第 3 章

Python 控制语句

学习目标

- 理解结构化程序设计的三种基本结构
- 掌握分支结构 if 语句的用法
- 掌握 for 语句循环结构
- 掌握 while 语句循环结构

3.1 结构化程序设计

结构化程序设计方法的基本思想是以系统的逻辑功能设计和数据流关系为基础，根据数据流程图和数据字典，借助于标准的设计准则和图表工具，通过“自上而下”和“自下而上”的反复，逐层把系统划分为多个大小适当、功能明确、具有一定独立性并容易实现的模块，从而把复杂系统的设计转变为多个简单模块的设计。结构化程序设计方法可以用三个词语进行概括：自上而下、逐步求精、模块化设计。

结构化的含义是指用一组标准的准则和工具从事某项工作。在结构化程序设计之前，每一个程序员都按照各自的习惯和思路编写程序，没有统一的标准，也没有统一的技术方法，因此，程序的调试、维护都很困难，这是造成软件危机的主要原因之一。20 世纪 60 年代，计算机科学家提出了有关程序设计的新理论，即结构化程序设计理论，该理论认为，任何一个程序都可以用三种基本逻辑结构编制，而且只需这三种结构。这三种结构分别是顺序结构、分支结构(选择结构)和循环结构。这种程序设计的新理论，促使人们采用模块化编制程序，把一个程序分成若干个功能模块，这些模块之间尽量彼此独立，用控制语句或过程调用语句将这些模块连接起来，形成一个完整的程序。一般来说，结构化程序设计方法不仅大大改进了程序的质量和程序员的工作效率，而且还增强了程序的可读性和可维护性。

计算机程序一般都由三种基本结构组成：顺序结构、分支结构(选择结构)和循环结构。结构化程序的结构可以用多种方式表示，例如程序流程图、N-S 图、PAD 图等。程序流程图是广泛使用的结构化设计表示工具，具有表达直观，易于掌握的特点。

3.1.1 顺序结构

顺序结构是指程序从第一行语句开始执行，执行到最后一行语句结束，程序中的每条语句都会被执行一次。顺序结构的程序流程图如图 3.1 所示。

3.1.2 分支结构

分支结构也称选择结构，表示程序的处理步骤出现了分支，它需要根据某一特定的条件选择其中的一个分支执行。分支结构有单分支、双分支和多分支三种形式。

1. 单分支结构

当判断条件为真值时，执行语句块 1；当判断条件为假值时，越过语句块往下执行其他语句或结束，通常用来指定某一段语句是否执行。单分支结构的程序流程图如图 3.2 所示。

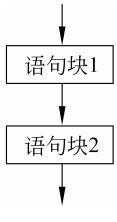

图 3.1 顺序结构程序流程图

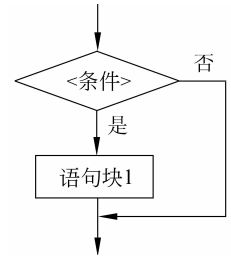

图 3.2 单分支结构程序流程图

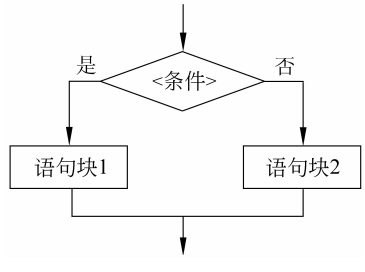

图 3.3 双分支结构程序流程图

2. 双分支结构

当判断条件为真值时，执行语句块 1；当判断条件为假值时，执行语句块 2，通常用来有选择地在两段语句中选择一段执行。双分支结构的程序流程图如图 3.3 所示。

3. 多分支结构

多分支结构即扩展的双分支结构，一般设有 n 个条件，n 或者 $n+1$ 个语句块，当判断

条件从上向下判断到某个为真值时，执行对应的语句块，然后退出多分支结构继续向下执行其他内容。需要注意的是，多分支结构在一次执行时只能选择一个分支，即使其他判断条件为真值，也不会继续判断执行。多分支结构的程序流程图如图 3.4 所示。

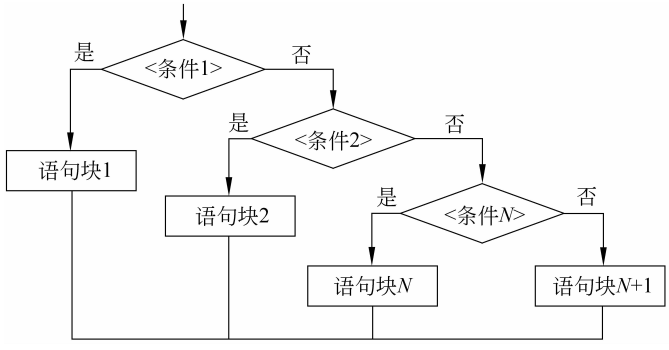

图 3.4　多分支结构程序流程图

3.1.3　循环结构

循环结构是程序根据条件判断，在满足条件的情况下反复执行某个语句块的运行方式。Python 中根据循环触发条件的不同，分为条件循环和遍历循环。条件循环结构在执行循环时先判断循环条件的取值，如果为 True 则执行一次语句块（循环体），然后返回继续判断循环条件，如果循环条件为 False，则结束循环。条件循环的程序流程图如图 3.5 所示。遍历循环在执行循环时也要进行判断，看循环变量是否在遍历队列中，如果在则取一个遍历元素，执行一次语句块（循环体），然后返回再取下一个遍历元素，直到遍历元素取完，循环结束。遍历循环的程序流程图如图 3.6 所示。

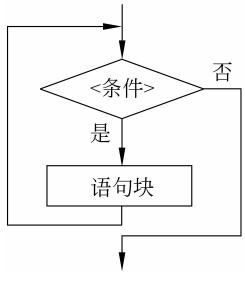

图 3.5　条件循环结构程序流程图

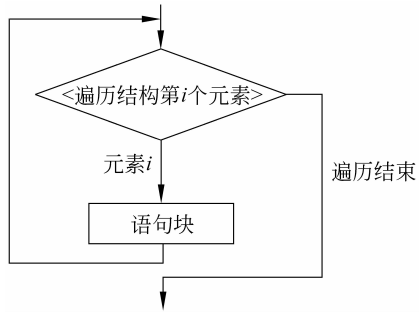

图 3.6　遍历循环结构程序流程图

3.2 分支结构

3.2.1 单分支结构

Python 中单分支 if 语句的语法格式如下：

```
if  <条件>:
<语句块>
```

条件是一个表达式，其结果一般为真值 True 或者假值 False。语句块是 if 条件满足后执行一条或多条语句的序列。程序执行到 if 语句时，如果判断到条件为 True，则执行语句块；若条件为假，则跳过语句块。不管条件为 True 还是 False，单分支执行结束后都会执行与 if 语句同级别的下一条语句继续执行程序。

分支结构中的条件在多数情况下是一个关系比较运算。形成比较的最常见方式是利用关系操作符。Python 语言共有 6 个关系操作符，如表 2.2 所示。

【例 3.1】 输入两个数字，输出其中较大的数字。

```
#example3.1
x=eval(input("请输入第一个数字:"))
y=eval(input("请输入第二个数字:"))
print("输入的两个数字为: ",x,y)
if x>y:
    print("较大的是:",x)
if x<y:
    print("较大的是:",y)
```

程序运行结果如下(输入 8 和 15)：

```
>>>
================RESTART:C:/Python/python36/example3.1.py=============
请输入第一个数字:8
请输入第二个数字:15
输入的两个数字为: 8 15
较大的是: 15
>>>
```

说明：

(1) 可以用 input()函数输入两个数字，然后对它们进行判断比较，如果某个数字比较大，则用 print()函数输出该数字。

(2) 运行后，程序对输入的两个数字进行判断，例如当输入的两个数字为 8 和 15 时，执行第一个 if 语句，判断条件 x>y 为假值 False，不执行 print("较大的是：",x)语句，分支判断结束；执行第二个 if 语句，判断条件 x<y 为真值 True，执行 print("较大的是：",y)语句，所以程序显示了“较大的是：15”。注意：如果输入的两个数字相等，由于该程序没有考虑到这种情况，则没有显示输出结果。

3.2.2　双分支结构

双分支结构是使用比较多的一种程序结构，Python 中双分支 if 语句的语法格式如下：

```
if  <条件>:
    <语句块 1>
else:
    <语句块 2>
```

双分支结构根据条件的真假值有选择地执行语句块 1 或语句块 2。当条件为 True 时，执行语句块 1，然后结束分支结构；当条件为 False 时，执行语句块 2，然后结束分支结构。

【例 3.2】 用双分支结构改写例 3.1。

```
#example3.2
x=eval(input("请输入第一个数字:"))
y=eval(input("请输入第二个数字:"))
print("输入的两个数字为: ",x,y)
if x>y:
    print("较大的是:",x)
else:
    print("较大的是:",y)
```

程序运行结果如下(输入顺序为 10 和 5)：

```
>>>
=========RESTART: C:/ Python/Python36/example3.2.py=========
```

```
请输入第一个数字:10
请输入第二个数字:5
输入的两个数字为: 10 5
较大的是: 10
>>>
```

说明：

(1) 如果不考虑两个数字相等的情况，可以将例 3.1 中的两个单分支 if 结构合并为一个双分支结构。

(2) 运行后，如果 x>y，则执行 if 结构下的 print("较大的是:",x)语句；否则执行 else 结构下的 print("较大的是:",y)语句。注意：如果输入的两个数字相等，使用此结构和例 3.1 的输出结果不同，因为使条件判断 x>y 为假的情况可能是 x<y，也可能是 x=y，所以一旦输入的两个数字相同，程序将显示“较大的是：y”的值。

3.2.3 多分支结构

多分支结构是对双分支结构的一种补充，当判断的条件有多个且判断结果有多个时，可用多分支 if 语句进行判断。多分支结构语法格式如下：

```
if  <条件 1>:
    <语句块 1>
elif<条件 2>:
    <语句块 2>
elif<条件 3>:
    <语句块 3>
…
else:
    <语句块 n>
```

程序执行时，会按照条件 n 的序列从上向下进行判断，当第一个条件 i 的值为 True，就执行该条件下的语句块，然后整个多分支 if 结构结束。如果没有任何条件为 True，则执行 else 下的语句块。注意：else 是可选的。

【例 3.3】 用多分支结构改写例 3.1，如果两个数字相等，也要给出说明。

```
#example3.3
x=eval(input("请输入第一个数字:"))
y=eval(input("请输入第二个数字:"))
```

```
print("输入的两个数字为：",x,y)
if x>y:
    print("较大的是:",x)
elif x<y:
    print("较大的是:",y)
else:
    print("这两个数字相等")
```

说明：

(1) 两个数字相互比较，只能有三种结果，即大于、小于、相等。

(2) 本例中有两个判断，在 x>y 和 x<y 这两种情况都不成立时，会执行 else 下的语句 print("这两个数字相等")。在多分支结构中，如果有多个判断都为 True，也只会执行第一个判断为 True 的分支中的语句块，执行后整个多分支结构结束，所以，哪个条件写在上，哪个条件写在下，有时会影响程序执行的结果。

【例 3.4】 输入一个分数，判断它对应到的学分绩点。90 分以上绩点为 4；80～90 分绩点为 3；70～79 分绩点为 2；60～69 分绩点为 1；60 以下绩点为 0。请问：以下两段代码哪一段是正确的。

程序 1 代码如下：

```
#example3.41
score=eval(input("请输入分数："))
if score>=90:
    gpa=4
elif score>=80:
    gpa=3
elif score>=70:
    gpa=2
elif score>=60:
    gpa=1
else:
    gpa=0
print("应得学分绩点为：",gpa)
```

程序 2 代码如下：

```
#example3.42
score=eval(input("请输入分数："))
```

```
if score<60:
    gpa=0
elif score>=60:
    gpa=1
elif score>=70:
    gpa=2
elif score>=80:
    gpa=3
else:
    gpa=4
print("应得学分绩点为：",gpa)
```

程序 1 运行结果如下(输入 85)：

```
>>>
=======RESTART: C:/Python/Python36/example3.41.py=======
请输入分数：85
应得学分绩点为：3
>>>
```

程序 2 运行结果如下(输入 85)：

```
>>>
=======RESTART: C:/ Python/Python36/example3.42.py=======
请输入分数：85
应得学分绩点为：1
>>>
```

说明：

(1) 经过比较可以看出，虽然两段程序都可以顺利执行，但是程序 2 并不符合题目的要求。

(2) 例如在输入 85 时，程序 1 可以得出 gpa 为 3，而程序 2 得到的 gpa 只有 1，原因就是在程序 2 中，进行 score>60 的判断结果为 True 时，将 gpa 赋值为 1，然后就结束了整个多分支结构。

使用多分支 if 语句时，一定要注意思路清晰，养成良好的程序书写风格，层次明确，便于阅读和修改程序。

3.2.4　分支结构的嵌套

如果一个 if 分支结构中包含另一个(或多个)if 分支,则称为分支结构的嵌套。

【例 3.5】　输入 3 个数字,利用分支结构对其进行降序排列输出。

```
#example3.5
a=eval(input("输入第 1 个数字"))
b=eval(input("输入第 2 个数字"))
c=eval(input("输入第 3 个数字"))
print("输入顺序为: ",a,b,c)
if a<b:
    a,b=b,a
if a<c:
    print("排序后为: ",c,a,b)
else:
    if c>b:
        print("排序后为: ",a,c,b)
    else:
        print("排序后为: ",a,b,c)
```

程序运行结果如下:

```
>>>
=======RESTART: C: /Python/Python36/example3.5.py =======
输入第 1 个数字 5
输入第 2 个数字 3
输入第 3 个数字 4
输入顺序为: 5 3 4
排序后为: 5 4 3
>>>
```

说明:

(1) 先比较输入的前两个数字 a 和 b 的大小,如果 a 小于 b,则交换 a 和 b 的顺序,第一个单分支结构保证了 a 一定要大于 b。

(2) 在双分支结构中,如果 a<c,那么 c 一定是最大的,按照 c、a、b 的顺序输出排序结果。如果 a>c,则再用一个双分支判断 b 和 c 的大小,如果 b>c,则按照 a、b、c 的顺序输出排序结果;如果 b<c,则按照 a、c、b 的顺序输出排序结果。

3.3 循环结构

循环结构是根据条件重复执行某些语句，它是程序设计中的一种重要结构。使用循环控制结构可以减少程序中的大量重复语句，从而编写出更简洁的程序。Python 提供了两种不同风格的循环结构，包括遍历循环 for 语句和条件循环 while 语句。

一般情况下，for 语句循环按给定的次数进行循环，而 while 语句循环是在条件满足时执行循环。

3.3.1 for 语句循环

遍历循环可以理解为让循环变量逐一使用一个遍历结构中的每个项目，遍历结构可以是字符串、列表、文件、range()函数等。for 语句循环语法格式如下：

```
for<循环变量>in<遍历结构>:
    <语句块>
```

for 语句的执行过程是：每次循环，判断循环变量值是否还在序列中，如果在，则取出该值提供给循环体内的语句使用；如果不在，则结束循环。

【例 3.6】 求 1 到 100 之间的奇数和。

程序 1 代码如下：

```
#example3.61
s=0
for i in range(1,101,2):
    s=s+i
print("1到100之间的奇数和为：",s)
```

程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.61.py=======
1到100之间的奇数和为：2500
>>>
```

程序 2 代码如下：

```
#example3.62
s=0
for i in range(1,101):
    if i %2==1:
        s=s+i
print("1 到 100 之间的奇数和为: ",s)
```

程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.62.py=======
1 到 100 之间的奇数和为: 2500
>>>
```

说明：

(1) 程序 1 定义一个初始值为 0 的变量 s，然后通过 range()函数产生 1 到 100 之间的奇数值，利用循环变量取遍这些值，然后将它们加到 s 中。

(2) 注意：range()函数可以产生某范围内的整数，如果只有一个参数，例如 range(5)，产生从 0 到 5 之间的整数，包括 0 但是不包括 5，即[0,4]，如果有两个参数，例如 range(3,7)，产生的数字范围为[3,6]；如果有三个参数，则第三个参数代表步长值，即从初值到终值过渡时每次加的数字，例如 range(5,12,2)产生的数字列表为[5,7,9,11]，range()函数也可以产生一个由大到小的数字列表，但是步长参数要为负数，例如 range(10,5,-1)产生的数字为[10,9,8,7,6]。

(3) 程序 1 中，for 语句执行时，range(1,101,2)函数产生遍历结构为[1,100]并且由 1 开始每次加数字 2，直到小于 101 结束，而循环变量 i 每次遍历其中一个数字，通过 s=s+i 语句累加到变量 s 中。

(4) 程序 2 定义一个初始值为 0 的变量 s，然后通过 range()函数产生 1 到 100 之间所有的整数，利用循环变量取遍这些值，再通过分支语句 if 判断这些值是不是奇数，如果是奇数，就加到 s 中。

(5) 程序 2 中，range(1,101)只负责产生[1,100]之间的整数，i 遍历取遍这些数字，由 if 语句判断这些数字中哪些是奇数，如果满足 i%2==1 这个条件，就累加到 s 中，虽然这段代码用了循环嵌套分支的结构，相比程序 1 复杂，但是这段代码更具有通用性，可以解决的问题种类更多。

for 语句循环还可以使用 else 关键字，其语法格式如下：

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

在这种结构中，当 for 循环正常执行后，程序会继续执行 else 语句中的内容。如果 for 循环因为某种原因没有正常执行完，如遇到了 break 语句，则不会执行 else 语句中的内容。所以通常用 else 检验 for 循环是否结束。例 3.6 中的程序 2 也可以改为以下格式：

```
#example3.62
s=0
for i in range(1,101):
    if i %2==1:
        s=s+i
else:
    print("计算完毕。",end="")
print("1 到 100 之间的奇数和为：",s)
```

程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.62.py=======
计算完毕。1 到 100 之间的奇数和为：2500
>>>
```

需要注意的是，在使用 else 的循环中，else 语句要和 for 对齐，而不是和 for 循环中的 if 语句对齐。

【例 3.7】 求字符串“Life is short，YOU need Python!”中有多少个字母“o”，不区分大小写字母。

```
#example3.7
n=0
str="Life is short, YOU need Python!"
for i in str:
    if i=="o" or i=="O":
        n=n+1
else:
```

```
    print("计算完毕。",end="")
print("字母 o 的个数为：",n)
```

代码执行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.7.py=======
计算完毕。字母'o'的个数为：3
>>>
```

说明：先设置一个初始值为 0 的变量 n，作为计数器，然后将字符串作为一个遍历结构，循环变量 i 会每次取字符串中的一个字母，再判断该字母是不是“o”，如果是，便将 n 增加 1。

3.3.2　while 语句循环

在明确知道循环的次数或者明确遍历结构时，一般使用 for 循环。但是更多时候无法明确遍历结构，或者不确定循环需要进行多少次。这时，就要使用 while 循环了。while 语句循环语法格式如下：

```
while<条件>:
    <语句块>
```

其中<条件>结果为 True 或者 False。当条件为 True，则执行一遍语句块，然后返回到 while 语句继续判断<条件>；当条件为 False 时循环结束。

和 for 一样，while 循环也有一种带有 else 的表达形式，语法格式如下：

```
while<条件>:
    <语句块 1>
else:
    <语句块 2>
```

在这种结构中，当 while 循环正常结束后，程序会继续执行 else 语句中的语句块 2，一般用来检验 while 循环是否结束。

【例 3.8】 猜价格。首先产生一个[10，100]之间的随机整数作为价格并赋予一个变量，如 n。然后用户可以输入数字猜价格，如果输入的数字大于 n 或者小于 n，则给予用户相应的提示；如果猜对了，则告诉用户猜中了。

```
#example3.8
from random import randint
n=randint(10,100)
print("商品价格已经产生,请输入 10 到 100 间的价格:")
bingo=False
while bingo==False:
    guess=eval(input("请输入您猜的价格:"))
    if guess>n:
        print("您输入的价格高于指定价格,请继续。")
    elif guess<n:
        print("您输入的价格低于指定价格,请继续。")
    else:
        print("恭喜您猜对了!价格为",guess)
        bingo=True
else:
    print("游戏结束!")
```

程序运行的结果如下:

```
>>>
=======RESTART: C: /Python/Python36/example3.8.py=======
商品价格已经产生,请输入 10 到 100 间的价格。
请输入您猜的价格: 80
您输入的价格低于指定价格,请继续。
请输入您猜的价格: 90
您输入的价格高于指定价格,请继续。
请输入您猜的价格: 85
您输入的价格低于指定价格,请继续。
请输入您猜的价格: 88
您输入的价格高于指定价格,请继续。
请输入您猜的价格: 87
恭喜您猜对了!价格为 87
游戏结束!
>>>
```

说明:

(1) 产生一个随机整数,可以引用函数库 random 中的 randint(m,n)函数,参数 m<=n,函数会产生[m,n]之间的随机整数。

(2) 当程序进入 while 循环时，判断条件 bingo=False 为真值，意味着开始循环。循环中使用 input()函数为变量 guess 赋值，然后用多分支 if 对 guess 进行判断，当 guess 的值大于或者小于 n，都给予文字提示，然后分支结束，返回 while 语句继续循环；当 guess 的值等于 n，多分支结构 if 结构执行 else 下的语句：输出“恭喜您猜对了!”，并且把 bingo 变量赋值为 True，分支结束。当返回 while 语句时，bingo=False 就变成了假值，循环结束，执行 while 语句同层的 else 下的语句。

(3) 本例中，循环是依靠一个条件“输入价格不等于初始产生的价格”进行的，由于用户输入的数字次数，也就是猜价格的次数未知，所以不能采用 for 语句循环结构。而 while 语句循环结构就可以处理这种未知循环次数的循环。

当然，while 语句循环也可以用来编写已知循环次数的循环。用 while 循环构造一个数字范围，一般来说是在 while 语句之前定义循环变量的初始值；在循环语句中指定循环变量的步长值；在 while 语句的条件处设置循环的终止值。

【例 3.9】 用 while 循环求 100 以内的奇数和。

```
#example3.9
s=0
i=1
while i<=100:
    if i%2==1:
        s=s+i
    i+=1
else:
    print("计算完毕。",end='')
print ("1 到 100 之间的奇数和为：",s)
```

程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.9.py=======
计算完毕。1 到 100 之间的奇数和为：2500
>>>
```

说明：i=1 的作用是定义循环变量的初始值，在 while 条件处设置 i<=100 为循环变量的终止值，在循环中 i+=1 语句和 i=i+1 等价，设置每次 i 增加的步长。

在编写 while 语句循环时，如果条件的计算结果一直为 True，而循环中没有 break 结束 while 循环，也就是没有逻辑出口，则循环将陷入永远执行的状态，称为死循环。例如

以下两行语句组成的程序，将一直输出数字 1。

```
while True:
    print(1)
```

在例 3.9 中，如果删除 i=i+1 语句，i 值在每次循环时都是 1，而循环条件是 i<=100，每次判断时都为 True，循环将始终执行。如果程序陷入死循环，在 IDLE 环境中可以按组合键 Ctrl+C，开发环境中会显示“KeyboardInterrupt”，程序终止。

3.3.3 循环的嵌套

在循环语句中使用另一个循环语句称为循环的嵌套，也称多重循环。for 语句循环和 while 语句循环都可以互相嵌套。利用循环的嵌套可以实现更复杂的程序设计。例如有如下代码段：

```
for i in range(1,4):
    for j in range(1,4):
        print("i 值为",i,";","j 值为",j)
```

可以看到，程序段的运行结果如下：

```
i 值为 1 ; j 值为 1
i 值为 1 ; j 值为 2
i 值为 1 ; j 值为 3
i 值为 2 ; j 值为 1
i 值为 2 ; j 值为 2
i 值为 2 ; j 值为 3
i 值为 3 ; j 值为 1
i 值为 3 ; j 值为 2
i 值为 3 ; j 值为 3
```

上层的 i 循环称为外层循环，下层的 j 循环称为内层循环，当外层循环执行一次时，内层循环就要整体循环一遍。从上面程序的结果中可以看出，当外层循环的循环变量 i 为 1 时，内层循环就要完成循环变量 j 从 1 到 3 的取值，当内层循环结束后，再次返回外层循环，i 值变为 2，内层循环再一次完整循环一次，以此类推。如果把两个循环加上 else 语句，则程序代码如下：

```
for i in range(1,4):
    for j in range(1,4):
        print("i 值为",i,";","j 值为",j)
    else:
        print("内层循环结束。")
else:
    print("外层循环结束。")
```

程序的运行结果如下：

```
i 值为 1 ; j 值为 1
i 值为 1 ; j 值为 2
i 值为 1 ; j 值为 3
内层循环结束。
i 值为 2 ; j 值为 1
i 值为 2 ; j 值为 2
i 值为 2 ; j 值为 3
内层循环结束。
i 值为 3 ; j 值为 1
i 值为 3 ; j 值为 2
i 值为 3 ; j 值为 3
内层循环结束。
外层循环结束。
```

可以看出，外层循环结束一次，而内层循环一共结束了 3 次。

【例 3.10】 解中国古代一道著名的数学题——百元百鸡问题。已知：公鸡 5 元 1 只；母鸡 3 元 1 只；小鸡 1 元 3 只。问要想 100 元正好买 100 只鸡，应该如何购买？

```
#example3.10
for x in range(1,21):
    for y in range(1,34):
        z=100-x-y
        if 5*x+3*y+z/3==100:
            print("公鸡",x,"母鸡",y,"小鸡",z)
else:
    print("计算完毕")
```

程序运行结果如下：

```
>>>
=======RESTART: C: \Python\Python36\例题 3.10.py=======
公鸡 4 母鸡 18 小鸡 78
公鸡 8 母鸡 11 小鸡 81
公鸡 12 母鸡 4 小鸡 84
计算完毕
>>>
```

说明：

(1) 如果用列方程的方法解决，可以知道只能根据鸡的数量等于 100 或者钱的数量等于 100 列出两个方程，而问题中有 3 个未知数，无法求得具体的值，在计算机中，可以采用穷举法求解，即对所有的可能解，逐个进行试验，若满足条件，就得到一组解，否则继续测试，直到循环结束为止。

(2) 假设公鸡有 x 只，则 100 元全部买公鸡的话，最多可以买 20 只，如果最少需要买 1 只公鸡，那么 x 的取值范围为 range(1,21)；假设母鸡有 y 只，则 100 元全部买母鸡的话，最多可以买 33 只，如果母鸡最少需要买 1 只，那么 y 的取值范围为 range(1,34)，利用双重循环组合 x 和 y 的值，当 x 和 y 确定为某一个数值后，小鸡的数量如果用 z 代表，则 z 的值为 $100-x-y$，每次循环都测试条件 $5*x+3*y+z/3=100$ 是否成立，如果条件成立，则找到一组合适的解。

和分支结构的嵌套一样，编写循环的嵌套时，要注意语句或者语句段的所属层次，认清是外层循环中的语句还是内层循环中的语句。

3.4 break 语句和 continue 语句

3.4.1 break 语句

Python 提供了一个提前结束循环的语句——break 语句。在循环中，执行到 break 语句时，可以结束本层的循环。一般来说，break 语句要放在一个分支结构中，当触发某个条件时，结束循环的运行。例如有如下代码：

```
for s in "python":
    for i in range(1,4):
        print(s,end="")
```

作用是把“python”中的每个字母都重复三遍，不换行输出。程序运行结果如下：

```
>>>
=======RESTART: C:/Python/Python36/temp.py=======
pppyyyttthhhooonnn
>>>
```

修改代码如下：

```
for s in "python":
    for i in range(1,4):
        if s=="h":
            break
        print(s,end="")
```

程序运行结果如下：

```
>>>
=======RESTART: C:/Python/Python36/temp.py=======
pppyyyttooonnn
>>>
```

说明：

(1) 在内层循环中，如果 s 等于字母 h，则跳出内层循环。但是外层循环将继续运行，程序依然会输出字母 h 以后的字母 o 和字母 n。

(2) 使用 break 时，要注意这条语句属于哪个循环。

【例 3.11】 求两个数字的最小公倍数。

```
x=eval(input("输入第一个数字"))
y=eval(input("输入第二个数字"))
if x<y:
    x,y=y,x
for i in range(x,x*y+1):
    if i%x==0 and i%y==0:
        print(x,"和",y,"的最小公倍数为",i)
        break
```

程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.11.py=======
```

```
输入第一个数字 5
输入第二个数字 10
10 和 5 的最小公倍数为 10
>>>
```

说明：

(1) 程序运行后，可以通过 input()函数输入两个数字给变量，例如 x 和 y，假设 x 大于 y。当 x 和 y 互质时，x 和 y 的最小公倍数为 $x*y$；当 x 能被 y 整除时，x 和 y 的最小公倍数为 x，即 x 和 y 的最小公倍数在区间$[x,x*y]$之间。

(2) 首先利用 if 分支判断 x 和 y 的大小，如果 $x<y$ 则交换 x 和 y 的值，这样保证了 $x>=y$。在 for 语句循环中，如果循环变量 i 能够被 x 和 y 整除，则 i 是 x 和 y 的公倍数，而不一定是最小公倍数，如当输入两个不互质的数字 10 和 5 时，如果代码省略了 break 语句，则程序运行结果如下：

```
>>>
=======RESTART: C: /Python/Python36/example3.11.py=======
10 和 5 的最小公倍数为 10
10 和 5 的最小公倍数为 20
10 和 5 的最小公倍数为 30
10 和 5 的最小公倍数为 40
10 和 5 的最小公倍数为 50
>>>
```

当使用了 break 时，循环变量 i 遍历到第一个满足 if 分支的数字，输出计算结果后循环就结束，程序只能输出一个数字，并且是最小公倍数。

3.4.2 continue 语句

continue 语句和 break 语句一样，用在循环结构中，用来结束循环的运行，但是 continue 语句只能结束本次循环的执行，而不终止循环。即当执行到 continue 语句时，程序会终止当前循环，并忽略循环中 continue 之后的语句，然后回到循环语句 for 或者 while，再次判断是否进行下一次循环。

【例 3.12】 输入 10 名同学的分数求及格同学的均值，如果分数低于 60，则不计入计算中。

程序 1 代码如下：