控制流

1. if 语句

`if` 语句用来检验一个条件， 如果 条件为真，我们运行一块语句（称为 if-块 ）， 否则 我们处理另外一块语句（称为 else-块 ）。 else 从句是可选的。

```
#!/usr/bin/python
# Filename: if.py

number = 23
guess = int(raw_input('Enter an integer : '))

if guess == number:
    print 'Congratulations, you guessed it.' # New block starts here
    print "(but you do not win any prizes!)" # New block ends here
elif guess &lt; number:
    print 'No, it is a little higher than that' # Another block
    # You can do whatever you want in a block ...
else:
    print 'No, it is a little lower than that'
    # you must have guess &gt; number to reach here

print 'Done'
# This last statement is always executed, after the if statement is
executed
```

输出

```
$ python if.py
Enter an integer : 50
No, it is a little lower than that
Done
$ python if.py
Enter an integer : 22
No, it is a little higher than that
Done
$ python if.py
Enter an integer : 23
Congratulations, you guessed it.
(but you do not win any prizes!)
```

```
Done
```

在这个程序中，我们从用户处得到猜测的数，然后检验这个数是否是我们手中的那个。我们把变量 `number` 设置为我们想要的任何整数，在这个例子中是 `23`。然后，我们使用 `raw_input()`函数取得用户猜测的数字。函数只是重用的程序段。我们将在下一章学习更多关于函数的知识。
我们为内建的 `raw_input` 函数提供一个字符串，这个字符串被打印在屏幕上，然后等待用户的输入。一旦我们输入一些东西，然后按**回车**键之后，函数返回输入。对于 `raw_input` 函数来说是一个字符串。我们通过 `int` 把这个字符串转换为整数，并把它存储在变量 `guess` 中。事实上，`int` 是一个类，不过你想在对它所需了解的只是它把一个字符串转换为一个整数（假设这个字符串含有一个有效的整数文本信息）。
接下来，我们将用户的猜测与我们选择的数做比较。如果他们相等，我们打印一个成功的消息。注意我们使用了缩进层次来告诉 Python 每个语句分别属于哪一个块。这就是为什么缩进在 Python 如此重要的原因。我希望你能够坚持“每个缩进层一个制表符”的规则。你是这样的吗？

注意 `if` 语句在结尾处包含一个冒号——我们通过它告诉 Python 下面跟着一个语句块。
然后，我们检验猜测是否小于我们的数，如果是这样的，我们告诉用户它的猜测大了一点。我们在这里使用的是 `elif` 从句，它事实上把两个相关联的 `if else-if else` 语句合并为一个 `if-elif-else` 语句。这使得程序更加简单，并且减少了所需的缩进数量。
`elif` 和 `else` 从句都必须在逻辑行结尾处有一个冒号，下面跟着一个相应的语句块（当然还包括正确的缩进）。
你也可以在一个 if 块中使用另外一个 `if` 语句，等等——这被称为嵌套的 `if` 语句。

记住，`elif` 和 `else` 部分是可选的。一个最简单的有效 `if` 语句是：

```
if True:
    print 'Yes, it is true'
```

在 Python 执行完一个完整的 `if` 语句以及与它相关联的 `elif` 和 `else` 从句之后，它移向 `if` 语句块的下一个语句。在这个例子中，这个语句块是主块。程序从主块开始执行，而下一个语句是 `print 'Done'`语句。在这之后，Python 看到程序的结尾，简单的结束运行。
尽管这是一个非常简单的程序，但是我已经在这个简单的程序中指出了许多你应该注意的地方。所有这些都是十分直接了当的（对于那些拥有 C/C++背景的用户来说是尤为简单的）。它们在开始时会引起你的注意，但是以后你会对它们感到熟悉、“自然”。

给 C/C++程序员的注释 在 Python 中没有 `switch` 语句。你可以使用 `if..elif..else` 语句来完成同样的工作（在某些场合，使用字典会更加快捷。）

2. while 语句

只要在一个条件为真的情况下，`while`语句允许你重复执行一块语句。`while`语句是所谓 循环 语句的一个例子。`while`语句有一个可选的`else`从句。

```
#!/usr/bin/python
# Filename: while.py

number = 23
running = True

while running:
    guess = int(raw_input('Enter an integer : '))

    if guess == number:
        print 'Congratulations, you guessed it.'
        running = False # this causes the while loop to stop
    elif guess &lt; number:
        print 'No, it is a little higher than that'
    else:
        print 'No, it is a little lower than that'
else:
    print 'The while loop is over.'
    # Do anything else you want to do here

print 'Done'
```

输出

```
$ python while.py
Enter an integer : 50
No, it is a little lower than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

它如何工作

在这个程序中，我们仍然使用了猜数游戏作为例子，但是这个例子的优势在于用户可以不断的猜数，直到他猜对为止——这样就不需要像前面那个例子那样为每次猜测重复执行一遍程序。这个例子恰当地说明了`while`语句的使用。

我们把`raw_input`和`if`语句移到了`while`循环内，并且在 while 循环开始前把`running`变量设置为`True`。首先，我们检验变量`running`是否为`True`，然后执

行后面的 while-块 。在执行了这块程序之后，再次检验条件，在这个例子中，条件是 `running` 变量。如果它是真的，我们再次执行 while-块，否则，我们继续执行可选的 else-块，并接着执行下一个语句。
当 `while` 循环条件变为 `False` 的时候，`else` 块才被执行——这甚至也可能是在条件第一次被检验的时候。如果 `while` 循环有一个 `else` 从句，它将始终被执行，除非你的 `while` 循环将永远循环下去不会结束！
`True` 和 `False` 被称为布尔类型。你可以分别把它们等效地理解为值 `1` 和 `0`。在检验重要条件的时候，布尔类型十分重要，它们并不是真实的值 `1`。
else 块事实上是多余的，因为你可以把其中的语句放在同一块（与 `while` 相同）中，跟在 `while` 语句之后，这样可以取得相同的效果。
给 C/C++程序员的注释 记住，你可以在 `while` 循环中使用一个 `else` 从句。

3. for 循环

`for..in` 是另外一个循环语句，它在一序列的对象上 递归 即逐一使用队列中的每个项目。我们会在后面的章节中更加详细地学习序列。

```
#!/usr/bin/python
# Filename: for.py

for i in range(1, 5):
    print i
else:
    print 'The for loop is over'
```

输出

```
$ python for.py
1
2
3
4
The for loop is over
```

它如何工作

在这个程序中，我们打印了一个 序列 的数。我们使用内建的 `range` 函数生成这个数的序列。
我们所做的只是提供两个数，`range` 返回一个序列的数。这个序列从第一个数开始到第二个数为止。例如，`range(1,5)`给出序列`[1, 2, 3, 4]`。默认地，`range` 的步长为 1。如果我们为 `range` 提供第三个数，那么它将成为步长。例如，`range(1,5,2)`给出`[1,3]`。记住，range 向上 延伸到第二个数，即它不包含第二个数。

for 循环在这个范围内递归——for i in range(1,5)等价于 for i in [1, 2, 3, 4]，这就如同把序列中的每个数（或对象）赋值给 i，一次一个，然后以每个 i 的值执行这个程序块。在这个例子中，我们只是打印 i 的值。
记住，else 部分是可选的。如果包含 else，它总是在 for 循环结束后执行一次，除非遇到 break 语句。
记住，for..in 循环对于任何序列都适用。这里我们使用的是一个由内建 range 函数生成的数的列表，但是广义说来我们可以使用任何种类的由任何对象组成的序列！我们会在后面的章节中详细探索这个观点。
给 C/C++/Java/C#程序员的注释 Python 的 for 循环从根本上不同于 C/C++ 的 for 循环。C#程序员会注意到 Python 的 for 循环与 C#中的 foreach 循环十分类似。Java 程序员会注意到它与 Java 1.5 中的 for (int i : IntArray)相似。 在 C/C++中，如果你想要写 for (int i = 0; i < 5; i++)，那么用 Python，你写成 for i in range(0,5)。你会注意到，Python 的 for 循环更加简单、明白、不易出错。

4. break 语句

break 语句是用来 终止 循环语句的，即哪怕循环条件没有称为 False 或序列还没有被完全递归，也停止执行循环语句。
一个重要的注释是，如果你从 for 或 while 循环中 终止 ，任何对应的循环 else 块将不执行。

```
#!/usr/bin/python
# Filename: break.py

while True:
    s = raw_input('Enter something : ')
    if s == 'quit':
        break
    print 'Length of the string is', len(s)
print 'Done'
```

输出

```
$ python break.py
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something :       use Python!
Length of the string is 12
Enter something : quit
```

```
Done
```

它如何工作

在这个程序中，我们反复地取得用户地输入，然后打印每次输入地长度。我们提供了一个特别的条件来停止程序，即检验用户的输入是否是`'quit'`。通过终止 循环到达程序结尾来停止程序。
输入字符串的长度通过内建的`len`函数取得。
记住，`break`语句也可以在`for`循环中使用。

G2 的 Python 诗

我在这里输入的是我所写的一段小诗，称为 **G2 的 Python 诗**：

```
Programming is fun
When the work is done
if you wanna make your work also fun:
      use Python!
```

5. continue 语句

`continue`语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后 继续进行下一轮循环。

```
#!/usr/bin/python
# Filename: continue.py

while True:
    s = raw_input('Enter something : ')
    if s == 'quit':
        break
    if len(s) &lt; 3:
        continue
    print 'Input is of sufficient length'
    # Do other kinds of processing here...
```

输出

```
$ python continue.py
Enter something : a
Enter something : 12
Enter something : abc
Input is of sufficient length
Enter something : quit
```

它如何工作

在这个程序中，我们从用户处取得输入，但是我们仅仅当它们有至少 3 个字符长的时候才处理它们。所以，我们使用内建的`len`函数来取得长度。如果

长度小于 3，我们将使用 `continue` 语句忽略块中的剩余的语句。否则，这个循环中的剩余语句将被执行，我们可以在这里做我们希望的任何处理。
注意，`continue` 语句对于 `for` 循环也有效。