第3章 语句与结构化程序设计

Python 程序是由语句构成的，有的语句完成一个具体的数据处理功能，有的语句控制程序的执行流程。为了让计算机完成一项任务，需要描述完成任务的一系列工作步骤，每一步的工作都由语句实现。语句是 Python 程序的重要组成部分，它控制程序的执行流程，实现程序的功能。

Python 语言既支持面向对象程序设计，也支持面向过程程序设计。面向过程程序设计可以看作面向对象程序设计的基础。结构化程序设计是一种规范的面向过程的程序设计方法，结构化程序强调程序逻辑的简洁明了和程序代码的易于理解。结构化程序由顺序结构、分支结构和循环结构 3 种基本结构组成。

3.1 赋值语句与顺序结构程序设计

3.1.1 赋值语句

赋值语句的语法格式如下：

变量名 1,变量名 2,…,变量名 *n*=表达式 1,表达式 2,…,表达式 *n*

功能：先计算出各表达式的值，然后依次赋给赋值运算符(=)左边的变量。一个赋值语句可以只给一个变量赋值，也可以同时给多个变量赋值；可以赋予常量值，也可以赋予表达式的值；可以给多个变量赋予相同的值，也可以给多个变量分别赋予不同的值。赋值运算符左边的变量要和右边的表达式一一对应。

示例：

```
>>>r=1024                    #给一个变量赋予整型值
>>>pi=3.14                   #给一个变量赋予浮点值
>>>l=2*pi*r                  #把表达式的计算结果赋给变量
>>>b=True                    #给一个变量赋予布尔值
>>>n=int(input("n="))        #把通过键盘输入的值赋给变量
>>>i=j=k=100                 #给多个变量赋予相同的值
>>>x,y,z=3.8,25,"Python"     #给多个变量分别赋予不同的值
>>>a,b=10,20,30              #报错,表达式(值)的个数多于变量个数
>>>a,b,c=10,20               #报错,变量个数多于表达式(值)的个数
```

为方便描述某些先进行运算再赋值的操作，Python 语言还提供了 12 个复合赋值运算符，除了 2.4.4 节介绍的 5 个与位运算有关的复合运算符(&=、|=、^=、<<=、>>=)外，还有如下 7 个常用的复合赋值运算符：

+=、-=、*=、/=、%=、//=、**=

这 7 个复合赋值运算符由算术运算符与赋值运算符组合而成，用于先进行算术运算再赋值。使用复合赋值运算符使程序代码更为简洁，更符合现代编程风格。

示例：

```
a+=3                    #等价于 a=a+3
b*=c-10*d               #等价于 b=b*(c-10*d)
e%=f+10                 #等价于 e=e%(f+10)
i//=6                   #等价于 i=i//6
j**=3                   #等价于 j=j**3
```

说明：

(1) 赋值运算符(=)虽然用的是数学中的等号，但其作用不同于数学中的等号，它不是相等的含义。对于 x=x+1，从数学意义上讲是不成立的；而在 Python 语言中，它是合法的赋值语句，其功能是取出变量 x 中的值加 1 后，再将运算结果存回到变量 x 中去。

(2) 在一个赋值语句中给多个变量同步赋值有时等同于给多个变量分别赋值，但有时不能简单看作给多个变量分别赋值。例如，赋值语句 x,y,z=3.8,25,"Python" 等同于如下 3 个赋值语句：

```
x=3.8
y=25
z="Python"
```

但是，赋值语句 x,y=y,x 的功能和如下两个赋值语句的功能不同：

```
x=y
y=x
```

如果之前有变量定义：x=10 和 y=20，则执行 x,y=y,x 后，x 和 y 的值分别为 20 和 10，即实现了两个变量值的互换；而执行 x=y 和 y=x 后，x 和 y 的值均为 20。

3.1.2　顺序结构程序设计

顺序结构是最简单的一种程序结构。在顺序结构程序中，程序的执行是按照语句块出现的先后顺序执行的，并且每个语句块都会被执行，如图 3.1 所示。在 Python 语言程序中，语句块既可以由具有相同缩进(左对齐)的多条语句组成，也可以只包括一条语句。

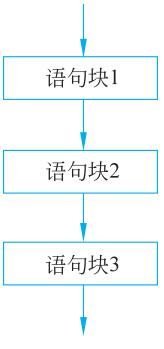

图 3.1　顺序结构

【例 3.1】 通过键盘输入圆的半径,计算圆的面积和周长并输出。

分析:如果圆的半径值是一个实数,编写解决该问题的程序要用到 4 个浮点型变量:r 用于存放从键盘输入的圆的半径值,pi 用于存放常量 π 的值,计算出的圆的周长存入变量 peri,圆的面积存入变量 area。

```
#P0301.py
r=float(input("请输入圆的半径值:"))
pi=3.14
peri=2*pi*r
area=pi*r*r
print("周长=",peri)
print("面积=",area)
```

这是一个典型的顺序结构程序。程序执行时,按书写顺序依次执行程序中的每一条语句。

【例 3.2】 通过键盘输入一个三位整数,求其逆序值并输出。例如,如果输入值为 269,则应输出 962。

分析:从输入的三位整数中先依次分解出个位、十位和百位数字,然后把百位数字和个位数字交换一下,再重新组合即可得到原值的逆序值。

```
#P0302.py
num1=int(input("请输入一个三位整数:"))
one=num1%10                    #分解出个位数字
ten=(num1//10)%10              #分解出十位数字
hun=num1//100                  #分解出百位数字
num2=one*100+ten*10+hun        #交换百位数字和个位数字后重新组合成一个三位整数
print(num1, "的逆序值是:",num2)
```

3.2 分支语句与分支结构程序设计

分支结构又称选择结构。在分支结构中,要根据逻辑条件的成立与否,选择执行不同的语句块,实现不同的功能。分支结构是通过分支语句实现的,Python 语言中的分支语句包括 if 语句、if-else 语句和 if-elif-else 语句。分支语句也称为条件语句或选择语句,其中要用到关系表达式和逻辑表达式。

3.2.1 关系表达式和逻辑表达式

关系表达式和逻辑表达式的运算结果为逻辑值真(True)或假(False),关系表达式也可以看作简单的逻辑表达式。

1. 关系运算符与关系表达式

在 Python 语言中,有 6 个关系运算符,可分为两类:大小判断和相等判断。

大小判断运算符有 4 个:>(大于)、>=(大于或等于)、<(小于)、<=(小于或等于)。

相等判断运算符有 2 个:==(相等)、!=(不相等)。

用关系运算符连接运算对象所形成的表达式称为关系表达式。

例如，数学中的 x≥10 可以写成 Python 中的关系表达式 x>=10，y≠6 可以写成 y!=6。

关系表达式的运算结果是逻辑值真或假，分别用 True 和 False 表示。也就是说，如果表达式成立，则表达式的结果为 True（真）；如果表达式不成立，则表达式的结果为 False（假）。表达式 10>5 的结果为 True，而表达式 10<5 的结果为 False。

执行以下赋值语句：

```
>>>b1=(10<=20)
>>>b2=(10>20)
```

b1 和 b2 的值分别为 True 和 False。

注意：

（1）判断相等要用两个等号（==），不能用单个等号（=），单个等号是赋值运算符，两个等号才是判断相等运算符。

（2）如果关系运算符由两个符号组成（如>=），这两个符号之间不能出现空格。

2. 逻辑运算符与逻辑表达式

关系运算能够进行比较简单的判断。如前所述，数学中的 score≥60 可以写成 score>=60。如果要进行比较复杂的判断，如数学中的 score1≥60 且 score2≥90，就需要用到逻辑运算符把多个关系表达式连接在一起，形成逻辑表达式。

在 Python 语言中，有 3 个逻辑运算符：and（逻辑与）、or（逻辑或）、not（逻辑非）。其中，and 和 or 是双目运算符，not 是单目运算符。逻辑运算规则如表 3.1 所示。

表 3.1　逻辑运算规则

x	y	not x	not y	x and y	x or y
False	False	True	True	False	False
False	True	True	False	False	True
True	False	False	True	False	True
True	True	False	False	True	True

逻辑运算符的运算规则可以描述如下：

（1）and（逻辑与）：当两个运算对象中有一个为 False 或两个均为 False 时，结果为 False；当两个运算对象都为 True 时，结果为 True。

（2）or（逻辑或）：当两个运算对象中有一个为 True 或两个均为 True 时，结果为 True；当两个运算对象都为 False 时，结果为 False。

（3）not（逻辑非）：当运算对象为 False 时，结果为 True；当运算对象为 True 时，结果为 False。

有了逻辑运算符，数学中的 score1≥60 且 score2≥90 就可以写成如下形式的 Python 逻辑表达式：

```
score1>=60 and score2>=90
```

对于数学表表达式 60≤score<90，在 Python 语言中，既可以写成

```
score>=60 and score<90
```

也可以写成

```
60<=score<90
```

下面再看几个逻辑表达式的**示例**：

两门课程都大于或等于 60 分，可以表示为

```
score1>=60 and score2>=60
```

两门课程都大于或等于 60 分，而且其中有一门课程大于或等于 90 分，可以表示为

```
score1>=60 and score2>=60 and (score1>=90 or score2>=90)
```

两门课程都大于或等于 60 分，而且两门课程的平均分大于或等于 90 分，可以表示为

```
score1>=60 and score2>=60 and (score1+score2)//2>=90
```

注意：逻辑运算符与运算对象之间至少要留有一个空格。

为便于正确书写和理解表达式，给出运算符的优先级和结合性，如表 3.2 所示。

表 3.2　运算符的优先级和结合性

运　算　符	优　先　级		结合性
**(幂运算) not(逻辑非) +、-(正号、负号)	同级	↓	右结合
*、/、//、%(乘除运算)	同级		左结合
+、-(加减运算)	同级		
>、>=、<、<=(判断大小关系运算)	同级		
==、!=(判断相等关系运算)			
and(逻辑与)	—		
or(逻辑或)	—	低	

给定如下 3 个逻辑表达式：

```
score1>=60 and score2>=60 or score3>=90
(score1>=60 and score2>=60) or score3>=90
score1>=60 and (score2>=60 or score3>=90)
```

根据表 3.2 所给运算符的优先级，可知前两个表达式的含义相同，与第三个表达式的含义不同。

如果一时没有准确掌握各运算符的优先级，可以用加小括号的形式明确指定各运算的优先级。实际上，应多采用后两个表达式的写法，能够更清晰地表达编程者的意图，也更易于程序阅读者理解表达式的含义。

说明：

(1) 左结合是指从左向右计算，右结合是指从右向左计算。例如，a＋b－c 等价于(a＋b)－c，而 a**b**c 等价于 a**(b**c)。

(2) 逻辑表达式的运算结果是一个逻辑值：真或假，真用 True 表示，假用 False 表示。True 和 False 为首字母大写，其他字母小写，TRUE、FALSE、true、false 等形式都是错误的。

3.2.2　if 语句

if 语句用来实现单分支选择，语法格式如下：

```
if 表达式:
    语句块
```

if 分支结构如图 3.2 所示。if 语句的执行过程是：先计算表达式的值，若值为 True(真)，则执行 if 子句(表达式后面的语句块)，然后执行 if 结构后面的语句；否则，跳过 if 子句，直接执行 if 结构后面的语句。

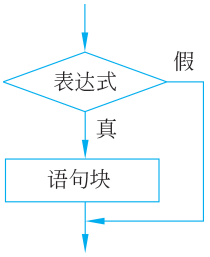

图 3.2　if 分支结构

示例：

```
if score<60:
    m+=1                #如果成绩不及格,则 m 的值加 1
n+=1                    #不管成绩是否及格,n 的值都要加 1
```

如果是对一门课程的考试成绩进行上述操作(多次执行上面的程序段)，其功能是统计参加考试的总人数(n 的值)和不及格人数(m 的值)。

注意：不要漏掉条件表达式后面的冒号(:)。

3.2.3　if-else 语句

if-else 语句用来实现双分支选择，即 if-else 语句可以根据条件的真(True)或假(False)执行不同的语句块。

if-else 语句的语法格式如下：

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

其中,语句块 1 称为 if 子句,语句块 2 称为 else 子句。

if-else 分支结构如图 3.3 所示。if-else 语句的执行过程是:先计算表达式的值,若结果为 True,则执行 if 子句(语句块 1),否则执行 else 子句(语句块 2)。

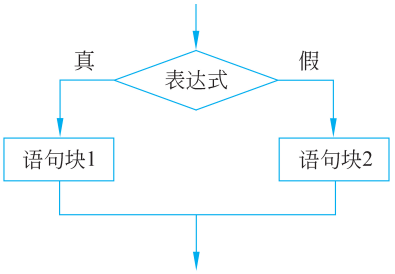

图 3.3 if-else 分支结构

示例:

```
if score<60:
    m+=1          #如果成绩不及格,则 m 的值加 1
else:
    n+=1          #如果成绩及格,则 n 的值加 1
```

如果是对一门课程的考试成绩进行上述操作(多次执行上面的程序段),其功能是统计不及格人数(m 的值)和及格人数(n 的值)。

【例 3.3】 根据输入的学生成绩对其进行判断处理:如果成绩及格,则输出“考试通过!”;否则输出“考试没通过!”。

```
#P0303_1.py
score=int(input("请输入考试成绩:"))
if score>=60:
    print("考试通过!")
else:
    print("考试没通过!")
```

一段程序中只有一个 if 和一个 else 是比较容易理解的。如果有多个 if 和 else,则 if 与 else 的匹配关系要书写准确。在 Python 语言中,if 与 else 的匹配是通过严格的对齐与缩进实现的。

示例:

```
score=int(input("score="))
if score<60:
    if score<45:
        print("不能参加补考,必须重修!")
    else:
        print("可以参加补考!")
```

该程序段中,else 与第二个 if 对齐匹配。其功能是:在成绩低于 60 分的前提下,如果成绩低于 45 分必须重修(没有补考机会),否则(成绩为 45～59 分)可以参加补考。

如果写成如下缩进形式(else 与第一个 if 对齐匹配):

```
score=int(input("score="))
if score<60:
    if score<45:
        print("不能参加补考,必须重修!")
else:
    print("可以参加补考!")
```

其功能变为：成绩低于 45 分必须重修，在成绩大于或等于 60 分时可以参加补考。很显然，这不是程序要表达的处理逻辑。程序实际想表达的逻辑是：如果考试成绩太低（低于 45 分），则没有参加补考的机会，必须重修；如果考试成绩为 45～59 分，可以参加补考。

一个更完整的程序如下：

```
#P0303_2.py
score=int(input("score="))
if score<60:
    if score<45:
        print("考试未通过,需要重修!")
    else:
        print("考试未通过,可以补考!")
else:
    print("考试通过!")
```

这个程序中有两个 if-else 语句，虽然表达的逻辑并不复杂，但对初学编程者来说，直接写程序代码或许有点难度。可以先画出体现逻辑思路的流程图，然后再按流程图写出程序代码，这样更容易保证程序代码正确实现预期功能。处理考试成绩完整程序的流程图如图 3.4 所示。

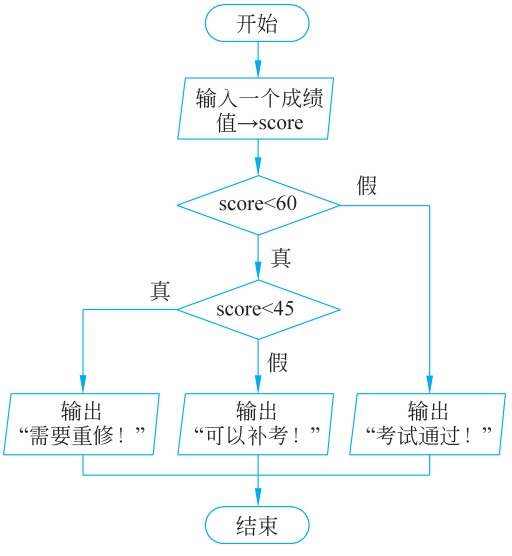

图 3.4　处理考试成绩完整程序的流程图

3.2.4　if-else 表达式

在 Python 语言中，if-else 语句还可以写成如下 if-else 表达式形式：

```
表达式 1  if 条件表达式 else 表达式 2
```

其执行过程是：先计算条件表达式的值。若其值为 True，则整个表达式的值为表达式 1 的值；否则，整个表达式的值为表达式 2 的值。

示例：

```
a-b if a>=b else b-a
```

若条件表达式 a＞＝b 的值为 True，则上面 if-else 表达式的值为 a－b，否则其值为 b－a。

对于以下赋值语句：

```
c=a-b if a>=b else b-a
```

若 a＝30、b＝20，则执行该语句后 c 的值为 10；若 a＝30、b＝50，则执行该语句后 c 的值为 20。即 c 的值为 a－b 的绝对值。

这个赋值语句也可以写成如下条件语句的形式：

```
if a>=b:
    c=a-b
else:
    c=b-a
```

显然，写成表达式的形式更为简洁。当然，实际编写程序时，是写成 if-else 表达式形式还是 if-else 语句形式，取决于编程人员自己的习惯。

3.2.5 if-elif-else 语句

如果成绩是 5 分制(5 分为优秀，4 分为良好，3 分为及格，2 分和 1 分为不及格)，现在需要把等级分(1～5) 转换为对应的“优秀”“良好”“及格”“不及格”等信息输出。

假设变量 grade 中存放了等级分，则实现上述功能的程序段如下：

```
if (grade==5):
    print("优秀")
else:
    if (grade==4):
        print("良好")
    else:
        if (grade==3):
            print("及格")
        else:
            if (grade==2):
                print("不及格")
            else:
                if (grade==1):
                    print("不及格")
                else:
                    print("成绩值错误")
```

这种多层嵌套方式从语法上是正确的，但书写起来比较麻烦，也不容易理解其功能。if-elif-else 语句更适合完成这种多选择功能，if-elif-else 语句的语法格式如下：

```
if 表达式 1:
    语句块 1
elif 表达式 2:
    语句块 2
  ⋮
elif 表达式 n:
    语句块 n
else:
    语句块 n+1
```

if-elif-else 语句的执行过程是：按先后顺序计算各表达式的值，如果表达式 1 的值为 True，则执行语句块 1；否则，如果表达式 2 的值为 True，则执行语句块 2；以此类推，如果表达式 n 的值为 True，则执行语句块 n；如果前面所有表达式都不成立，则执行语句块 $n+1$。即按顺序依次判断表达式 1、表达式 2……表达式 n 的取值是否为 True。如果某个表达式的取值为 True，则执行与之对应的语句块，然后结束整个 if-elif-else 语句；如果所有表达式的取值都不为 True，则执行与 else 对应的语句块。

上述多重 if-else 嵌套语句改写成 if-elif-else 语句如下：

```
if grade==5:
    print("优秀")
elif grade==4:
    print("良好")
elif grade==3:
    print("及格")
elif grade==2:
    print("不及格")
elif grade==1:
    print("不及格")
else:
    print("成绩值错误")
```

相对于多层嵌套的 if-else 语句，相同功能的 if-elif-else 语句既简洁又易于阅读和理解。

关于 if-elif-else 语句的几点解释如下：

(1) 在依次对各表达式进行判断时，遇到一个结果为 True 的表达式，执行完对应的语句块就结束整个 if-elif-else 语句。即使有多个表达式成立，后面的表达式也不再判断。

示例：

```
if score>=90:
    num1+=1
elif score>=80:          #等价于 90>score>=80
    num2+=1
elif score>=60:          #等价于 80>score>=60
    num3+=1
else:                    #等价于 score<60
    num4+=1
```

对于此程序段，如果有多个成绩值(score)需要处理，实际上是统计几个成绩段的人数。如果 score 的值为 95，第一个表达式(score>=90)成立，执行赋值语句 num1+=1，然后结束整个 if-elif-else 语句，后面的表达式不再判断，自然也不会再执行对应的语句。

(2) 每个语句块可以包括多条语句,也可以只包括一条语句。同一语句块中的各条语句要有相同的缩进,否则会被认为是不同语句块的语句。

(3) 最后的 else 以及与之对应的语句块可以省略,此时,若前述所有表达式都不成立,则 if-elif-else 语句什么都不执行。

3.2.6 流程图的画法

编写程序的前提是对待解问题进行深入分析并且得到解决问题的合适算法,算法是解决某一问题的思路和步骤。描述算法的方式有多种,其中流程图方式简单明了,易于画出,易于理解,也易于转换为相应的程序,是一种常用的算法描述方法。图 3.4 就是一个典型的流程图,比较清晰地描述了关于考试成绩的相关规定。

在流程图中,用一些框图表示各种类型的操作,用带箭头的线段表示操作的执行顺序,常用的图形符号如图 3.5 所示。

图 3.5 流程图中常用的图形符号

流程图中用到的各图形符号的含义如下:

- 起止框是一个圆角矩形,表示算法由此开始或到此结束。
- 处理框是一个矩形,表示一个具体的数据处理操作。
- 判断框是一个菱形,表示一种判断条件。
- 输入输出框是一个平行四边形,表示输入或输出操作。
- 连接框是一个圆形,连接画在两页上的同一个流程图。
- 流程线是一个带箭头的线段,表示程序的执行走向。

画流程图是为了编程人员更为清晰准确地理解算法、描述算法,进而编写出逻辑清晰、功能正确的程序。对于一个要解决的问题,是先画出流程图再编写程序,还是直接编写程序,应根据实际情况而定。基本原则是:如果对算法思路清晰明了,可以直接编写程序;如果对算法思路的理解不是很清晰,可以先画出流程图,进一步理清算法思路后再编写程序。

3.3 循环语句与循环结构程序设计

用计算机解决实际问题时,经常会遇到需要重复进行的数据处理工作,例如把通过键盘输入的数值累加起来,此时要重复完成的工作是输入数据和累加。解决这类问题有两种方式:一是在程序中重复书写有关输入数据和累加的程序代码;二是让有关输入数据和累加的代码重复执行。前者会使程序代码书写量很大(如果输入 10 000 个数据并累加,就需要

把输入数据和数据累加代码重复书写 10 000 次），大幅度增加编写程序的工作量；而后者只写少量代码就能实现同样的功能。实际上，后者是通过一种控制结构使某些语句重复执行的，这种结构称为循环结构。

一组被重复执行的语句称为循环体语句块。程序执行过程中，每执行一次循环体语句块，必须作出是继续循环还是停止的决定，这个决定所依据的条件称为循环条件。

循环结构通过循环语句实现。Python 语言中有两种循环语句：for 循环语句和 while 循环语句。

3.3.1　for 循环语句

for 循环语句的语法格式如下：

```
for 循环变量 in 遍历结构:
    语句块
```

for 循环结构如图 3.6 所示。for 循环语句的执行过程是：循环变量从遍历结构中取值，如果能从遍历结构中取到值，就执行循环体语句块，然后再返回遍历结构取值，如果还能取到值，再次执行循环体语句块，直至遍历结构中的数据取完为止。遍历结构是指包含多个数据元素的复合数据。

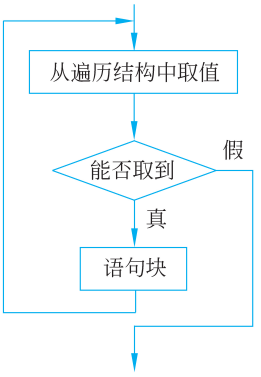

图 3.6　for 循环结构

【例 3.4】　从键盘输入若干个以正整数表示的考试成绩，计算总成绩和平均成绩并输出。

分析：这是一个重复累加问题，从键盘上输入一个成绩值，进行一次累加，再输入一个成绩值，再进行一次累加……直至成绩值全部输入并累加完毕。这里循环继续的条件是成绩值未累加完，而每次重复的工作为输入数据并进行累加。设 score 为接收键盘输入的成绩值并参与累加的变量；total_score 为存放累加和的变量，简称累加变量；i 为循环控制变量，用于控制循环的次数。

```
#P0304.py
n=int(input("成绩个数 n="))                    #输入成绩个数
total_score=0                                  #累加变量清零
for i in range(1,n+1):                         #i 的取值为 1～n,即循环次数为 n
    score=int(input("请输入一个成绩值:"))      #通过键盘输入一个成绩值
    total_score+=score                         #累加求和
average_score=total_score//n                   #计算平均成绩
print("总成绩=",total_score)                   #输出总成绩
print("平均成绩=",average_score)               #输出平均成绩
```

为了调试程序时输入数据简单快捷，一开始可以为 n 输入值 5，程序调试正确后，可以用于计算班内某门课程的成绩，也可以用于计算自己已学课程的成绩，为变量 n 输入实际的成绩个数就可以了。

关于 for 语句的几点说明如下：

（1）此处的循环变量 i 在 range(1,n+1)范围内取值，取值为 1～n(不包括 n+1)，所以循环体语句块执行 n 次。range()函数的一般格式如下：

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

其功能是生成若干整数值,起始值为 start,结束值为 end－1 或 end＋1(注意,不包括 end),步长为 step。当步长 step 为正数时,结束值为 end－1;当步长 step 为负数时,结束值为 end＋1。其中,start 和 step 都可以省略,省略时默认值分别为 0 和 1。

示例:

```
range(0,10,1)          #生成的值为 0～9
range(10)              #生成的值为 0～9,默认起始值为 0,步长为 1
range(1,10)            #生成的值为 1～9,默认步长为 1
range(1,11,2)          #生成的值为 1、3、5、7、9,即 1～10 的奇数
range(2,11,2)          #生成的值为 2、4、6、8、10,即 2～10 的偶数
range(10,0,-1)         #生成的值为 10～1,步长可以为负数
range(30,20,-2)        #生成的值为 30、28、26、24、22,即 30～21 之间的偶数
```

(2) 在该程序中,for i in range(1,n＋1),也可以写成 for i in range(0,n)或 for i in range(2,n＋2)等形式,因为 i 的取值只起控制循环次数的作用,只要结束值和起始值之间的差值为 n 即可,当然还是 for i in range(1,n＋1)更容易理解一些。如果循环变量要参与循环体语句块的运算,不同写法的功能就不一样了。

如下程序段的功能是计算 $1^2+2^2+\cdots+n^2$:

```
n=int(input("n="))
total=0
for i in range(1,n+1):          #循环变量 i 的取值为 1～n
    total+=i*i
print("total=",total)
```

如下程序段的功能是计算 $0^2+1^2+2^2+\cdots+(n-1)^2$:

```
n=int(input("n="))
total=0
for i in range(0,n):            #循环变量 i 的取值为 0～n-1
    total+=i*i
print("total=",total)
```

(3)根据数据处理需要,正确书写缩进格式。不同的缩进格式实现的功能是不一样的。上面的两个程序段都是计算完累加和后再输出最后累加和的值。如果改成如下缩进格式,功能变为每累加一次,都会输出中间累加结果。通过本例,仔细体会 Python 缩进格式的作用。

```
n=int(input("n="))
total=0
for i in range(1,n+1):
    total+=i*i
    print("total=",total)
```

【例 3.5】 从键盘输入一个大于 1 的自然数,判断其是否为素数。

分析: 根据素数的定义,一个大于 1 的自然数 n,如果只能被 1 和其本身整除,则 n 为素数。根据素数的定义直接编写程序判断一个数是否为素数不太容易。可以对素数定义转换一下描述,用 2～n－1 之间的所有整数逐一去除 n,如果都不能整除,则确定 n 为素数;如

果至少有一个能够整除，则确定 n 不是素数。所以，判断一个数是否为素数需要用循环结构实现。

解决这个问题的思路有一点复杂。如果直接编写程序感觉有些困难，可以先画一个流程图，明晰解决问题的思路，然后再按流程图写出程序代码。判断素数流程图如图 3.7 所示。

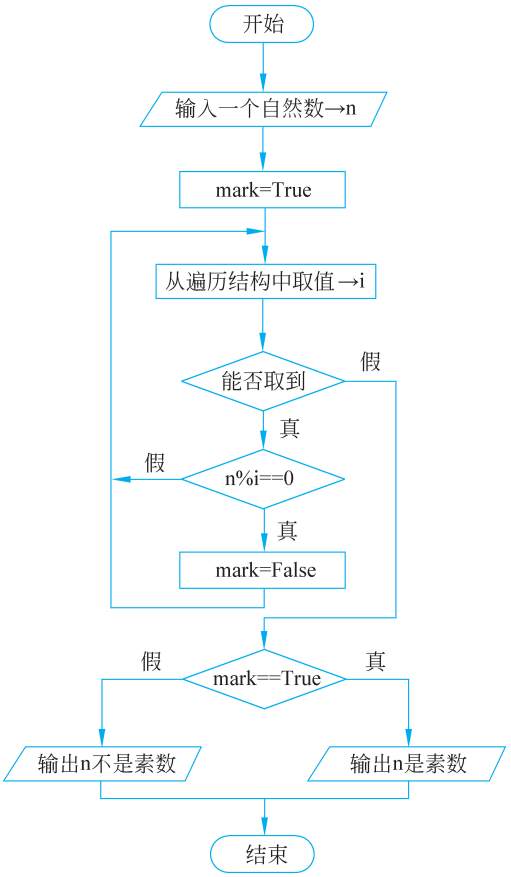

图 3.7　判断素数流程图

```
#P0305_1.py
n=int(input("请输入一个大于 1 的自然数:"))
mark=True                          #设定标记的初值,假定 n 为素数
for i in range(2,n):               #变量 i 的取值为 2~n-1
    if (n%i==0):                   #如果能够整除
        mark=False                 #则把 mark 的值改为 False
if mark==True:                     #如果 mark 的值保持为 True,说明都不能整除
    print(n,"是素数")              #进而说明假定 n 为素数是对的,输出 n 是素数信息
else:
    print(n,"不是素数")
```

说明：

(1) 用 for 循环语句时，不要忘记第一行末尾的冒号(:)。

(2) 该程序用最基本的方法判断一个数是否为素数。还有更节省除法次数的方法,可自己思考并改写上述程序。

(3) 语句之间的缩进关系决定了语句之间的层次关系,也决定着每条语句的作用及整个程序的功能,因此要正确书写缩进形式。如果把上述程序改为如下缩进格式,思考程序功能将有什么变化,并上机查看输入数值分别为 4、17、25 时的运行结果。

```
#P0305_2.py
n=int(input("请输入一个大于 1 的自然数: "))
mark=True                          #设定标记的初值,假定 n 为素数
for i in range(2,n):               #变量 i 的取值为 2~n-1
    if (n%i==0):                   #如果能够整除
        mark=False                 #则把 mark 的值改为 False
    if mark==True:
        print(n,"是素数")
    else:
        print(n,"不是素数")
```

3.3.2 while 循环语句

while 循环语句的语法格式如下:

```
while 表达式:
    语句块
```

while 循环结构如图 3.8 所示。while 循环语句的执行过程是:先计算表达式的值,若表达式的值为真(True),则执行循环体语句块;然后再次计算表达式的值,若结果仍为真(True),再次执行循环体语句块;如此继续下去,直至表达式的值变为假(False),则结束 while 循环语句的执行。

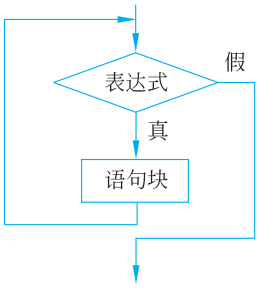

图 3.8 while 循环结构

一般来说,一开始表达式的值为真(True)。在循环体语句块中要有修改循环条件的语句,以使表达式的值在执行若干次循环体语句块后变为假。若表达式的值总为真,则形成一直循环下去的永真循环;若表达式的值一开始就为假,则循环体语句块一次也不执行。

【例 3.6】 用 while 语句实现例 3.4 的功能。

```
#P0306.py
n=int(input("成绩个数 n="))                  #输入成绩个数作为循环次数
total_score=0                                #累加变量初值赋 0
num=1                                        #给计数变量 num 赋初值为 1
while num<=n:                                #判断 num 的值是否小于或等于 n 的值
    score=int(input("请输入一个成绩值:"))    #通过键盘输入一个成绩值
    total_score+=score                       #对成绩值进行累加
    num+=1                                   #计数变量 num 的值增 1
average_score=total_score//n                 #计算平均成绩
```

```
print("总成绩=",total_score)
print("平均成绩=",average_score)
```

说明： for 循环语句的循环次数决定于所用遍历结构(如 range()函数)中元素的个数，其循环变量依次取遍历结构中各元素的值；使用 while 循环语句，也需要用一个变量控制循环次数，循环开始前给该变量设定一个合适的初值，使 while 后面的表达式成立，在循环体语句块中要有改变该变量值的语句，随着一次一次的循环，逐渐改变循环变量的值，经过若干次循环后，使 while 后面的表达式变为不成立，循环结束。仔细体会一下例 3.6 中变量 num、例 3.7 中变量 score 对控制循环次数的作用。

【例 3.7】 从键盘上输入若干以正整数表示的考试成绩，计算总成绩和平均成绩并输出。与前面例子不同的是，本例不知道成绩的个数，用输入－1 作为结束。

分析： 该问题是一个条件控制的循环，知道循环结束条件，不知道循环次数，此时更适合用 while 循环语句实现。流程图如图 3.9 所示。

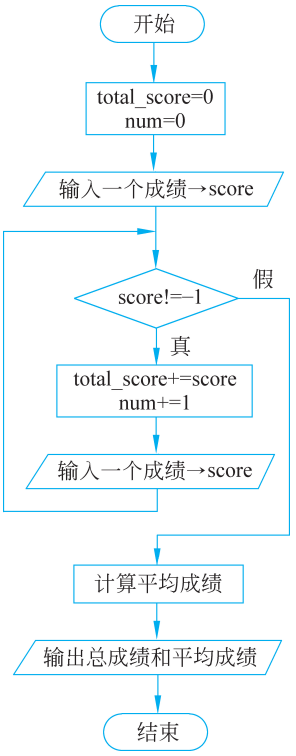

图 3.9　计算成绩流程图

```
#P0307.py
total_score=0                              #累加变量初值为 0
num=0                                      #计数变量初值为 0
score=int(input("请输入一个成绩值: "))     #输入一个成绩值
while score!=-1:                           #判断是否满足循环结束条件
```

```
        num+=1                                  #成绩个数加 1
        total_score+=score                      #对成绩进行累加
        score=int(input("请输入一个成绩值: "))   #再次输入成绩值
    average_score=total_score//num              #计算平均成绩
    print("总成绩=",total_score)
    print("平均成绩=",average_score)
```

说明：从上面几个例子可以看出，对于已知循环次数的循环程序，用 for 语句实现比较简单；对于不知道循环次数但知道结束条件的循环程序，用 while 语句实现更为合适。

3.3.3 循环语句的嵌套

循环是可以嵌套的，可以把一个循环语句嵌套在另一个循环语句内，形成二重循环，以解决一些比较复杂的问题。for 语句和 while 语句既可以自身嵌套，也可以相互嵌套，但要求一点：一个循环语句要完全嵌套在另一个循环语句之内，即先开始的循环语句后结束，后开始的循环语句先结束。按此原则，也可以进行多层嵌套，形成多重循环。

【例 3.8】 找出 1000～3000 的所有素数并统计素数个数。

分析：例 3.4 是判断某一个数是否为素数，用了 for 循环语句实现，可以看作单重循环程序。现在有 1000～3000 的 2001 个数需要判断是否为素数并统计素数个数，需要再加一层循环，构成二重循环程序，外层循环依次取出需要判断的数，内层循环具体判断取到的某个数是否为素数。

```
#P0308_1.py
num=0                               #统计素数个数的计数器变量,初值赋 0
for n in range(1000,3001):
    mark=True                       #设定标记的初值,假定 n 为素数
    for i in range(2,n):            #变量 i 的取值为 2～n-1
        if (n%i==0):                #如果能够整除
            mark=False              #则把 mark 的值改为 False
    if mark==True:                  #如果 mark 的值保持为 True,说明都不能整除
        print(n,"是素数")           #说明假定 n 为素数是对的,输出 n 是素数信息
        num+=1                      #计数器的值加 1
print("素数总数:",num)              #1000～3000 的素数总数
```

上面的二重循环用的都是 for 语句，也可以都用 while 语句，给出程序如下：

```
#P0308_2.py
num=0                               #统计素数个数的计数器变量,初值赋 0
n=1000                              #需要判断是否为素数的初始值
while n<=3000:                      #如果 n≤3000 进行下面的判断,否则结束循环
    mark=True                       #设定标记的初值,假定 n 为素数
    i=2                             #变量 i 的初值为 2
    while i<=n-1:                   #如果 i≤n-1 进行下面的判断,否则结束循环
        if n%i==0:
            mark=False              #如果有能够整除的数,则把 mark 的值改为 False
        i+=1                        #控制内层循环次数的变量 i 的值加 1
    if mark==True:                  #如果 mark 的值保持为 True,说明都不能整除
```

```
            print(n,"是素数")          #进而说明假定 n 为素数是对的,确定 n 为素数
            num+=1                     #计数器的值加 1
        n+=1                           #控制外层循环次数的变量 n 的值加 1
print("素数总数:",num)                 #1000～3000 的素数总数
```

本例用二重循环程序实现其功能,虽然用 while 语句可以写出循环程序,但是由于外层循环的次数和内层循环的次数都是确定的,所以用 for 语句写出循环程序更为简洁且更易于理解。

3.3.4　带 else 的循环语句

前面介绍的是循环语句 for 和 while 的基本结构。在 Python 语言中,for 语句和 while 语句还都有带 else 的扩展形式,语法格式分别如下:

```
for 循环变量 in 遍历结构:
    语句块 1
else:
    语句块 2

while 表达式:
    语句块 1
else:
    语句块 2
```

其共同点是:当循环语句正常结束时,执行 else 对应的语句块;当循环语句提前结束时,不执行 else 对应的语句块。带 else 的循环语句一般要和 3.4 节介绍的 break 语句配合使用。

3.4　退出循环语句

有时循环不一定要执行完预定的次数,可以提前退出。例如,判断一个大于 1 的自然数 n 是否为素数时,只要 2～$n-1$ 中有一个数能够整除 n,就能确定 n 不是素数,此时可提前结束循环程序,以提高程序执行效率。提前结束循环的语句称为退出循环语句,也称为转移语句。退出循环语句有两个,分别为 break 语句和 continue 语句,其共同特点是改变程序的当前执行顺序,转到程序的另一个位置继续执行。

3.4.1　break 语句

break 语句的语法格式如下:

```
break
```

break 语句主要用于循环结构中,其功能是提前结束整个循环,转去执行循环结构后面的语句。

【例 3.9】　计算总成绩。如果输入的成绩值都有效,计算出总成绩并输出;如果输入的成绩值中有无效值,中止累加计算并给出提示信息。

```
#P0309_1.py
total_score=0                                    #定义累加变量并赋初值为 0
for i in range(1,11):                            #设定循环次数为 10
    score=int(input("请输入一个成绩值:"))          #输入一个成绩值
    if (score<0 or score>100):                   #若是无效成绩值
        print("成绩值无效,中止程序!")              #给出提示信息
        break                                    #提前结束循环
    else:
        total_score+=score                       #对有效成绩值进行累加
else:
    print("总成绩=",total_score)                  #循环正常结束,输出总成绩
```

该程序的功能是对通过键盘输入的成绩值进行累加,预定的循环次数为 10。但是,如果输入的某个成绩值是无效成绩,即不在 0～100 范围内,便提前结束整个循环语句的执行,并给出相应的提示信息,不执行 else 对应的语句块;如果 10 个成绩值都是有效值,则执行 else 对应的语句块,输出总成绩。

如果不用带 else 的 for 循环语句,也能实现上述效果,但不如用带 else 的 for 循环语句简洁。不带 else 的 for 循环程序如下:

```
#P0309_2.py
total_score=0                                    #定义累加变量并赋初值 0
num=0                                            #成绩个数计数,初值赋 0
for i in range(1,11):                            #设定循环次数为 10
    score=int(input("请输入一个成绩值:"))
    if (score<0 or score>100):                   #若是无效成绩值
        break                                    #提前结束循环
    else:
        total_score+=score                       #对有效成绩值进行累加
        num+=1                                   #累加成绩个数加 1
if num==10:                                      #如果累加成绩个数为 10
    print("总成绩=",total_score)                  #循环正常结束,输出总成绩
else:
    print("成绩值无效,中止程序!")                  #循环提前结束,给出提示信息
```

3.4.2 continue 语句

continue 语句的语法格式如下:

```
continue
```

continue 语句用于循环结构中,其功能是提前结束本次循环,转回循环的开始,判断是否执行下一次循环。

【例 3.10】 计算总成绩并输出。在成绩值输入过程中,输入－1 结束成绩输入。如果遇到无效成绩值,提示用户重新输入。

```
#P0310.py
total_score=0                                    #累加变量赋初值 0
score=int(input("请输入一个成绩值:"))
while score!=-1:                                 #判断循环条件
```

```
    if (score<0 or score>100):                    #若是无效成绩值
        print("成绩值无效,请重新输入!")            #提示重新输入
        score=int(input("请输入一个成绩值:"))      #重新输入成绩值
        continue                                  #结束本次循环,转回循环开始
    else:
        total_score+=score                        #对有效成绩值进行累加
        score=int(input("请输入一个成绩值:"))      #继续输入成绩值
print("总成绩=",total_score)
```

注意：一个循环结构一般执行若干次循环体语句块。continue 语句的作用是结束本次循环，执行下一次循环；break 语句的作用是结束整个循环，执行循环结构之后的语句。break 和 continue 功能对比如图 3.10 所示。

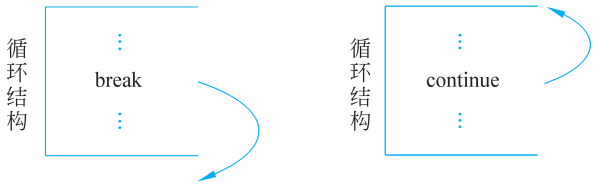

图 3.10　break 和 continue 功能对比

3.5　Python 语句的书写规则

书写程序代码，除了把每条语句书写正确外，还要注意语句之间的结构关系。Python 语言在缩进、一行写多条语句、一条语句写在多行、注释等方面都有相应的规定。遵守语句书写规则，才能保证程序正确实现预期功能，而且使程序易于理解，易于发现和修改其中的错误。

3.5.1　缩进

Python 与其他语言最大的区别就是构成 Python 程序的语句必须严格按照缩进规则书写，Python 解释器是通过缩进量识别语句之间的层次关系和界定语句块的。缩进的空格数是可变的，具有相同缩进量的一组语句构成一个语句块。语句的缩进可以通过制表符（Tab 键）或空格键实现，缩进量可多可少，一般设置为 4 个空格。

示例：

```
from random import *                  #导入标准库 random 中的所有函数
sum1=0                                #累加变量初值为 0
for i in range(1,6):                  #设定循环次数为 5
    n=randint(10,20)                  #生成一个 10～20 的随机整数
    print(n)                          #输出生成的随机整数
    sum1+=n                           #把随机整数累加到累加变量中
print(sum1)                           #输出 5 个随机整数的累加和
```

该程序实现的功能为：生成 5 个 10～20 的随机整数，输出每个随机整数及其累加和。某