、文件，甚至可以是自定义类或者函数，举例如下。

1. for 循环作用于列表

【例 3-12】 向姓名列表添加新姓名。(3-12.py)

```
Names =['宋爱梅','王志芳','于光','贾隽仙','贾燕青','刘振杰','郭卫东','崔红宇',
'马福平']
print("-----添加之前,列表 A 的数据-----")
for Name in Names:
    print( Name,end=' ')
print(' ')
continueLoop='y'                                    #让用户来决定是否继续添加
while continueLoop=='y':
    temp =input('请输入要添加的学生姓名:')          #提示并添加姓名
```

```
    Names.append(temp)
    continueLoop=input('输入 y 继续添加,输入 n 退出添加: ')
print ("-----添加之后,列表 A 的数据-----")
for Name in Names:
    print(Name, end=' ')
```

在 IDLE 中,3-12.py 运行的结果如下。

```
-----添加之前,列表 A 的数据-----
宋爱梅 王志芳 于光 贾隽仙 贾燕青 刘振杰 郭卫东 崔红宇 马福平
请输入要添加的学生姓名:李明
输入 y 继续添加,输入 n 退出添加: y
请输入要添加的学生姓名:刘涛
输入 y 继续添加,输入 n 退出添加: n
-----添加之后,列表 A 的数据-----
宋爱梅 王志芳 于光 贾隽仙 贾燕青 刘振杰 郭卫东 崔红宇 马福平 李明 刘涛
```

2. for 循环作用于元组

【例 3-13】 遍历元组。(3-13.py)

```
test_tuple =[("a",1),("b",2),("c",3),("d",4)]
print("准备遍历的元组列表:", test_tuple)
print('遍历列表中的每一个元组')
for (i, j) in test_tuple:
  print(i, j)
```

在 IDLE 中,3-13.py 运行的结果如下。

```
准备遍历的元组列表:[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
遍历列表中的每一个元组
a 1
b 2
c 3
d 4
```

3. for 循环作用于字符串

【例 3-14】 遍历输出字符串中的汉字,遇到标点符号换行输出。(3-14.py)

```
import string
str1 ="大梦谁先觉?平生我自知.草堂春睡足,窗外日迟迟."
for i in str1:
    if i not in string.punctuation:
        print(i,end='')
    else:
        print(' ')
```

在 IDLE 中,3-14.py 运行的结果如下。

```
大梦谁先觉
平生我自知
草堂春睡足
窗外日迟迟
```

4. for 循环作用于字典

【例 3-15】 遍历输出字典元素。(3-15.py)

```
person={'姓名':'李明', '年龄':'26', '籍贯':'北京'}
#items()方法把字典中每对 key 和 value 组成一个元组,并把这些元组放在列表中返回
for key,value in person.items():
    print('key=',key,',value=',value)
for x in person.items(): #只有一个控制变量时,返回每一对 key、value 对应的元组
    print(x)
for x in person:                          #不使用 items(),只能取得每一对元素的 key 值
    print(x)
```

在 IDLE 中,3-15.py 运行的结果如下。

```
key=姓名 ,value=李明
key=年龄 ,value=26
key=籍贯 ,value=北京
('姓名', '李明')
('年龄', '26')
('籍贯', '北京')
姓名
年龄
籍贯
```

5. for 循环作用于集合

【例 3-16】 遍历输出集合元素。(3-16.py)

```
weekdays ={'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'}
#for 循环在遍历集合时,遍历的顺序和集合中元素书写的顺序很可能是不同的
for d in weekdays:
    print (d,end=' ')
```

在 IDLE 中,3-16.py 运行的结果如下。

```
THU TUE MON FRI WED SAT SUN
```

6. for 循环作用于文件

【例 3-17】 for 循环遍历文件,输出文件的每一行。(3-17.py)

文件 1.txt 存有两行文字：

```
向晚意不适,驱车登古原。
夕阳无限好,只是近黄昏。
fd =open('D:\\Python\\1.txt')                    #打开文件,创建文件对象
for line in fd:
    print(line,end='')
```

在 IDLE 中，3-17.py 运行的结果如下。

```
向晚意不适,驱车登古原。
夕阳无限好,只是近黄昏。
```

3.6.3 for 循环与 range()函数的结合使用

很多时候，for 语句都是和 range()函数结合使用的，例如利用两者来输出 0～20 的偶数，如下所示。

```
for x in range(21):
    if x%2 ==0:
        print(x,end=' ')
```

在 IDLE 中，上述程序代码执行的结果如下。

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

现在解释一下程序的执行过程。首先，for 语句开始执行时，range(21)会生成一个由 0～20 这 21 个值组成的序列；然后，将序列中的第一个值（即 0）赋给变量 x，并执行循环体。在循环体中，x% 2 为取余运算，得到 x 除以 2 的余数，如果余数为零，则输出 x 值；否则跳过输出语句。执行循环体中的选择语句后，序列中的下一个值将被装入变量 x，继续循环，以此类推，直到遍历完序列中的所有元素为止。

range 函数用来生成整数序列，其语法格式如下。

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

参数说明如下。

start：计数从 start 开始。默认是从 0 开始。例如 range(5)等价于 range(0，5)。

stop：计数到 stop 结束，但不包括 stop。range(a，b)返回连续整数 a、a+1、…、b−2 和 b−1 所组成的序列。

step：步长，默认为 1。例如，range(0，5)等价于 range(0，5，1)。

range 函数用法举例：

(1) range 函数内只有一个参数时，表示会产生从 0 开始计数的整数序列：

```
>>>list(range(4))
[0, 1, 2, 3]
```

(2) range 函数内有两个参数时，则将第一个参数作为起始位，第二个参数为结束位：

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

(3) range 函数内有 3 个参数时,第三个参数是步长值(步长值默认为 1):

```
>>>list(range(0,10,2))
[0, 2, 4, 6, 8]
```

(4) 如果 range(a,b,k)中的 k 为负数,则可以反向计数,在这种情况下,序列为 a、a+k、a+2k、…,但 k 为负数,最后一个数必须大于 b:

```
>>>list(range(10,2,-2))
[10, 8, 6, 4]
>>>list( range(4,-4,-1))
[4, 3, 2, 1, 0, -1, -2, -3]
```

注意:

(1) 如果直接 print(range(5)),将会得到 range(0, 5),而不会是一个列表。这是为了节省空间,防止过大的列表产生。虽然在大多数情况下,range(0, 5)就像一个列表。

(2) range(5)的返回值类型是 range 类型,如果想得到一个列表,使用 list(range(5))得到的就是一个列表[0, 1, 2, 3, 4]。如果想得到一个元组,使用 tuple(range(5))得到的就是一个元组(0, 1, 2, 3, 4)。

【例 3-18】 输出斐波那契数列的前 n 项。斐波那契数列以兔子繁殖为例子而引入,故又称为"兔子数列",指的是这样一个数列:1、1、2、3、5、8、13、21、34、…,可通过递归的方法定义:F(1)=1,F(2)=1, F(n)=F(n−1)+F(n−2)(n≥3,n∈**N**)。(3-18.py)

问题分析: 从斐波那契数列可以看出,从第三项起每一项的数值都是前两项(可分别称为倒数第二项、倒数第一项)的数值之和,斐波那契数列每增加一项,对下一个新的项来说,刚生成的项为倒数第一项,其前面的项为倒数第二项。

```
a=1
b=1
n=int(input('请输入斐波那契数列的项数(>2 的整数): '))
print('前%d 项斐波那契数列为: '%(n),end='')
print(a,b,end=' ')
for k in range(3,n+1):
    c=a+b
    print(c,end=' ')
    a=b
    b=c
```

在 IDLE 中,3-18.py 运行的结果如下。

```
请输入斐波那契数列的项数(>2 的整数): 8
前 8 项斐波那契数列为: 1 1 2 3 5 8 13 21
```

【例 3-19】 输出斐波那契数列的前 n 项也可以用列表更简单地来实现。(3-19.py)

```
fibs =[1, 1]
n=int(input('请输入斐波那契数列的项数(>2 的整数): '))
for i in range(3,n+1):
    fibs.append(fibs[-2] +fibs[-1])
print('前%d 项斐波那契数列为: '%(n),end='')
print(fibs)
```

在 IDLE 中，执行上述程序代码得到的输出结果如下。

```
请输入斐波那契数列的项数(>2 的整数): 8
前 8 项斐波那契数列为:[1, 1, 2, 3, 5, 8, 13, 21]
```

3.7 循环中的 break、continue 和 else

break 语句和 continue 语句提供了另一种控制循环的方式。break 语句用来终止循环语句，即循环条件没有 False 或者序列还没被完全遍历完，也会停止执行循环语句。如果使用嵌套循环，break 语句将停止执行最深层的循环，并开始执行下一行代码。continue 语句终止当前迭代而进行循环的下一次迭代。Python 的循环语句可以带有 else 子句，else 子句在序列遍历结束(for 语句)或循环条件为假(while 语句)时执行，但循环被 break 终止时不执行。

3.7.1 用 break 语句提前终止循环

可以使用 break 语句跳出最近的 for 或 while 循环。下面的 TestBreak.py 程序演示了在循环中使用 break 的效果。

```
1. sum=0
2. for k in range(1, 30):
3.     sum=sum +k
4.     if sum>=200:
5.         break
6.
7. print('k 的值为', k)
8. print('sum 的值为', sum)
```

TestBreak.py 程序运行的结果：

```
k 的值为 20
sum 的值为 210
```

这个程序从 1 开始，把相邻的整数依次加到 sum 上，直到 sum 大于或等于 200。如果没有第 4 行和第 5 行，这个程序将会计算 1～29 的所有数的和。但有了第 4 行和第 5 行，循环会在 sum 大于或等于 200 时终止，跳出 for 循环。没有了第 4 行和第 5 行，输出将会为