Full Circle 中文杂志

——之 python 教程二（猴哥稍微修改版，欢迎修正）

在上一期里，我们使用“raw_input”函数设计了一个简单的交互式程序，用“for”语句打印了一个循环，并也概览了基本的变量类型。现在，我们再次深入地了解变量的作用。

列表

我们先看到另一种变量类型，我们称之为“列表（List）”。这与其他语言不同的是，Python 并不将列表当作数组进行处理。我们这次再次用盒子来作 比，而一个列表便是若干个并列的盒子，我们可以将各种物品分别放在不同的盒子里。下面是一个简单的例子，用列表制作一个日历，代码如下:

```
months =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
```

在这个列表中，我们将所有的变量使用方括号括起来，并命名为“months”。接着，我们在 Python Shell 中输入类似于“print months[0]”、“months[1]”的代码，解释器将分别输出“Jan”和“Feb”。当然，要注意这里要从 0 开始算起。再用命令看看列表的长度:

```
print len(months)
```

解释器返回的结果是“12”。

再举一个类似的例子，用列表表达一个菜谱的分类目录:

```
categories = ['Main dish','Meat','Fish','Soup','Cookies']
```

这时，“categories[0]”将指代“Main dish”，以此类推 categories[4]便是“Cookies”。现在您可以实践一下，用清单列出您所想的事物。

通过刚才的讲解，我们已经学会了把若干个字符串组成列表。当然也同样可以将整数组成列表。返回到刚才日历的例子，我们可以再将每个月的天数 列出来:

```
DaysInMonth = [31,28,31,30,31,30,31,31,30,31,30,31]
```

现在，如果您输入“print DaysInMonth[1]”(二月)，解释器就会输出二月份的天数“28”。先注意我们将列表命名为“DaysInMonth”，其实我们可以随意改 为“daysinmonth”，甚至是一个“X”，但这样并不利于代码阅读。良好的编程规范（主要针对解释型语言）建议我们使用一个易于理解名称对变量进 行命名。稍后我们会解释一下原因，现在先来看看别的例子。

在我们进入下一个例程之前，我们先看看关于 Python 的其他问题。

进一步的了解字符串

在上一节中，我们简单地论及了字符串。现在让我们更进一步地了解字符串。故名思义，所谓字

符串就是一连串的字符。事实上，您可以把它看作是由多个字符组成 的数组。例如，我们将“strng”变量赋值为字符串“The time has come”，然后若想知道该字符串中的第二个字符是什么，我们可以输入：

```
strng = 'The time has come'
print strng[1]
```

解释器返回的结果是“h”——不要忘了在 Python 中，一律从 0 算起。所以“[0]”才是第一个字符，第二个是“[1]”，第三个是“[2]”，以此类 推。再者，如果您想得到从[4]到[8]之间的字符，我们可以输入：

```
print strng[4:8]
```

这次返回的结果是“time”，就像在上节中的循环那样，解释器总是终止在第八位，但并不会返回在第八位的空格。

同样，我们可以用“len()”函数查看字符串的长度，如：

```
print len(strng)
```

解释器返回的结果是 17，如果我们要查看字符串“time”的位置，我们可以使用以下命令：

```
pos = strng.find('time')
```

现在变量“pos”（英文方位一词的缩略词）赋值为 4，意指“time”从这一字符串的第四位开始。但是，如果我们让查询函数查询一个字符串中不存在的字 符，例如：

```
pos = strng.find('apples')
```

这时“pos”的返回值为“-1”。

我们也可以通过“split”函数获取在字符串中的每一个单词。我们现在用以下语句在字符串的每一个空格符处打断字符串：

```
print strng.split(' ')
```

最后返回的是一个列表“['The', 'time', 'has', 'come']”。这是一个非常实用的函数。在 Python 语言中，类似的字符串函数还有很多，我们接下来会细细讲解。

文字替换

在我们进入下一个例程之前，不得不插一个题外话。当我们用“print”打印变量等东西的时候，可以巧用文字替换。这个其实也非常简单，如果我们想替换一 个字符串，我们可以用“%s”来指代它<译者注：与此相对应的，“%d”是用于指代整数的>，并告诉 Python 解释器需要替换的东西。例 如，我们想输出从我们刚才的日历中输出其中一个月份，我们可以输入以下命令：

（转下一页）

```
print 'Month = month[0]
```

结果解释器打印出“Month = January”。如果我们想用“%d”替换一个整数，可以用以下一组代码<译者注：在 Python 中，所指代的符号，和被指代的符号是相对应的。 例如，在这一例程中，第一个所指代的符号是“%s”，则相对应第一个被指代的“Months[cntr]”。这是从左到右顺序推起，故不必担心混淆。具体 这一疑惑，请参见本文倒数第二自然段的说明>:

```
Months =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
DaysInMonth = [31,28,31,30,31,30,31,31,30,31,30,31]
for cntr in range(0,12):
       print '%s has %d days.' % (Months[cntr],DaysInMonth[cntr])
```

所返回的结果是:

```
Jan has 31 days.
Feb has 28 days.
Mar has 31 days.
Apr has 30 days.
May has 31 days.
Jun has 30 days.
Jul has 31 days.
Aug has 31 days.
Sep has 30 days.
Oct has 31 days.
Nov has 30 days.
Dec has 31 days.
```

这一例程的要点在于单双引号的使用。如果您将一个变量赋值为一个字符串，例如:

```
st = 'The time has come'
```

或者像这样:

```
st = “The time has come”
```

最后返回的结果是相同的，然而，如果在字符串中包含单引号:

```
st = 'He said he's on his way'
```

解释器就会报错，您需要将代码改为:

```
st = “He said he's on his way”
```

简单思考一下，不难发现在赋值一个字符串时，必须要用其中一种引号将其前后引起来。如果您需要混合引号，可以用另一种在字符串中没有的引号引起来。说到这 里，您可能会问了，如果我需要解释的字符串像“She said “Don't Worry”” 那样，拥有两种括号的字符串怎么办？别着急，您可以这样做:

```
st = 'She said "Don\'t Worry"'
```

您可能立刻注意到“Don't”中的单引号前的反斜杆。这个称为换码符或是转义符，在这个例程中，它的作用是告诉 Python 直接打印单引号，而不将它作 为定义符号。像这样的符号还有很多，例如用“\n”代表换行，用“\t”代替制表符，这些在接下来的例程中会用到。

赋值与等于

现在我们需要了解更多的特性，为接下来的学习做一个铺垫。首先必须要区别开来， 赋值与等于根本不能混为一谈。在我们的例程中，常用到赋值，当我们赋值的时候，总会用等号等赋值操作符（亦称赋值算符）<译者注：如果要使下面的代 码能正确通过，必须保证在上文中已对“value”赋过值。该代码的意思是将“variable”变量赋值为“value”变量的值>：

```
variable = value
```

然而，当我们要运算一个变量的值,就必须使用比较运算符。如果我们想检测两个不同变量变量是否相等，我们可以用上“==”：

```
variable == value
```

所以，例如我们想检测一个名叫“loop”的变量是否等于 12，可以用：

```
if loop == 12:
```

您可能已经注意到这一例程的 if 以及末尾的冒号，但且不要管它们。只要记住我们必须使用比较运算符（双等于号）才能使这个语句正常运作。

注释

接下来要讲的是在代码中的注释。在很多时候，注释是一件很重要的事情。其意义不仅仅是向他人解释您的思路，如果您将您的代码搁置在一旁一年半载，当您想继 续完成之前的程序，那么注释就会帮上您的大忙。Python 语言允许您注释掉代码中的某一行。您只需在需要注释掉的那一行代码前加一个“#”符号，例如：

```
# This is a comment
```

您可以将“#”号放在代码行的任何位置，Python 将不会去解释“#”号之后的代码。

“If”语句

在上文中，我们曾一笔带过地讲了一些关于“If”语句的问题。如果我们需要对一个变量的值做出判断，我们可以用“If”语句：

```
if loop == 12:
```

这行将检验“loop”变量是否等于“12”。接下来，我们再花时间理清下思路，如果我们想检测一个变量是否于另一个值相等，如果相等，执行一条命令，如 果不同则执行另一条命令，用白话文来讲就是这样：

```
if x == y then
       do something
else
       do something else
```

我们再将之前的想法应用到 Python 中是这样的：

```
if x == y:
       do something
else:
       do something else
       more things to do
```

根据上文所讲，总结要点如下：

1. 在判断行末尾加上冒号
2. 注意您的代码缩进

假若您要判断的不止一个，您可以用“if……elif……else”语句，例如<译者注：在以下的代码中，同时符合了，“x < 6”和“x < 10”两个条件，但是因为 Python 在发现一个条件符合后，就会将接下来的其他条件忽略。因此只会打印出代码顺序中最靠前的一种正确情况，故在此只会输 出“X is less than 6”>：

```
x = 5
if x == 1:
       print 'X is 1'
elif x < 6:
       print 'X is less than 6'
elif x < 10:
       print 'X is less than 10'
else:
       print 'X is 10 or greater'
```

您可能注意到我们使用了小于号来表达“如果“x”小于我们指定的值（这里的 6 和 10）……”与此类似的比较运算符还有大于号“>”，小于或等于号 “<=”，大于或等于号“>=”，以及不等号“!=”。

"While"语句

最后，我们要看看一个关于“While”语句的简单例程。“While”语句可用于反复循环一个步骤，直到已达预定的条件。下面的代码将变量“loop” 赋值为 1，并设置当“loop”的值小于或等于 10 时，继续循环，打印出“loop”的值，然后再在原有的值的基础上加 1，直到“loop”的值达到 10：

```
loop = 1
while loop <= 10:
       print loop
       loop = loop + 1
```

解释器打印到终端模拟器的内容是：

```
1
2
3
4
5
6
7
8
9
10
```

这就是我们所希望达到的目的。下面的例程与此相类似，但相比之下稍复杂一些：

```
loop = 1
while loop == 1:
       response = raw_input("Enter something or 'quit' to end => ")
       if response == 'quit':
              print 'quitting'
              loop = 0
       else:
              print 'You typed %s' % response
```

这个例子里，我们将前面所学到的“If”判断和“While”循环、“raw_input”函数、代码缩进、赋值操作符、比较运算符有机结合在这 8 行代码上。

我们试运行一下，看看效果：

```
Enter something or 'quit' to end
 => FROG
You typed FROG
Enter something or 'quit' to end
 => bird
You typed bird
Enter something or 'quit' to end
 => 42
You typed 42
Enter something or 'quit' to end
 => QUIT
You typed QUIT
Enter something or 'quit' to end
 => quit
quitting
```

最后，我们还是要提醒大家一下，如果您输入的是“QUIT”，这一程序是不会终止的。因为我们将“response”变量赋值为小写的 “quit”。在机器看来，“QUIT”与“quit”并不相关。

在我们结束本期的教程前，我们在来看一个简单的例程。通过这个例程，我们可以检测一个用户是否有足够的权限。可是，如果真的要将其应用开 来，这个小程序尚不完善。所以我们的目的只是为了检测您是否已经掌握了这两期的课程。简单的说，这一代码将要求用户输入他们的帐户密码，然后检测所输入的 是否与我们在代码中预设的匹配，最后据此作出回答。我们在代码中建立了两个列表，分别保存帐号和密码。然后用“raw_input”函数获取用户输入的帐 户密码，最后用“if……elif……else……”语句检测其是否正确。说到底，这是一个很不成熟的方案，我们会在接下来的教程中完善它：

```
#---------------------------------------------
#password_test.py
#    example of if/else, lists, assignments,raw_input,
#    comments and evaluations
#---------------------------------------------
# Assign the users and passwords
users = ['Fred','John','Steve','Ann','Mary']
passwords = ['access','dog','12345','kids','qwerty']
#---------------------------------------------
# Get username and password
usrname = raw_input('Enter your username => ')
pwd  = raw_input('Enter your password => ')
#---------------------------------------------
# Check to see if user is in the list
if usrname in users:
       position = users.index(usrname) #Get the position in the list of the users
       if pwd == passwords[position]:  #Find the password at position
              print 'Hi there, %s.  Access granted.' % usrname
       else:
              print 'Password incorrect.  Access denied.'
else:
       print "Sorry...I don't recognize you.  Access denied."
```

保存代码为“password_test.py”，并进行测试。

在这里我还有补充几句，在检测列表时，通常以“if usrname in users:”引启。它所作的是检查用户是否在列表中存在。然后用“users.index(usrname)”来查找该用户在 users 列表中对应的位置（position），得到这个位置后就可以通过 passwards[position]得到 passwords 列表中对应位置的密码。例如 “John”在 users 列表位置 1，“dog”在 password 列表位置 1，那么用户“John”跟密码“dog”就可以配成对了。这应该相当容易理解。

好了，本期的教程就到此结束了。好好利用所学过的知识制作更有趣的程序吧！ 在下一期中，我们将会讲讲 Python 的函数以及模块。