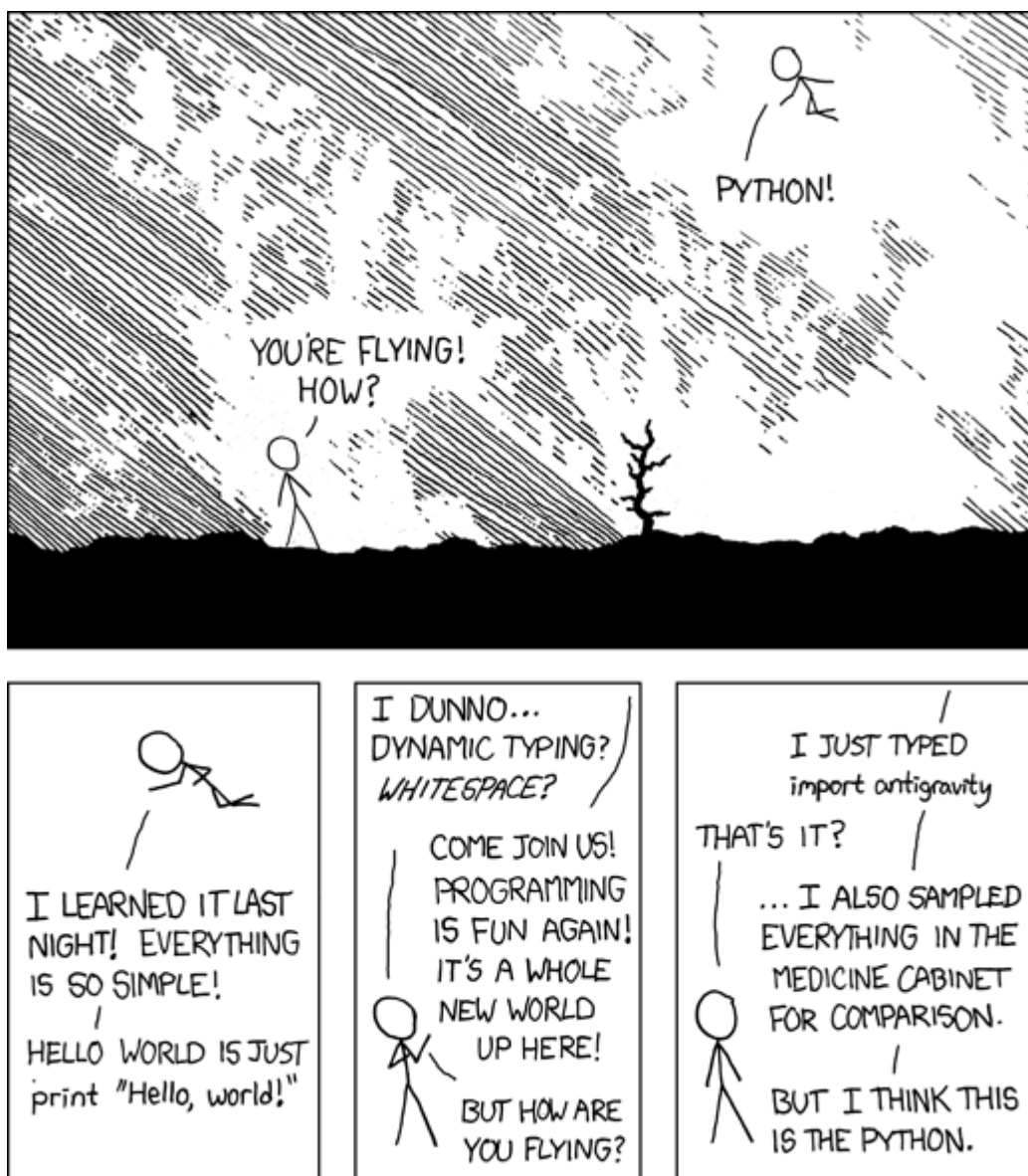


【Python 第 1 课】安装	4
【Python 第 2 课】print	5
【Python 第 3 课】IDE	8
【Python 第 4 课】输入	10
【Python 第 5 课】变量	12
【Python 第 6 课】bool	14
【Python 第 7 课】if	17
【Python 第 8 课】while	21
【Python 第 9 课】random	24
【Python 第 10 课】变量 2	26
【Python 第 11 课】逻辑判断	27
【Python 第 12 课】for 循环	28
【Python 第 13 课】字符串	30
【Python 第 14 课】字符串格式化	32
【Python 第 15 课】循环的嵌套	33
【Python 第 16 课】字符串格式化 2	35
【Python 第 17 课】类型转换	36
【Python 第 18 课】bool 类型转换	38
【Python 第 19 课】函数	39
【Python 第 21 课】函数的参数	44
【Python 第 22 课】函数应用示例	45
【Python 第 23 课】if, elif, else	47
【Python 第 24 课】if 的嵌套	52
【Python 第 25 课】初探 list	54
【Python 第 26 课】操作 list	56
【Python 第 28 课】字符串的分割	63
【Python 第 29 课】连接 list	69
【Python 第 30 课】字符串的索引和切片	70
【Python 第 31 课】读文件	72
【Python 第 32 课】写文件	74
【Python 第 33 课】处理文件中的数据	75
【Python 第 34 课】break	81
【Python 第 35 课】continue	82
【Python 第 36 课】异常处理	85
【Python 第 37 课】字典	88
【Python 第 38 课】模块	91
【Python 第 39 课】用文件保存游戏（1）	94
【Python 第 40 课】用文件保存游戏（2）	96
【Python 第 41 课】用文件保存游戏（3）	99
【Python 第 42 课】函数的默认参数	103
【Python 第 43 课】查天气（1）	105
【Python 第 44 课】查天气（2）	106
【Python 第 45 课】查天气（3）	109
【Python 第 46 课】查天气（4）	111

【Python 第 47 课】 面向对象（1）	114
【Python 第 48 课】 面向对象（2）	115
【Python 第 49 课】 面向对象（3）	116
【Python 第 50 课】 面向对象（4）	118
【Python 第 51 课】 and-or 技巧	121
【Python 第 52 课】 元组	122
【Python 第 53 课】 数学运算	123
【Python 第 54 课】 真值表	125
【Python 第 55 课】 正则表达式（1）	126
【Python 第 56 课】 正则表达式（2）	128
【Python 第 57 课】 正则表达式（3）	129
【Python 第 58 课】 正则表达式（4）	132
【Python 第 59 课】 正则表达式（5）	133
【Python 第 60 课】 随机数	135
python 模块的常用安装方式	137
正则表达式 30 分钟入门教程	138
目录	138
本文目标	139
如何使用本教程	139
正则表达式到底是什么东西？	140
入门	140
测试正则表达式	141
元字符	142
字符转义	144
重复	144
字符类	144
分枝条件	145
分组	145
反义	146
后向引用	146
零宽断言	147
负向零宽断言	148
注释	149
贪婪与懒惰	149
处理选项	150
平衡组/递归匹配	150
还有什么东西没提到	152
联系作者	153
网上的资源及本文参考文献	153
更新纪录	153

【Python 第 0 课】Why Python?



为什么用 Python 作为编程入门语言？

原因很简单。

每种语言都会有它的支持者和反对者。去 Google 一下 “why python”，你会得到很多结果，诸如应用范围广泛、开源、社区活跃、丰富的库、跨平台等等等等，也可能找到不少对它的批评，格式死板、效率低、国内用的人很少之类。不过这些优缺点的权衡都是程序员们的烦恼。作为一个想要学点编程入门的初学者来说，简单才是最重要的。当学 C++ 的同学还在写链表，学 Java 的同学还在折腾运行环境的时候，学 Python 的你已经像上图一样飞上天了。

当然，除了简单，还有一个重要的原因：因为我现在每天都在写 Python。虽然以后可能会讲些手机编程之类（如果真的有那么一天 $\pi_ \pi$ ），但目前这时候，各位也就看菜吃饭，有啥吃啥了。每天 5 分钟，先别计较太多。况且 Python 还是挺有利于形成良好编程思维的一门语言。

推荐两本我个人比较喜欢的 Python 入门书籍，一本是《简明 Python 教程》，我自己最开始就是看着它学的，接下来也会大体参考里面的内容讲。另一本是《Head First Python》，Head First 系列都是非常浅显易懂的入门类书籍，虽然我只瞄过几眼，但感觉还是不错的。

【Python 第 1 课】安装

进入 Python 的官方下载页面

<http://www.python.org/download/>

你会看到一堆下载链接。我们就选“Python 2.7.5 Windows Installer”，如果是 64 位系统的同学选下面那个“Python 2.7.5 Windows X86-64 Installer”。为什么不选最上面那个 3.3.2 的新版本？因为我在用 python2.7.x，python3 改了不少地方，不熟。

下载之后，就和装其他软件一样，双击，一路 Next，想换安装路径的同学可以换个位置。但不管换不换，请把这个路径复制下来，比如我的是“C: \python27\”，后面要用到它。

安装结束还没完，我们还差最后一步：设置环境变量。这是什么东西我暂时先不解释，大家照着做就好。右键单击我的电脑（不，是你的电脑），依次点击“属性”->“高级”->“环境变量”，在“系统变量”表单中点击叫做 Path 的变量，然后编辑这个变量，把“;C:\Python27\”，也就是你刚才复制的安装路径，加到它的结尾。注意！要用英文分号和前面已有的内容隔开。然后点确定，点确定，再点确定。完成。

怎么知道你已经成功安装了 Python 呢？这时候你需要打开命令行，或者叫命令提示符、控制台。方法是：点击开始菜单->程序->附件->命令提示符；或者直接在桌面按快捷键“Win+r”，Win 键就是 Ctrl 和 Alt 旁边那个有 windows 图标的键，输入 cmd，回车。这时候你就看到可爱的黑底白字了。

在命令行里输入 python，回车。如果看到诸如：

Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32

的提示文字，恭喜你！否则，请重新检查你哪里的打开方式不对，或者直接给我留言。

接下来，你就可以输入那句程序员最爱的

```
print "Hello World"
```

向 Python 的世界里发出第一声啼哭。

嗯。。。如果这么几步你还是被绕晕了，没关系，我还留了一手：打开你的浏览器，Google 一下 “python online”，点击第一条结果 “Execute Python Script Online”；或者直接打开 [compileonline.com](http://www.compileonline.com)，找到 Python 点进去。

http://www.compileonline.com/execute_python_online.php

这是一个在线的 python 运行环境，你可以在这里练习，无需任何下载安装配置。左边页面是写代码的地方，点击左上角的 “Execute Sctipt”，就可以在右边页面看到输出结果。

那 Mac 的同学怎么办？Mac 上叫 “终端”，英文版叫 Terminal，可以在 “应用程序” 里找到，也可以直接在你的 Mac 上搜索 “终端” 或者 “Terminal” 找到。打开之后输入 “python”，回车，就可以进入 python 了。

好了，今天就这么多，快去试试你的 python，输出一行 “Hello World” 吧。完成的同学可以截个屏发给我。欢迎各种建议、讨论和闲聊，当然更欢迎你在这里分享给更多的朋友。

我今天发现昨天提供的 [compileonline.com](http://www.compileonline.com) 网站有时候会很慢，甚至无法正常运行，于是我又找了一个：

<http://www.pythonfiddle.com>

似乎要快一点，不过好像只能在电脑上的浏览器打开。另外就是，昨天忘了给 Mac 的同学们说一下怎么打开命令行。Mac 上叫做 “终端” 或者 “Terminal”，可以在 “应用程序” 里找到，也可以直接在 “spotlight” 里直接输入 “Terminal” 打开。打开后就可以通过 “python” 命令进入开发环境了。

【Python 第 2 课】print

print，中文意思是打印，在 python 里它不是往纸上打印，而是打印在命令行，或者叫终端、控制台里面。print 是 python 里很基本很常见的一个操作，它的操作对象是一个字符串（什么是字符串，此处按住不表，且待日后慢慢道来）。基本格式是：print 你要打印的东西 或者 print(你要打印的东西) 这里一定要英文字符的括号，所有程序中出现的符号都必须是英文字符，注意别被你的输入法坑了。

各位同学可以在自己的 python 环境中试着输出以下内容（这里是命令行下的效果，使用在线编辑器或者 IDE 的同学，只需要输入 “>>>” 后面的内容就可以了）：

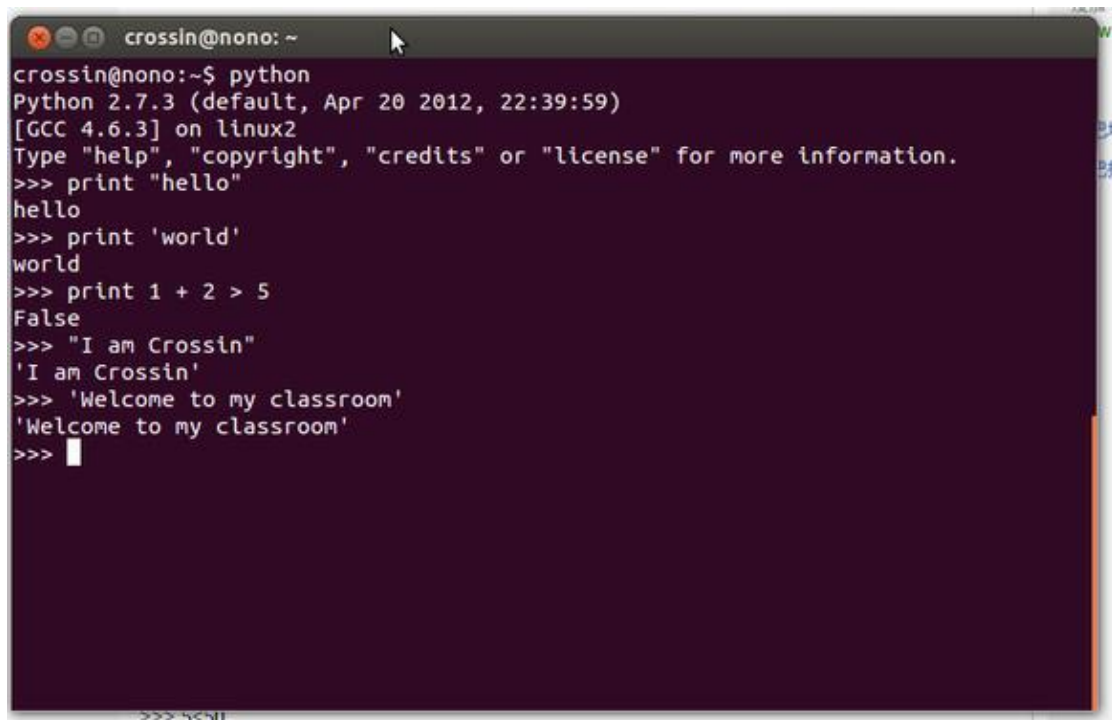
```
>>> print "hello"
hello
>>> print 'world'
world
>>> print 1
1
>>> print 3.14
3.14
>>> print 3e30
3e+30
>>> print 1 + 2 * 3
7
>>> print 2 > 5
False
```

直接在 print 后面加一段文字来输出的话，需要给文字加上双引号或者单引号。大家发现，print 除了打印文字之外，还能输出各种数字、运算结果、比较结果等。你们试着自己 print 一些别的东西，看看哪些能成功，哪些会失败，有兴趣的话再猜一猜失败的原因。

其实在 python 命令行下，print 是可以省略的，默认就会输出每一次命令的结果。就像这样：

```
>>> 'Your YiDa!'
'Your YiDa!'
>>> 2+13+250
265
>>> 5<50
True
```

今天内容就这么多。没听出个所以然？没关系，只要成功 print 出来结果就可以，我们以后还有很多时间来讨论其中的细节。



```
crossin@nono: ~  
crossin@nono:~$ python  
Python 2.7.3 (default, Apr 20 2012, 22:39:59)  
[GCC 4.6.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print "hello"  
hello  
>>> print 'world'  
world  
>>> print 1 + 2 > 5  
False  
>>> "I am Crossin"  
'I am Crossin'  
>>> 'Welcome to my classroom'  
'Welcome to my classroom'  
>>> 
```

这个短期目标就是一个很简单很弱智的小游戏：

COM: Guess what I think?

5

COM: Your answer is too small.

12

COM: Your answer is too large.

9

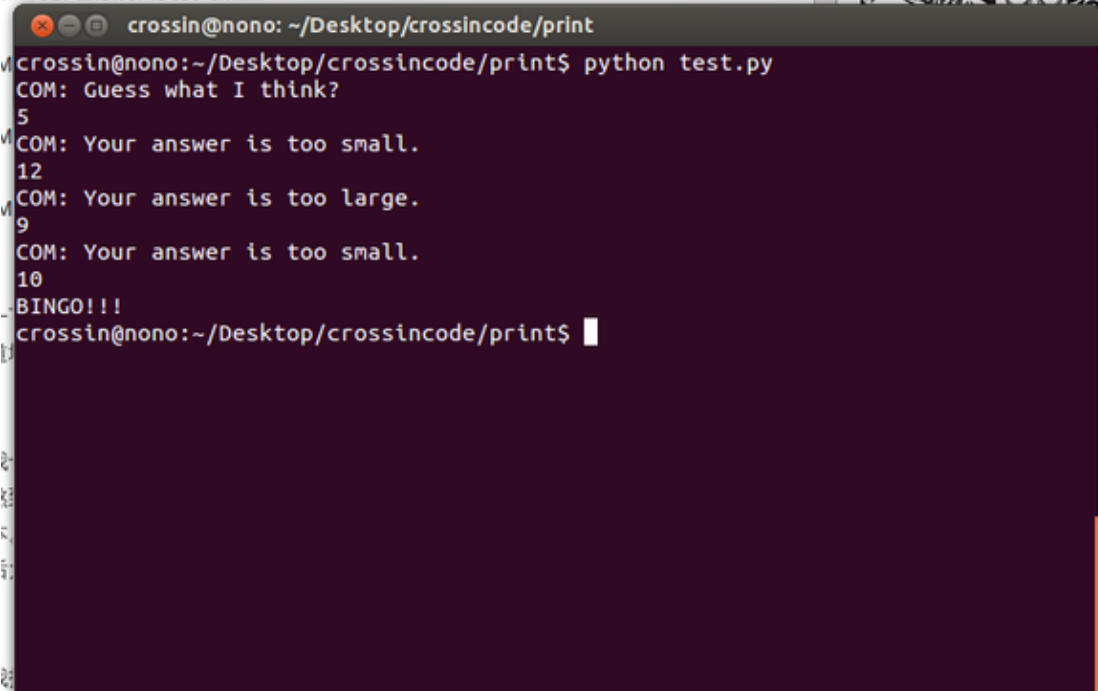
COM: Your answer is too small.

10

COM: BINGO!!!

解释一下：首先电脑会在心中掐指一算，默念一个数字，然后叫你猜。你猜了个答案，电脑会厚道地告诉你大了还是小了，直到最终被你果断猜中。

这是我十几年前刚接触编程时候写的第一个程序，当时家里没有电脑，在纸上琢磨了很久之后，熬到第二个星期的电脑课才在学校的 486 上 run 起来。后来我还写过一个 windows 下的窗口版本。现在就让它也成为你们第一个完整的程序吧。照我们每天 5 分钟的进度，初步估计半个月后大约能完成了。

A terminal window with a dark purple background and light green text. The window title is 'crossin@nono: ~/Desktop/crossincode/print'. The prompt is 'crossin@nono:~/Desktop/crossincode/print\$'. The user has run 'python test.py'. The program output is: 'COM: Guess what I think?', '5', 'COM: Your answer is too small.', '12', 'COM: Your answer is too large.', '9', 'COM: Your answer is too small.', '10', 'BINGO!!!'. The prompt is now 'crossin@nono:~/Desktop/crossincode/print\$' with a cursor.

```
crossin@nono: ~/Desktop/crossincode/print
crossin@nono:~/Desktop/crossincode/print$ python test.py
COM: Guess what I think?
5
COM: Your answer is too small.
12
COM: Your answer is too large.
9
COM: Your answer is too small.
10
BINGO!!!
crossin@nono:~/Desktop/crossincode/print$
```

【Python 第3课】IDE

打个不恰当的比方，如果说写代码是制作一件工艺品，那 IDE 就是机床。再打个不恰当的比方，PS 就是图片的 IDE，Word 就是 doc 文档的 IDE，PowerPoint 就是 ppt 文件的 IDE。python 也有自己的 IDE，而且还有很多。

python 自带了一款 IDE，叫做 IDLE。先说 Windows，Windows 上安装了之后，可以在“开始菜单”->“程序”->“Python 2.7”里找到它。打开后之后很像我们之前用过的命令行。没错，它就是的，在里面 print 一下试试。不知之前用命令行的同学有没有注意到，命令行输一行命令就会返回结果，而且之前 print 了那么多，关掉之后也不知道到哪里去了。所以它没法满足我们编写弱智小游戏的大计划。我们需要用新的方法！

点击窗口上方菜单栏的“File”->“New Window”，会打一个长得很像的新窗口，但里面什么也没有。这是一个文本编辑器，在这里面就可以写我们的 python 程序了。继续 print 几行，这次可以多 print 一点：

```
print 'Hello'
print 'IDE'
print 'Here I am.'
```

现在是，见证奇迹的时刻！点击“Run”->“Run Module”，或者直接按快捷键 F5。会提示你保存刚才文件，随便取个名字，比如“lesson3.py”。(.py 是 python 代码文件的类型，虽

然不指定.py 也是可以的，但建议还按规范来）保存完毕后，之前那个控制台窗口里就会一次性输出你要的结果。

以后想再次编辑或运行刚才的代码，只要在 IDLE 里选择“File”->“Open...”，打开刚才保存的.py 文件就可以了。

Mac 上的 IDLE 是预装好了，在“终端”里输入“IDLE”就可以启动，使用方法同 Windows。也可以在文件夹/usr/bin 里可以找到 IDLE。如果是重新下载安装了 python，似乎是在“应用程序”里找到 IDLE 的，Mac 的同学可以验证下。

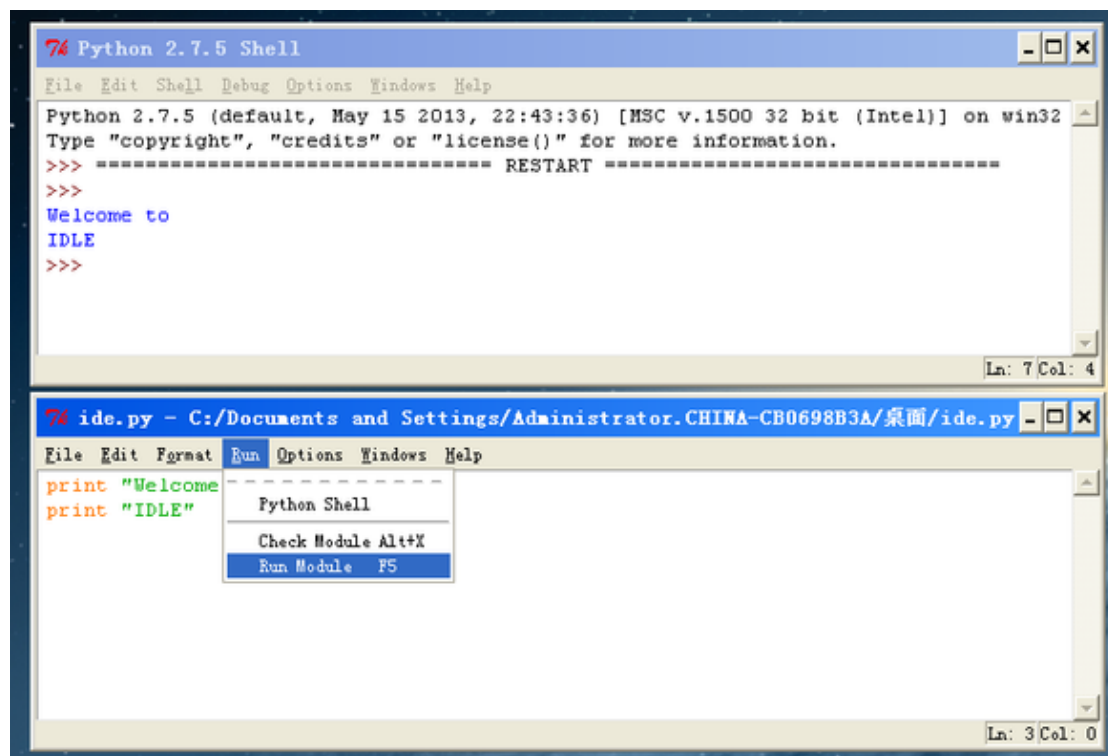
另外，Windows 下有一个第三方的免费 IDE，叫 PyScripter，把文件目录、文本编辑器、命令行都整合到了一起，还增加了很多辅助功能。有兴趣的同学也可以去找来试试看。地址：

<http://code.google.com/p/pyscripter/>

用起来应该比 IDLE 方便，但有一点要注意，它的安装位置和.py 文件的保存位置都不要有中文，不然可能会有问题。

今天的内容有点长。配置开发环境这种事最麻烦了，大家耐心一点，毕竟一次投入，长期受益。以后我们的课程都会在 IDE 中进行，基本不再往命令行里直接敲代码了。

最后说下，有很多 python 程序员都不使用任何 IDE。至于原因嘛，可能就像优秀的手工艺人是不会用机床来加工艺术品的吧。



【Python 第4课】输入

前 `print` 了那么多，都是程序在向屏幕“输出”。那有来得有往，有借得有还，有吃。。。咳咳！那啥，我们得有向程序“输入”信息的办法，才能和程序对话，进行所谓的“人机交互”。

python 有一个接收命令行下输入的方法：

`input()`

注意，和 `print` 不同的是，这次我们必须得加上`()`了，而且得是英文字符的括号。

好了，终于可以搬出那个弱智小游戏了，耶！游戏里我们需要跟程序一问一答，所以我们先把话给说上。

打开我们的 python 编辑器，不管是 IDLE，在线编辑器，还是其他的 IDE。在代码编辑器中输入下面几句代码：

```
print "Who do you think I am?"  
input()  
print "Oh, yes!"
```

然后，Run！（Forrest Run！）你会在命令行中看到，程序输出了第一句之后就停住了，这是 `input` 在等待你的输入。

输入你的回答，回车。你会看到程序的回答。注意！引号！！又是引号!!! 和 `print` 一样，如果你输的是一串文字，要用引号"或者'"引起来，如果是数字则不用。

（插一句，python 还有一个输入的方法：`raw_input()`，它把所有的输入都直接当作一串字符，于是就可以不用加引号，有兴趣的同学可以试一试，体会一下两者的不同。关于这个令人纠结的引号，我们以后会再讨论它。）

看上去不错哦，似乎就这么对上话了。是不是觉得离小游戏的完成迈进了一大步？可是大家发现没有，即使你说"`Idiot!`"，程序仍然会淡定地回答"`Oh, yes!`"因为它左耳进右耳出，根本就听进去我们到底说了啥。那怎么才能让它认真听话呢？啪！且听下回分解。



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
Who do you think I am?
'Genius'
How old are you?
17
Oh, yes!
>>> |
Ln: 89 Col: 4

input - C:/Documents and Settings/Administ...
File Edit Format Run Options Windows Help
print "Who do you think I am?"
input()
print "How old are you?"
input()
print "Oh, yes!"
Ln: 6 Col: 0
```

回顾一下我们之前几节课。我们到现在一共提到了三种可以运行 `print` 的方式：

1. 命令行，包括 Win 下的控制台（CMD）和 Mac 下的终端（Terminal）。
它可以帮我们确认自己电脑上的 `python` 是不是正常。但是这种方法很难帮我们实现写一个完整小程序的目标。
2. IDE，包括 `python` 自带的 IDLE 和其他第三方的 IDE。
不知道大家是不是都顺利搞定，并且能顺利保存并打开 `py` 文件了呢？以后我们课程里的内容，你都可以在这里面进行。
3. 在线编辑器，`compileonline` 或者 `pythonfiddle`。
他们同样包括代码编辑器（写代码的地方）和控制台（输出结果的地方）两部分。所以我们在本地 IDE 里的操作都可以在其中实现。只不过保存文件会有些复杂，`compileonline` 是点击 `download files` 打包下载，`pythonfiddle` 需要注册一下。当然，你也可以直接把你写好的代码

复制下来，保存在本地，下次再粘贴上去接着写。

【Python 第5课】变量

昨天说到，需要让程序理解我们输入的东西。那首先，就需要有东西把我们输入的内容记录下来，好为接下来的操作做准备。Python 之神说，要有变量！于是就有了变量。

变量，望文生义，就是变化的量。python 里创建一个变量的方法很简单，给它起个名字，然后给它一个值。举起几个栗子：

```
name = 'Crossin'
myVar = 123
price = 5.99
visible = True
```

“=” 的作用是把右边的值赋予给左边的变量。

这里说一下另外一个概念，叫做“数据类型”，上面 4 颗栗子分别代表了 python 中较常见的四种基本类型：

字符串 - 表示一串字符，需要用"或"'"引起来

整数

浮点数 - 就是小数

bool（布尔） - 这个比较特殊，是用来表示逻辑“是”“非”的一种类型，它只有两个值，True 和 False。（注意这里没有引号，有了引号就变成字符串了）

再次用到我们熟悉的 print。这次，我们升级了，要用 print 输出一个“变量”：

```
name = 'Crossin'  
print name
```

看到结果了吗？没有输出“name”，也没有报错，而是输出了“Crossin”。现在是不是能想明白一些，为什么之前 print 一段文字没加引号就会报错，而 print 一个数字就没有问题呢？

它叫变量，那就是能变的。所以在一次“赋值”操作之后，还可以继续给它赋予新的值，而且可以是不同类型的值。

```
a = 123  
print a  
a = 'hi'  
print a
```

“=”的右边还可以更复杂一点，比如是一个计算出的值：

```
value = 3 * 4  
print value  
value = 2 < 5  
print value
```

甚至，也可以是 input()：

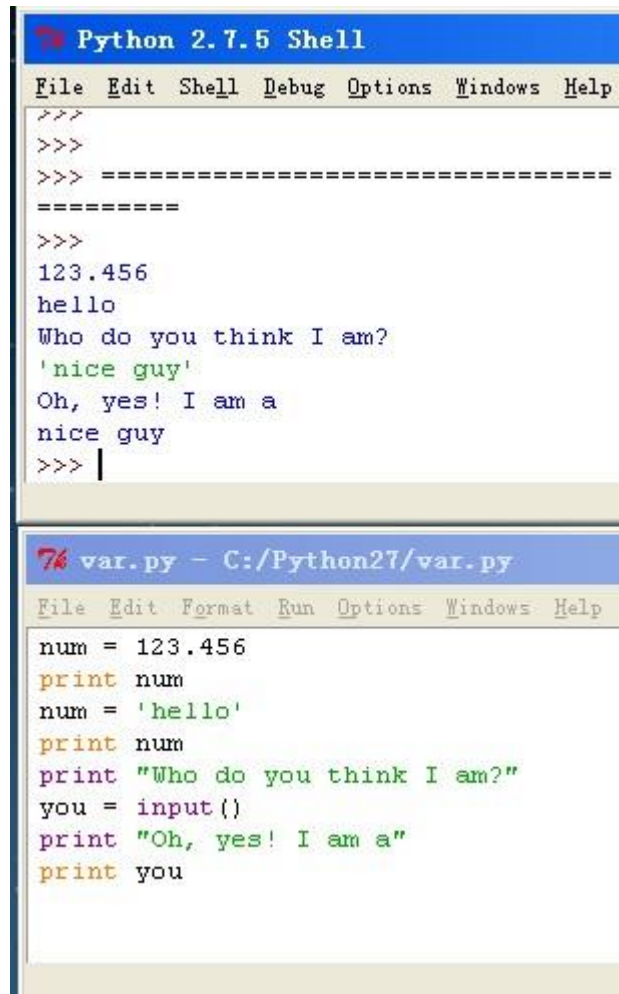
```
name = input()  
print name
```

于是，我们又可以进化一下我们的小游戏了。把上次写的内容稍微改一下，加上变量：

```
print "Who do you think I am?"  
you = input()  
print "Oh, yes! I am a"  
print you
```

看来程序已经知道我们的输入了。接下来，就要让它学会对不同的答案做出判断。这个我们留到下次再说。

今天是周五。我觉得吧，到周末了，大家应该远离一下电脑，多陪陪家人朋友，吃吃饭，出去走走。祝大家周末愉快！



The image shows two screenshots from a Windows environment. The top screenshot is a 'Python 2.7.5 Shell' window. It has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The shell prompt is '>>>'. The user has entered several lines of code: an equals sign followed by another equals sign, the number '123.456', the string 'hello', the string 'Who do you think I am?', the string 'nice guy' (highlighted in green), and the string 'Oh, yes! I am a nice guy'. The prompt is now '>>> |'. The bottom screenshot is a file editor window titled 'var.py - C:/Python27/var.py'. It has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Windows', and 'Help'. The code in the file is: 'num = 123.456', 'print num', 'num = 'hello'', 'print num', 'print "Who do you think I am?"', 'you = input()', 'print "Oh, yes! I am a"', and 'print you'.

```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
>>>
>>>
>>> =====
>>> =====
>>>
>>> 123.456
>>> hello
>>> Who do you think I am?
>>> 'nice guy'
>>> Oh, yes! I am a
>>> nice guy
>>> |

var.py - C:/Python27/var.py
File Edit Format Run Options Windows Help
num = 123.456
print num
num = 'hello'
print num
print "Who do you think I am?"
you = input()
print "Oh, yes! I am a"
print you
```

【Python 第6课】bool

昨天说到了 python 中的几个基本类型，字符串、整数、浮点数都还算好理解，关于剩下的那个 bool（布尔值）我要稍微多说几句。

逻辑判断在编程中是非常重要的。大量的复杂程序在根本上都是建立在“真”与“假”的基本逻辑之上。而 bool 所表示的就是这种最单纯最本质的 True / Flase，真与假，是与非。

来看下面的例子：

`a = 1 < 3`

```
print a
b = 1
c = 3
print b > c
```

通过用 “>” “<” 来比较两个数值，我们就得到了一个 bool 值。这个 bool 值的真假取决于比较的结果。

“>” “<” 在编程语言中被成为逻辑运算符，常用的逻辑运算符包括：

>: 大于
<: 小于
>=: 大于等于
<=: 小于等于
==: 等于。比较两个值是否相等。之所以用两个等号，是为了和变量赋值区分开来。
!=: 不等与
not: 逻辑“非”。如果 x 为 True，则 not x 为 False
and: 逻辑“与”。如果 x 为 True，且 y 为 True，则 x and y 为 True
or: 逻辑“或”。如果 x、y 中至少有一个为 True，则 x or y 为 True

关于 bool 值和逻辑运算其实远不止这些，但现在我们暂时不去考虑那么多，以免被绕得找不到北。最基本的大于、小于、等于已经够我们先用一用的了。

试试把 bool 加到我们的小游戏里：

```
num = 10
print 'Guess what I think?'
answer = input()
```

```
result = answer < num
print 'too small?'
print result
```

```
result = answer > num
print 'too big?'
print result
```

```
result = answer==num  
print 'equal?'  
print result
```

代码比之前稍微多了一点，解释一下。

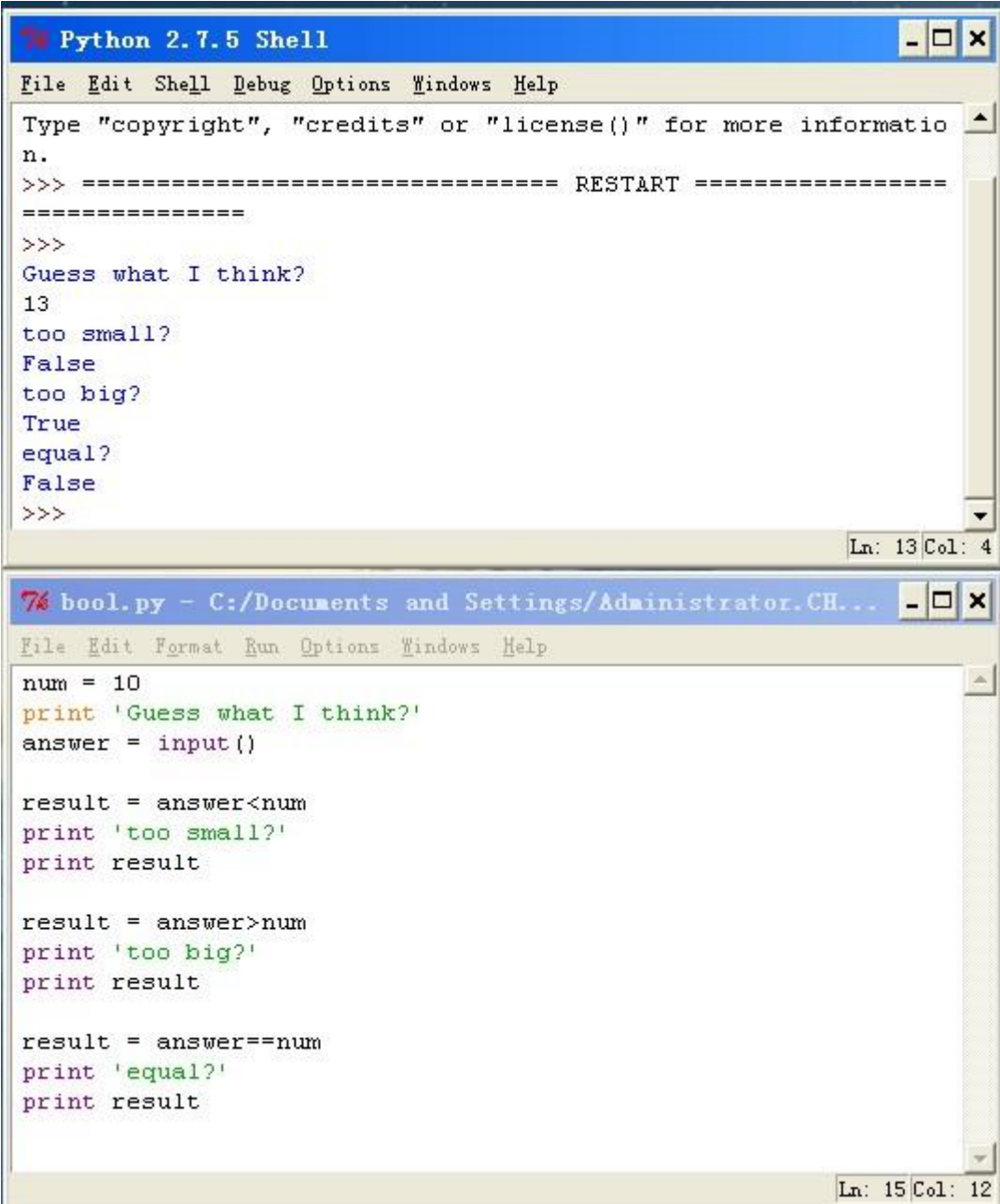
第一段代码：先创建一个值为 10 的变量 `num`，输出一句提示，然后再输入一个值给变量 `answer`。

第二段代码：计算 `answer<num` 的结果，记录在 `result` 里，输出提示，再输出结果。

第三段、第四段都与第二段类似，只是比较的内容不一样。

看看结果是不是跟你预期的一致？虽然看上去还是有点傻，但是离目标又进了一步。

现在数数你手上的工具：输入、输出，用来记录数值的变量，还有可以比较数值大小的逻辑运算。用它们在你的 `python` 里折腾一番吧。



The image shows two screenshots from a Python 2.7.5 environment. The top screenshot is a terminal window titled 'Python 2.7.5 Shell'. It displays the following text:

```
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Guess what I think?
13
too small?
False
too big?
True
equal?
False
>>>
```

The bottom screenshot is a text editor window titled 'bool.py - C:/Documents and Settings/Administrator.CH...'. It contains the following Python code:

```
num = 10
print 'Guess what I think?'
answer = input()

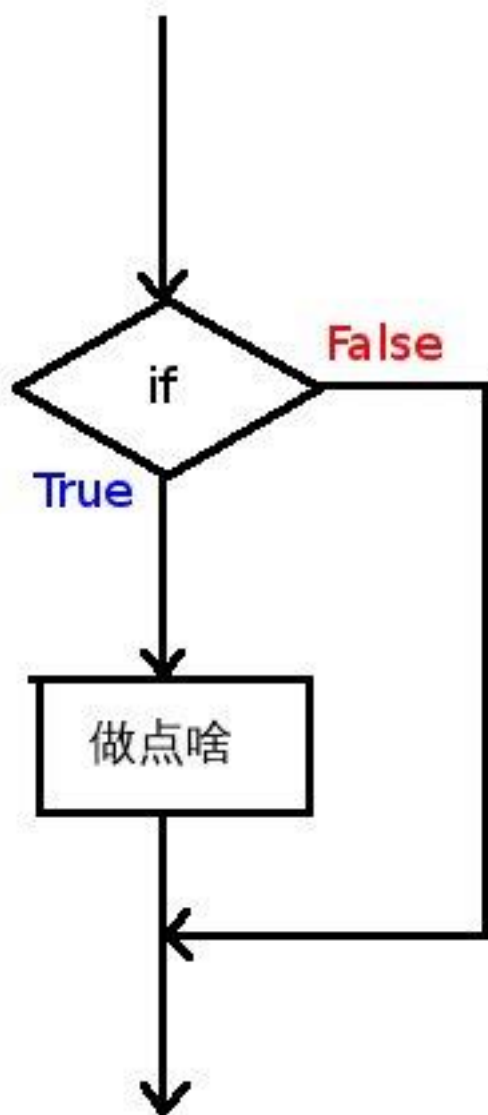
result = answer<num
print 'too small?'
print result

result = answer>num
print 'too big?'
print result

result = answer==num
print 'equal?'
print result
```

【Python 第7课】if

继续上次的程序。我们已经能让判断我们输入的值了，但这程序还是有点呆，不过怎样都要把话说三遍。因为到目前为止，我们的程序都是按照顺序从上到下一行接一行地执行。有同学发来了问题：怎么能让它根据我们输入的结果来选择执行呢？答案就是 - if
来看一张图（纯手绘，渣画质）



解释一下，程序顺序往下执行遇到 if 语句的时候，会去判断它所带条件的真假。

“如果”为 True，就会去执行接下来的内容。“如果”为 False，就跳过。

语法为：

if 条件:

选择执行的语句

特别说明：条件后面的冒号不能少，同样必须是英文字符。

特别特别说明：if 内部的语句需要有一个统一的缩进，一般用 4 个空格。python 用这种方法

替代了其他很多编程语言中的{}。你也可以选择 1/2/3...个空格或者按一下 **tab** 键，但必须整个文件中都统一起来。千万不可以 **tab** 和空格混用，不然就会出现各种莫名其妙的错误。所以建议都直接用 4 个空格。

上栗子：

```
thisIsLove = input()
if thisIsLove:
    print "再转身就该勇敢留下来"
```

试试看？输入 **True**，就会得到回答。输入 **False**，什么也没有。（如果你那里输出中文有问题，请自行改成英文）

所以，我们的游戏可以这样改写：

```
num = 10
print 'Guess what I think?'
answer = input()
if answer < num:
    print 'too small!'

if answer > num:
    print 'too big!'
if answer == num:
    print 'BINGO!'
```

```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
>>>
Guess what I think?
13
too big!
>>> ===== RESTART =====
>>>
Guess what I think?
10
BINGO!
>>>
Ln: 167 Col: 0

if.py - C:/Python27/if.py
File Edit Format Run Options Windows Help
num = 10
print 'Guess what I think?'
answer = input()

if answer < num:
    print 'too small!'

if answer > num:
    print 'too big!'

if answer == num:
    print 'BINGO!'
Ln: 13 Col: 0
```

if 在编程语言中被称为“控制流语句”，用来控制程序的执行顺序。还有其他的控制流语句，后面我们会用到。

重新发一下代码

```
thisIsLove = input()
if thisIsLove:
    print "再转身就该勇敢留下来"
```

=====

```
num = 10
print 'Guess what I think?'
answer = input()
if answer < num:
    print 'too small!'
if answer > num:
```

```
    print 'too big!'
if answer==num:
    print 'BINGO!'
```

【Python 第8课】while

先介绍一个新东西：注释。

python 里，以“#”开头的文字都不会被认为是可执行的代码。

```
    print “hello world”
```

和

```
    print "hello world" #输出一行字
```

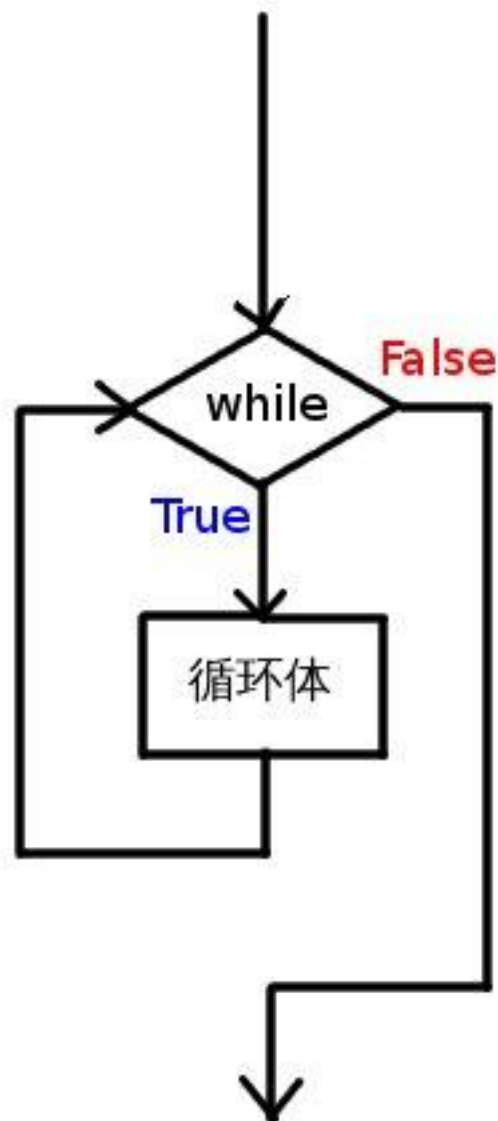
是同样的效果。但后者可以帮助开发者更好地理解代码。

在接下来的课程中，我会经常用注释来解释代码。

用 if 改进完我们的小游戏后，功能已经基本实现了。很多同学做完后纷纷表示，每次只能猜一次，完了之后又得重新 run，感觉好麻烦。能不能有办法让玩家一直猜，直到猜中为止？答案很显然，如果这种小问题都解决不了，那 python 可就弱爆了。

最简单的解决方法就是 while。

同 if 一样，while 也是一种控制流语句，另外它也被称作循环语句。继续来看渣画质手绘流程图：



程序执行到 `while` 处，“当”条件为 `True` 时，就去执行 `while` 内部的代码，“当”条件为 `False` 时，就跳过。

语法为：

```
while 条件:  
    循环执行的语句
```

同 `if` 一样，注意冒号，注意缩进。

今天的栗子：

```
a = 1 #先 a 设为 1
```

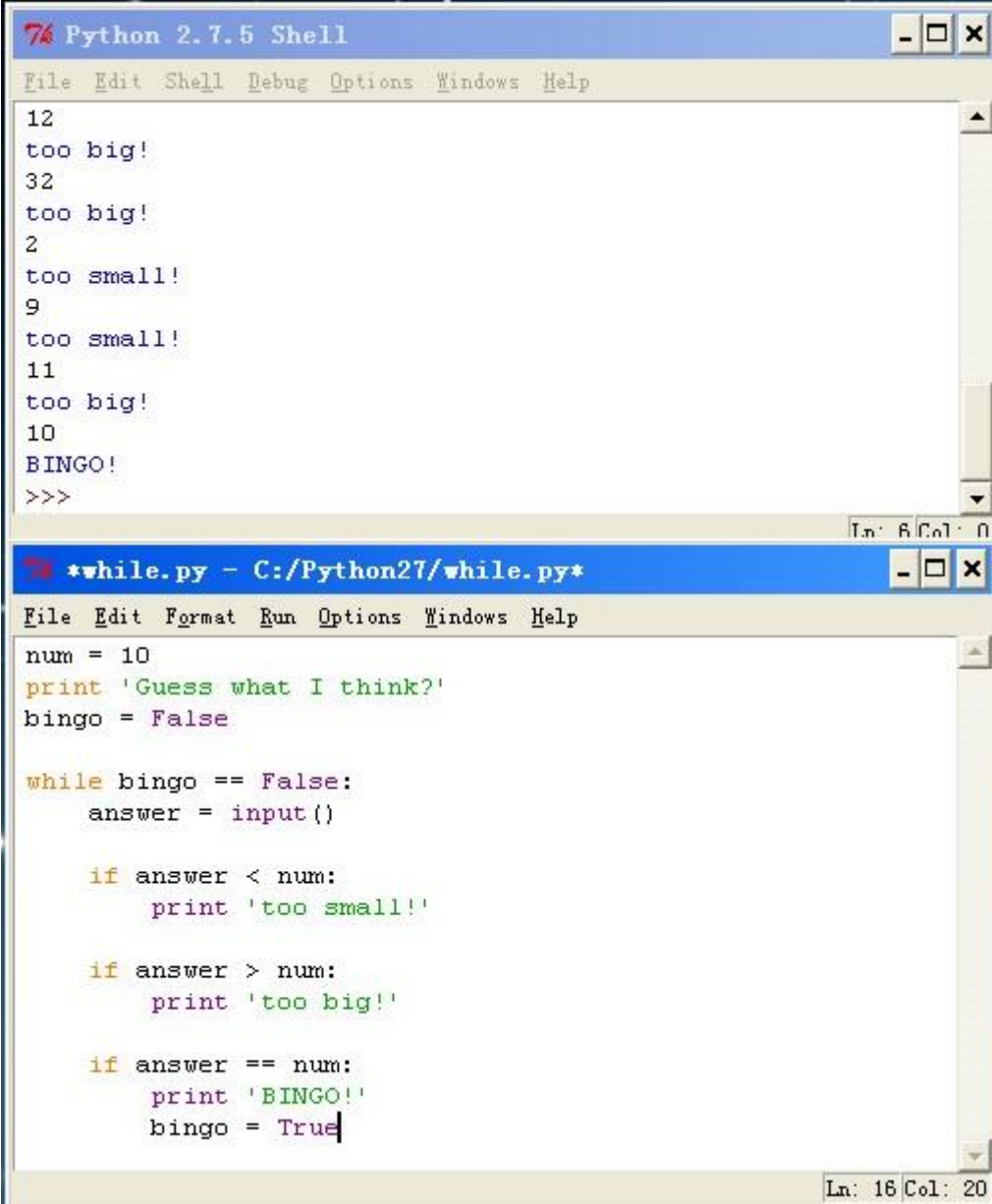
```
while a != 0: #a 不等于 0 就一直做
```

```
    print "please input"
```

```
    a = input()
```

```
print "over"
```

想想怎么用 while 改进小游戏？有多种写法，大家自己思考下，我不多做说明了。下图给出一种方法。



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
12
too big!
32
too big!
2
too small!
9
too small!
11
too big!
10
BINGO!
>>>

*while.py - C:/Python27/while.py*
File Edit Format Run Options Windows Help
num = 10
print 'Guess what I think?'
bingo = False

while bingo == False:
    answer = input()

    if answer < num:
        print 'too small!'

    if answer > num:
        print 'too big!'

    if answer == num:
        print 'BINGO!'
        bingo = True
```

注意，这里出现了两层缩进，要保持每层缩进的空格数相同。

到此为止，小游戏已经基本成型了。不过好像还差一点：每次自己都知道答案，这玩起来有神马意思。

明天来讲，怎么让你不知道电脑的答案。

【Python 第9课】random

之前我们用了很多次的 `print` 和 `input` 方法，它们的作用是实现控制台的输入和输出。除此之外，`python` 还提供了很多模块，用来实现各种常见的功能，比如时间处理、科学计算、网络请求、随机数等等等等。今天我就来说说，如何用 `python` 自带的随机数模块，给我们的小游戏增加不确定性。

引入模块的方法：

`from 模块名 import 方法名`

看不懂没关系，这东西以后我们会反复用到。今天你只要记住，你想要产生一个随机的整数，就在程序的最开头写上：

`from random import randint`

之后你就可以用 `randint` 来产生随机数了。

还记得 `input` 后面的 `()` 吗，我们使用 `randint` 的时候后面也要有 `()`。而且，还要在括号中提供两个数字，先后分别是产生随机整数范围的下限和上限。例如：

`randint(5, 10)`

这样将会产生一个 5 到 10 之间（包括 5 和 10）的随机整数。

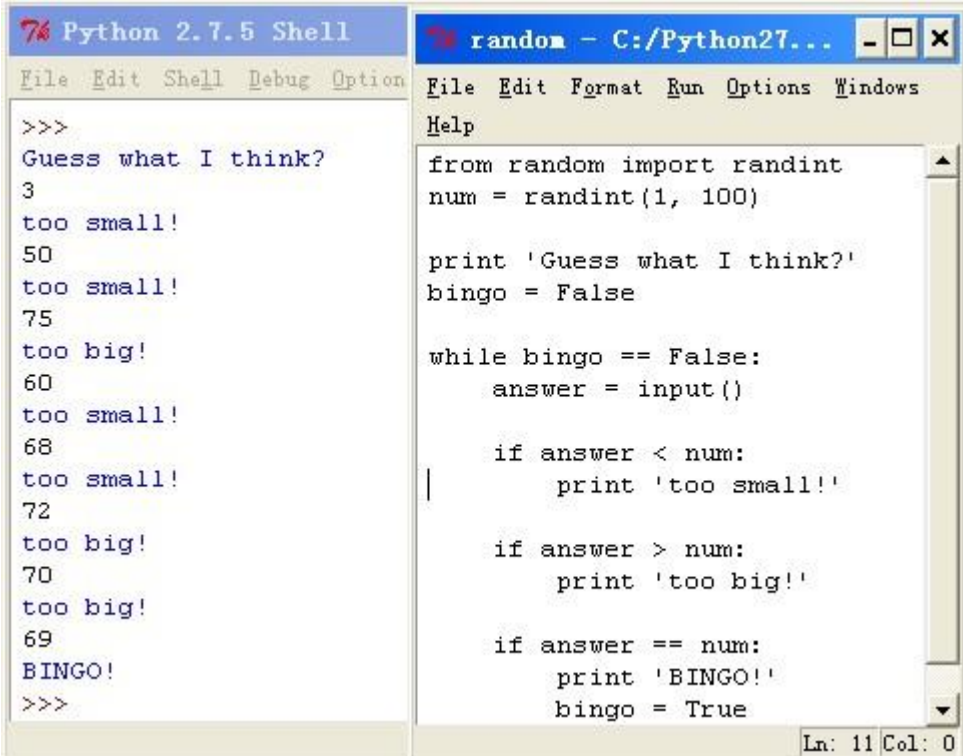
放到我们的小游戏里，用

```
answer = randint(1, 100)
```

替代

```
answer = 10
```

程序在运行时候，会产生一个 1 到 100 的随机整数，存在 `answer` 里，我们也不知道是多少，真的全靠猜了。



```
Python 2.7.5 Shell
File Edit Shell Debug Option
>>>
Guess what I think?
3
too small!
50
too small!
75
too big!
60
too small!
68
too small!
72
too big!
70
too big!
69
BINGO!
>>>
```

```
random - C:/Python27...
File Edit Format Run Options Windows
Help
from random import randint
num = randint(1, 100)

print 'Guess what I think?'
bingo = False

while bingo == False:
    answer = input()

    if answer < num:
        print 'too small!'

    if answer > num:
        print 'too big!'

    if answer == num:
        print 'BINGO!'
        bingo = True
Ln: 11 Col: 0
```

好了，觉得还有点意思么？我们终于一步步把这个弱智小游戏给做出来了，有没有一丁点的成就感呢？

如果你对其中的某些细节还不是很理解，恭喜你，你已经开始入门了。相信你会带着一颗追求真相的心，在编程这条路上不断走下去。

我们的课程，也才刚刚开始。

【Python 第 10 课】变量 2

变量这东西，我们已经用过。有了变量，就可以存储和计算数据。今天来讲点变量的细节。

#==== 变量命名规则 ====#

变量名不是你想起就能起的：

第一个字符必须是字母或者下划线 “_”

剩下的部分可以是字母、下划线 “_” 或数字（0-9）

变量名称是对大小写敏感的，myname 和 myName 不是同一个变量。

几个有效的栗子：

```
i
__my_name
name_23
a1b2_c3
```

几个坏掉的栗子（想一下为什么不对）：

```
2things
this is spaced out
my-name
```

#==== 变量的运算 ====#

我们前面有用到变量来存储数据： `num = 10`
`answer = input()`

也有用到变量来比较大小： `answer < num`

除此之外，变量还可以进行数学运算： `a = 5 b = a + 3 c = a + b`

python 中运算的顺序是，先把 “=” 右边的结果算出了，再赋值给左边的变量。下面这个例子： `a = 5 a = a + 3 print a`

你会看到，输出了 8，因为先计算出了右边的值为 8，再把 8 赋给左边的 a。

通过这种方法，可以实现累加求和的效果。它还有个简化的写法：

`a += 3` 这个和 `a = a + 3` 是一样的。

于是，利用变量、循环、累加，可以写一个程序，来完成传说中高斯大牛在小时候做过的题：
 $1+2+3+\dots+100=?$ 从 1 加到 100 等于多少？

提示：你可以用一个变量记录现在加到几了，再用一个变量记录加出来的结果，通过 `while` 来判断是不是加到 100 了。

【Python 第 11 课】逻辑判断

之前粗略地提到 `bool` 类型的变量，又说到 `if` 和 `while` 的判断条件。有些同学反馈说没怎么理解，为什么一会儿是 `bingo=False`，一会又是 `bingo==False`，一会儿是 `while` 在条件为 `True` 的时候执行，一会儿又是 `while` 在 `bingo==False` 的时候执行。别急，你听我说。

首先，要理解，一个逻辑表达式，其实最终是代表了一个 `bool` 类型的结果，比如：

`1 < 3`

这个就像当于是一个 `True` 的值

`2 == 3`

这个就是 `False`

把它们作为判断条件放到 `if` 或者 `while` 的后面，就是根据他们的值来决定要不要执行。

同样的栗子再来几颗：

```
a = 1
print a > 3 #False
print a == 2 - 1 #True
b = 3
print a + b == 2 + 2 #True
```

比较容易搞混的，是 `bool` 变量的值和一个逻辑表达式的值，比如：

```
a = False
print a #False
print a == False #True
```

虽然 `a` 本身的值是 `False`，但是 `a==False` 这个表达式的值是 `True`。（说人话！）“`a`”是错的，但“`a`是错的”这句话是对的。

回到上面那几个概念：

`bingo=False`

把 `bingo` 设为一个值为 `False` 的变量

`bingo==False`

判断 `bingo` 的值是不是 `False`，如果是，那么这句话就是 `True`

`while` 在判断条件为 `True` 时执行循环，所以当 `bingo==False` 时，条件为 `True`，循环是要执行的。

晕了没？谁刚学谁都晕。不晕的属于骨骼惊奇百年一遇的编程奇才，还不赶紧转行做程序员！

逻辑这东西是初学编程的一大坑，我们后面还要在这个坑里挣扎很久。

留个习题：`a = True`

`b = not a` #不记得 `not` 请回复 6 想想下面这些逻辑运算的结果，然后用 `print` 看看你想的对不对：`bnot ba == ba != ba and ba or b1<2 and b==True`

【Python 第 12 课】for 循环

大家对 `while` 循环已经有点熟悉了吧？今天我们来讲另一种循环语句：

`for ... in ...`

同 `while` 一样，`for` 循环可以用来重复做一件事情。在某些场景下，它比 `while` 更好用。

比如之前的一道习题：输出 1 到 100（回复 903 可看详细内容）。

我们用 `while` 来做，需要有一个值来记录已经做了多少次，还需要在 `while` 后面判断是不是到了 100。

如果用 `for` 循环，则可以这么写：

```
for i in range(1, 101):
```

```
    print i
```

解释一下，`range(1, 101)`表示从 1 开始，到 101 为止（不包括 101），取其中所有的整数。`for i in range(1, 101)`就是说，把这些数，依次赋值给变量 `i`。相当于一个一个循环过去，第一次 `i = 1`，第二次 `i = 2`，……，直到 `i = 100`。当 `i = 101` 时跳出循环。所以，当你需要一个循环 10 次的循环，你就只需要写：

```
for i in range(1, 11)
```

或者

```
for i in range(0, 10)
```

区别在于前者 `i` 是从 1 到 10，后者 `i` 是从 0 到 9。当然，你也可以不用 `i` 这个变量名。比如一个循环 `n` 次的循环：

```
for count in range(0, n)
```

`for` 循环的本质是对一个序列中的元素进行递归。什么是序列，以后再说。先记住这个最简单的形式：

```
for i in range(a, b)
```

从 `a` 循环至 `b-1`

现在，你可以用 `for` 循环来改写习题 903,904,905,906 了。

【Python 第 13 课】字符串

字符串就是一组字符的序列（序列！又见序列！还记得我说过，`range` 就是产生一组整数序列。今天仍然不去细说它。），它一向是编程中的常见问题。之前我们用过它，以后我们还要不停地用它。

python 中最常用的字符串表示方式是单引号（`'`）和双引号（`"`）。我还是要再说：一定得是英文字符！

`'string'`和 `"string"` 的效果是一样的。

可以直接输出一个字符串 `print 'good'`

也可以用一个变量来保存字符串，然后输出 `str = 'bad' print str`

如果你想表示一段带有英文单引号或者双引号的文字，那么表示这个字符串的引号就要与内容区别开。

内容带有单引号，就用双引号表示 `"It's good"`

反之亦然

`'You are a "BAD" man'`

python 中还有一种表示字符串的方法：三个引号（`''` 或者 `"""`）

在三个引号中，你可以方便地使用单引号和双引号，并且可以直接换行

```
"""
```

```
"What's your name?" I asked.
```

```
"I'm Han Meimei."
```

```
"""
```

还有一种在字符串中表示引号的方法，就是用`\`，可以不受引号的限制

`\'`表示单引号，`\"`表示双引号

`'I\'m a \"good\" teacher'`

\被称作转译字符，除了用来表示引号，还有比如用
\\表示字符串中的\
\n 表示字符串中的换行

\还有个用处，就是用来在代码中换行，而不影响输出的结果：
"this is the\
same line"

这个字符串仍然只有一行，和
"this is thesame line"
是一样的，只是在代码中换了行。当你要写一行很长的代码时，这个会派上用场。

作业时间】用 `print` 输出以下文字：

1.
He said, "I'm yours!"

2.
v//

3.
Stay hungry,
stay foolish.
-- Steve Jobs

4.
*

*

【Python 第 14 课】字符串格式化

我们在输出字符串的时候，如果想对输出的内容进行一些整理，比如把几段字符拼接起来，或者把一段字符插入到另一段字符中间，就需要用到字符串的格式化输出。

先从简单的开始，如果你想把两段字符连起来输出

```
str1 = 'good'
str2 = 'bye'
```

你可以

```
print str1 + str2
```

或者还可以把字符变量一个字符串相加

```
print 'very' + str1
print str1 + ' and ' + str2
```

但如果你想要把一个数字加到文字后面输出，比如这样

```
num = 18
print 'My age is' + num
```

程序就会报错。因为字符和数字不能直接用+相加。

一种解决方法是，用 `str()` 把数字转换成字符串

```
print 'My age is' + str(18)
或
num = 18
print 'My age is' + str(num)
```

还有一种方法，就是用 `%` 对字符串进行格式化

```
num = 18
print 'My age is %d' % num
```

输出的时候，`%d` 会被 `%` 后面的值替换。输出

```
My age is 18
```

这里，`%d` 只能用来替换整数。如果你想格式化的数值是小数，要用 `%f`

```
print 'Price is %f' % 4.99
```


输出

```
Price is 4.990000
```

如果你想保留两位小数，需要在 f 前面加上条件：%.2f

```
print 'Price is %.2f' % 4.99
```

输出

```
Price is 4.99
```

另外，可以用%s 来替换一段字符串

```
name = 'Crossin'
```

```
print '%s is a good teacher.' % name
```

输出

```
Crossin is a good teacher.
```

或者

```
print 'Today is %s.' % 'Friday'
```

输出

```
Today is Friday.
```

注意区分：有引号的表示一段字符，没有引号的就是一个变量，这个变量可能是字符，也可能是数字，但一定要和%所表示的格式相一致。

现在，试试看用字符串格式化改进一下之前你写的小游戏。比如你输了一个数字 72，程序会回答你

```
72 is too small.
```

或者

```
Bingo, 72 is the right answer!
```

【Python 第 15 课】 循环的嵌套

设想一样，如果我们要输出 5 个*，用 for 循环要这么写

```
for i in range(0, 5):
```

```
    print '*'
```

如果想让这 5 个*在同一行，就在 print 语句后面加上逗号

```
for i in range(0, 5):  
    print '*',
```

但如果我想要这样一个图形，怎么办？

```
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```

当然，你可以循环 5 次，每次输出一行 “* * * * *”。那如果再进一步，这样呢？

```
*  
  
**  
  
***  
  
****  
  
*****
```

除了你自己动手打好一个多行字符串外，也可以让程序帮我们解决这种问题，我们需要的是两个嵌套在一起的循环：

```
for i in range(0, 5):  
    for j in range(0, 5):  
        print i, j
```

第二个 for 循环在第一个 for 循环的内部，表示每一次外层的循环中，都要进行一遍内层的循环。

看一下输出的结果：

```
0 0  
0 1  
0 2  
0 3  
0 4  
1 0  
...  
4 4
```

内层循环中的 print 语句一共被执行了 25 次。

i 从 0 到 4 循环了 5 次。对应于每一个 i 的值，j 又做了从 0 到 4 五次循环。所以 5*5 一共 25 次。

所以如果要输出一个 5*5 的方阵图案，我们可以

```
for i in range(0, 5):
```

```
for j in range(0, 5):  
    print '*',  
print
```

注意：第二个 `print` 的缩进和内层的 `for` 是一样的，这表明它是外层 `for` 循环中的语句，每次 `i` 的循环中，它会执行一次。

`print` 后面没有写任何东西，是起到换行的作用，这样，每输出 5 个*，就会换行。

要输出第二个三角图案时，我们需要根据当前外层循环的序数，设置内层循环应当执行的次数。

```
for i in range(0, 5):  
    for j in range(0, i+1):  
        print '*',  
    print
```

内层的 `j` 每次从 0 到 `i+1` 进行循环。

这样，当第一次 `i=0` 时，`j` 就是 `range(0,1)`，只输出 1 个*。

而当最后一次 `i=4` 时，`j` 就是 `range(0,5)`，输出 5 个*。



最后顺便说下，如果有同学用的是 `PyScripter`，或者其他第三方 IDE，可以通过 `debug` 中的 `step`，查看程序是怎样一行一行运行的。`IDLE` 在这方面做得不太好，虽然也可以步进调试，但是很麻烦且不直观，所以就不推荐去用了。

【Python 第 16 课】字符串格式化 2

之前我们说到，可以用 `%` 来构造一个字符串，比如

```
print '%s is easy to learn' % 'Python'
```

有时候，仅仅代入一个值不能满足我们构造字符串的需要。假设你现在有一组学生成绩的数据，你要输出这些数据。在一行中，既要输出学生的姓名，又要输出他的成绩。例如

Mike 's score is 87.

Lily 's score is 95.

在 python 中，你可以这样实现：

```
print "%s's score is %d" % ('Mike', 87)
```

或者

```
name = 'Lily'
score = 95
print "%s's score is %d" % (name, score)
```

无论你有多个值需要代入字符串中进行格式化，只需要在字符串中的合适位置用对应格式的%表示，然后在后面的括号中按顺序提供代入的值就可以了。占位的%和括号中的值在数量上必须相等，类型也要匹配。

('Mike', 87)这种用()表示的一组数据在 python 中被称为元组 (tuple)，是 python 的一种基本数据结构，以后我们还会用到。

【Python 第 17 课】类型转换

python 的几种最基本的数据类型，我们已经见过：

字符串

整数

小数（浮点数）

bool 类型

python 在定义一个变量时不需要给它限定类型。变量会根据赋给它的值，自动决定它的类型。你也可以在程序中，改变它的值，于是也就改变了它的类型。例如

```
a = 1
print a
a = 'hello'
print a
```

```
a = True
print a
```

变量 **a** 先后成为了整数、字符串、**bool** 类型。

虽然类型可以随意改变，但当你对一个特定类型的变量进行操作时，如果这个操作与它的数据类型不匹配，就会产生错误。比如以下几行代码

```
print 'Hello' +1
print 'hello%d' % '123'
```

程序运行时会报错。因为第一句里，字符串和整数不能相加；第二句里，**%d** 需要的是一个整数，而 **'123'** 是字符串。

这种情况下，python 提供了一些方法对数值进行类型转换：

```
int(x) #把 x 转换成整数
float(x) #把 x 转换成浮点数
str(x) #把 x 转换成字符串
bool(x) #把 x 转换成 bool 值
```

上述两个例子就可以写成：

```
print 'Hello' +str(1)
print 'hello%d' % int('123')
```

以下等式的结果均为真：

```
int('123') == 123
float('3.3') == 3.3
str(111) == '111'
bool(0) == False
```

并不是所有的值都能做类型转换，比如 `int('abc')` 同样会报错，python 没办法把它转成一个整数。

另外关于 `bool` 类型的转换，我们会专门再详细说明。大家可以先试试以下结果的值，自己摸索一下转换成 `bool` 类型的规律：

```
bool(-123)
bool(0)
bool('abc')
bool('False')
bool('')
```

【Python 第 18 课】 bool 类型转换

昨天最后留的几句关于 `bool` 类型的转换，其中有一行：

```
bool('False')
print 一下结果，会发现是 True。这是什么原因？
```

因为在 python 中，以下数值会被认为是 `False`：

为 0 的数字，包括 0，0.0

空字符串，包括 "", ""

表示空值的 `None`

空集合，包括 `()`，`[]`，`{}`

其他的值都认为是 `True`。

`None` 是 python 中的一个特殊值，表示什么都没有，它和 0、空字符串、`False`、空集合都不一样。关于集合，我们后面的课程再说。

所以，‘`False`’ 是一个不为空的字符串，当被转换成 `bool` 类型之后，就得到 `True`。

同样 `bool(' ')` 的结果是 `True`，一个空格也不能算作空字符串。

`bool('')` 才是 `False`。

在 `if`、`while` 等条件判断语句里，判断条件会自动进行一次 `bool` 的转换。比如

```
a = '123'
if a:
    print 'this is not a blank string'
```

这在编程中是很常见的一种写法。效果等同于

```
if bool(a)
```

或者

```
if a != "
```

【Python 第 19 课】函数

数学上的函数，是指给定一个输入，就会有唯一输出的一种对应关系。编程语言里的函数跟这个意思差不多，但也有不同。函数就是一块语句，这块语句有个名字，你可以在需要时反复地使用这块语句。它有可能需要输入，有可能会返回输出。

举一个现实中的场景：我们去餐厅吃饭，跟服务员点了菜，过了一会儿，服务员把做好的菜端上来。餐厅的厨房就可以看作是一个函数，我们点的菜单，就是给这个函数的参数；厨师在厨房里做菜的过程就是这个函数的执行过程；做好的菜是返回结果，返回到我们的餐桌上。

我们之前已经用到过 python 里内建的函数，比如 `input` 和 `range`。

以 `range(1,10)` 为例，`range` 是这个函数的名称，后面括号里的 1 和 10 是 `range` 需要的参数。它有返回结果，就是一个从 1 到 9 的序列。

再来看 `input()`，括号里面没有，表示我们没有给参数。函数执行过程中，需要我们从控制台输入一个值。函数的返回结果就是我们输入的内容。

PS: `range` 还可以接受 1 个或 3 个参数，`input` 也可以接受 1 个字符串参数。可以等我以后讲，或去查阅相关资料了解详细。

如果我们要自己写一个函数，就需要去 定义 它。python 里的关键字叫 `def`（`define` 的缩写），格式如下：

```
def sayHello():  
    print 'hello world!'
```

`sayHello` 是这个函数的名字，后面的括号里是参数，这里没有，表示不需要参数。但括号和后面的冒号都不能少。下面缩进的代码块就是整个函数的内容，称作函数体。

然后我们去调用这个函数：

```
sayHello()
```

得到和直接执行 `print 'hello world!'` 一样的结果。



```
Python 2.7.5 Shell
File Edit Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>>
hello world!
hello world!
hello world!
>>>

func.py - C:/Python27/func.py
File Edit Format Run Options Windows Help
def sayHello(): #定义函数sayHello
    print 'hello world!'

sayHello() #调用函数sayHello
sayHello() #可以重复调用
sayHello()
```

【Python 第 20 课】 命令行常用命令

今天茬开话题，说一下命令行（Windows 下叫“命令提示符”，Mac 下叫“终端”）里的常用命令。已经熟悉同学可略过。

打开命令行，我们会看到每行前面都有诸如

C:\Documents and Settings\Crossin>

或者

MyMacBook:~ crossin\$

之类的。

这个提示符表示了当前命令行所在目录。

在这里，我们输入 `python` 就可以进入 `python` 环境了。但今天我们暂时不这么做。

第一个常用的命令是：

`dir` （windows 环境下）

`ls` （mac 环境下）

`dir` 和 `ls` 的作用差不多，都是显示出当前目录下的文件和文件夹。

具体效果可参见文末的附图。

第二个常用命令是：

`cd` 目录名

通过 `dir` 或 `ls` 了解当前目录的结构之后，可以通过“`cd` 目录名”的方式，进入到当前目录下的子目录里。

如果要跳回到上级目录，可以用命令：

`cd ..`

另外，Windows 下如果要写换盘符，需要输入盘符：

比如从 `c` 盘切换到 `d` 盘

`C:\Documents and Settings\Crossin>d:`

有了以上两个命令，就可以在文件目录的迷宫里游荡了。虽然没可视化的目录下的操作那么直观，但是会显得你更像个程序员。。。

于是乎，再说个高阶玩法：现在你可以不用 `idle` 那套东西了，随便找个顺手的文本软件，把你的代码写好，保存好，最好是保存成 `py` 文件。

然后在命令行下进入到 `py` 文件保存的目录，使用命令：

`python` 你把程序保存的文件名

就可以运行你写的程序了。



```
C:\>cd d:

D:\>cd 32Bit

D:\32Bit>dir
驱动器 D 中的卷是 UBOXADDITIONS_4.
卷的序列号是 F5AE-9434

D:\32Bit 的目录
2012-04-03  20:49    <DIR>          .
2012-04-03  20:49    <DIR>          ..
2012-04-03  20:49    <DIR>          OS2
2011-08-17  04:00          756 Readme.txt
                  1 个文件          756 字节
                  3 个目录          0  可用字节

D:\32Bit>cd ..

D:\>cd c:

C:\>cd Python27

C:\Python27>python cmd.py
Hello Terminal!

C:\Python27>
```

嗯，这才像个 `python` 程序员的样！

其他常用命令，诸如拷贝文件、删除文件、新建文件夹之类的，请自行搜索相关资料。很容易的，比如你搜“**mac 终端 常用命令**”，就可以找到很多了。

PS：贴吧里转了一篇关于怎么把 **py** 文件转成别人电脑上也可执行的 **exe** 文件，稍稍有点复杂，想挑战的可以去试试。

【Python 第 21 课】函数的参数

今天发现了一个 iPad 上的游戏，叫 **Cargo-Bot**。这个游戏需要你用指令控制一个机械臂去搬箱子。游戏里蕴含了很多编程的思想，包括循环、函数调用、条件判断、寄存器、递归等等，挺有意思的。更厉害的是，这个游戏是用一个叫 **Codea** 的 app 直接在 iPad 上编写出来的。有 iPad 的同学不妨玩玩看，挑战一下你的“程商”。

言归正传，在 19 课里，我们讲了怎样定义一个自己的函数，但我们没有给他提供输入参数的功能。不能指定参数的函数就好比你去餐厅吃饭，服务员告诉你，不能点菜，有啥吃啥。这显然不能满足很多情况。

所以，如果我们希望自己定义的函数里允许调用者提供一些参数，就把这些参数写在括号里，如果有多个参数，用逗号隔开，如：

```
def sayHello(someone):  
    print someone + ' says Hello!'
```

或者

```
def plus(num1, num2):  
    print num1+num2
```

参数在函数中相当于一个变量，而这个变量的值是在调用函数的时候被赋予的。在函数内部，你可以像过去使用变量一样使用它。

调用带参数的函数时，同样把需要传入的参数值放在括号中，用逗号隔开。要注意提供的参数值的数量和类型需要跟函数定义中的一致。如果这个函数不是你自己写的，你需要先了解它的参数类型，才能顺利调用它。

比如上面两个函数，我们可以直接传入值：

```
sayHello('Crossin')
```

还是注意，字符串类型的值不能少了引号。

或者也可以传入变量：

```
x = 3  
y = 4  
plus(x, y)
```

在这个函数被调用时，相当于做了 `num1=x, num2=y` 这么一件事。所以结果是输出了 7。

【Python 第 22 课】函数应用示例

前两课稍稍介绍了一下函数，但光说概念还是有些抽象了，今天就来把之前那个小游戏用函数改写一下。

我希望有这样一个函数，它比较两个数的大小。

如果第一个数小了，就输出“too small”

如果第一个数大了，就输出“too big”

如果相等，就输出“bingo”

函数还有个返回值，当两数相等的时候返回 **True**，不等就返回 **False**。

于是我们来定义这个函数：

```
def isEqual(num1, num2):  
    if num1<num2:  
        print 'too small'  
        return False;  
    if num1>num2:  
        print 'too big'  
        return False;  
    if num1==num2:  
        print 'bingo'  
        return True
```

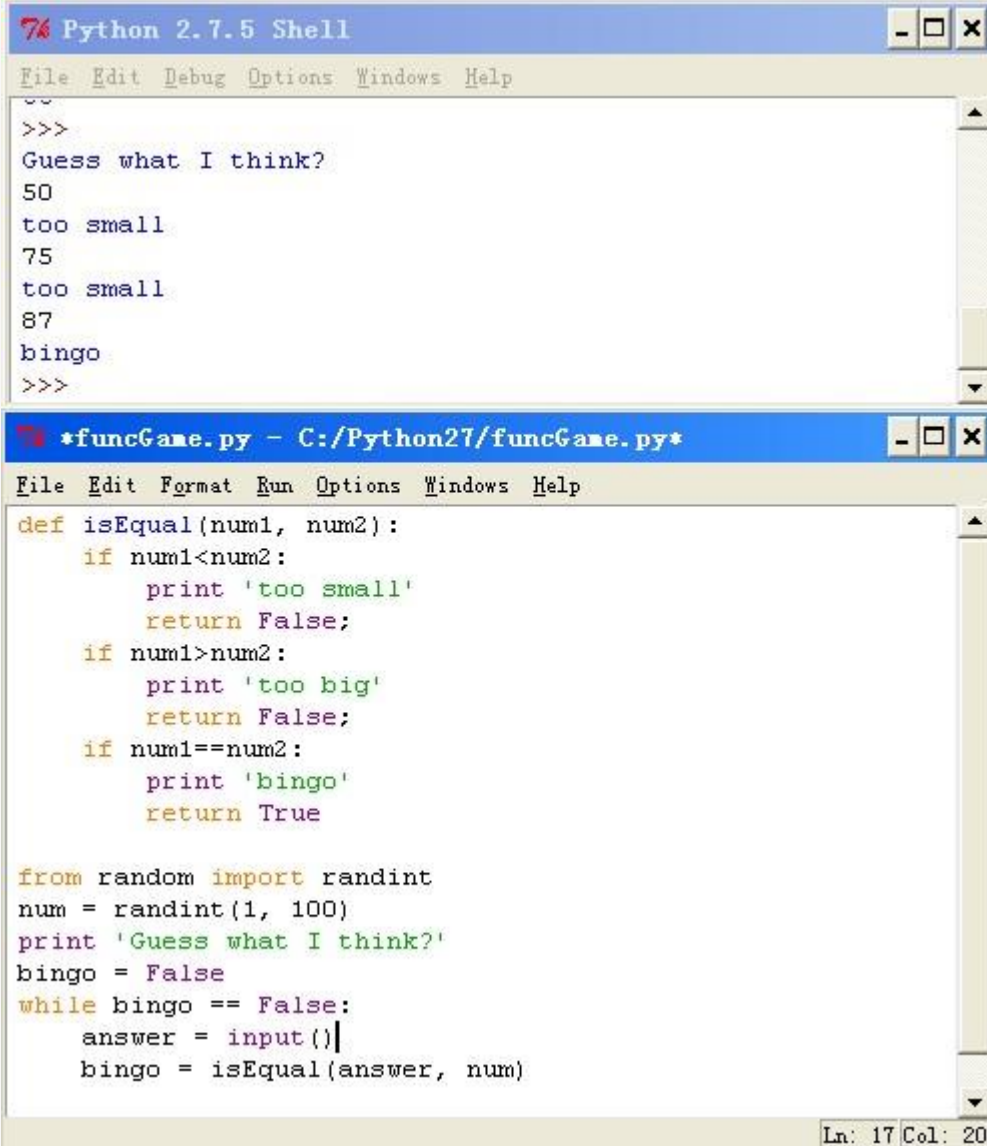
这里说一下，**return** 是函数的结束语句，**return** 后面的值被作为这个函数的返回值。函数中任何地方的 **return** 被执行到的时候，这个函数就会结束。

然后在我们的游戏里使用这个函数：

```
from random import randint  
num = randint(1, 100)  
print 'Guess what I think?'  
bingo = False  
while bingo == False:  
    answer = input()
```

```
bingo = isEqual(answer, num)
```

在 `isEqual` 函数内部，会输出 `answer` 和 `num` 的比较结果，如果相等的话，`bingo` 会得到返回值 `True`，否则 `bingo` 得到 `False`，循环继续。



```
Python 2.7.5 Shell
File Edit Debug Options Windows Help
>>>
Guess what I think?
50
too small
75
too small
87
bingo
>>>

*funcGame.py - C:/Python27/funcGame.py*
File Edit Format Run Options Windows Help
def isEqual(num1, num2):
    if num1<num2:
        print 'too small'
        return False;
    if num1>num2:
        print 'too big'
        return False;
    if num1==num2:
        print 'bingo'
        return True

from random import randint
num = randint(1, 100)
print 'Guess what I think?'
bingo = False
while bingo == False:
    answer = input()
    bingo = isEqual(answer, num)

Ln: 17 Col: 20
```

函数可以把某个功能的代码分离出来，在需要的时候重复使用，就像拼装积木一样，这会让程序结构更清晰。

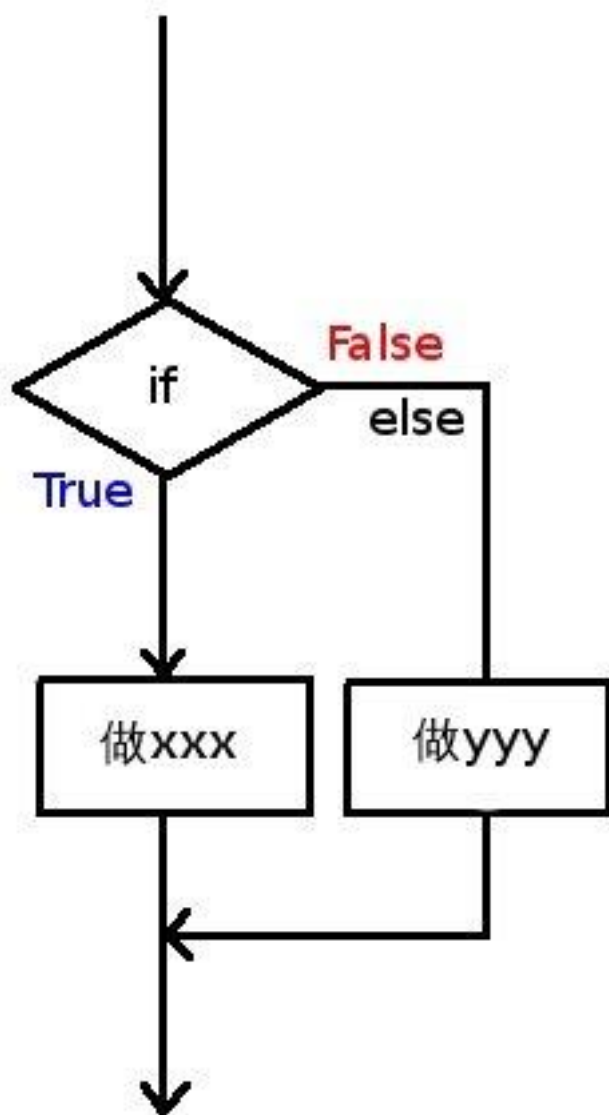
【Python 第 23 课】 if, elif, else

今天补充之前讲过的一个语句：**if**。为什么我跳要着讲，因为我的想法是先讲下最基本的概念，让你能用起来，之后你熟悉了，再说些细节。

关于 **if**，可以发送数字『7』回顾之前的课程。它除了我们之前讲的用法外，还可以配合 **elif** 和 **else** 使用，使程序的运行顺序更灵活。

之前说的 **if**，是：“如果”条件满足，就做 **xxx**，否则就不做。

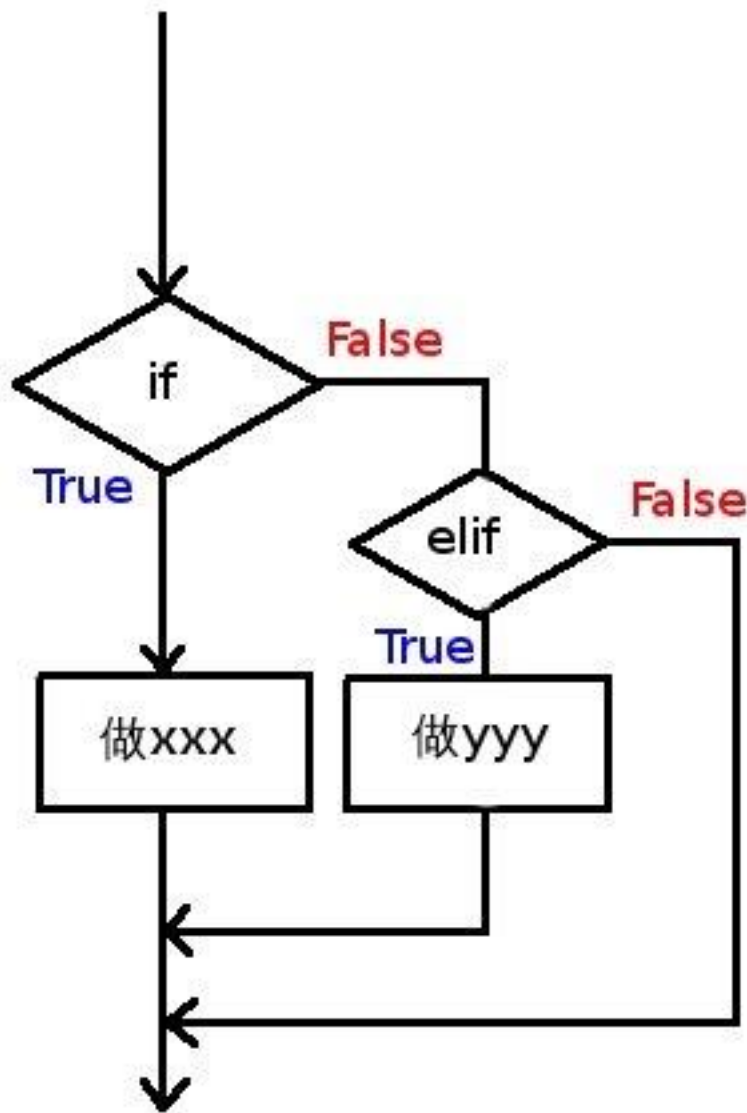
else 顾名思义，就是：“否则”就做 **yyy**。



当 `if` 后面的条件语句不满足时，与之相对应的 `else` 中的代码块将被执行。

```
if a == 1:
    print 'right'
else
    print 'wrong'
```

elif 意为 else if，含义就是：“否则如果”条件满足，就做 yyy。elif 后面需要有一个逻辑判断语句。



当 if 条件不满足时，再去判断 elif 的条件，如果满足则执行其中的代码块。

```
if a == 1:
    print 'one'
elif a == 2:
```

```
print 'two'
```

`if`, `elif`, `else` 可组成一个整体的条件语句。

`if` 是必须有的；

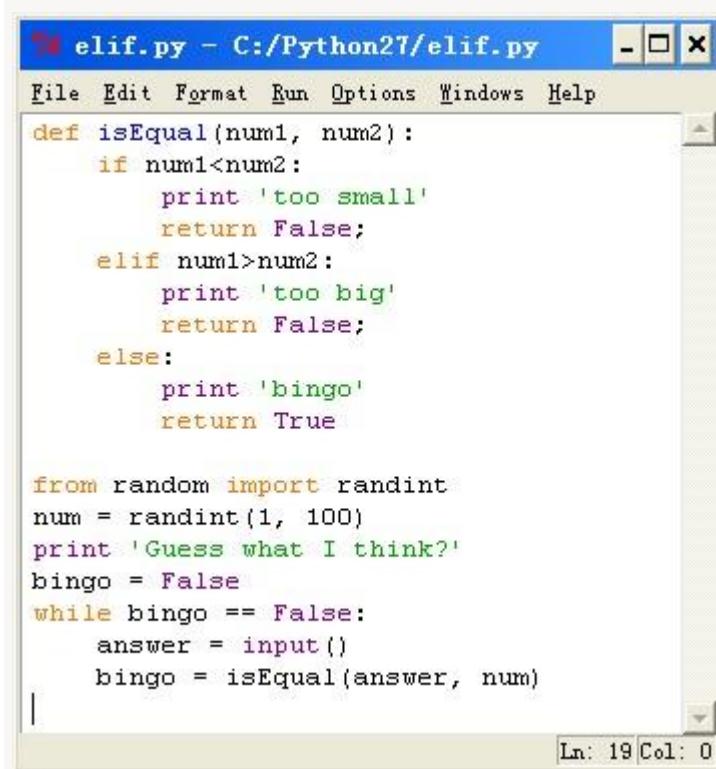
`elif` 可以没有，也可以有很多个，每个 `elif` 条件不满足时会进入下一个 `elif` 判断；

`else` 可以没有，如果有的话只能有一个，必须在条件语句的最后。

```
if a == 1:
    print 'one'
elif a == 2:
    print 'two'
elif a == 3:
    print 'three'
else:
    print 'too many'
```

我们昨天刚改写的小游戏中的函数 `isEqual`，用了三个条件判断，我们可以再改写成一个包含 `if...elif...else` 的结构：

```
def isEqual(num1, num2):
    if num1 < num2:
        print 'too small'
        return False;
    elif num1 > num2:
        print 'too big'
        return False;
    else:
        print 'bingo'
        return True
```



```
def isEqual(num1, num2):  
    if num1<num2:  
        print 'too small'  
        return False;  
    elif num1>num2:  
        print 'too big'  
        return False;  
    else:  
        print 'bingo'  
        return True  
  
from random import randint  
num = randint(1, 100)  
print 'Guess what I think?'  
bingo = False  
while bingo == False:  
    answer = input()  
    bingo = isEqual(answer, num)
```

【Python 第 24 课】if 的嵌套

和 for 循环一样，if 也可以嵌套使用，即在一个 if/elif/else 的内部，再使用 if。这有点类似于电路的串联。

if 条件 1:

 if 条件 2:

 语句 1

 else:

 语句 2

else:

 if 条件 2:

 语句 3

 else:

 语句 4

在上面这个两层 if 的结构中，当

条件 1 为 True，条件 2 为 True 时，

执行语句 1；

条件 1 为 True，条件 2 为 False 时，

执行语句 2；

条件 1 为 False，条件 2 为 True 时，

执行语句 3；

条件 1 为 False，条件 2 为 False 时，

执行语句 4。

假设需要这样一个程序：

我们先向程序输入一个值 x ，再输入一个值 y 。 (x,y) 表示一个点的坐标。

程序要告诉我们这个点处在坐标系的哪一个象限。

$x \geq 0$ ， $y \geq 0$ ，输出 1；

$x < 0$ ， $y \geq 0$ ，输出 2；

$x < 0$ ， $y < 0$ ，输出 3；

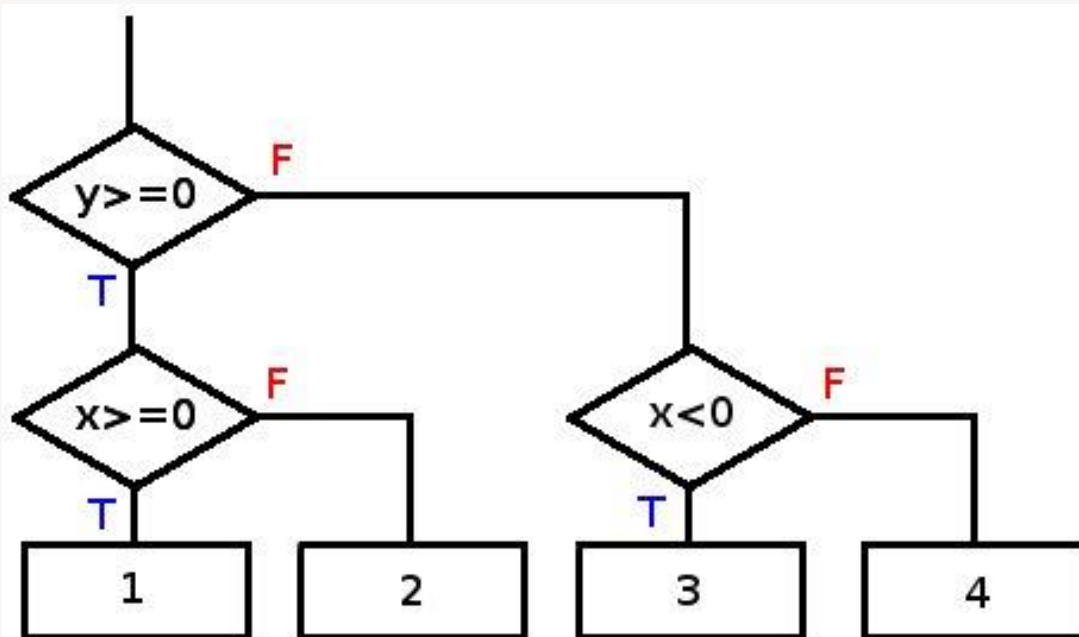
$x \geq 0$ ， $y < 0$ ，输出 4。

你可以分别写 4 个 if，也可以用 if 的嵌套：

```
if y >= 0:
    if x >= 0:
        print 1
    else:
        print 2
else:
```

```
if x < 0:
    print 3
else:
    print 4
```

从流程图上来看，应该是这样。



【Python 第 25 课】初探 list

昨天课程里的例子有点没说清楚，有同学表示写在程序里发生了错误。因为我当时写这个代码片段时，心里假想着这是在一个函数的内部，所以用了 `return` 语句。如果你没有把它放在函数里，那 `return` 的话就会出错，你可以换成 `print`。

今天要说一个新概念--`list`，中文可以翻译成列表，是用来处理一组有序项目的数据结构。想象一下你的购物清单、待办工作、手机通讯录等等，它们都可以看作是一个列表。说它是新概念也不算确切，因为我们之前已经用过它，就在这个语句里：

```
for i in range(1, 10):  
    #此处略过数行代码
```

看出来 **list** 在哪里了吗？你试一下：

```
print range(1,10)
```

得到的结果是：

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

这就是一个 **list**。它由 **range** 产生。把上面那个 **for** 循环语句写成：

```
l = range(1, 10)  
for i in l:
```

效果是一样的。

于是可以看出，**for** 循环做的事情其实就是遍历一个列表中的每一项，每次循环都把当前项赋值给一个变量（这里是 **i**），直到列表结束。

我们也可以定义自己的列表，格式就是用中括号包围、逗号隔开的一组数值：

```
l = [1, 1, 2, 3, 5, 8, 13]
```

可以用 **print** 输出这个列表：

```
print l
```

同样也可以用 `for...in` 遍历这个列表，依次输出了列表中的每一项：

```
for i in l:  
    print l,
```

列表中的元素也可以是别的类型，比如：

```
l = ['meat', 'egg', 'fish', 'milk']
```

甚至是不同类型的混合：

```
l = [365, 'everyday', 0.618, True]
```

`l` 身为一个列表，有一些特有的功能，这个我们下回再说。

【Python 第 26 课】操作 list

上周给 `list` 开了个头，知道了什么是 `list`。假设我们现在有一个 `list`：

```
l = [365, 'everyday', 0.618, True]
```

除了用 `for...in` 遍历 `l` 中的元素，我们还能做点啥？

1. 访问 list 中的元素

`list` 中的每个元素都对应一个递增的序号。与现实中习惯的序号不同在于，计算机中的计数通常都是从 0 开始，`python` 也不例外。如果你记不清这个而导致了错误，请去听一下孙燕姿的《爱从零开始》。

要访问 `l` 中的第 1 个元素 `365`，只要用 `l[0]` 就可以了。依次类推，

```
print l[1]
```

就会输出 `'everyday'`

注意，你不能访问一个不存在的元素，比如 `l[10]`，程序就会报错，提示你 `index` 越界了。

2. 修改 list 中的元素

修改 `list` 中的某一个元素，只需要直接给那个元素赋值就可以了：

```
l[0] = 123
```

输出 `l`，得到 `[123, 'everyday', 0.618, True]`，第 1 个元素已经从 365 被改成了 123。

3. 向 list 中添加元素

`list` 有一个 `append` 方法，可以增加元素。以 `l` 这个列表为例，调用的方法是：

```
l.append(1024)
```

输出 `l`，你会看到 `[123, 'everyday', 0.618, True, 1024]`，1024 被添加到了 `l`，成为最后一个元素。（第一个元素在上一步被改成了 123）

然后同样可以用 `l[4]` 得到 1024。

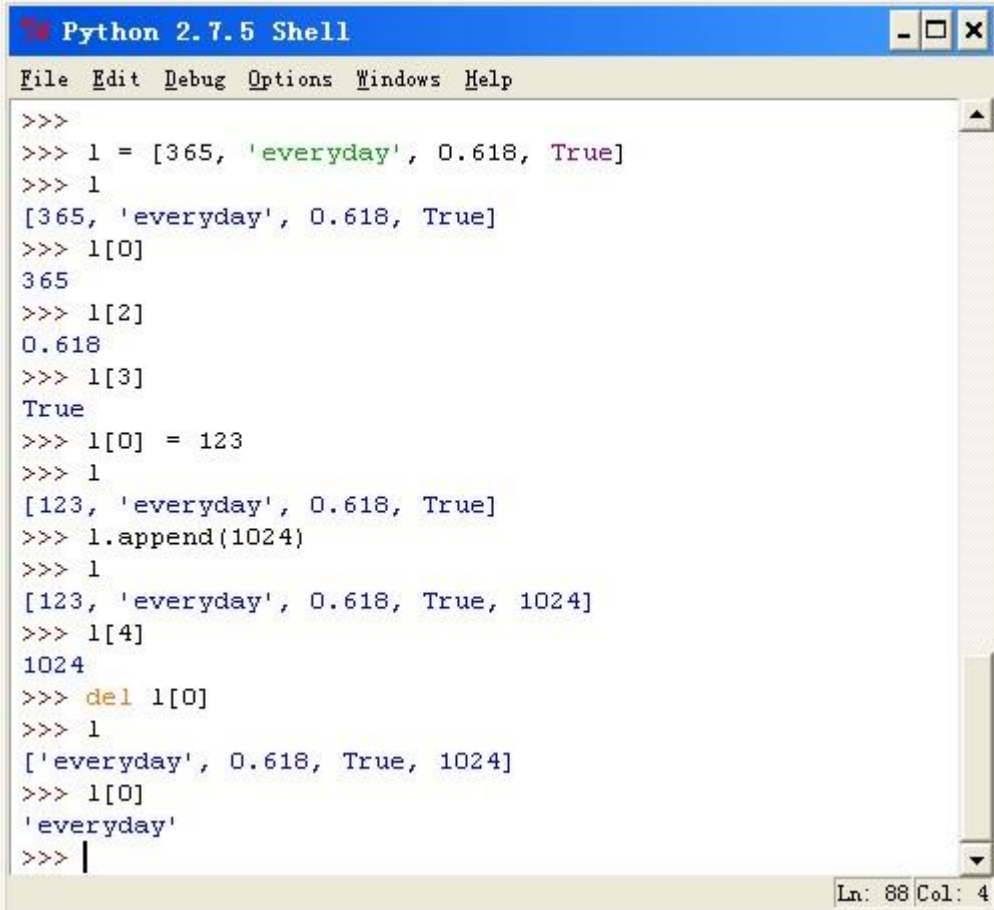
4. 删除 list 中的元素

删除 `list` 中的某一个元素，要用到 `del`：

```
del l[0]
```

输出 l，得到['everyday', 0.618, True, 1024]。这时候再调用 l[0]，会得到'everyday'，其他元素的序号也相应提前。

以上这些命令，你可以直接在 python shell 中尝试。

A screenshot of a Python 2.7.5 Shell window. The window has a blue title bar with the text 'Python 2.7.5 Shell' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Debug', 'Options', 'Windows', and 'Help'. The main area is a text editor showing a series of Python commands and their outputs. The commands are: 1. l = [365, 'everyday', 0.618, True] 2. l 3. l[0] 4. l[2] 5. l[3] 6. l[0] = 123 7. l 8. l.append(1024) 9. l 10. l[4] 11. del l[0] 12. l 13. l[0] The outputs are: [365, 'everyday', 0.618, True], 365, 0.618, True, [123, 'everyday', 0.618, True], [123, 'everyday', 0.618, True, 1024], 1024, ['everyday', 0.618, True, 1024], and 'everyday'. The status bar at the bottom right shows 'Ln: 88 Col: 4'.

#==== 点球小游戏 ====#

我打算从今天开始，每天说一点这个小游戏的做法。方法有很多种，我只是提供一种参考。你可以按照自己喜欢的方式去做，那样她才是属于你的游戏。

先说一下方向的设定。我的想法比较简单，就是左中右三个方向，用字符串来表示。射门或者扑救的时候，直接输入方向。所以这里我准备用

`raw_input`。有同学是用 1-8 的数字来表示八个方向，每次输入一个数字，这也是可以的。不过这样守门员要扑住的概率可就小多了。

至于电脑随机挑选方向，如果你是用数字表示，就用我们之前讲过的 `randint` 来随机就行。不过我这次打算用 `random` 的另一个方法：`choice`。它的作用是从一个 `list` 中随机挑选一个元素。

于是，罚球的过程可以这样写：

```
from random import choice
print 'Choose one side to shoot:'
print 'left, center, right'
you = raw_input()
print 'You kicked ' + you
direction = ['left', 'center', 'right']
com = choice(direction)
print 'Computer saved ' + com
if you != com:
    print 'Goal!'
else:
    print 'Oops...'
```

反之亦然，不赘述。

`list` 有两类常用操作：索引(`index`)和切片(`slice`)。

昨天我们说的用 `[]` 加序号访问的方法就是索引操作。

除了指定位置进行索引外，`list` 还可以处理负数的索引。继续用昨天的例子：

```
l = [365, 'everyday', 0.618, True]
```

`l[-1]`表示 `l` 中的最后一个元素。

`l[-3]`表示倒数第 3 个元素。

切片操作符是在 `[]` 内提供一对可选数字，用 `:` 分割。冒号前的数表示切片的开始位置，冒号后的数字表示切片到哪里结束。同样，计数从 0 开始。注意，开始位置包含在切片中，而结束位置不包括。

```
l[1:3]
```

得到的结果是 `['everyday', 0.618]`。

如果不指定第一个数，切片就从列表第一个元素开始。

如果不指定第二个数，就一直到最后元素结束。

都不指定，则返回整个列表的一个拷贝。

```
l[:3]
```

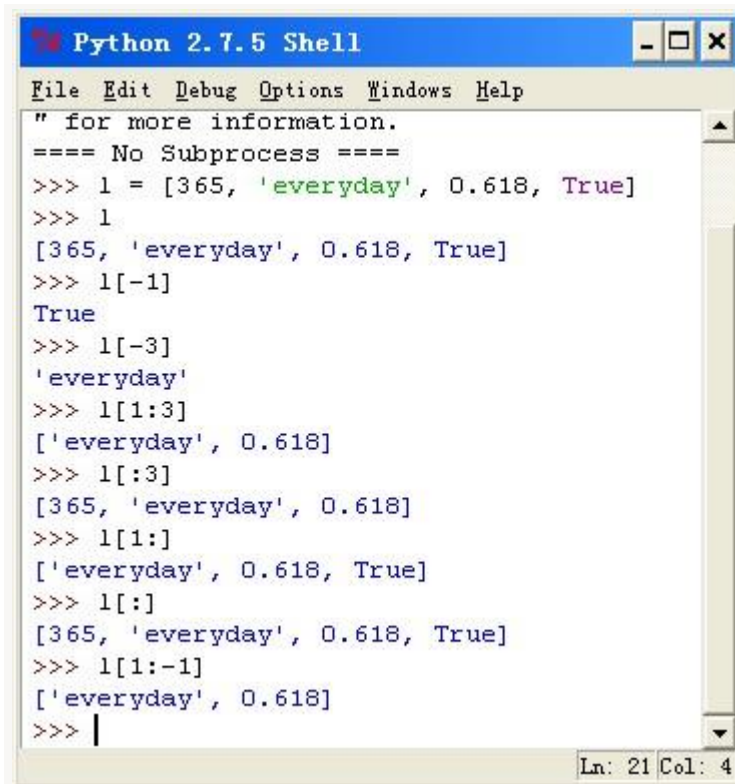
```
l[1:]
```

```
l[:]
```

同索引一样，切片中的数字也可以使用负数。比如：

```
l[1:-1]
```

得到 `['everyday', 0.618]`

A screenshot of a Python 2.7.5 Shell window. The window has a blue title bar with the text "Python 2.7.5 Shell" and standard window controls. Below the title bar is a menu bar with "File", "Edit", "Debug", "Options", "Windows", and "Help". The main area is a text editor showing a series of Python commands and their outputs. The commands include creating a list 'l' with elements [365, 'everyday', 0.618, True] and performing various slicing operations on it. The status bar at the bottom right shows "Ln: 21 Col: 4".

```
Python 2.7.5 Shell
File Edit Debug Options Windows Help
" for more information.
==== No Subprocess ====
>>> l = [365, 'everyday', 0.618, True]
>>> l
[365, 'everyday', 0.618, True]
>>> l[-1]
True
>>> l[-3]
'everyday'
>>> l[1:3]
['everyday', 0.618]
>>> l[:3]
[365, 'everyday', 0.618]
>>> l[1:]
['everyday', 0.618, True]
>>> l[:]
[365, 'everyday', 0.618, True]
>>> l[1:-1]
['everyday', 0.618]
>>> |
Ln: 21 Col: 4
```

#==== 点球小游戏 ====#

昨天有了一次罚球的过程，今天我就让它循环 5 次，并且记录下得分。
先不判断胜负。

用 `score_you` 表示你的得分，`score_com` 表示电脑得分。开始都为 0，
每进一球就加 1。

```
from random import choice
```

```
score_you = 0
```

```
score_com = 0
```

```
direction = ['left', 'center', 'right']
```

```
for i in range(5):
```

```
    print '==== Round %d - You Kick! ==== ' % (i+1)
```

```
    print 'Choose one side to shoot:'
```

```

print 'left, center, right'
you = raw_input()
print 'You kicked ' + you
com = choice(direction)
print 'Computer saved ' + com
if you != com:
    print 'Goal!'
    score_you += 1
else:
    print 'Oops...'
print 'Score: %d(you) - %d(com)\n' % (score_you, score_com)

print '==== Round %d - You Save! ==== ' % (i+1)
print 'Choose one side to save:'
print 'left, center, right'
you = raw_input()
print 'You saved ' + you
com = choice(direction)
print 'Computer kicked ' + com
if you == com:
    print 'Saved!'
else:
    print 'Oops...'
    score_com += 1
print 'Score: %d(you) - %d(com)\n' % (score_you, score_com)

```

注意：手机上代码有可能会被换行。

这段代码里有两段相似度很高，想想是不是可以有办法可以用个函数把它们分离出来。

```
*soccer1.py - C:/Python27/soccer1.py*
File Edit Format Run Options Windows Help

from random import choice

score_you = 0
score_com = 0
direction = ['left', 'center', 'right']

for i in range(5):
    print '==== Round %d - You Kick! ==== ' % (i+1)
    print 'Choose one side to shoot:'
    print 'left, center, right'
    you = raw_input() #输入射门方向
    print 'You kicked ' + you
    com = choice(direction) #电脑随机扑救方向
    print 'Computer saved ' + com
    if you != com: #方向不同, 球进
        print 'Goal!'
        score_you += 1 #玩家得分
    else:
        print 'Oops...'
    print 'Score: %d(you) - %d(com)\n' % (score_you, score_com)

    print '==== Round %d - You Save! ==== ' % (i+1)
    print 'Choose one side to save:'
    print 'left, center, right'
    you = raw_input() #输入扑救方向
    print 'You saved ' + you
    com = choice(direction) #电脑随机射门方向
    print 'Computer kicked ' + com
    if you == com: #方向相同, 球被扑出
        print 'Saved!'
    else:
        print 'Oops...'
        score_com += 1 #电脑得分
    print 'Score: %d(you) - %d(com)\n' % (score_you, score_com)

Ln: 31 Col: 9
```

【Python 第 28 课】字符串的分割

字符串和 list 之间有很多不得不说的事情。比如有同学想要用 python 去自动抓取某个网页上的下载链接，那就需要对网页的代码进行处理。处理的过程中，免不了要在字符串和 list 之间进行很多操作。

我们先从最基本的开始。假设你现在拿到了一个英语句子，需要把这个句子中的每一个单词拿出来单独处理。

```
sentence = 'I am an English sentence'
```

这时就需要对字符串进行分割。

```
sentence.split()
```

`split()`会把字符串按照其中的空格进行分割，分割后的每一段都是一个新的字符串，最终返回这些字符串组成一个 `list`。于是得到

```
['I', 'am', 'an', 'English', 'sentence']
```

原来字符串中的空格不再存在。

除了空格外，`split()`同时也会按照换行符`\n`，制表符`\t`进行分割。所以应该说，`split`默认是按照空白字符进行分割。

之所以说默认，是因为 `split` 还可以指定分割的符号。比如你有一个很长的字符串

```
section = 'Hi. I am the one. Bye.'
```

通过指定分割符号为`'.'`，可以把每句话分开

```
section.split('.')
```

得到

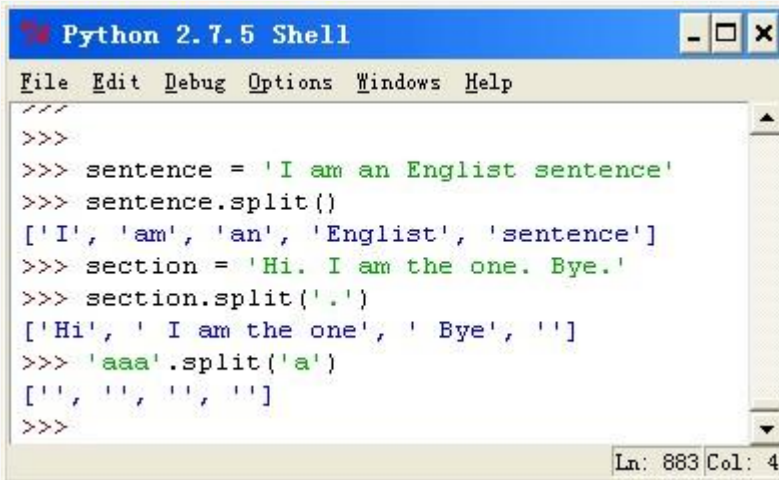

```
['Hi', ' I am the one', ' Bye', '']
```

这时候，'!'作为分割符被去掉了，而空格仍然保留在它的位置上。

注意最后那个空字符串。每个'!'都会被作为分割符，即使它的后面没有其他字符，也会有一个空串被分割出来。例如

```
'aaa'.split('a')
```

将会得到['', '', '', '']，由四个空串组成的 list。

A screenshot of a Python 2.7.5 Shell window. The title bar says "Python 2.7.5 Shell". The menu bar includes "File", "Edit", "Debug", "Options", "Windows", and "Help". The shell contains the following code:

```
///  
>>>  
>>> sentence = 'I am an Englist sentence'  
>>> sentence.split()  
['I', 'am', 'an', 'Englist', 'sentence']  
>>> section = 'Hi. I am the one. Bye.'  
>>> section.split('.')  
['Hi', ' I am the one', ' Bye', '']  
>>> 'aaa'.split('a')  
['', '', '', '']  
>>>
```

The status bar at the bottom right shows "Ln: 883 Col: 4".

既然有把字符串分割成 list，那也相应就有把 list 连接成字符串，这个明天说。

```
#==== 点球小游戏 ====#
```

在昨天代码的基础上，我们加上胜负判断，如果 5 轮结束之后是平分，就继续踢。

所以我们将一轮的过程单独拿出来作为一个函数 `kick`，在 5 次循环之后再加上一个 `while` 循环。

另外，这里把之前的 `score_you` 和 `score_com` 合并成了一个 `score` 数组。这里的原因是，要让 `kick` 函数里用到外部定义的变量，需要使用全局变量的概念。暂时想避免说这个，而用 `list` 不存在这个问题。

```
from random import choice
```

```
score = [0, 0]
```

```
direction = ['left', 'center', 'right']
```

```
def kick():
```

```
    print '==== You Kick! ===='
```

```
    print 'Choose one side to shoot:'
```

```
    print 'left, center, right'
```

```
    you = raw_input()
```

```
    print 'You kicked ' + you
```

```
    com = choice(direction)
```

```
    print 'Computer saved ' + com
```

```
    if you != com:
```

```
        print 'Goal!'
```

```
        score[0] += 1
```

```
    else:
```

```
        print 'Oops...'
```

```
    print 'Score: %d(you) - %d(com)\n' % (score[0], score[1])
```

```
    print '==== You Save! ===='
```

```
    print 'Choose one side to save:'
```

```
    print 'left, center, right'
```

```
    you = raw_input()
```

```
    print 'You saved ' + you
```

```
    com = choice(direction)
```

```
    print 'Computer kicked ' + com
```

```
    if you == com:
```

```
        print 'Saved!'
    else:
        print 'Oops...'
        score[1] += 1
    print 'Score: %d(you) - %d(com)\n' % (score[0], score[1])

for i in range(1):
    print '==== Round %d ==== ' % (i+1)
    kick()

while(score[0] == score[1]):
    i += 1
    print '==== Round %d ==== ' % (i+1)
    kick()

if score[0] > score[1]:
    print 'You Win!'
else:
    print 'You Lose.'
```

```
soccer2.py - C:/Python27/soccer2.py
File Edit Format Run Options Windows Help

from random import choice

score = [0, 0]
direction = ['left', 'center', 'right']

def kick():
    print '==== You Kick! ==== '
    print 'Choose one side to shoot:'
    print 'left, center, right'
    you = raw_input()
    print 'You kicked ' + you
    com = choice(direction)
    print 'Computer saved ' + com
    if you != com:
        print 'Goal!'
        score[0] += 1
    else:
        print 'Oops...'
    print 'Score: %d(You) - %d(Com)\n' % (score[0], score[1])

    print '==== You Save! ==== '
    print 'Choose one side to save:'
    print 'left, center, right'
    you = raw_input()
    print 'You saved ' + you
    com = choice(direction)
    print 'Computer kicked ' + com
    if you == com:
        print 'Saved!'
    else:
        print 'Oops...'
        score[1] += 1
    print 'Score: %d(You) - %d(Com)\n' % (score[0], score[1])

for i in range(1):
    print '==== Round %d ==== ' % (i+1)
    kick()

while(score[0] == score[1]):
    i += 1
    print '==== Round %d ==== ' % (i+1)
    kick()

if score[0] > score[1]:
    print 'You Win!'
else:
    print 'You Lose.'
```

Ln: 33 Col: 35

【Python 第 29 课】连接 list

今天要说的方法是 join。它和昨天说的 split 正好相反：split 是把一个字符串分割成很多字符串组成的 list，而 join 则是把一个 list 中的所有字符串连接成一个字符串。

join 的格式有些奇怪，它不是 list 的方法，而是字符串的方法。首先你需要有一个字符串作为 list 中所有元素的连接符，然后再调用这个连接符的 join 方法，join 的参数是被连接的 list：

```
s = ';'
li = ['apple', 'pear', 'orange']
fruit = s.join(li)
print fruit
```

得到结果 'apple;pear;orange'。

从结果可以看到，分号把 list 中的几个字符串都连接了起来。

你也可以直接在 shell 中输入：

```
';'.join(['apple', 'pear', 'orange'])
```

得到同样的结果。

用来连接的字符串可以是多个字符，也可以是一个空串：

```
''.join(['hello', 'world'])
```

得到'helloworld'，字符串被无缝连接在一起。

```
#==== 点球小游戏 ====#
```

昨天的代码已经能实现一个完整的点球比赛过程，但有同学提出：这不符合真实比赛规则，说好的提前结束比赛呢？！

关于这个，我想了下，可以有好几种解决方法，但似乎都有些绕。所以放到明天单独来讲，把这个小游戏收尾。

【Python 第 30 课】字符串的索引和切片

之前说了，字符串和 list 有很多不得不说的。今天就来说说字符串的一些与 list 相似的操作。

1. 遍历

通过 for...in 可以遍历字符串中的每一个字符。

```
word = 'helloworld'
for c in word:
    print c
```

2. 索引访问

通过[]加索引的方式，访问字符串中的某个字符。

```
print word[0]
print word[-2]
```

与 list 不同的是，字符串能通过索引访问去更改其中的字符。

```
word[1] = 'a'
```

这样的赋值是错误的。

3. 切片

通过两个参数，截取一段子串，具体规则和 list 相同。

```
print word[5:7]
```

```
print word[: -5]
```

```
print word[:]
```

4. 连接字符

join 方法也可以对字符串使用，作用就是用连接符把字符串中的每个字符重新连接成一个新字符串。不过觉得这个方法有点鸡肋，不知道在什么场景下会用到。



```
Python 2.7.5 Shell
File Edit Debug Options Windows Help

n.
==== No Subprocess ====
>>> word = 'helloworld'
>>> for c in word:
>>>     print c

h
e
l
l
o
w
o
r
l
d
>>> word[0]
'h'
>>> word[-2]
'l'
>>> word[5:7]
'wo'
>>> word[: -5]
'hello'
>>> word[:]
'helloworld'
>>> ','.join(word)
'h,e,l,l,o,w,o,r,l,d'
>>> |
```

newword = ','.join(word)

Ln: 31 Col: 4

【Python 第 31 课】 读文件

之前，我们写的程序绝大多数都依赖于从命令行输入。假如某个程序需要输入很多数据，比如一次考试的全班学生成绩，再这么输就略显痛苦了。一个常见的办法就是把学生的成绩都保存在一个文件中，然后让程序自己从这个文件里取数据。

要读取文件，先得有文件。我们新建个文件，就叫它 data.txt。在里面随便写上一些话，保存。把这个文件放在接下来你打算保存代码的文件夹下，这么做是为了方便我们的程序找到它。准备工作就绪，可以来写我们的代码了。

打开一个文件的命令很简单：

```
file('文件名')
```

这里的文件名可以用文件的完整路径，也可以是相对路径。因为我们把要读取的文件和代码放在了同一个文件夹下，所以只需要写它的文件名就够了。

```
f = file('data.txt')
```

但这一步只是打开了一个文件，并没有得到其中的内容。变量 f 保存了这个文件，还需要去读取它的内容。你可以通过 read() 函数把文件内所有内容读进一个字符串中。

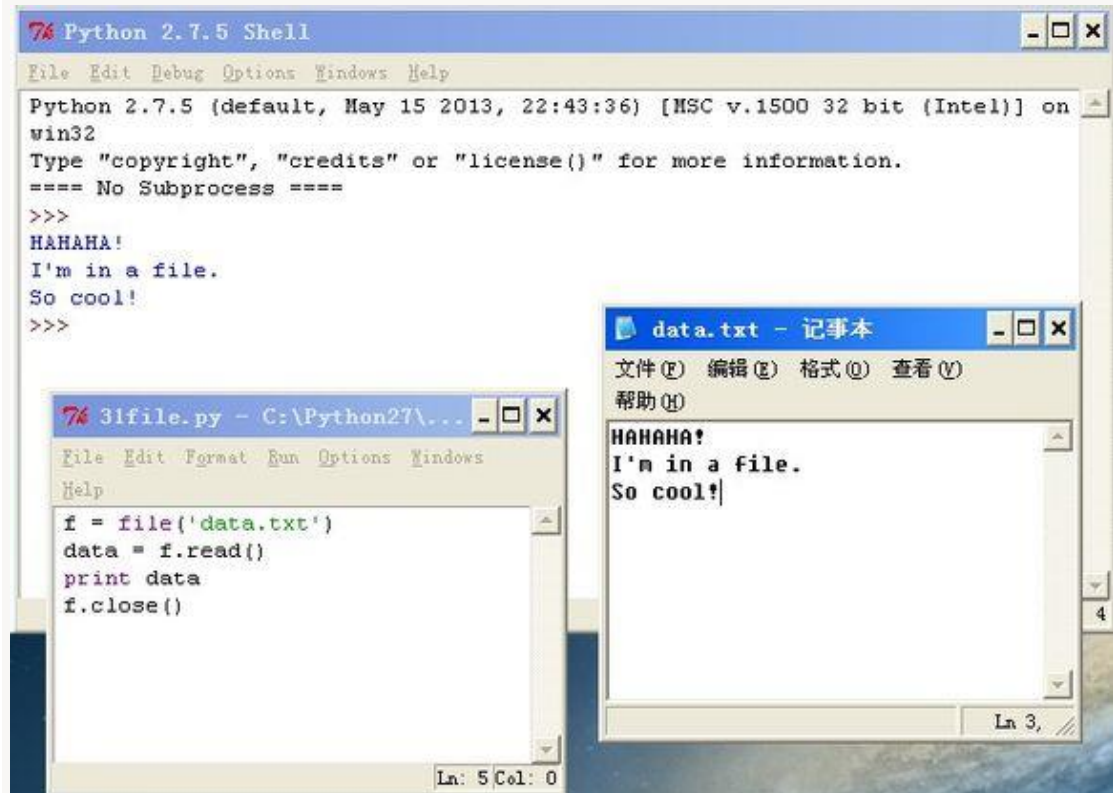
```
data = f.read()
```

做完对文件的操作之后，记得用 close() 关闭文件，释放资源。虽然现在这样一个很短的程序，不做这一步也不会影响运行结果。但养成好习惯，可以避免以后发生莫名的错误。

完整程序示例：

```
f = file('data.txt')
data = f.read()
print data
f.close()
```

是不是很简单？



读取文件内容的方法还有

readline() #读取一行内容

readlines() #把内容按行读取至一个 list 中
去替换程序的第二行，看看它们的区别。

【Python 第 32 课】 写文件

打开文件我们昨天已经讲过。但 python 默认是以只读模式打开文件。如果想要写入内容，在打开文件的时候需要指定打开模式为写入：

```
f = file('output.txt', 'w')
```

'w' 就是 writing，以这种模式打开文件，原来文件中的内容会被你新写入的内容覆盖掉，如果文件不存在，会自动创建文件。

不加参数时，file 为你默认为 'r'，reading，只读模式，文件必须存在，否则引发异常。另外还有一种模式是 'a'，appending。它也是一种写入模式，但你写入的内容不会覆盖之前的内容，而是添加到文件中。

打开文件还有一种方法，就是 open()，用法和 file() 是一致的。

写入内容的方法同样简单：

```
f.write('a string you want to write')
```

write 的参数可以是一个字符串，或者一个字符串变量。

示例程序：

```
data = 'I will be in a file.\nSo cool!'
out = open('output.txt', 'w')
out.write(data)
out.close()
```

在你的程序保存目录下，打开 output.txt 就会看到结果。

留两道课后作业：

1. 从一个文件中读出内容，保存至另一个文件。
2. 从控制台输入一些内容，保存至一个文件。

【Python 第 33 课】 处理文件中的数据

比如我现在拿到一份文档，里面有某个班级里所有学生的平时作业成绩。因为每个人交作业的次数不一样，所以成绩的数目也不同，没交作业的时候就没有分。我现在需要统计每个学生的平时作业总得分。

记得我小的时候，经常有同学被老师喊去做统计分数这种“苦力”。现在电脑普及了，再这么干就太弱了。用 python，几行代码就可以搞定。

看一下我们的文档里的数据：

```
#-- scores.txt
```

```
刘备 23 35 44 47 51
```

```
关羽 60 77 68
```

```
张飞 97 99 89 91
```

```
诸葛亮 100
```

1. 先把文件读进来：

```
f = file('scores.txt')
```

2. 取得文件中的数据。因为每一行都是一条学生成绩的记录，所以用 `readlines`，把每一行分开，便于之后的数据处理：

```
lines = f.readlines()
```

```
f.close()
```

提示: 在程序中, 经常使用 `print` 来查看数据的中间状态, 可以便于你理解程序的运行。
比如这里你可以 `print lines`, 看一下内容被存成了什么格式。

3. 对每一条数据进行处理。按照空格, 把姓名、每次的成绩分割开:

```
for line in lines:  
    data = line.split()
```

接下来的 4、5 两个步骤都是针对一条数据的处理, 所以都是在 `for` 循环的内部。

4. 整个程序最核心的部分到了。如何把一个学生的几次成绩合并, 并保存起来呢? 我的做法是: 对于每一条数据, 都新建一个字符串, 把学生的名字和算好的总成绩保存进去。最后再把这些字符串一起保存到文件中:

```
sum = 0  
for score in data[1:]:  
    sum += int(score)  
result = '%s\t: %d\n' % (data[0], sum)
```

这里几个要注意的点:

对于每一行分割的数据, `data[0]` 是姓名, `data[1:]` 是所有成绩组成的列表。

每次循环中, `sum` 都要先清零。

`score` 是一个字符串, 为了做计算, 需要转成整数值 `int`。

`result` 中, 我加了一个制表符 `\t` 和换行符 `\n`, 让输出的结果更好看些。

5. 得到一个学生的总成绩后, 把它添加到一个 `list` 中。

```
results.append(result)
```

results 需要在循环之前初始化 results = []

6. 最后，全部成绩处理完毕后，把 results 中的内容保存至文件。因为 results 是一个字符串组成的 list，这里我们直接用 writelines 方法：

```
output = file('result.txt', 'w')
output.writelines(results)
output.close()
```

大功告成，打开文件检验一下结果吧。

以下是完整程序，把其中 print 前面的注释符号去掉，可以查看关键步骤的数据状态。不过因为字符编码的问题，list 的中文可能会显示为你看不懂的字符。

```
f = file('scores.txt')
lines = f.readlines()
#print lines
f.close()
```

```
results = []
```

```
for line in lines:
    #print line
    data = line.split()
    #print data

    sum = 0
    for score in data[1:]:
        sum += int(score)
```

```
    result = '%s \t: %d\n' % (data[0], sum)
    #print result

    results.append(result)

#print results
output = file('result.txt', 'w')
output.writelines(results)
```

```
output.close()
```

```
Python 2.7.5 Shell
File Edit Debug Options Windows Help

['\xc1\xf5\xb1\xb8 23 35 44 47 51\n', '\xb9\xd8\xd3\xf0 60 77 68\n',
'\xd5\xc5\xb7\xc9 97 99 89 91\n', '\xd6\xee\xb8\xf0\xc1\xc1 100']
刘备 23 35 44 47 51

['\xc1\xf5\xb1\xb8', '23', '35', '44', '47', '51']
刘备      : 200

关羽 60 77 68

['\xb9\xd8\xd3\xf0', '60', '77', '68']
关羽      : 205

张飞 97 99 89 91

['\xd5\xc5\xb7\xc9', '97', '99', '89', '91']
张飞      : 376

诸葛亮 100
['\xd6\xee\xb8\xf0\xc1\xc1', '100']
诸葛亮    : 100

['\xc1\xf5\xb1\xb8 \t: 200\n', '\xb9\xd8\xd3\xf0 \t: 205\n', '\xd5\xc5\xb7\xc9 \t: 376\n', '\xd6\xee\xb8\xf0\xc1\xc1 \t: 100\n']
>>>

33calscore.py - C:/Python27/33calscore.py
File Edit Format Run Options Windows Help

f = file('scores.txt')
lines = f.readlines()
#print lines
f.close()

results = []

for line in lines:
    #print line
    data = line.split()
    #print data

    sum = 0
    for score in data[1:]:
        sum += int(score)
    result = '%s \t: %d\n' % (data[0], sum)
    #print result

    results.append(result)

#print results
output = file('result.txt', 'w')
output.writelines(results)
output.close()
```

```
scores.txt - 记...
文件(F) 编辑(E) 格式(O) 查看(V)
帮助(H)

刘备 23 35 44 47 51
关羽 60 77 68
张飞 97 99 89 91
诸葛亮 100
```

```
result.txt -...
文件(F) 编辑(E) 格式(O)
查看(V) 帮助(H)

刘备      : 200
关羽      : 205
张飞      : 376
诸葛亮    : 100
```


【Python 第 34 课】 break

while 循环 在条件不满足时 结束，

for 循环 遍历完序列后 结束。

如果在循环条件仍然满足或序列没有遍历完的时候，想要强行跳出循环，就需要用到 break 语句。

```
while True:

    a = raw_input()

    if a == 'EOF':

        break
```

上面的程序不停接受用户输入。当用户输入一行“EOF”时，程序结束。

```
for i in range(10):

    a = raw_input()

    if a == 'EOF':

        break
```

上面的程序接受用户 10 次输入，当用户输入一行“EOF”时，程序提前结束。

回到我们最早的那个猜数字小游戏。用 break 可以加上一个功能，当用户输入负数时，游戏就结束。如此一来，假如有玩家猜了几次之后仍然猜不中，一怒之下想要直接退出游戏，就猜一个负数。

添加的代码是：

```
if answer < 0:

    print 'Exit game...'

    break
```

与 break 类似的还有一个 continue 语句，明天说。

【Python 第 35 课】continue

break 是彻底地跳出循环，而 continue 只是略过本次循环的余下内容，直接进入下一次循环。

在我们前面写的那个统计分数的程序里，如果发现有成绩不足 60 分，就不计入总成绩。当然，你可以用 if 判断来实现这个效果。但我们今天要说另一种方法：continue。

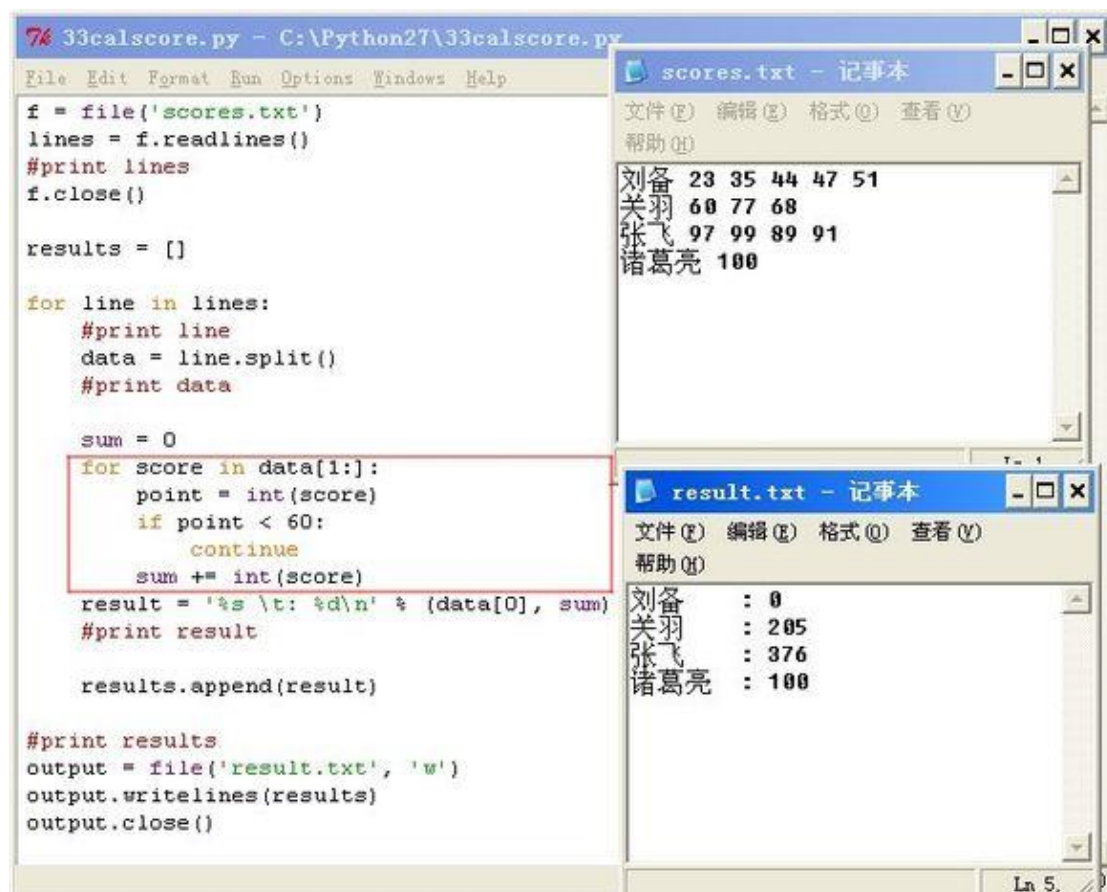
```
for score in data[1:]:
```

```
    point = int(score)
```

```
    if point < 60:
```

```
        continue
```

```
    sum += point
```



注意：无论是 `continue` 还是 `break`，其改变的仅仅是当前所处的最内层循环的运行，如果外层还有循环，并不会因此略过或跳出。

在脑中模拟运行下面这段程序，想想会输出什么结果。再敲到代码里验证一下：

```
i = 0

while i < 5:

    i += 1

    for j in range(3):

        print j

        if j == 2:

            break

    for k in range(3):

        if k == 2:

            continue

        print k

    if i > 3:

        break

print i
```

【Python 第 36 课】 异常处理

在程序运行时，如果我们的代码引发了错误，python 就会中断程序，并且输出错误提示。

比如我们写了一句：

```
print int('0.5')
```

运行后程序得到错误提示：

```
Traceback (most recent call last):
```

```
File "C:/Python27/test.py", line 1, in <module>
```

```
print int('0.5')
```

```
ValueError: invalid literal for int() with base 10: '0.5'
```

意思是，在 test.py 这个文件，第 1 行，`print int('0.5')` 这里，你拿了一个不是 10 进制能够表示的字符，我没法把它转成 int 值。

上面的错误可以避免，但在实际的应用中，有很多错误是开发者无法控制的，例如用户输入了一个不合规定的值，或者需要打开的文件不存在。这些情况被称作“异常”，一个好的程序需要能处理可能发生的异常，避免程序因此而中断。

例如我们去打开一个文件：

```
f = file('non-exist.txt')  
  
print 'File opened!'  
  
f.close()
```

假如这个文件因为某种原因并没有出现在应该出现的文件夹里，程序就会报错：

```
IOError: [Errno 2] No such file or directory: 'non-exist.txt'
```

程序在出错处中断，后面的 print 不会被执行。

在 python 中，可以使用 try...except 语句来处理异常。做法是，把可能引发异常的语句放在 try-块中，把处理异常的语句放在 except-块中。

把刚才那段代码放入 try...except 中：

```
try:

    f = file('non-exist.txt')

    print 'File opened!'

    f.close()

except:

    print 'File not exists.'

print 'Done'
```

当程序在 try 内部打开文件引发异常时，会跳过 try 中剩下的代码，直接跳转到 except 中的语句处理异常。于是输出了“File not exists.”。如果文件被顺利打开，则会输出“File opened!”，而不会去执行 except 中的语句。

但无论如何，整个程序不会中断，最后的“Done”都会被输出。

在 try...except 语句中，try 中引发的异常就像是扔出了一只飞盘，而 except 就是一只灵敏的狗，总能准确地接住飞盘。

【Python 第 37 课】字典

今天介绍一个 python 中的基本类型--字典（dictionary）。

字典这种数据结构有点像我们平常用的通讯录，有一个名字和这个名字对应的信息。在字典中，名字叫做“键”，对应的内容信息叫做“值”。字典就是一个键/值对的集合。

它的基本格式是（key 是键，value 是值）：

```
d = {key1 : value1, key2 : value2 }
```

键/值对用冒号分割，每个对之间用逗号分割，整个字典包括在花括号中。

关于字典的键要注意的是：

1. 键必须是唯一的；
2. 键只能是简单对象，比如字符串、整数、浮点数、bool 值。

list 就不能作为键，但是可以作为值。

举个简单的字典例子：

```
score = {  
  
    '萧峰': 95,  
  
    '段誉': 97,  
  
    '虚竹': 89  
  
}
```

python 字典中的键/值对没有顺序，我们无法用索引访问字典中的某一项，而是要用键来访问。

```
print score['段誉']
```

注意，如果你的键是字符串，通过键访问的时候就需要加引号，如果是数字作为键则不用。
字典也可以通过 for...in 遍历：

```
for name in score:
```

```
print score[name]
```

注意，遍历的变量中存储的是字典的键。

如果要改变某一项的值，就直接给这一项赋值：

```
score['虚竹'] = 91
```

增加一项字典项的方法是，给一个新键赋值：

```
score['慕容复'] = 88
```

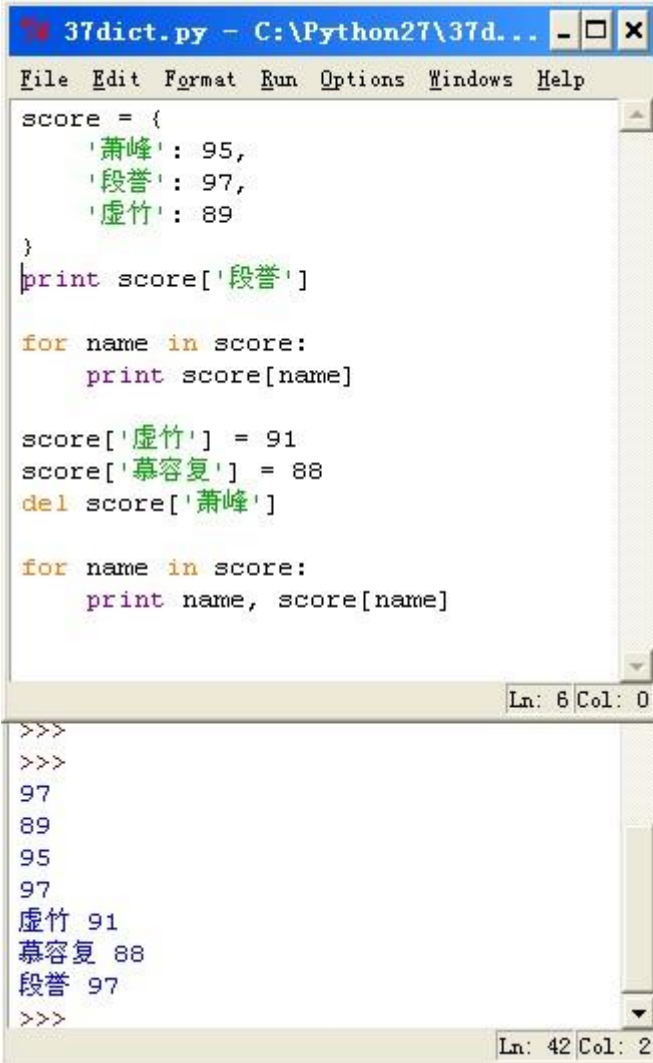
删除一项字典项的方法是 del：

```
del score['萧峰']
```

注意，这个键必须已存在于字典中。

如果你想新建一个空的字典，只需要：

d = {}



The screenshot shows a Python IDE window titled "37dict.py - C:\Python27\37d...". The window contains a script that defines a dictionary 'score' with three entries: '萧峰' (95), '段誉' (97), and '虚竹' (89). It then prints the value for '段誉', iterates over the dictionary to print each name and score, updates '虚竹' to 91 and adds '慕容复' (88), deletes '萧峰', and iterates again to print the updated dictionary. The output window below shows the execution results: three blank lines followed by the scores 97, 89, and 95, then the updated entries '虚竹 91', '慕容复 88', and '段誉 97'.

```
File Edit Format Run Options Windows Help

score = {
    '萧峰': 95,
    '段誉': 97,
    '虚竹': 89
}
print score['段誉']

for name in score:
    print score[name]

score['虚竹'] = 91
score['慕容复'] = 88
del score['萧峰']

for name in score:
    print name, score[name]
```

Ln: 6 Col: 0

```
>>>
>>>
97
89
95
97
虚竹 91
慕容复 88
段誉 97
>>>
```

Ln: 42 Col: 2

【Python 第 38 课】 模块

python 自带了功能丰富的标准库，另外还有数量庞大的各种第三方库。使用这些“巨人的”代码，可以让开发事半功倍，就像用积木一样拼出你要的程序。

使用这些功能的基本方法就是使用模块。通过函数，可以在程序里重用代码；通过模块，则可以重用别的程序中的代码。

模块可以理解为是一个包含了函数和变量的 py 文件。在你的程序中引入了某个模块，就可以使用其中的函数和变量。

来看一个我们之前使用过的模块：

```
import random
```

import 语句告诉 python，我们要用 random 模块中的内容。然后便可以使用 random 中的方法，比如：

```
random.randint(1, 10)
random.randrange([1, 3, 5])
```

注意，函数前面需要加上“random.”，这样 python 才知道你是要调用 random 中的方法。

想知道 random 有哪些函数和变量，可以用 dir() 方法：

```
dir(random)
```

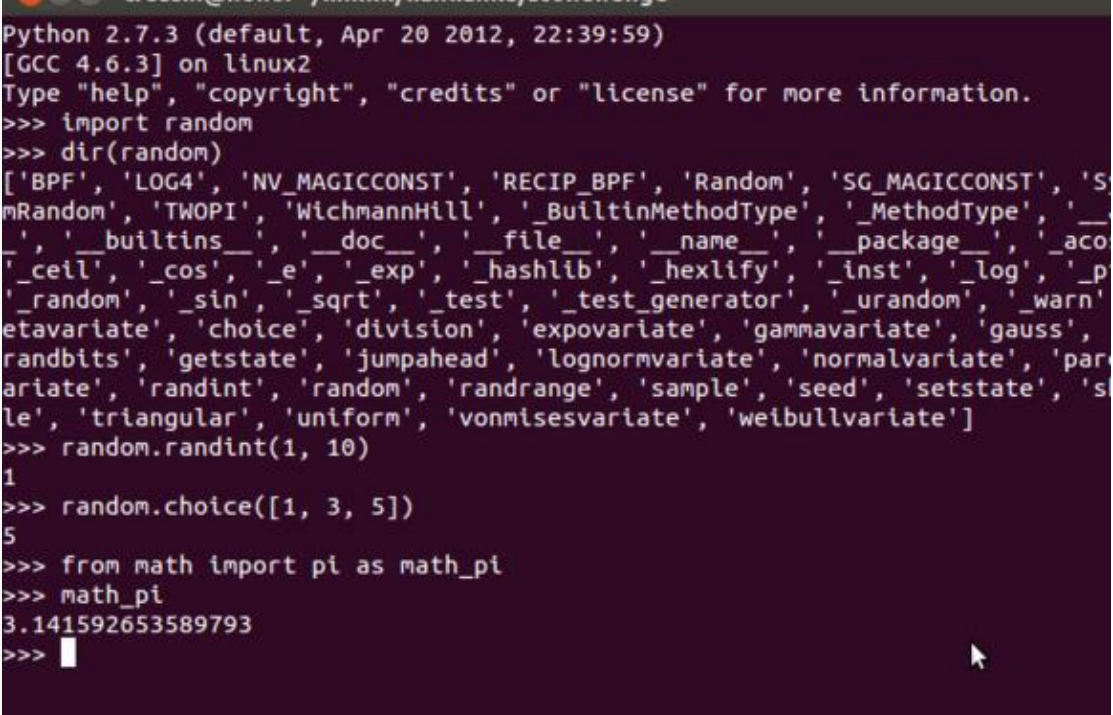
如果你只是用到 random 中的某一个函数或变量，也可以通过 from...import... 指明：

```
from math import pi
print pi
```

为了便于理解和避免冲突，你还可以给引入的方法换个名字：

```
from math import pi as math_pi
print math_pi
```

想要了解 python 有哪些常用库，可自行搜索。我在群共享里上传了一份中文版的 python 标准库的非官方文档，供参考。



```
Python 2.7.3 (default, Apr 20 2012, 22:39:59)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'S
mRandom', 'TWOPI', 'WichmannHill', '_BuiltinMethodType', '_MethodType', '_
', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__ac
', '_ceil', '_cos', '_e', '_exp', '_hashlib', '_hexlify', '_inst', '_log', '_p
', '_random', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn
etavariate', 'choice', 'division', 'expovariate', 'gammavariate', 'gauss',
randbits', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate', 'par
ariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 's
le', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
>>> random.randint(1, 10)
1
>>> random.choice([1, 3, 5])
5
>>> from math import pi as math_pi
>>> math_pi
3.141592653589793
>>> █
```

【Python 第 39 课】用文件保存游戏（1）

到目前为止，python 最入门的语法我们都已经有所涉及，相信大家一路学过来，多少也能写出一些小程序。在接下来的课程中，我会基于实例来更深入地介绍 python
现在，我要在最早我们开发的那个猜数字游戏的基础上，增加保存成绩的功能。用到的方法就是前几课讲过的文件读写。今天是第一部分。

在动手写代码前，先想清楚我们要解决什么问题，打算怎么去解决。你可以选择根据每次游戏算出一个得分，记录累计的得分。也可以让每次猜错都扣 xx 分，猜对之后再加 xx 分，记录当前分数。而我现在打算记录下我玩了多少次，最快猜出来的轮数，以及平均每次猜对用的轮数。

于是，我要在文件中记录 3 个数字，如：

```
3 5 31
```

它们分别是：总游戏次数，最快猜出的轮数，和猜过的总轮数（这里我选择记录总轮数，然后每次再算出平均轮数）

接下来可以往代码里加功能了，首先是读取成绩。新建好一个 game.txt，里面写上：

```
0 0 0
```

作为程序的初始数据。

用之前的方法，读入文件：

```
f = open('e:\py\game.txt')  
score = f.read().split()
```

这里，我用了 open 方法，它和 file() 的效果一样。另外，我还用了绝对路径。当你写这个程序时，记得用你自己电脑上的路径。

为便于理解，把数据读进来后，分别存在 3 个变量中。

```
game_times = int(score[0])  
min_times = int(score[1])  
total_times = int(score[2])
```

平均轮数根据总轮数和游戏次数相除得到：

```
avg_times = float(total_times) / game_times
```

注意两点：

1. 我在 total_times 前加上了 float，把它转成了浮点数类型再进行除法运算。如果不这样做，两个整数相除的结果会默认为整数，而且不是四舍五入。

2. 因为 0 是不能作为除数的，所以这里还需要加上判断：

```
if game_times > 0:
    avg_times = float(total_times) / game_times
else:
    avg_times = 0
```

然后，在让玩家开始猜数字前，输出他之前的成绩信息：

```
print '你已经玩了%d次，最少%d轮猜出答案，平均%.2f轮猜出答案' % (game_times,
min_times, avg_times)
```

%.2f 这样的写法我们以前也用过，作用是保留两位小数。

好了，运行程序看一下效果：

你已经玩了 0 次，最少 0 轮猜出答案，平均 0 轮猜出答案

由于还没有做保存功能，我们手动去文件里改一下成绩看运行效果。（其实有些小游戏就可以用类似的方法作弊）

下一课，我们要把真实的游戏数据保存到文件中。



【Python 第 40 课】用文件保存游戏（2）

话接上回。我们已经能从文件中读取游戏成绩数据了，接下来就要考虑，怎么把我们每次游戏的结果保存进去。

首先，我们需要有一个变量来记录每次游戏所用的轮数：

```
times = 0
```

然后在游戏每进行一轮的时候，累加这个变量：

```
times += 1
```

当游戏结束后，我们要把这个变量的值，也就是本次游戏的数据，添加到我们的记录中。

如果是第一次玩，或者本次的轮数比最小轮数还少，就记录本次成绩为最小轮数：

```
if game_times == 0 or times < min_times:  
    min_times = times
```

把本次轮数加到游戏总轮数里：

```
total_times += times
```

把游戏次数加 1：

```
game_times += 1
```

现在有了我们需要的数据，把它们拼成我们需要存储的格式：

```
result = '%d %d %d' % (game_times, min_times, total_times)
```

写入到文件中：

```
f = open('e:\py\game.txt', 'w')  
f.write(result)  
f.close()
```

The screenshot shows a Python 2.7.5 Shell window and a Notepad window. The Python shell window displays the execution of a script named 40savegame2.py. The script reads data from a file named game.txt, which contains the number of times the game has been played, the minimum number of guesses, and the total number of guesses. The script then prompts the user to guess a number between 1 and 100. The user enters 50, which is too small, then 75, which is too big, then 63, which is too big, then 57, which is too big, then 53, which is too small, and finally 55, which is BINGO! The script then prompts the user to restart the game. The Notepad window shows the contents of game.txt, which are 5 8 61.

```
40savegame2.py - E:/py/40savegame2.py
File Edit Format Run Options Windows Help

from random import randint

f = open('e:\py\game.txt')
score = f.read().split()
f.close()
game_times = int(score[0])
min_times = int(score[1])
total_times = int(score[2])
if game_times > 0:
    avg_times = float(total_times) / game_times
else:
    avg_times = 0
print '你已经玩了%d次, 最少%d轮猜出答案, 平均%.2f轮猜出答案' % (
    game_times, min_times, avg_times)

num = randint(1, 100)
times = 0 #记录本次游戏轮数
print 'Guess what I think?'
bingo = False
while bingo == False:
    times += 1 #轮数+1
    answer = input()
    if answer < num:
        print 'too small!'
    if answer > num:
        print 'too big!'
    if answer == num:
        print 'BINGO!'
        bingo = True

#如果是第一次玩, 或者轮数比最小轮数少, 则更新最小轮数
if game_times == 0 or times < min_times:
    min_times = times
total_times += times #总游戏轮数增加
game_times += 1 #游戏次数增加
result = '%d %d %d' % (game_times, min_times, total_times)
f = open('e:\py\game.txt', 'w')
f.write(result)
f.close()

Python 2.7.5 Shell - E:/py/40saveg...
File Edit Shell Debug Options Windows Help

>>> ===== RESTART
>>>
>>> 你已经玩了7次, 最少8轮猜出答案, 平均11.00轮猜出答案
Guess what I think?
50
too small!
75
too big!
63
too big!
57
too big!
53
too small!
55
BINGO!
>>> ===== RESTART
>>>
>>> 你已经玩了8次, 最少6轮猜出答案, 平均10.38轮猜出答案
Guess what I think?

game.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
帮助(H)
5 8 61
```

按照类似的方法, 你也可以记录一些其他的数据, 比如设定一种记分规则作为游戏得分。虽然在这个小游戏里, 记录成绩并没有太大的乐趣, 但通过文件来记录数据的方法, 以后会在很多程序中派上用场。

【Python 第 41 课】用文件保存游戏 (3)

你的小游戏现在已经可以保存成绩了，但只有一组成绩，不管谁来玩，都会算在里面。所以今天我还要加上一个更多的功能：存储多组成绩。玩家需要做的就是，在游戏开始前，输入自己的名字。而我会根据这个名字记录他的成绩。这个功能所用到的内容我们几乎都说过，现在要把它们结合起来。

首先要输入名字，这是我们用来区分玩家成绩的依据：

```
name = raw_input('请输入你的名字：')
```

接下来，我们读取文件。与之前不同，我们用 `readlines` 把每组成绩分开来：

```
lines = f.readlines()
```

再用一个字典来记录所有的成绩：

```
scores = {}  
for l in lines:  
    s = l.split()  
    scores[s[0]] = s[1:]
```

这个字典中，每一项的 `key` 是玩家的名字，`value` 是一个由剩下的数据组成的数组。这里每一个 `value` 就相当于我们之前的成绩数据。

我们要找到当前玩家的数据：

```
score = scores.get(name)
```

字典类的 `get` 方法是按照给定 `key` 寻找对应项，如果不存在这样的 `key`，就返回空值 `None`。

所以如果没有找到该玩家的数据，说明他是一个新玩家，我们给他初始化一组成绩：

```
if score is None:  
    score = [0, 0, 0]
```

这是我们拿到的 score, 已经和上一课中的 score 一样了, 因此剩下的很多代码都不用改动。当游戏结束, 记录成绩的时候, 和之前的方法不一样。我们不能直接把这次成绩存到文件里, 那样就会覆盖掉别人的成绩。必须先把成绩更新到 scores 字典中, 再统一写回文件中。

把成绩更新到 scores 中, 如果没有这一项, 会自动生成新条目:

```
scores[name] = [str(game_times), str(min_times), str(total_times)]
```

对于每一项成绩, 我们要将其格式化:

```
result = ''
for n in scores:
    line = n + ' ' + ' '.join(scores[n]) + '\n'
    result += line
```

把 scores 中的每一项按照“名字 游戏次数 最低轮数 总轮数\n”的格式拼成字符串, 再全部放到 result 里, 就得到了我们要保存的结果。

最后就和之前一样, 把 result 保存到文件中。

```
*41savegame3.py - E:/py/41savegame3.py*
File Edit Format Run Options Windows Help

from random import randint

name = raw_input('请输入你的名字:') #输入玩家名字

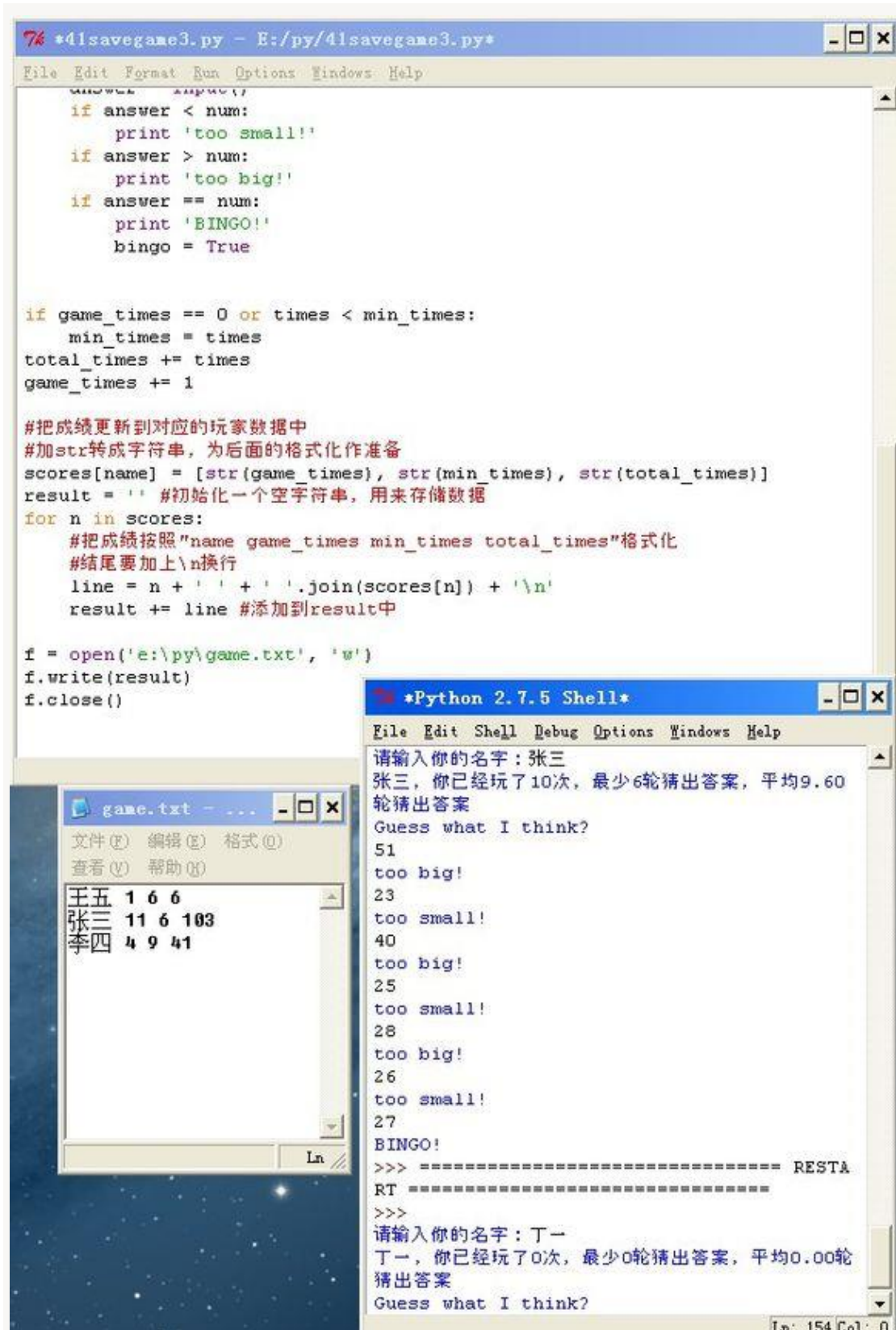
f = open('e:\py\game.txt')
lines = f.readlines()
f.close()

scores = {} #初始化一个空字典
for l in lines:
    s = l.split() #把每一行的数据拆分成list
    scores[s[0]] = s[1:] #第一项作为key, 剩下的作为value
score = scores.get(name) #查找当前玩家的数据
if score is None: #如果没找到,
    score = [0, 0, 0] #初始化数据

game_times = int(score[0])
min_times = int(score[1])
total_times = int(score[2])
if game_times > 0:
    avg_times = float(total_times) / game_times
else:
    avg_times = 0
#加上显示玩家的名字
print '%s, 你已经玩了%d次, 最少%d轮猜出答案, 平均%.2f轮猜出答案' % (
    name, game_times, min_times, avg_times)

num = randint(1, 100)
times = 0
#猜数游戏开始
```

Ln: 28 Col: 0



```
7% *41savegame3.py - E:/py/41savegame3.py*
File Edit Format Run Options Windows Help

if answer < num:
    print 'too small!'
if answer > num:
    print 'too big!'
if answer == num:
    print 'BINGO!'
    bingo = True

if game_times == 0 or times < min_times:
    min_times = times
total_times += times
game_times += 1

#把成绩更新到对应的玩家数据中
#加str转成字符串, 为后面的格式化作准备
scores[name] = [str(game_times), str(min_times), str(total_times)]
result = '' #初始化一个空字符串, 用来存储数据
for n in scores:
    #把成绩按照"name game_times min_times total_times"格式化
    #结尾要加上\n换行
    line = n + ' ' + ' '.join(scores[n]) + '\n'
    result += line #添加到result中

f = open('e:\py\game.txt', 'w')
f.write(result)
f.close()
```

```
*Python 2.7.5 Shell*
File Edit Shell Debug Options Windows Help

请输入你的名字: 张三
张三, 你已经玩了10次, 最少6轮猜出答案, 平均9.60
轮猜出答案
Guess what I think?
51
too big!
23
too small!
40
too big!
25
too small!
28
too big!
26
too small!
27
BINGO!
>>> ===== RESTA
RT =====
>>>
请输入你的名字: 丁一
丁一, 你已经玩了0次, 最少0轮猜出答案, 平均0.00轮
猜出答案
Guess what I think?
```

```
game.txt - ...
文件(F) 编辑(E) 格式(O)
查看(V) 帮助(H)

王五 1 6 6
张三 11 6 103
李四 4 9 41

Ln
```

如果你充分理解了程序, 恭喜你, 你对文件处理已经有了一个基本的了解。在日常工作学习中, 如果需要处理一些大量重复机械的文件操作, 比如整理格式、更改文件中的部分文字、统计数据等等, 都可以试着用 python 来解决。

【Python 第 42 课】函数的默认参数

之前我们用过函数，比如：

```
def hello(name):  
    print 'hello ' + name
```

然后我们去调用这个函数：

```
hello('world')
```

程序就会输出

```
hello world
```

如果很多时候，我们都是用 world 来调用这个函数，少数情况才会去改参数。那么，我们就可以给这个函数一个默认参数：

```
def hello(name = 'world'):  
    print 'hello ' + name
```

当你没有提供参数值时，这个参数就会使用默认值；如果你提供了，就用你给的。

这样，在默认情况下，你只要调用

```
hello()
```

就可以输出

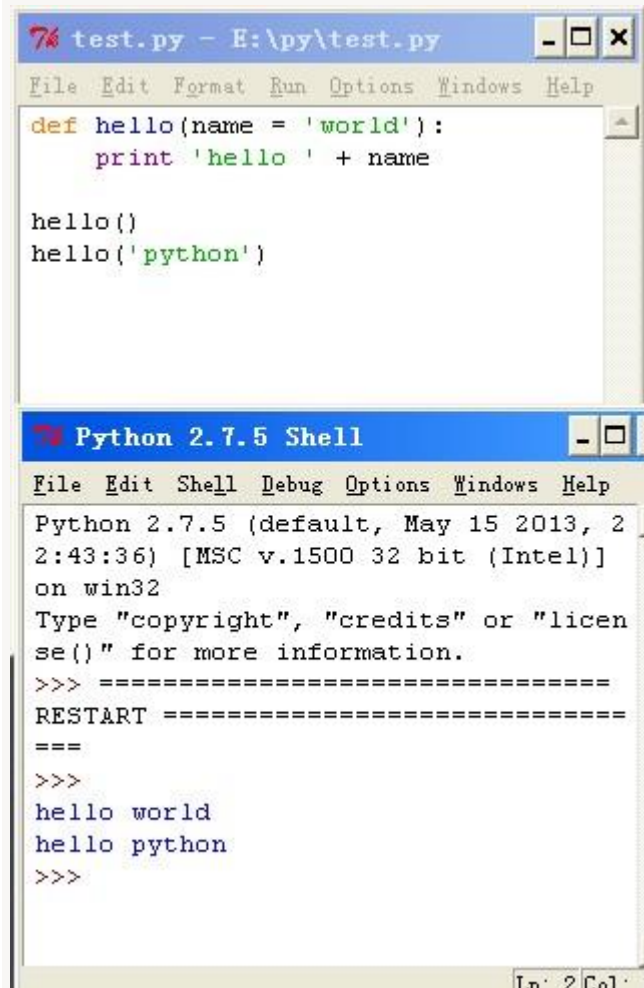
```
hello world
```

同样你也可以指定参数：

```
hello('python')
```


输出

hello python



The image shows a screenshot of a Python IDE. The top window, titled 'test.py - E:\py\test.py', contains the following code:

```
def hello(name = 'world'):  
    print 'hello ' + name  
  
hello()  
hello('python')
```

The bottom window, titled 'Python 2.7.5 Shell', shows the output of running the script:

```
Python 2.7.5 (default, May 15 2013, 2  
2:43:36) [MSC v.1500 32 bit (Intel)]  
on win32  
Type "copyright", "credits" or "licen  
se()" for more information.  
>>> =====  
RESTART =====  
===  
>>>  
hello world  
hello python  
>>>
```

注意，当函数有多个参数时，如果你想给部分参数提供默认参数，那么这些参数必须在参数的末尾。比如：

```
def func(a, b=5)
```

是正确的

```
def func(a=5, b)
```

就会出错

【Python 第 43 课】查天气 (1)

```
crossin@nono: ~/Desktop
crossin@nono:~/Desktop$ python weather.py
你想查哪个城市的天气？
南京
多云
26°C~ 36°C
crossin@nono:~/Desktop$ python weather.py
你想查哪个城市的天气？
上海
晴
28°C~ 38°C
crossin@nono:~/Desktop$ python weather.py
你想查哪个城市的天气？
北京
阴
23°C~ 31°C
crossin@nono:~/Desktop$ python weather.py
你想查哪个城市的天气？
█
```

你输入一个城市的名称，就会告诉你这个城市现在的天气情况。接下来的几节课，我就说一下怎么实现这样一个小程序。

之所以能知道一个城市的天气，是因为用了中国天气网 (www.weather.com.cn) 提供的天气查询接口。在浏览器里试着访问一下：

<http://www.weather.com.cn/data/cityinfo/101010100.html>

你就能看到北京现在的天气。这段看上去有点像 python 中字典类的文字是一种称作 json 格式的数据。

而我们的程序要做的事情，就是按照用户输入的城市名称，去天气网的接口请求对应的天气信息，再把结果展示给用户。

于是，在这个程序中，我们要用到两个新模块：

1. urllib2

用来发送网络请求，获取数据

2. json

用来解析获得的数据

听上去似乎还挺不算太复杂？但是注意刚才那个例子，我们请求北京天气时，用了“101010100”这样的数字。这是天气网设定的城市代码。然而令人蛋疼的是，天气网并没有直接给出所有城市代码的对应关系，而是给了 3 个接口：

1. <http://m.weather.com.cn/data5/city.xml>

获取所有省/直辖市的编号，如“01|北京,02|上海,03|天津”

2. <http://m.weather.com.cn/data5/city> 省编号.xml

获取二级地区编号，如江苏是：city19.xml

3. <http://m.weather.com.cn/data5/city> 二级编号.xml

获取三级编号，如南京是：city1901.xml

得到最终的三级编号之后，再加上中国 101 的前缀，就得到了城市代码，如南京市区就是“101190101”

所以，你可以选择，再写一个 python 程序，事先把这些复杂的编码全部抓取下来，整理成你要的格式；或者，偷懒一下，跳过这个过程，直接拿我抓好的编码。我把它放在了贴吧上。

今天先卖个关子，不说具体的写法。想挑战的同学可以试试再我说之前就把这个程序搞定。

【Python 第 44 课】查天气 (2)

先来看 python 中的 urllib2,这是 python 中一个用来获取网络资源的模块。我们平常上网，在浏览器地址栏中输入一个网址，浏览器根据这个网址拿到一些内容，然后展现在页面上，这大约就是浏览网页的过程。类似的，urllib2 会跟你提供的网址，请求对应的内容。

打开一个链接和打开一个文件有点像：

```
import urllib2
web = urllib2.urlopen('http://www.baidu.com')
content = web.read()
print content
```

我们引入 urllib2 的模块，用其中的 urlopen 方法打开百度，然后用 read 方法把其中的内容读取到一个变量中并输出。运行后，你会看到控制台中输出了一堆看不懂的代码文字。这段代码中有 html，有 css，还有 javascript。我们在浏览器中看到的网页大部分就是由这些代码所组成。如果你把 content 保存到一个以 “.html” 结尾的文件中（保存文件的方法前面已经说过很多），再打开这个 html 文件，就会看到“百度的首页”，只是这个首页在你的电脑上，所以你无法进行搜索。

打开一个链接和打开一个文件有点像：

回到我们的查天气程序，我们要向中国天气网发一个查询天气的请求。昨天说了，如何获取查询的 url 是个问题。先说简单的办法，用我提供的城市代码列表 city.py。

city.py 这个文件里有一个叫做 city 的字典，它里面的 key 是城市的名称，value 是对应的城市代码。不用把它 copy 到自己的程序中，只要放在和你的代码同一路径下，用

```
from city import city
```

就可以引入 city 这个字典。这里相当于用了一个自定义的模块，前一个“city”是模块名，也就是 py 文件的名称，后一个“city”是模块中变量的名称。

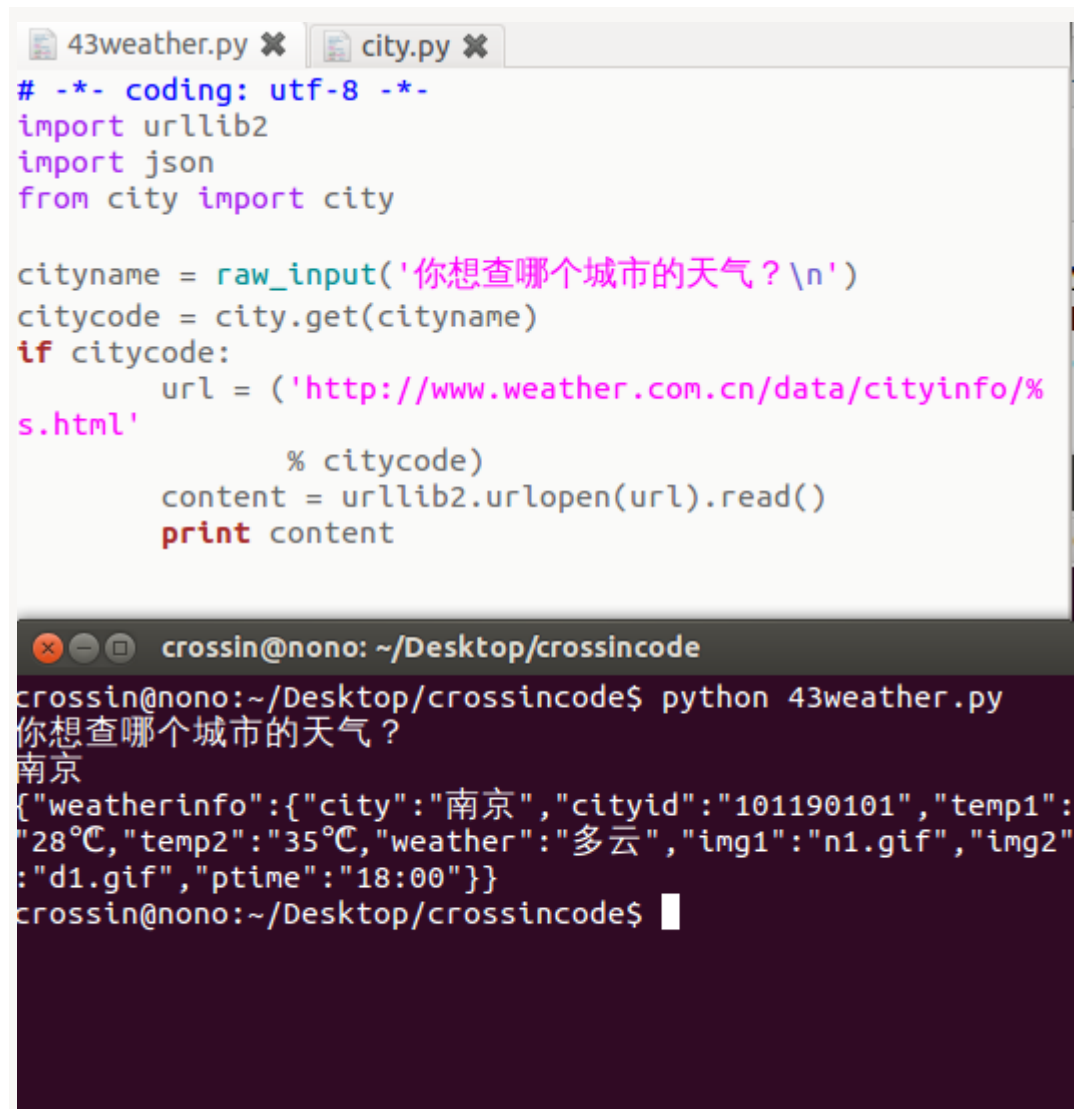
构造我们需要的 url：

```
cityname = raw_input('你想查哪个城市的天气? \n')
citycode = city.get(cityname)
if citycode:
    url = ('http://www.weather.com.cn/data/cityinfo/%s.html' % citycode)
    content = urllib2.urlopen(url).read()
```

为了防止你输入列表中没有的城市，所以用了 if 判断 citycode 是否存在。

运行一下看看能不能得到结果。如果提示编码的错误，试试在文件最开始加上：

```
# -*- coding: utf-8 -*-
```



The image shows a code editor with two tabs: '43weather.py' and 'city.py'. The '43weather.py' tab is active, displaying a Python script. The script imports 'urllib2' and 'json', and imports 'city' from a module named 'city'. It prompts the user for a city name, gets the city code from the 'city' module, and then constructs a URL to fetch weather data from 'http://www.weather.com.cn/data/cityinfo/%s.html'. The script prints the content of the URL. Below the code editor, a terminal window shows the execution of the script. The user enters '南京' (Nanjing), and the script outputs a JSON object containing weather information for Nanjing, including temperature, weather condition, and image links.

```
# -*- coding: utf-8 -*-
import urllib2
import json
from city import city

cityname = raw_input('你想查哪个城市的天气?\n')
citycode = city.get(cityname)
if citycode:
    url = ('http://www.weather.com.cn/data/cityinfo/%
s.html'
          % citycode)
    content = urllib2.urlopen(url).read()
    print content
```

```
crossin@nono: ~/Desktop/crossincode
crossin@nono:~/Desktop/crossincode$ python 43weather.py
你想查哪个城市的天气?
南京
{"weatherinfo":{"city":"南京","cityid":"101190101","temp1":
"28°C","temp2":"35°C","weather":"多云","img1":"n1.gif","img2"
:"d1.gif","ptime":"18:00"}}
crossin@nono:~/Desktop/crossincode$
```

可以看到，已经拿到了 json 格式的天气信息。下一课再来处理它。

【Python 第 45 课】查天气 (3)

看一下我们已经拿到的 json 格式的天气数据：

```
1. {  
2.   "weatherinfo": {  
3.     "city": "南京",  
4.     "cityid": "101190101",  
5.     "templ": "37°C",  
6.     "temp2": "28°C",  
7.     "weather": "多云",  
8.     "img1": "d1.gif",  
9.     "img2": "n1.gif",  
10.    "ptime": "11:00"  
11.  }  
12. }
```

复制代码

直接在命令行中看到的应该还是没有换行和空格的一长串字符，这里我把格式整理了一下。可以看出，它像是一个字典的结构，但是有两层。最外层只有一个 key——“weatherinfo”，它的 value 是另一个字典，里面包含了好几项天气信息，现在我们最关心的就是其中的 temp1, temp2 和 weather。

虽然看上去像字典，但它对于程序来说，仍然是一个字符串，只不过是一个满足 json 格式的字符串。我们用 python 中提供的另一个模块 json 提供的 loads 方法，把它转成一个真正的字典。

```
1. import json  
2.  
3. data = json.loads(content)
```

复制代码

这时候的 data 已经是一个字典，尽管在控制台中输出它，看上去和 content 没什么区别，只是编码上有些不同：

```
1. {u'weatherinfo': {u'city': u'\u5357\u4eac', u'ptime': u'11:00',  
   u'cityid': u'101190101', u'temp2': u'28\u2103', u'templ': u'37\u2103',  
   u'weather': u'\u591a\u4e91', u'img2': u'n1.gif', u'img1': u'd1.gif'}}
```

复制代码

但如果你用 type 方法看一下它们的类型：

```
1. print type(content)
```

```
2. print type(data)
3.
```

复制代码

就知道区别在哪里了。

之后的事情就比较容易了。

```
1. result = data['weatherinfo']
2. str_temp = ('%s\n%s ~ %s') % (
3. result['weather'],
4. result['temp1'],
5. result['temp2'])
6. )
7. print str_temp
```

复制代码

为了防止在请求过程中出错，我加上了一个异常处理。

```
1. try:
2.     ###
3.     ###
4. except:
5.     print '查询失败'
```

复制代码

以及没有找到城市时的处理：

```
1. if citycode:
2.     ###
3.     ###
4. else:
5.     print '没有找到该城市'
```

复制代码

完整代码：

```
1. # -*- coding: utf-8 -*-
2. import urllib2
3. import json
4. from city import city
5.
6. cityname = raw_input('你想查哪个城市的天气? \n')
```

```

7. citycode = city.get(cityname)
8. if citycode:
9.     try:
10. url = ('http://www.weather.com.cn/data/cityinfo/%s.html'
11. % citycode)
12. content = urllib2.urlopen(url).read()
13. data = json.loads(content)
14. result = data['weatherinfo']
15. str_temp = ('%s\n%s ~ %s') % (
16. result['weather'],
17. result['temp1'],
18. result['temp2']
19. )
20. print str_temp
21. except:
22. print '查询失败'
23. else:
24. print '没有找到该城市'

```

【Python 第 46 课】查天气（4）

明天俺就要出发了，今天赶在睡觉前来个深夜档。

这一课算是“查天气”程序的附加内容。没有这一课，你也查到天气了。但了解一下城市代码的抓取过程，会对网页抓取有更深入的理解。

天气网的城市代码信息结构比较复杂，所有代码按层级放在了很多 xml 为后缀的文件中。而这些所谓的“xml”文件又不符合 xml 的格式规范，导致在浏览器中无法显示，给我们的抓取又多加了一点难度。

首先，抓取省份的列表：

```

1. url1 = 'http://m.weather.com.cn/data5/city.xml'
2. content1 = urllib2.urlopen(url1).read()
3. provinces = content1.split(',')

```

输出 content1 可以查看全部省份代码：

```

1. 01|北京,02|上海,03|天津,...

```

对于每个省，抓取城市列表：

```
1. url = 'http://m.weather.com.cn/data3/city%s.xml'
2. for p in provinces:
3.     p_code = p.split('|')[0]
4.     url2 = url % p_code
5.     content2 = urllib2.urlopen(url2).read()
6.     cities = content2.split(',')
```

输出 content2 可以查看此省份下所有城市代码：

```
1. 1901|南京,1902|无锡,1903|镇江,...
```

再对于每个城市，抓取地区列表：

```
1. for c in cities[:3]:
2.     c_code = c.split('|')[0]
3.     url3 = url % c_code
4.     content3 = urllib2.urlopen(url3).read()
5.     districts = content3.split(',')
```

content3 是此城市下所有地区代码：

```
1. 190101|南京,190102|溧水,190103|高淳,...
```

最后，对于每个地区，我们把它的名字记录下来，然后再发送一次请求，得到它的最终代码：

```
1. for d in districts:
2.     d_pair = d.split('|')
3.     d_code = d_pair[0]
4.     name = d_pair[1]
5.     url4 = url % d_code
6.     content4 = urllib2.urlopen(url4).read()
7.     code = content4.split('|')[1]
```

name 和 code 就是我们最终要得到的城市代码信息。它们格式化到字符串中，最终保存在文件里：

```
1. line = "'%s': '%s',\n" % (name, code)
2. result += line
```

同时你也可以输出它们，以便在抓取的过程中查看进度：

```
1. print name + ':' + code
```

完整代码：


```
1. import urllib2
2.
3. url1 = 'http://m.weather.com.cn/data5/city.xml'
4. content1 = urllib2.urlopen(url1).read()
5. provinces = content1.split(',')
6. result = 'city = {\n'
7. url = 'http://m.weather.com.cn/data3/city%s.xml'
8. for p in provinces:
9.     p_code = p.split('|')[0]
10. url2 = url % p_code
11. content2 = urllib2.urlopen(url2).read()
12. cities = content2.split(',')
13. for c in cities:
14.     c_code = c.split('|')[0]
15. url3 = url % c_code
16. content3 = urllib2.urlopen(url3).read()
17. districts = content3.split(',')
18. for d in districts:
19.     d_pair = d.split('|')
20.     d_code = d_pair[0]
21.     name = d_pair[1]
22. url4 = url % d_code
23. content4 = urllib2.urlopen(url4).read()
24. code = content4.split('|')[1]
25. line = " '%s': '%s',\n" % (name, code)
26. result += line
27. print name + ':' + code
28. result += '}'
29. f = file('/home/crossin/Desktop/city.py', 'w')
30. f.write(result)
31. f.close()
```

如果你只是想抓几个测试一下，并不用全部抓下来，在 provinces 后面加上[:3]，抓 3 个省的试试看就好了。

【Python 第 47 课】面向对象（1）

我们之前已经写了不少小程序，都是按照功能需求的顺序来设计程序。这种被称为“面向过程”的编程。

还有一种程序设计的方法，把数据和对数据的操作用一种叫做“对象”的东西包裹起来。这种被成为“面向对象”的编程。这种方法更适合较大型的程序开发。

面向对象编程最主要的两个概念就是：**类**（class）和**对象**（object）

类是一种抽象的类型，而对象是这种类型的实例。

举个现实的例子：“笔”作为一个抽象的概念，可以被看成是一个类。而一支实实在在的笔，则是“笔”这种类型的对象。

一个类可以有属于它的函数，这种函数被称为类的“方法”。一个类/对象可以有属于它的变量，这种变量被称作“域”。域根据所属不同，又分别被称作“类变量”和“实例变量”。

继续笔的例子。一个笔有书写的功能，所以“书写”就是笔这个类的一种方法。每支笔有自己的颜色，“颜色”就是某支笔的域，也是这支笔的实例变量。

而关于“类变量”，我们假设有一种限量版钢笔，我们为这种笔创建一种类。而这种笔的“产量”就可以看做这种笔的类变量。因为这个域不属于某一支笔，而是这种类型的笔的共有属性。

域和方法被合称为类的属性。

python 是一种高度面向对象的语言，它其中的所有东西其实都是对象。所以我们之前也一直在使用着对象。看如下的例子：

```
1. s = 'how are you'
2. #s 被赋值后就是一个字符串类型的对象
3. l = s.split()
4. #split 是字符串的方法，这个方法返回一个 list 类型的对象
5. #l 是一个 list 类型的对象
```

通过 dir() 方法可以查看一个类/变量的所有属性：

```
1. dir(s)
2. dir(list)
```

【Python 第 48 课】 面向对象（2）

昨天介绍了面向对象的概念，今天我们来创建一个类。

```
1. class MyClass:
2.     pass
3.
4. mc = MyClass()
5. print mc
```

关键字 `class` 加上类名用来创建一个类。之后缩进的代码块是这个类的内部。在这里，我们用 `pass` 语句，表示一个空的代码块。

类名加圆括号 `()` 的形式可以创建一个类的实例，也就是被称作对象的东西。我们把这个对象赋值给变量 `mc`。于是，`mc` 现在就是一个 `MyClass` 类的对象。

看一下输出结果：

```
1. <__main__.MyClass instance at 0x7fd1c8d01200>
2.
```

这个意思就是说，`mc` 是 `__main__` 模块中 `MyClass` 来的一个实例（instance），后面的一串十六进制的数字是这个对象的内存地址。

我们给这个类加上一些域：

```
1. class MyClass:
2.     name = 'Sam'
3.
4.     def sayHi(self):
5.         print 'Hello %s' % self.name
6.
7. mc = MyClass()
8. print mc.name
9. mc.name = 'Lily'
10. mc.sayHi()
```

我们给 `MyClass` 类增加了一个类变量 `name`，并把它值设为 `'Sam'`。然后又增加了一个类方法 `sayHi`。

调用类变量的方法是“对象.变量名”。你可以得到它的值，也可以改变它的值。

注意到，类方法和我们之前定义的函数区别在于，第一个参数必须为 `self`。而在调用类方法的时候，通过“对象.方法名`()`”格式进行调用，而不需要额外提供 `self` 这个参数的值。`self` 在类方法中的值，就是你调用的这个对象本身。

输出结果：

```
1. Sam
2. Hello Lily
```

之后，在你需要用到 MyClass 这种类型对象的地方，就可以创建并使用它。

【Python 第 49 课】面向对象（3）

面向对象是比较复杂的概念，初学很难理解。我曾经对人夸张地说，面向对象是颠覆你编程三观的东西，得花上不少时间才能搞清楚。我自己当年初学 Java 的时候，也是折腾了很久才理清头绪。所以我在前面的课程中没有去提及类和对象这些概念，不想在一开始给大家造成混淆。

在刚开始编程的时候，从上到下一行行执行的简单程序容易被理解，即使加上 if、while、for 之类的语句以及函数调用，也还是不算困难。有了面向对象之后，程序的执行路径就变得复杂，很容易让人混乱。不过当你熟悉之后会发现，面向对象是比面向过程更合理的程序设计方式。

今天我用一个例子来展示两种程序设计方式的不同。

假设我们有一辆汽车，我们知道它的速度(60km/h)，以及 A、B 两地的距离(100km)。要算出开着这辆车从 A 地到 B 地花费的时间。（很像小学数学题是吧？）

面向过程的方法：

```
1. speed = 60.0
2. distance = 100.0
3. time = distance / speed
4. print time
5.
```

面向对象的方法：

```
1. class Car:
2.     speed = 0
3.     def drive(self, distance):
4.         time = distance / self.speed
5.         print time
6.
7. car = Car()
8. car.speed = 60.0
9. car.drive(100.0)
```

看上去似乎面向对象没有比面向过程更简单，反而写了更多行代码。

但是，如果我们让题目再稍稍复杂一点。假设我们又有了一辆更好的跑车，它的速度是

150km/h，然后我们除了想从 A 到 B，还要从 B 到 C（距离 200km）。要求分别知道这两种车在这两段路上需要多少时间。

面向过程的方法：

```
1. speed1 = 60.0
2. distance1 = 100.0
3. time1 = distance1 / speed1
4. print time1
5.
6. distance2 = 200.0
7. time2 = distance2 / speed1
8. print time2
9.
10. speed2 = 150.0
11. time3 = distance1 / speed2
12. print time3
13.
14. time4 = distance2 / speed2
15. print time4
```

面向对象的方法：

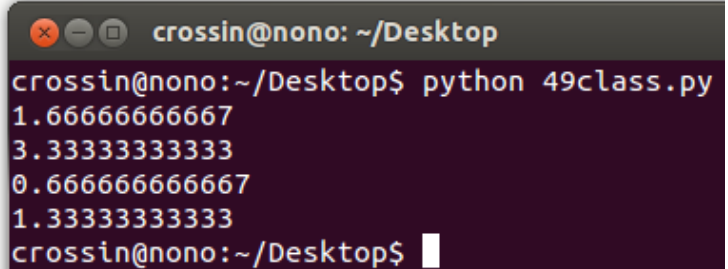
```
1. class Car:
2.     speed = 0
3.     def drive(self, distance):
4.         time = distance / self.speed
5.         print time
6.
7. car1 = Car()
8. car1.speed = 60.0
9. car1.drive(100.0)
10. car1.drive(200.0)
11.
12. car2 = Car()
13. car2.speed = 150.0
14. car2.drive(100.0)
15. car2.drive(200.0)
16.
```

对比两种方法，面向过程把数据和处理数据的计算全部放在一起，当功能复杂之后，就会显得很混乱，且容易产生很多重复的代码。而面向对象，把一类数据和处理这类数据的方法封装在一个类中，让程序的结构更清晰，不同的功能之间相互独立。这样更有利于进行模块化的开发方式。

```
class Car:
    speed = 0
    def drive(self, distance):
        time = distance / self.speed
        print time

car1 = Car()
car1.speed = 60.0
car1.drive(100.0)
car1.drive(200.0)

car2 = Car()
car2.speed = 150.0
car2.drive(100.0)
car2.drive(200.0)
```



```
crossin@nono: ~/Desktop
crossin@nono:~/Desktop$ python 49class.py
1.66666666667
3.33333333333
0.66666666667
1.33333333333
crossin@nono:~/Desktop$
```

面向对象的水还很深，我们这里只是粗略一瞥。它不再像之前 print、while 这些概念那么一目了然。但也没必要对此畏惧，等用多了自然就熟悉了。找一些实例亲手练练，会掌握得更快。遇到问题时，欢迎来论坛和群里讨论。

【Python 第 50 课】面向对象（4）

上一课举了一个面向对象和面向过程相比较的例子之后，有些同学表示，仍然没太看出面向对象的优点。没关系，那是因为我们现在接触的程序还不够复杂，等以后你写的程序越来越大，就能体会到这其中的差别了。

今天我们就来举一个稍稍再复杂一点点的例子。

仍然是从 A 地到 B 地，这次除了有汽车，我们还有了一辆自行车！

自行车和汽车有着相同的属性：速度（speed）。还有一个相同的方法（drive），来输出行驶/骑行一段距离所花的时间。但这次我们要给汽车增加一个属性：每公里油耗（fuel）。而在汽车行驶一段距离的方法中，除了要输出所花的时间外，还要输出所需的油量。

面向过程的方法，你可能需要写两个函数，然后把数据作为参数传递进去，在调用的时候要搞清楚应该使用哪个函数和哪些数据。有了面向对象，你可以把相关的数据和方法封装在一起，并且可以把不同类中的相同功能整合起来。这就需要用到面向对象中的另一个重要概念：继

承。

我们要使用的方法是，创建一个叫做 Vehicle 的类，表示某种车，它包含了汽车和自行车所共有的东西：速度，行驶的方法。然后让 Car 类和 Bike 类都继承这个 Vehicle 类，即作为它的子类。在每个子类中，可以分别添加各自独有的属性。

Vehicle 类被称为基本类或超类，Car 类和 Bike 类被成为导出类或子类。

```
1. class Vehicle:
2.     def __init__(self, speed):
3.         self.speed = speed
4.
5.     def drive(self, distance):
6.         print 'need %f hour(s)' % (distance / self.speed)
7.
8.     class Bike(Vehicle):
9.         pass
10.
11.     class Car(Vehicle):
12.         def __init__(self, speed, fuel):
13.             Vehicle.__init__(self, speed)
14.             self.fuel = fuel
15.
16.         def drive(self, distance):
17.             Vehicle.drive(self, distance)
18.             print 'need %f fuels' % (distance * self.fuel)
19.
20. b = Bike(15.0)
21. c = Car(80.0, 0.012)
22. b.drive(100.0)
23. c.drive(100.0)
24.
```

解释一下代码：

`__init__` 函数会在类被创建的时候自动调用，用来初始化类。它的参数，要在创建类的时候提供。于是我们通过提供一个数值来初始化 speed 的值。

class 定义后面的括号里表示这个类继承于哪个类。Bike(Vehicle) 就是说 Bike 是继承自 Vehicle 中的子类。Vehicle 中的属性和方法，Bike 都会有。因为 Bike 不需要有额外的功能，所以用 pass 在类中保留空块，什么都不用写。

Car 类中，我们又重新定义了 `__init__` 和 `drive` 函数，这样会覆盖掉它继承自 Vehicle 的同名函数。但我们依然可以通过“Vehicle.函数名”来调用它的超类方法。以此来获得它作为 Vehicle 所具有的功能。注意，因为是通过类名调用方法，而不是像之前一样通过对象来调用，所以这里必须提供 self 的参数值。在调用超类的方法之后，我们又给 Car 增加了一个

fuel 属性，并且在 drive 中多输出一行信息。

最后，我们分别创建一个速度为 15 的自行车对象，和一个速度为 80、耗油量为 0.012 的汽车，然后让它们去行驶 100 的距离。

```
50class.py ✕
class Vehicle:
    def __init__(self, speed):
        self.speed = speed

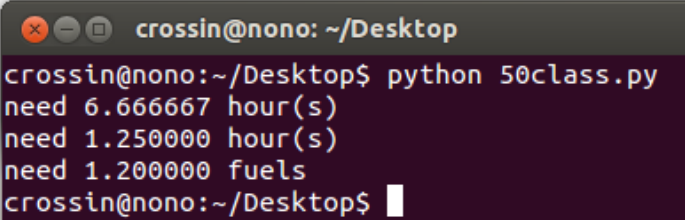
    def drive(self, distance):
        print 'need %f hour(s)' % (distance / self.speed)

class Bike(Vehicle):
    pass

class Car(Vehicle):
    def __init__(self, speed, fuel):
        Vehicle.__init__(self, speed)
        self.fuel = fuel

    def drive(self, distance):
        Vehicle.drive(self, distance)
        print 'need %f fuels' % (distance * self.fuel)

b = Bike(15.0)
c = Car(80.0, 0.012)
b.drive(100.0)
c.drive(100.0)
```



```
crossin@nono: ~/Desktop
crossin@nono:~/Desktop$ python 50class.py
need 6.666667 hour(s)
need 1.250000 hour(s)
need 1.200000 fuels
crossin@nono:~/Desktop$
```


【Python 第 51 课】 and-or 技巧

今天介绍一个 python 中的小技巧：and-or

看下面这段代码：

```
1. a = "heaven"
2. b = "hell"
3. c = True and a or b
4. print c
5. d = False and a or b
6. print d
```

输出：

heaven

hell

结果很奇怪是不是？

表达式从左往右运算，1 和"heaven"做 and 的结果是"heaven"，再与"hell"做 or 的结果是"heaven"；0 和"heaven"做 and 的结果是 0，再与"hell"做 or 的结果是"hell"。

抛开绕人的 and 和 or 的逻辑，你只需记住，在一个 bool and a or b 语句中，当 bool 条件为真时，结果是 a；当 bool 条件为假时，结果是 b。

有学过 c/c++的同学应该会发现，这和 bool?a:b 表达式很像。

有了它，原本需要一个 if-else 语句表述的逻辑：

```
1. if a > 0:
2.     print "big"
3. else:
4.     print "small"
```

就可以直接写成：

```
1. print (a > 0) and "big" or "small"
2.
```

然而不幸的是，如果直接这么用，有一天你会踩到坑的。和 c 语言中的?:表达式不同，这里的 and or 语句是利用了 python 中的逻辑运算实现的。当 a 本身是个假值（如 0，""）时，结果就不会像你期望的那样。

比如：

```
1. a = ""
2. b = "hell"
3. c = True and a or b
4. print c
```

得到的结果就是“hell”。因为“”和“hell”做 and 的结果是“hell”。

所以，and-or 真正的技巧在于，确保 a 的值不会为假。最常用的方式是使 a 成为 [a] 、 b 成为 [b] ， 然后使用返回值列表的第一个元素：

```
1. a = ""
2. b = "hell"
3. c = (True and [a] or [b])[0]
4. print c
```

由于[a]是一个非空列表，所以它决不会为假。即使 a 是 0 或者'' 或者其它假值，列表[a] 也为真，因为它有一个元素。

在两个常量值进行选择时，and-or 会让你的代码更简单。但如果你觉得这个技巧带来的副作用已经让你头大了，没关系，用 if-else 可以做相同的事情。不过在 python 的某些情况下，你可能没法使用 if 语句，比如 lambda 函数中，这时候你可能就需要 and-or 的帮助了。

什么是 lambda 函数？呵呵，这是 python 的高阶玩法，暂且按住不表，以后有机会再说。

【Python 第 52 课】 元组

上一次 pygame 的课中有这样一行代码：

```
1. x, y = pygame.mouse.get_pos()
```

复制代码

这个函数返回的其实是一个“元组”，今天我们来讲讲这个东西。

元组 (tuple) 也是一种序列，和我们用了很多次的 list 类似，只是元组中的元素在创建之后就不能被修改。

如：

```
1. postion = (1, 2)
2. geeks = ('Sheldon', 'Leonard', 'Rajesh', 'Howard')
```

复制代码

都是元组的实例。它有和 list 同样的索引、切片、遍历等操作（参见 25~27 课）：

```
1. print postion[0]
2. for g in geeks:
3.     print g
4. print geeks[1:3]
```

复制代码

其实我们之前一直在用元组，就是在 print 语句中：

```
1. print '%s is %d years old' % ('Mike', 23)
```

('Mike', 23) 就是一个元组。这是元组最常见的用处。

再来看一下元组作为函数返回值的例子：

```
1. def get_pos(n):
2.     return (n/2, n*2)
```

得到这个函数的返回值有两种形式，一种是根据返回值元组中元素的个数提供变量：

```
1. x, y = get_pos(50)
2. print x
3. print y
```

这就是我们在开头那句代码中使用的方式。

还有一种方法是用一个变量记录返回的元组：

```
1. pos = get_pos(50)
2. print pos[0]
3. print pos[1]
```

【Python 第 53 课】 数学运算

今天从打飞机游戏里中断一下，说些 python 的基础。

在用计算机编程解决问题的过程中，数学运算是很常用的。python 自带了一些基本的数学运算方法，这节课给大家介绍一二。

python 的数学运算模块叫做 math，再用之前，你需要
import math

math 包里有两个常量：
math.pi

圆周率 π : 3.141592...

`math.e`

自然常数: 2.718281...

数值运算:

`math.ceil(x)`

对 x 向上取整, 比如 $x=1.2$, 返回 2

`math.floor(x)`

对 x 向下取整, 比如 $x=1.2$, 返回 1

`math.pow(x,y)`

指数运算, 得到 x 的 y 次方

`math.log(x)`

对数, 默认基底为 e 。可以使用第二个参数, 来改变对数的基底。比如 `math.log(100, 10)`

`math.sqrt(x)`

平方根

`math.fabs(x)`

绝对值

三角函数:

`math.sin(x)`

`math.cos(x)`

`math.tan(x)`

`math.asin(x)`

`math.acos(x)`

`math.atan(x)`

注意: 这里的 x 是以弧度为单位, 所以计算角度的话, 需要先换算

角度和弧度互换:

`math.degrees(x)`

弧度转角度

`math.radians(x)`

角度转弧度

以上是你平常可能会用到的函数。除此之外, 还有一些, 这里就不罗列, 可以去

<http://docs.python.org/2/library/math.html>

查看官方的完整文档。

有了这些函数，可以更方便的实现程序中的计算。比如中学时代算了无数次的

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

现在你就可以写一个函数，输入一元二次方程的 a、b、c 系数，直接给你数值解。好，这题就留作课后作业吧。

以后我还会不定期地介绍 python 中的模块，例如 random（随机数）、re（正则表达式）、time（时间）、urllib2（网络请求）等等。

【Python 第 54 课】真值表

逻辑判断是编程中极为常用的知识。之前的课我们已经说过，见第 6 课和第 11 课。但鉴于逻辑运算的重要性，今天我再把常用的运算结果总结一下，供大家参考。

这种被称为“真值表”的东西，罗列了基本逻辑运算的结果。你不一定要全背下来，但应该对运算的规律有所了解。

为了便于看清，我用 \Leftrightarrow 来表示等价关系。

\Leftarrow 左边表示逻辑表达式， \Rightarrow 右边表示它的结果。

NOT

not False \Leftrightarrow True

not True \Leftrightarrow False

（not 的结果与原值相反）

OR

True or False \Leftrightarrow True

True or True \Leftrightarrow True

False or True \Leftrightarrow True

False or False \Leftrightarrow False

（只要有一个值为 True，or 的结果就是 True）

AND

True and False \Leftrightarrow False

True and True \Leftrightarrow True

False and True \Leftrightarrow False

False and False \Leftrightarrow False

（只要有一个值为 False，and 的结果就是 False）

NOT OR

not (True or False) \Leftrightarrow False

not (True or True) \Leftrightarrow False

not (False or True) \Leftrightarrow False

not (False or False) \Leftrightarrow True

NOT AND

```
not (True and False) <=> True
not (True and True) <=> False
not (False and True) <=> True
not (False and False) <=> True
```

```
!=
```

```
1 != 0 <=> True
1 != 1 <=> False
0 != 1 <=> True
0 != 0 <=> False
```

```
==
```

```
1 == 0 <=> False
1 == 1 <=> True
0 == 1 <=> False
0 == 0 <=> True
```

以上就是基本的逻辑运算，你会在编程中反复用到它们。就算刚开始搞不清也没关系，多写几段代码就会熟悉了。

【Python 第 55 课】 正则表达式 (1)

今天来挖个新坑，讲讲正则表达式。

什么是正则表达式？在回答这个问题之前，先来看看为什么要有正则表达式。

在编程处理文本的过程中，经常会需要按照某种规则去查找一些特定的字符串。比如知道一个网页上的图片都是叫做 'image/8554278135.jpg' 之类的名字，只是那串数字不一样；又或者在一堆人员电子档案中，你要把他们的电话号码全部找出来，整理成通讯录。诸如此类工作，如果手工去做，当量大的时候那简直就是悲剧。但你知道这些字符信息有一定的规律，可不可以利用这些规律，让程序自动来做这些无聊的事情？答案是肯定的。这时候，你就需要一种描述这些规律的方法，正则表达式就是干这事的。

正则表达式就是记录文本规则的代码。

所以正则表达式并不是 python 中特有的功能，它是一种通用的方法。python 中的正则表达式库，所做的事情是利用正则表达式来搜索文本。要使用它，你必须会自己用正则表达式来描述文本规则。之前多次有同学表示查找文本的事情经常会遇上，希望能介绍一下正则表达式。既然如此，我们就从正则表达式的基本规则开始说起。

1.

首先说一种最简单的正则表达式，它没有特殊的符号，只有基本的字母或数字。它满足的匹配规则就是完全匹配。例如：有个正则表达式是“hi”，那么它就可以匹配出文本中所有含有 hi 的字符。

来看如下的一段文字：

Hi, I am Shirley Hilton. I am his wife.

如果我们用“hi”这个正则表达式去匹配这段文字，将会得到两个结果。因为是完全匹配，所以每个结果都是“hi”。这两个“hi”分别来自“Shirley”和“his”。默认情况下正则表达式是严格区分大小写的，所以“Hi”和“Hilton”中的“Hi”被忽略了。

为了验证正则表达式匹配的结果，你可以用以下这段代码做实验：

```
1. import re
2. text = "Hi, I am Shirley Hilton. I am his wife."
3. m = re.findall(r"hi", text)
4. if m:
5.     print m
6. else:
7.     print 'not match'
```

暂时先不解释这其中代码的具体含义，你只要去更改 text 和 findall 中的字符串，就可以用它来检测正则表达式的实际效果。

2.

如果我们只想找到“hi”这个单词，而不把包含它的单词也算在内，那就可以使用“\bhi\b”这个正则表达式。在以前的字符串处理中，我们已经见过类似“\n”这种特殊字符。在正则表达式中，这种字符更多，以后足以让你眼花缭乱。

“\b”在正则表达式中表示单词的开头或结尾，空格、标点、换行都算是单词的分割。而“\b”自身又不会匹配任何字符，它代表的只是一个位置。所以单词前后的空格标点之类不会出现在结果里。

在前面那个例子里，“\bhi\b”匹配不到任何结果。但“\bhi”的话就可以匹配到1个“hi”，出自“his”。用这种方法，你可以找出一段话中所有单词“Hi”，想一下要怎么写。

3.

最后再说一下[]这个符号。在正则表达式中，[]表示满足括号中任一字符。比如“[hi]”，它就不是匹配“hi”了，而是匹配“h”或者“i”。

在前面例子中，如果把正则表达式改为 “[Hh]i”，就可以既匹配 “Hi”，又匹配 “hi” 了。

【Python 第 56 课】正则表达式（2）

有同学问起昨天那段测试代码里的问题，我来简单说一下。

1.

```
r"hi"
```

这里字符串前面加了 `r`，是 `raw` 的意思，它表示对字符串不进行转义。为什么要加这个？你可以试试 `print "\bhi"` 和 `r"\bhi"` 的区别。

```
>>> print "\bhi"
```

```
hi
```

```
>>> print r"\bhi"
```

```
\bhi
```

可以看到，不加 `r` 的话，`\b` 就没有了。因为 `python` 的字符串碰到“\”就会转义它后面的字符。如果你想在字符串里打“\”，则必须要打“\\”。

```
>>> print "\\bhi"
```

```
\bhi
```

这样的话，我们的正则表达式里就会多出很多“\”，让本来就已经复杂的字符串混乱得像五仁月饼一般。但加上了“`r`”，就表示不要去转义字符串中的任何字符，保持它的原样。

2.

```
re.findall(r"hi", text)
```

`re` 是 `python` 里的正则表达式模块。`findall` 是其中一个方法，用来按照提供的正则表达式，去匹配文本中的所有符合条件的字符串。返回结果是一个包含所有匹配的 `list`。

3.

今天主要说两个符号“`.`”和“`*`”，顺带说下“`\s`”和“`?`”。

“`.`”在正则表达式中表示除换行符以外的任意字符。在上节课提供的那段例子文本中：

```
Hi, I am Shirley Hilton. I am his wife.
```

如果我们用“`i.`”去匹配，就会得到

```
['i,', 'ir', 'il', 'is', 'if']
```

你若是暴力一点，也可以直接用“`.`”去匹配，看看会得到什么。

与“`.`”类似的一个符号是“`\s`”，它表示的是不是空白符的任意字符。注意是大写字符 `S`。

4.

在很多搜索中，会用“?”表示任意一个字符，“*”表示任意数量连续字符，这种被称为通配符。但在正则表达式中，任意字符是用“.”表示，而“*”则表示数量：它表示前面的字符可以重复任意多次（包括 0 次），只要满足这样的条件，都会被表达式匹配上。

结合前面的“.*”，用“l.*e”去匹配，想一下会得到什么结果？

`['I am Shirley Hilton. I am his wife']`

是不是跟你想的有些不一样？也许你会以为是

`['I am Shirle', 'I am his wife']`

这是因为“*”在匹配时，会匹配尽可能长的结果。如果你想让他匹配到最短的就停止，需要用“.*?”。如“l.*?e”，就会得到第二种结果。这种匹配方式被称为懒惰匹配，而原本尽可能长的方式被称为贪婪匹配。

最后留一道习题：

从下面一段文本中，匹配出所有 s 开头，e 结尾的单词。

site sea sue sweet see case sse ssee loses

【Python 第 57 课】正则表达式（3）

先来公布上一课习题的答案：

`\bs\S*?e\b`

有的同学给出的答案是“`\bs.*?e\b`”。测试一下就会发现，有奇怪的'sea sue'和'sweet see'混进来了。既然是单词，我们就不要空格，所以需要“`\S`”而不是“`.`”。

昨天有位同学在论坛上说，用正则表达式匹配出了文件中的手机号。这样现学现用很不错。匹配的规则是“`1.*?\n`”，在这个文件的条件下，是可行的。但这规则不够严格，且依赖于手机号结尾有换行符。今天我来讲讲其他的方法。

匹配手机号，其实就是找出一串连续的数字。更进一步，是 11 位，以 1 开头的数字。

还记得正则第 1 讲里提到的[]符号吗？它表示其中任意一个字符。所以要匹配数字，我们可以用

`[0123456789]`

由于它们是连续的字符，有一种简化的写法：`[0-9]`。类似的还有`[a-zA-Z]`的用法。

还有另一种表示数字的方法：

`\d`

要表示任意长度的数字，就可以用

`[0-9]*`

或者

`\d*`

但要注意的是，*表示的任意长度包括 0，也就是没有数字的空字符也会被匹配出来。一个与*类似的符号+，表示的则是 1 个或更长。

所以要匹配出所有的数字串，应当用

`[0-9]+`

或者

`\d+`

如果要限定长度，就用`{}`代替`+`，大括号里写上你想要的长度。比如 11 位的数字：

`\d{11}`

想要再把第一位限定为 1，就在前面加上 1，后面去掉一位：

`1\d{10}`

OK. 总结一下今天提到的符号：

`[0-9]`

`\d`

`+`

`{}`

现在你可以去一个混杂着各种数据的文件里，抓出里面的手机号，或是其他你感兴趣的数字了。

【Python 第 58 课】 正则表达式 (4)

1.

我们已经了解了正则表达式中的一些特殊符号，如**\b**、**\d**、**.**、**\S** 等等。这些具有特殊意义的专用字符被称作“元字符”。常用的元字符还有：

\w - 匹配字母或数字或下划线或汉字（我试验下了，发现 3.x 版本可以匹配汉字，但 2.x 版本不可以）

\s - 匹配任意的空白符

^ - 匹配字符串的开始

\$ - 匹配字符串的结束

2.

\S 其实就是**\s**的反义，任意不是空白符的字符。同理，还有：

\W - 匹配任意不是字母，数字，下划线，汉字的字符

\D - 匹配任意非数字的字符

\B - 匹配不是单词开头或结束的位置

[a]的反义是**[^a]**，表示除 a 以外的任意字符。**[^abcd]**就是除 abcd 以外的任意字符。

3.

之前我们用过*****、**+**、**{}**来表示字符的重复。其他重复的方式还有：

? - 重复零次或一次

{n,} - 重复 n 次或更多次

{n,m} - 重复 n 到 m 次

正则表达式不只是用来从一大段文字中抓取信息，很多时候也被用来判断输入的文本是否符合规范，或进行分类。来点例子看看：

^\w{4,12}\$

这个表示一段 4 到 12 位的字符，包括字母或数字或下划线或汉字，可以用来作为用户注册时检测用户名的规则。（但汉字在 python2.x 里面可能会有问题）

\d{15,18}

表示 15 到 18 位的数字，可以用来检测身份证号码

^1\d*[x]?

以 1 开头的一串数字，数字结尾有字母 x，也可以没有。有的话就带上 x。

另外再说一下之前提到的转义字符\。如果我们确实要匹配. 或者*字符本身，而不是要它们所代表的元字符，那就需要用\. 或*。 \本身也需要用\\。比如"\d+\. \d+"可以匹配出 123. 456 这样的结果。

留一道稍稍有难度的习题：

写一个正则表达式，能匹配出多种格式的电话号码，包括

(021)88776543

010-55667890

02584453362

0571 66345673

【Python 第 59 课】 正则表达式 (5)

听说有人已经开始国庆假期了，甚至还有人中秋之后就请了年假一休到底，表示羡慕嫉妒恨！今天发完这课，我也要进入休假状态，谁也别拦着我。

来说上次的习题：

(021)88776543

010-55667890

02584453362

0571 66345673

一个可以匹配出所有结果的表达式是

`\(?:0\d{2,3}[)] -\)?\d{7,8}`

解释一下：

`\(?`

()在正则表达式里也有着特殊的含义，所以要匹配字符"("，需要用"\("。?表示这个括号是可有可无的。

`0\d{2,3}`

区号，0xx 或者 0xxx

`[] -]?`

在区号之后跟着的可能是")"、" "、"-", 也可能什么也没有。

`\d{7,8}`

7 或 8 位的电话号码

可是，这个表达式虽然能匹配出所有正确的数据（一般情况下，这样已经足够），但理论上也会匹配到错误的数据。因为()应当是成对出现的，表达式中对于左右两个括号并没有做关联处理，例如(02188776543 这样的数据也是符合条件的。

我们可以用正则表达式中的“|”符号解决这种问题。“|”相当于 python 中“or”的作用，它连接的两个表达式，只要满足其中之一，就会被算作匹配成功。

于是我们可以把()的情况单独分离出来：

`\(0\d{2,3})\d{7,8}`

其他情况：

```
0\d{2,3}[-]?\d{7,8}
```

合并：

```
\(0\d{2,3})\d{7,8}|0\d{2,3}[-]?\d{7,8}
```

使用“|”时，要特别提醒注意的是不同条件之间的顺序。匹配时，会按照从左往右的顺序，一旦匹配成功就停止验证后面的规则。假设要匹配的电话号码还有可能是任意长度的数字（如一些特殊的服务号码），你应该把

```
|\d+
```

这个条件加在表达式的最后。如果放在最前面，某些数据就可能会被优先匹配为这一条件。你可以写个测试用例体会一下两种结果的不同。

关于正则表达式，我们已经讲了 5 篇，介绍了正则表达式最最皮毛的一些用法。接下来，这个话题要稍稍告一段落。推荐一篇叫做《正则表达式 30 分钟入门教程》的文章（直接百度一下就能找到，我也会转到论坛上），想要对正则表达式进一步学习的同学可以参考。这篇教程是个标题党，里面涉及了正则表达式较多的内容，30 分钟绝对看不完。

好了，祝大家过个欢脱的长假，好好休息，多陪家人

【Python 第 60 课】随机数

有些时日没发新课了，今天来说一说 python 中的 random 模块。

random 模块的作用是产生随机数。之前的小游戏中用到过 random 中的 randint：

```
import random
```

```
num = random.randint(1,100)
```

`random.randint(a, b)`可以生成一个 `a` 到 `b` 间的随机整数，包括 `a` 和 `b`。

`a`、`b` 都必须是整数，且必须 `b ≥ a`。当等于的时候，比如：

```
random.randint(3, 3)
```

的结果就永远是 3

除了 `randint`，`random` 模块中比较常用的方法还有：

```
random.random()
```

生成一个 0 到 1 之间的随机浮点数，包括 0 但不包括 1，也就是 `[0.0, 1.0)`。

```
random.uniform(a, b)
```

生成 `a`、`b` 之间的随机浮点数。不过与 `randint` 不同的是，`a`、`b` 无需是整数，也不用考虑大小。

```
random.uniform(1.5, 3)
```

```
random.uniform(3, 1.5)
```

这两种参数都是可行的。

`random.uniform(1.5, 1.5)`永远得到 1.5。

```
random.choice(seq)
```

从序列中随机选取一个元素。`seq` 需要是一个序列，比如 `list`、元组、字符串。

```
random.choice([1, 2, 3, 5, 8, 13]) #list
```

```
random.choice('hello') #字符串
```

```
random.choice(['hello', 'world']) #字符串组成的 list
```

```
random.choice((1, 2, 3)) #元组
```

都是可行的用法。

```
random.randrange(start, stop, step)
```

生成一个从 `start` 到 `stop`（不包括 `stop`），间隔为 `step` 的一个随机数。`start`、`stop`、`step` 都要为整数，且 `start < stop`。

比如：

```
random.randrange(1, 9, 2)
```

就是从 `[1, 3, 5, 7]` 中随机选取一个。

`start` 和 `step` 都可以不提供参数，默认是从 0 开始，间隔为 1。但如果需要指定 `step`，则必须指定 `start`。

```
random.randrange(4) #[0, 1, 2, 3]
```

```
random.randrange(1, 4) #[1, 2, 3]
```

`random.randrange(start, stop, step)` 其实在效果上等同于

```
random.choice(range(start, stop, step))
```

```
random.sample(population, k)
```

从 `population` 序列中，随机获取 `k` 个元素，生成一个新序列。`sample` 不改变原来序列。

```
random.shuffle(x)
```


把序列 `x` 中的元素顺序打乱。`shuffle` 直接改变原有的序列。

以上是 `random` 中常见的几个方法。如果你在程序中需要其中某一个方法，也可以这样写：

```
from random import randint
randint(1, 10)
```

另外，有些编程基础的同学可能知道，在随机数中有个 `seed` 的概念，需要一个真实的随机数，比如此刻的时间、鼠标的位置等等，以此为基础产生伪随机数。在 `python` 中，默认用系统时间作为 `seed`。你也可以手动调用 `random.seed(x)` 来指定 `seed`。

python 模块的常用安装方式

之前我们讲过一些 `python` 的模块，如 `chardet`、`pygame`，这些模块不包含在 `python` 的默认代码中，需要从外部下载并安装。有些模块提供了自动安装的文件，比如 `pygame` 的 `windows` 版本，直接双击安装就可以。但大多数模块没有提供这样的安装方式，有些同学没能成功安装而导致无法在程序中引入模块。在这里，介绍一下 `python` 模块几种常见的安装方法。

1. 直接 copy

下载的模块文件中已经有了模块的文件，有些模块只有一个文件，比如较早版本的 `BeautifulSoup`，有些是一个文件夹，比如新版本 `BeautifulSoup` 就是一个叫做 `bs4` 的文件夹。

把这些文件直接 `copy` 到你的 `python` 路径下的 `/Lib/site-packages` 文件夹中，比如 `C:/Python27/Lib/site-packages`。之后就可以在程序里直接引用了：

```
import BeautifulSoup
或者
from bs4 import BeautifulSoup
```

这是根据你放置的文件位置不同而决定的。

网上有人说直接放在 `Lib` 文件夹中就可以了。的确这样也行，但 `Lib` 文件夹中都是自带的模块，看一下就会发现我们用过的 `random`、`re` 等模块的代码文件。而外部模块一般放在 `site-packages` 文件夹中。

2. setup.py

很多模块里都附带了 `setup.py` 文件，有同学直接双击了，然后发现没有用。

它的使用方法是到命令行去到 `setup.py` 所在的路径下，运行

```
python setup.py install
```

仔细看一下安装时输出的信息可以发现，这个命令做的事情其实也就是帮你把模块的代码 `copy` 到 `site-packages` 文件夹。

3. setuptools

使用 `setuptools` 可以直接根据模块名称来自动下载安装，不需要自己再去寻找模块的安装文件。不过在使用之前，你得先安装 `setuptools` 自身。

windows 平台的 32 位 python，可以直接下载 `setuptools` 的 exe 文件安装。（去搜索 `setuptools windows` 可以找到，我也上传了一份在论坛本帖后面）

Linux 用户可以从包管理器中安装，比如 ubuntu:

```
apt-get install python-setuptools
```

windows 平台 64 位 python 得用 `ez_setup.py` 进行安装（文件我也上传了）。这种方式也适用于所有平台。

在 `ez_setup.py` 所在文件夹下运行：

```
python ez_setup.py
```

`setuptools` 会被安装在 python 路径\Scripts 下。之后，你可以把这个路径添加到环境变量 `path` 中，也可以直接从命令行进入到 `Scripts` 文件夹下，执行 `easy_install`，看看是否安装成功了。

之后，你就可以直接用它来安装你想要的模块，比如 `PIL`：

```
easy_install PIL
```

程序就会帮你自动下载安装到 `site-packages` 里。

最后，介绍几个不错的模块，供大家参考使用。

`PIL` - 图形处理

`PyXML` - 解析和处理 XML 文件

`MySQLdb` - 连接 MySQL 数据库

`Tkinter` - 图形界面接口，python 自带

`smtplib` - 发送电子邮件

`ftplib` - ftp 编程

`PyMedia` - 多媒体操作

`PyOpenGL` - OpenGL 接口

`BeautifulSoup` - HTML/XML 的解析器

正则表达式 30 分钟入门教程

目录

[跳过目录](#)

1. [本文目标](#)
2. [如何使用本教程](#)
3. [正则表达式到底是什么东西？](#)
4. [入门](#)
5. [测试正则表达式](#)
6. [元字符](#)
7. [字符转义](#)
8. [重复](#)
9. [字符类](#)
10. [分枝条件](#)
11. [反义](#)
12. [分组](#)
13. [后向引用](#)
14. [零宽断言](#)
15. [负向零宽断言](#)
16. [注释](#)
17. [贪婪与懒惰](#)
18. [处理选项](#)
19. [平衡组/递归匹配](#)
20. [还有什么东西没提到](#)
21. [联系作者](#)
22. [网上的资源及本文参考文献](#)
23. [更新纪录](#)

本文目标

30 分钟内让你明白正则表达式是什么，并对它有一些基本的了解，让你可以在自己的程序或网页里使用它。

如何使用本教程

最重要的是——请给我 *30 分钟*，如果你没有使用正则表达式的经验，请不要试图在 *30 秒*内入门——除非你是超人 :))

别被下面那些复杂的表达式吓倒，只要跟着我一步一步来，你会发现正则表达式其实并没有想像中的那么困难。当然，如果你看完了这篇教程之后，发现自己明白了很多，却又几乎什么都记不得，那也是很正常的——我认为，没接触过正则表达式的人在看完这篇教程后，能把提到过的语法记住 80% 以上的可能性为零。这里只是让你明白基本的原理，以后你还需要多练习，多使用，才能熟练掌握正则表达式。

除了作为入门教程之外，本文还试图成为可以在日常工作中使用的正则表达式语法参考手册。就作者本人的经历来说，这个目标还是完成得不错的——你看，我自己也没能把所有的东西记下来，不是吗？

[清除格式](#) 文本格式约定：专业术语 元字符/语法格式 正则表达式 正则表达式中的一部分(用于分析) 对其进行匹配的源字符串 对正则表达式或其中一部分的说明

[隐藏边注](#) 本文右边有一些注释，主要是用来提供一些相关信息，或者给没有程序员背景的读者解释一些基本概念，通常可以忽略。

正则表达式到底是什么东西？

字符是计算机软件处理文字时最基本的单位，可能是字母，数字，标点符号，空格，换行符，汉字等等。字符串是 0 个或多个字符的序列。文本也就是文字，字符串。说某个字符串匹配某个正则表达式，通常是指这个字符串里有一部分(或几部分分别)能满足表达式给出的条件。

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。

很可能你使用过 Windows/Dos 下用于文件查找的通配符(wildcard)，也就是*和?。如果你想查找某个目录下的所有的 Word 文档的话，你会搜索*.doc。在这里，*会被解释成任意的字符串。和通配符类似，正则表达式也是用来进行文本匹配的工具，只不过比起通配符，它能更精确地描述你的需求——当然，代价就是更复杂——比如你可以编写一个正则表达式，用来查找所有以 0 开头，后面跟着 2-3 个数字，然后是一个连字号“-”，最后是 7 或 8 位数字的字符串(像 010-12345678 或 0376-7654321)。

入门

学习正则表达式的最好方法是从例子开始，理解例子之后再自己对例子进行修改，实验。下面给出了不少简单的例子，并对它们作了详细的说明。

假设你在一篇英文小说里查找 hi，你可以使用正则表达式 hi。

这几乎是最简单的正则表达式了，它可以精确匹配这样的字符串：由两个字符组成，前一个字符是 h，后一个是 i。通常，处理正则表达式的工具会提供一个忽略大小写的选项，如果选中了这个选项，它可以匹配 hi, HI, Hi, hI 这四种情况中的任意一种。

不幸的是,很多单词里包含hi这两个连续的字符,比如him,history,high等等。用hi来查找的话,这里边的hi也会被找出来。如果要精确地查找hi这个单词的话,我们应该使用**\bhi\b**。

\b是正则表达式规定的一个特殊代码(好吧,某些人叫它元字符,metacharacter),代表着单词的开头或结尾,也就是单词的分界处。虽然通常英文的单词是由空格,标点符号或者换行来分隔的,但是**\b**并不匹配这些单词分隔字符中的任何一个,它**只匹配一个位置**。

如果需要更精确的说法,**\b**匹配这样的位置:它的前一个字符和后一个字符不全是(一个是,一个不是或不存在)**\w**。

假如你要找的是hi后面不远处跟着一个Lucy,你应该用**\bhi\b.*\bLucy\b**。

这里,**.**是另一个元字符,匹配除了换行符以外的任意字符。*****同样是元字符,不过它代表的不是字符,也不是位置,而是数量——它指定*****前边的内容可以连续重复使用任意次以使整个表达式得到匹配。因此,**.***连在一起就意味着任意数量的不包含换行的字符。现在**\bhi\b.*\bLucy\b**的意思就很明显了:先是一个单词hi,然后是任意个任意字符(但不能是换行),最后是Lucy这个单词。

换行符就是'**\n**',ASCII编码为10(十六进制0x0A)的字符。

如果同时使用其它元字符,我们就能构造出功能更强大的正则表达式。比如下面这个例子:

0\d\d-\d\d\d\d\d\d\d\d匹配这样的字符串:以0开头,然后是两个数字,然后是一个连字号“-”,最后是8个数字(也就是中国的电话号码。当然,这个例子只能匹配区号为3位的情形)。

这里的**\d**是个新的元字符,匹配一位数字(0,或1,或2,或……)。-不是元字符,只匹配它本身——连字符(或者减号,或者中横线,或者随你怎么称呼它)。

为了避免那么多烦人的重复,我们也可以这样写这个表达式:**0\d{2}-\d{8}**。这里**\d**后面的**{2}**(**{8}**)的意思是前面**\d**必须连续重复匹配2次(8次)。

测试正则表达式

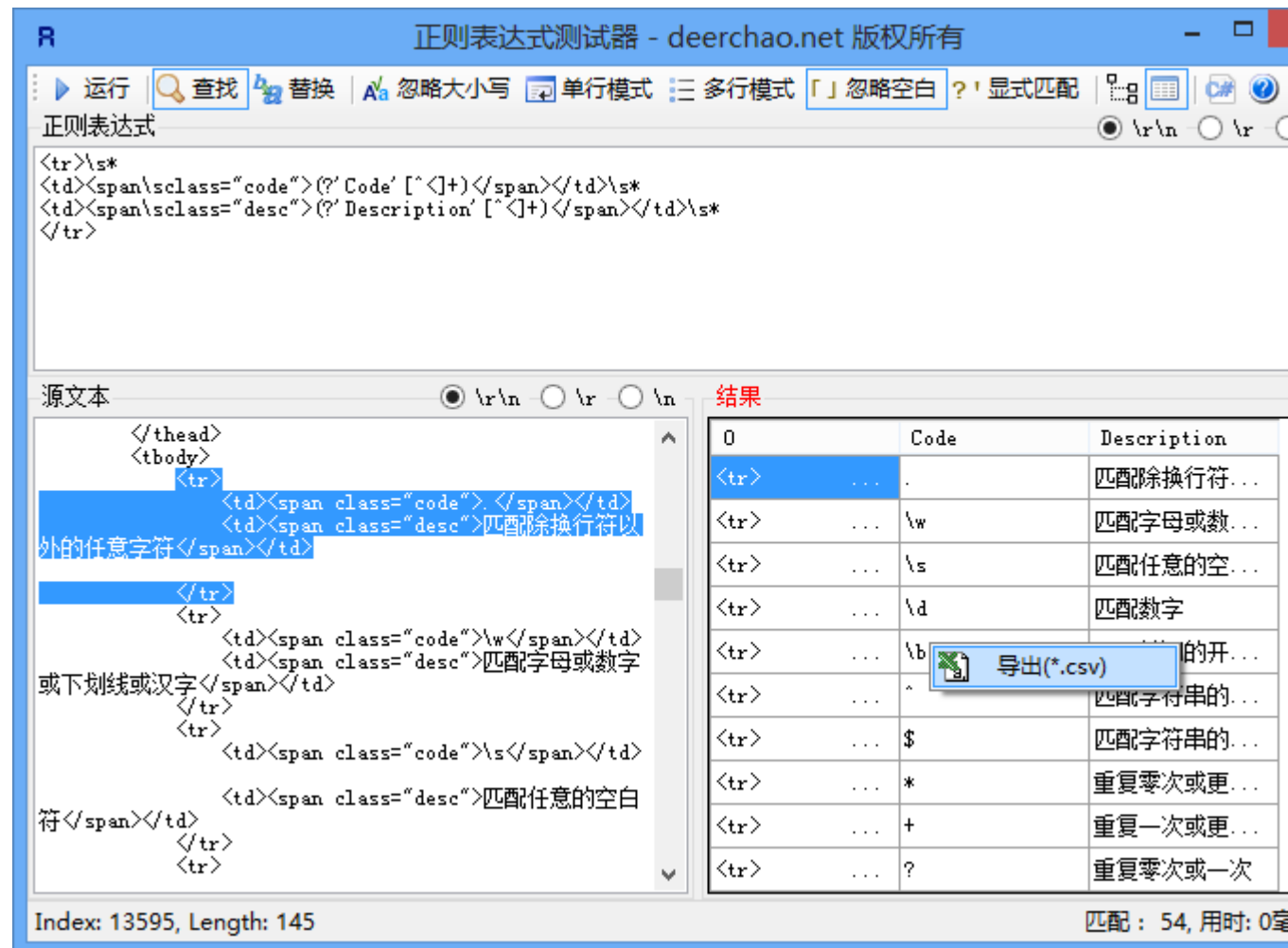
其它可用的测试工具:

- [RegexBuddy](#)
- [Javascript 正则表达式在线测试工具](#)

如果你不觉得正则表达式很难读写的话,要么你是一个天才,要么,你不是地球人。正则表达式的语法很令人头疼,即使对经常使用它的人来说也是如此。由于难于读写,容易出错,所以找一种工具对正则表达式进行测试是很有必要的。

不同的环境下正则表达式的一些细节是不相同的，本教程介绍的是微软 .Net Framework 4.0 下正则表达式的行为，所以，我向你推荐我编写的.Net 下的工具 [正则表达式测试器](#)。请参考该页面的说明来安装和运行该软件。

下面是 Regex Tester 运行时的截图:



元字符

现在你已经知道几个很有用的元字符了，如**\b**、**.**、*****，还有**\d**。正则表达式里还有更多的元字符，比如**\s**匹配任意的空白符，包括空格，制表符(Tab)，换行符，中文全角空格等。**\w**匹配字母或数字或下划线或汉字等。

对中文/汉字的特殊处理是由 .Net 提供的正则表达式引擎支持的，其它环境下的具体情况请查看相关文档。

下面来看看更多的例子:

`\ba\w*\b` 匹配以字母 a 开头的单词——先是某个单词开始处(`\b`)，然后是字母 a，然后是任意数量的字母或数字(`\w*`)，最后是单词结束处(`\b`)。

好吧，现在我们说说正则表达式里的单词是什么意思吧：就是不少于一个的连续的`\w`。不错，这与学习英文时要背的成千上万个同名的东西的确关系不大：)

`\d+` 匹配 1 个或更多连续的数字。这里的+是和*类似的元字符，不同的是*匹配重复任意次(可能是 0 次)，而+则匹配重复 1 次或更多次。

`\b\w{6}\b` 匹配刚好 6 个字符的单词。

表 1.常用的元字符

代码	说明
<code>.</code>	匹配除换行符以外的任意字符
<code>\w</code>	匹配字母或数字或下划线或汉字
<code>\s</code>	匹配任意的空白符
<code>\d</code>	匹配数字
<code>\b</code>	匹配单词的开始或结束
<code>^</code>	匹配字符串的开始
<code>\$</code>	匹配字符串的结束

正则表达式引擎通常会提供一个“测试指定的字符串是否匹配一个正则表达式”的方法，如 JavaScript 里的 `RegExp.test()` 方法或 .NET 里的 `Regex.IsMatch()` 方法。这里的匹配是指是字符串里有没有符合表达式规则的部分。如果不使用`^`和`$`的话，对于`\d{5,12}`而言，使用这样的方法就只能保证字符串里包含 5 到 12 连续位数字，而不是整个字符串就是 5 到 12 位数字。

元字符`^`（和数字 6 在同一个键位上的符号）和`$`都匹配一个位置，这和`\b`有点类似。`^`匹配你要用来查找的字符串的开头，`$`匹配结尾。这两个代码在验证输入的内容时非常有用，比如一个网站如果要求你填写的 QQ 号必须为 5 位到 12 位数字时，可以使用：`^\d{5,12}$`。

这里的`{5,12}`和前面介绍过的`{2}`是类似的，只不过`{2}`匹配只能不多不少重复 2 次，`{5,12}`则是重复的次数不能少于 5 次，不能多于 12 次，否则都不匹配。

因为使用了`^`和`$`，所以输入的整个字符串都要用来和`\d{5,12}`来匹配，也就是说整个输入必须是 5 到 12 个数字，因此如果输入的 QQ 号能匹配这个正则表达式的话，那就符合要求了。

和忽略大小写的选项类似，有些正则表达式处理工具还有一个处理多行的选项。如果选中了这个选项，`^`和`$`的意义就变成了匹配行的开始处和结束处。

字符转义

如果你想查找元字符本身的话，比如你查找`.`，或者`*`，就出现了问题：你没办法指定它们，因为它们会被解释成别的意思。这时你就得使用`\`来取消这些字符的特殊意义。因此，你应该使用`\.`和`*`。当然，要查找`\`本身，你也得用`\\`。

例如：`deerchao\.net` 匹配 `deerchao.net`，`C:\\Windows` 匹配 `C:\Windows`。

重复

你已经看过了前面的`*`，`+`，`{2}`，`{5, 12}`这几个匹配重复的方式了。下面是正则表达式中所有的限定符(指定数量的代码，例如`*`，`{5, 12}`等)：

表 2.常用的限定符

代码/语法	说明
<code>*</code>	重复零次或更多次
<code>+</code>	重复一次或更多次
<code>?</code>	重复零次或一次
<code>{n}</code>	重复 <code>n</code> 次
<code>{n,}</code>	重复 <code>n</code> 次或更多次
<code>{n,m}</code>	重复 <code>n</code> 到 <code>m</code> 次

下面是一些使用重复的例子：

`Windows\d+`匹配 `Windows` 后面跟 1 个或更多数字

`^\w+`匹配一行的第一个单词(或整个字符串的第一个单词，具体匹配哪个意思得看选项设置)

字符类

要想查找数字，字母或数字，空白是很简单的，因为已经有了对应这些字符集合的元字符，但是如果你想匹配没有预定义元字符的字符集合(比如元音字母 `a, e, i, o, u`)，应该怎么办？

很简单，你只需要在方括号里列出它们就行了，像`[aeiou]`就匹配任何一个英文元音字母，`[.?!]`匹配标点符号(`.`或`?`或`!`)。

我们也可以轻松地指定一个字符范围，像`[0-9]`代表的含意与`\d`就是完全一致的：一位数字；同理`[a-z0-9A-Z_]`也完全等同于`\w`（如果只考虑英文的话）。

下面是一个更复杂的表达式：`\(?:\d{2}[-]? \d{8})`。

“(”和“)”也是元字符，后面的[分组节](#)里会提到，所以在这里需要使用[转义](#)。

这个表达式可以匹配几种格式的电话号码，像(010)88886666，或022-22334455，或02912345678等。我们对它进行一些分析吧：首先是一个转义字符\，它能出现0次或1次(?)，然后是一个0，后面跟着2个数字(\d{2})，然后是)或-或空格中的一个，它出现1次或不出现(?)，最后是8个数字(\d{8})。

分枝条件

不幸的是，刚才那个表达式也能匹配010)12345678或(022-87654321这样的“不正确”的格式。要解决这个问题，我们需要用到分枝条件。正则表达式里的分枝条件指的是有几种规则，如果满足其中任意一种规则都应该当成匹配，具体方法是用|把不同的规则分隔开。听不明白？没关系，看例子：

0\d{2}-\d{8}|0\d{3}-\d{7} 这个表达式能匹配两种以连字号分隔的电话号码：一种是三位区号，8位本地号(如010-12345678)，一种是4位区号，7位本地号(0376-2233445)。

\(?:0\d{2}\)?[-]?\d{8}|0\d{2}[-]?\d{8} 这个表达式匹配3位区号的电话号码，其中区号可以用小括号括起来，也可以不用，区号与本地号间可以用连字号或空格间隔，也可以没有间隔。你可以试试用分枝条件把这个表达式扩展成也支持4位区号的。

\d{5}-\d{4}|\d{5} 这个表达式用于匹配美国的邮政编码。美国邮编的规则是5位数字，或者用连字号间隔的9位数字。之所以要给出这个例子是因为它能说明一个问题：**使用分枝条件时，要注意各个条件的顺序**。如果你把它改成\d{5}|\d{5}-\d{4}的话，那么就只会匹配5位的邮编(以及9位邮编的前5位)。原因是匹配分枝条件时，将会从左到右地测试每个条件，如果满足了某个分枝的话，就不会再去再管其它的条件了。

分组

我们已经提到了怎么重复单个字符(直接在字符后面加上限定符就行了)；但如果想要重复多个字符又该怎么办？你可以用小括号来指定子表达式(也叫做分组)，然后你就可以指定这个子表达式的重复次数了，你也可以对子表达式进行其它一些操作(后面会有介绍)。

(\d{1,3}\.){3}\d{1,3} 是一个简单的IP地址匹配表达式。要理解这个表达式，请按下列顺序分析它：\d{1,3}匹配1到3位的数字，(\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个分组)重复3次，最后再加上一个一到三位的数字(\d{1,3})。

IP 地址中每个数字都不能大于 255。经常有人问我, 01.02.03.04 这样前面带有 0 的数字, 是不是正确的 IP 地址呢? 答案是: 是的, IP 地址里的数字可以包含有前导 0 (leading zeroes)。

不幸的是, 它也将匹配 256.300.888.999 这种不可能存在的 IP 地址。如果能使用算术比较的话, 或许能简单地解决这个问题, 但是正则表达式中并不提供关于数学的任何功能, 所以只能使用冗长的分组, 选择, 字符类来描述一个正确的 IP 地址:

`((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)`。

理解这个表达式的关键是理解 `2[0-4]\d|25[0-5]|[01]?\d\d?`, 这里我就不细说了, 你自己应该能分析得出来它的意义。

反义

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外, 其它任意字符都行的情况, 这时需要用到反义:

表 3.常用的反义代码

代码/语法	说明
<code>\W</code>	匹配任意不是字母, 数字, 下划线, 汉字的字符
<code>\S</code>	匹配任意不是空白符的字符
<code>\D</code>	匹配任意非数字的字符
<code>\B</code>	匹配不是单词开头或结束的位置
<code>[^x]</code>	匹配除了 x 以外的任意字符
<code>[^aeiou]</code>	匹配除了 aeiou 这几个字母以外的任意字符

例子: `\S+`匹配不包含空白符的字符串。

`<a[^>]+>`匹配用尖括号括起来的以 a 开头的字符串。

后向引用

使用小括号指定一个子表达式后, **匹配这个子表达式的文本**(也就是此分组捕获的内容)可以在表达式或其它程序中作进一步的处理。默认情况下, 每个分组会自动拥有一个组号, 规则是: 从左向右, 以分组的左括号为标志, 第一个出现的分组的组号为 1, 第二个为 2, 以此类推。

呃……其实, 组号分配还不像我刚说得那么简单:

- 分组 0 对应整个正则表达式
- 实际上组号分配过程是要从左向右扫描两遍的: 第一遍只给未命名组分配, 第二遍只给命名组分配——因此所有命名组的组号都大于未命名的组号

- 你可以使用(?:exp)这样的语法来剥夺一个分组对组号分配的参与权。

后向引用用于重复搜索前面某个分组匹配的文本。例如，\1 代表分组 1 匹配的文本。难以理解？请看示例：

\b(\w+)\b\s+\1\b 可以用来匹配重复的单词，像 go go，或者 kitty kitty。这个表达式首先是一个单词，也就是单词开始处和结束处之间的多于一个的字母或数字(\b(\w+)\b)，这个单词会被捕获到编号为 1 的分组中，然后是 1 个或几个空白符(\s+)，最后是分组 1 中捕获的内容（也就是前面匹配的那个单词）(\1)。

你也可以自己指定子表达式的组名。要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+)（或者把尖括号换成'也行：(?' Word' \w+)），这样就把 \w+ 的组名指定为 Word 了。要反向引用这个分组捕获的内容，你可以使用 \k<Word>，所以上一个例子也可以写成这样：\b(?<Word>\w+)\b\s+\k<Word>\b。

使用小括号的时候，还有很多特定用途的语法。下面列出了最常用的一些：

表 4.常用分组语法

分类	代码/语法	说明
	(exp)	匹配 exp,并捕获文本到自动命名的组里
捕获	(?<name>exp)	匹配 exp,并捕获文本到名称为 name 的组里，也可以写成(?'name'exp)
	(?:exp)	匹配 exp,不捕获匹配的文本，也不给此分组分配组号
	(?=exp)	匹配 exp 前面的位置
零宽断言	(?<=exp)	匹配 exp 后面的位置
言	(?!exp)	匹配后面跟的不是 exp 的位置
	(?<!exp)	匹配前面不是 exp 的位置
注释	(?#comment)	这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读

我们已经讨论了前两种语法。第三个(?:exp)不会改变正则表达式的处理方式，只是这样的组匹配的内容不会像前两种那样被捕获到某个组里面，也不会拥有组号。“我为什么会想要这样做？”——好问题，你觉得为什么呢？

零宽断言

地球人，是不是觉得这些术语名称太复杂，太难记了？我也有同感。知道有这么一种东西就行了，它叫什么，随它去吧！人若无名，便可专心练剑；物若无名，便可随意取舍……

接下来的四个用于查找在某些内容(但并不包括这些内容)之前或之后的东西，也就是说它们像\b, ^, \$那样用于指定一个位置，这个位置应该满足一定的条件(即断言)，因此它们也被称为零宽断言。最好还是拿例子来说明吧：

断言用来声明一个应该为真的事实。正则表达式中只有当断言为真时才会继续进行匹配。

(?=exp) 也叫零宽度正预测先行断言，它断言自身出现的位置的后面能匹配表达式 exp。比如 \b\w+(?=ing\b)，匹配以 ing 结尾的单词的前面部分(除了 ing 以外的部分)，如查找 I'm singing while you're dancing. 时，它会匹配 sing 和 danc。

(?<=exp) 也叫零宽度正回顾后发断言，它断言自身出现的位置的前面能匹配表达式 exp。比如 (?<=\bre)\w+\b 会匹配以 re 开头的单词的后半部分(除了 re 以外的部分)，例如在查找 reading a book 时，它匹配 ading。

假如你想要给一个很长的数字中每三位间加一个逗号(当然是从右边加起了)，你可以这样查找需要在前面和里面添加逗号的部分：((?<=\d)\d{3})+\b，用它对 1234567890 进行查找时结果是 234567890。

下面这个例子同时使用了这两种断言：(?<=\s)\d+(?=\s) 匹配以空白符间隔的数字(再次强调，不包括这些空白符)。

负向零宽断言

前面我们提到过怎么查找不是某个字符或不在某个字符类里的字符的方法(反义)。但是如果只是想要确保某个字符没有出现，但并不想去匹配它时怎么办？例如，如果我们想查找这样的单词——它里面出现了字母 q，但是 q 后面跟的不是字母 u，我们可以尝试这样：

\b\w*q[[^]u]\w*\b 匹配包含后面不是字母 u 的字母 q 的单词。但是如果多做测试(或者你思维足够敏锐，直接就观察出来了)，你会发现，如果 q 出现在单词的结尾的话，像 **Iraq, Benq**，这个表达式就会出错。这是因为 [[^]u] 总要匹配一个字符，所以如果 q 是单词的最后一个字符的话，后面的 [[^]u] 将会匹配 q 后面的单词分隔符(可能是空格，或者是句号或其它的什么)，后面的 \w*\b 将会匹配下一个单词，于是 \b\w*q[[^]u]\w*\b 就能匹配整个 Iraq fighting。负向零宽断言能解决这样的问题，因为它只匹配一个位置，并不消费任何字符。现在，我们可以这样来解决这个问题：\b\w*q(?!u)\w*\b。

零宽度负预测先行断言 (?!exp)，断言此位置的后面不能匹配表达式 exp。例如：

\d{3} (?!\d) 匹配三位数字，而且这三位数字的后面不能是数字；

\b((?!abc)\w)+\b 匹配不包含连续字符串 abc 的单词。

同理，我们可以用 (?<!\exp)，零宽度负回顾后发断言来断言此位置的前面不能匹配表达式 exp：(?<![a-z])\d{7} 匹配前面不是小写字母的七位数字。

请详细分析表达式 (?<=(\w+>)).*(?<=\\|1>)，这个表达式最能表现零宽断言的真正用途。

一个更复杂的例子：`(?<=<(\w+)>).*?(?=<\/\1>)` 匹配不包含属性的简单 HTML 标签内里的内容。`(?<=<(\w+)>)` 指定了这样的前缀：被尖括号括起来的单词（比如可能是 ``），然后是 `.*`（任意的字符串），最后是一个后缀 `(?=<\/\1>)`。注意后缀里的 `\`，它用到了前面提过的字符转义；`\1` 则是一个反向引用，引用的正是捕获的第一组，前面的 `(\w+)` 匹配的内容，这样如果前缀实际上是 `` 的话，后缀就是 `` 了。整个表达式匹配的是 `` 和 `` 之间的内容（再次提醒，不包括前缀和后缀本身）。

注释

小括号的另一种用途是通过语法 `(?#comment)` 来包含注释。例如：

```
2[0-4]\d(?#200-249)|25[0-5](?#250-255)|[01]? \d\d(?#0-199)。
```

要包含注释的话，最好是启用“忽略模式里的空白符”选项，这样在编写表达式时能任意的添加空格，Tab，换行，而实际使用时这些都将忽略。启用这个选项后，在 `#` 后面到这一行结束的所有文本都将被当成注释忽略掉。例如，我们可以前面的一个表达式写成这样：

```
(?<=      # 断言要匹配的文本的前缀
<(\w+)>  # 查找尖括号括起来的字母或数字(即 HTML/XML 标签)
)        # 前缀结束
.*       # 匹配任意文本
(?=      # 断言要匹配的文本的后缀
<\/\1>  # 查找尖括号括起来的内容：前面是一个"/"，后面是先前捕获
的标签
)        # 后缀结束
```

贪婪与懒惰

当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配**尽可能多**的字符。以这个表达式为例：`a.*b`，它将会匹配最长的以 `a` 开始，以 `b` 结束的字符串。如果用它来搜索 `aabab` 的话，它会匹配整个字符串 `aabab`。这被称为贪婪匹配。

有时，我们更需要懒惰匹配，也就是匹配**尽可能少**的字符。前面给出的限定符都可以被转化为懒惰匹配模式，只要在其后面加上一个问号 `?`。这样 `.*?` 就意味着匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。现在看看懒惰版的例子吧：

`a.*?b` 匹配最短的，以 `a` 开始，以 `b` 结束的字符串。如果把它应用于 `aabab` 的话，它会匹配 `aab`（第一到第三个字符）和 `ab`（第四到第五个字符）。

为什么第一个匹配是 aab (第一到第三个字符) 而不是 ab (第二到第三个字符)? 简单地说, 因为正则表达式有另一条规则, 比懒惰 / 贪婪规则的优先级更高: 最先开始的匹配拥有最高的优先权——The match that begins earliest wins。

表 5.懒惰限定符

代码/语法	说明
*?	重复任意次, 但尽可能少重复
+?	重复 1 次或更多次, 但尽可能少重复
??	重复 0 次或 1 次, 但尽可能少重复
{n,m}?	重复 n 到 m 次, 但尽可能少重复
{n,}?	重复 n 次以上, 但尽可能少重复

处理选项

在 C# 中, 你可以使用 [Regex\(String, RegexOptions\) 构造函数](#) 来设置正则表达式的处理选项。如: `Regex regex = new Regex(@"\ba\w{6}\b", RegexOptions.IgnoreCase);`

上面介绍了几个选项如忽略大小写, 处理多行等, 这些选项能用来改变处理正则表达式的方式。下面是 .Net 中常用的正则表达式选项:

表 6.常用的处理选项

名称	说明
IgnoreCase(忽略大小写)	匹配时不区分大小写。
Multiline(多行模式)	更改 ^ 和 \$ 的含义, 使它们分别在任意一行的行首和行尾匹配, 而不仅仅在整个字符串的开头和结尾匹配。(在此模式下, \$ 的精确含意是: 匹配 \n 之前的位置以及字符串结束前的位置。)
Singleline(单行模式)	更改 . 的含义, 使它与每一个字符匹配 (包括换行符 \n)。
IgnorePatternWhitespace(忽略空白)	忽略表达式中的非转义空白并启用由 # 标记的注释。
ExplicitCapture(显式捕获)	仅捕获已被显式命名的组。

一个经常被问到的问题是: 是不是只能同时使用多行模式和单行模式中的一种? 答案是: 不是。这两个选项之间没有任何关系, 除了它们的名字比较相似 (以至于让人感到疑惑) 以外。

平衡组/递归匹配

这里介绍的平衡组语法是由 .Net Framework 支持的; 其它语言 / 库不一定支持这种功能, 或者支持此功能但需要使用不同的语法。

有时我们需要匹配像 $(100 * (50 + 15))$ 这样的可嵌套的层次性结构，这时简单地使用 `\(.+\)` 则只会匹配到最左边的左括号和最右边的右括号之间的内容（这里我们讨论的是贪婪模式，懒惰模式也有下面的问题）。假如原来的字符串里的左括号和右括号出现的次数不相等，比如 $(5 / (3 + 2))$ ，那我们的匹配结果里两者的个数也不会相等。有没有办法在这样的字符串里匹配到最长的，配对的括号之间的内容呢？

为了避免（和\（把你的大脑彻底搞糊涂，我们还是用尖括号代替圆括号吧。现在我们的问题变成了如何把 `xx <aa <bbb> <bbb> aa> yy` 这样的字符串里，最长的配对的尖括号内的内容捕获出来？

这里需要用到以下的语法构造：

- `(?'group')` 把捕获的内容命名为 `group`，并压入堆栈(Stack)
- `(?'-group')` 从堆栈上弹出最后压入堆栈的名为 `group` 的捕获内容，如果堆栈本来为空，则本分组的匹配失败
- `(?(group)yes|no)` 如果堆栈上存在以名为 `group` 的捕获内容的话，继续匹配 `yes` 部分的表达式，否则继续匹配 `no` 部分
- `(?!)` 零宽负向先行断言，由于没有后缀表达式，试图匹配总是失败

如果你不是一个程序员（或者你自称程序员但是不知道堆栈是什么东西），你就这样理解上面的三种语法吧：第一个就是在黑板上写一个“group”，第二个就是从黑板上擦掉一个“group”，第三个就是看黑板上写的还有没有“group”，如果有就继续匹配 `yes` 部分，否则就匹配 `no` 部分。

我们需要做的是每碰到了左括号，就在压入一个“Open”，每碰到一个右括号，就弹出一个，到了最后就看看堆栈是否为空——如果不为空那就证明左括号比右括号多，那匹配就应该失败。正则表达式引擎会进行回溯（放弃最前面或最后面的一些字符），尽量使整个表达式得到匹配。

```
<                                #最外层的左括号
[ ^ < > ] *                      #最外层的左括号后面的不是括号的内容
(
    (
        (?'Open' <)              #碰到了左括号，在黑板上写一个“Open”
        [ ^ < > ] *              #匹配左括号后面的不是括号的内容
    ) +
    (
        (?'-Open' >)             #碰到了右括号，擦掉一个“Open”
        [ ^ < > ] *              #匹配右括号后面不是括号的内容
    ) +
) *
(?(Open) (?!))                  #在遇到最外层的右括号前面，判断黑板上还有没有没擦掉的“Open”；如果还有，则匹配失败
```

> #最外层的右括号

平衡组的一个最常见的应用就是匹配 HTML, 下面这个例子可以匹配嵌套的<div>标签:

```
<div[^>]*>[<>]*(((?'Open' <div[^>]*>)[<>]*)+(((?'-Open' </div>)[<>]*)+)*(? (Open) (!))</div>.
```

还有些什么东西没提到

上边已经描述了构造正则表达式的大量元素, 但是还有很多没有提到的东西。下面是一些未提到的元素的列表, 包含语法和简单的说明。你可以在网上找到更详细的参考资料来学习它们——当你需要用到它们的时候。如果你安装了 MSDN Library, 你也可以在里面找到 .net 下正则表达式详细的文档。这里的介绍很简略, 如果你需要更详细的信息, 而又没有在电脑上安装 MSDN Library, 可以查看[关于正则表达式语言元素的 MSDN 在线文档](#)。

表 7.尚未详细讨论的语法



代码/语法	说明
\a	报警字符(打印它的效果是电脑嘀一声)
\b	通常是单词分界位置, 但如果在字符类里使用代表退格
\t	制表符, Tab
\r	回车
\v	竖向制表符
\f	换页符
\n	换行符
\e	Escape
\Onn	ASCII 代码中八进制代码为 nn 的字符
\xnn	ASCII 代码中十六进制代码为 nn 的字符
\unnnn	Unicode 代码中十六进制代码为 nnnn 的字符
\cN	ASCII 控制字符。比如\cC 代表 Ctrl+C
\A	字符串开头(类似^, 但不受处理多行选项的影响)
\Z	字符串结尾或行尾(不受处理多行选项的影响)
\z	字符串结尾(类似\$, 但不受处理多行选项的影响)
\G	当前搜索的开头
\p{name}	Unicode 中命名为 name 的字符类, 例如\p{IsGreek}
(?>exp)	贪婪子表达式
(?<x>-<y>exp)	平衡组
(?im-nsx:exp)	在子表达式 exp 中改变处理选项
(?im-nsx)	为表达式后面的部分改变处理选项
(?(exp)yes no)	把 exp 当作零宽正向先行断言, 如果在这个位置能匹配, 使用 yes 作为此组的表达式; 否则使用 no

表 7.尚未详细讨论的语法

代码/语法	说明
(?(exp)yes)	同上，只是使用空表达式作为 no
(?(name)yes no)	如果命名为 name 的组捕获到了内容，使用 yes 作为表达式；否则使用 no
(?(name)yes)	同上，只是使用空表达式作为 no

联系作者

好吧，我承认，我骗了你，读到这里你肯定花了不少 30 分钟。相信我，这是我的错，而不是因为你太笨。我之所以说“30 分钟”，是为了让你有信心，有耐心继续下去。既然你看到了这里，那证明我的阴谋成功了。被忽悠的感觉很爽吧？

要投诉我，或者觉得我其实可以忽悠得更高明，欢迎来  [我的微博](#) 让我知道。如果你有关于正则表达式的问题，可以到  [stackoverflow](#) 网站上提问，记得要添加 regex 标签。如果你更习惯于用中文交流，可以到微博上用 #正则# 标签提出问题。

网上的资源及本文参考文献

- [精通正则表达式（第 3 版）](#)
- [微软的正则表达式教程](#)
- [System.Text.RegularExpressions.Regex 类\(MSDN\)](#)
- [专业的正则表达式教学网站\(英文\)](#)
- [关于 .Net 下的平衡组的详细讨论（英文）](#)

更新纪录

1. 2006-3-27 第一版
2. 2006-10-12 第二版
 - 修正了几个细节上的错误和不准确的地方
 - 增加了对处理中文时的一些说明
 - 更改了几个术语的翻译（采用了 MSDN 的翻译方式）
 - 增加了平衡组的介绍
 - 放弃了对 The Regulator 的介绍，改用 Regex Tester
3. 2007-3-12 V2.1
 - 修正了几个小的错误
 - 增加了对处理选项(RegexOptions)的介绍
4. 2007-5-28 V2.2
 - 重新组织了对零宽断言的介绍
 - 删除了几个不太合适的示例，添加了几个实用的示例
 - 其它一些微小的更改
5. 2007-8-3 V2.21

- 修改了几处文字错误
 - 修改/添加了对`$,\b`的精确说明
 - 承认了作者是个骗子
 - 给 `RegexTester` 添加了 `Singleline` 选项的相关功能
6. 2008-4-13 v2.3
- 调整了部分章节的次序
 - 修改了页面布局，删除了专门的参考节
 - 针对读者的反馈，调整了部分内容
7. 2009-4-11 v2.31
- 修改了几处文字错误
 - 添加了一些注释说明
 - 调整了一些措词
8. 2011-8-17 v2.32
- 更改了工具介绍，换用自行开发的正则表达式测试器
9. 2013-1-10 v2.33
- 说明包含前导 0 的 IP 地址是合法的