第5章 程序控制结构

在传统的面向过程程序设计中有3种经典的控制结构，即顺序结构、选择结构和循环结构，再加上一些方便程序编写的其他语句，一个实际工程项目的编程问题就有了语句基础了。即使是在面向对象程序设计语言以及事件驱动或消息驱动应用开发中，也无法脱离这3种基本的程序结构。可以说，不管是用哪种程序设计语言，在实际开发中，为了实现特定的业务逻辑或算法，都不可避免地要用到大量的选择结构和循环结构，并且经常需要将选择结构和循环结构嵌套使用。

5.1 条件表达式

在选择结构和循环结构中，都要使用条件表达式来确定下一步的执行流程。在Python中，单个常量、变量或者任意合法表达式都可以作为条件表达式。在条件表达式中可以使用的运算符包括：

(1) 算术运算符：+、-、*、/、//、%、**。

(2) 关系运算符：>、<、==、<=、>=、!=。

(3) 测试运算符：in、not in、is、is not。

(4) 逻辑运算符：and、or、not。

(5) 矩阵运算符：~、&、|、^、<<、>>。

(6) 位运算符：@。

在选择和循环结构中，条件表达式的值只要不是False、0(或0.0、0j等)、空值None、空列表、空元组、空集合、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价。从这个意义上讲，几乎所有的Python合法表达式都可以作为条件表达式，包括含有函数调用的表达式。例如：

例5-1 条件表达式示例。

```
>>> if 3:                    #使用整数作为条件表达式
    print 5
5
>>> a = [1,2,3]              #使用列表作为条件表达式
>>> if a:
    print a
[1, 2, 3]
>>> a = []
```

```
>>> if a:
    print a
else:
    print 'empty'
empty
>>> s = t = 0
>>> while s< = 10:          #使用关系表达式作为条件表达式
    t += s
    s += 1
>>> s = t = 0
>>> while True:             #使用常量 True 作为条件表达式
    t += s
    s += 1
    if s>10:
        break
>>> t = 0
>>> for s in range(0,11,1):
    t += s
>>> s
10
>>> t
55
```

注意：在 Python 中，条件表达式不允许使用赋值运算符“=”，避免误将关系运算符“==”写作赋值运算符“=”带来的麻烦，例如下面的代码，在条件表达式中使用赋值运算符“=”将抛出异常，提示语法错误。

```
>>> if a = 3:
SyntaxError: invalid syntax
```

5.2 顺序结构

顺序结构是所有程序设计语言中执行流程的默认结构。在一个没有选择结构和循环结构的程序中，程序是按照语句书写的先后顺序依次执行的。图 5-1 是一个顺序结构的流程图，它有一个入口、一个出口，依次执行语句 1 和语句 2。实现程序顺序结构的语句主要是赋值语句和内置的输入函数(input())和输出函数(print())。

5.2.1 赋值语句

1. 赋值语句的格式

基本赋值语句的格式：

<变量 1>, <变量 2>, …, <变量 n> = <表达式 1>, <表达式 2>, …, <表达式 n>

赋值语句的功能是分别将<表达式 1>,<表达式 2>,…,<表达式 n>的值赋给<变量 1>,<变量 2>,…,<变量 n>。

赋值语句还有增量赋值的形式：

```
<变量> += <表达式>
```

这种增量赋值语句等价于：

```
<变量> = <变量> + <表达式>
```

增量赋值语句不可以对多个变量增量赋值。可以用于增量赋值语句的运算符有＋＝、－＝、＊＝、/＝、//＝、＊＊＝、%＝、&＝、|＝、^＝、＞＞＝、＜＜＝。

赋值语句还可以写成下面的形式：

```
x = y = z = 1
```

该语句是将三个变量 x、y、z 都赋值为 1。

注意：Python 系统定义的对象是有类型的，变量没有类型。虽然通过赋值语句让某个变量得到表达式的值，但只是引用了这个对象的值(表达式的值)。所以，对于同一个变量，第一次通过赋值语句得到一个整数值，之后又可以通过赋值语句得到一个浮点类型的值。这就是变量没有类型，只是引用值的原因。

2. 赋值语句的应用

应用赋值语句的一个最经典的例子是交换两个变量的值。因为交换两个变量的值在后续内容中会经常用到，大量的实际问题中也需要交换两个变量的值。在其他程序设计语言中，这一段代码的经典写法是(使用第三方变量 t 暂存数据)：

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

Python 赋值语句的设计可以极其简单地完成交换两个变量的值的工作，只要一条语句即可解决：

```
>>> x, y = y, x
```

5.2.2 基本输入输出

数据的输入输出是应用程序不可缺少的功能。在 Python 3.x 中，数据的输入输出是通过调用函数来实现的，主要有 input()函数、print()函数。而在 Python 2.x 中，输入通过 input()函数，输出通过 print 语句。

1. input()函数

在 Python 语言中，使用 input()函数实现数据输入，input()函数的一般格式：

```
x = input('提示串')
```

例 5-2 input()函数示例。

```
>>> x = input('x =  ')
```

```
x =                          #直接输入 12.5,x 是一个数字的字符串
>>> x
12.5
>>> x = input('x')
x = 'abcd'                   #直接输入'abcd',x 是字符串'abcd'
>>> x
'abcd'
>>> x = float(input('x = '))
x =                          #直接输入 3.1415926,并转换为浮点型
>>> x
3.1415926
```

2. print 语句

print 语句可以采用格式化输出形式：

```
print '格式串' % (对象 1, 对象 2, ...)
```

其中，格式串用于指定后面输出对象的格式，格式串中可以包含随格式输出的字符，当然主要是对每个输出对象定义的输出格式。对于不同类型的对象采用不同的格式：

输出字符串：　%s
输出整数：　%d
输出浮点数：　%f
指定占位宽度：　%10s，%10d，%－10f(都是指定 10 位宽度)
指定小数位数：　%10.3f
指定左对齐：　%－10s，%－10d，%－10f，%－10.3f

例如：

例 5-3　print 语句示例。

```
>>> print '%10d%10d%10.2f' % (1234568, 87655, 12.34567890123)
   1234568     87655     12.35
>>> print '%-10d%-10d%-10.2f' % (1234568, 87655, 12.34567890123)
1234568   87655     12.35
```

5.2.3　案例精选

案例 5-1　已知三角形三条边的边长，求三角形的面积。

提示：三角形面积＝$\sqrt{p(p-a)(p-b)(p-c)}$，其中，a、b、c 是三角形三边的边长，p 是三角形周长的一半。

相关代码如下：

```
import math
a = float(input("请输入三角形的边长 a: "))
b = float(input("请输入三角形的边长 b: "))
c = float(input("请输入三角形的边长 c:"))
p = (a + b + c)/2     #三角形周长的一半
area = math.sqrt(p * (p - a) * (p - b) * (p - c))
```

```
print str.format("三角形三边分别为:a = {0}, b = {1}, c = {2}", a, b, c)
print str.format("三角形的面积 = {0}", area)
```

案例 5-1 运行结果如下：

```
请输入三角形的边长 a: 3
请输入三角形的边长 b: 4
请输入三角形的边长 c: 5
三角形三边分别为:a = 3.0, b = 4.0, c = 5.0
三角形的面积 = 6.0
```

5.3　选择结构

在顺序结构中，程序只能机械地从头运行到尾，要想使计算机变得更“智能”，就需要应用选择结构。所谓选择结构，就是按照给定条件有选择地执行程序中的语句。在 Python 语言中，程序选择结构有：单分支选择结构(if 语句)、双分支选择结构(if…else 语句)和多分支选择结构(if…elif 语句)。

5.3.1　单分支选择结构

单分支选择结构是最简单的一种选择形式，其语法如下所示：

```
if <表达式>:
   <语句块>
```

其中：

(1) 表达式是任意的数值、字符、关系或逻辑表达式，或用其他数据类型表示的表达式。它表示条件，以 True(1)表示真，False(0)表示假。

(2) 语句块称为 if 语句的内嵌语句或字句，内嵌语句严格地以缩进方式表达，编辑器也会提示程序员开始书写内嵌语句的位置，如果不再缩进，表示内嵌语句在上一行已经完成。

执行顺序：首先计算<表达式>的值，若<表达式>的值为 True，则执行内嵌语句，否则不做任何操作。if 语句的流程图如图 5-1 所示。

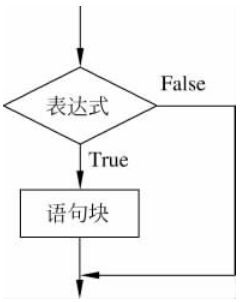

图 5-1　if 语句的流程图

当表达式值为 True 或其他等价值时，表示条件满足，语句块将被执行，否则该语句块将不被执行。

```
x = input('Input two numbers:')
a,b = map(int,x.split())
if a > b:
    a,b = b,a
print a,b
```

注意：表达式后面的冒号“:”是不可缺少的，表示一个语句块的开始，后面几种其他形式的选择结构和循环结构中的冒号也是必须有的。

5.3.2 双分支选择结构

双分支选择结构的语法为：

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

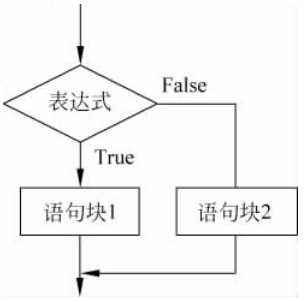

图 5-2 if…else 语句的流程图

执行顺序：首先计算<表达式>的值，若<表达式>的值为 True 或其他等价值时，执行<语句块 1>，否则执行<语句块 2>。if…else 语句的流程图如图 5-2 所示。

```
>>> a = ['1','2','3','4','5']
>>> if a:
        print a
    else:
        print 'Empty'
['1', '2', '3', '4', '5']
```

例 5-4 输入一个年份 year，判断是否为闰年。

分析：闰年的条件是：①能被 4 整除但不能被 100 整除；②能被 400 整除。

用逻辑表达式表示为：

```
(year % 4 == 0 and year % 100!= 0) or (year % 400 == 0)
```

程序代码如下：

```
year = input("输入年份:")                          # 可用 int()函数
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print year,":闰年"
else:
    print year,": 平年"
```

两次运行程序，程序运行结果如下：

```
输入年份:'2008'
2008 :闰年
输入年份:'2017'
2017 : 平年
```

Python 还支持如下形式的表达式：

```
value1 if condition else value2
```

当条件表达式 condition 的值与 True 等价时，表达式的值为 value1，否则表达式的值为 value2。另外，在 value1 和 value2 中还可以使用复杂表达式，包括函数调用和基本输出语句。下面代码演示了上述表达式的用法。

```
>>> a = 10
>>> b = 1 if a > 5 else 0
```

```
>>> b
1
```

5.3.3　多分支选择结构

多分支选择结构为用户提供了更多的选择，可以实现复杂的业务逻辑，多分支选择结构的语法为：

```
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>
```

其中，关键字 elif 是 else if 的缩写。

执行顺序：首先计算<表达式 1>的值，若其值为 True 或其他等价值时，执行<语句块 1>；否则，继续计算<表达式 2>的值，若其值为 True 或其他等价值时，执行<语句块 2>；以此类推，如果所有表达式的值都为 False，则执行<语句块 n+1>。if…elif 语句的流程图如图 5-3 所示。

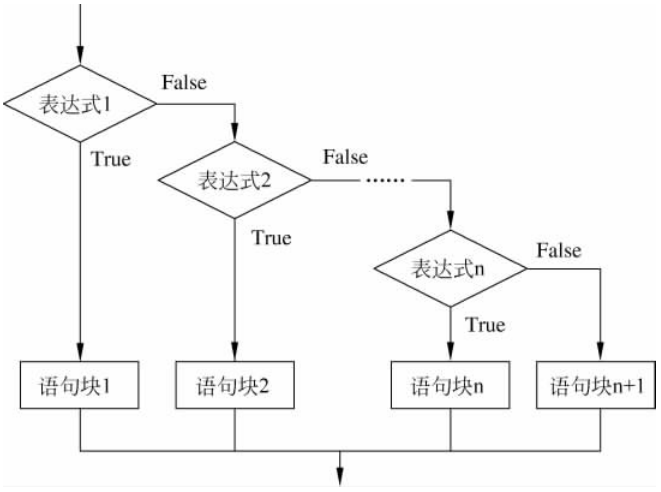

图 5-3　if…elif 语句的流程图

注意：

(1) 不管有几个分支，程序执行了一个分支以后，其余分支不再执行。

(2) 当多分支中有多个表达式同时满足条件时，则只执行第一条与之匹配的语句。

例 5-5　多分支选择 if…elif 语句实例。

```
score = int(input("please input your score:"))

if score >= 90:
```

```
    print "Grade A"
elif score >= 80:
    print "Grade B"
elif score >= 70:
    print "Grade C"
elif score >= 60:
    print "Grade D"
else:
    print "You fail in the test!"
```

运行本程序三次，运行结果如下：

```
please input your score:78
Grade C
please input your score:99
Grade A
please input your score:59
You fail in the test!
```

5.3.4 if 语句和 if…else 语句的嵌套形式

如果 if 语句和 if…else 语句的内嵌语句又是一个 if 语句或 if…else 语句，则称这种形式为 if 语句(或 if…else 语句)的嵌套形式。例如：

```
if <表达式 1>:
   if <表达式 2>:
      <语句块 1>
   else:
      <语句块 2>
[else:
   if <表达式 3>:
      <语句块 3>
   else:
      <语句块 4>]
```

如果上面的一般格式中没有方括号中的内容，就会出现 else 与哪个 if 匹配的问题，有可能导致语义错误，就是所谓的 else 悬挂问题。对于其他语言程序，如 C++，就要强制 else 与最近的 if 匹配。好在 Python 以严格的缩进方式表达匹配，不需要指定。

实际上，用 if 语句(或 if…else 语句)的嵌套形式完全可以替代 if…elif 语句。但从程序结构上讲，后者更清晰。

对于例 5-5，完全可用嵌套形式表达如下：

```
score = int(input("please input your score:"))
if score >=  60:

    if score >= 90:
        print "Grade A"
    elif score >= 80:
        print "Grade B"
```

```
    elif score >= 70:
        print "Grade C"
    elif score >= 60:
        print "Grade D"
else:
    print "You fail in the test!"
```

同样运行本程序三次，运行结果如下：

```
please input your score:78
Grade C
please input your score:99
Grade A
please input your score:59
You fail in the test!
```

由上可见，程序语言中的某些语句只是为了方便程序员写程序，不一定是必要的，但是程序中判断语句的层越多，程序的可读性就越差。如果在设计的时候出现了太多层，还是要想办法再重新构想一下比较简单的流程。

5.3.5　案例精选

案例 5-2　面试资格确认。

某公司招聘，要求至少满足以下条件中的两项：

(1) 不超过 25 岁；

(2) 计算机专业；

(3) 硕士研究生学历。

编写程序，判断 24 岁的计算机学士是否能获得面试机会。

参考代码如下：

```
age = 24
subject = "计算机"
degree = "学士"
if (age > 25 and degree == "硕士") or (degree == "硕士" and subject == "计算机") or (age <=
25 and subject == "计算机"):
    print '恭喜，您已获得我公司的面试机会!'
else:
    print '抱歉，您未达到面试要求'
```

案例 5-2 运行结果如下：

```
恭喜，您已获得我公司的面试机会!
```

案例 5-3　已知坐标点(x,y)，判断其所在的象限。

相关代码如下：

```
x = int(input("请输入 x 坐标:"))
y = int(input("请输入 y 坐标:"))
if(x == 0 and y == 0):print "位于原点"
elif(x == 0):print "位于 y 轴"
```

```
elif(y == 0):print "位于 x 轴"
elif(x > 0 and y > 0):print "位于第一象限"
elif(x < 0 and y > 0):print "位于第二象限"
elif(x < 0 and y < 0):print "位于第三象限"
else:print "位于第四象限"
```

案例 5-3 运行结果如下：

```
请输入 x 坐标: -2
请输入 y 坐标: 5
位于第二象限
```

案例 5-4 输入三个数，按照从大到小的顺序排序。

假设有三个数 a、b 和 c，先让 a 和 b 比较，使得 a＞b；然后比较 a 和 c，使得 a＞c，此时 a 最大；最后 b 和 c 比较，使得 b＞c。

```
a = int(input("请输入整数 a:"))
b = int(input("请输入整数 b:"))
c = int(input("请输入整数 c:"))
if(a<b): t=a; a=b; b=t
if(a<c): t=a; a=c; c=t
if(b<c): t=b; b=c; c=t
print "排序结果(降序):", a, b, c
```

案例 5-4 运行结果如下：

```
请输入整数 a: 5
请输入整数 b: 4
请输入整数 c: 6
排序结果(降序): 6 5 4
```

案例 5-5 判断某一年是否为闰年(例 5-4 的多种解决方法)。

判断闰年的条件是：年份能被 4 整除但不能被 100 整除，或者能被 400 整除，其判断流程参见图 5-3。

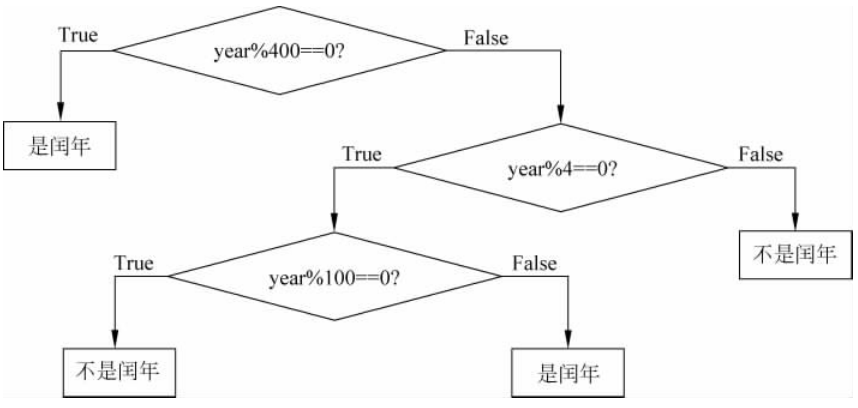

图 5-3 是否为闰年的判断条件

```
year = 2017
```

方法一：使用一个逻辑表达式包含所有的闰年条件。相关语句如下：

```
if(year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print "是闰年"
else:print("不是闰年")
```

方法二：使用嵌套的 if 语句。相关语句如下：

```
if(year % 400 == 0):print("是闰年")
else:
    if(year % 4 == 0):
        if(year % 100 == 0):print "不是闰年"
        else:print "是闰年"
    else:print "不是闰年"
```

方法三：使用 if…elif 语句。相关语句如下：

```
if(year % 400 == 0):print "是闰年"
elif(year % 4 != 0):print "不是闰年"
elif(year % 100 == 0):print "不是闰年"
else:print "是闰年"
```

方法四：使用 calendar 模块的 isleap 函数来判断闰年。相关语句如下：

```
import calendar
if(calendar.isleap(year)):print "是闰年"
else:print "不是闰年"
```

5.4 循环结构

所谓循环结构，就是按照给定规则重复地执行程序中的语句。实现程序循环结构的语句称为循环语句。Python 提供了两种基本的循环结构：while 循环和 for 循环。其中，while 循环一般用于循环次数难以提前确定的情况，当然也可以用于循环次数确定的情况；for 循环一般用于循环次数可以提前知道的情况，尤其适用于枚举或遍历序列或迭代对象中元素的场合，编程时一般建议优先考虑使用 for 循环。相同或不同的循环结构之间可以互相嵌套，也可以与选择结构嵌套使用，用来实现更为复杂的逻辑。

5.4.1 while 语句

while 语句用于实现当型循环结构，其特点是：先判断、后执行。

语法格式：

```
while <条件表达式>:
    <循环体>
```

其中：

(1) <条件表达式>称为循环条件，可以是任何合法的表达式，其值为 True 或 False，它

用于控制循环是否继续进行。

(2) <循环体>是要被重复执行的代码行。

执行顺序：首先判断条件<表达式>的值，若为 True，则执行<循环体>，继而再判断<条件表达式>，直至条件<表达式>值为 False 时退出循环，如图 5-4 所示。

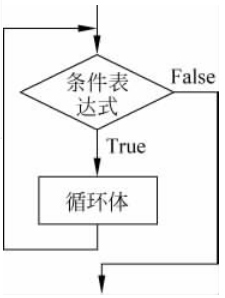

图 5-4　while 语句流程图

例 5-6　while 语句示例：求自然数 1～100 之和。

分析：这是一个累加求和的问题，循环结构的算法是，定义两个 int 变量，i 表示加数，其初值为 1；sum 表示和，其初值为 0。首先将 sum 和 i 相加，然后 i 增 1，再与 sum 相加并存入 sum，直到 i 大于 100 为止。

程序代码如下：

```
i = 1
sum = 0
while i <= 100:
    sum += i
    i += 1

print "sum = ", sum
```

程序运行结果如下：

```
sum = 5050
```

当应用 while 语句时，要注意以下几点：

(1) 在循环体中应该有改变循环条件表达式值的语句，否则将会造成无限循环。例如在例 5-6 中，如果没有语句 i＋＝1，那么 i 的值始终不发生变化，循环也就永远不会终止。

(2) 该循环结构是先判断后执行循环体，因此，若<条件表达式>的值一开始就是 False，则循环体一次也不执行，直接退出循环。

(3) 在设置循环条件时，要留心边界值，以免多执行一次或少执行一次。

例 5-7　while 语句示例：求出满足不等式 1＋2＋3＋…＋n≤100 的最大 n 值。

此不等式的左边是一个和式，该和式中的数据项个数是未知的，也正是要求出的。对于和式中的每个数据项，对应的通式为 i(i ＝ 1，2，3，…，n)，所以可以采用循环累加的方法计算出和式的和。设循环变量为 i，它应从 1 开始取值，每次增加 1，直到和式的值不大于 100 为止，此时的 i 值就是所求的 n。设累加变量为 s，在循环体内应把 i 的值累加到 s。

程序代码如下：

```
i = 0
s = 0
while s<100:
    i += 1
    s += i

print "n = ",i
```

程序运行结果如下：

```
n = 14
```

5.4.2 for 语句

for 语句的语法格式为：

```
for <变量> in <可迭代容器>:
    <语句块>
```

其中，<变量>可以扩展为变量表，变量与变量之间用“,”分开，<可迭代容器>可以是序列、迭代器或其他支持迭代的对象。

执行顺序：<变量>取遍<可迭代容器>中的每一个值。每取一个值，如果这个值在<可迭代容器>中，执行<语句块>，返回，再取下一个值，再判断，再执行，...，直到遍历完成或发生异常退出循环。

for 语句是 Python 语言提供的最强大的循环结构。for 语句主要用于访问序列和迭代器(iterator)。迭代器是一个可以标识序列中元素的对象。序列与迭代器的区别为：for 语句的语法格式中的<可迭代容器>可以直接是一个字符串、列表、元组等，还可以用一些函数产生序列或迭代器。如果函数产生的是序列，如 range()、sorted()函数，那么，for 语句访问的是序列。如果函数产生的是迭代器，如 enumerate()、reversed()、zip()函数，for 语句访问的是迭代器，例如：

```
>>> s = [0, 1, 2, 3, 100, 'ABC']
>>> s
[0, 1, 2, 3, 100, 'ABC']
>>> enumerate(s)
<enumerate at 0xa31d048>
>>> type(enumerate(s))
enumerate
>>> zip(s)
[(0,), (1,), (2,), (3,), (100,), ('ABC',)]
>>> type(zip(s))
list
>>> reversed(s)
<listreverseiterator at 0xa1f5fd0>
>>> type(reversed(s))
listreverseiterator
>>> s = ["XYZ", "Hello", "ABC", "Python"]
>>> sorted(s)
['ABC', 'Hello', 'Python', 'XYZ']
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x = sorted(s)
>>> x
['ABC', 'Hello', 'Python', 'XYZ']
>>> x[3]
'XYZ'
```

```
>>> y = range(10)
>>> y[8]
8
```

例 5-8 for 语句应用示例。

```
>>> s = ["XYZ", "Hello", "ABC", "Python"]          #使用序列迭代
>>> for i in s:
    print i
XYZ
Hello
ABC
Python
>>> s = ["XYZ", "Hello", "ABC", "Python"]          #使用序列索引迭代
>>> for i in range(len(s)):
    print i, s[i]
0 XYZ
1 Hello
2 ABC
3 Python
>>> x = range(5)
>>> for i in x:                                    #使用数字对象迭代
    print i,x[i]
0 0
1 1
2 2
3 3
4 4
>>> s = ["XYZ", "Hello", "ABC", "Python"]
>>> s1 = [200, 300, 1000, 500, 800]
>>> for x, y in zip(s, s1):                        #使用迭代器迭代
    print " %8s %8s" %(x,y)
   XYZ       200
   Hello     300
   ABC       1000
  Python     500
```

5.4.3 多重循环

多重循环又称为循环嵌套，是指在某个循环语句的循环体内还可以包含有循环语句。在实际应用中，两种循环语句不仅可以自身嵌套，还可以相互嵌套，嵌套的层数没有限制，呈现出多种复杂形式。在嵌套时，要注意在一个循环体内包含另一个完整的循环体。例如：

```
while <表达式 1>:
    …
    while <表达式 2>:
        <循环体>
    …
    for <变量> in <可迭代容器>:
        <循环体>
    …
```

例 5-9　编程输出“9×9 乘法表”。

```
>>> for i in range(1, 10):
>>>         for j in range(1,i+1):
>>>             print i, '*', j, '=', i*j
1 * 1 = 1
2 * 1 = 2
2 * 2 = 4
…… 
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

5.4.4　break、continue、pass、else 语句

1. break 语句

break 语句用在循环语句(迭代)中,结束当前的循环(迭代)跳转到循环语句的下一条。break 语句常常与 if 语句联合,满足某条件时退出循环(迭代)。

例 5-10　break 语句示例。

```
#输入一个数,判断是否为质数。
from math import *
x = input("输入一个数:")
i = 2
while i<=int(sqrt(x)):
    if x%i == 0:
        break
    i = i+1

if i>int(sqrt(x)):
    print x,":质数"
else:
    print x,":非质数"
```

运行程序两次,运行结果如下:

```
输入一个数:'4'
4 :非质数
输入一个数:'5'
5 :质数
```

2. continue 语句

continue 语句用在循环语句(迭代)中,忽略循环体内 continue 语句后面的语句,回到下

一次循环(迭代)。

图 5-5 是以 while 循环结构为例,说明执行含有 break 语句或 continue 语句的循环时流程变化的示意图。

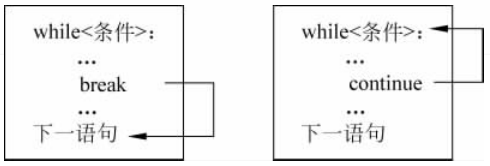

图 5-5 break 和 continue 语句的区别

例 5-11 continue 语句应用示例。

```
s = 0
for i in range(1,11):
     if i % 2 == 0:
          continue
     if i % 10 == 7:
          break
     s = s + i
print "s = ", s
```

程序运行结果如下:

```
s = 9
```

3. pass 语句

pass 语句可以用在任何地方,不做任何事情,只起到占位的作用,例如:

```
>>> pass
>>> if True:
    pass
>>> while 1:
    pass
```

4. else 语句

在 Python 语句中,else 语句还可以在 while 语句或 for 语句中使用。else 语句(块)写在 while 语句或 for 语句尾部,当循环语句或迭代语句正常退出(达到循环重点,或迭代完所有元素)时,执行 else 语句下面的语句块。这意味着 break 语句也会跳出 else 语句。

例 5-12 输入一个数,判断是否为质数。

```
from math import *
x = input("输入一个数:")
i = 2
while i <= int(sqrt(x)):
    if x % i == 0:
        print x," :非质数"
        break
```

```
    i = i + 1
else:
    print  x," :质数"
```

运行程序两次，运行结果如下：

```
输入一个数:'4'
4 :非质数
输入一个数:'5'
5 :质数
```

break 语句和 continue 语句在 while 循环和 for 循环中都可以使用，并且一般常与选择结构结合使用，以达到在特定条件得到满足时跳出循环的目的。一旦 break 语句被执行，将使得整个循环提前结束。continue 语句的作用是终止本次循环，并忽略 continue 之后的所有语句，直接回到循环的顶端，提前进入下一次循环。需要注意的是，过多的 break 和 continue 语句会严重降低程序的可读性。除非 break 和 continue 语句可以让代码更简单或更清晰，否则不要轻易使用。

5.4.5　案例精选

案例 5-6　求 1～100 能被 5 整除，但不能同时被 6 整除的所有整数。

```
t = []
for i in range(1, 101):
    if i % 5 == 0 and i % 6 != 0:
        t.append(i)
print t
```

案例 5-6 运行结果如下：

```
[5, 10, 15, 20, 25, 35, 40, 45, 50, 55, 65, 70, 75, 80, 85, 95, 100]
```

习题 5

一、单选题

1. 执行下列 Python 语句将产生的结果是(　　)。

```
x = 2; y = 2.0
if(x == y): print "Equal"
else: print "Not Equal"
```

A. Equal　　B. Not Equal　　C. 编译错误　　D. 运行时错误

2. 执行下列 Python 语句将产生的结果是(　　)。

```
i = 1
if(i):print True
else:print False
```

A. True　　B. False　　C. 1　　D. 运行时错误

3. 下面 if 语句统计满足“性别(gender)为男、职称(duty)为副教授、年龄(age)不大于40岁”条件的人数,正确的语句为(　　)。

A. if (gender==“男” or age<=40 and duty==“副教授”): n+=1

B. if (gender==“男” and age<=40 and duty==“副教授”): n+=1

C. if (gender==“男” and age<=40 or duty==“副教授”): n+=1

D. if (gender==“男” or age<=40 or duty==“副教授”): n+=1

4. Python 语句 x=True; y=False; z=False; print(x or y and z)的运行结果是(　　)。

A. True　　B. False

C. 1　　D. 运行时错误

5. 循环语句 for i in range(-3,21,4)的循环次数为(　　)。

A. 5　　B. 6　　C. 7　　D. 8

6. 若 k 为整型,下述 while 循环执行的次数为(　　)。

```
k = 1000
while k > 1:
    print k
    k = k/2
```

A. 9　　B. 10　　C. 11　　D. 1000

7. 下列语句不符合语法要求的表达式为(　　)。

```
for var in _____________:
    print var
```

A. range(0,10)　　B. “Hello”

C. (1,2,3)　　D. {1,2,3,4,5}

二、多项选择题

1. 以下 for 语句结构中,(　　)能完成 1～10 的累加功能。

A. sum=0; for i in range(10,0): sum+=i

B. sum=0; for i in range(1,11): sum+=i

C. sum=0; for i in range(10,0,-1): sum+=i

D. sum=0; for i in (10,9,8,7,6,5,4,3,2,1): sum+=i

E. sum=0; for i in (1,2,3,4,5,6,7,8,9,10): sum+=i

2. Python 提供了两种基本的循环结构为(　　)。

A. for 循环　　B. while 循环　　C. if 循环　　D. if…elif 循环

E. continue 循环

三、判断题

1. Python 语句 x=True; y=False; z=False; print x or y and z 的运行结果是 True。(　　)

2. 假设有表达式“表达式 1 or 表达式 2”,如果表达式 1 的值等价于 True,那么无论表达式 2 的值是什么,整个表达式的值总是等价于 True。(　　)

3. 已知 a＝3；b＝5；c＝6；d＝True，则表达式 not d or a＞＝0 and a＋c＞b＋3 的值是 False。（　　）

四、上机实践

1. 编写程序，计算 10＋9＋8＋…＋1。

2. 编写程序，计算 2＋4＋6＋…＋100。

3. 编写程序，输出 2000—3000 年之间的所有闰年。请思考有哪几种实现方法。