

对于坚持练习的一点提示

在你通过这本书学习编程时，我正在学习弹吉他。我每天至少训练 2 小时，至少花一个小时练习音阶、和声、和琶音，剩下的时间用来学习音乐理论和歌曲演奏以及训练听力等。有时我一天会花 8 个小时来练习，因为我觉得这是一件有趣的事情。对我来说，要学好一样东西，每天的练习是必不可少的。就算这天个人状态很差，或者说学习的课题实在太难，你也不必介意，只要坚持尝试，总有一天困难会变得容易，枯燥也会变得有趣了。

在你通过这本书学习编程的过程中要记住一点，就是所谓的“万事开头难”，对于有价值的事情尤其如此。也许你是一个害怕失败的人，一碰到困难就想放弃。也许你是一个缺乏自律的人，一碰到“无聊”的事情就不想上手。也许因为有人夸你“有天赋”而让你自视甚高，不愿意做这些看上去很笨拙的事情，怕有负你“神童”的称号。也许你太过激进，把自己跟有 20 多年经验的编程老手相比，让自己失去了信心。

不管是什么原因，你一定要坚持下去。如果你碰到做不出来的加分习题，或者碰到一节看不懂的习题，你可以暂时跳过去，过一阵子回来再看。只要坚持下去，你总会弄懂的。

一开始你可能什么都看不懂。这会让你感觉很不舒服，就像学习人类的自然语言一样。你会发现很难记住一些单词和特殊符号的用法，而且会经常感到很迷茫，直到有一天，忽然一下子你会觉得豁然开朗，以前不明白的东西忽然就明白了。如果你坚持练习下去，坚持去上下求索，你最终会学会这些东西的。也许你不会成为一个编程大师，但你至少会明白程序是怎么工作的。

如果你放弃的话，你会失去达到这个程度的机会。你会在第一次碰到不明白的东西时(几乎是所有的东西)放弃。如果你坚持尝试，坚持写习题，坚持尝试弄懂习题的话，你最终一定会明白里边的内容的。

如果你通读了这本书，却还是不知道编程是怎么回事。那也没关系，至少你尝试过了。你可以说你已经尽过力但成效不佳，但至少你尝试过了。这也是一件值得你骄傲的事情。

给“小聪明”们的警告

有的学过编程的人读到这本书，可能会有一种被侮辱的感觉。其实本书中没有任何要居高临下地贬低任何人的意思。只不过是比我面向的读者群知道的更多而已。如果你觉得自己比我聪明，然后觉得我在居高临下，那我也没办法，因为你根本就不属于我的目的读者群。

如果你觉得这本书里到处都在侮辱你的智商，那我对你有三个建议：

1. 别读这本书了。我不是写给你的，我是写给需要学习的人的。
2. 放下架子好好学。如果你认为你什么都知道，那你就很难从比你强的人身上学到什么了。
3. 学 Lisp 去。我听说什么都知道的人可喜爱 Lisp 了。

对于其他在这里学习的人，你们读的时候就想着我在微笑就可以了，虽然我的眼睛里还带着恶作剧的闪光。

许可协议

Copyright (C) 2010 by Zed A. Shaw. 你可以在不收取任何费用，而且不修改任何内容的前提下自由分发这本书给任何人。但是本书的内容只允许完整原封不动地进行分发和传播。也就是说如果你用这本书给人上课，只要你不向学生收费，而且给他们看的书是完整未加修改的，那就没问题。

特别感谢

首先我要感谢帮助我完成这版书的人。首先是 **Pretty Girl Editing Services** 可爱的编辑所做的编辑工作。然后是 **Greg Newman**，他提供了美工图并帮我设计了封面，而且还帮忙复审了本书。是他让这本书看上去像本真正的书籍，而且就算我没在第一版里提到他的辛劳，他也没跟我计较。我还要感谢 **Brian Shumate** 在网站设计方面的帮助，这方面的帮助也是我非常需要的。

最后，我还要感谢成千上万读过本书第一版而且提出 **bug** 报告和改进建议的读者。你们的贡献让这本书的内容更为扎实，没有你们我是做不到的。谢谢你们。

习题 0: 准备工作

这道习题并没有代码内容，它的主要目的是让你在计算机上安装好 Python。你应该尽量照着说明进行操作，例如 Mac OSX 默认已经安装了 Python 2，所以就不要再在上面安装 Python 3 或者别的 Python 版本了。

Warning

如果你不知道怎样使用 Windows 下的 PowerShell，或者 OSX 下的 Terminal，或者 Linux 下的 “bash”，那你就需要学习了。我有一个免费的快速入门教程放在

<http://cli.learncodethehardway.org/>，你可以快速学到 PowerShell 和 Terminal 的基本用法。学完后再回来看这本书吧。

Mac OSX

你需要做下列任务来完成这个练习：

1. 用浏览器打开 <http://www.barebones.com/products/textwrangler/> 找到并安装 TextWrangler 文本编辑器。
2. 把 TextWrangler (也就是你的编辑器) 放到 Dock 中，以方便日后使用。
3. 找到系统中的 “命令行终端(Terminal)” 程序。到处找找，你会找到的。
4. 把 Terminal 也放到 Dock 里面。
5. 运行 Terminal 程序，这个程序看上去不怎么地。
6. 在 Terminal 程序里边运行 python。运行的方法是输入程序的名字再敲一下回车。
7. 敲击 CTRL-D (^D) 退出 python。
8. 这样你就应该退回到敲 python 前的提示界面了。如果没有的话自己研究一下为什么。
9. 学着使用 Terminal 创建一个目录，你可以上网搜索怎样做。
10. 学着使用 Terminal 进入一个目录，同样你可以上网搜索。
11. 使用你的编辑器在你进入的目录下建立一个文件。你将建立一个文件。使用 “Save” 或者 “Save As...” 选项，然后选择这个目录。
12. 使用键盘切换回到 Terminal 窗口，如果不知道怎样使用键盘切换，你一样可以上网搜索。
13. 回到 Terminal，看看你能不能使用命令看到你新建的文件，上网搜索如何将文件夹中的内容列出来。

OSX: 你应该看到的结果

以下是我在自己电脑的 Terminal 中执行上述练习时看到的内容。和你做的结果会有一些不同，所以看看你能不能找出两者不同点来。

```
Last login: Sat Apr 24 00:56:54 on ttys001
~ $ python
Python 2.5.1 (r251:54863, Feb  6 2009, 19:02:12)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> ^D
~ $ mkdir mystuff
~ $ cd mystuff
mystuff $ ls
# ... 使用 TextWrangler 编辑 test.txt ...
mystuff $ ls
test.txt
mystuff $
```

Windows

Note

感谢 zhmark 的贡献。

1. 用浏览器打开 <http://notepad-plus-plus.org/> 下载并安装 Notepad++ 文本编辑器。这个操作无需管理员权限。
2. 把 Notepad++ 放到桌面或者快速启动栏，这样你就可以方便地访问到该程序了。这两条在安装选项中可以看到。
3. 从开始菜单运行 “PowerShell” 程序。你可以使用开始菜单的搜索功能，输入名称后敲回车即可打开。
4. 为它创建一个快捷方式，放到桌面或者快速启动栏中以方便使用。
5. 运行命令行终端程序(也就是 PowerShell)，这个程序看上去不怎么地。
6. 在命令行终端里边运行 python。运行的方法是输入程序的名字再敲一下回车。
 1. 如果你运行 python 发现它不存在(python 不是可执行命令，或者系统找不到 python 云云)。你需要访问 <http://python.org/download> 并且安装 Python。
 2. 确认你安装的是 **Python 2** 而不是 Python 3。
 3. 你也可以试试 ActiveState Python，尤其是你没有管理员权限的时候。
 4. 如果你安装好了但是 python 还是不能被识别，那你需要在 powershell 下输入并执行以下命令：

```
[Environment]::SetEnvironmentVariable("Path",
"$env:Path;C:\Python27", "User")
```
 5. 关闭并重启 powershell，确认 python 现在可以运行。如果不行的话你可能需要重启电脑。
7. 键入 CTRL-Z (^Z)，再敲回车以退出 python。
8. 这样你就应该退回到敲 python 前的提示界面了。如果没有的话自己研究一下为什么。
9. 学着使用 Terminal 创建一个目录，你可以上网搜索怎样做。
10. 学着使用 Terminal 进入一个目录。同样你可以上网搜索。
11. 使用你的编辑器在你进入的目录下建立一个文件。你将建立一个文件，使用 “Save” 或者 “Save As...” 选项，然后选择这个目录。
12. 使用键盘切换回到 Terminal 窗口，如果不知道怎样使用键盘切换，你一样可以上网搜索。
13. 回到 Terminal，看看你能不能使用命令看到你新建的文件，上网搜索如何将文件夹中的内容列出来。

Warning

有时这一步你会漏掉：Windows 下装了 Python 但是没有正确配置路径。确认你在 powershell 下输入了 `[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27", "User")`。你也许需要重启 powershell 或者计算机来让路径设置生效。

Windows: 你应该看到的结果

```
> python
ActivePython 2.6.5.12 (ActiveState Software Inc.) based on
```

```
Python 2.6.5 (r265:79063, Mar 20 2010, 14:22:52) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ^Z
```

```
> mkdir mystuff
```

```
> cd mystuff
```

```
... 使用 Notepad++ 编辑 mystuff 目录下的 test.txt ...
```

```
>
```

<如果你没有使用管理员权限安装，你会看到一堆错误。忽略它们，按回车即可。>

```
> dir
```

```
Volume in drive C is
```

```
Volume Serial Number is 085C-7E02
```

```
Directory of C:\Documents and Settings\you\mystuff
```

```
04.05.2010  23:32    <DIR>          .
04.05.2010  23:32    <DIR>          ..
04.05.2010  23:32                6 test.txt
               1 File(s)                6 bytes
               2 Dir(s)  14 804 623 360 bytes free
```

```
>
```

你看到的命令行信息，Python 信息，以及其它一些东西可能会非常不一样，不过应该大致不差。你可以通过 <http://learnpythonthehardway.org> 把你找到的错处告诉我们，我们会修正过来。

Linux

Linux 系统可谓五花八门，安装软件的方式也各有不同。我们假设作为 Linux 用户的你已经知道如何安装软件包了，以下是给你的操作说明：

1. 用浏览器打开 <http://learnpythonthehardway.org/exercise0.html> 下载并安装 gedit 文本编辑器。
2. 把 gedit (也就是你的编辑器) 放到窗口管理器显见的位置，以方便日后使用。
 1. 运行 gedit，我们要先改掉一些愚蠢的默认设定。
 2. 从 gedit menu 中打开 Preferences，选择 Editor 页面。
 3. 将 Tab width: 改为 4。
 4. 选择 (确认有勾选到该选项) Insert spaces instead of tabs。
 5. 然后打开 “Automatic indentation” 选项。
 6. 转到 View 页面，打开 “Display line numbers” 选项。
3. 找到 “Terminal” 程序。它的名字可能是 GNOME Terminal、Konsole、或者 xterm。
4. 把 Terminal 也放到 Dock 里面。
5. 运行 Terminal 程序，这个程序看上去不怎么地。
6. 在 Terminal 程序里边运行 python。运行的方法是输入程序的名字再敲一下回车。a. 如果你运行 python 发现它不存在的话，你需要安装它，而且要确认你安装的是 Python 2 而非 Python 3。
7. 敲击 CTRL-D (^D) 以退出 python。
8. 这样你就应该退回到敲 python 前的提示界面了。如果没有的话自己研究一下为什么。

9. 学着使用 **Terminal** 创建一个目录。你可以上网搜索怎样做。
10. 学着使用 **Terminal** 进入一个目录。同样你可以上网搜索。
11. 使用你的编辑器在你进入的目录下建立一个文件。你将建立一个文件，使用 “**Save**” 或者 “**Save As...**” 选项，然后选择这个目录。
12. 使用键盘切换回到 **Terminal** 窗口，如果不知道怎样使用键盘切换，你一样可以上网搜索。
13. 回到 **Terminal**，看看你能不能使用命令看到你新建的文件，上网搜索如何将文件夹中的内容列出来。

Linux: 你应该看到的结果

```
[~]$ python
Python 2.6.5 (r265:79063, Apr  1 2010, 05:28:39)
[GCC 4.4.3 20100316 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
[~]$ mkdir mystuff
[~]$ cd mystuff
# ... 使用 gedit 编辑 text.txt ...
[mystuff]$ ls
test.txt
[mystuff]$
```

你看到的命令行信息，Python 信息，以及其它一些东西可能会非常不一样。不过应该大致不差就是了。

给新手的告诫

你已经完成了这节练习，取决于你对计算机的熟悉程度，这个练习对你而言可能会有些难。如果你觉得有难度的话，你要自己克服困难，多花点时间学习一下。因为如果你不会这些基础操作的话，编程对你来说将会更难学习。

如果有程序员告诉你让你使用 **vim** 或者 **emacs**，那你应该拒绝他们。当你成为一个更好的程序员的时候，这些编辑器才会适合你使用。你现在需要的只是一个可以编辑文本的编辑器。我们使用 **gedit** **TextWrangler** 或者 **Notepad++** 是因为它很简单，而且在不同的系统上面使用起来是一样的。就连专业程序员也会使用 **gedit**，所以对于初学而言它已经足够了。

也许有程序员会告诉你让你安装和学习 **Python 3**。你应该告诉他们“等你电脑里的所有 **python** 代码都支持 **Python 3** 了，我再试着学学吧。”你这句话足够他们忙活个十来年的了。

总有一天你会听到有程序员建议你使用 **Mac OSX** 或者 **Linux**。如果他喜欢字体美观，他会告诉你让你弄台 **Mac OSX** 计算机，如果他们喜欢操作控制而且留了一部大胡子，他会让你安装 **Linux**。这里再次向你说明，只要是一台手上能用的电脑就可以了。你需要的只有三样东西: **gedit**、一个命令行终端、还有 **python**。

最后要说的是这节练习的准备工作目的，也就是让你可以在以后的练习中顺利地做到下面的这些事情：

1. 写出习题的代码，在 **Linux** 下用 **gedit**，**OSX** 下用 **TextWrangler**，**Windows** 下用 **Notepad++**。
2. 运行你写的习题。
3. 修改错误的地方。
4. 重复上述步骤。

其他的事情只会让你更困惑，所以还是坚持按计划进行吧。

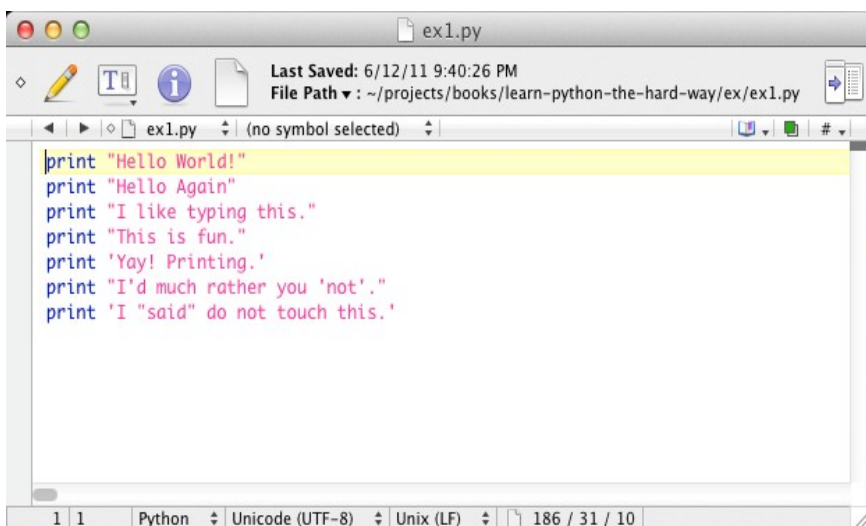
习题 1: 第一个程序

你应该在练习 0 中花了不少的时间，学会了如何安装文本编辑器、运行文本编辑器、以及如何运行命令行终端，而且你已经花时间熟悉了这些工具。请不要跳过前一个练习的内容直接进行下面的内容，这也是本书唯一的一次这样的警示。

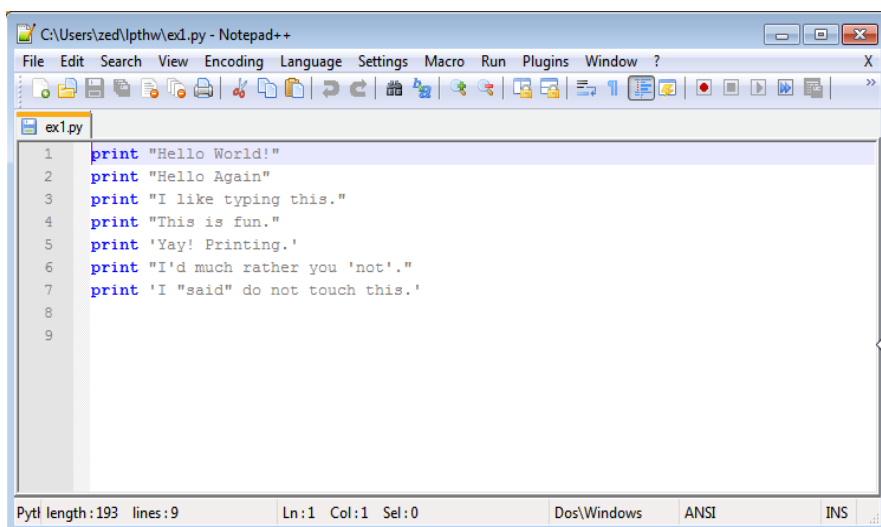
将下面的内容写到一个文件中，取名为 `ex1.py`。这个命名方式很重要，Python 文件最好以 `.py` 结尾。

```
1 print "Hello World!"
2 print "Hello Again"
3 print "I like typing this."
4 print "This is fun."
5 print 'Yay! Printing.'
6 print "I'd much rather you 'not'."
7 print 'I "said" do not touch this.'
```

如果你使用的是 Mac OSX 下的 TextWrangler，那你的文本编辑器大致是这个样子：



如果你在 Windows 下使用 Notepad++，那你看到的应该是这个：



别担心编辑器长得是不是一样，关键是以下几点：

1. 注意我没有输入左边的行号（1-7）。这些是额外打印到书里边的，以方便对代码具体的某一行进行讨论。例如“参见第 5 行……”你无需将这些也写进 python 脚本中去。

2. 注意我截图中开始的 `print` 语句，它和代码范例中是完全一样的，而且是精确的完全相同，不仅仅是表面相似而已。要让这段脚本正常工作，代码中的每个字符都必须完全匹配。当然，显示的颜色可能是不同的，颜色并不重要，只有字符才是重要的。

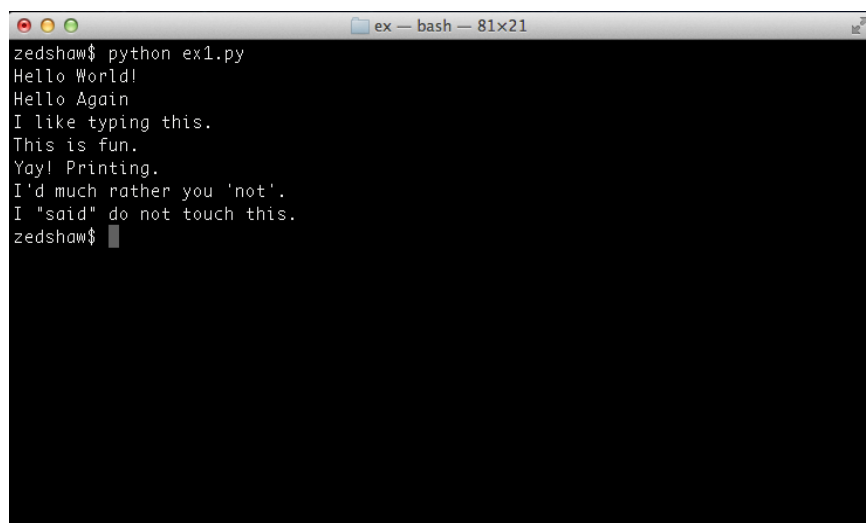
然后你需要在命令行终端通过输入以下内容来运行这段代码：

```
python ex1.py
```

如果你写对了的话，你应该看到和下面一样的内容。如果不一样，那就是你弄错了什么东西。不是计算机出错了，计算机没错。

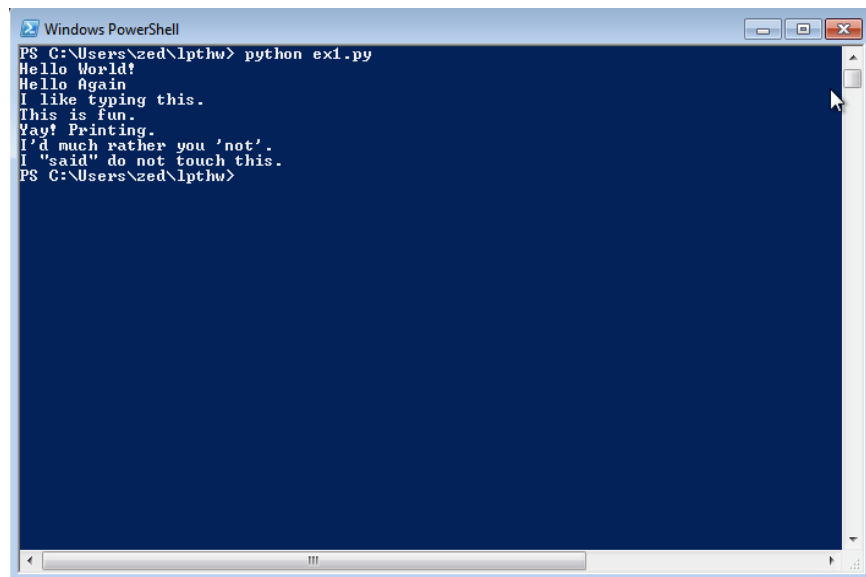
你应该看到的结果

在 Mac OSX 的 Terminal 下面你应该看到以下内容：



```
zedshaw$ python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
zedshaw$
```

在 Windows 的 PowerShell 下你应该看到这些：



```
PS C:\Users\zed\lpthw> python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
PS C:\Users\zed\lpthw>
```

你也许会看到 `python ex1.py` 前面显示了不同的用户名，计算机名，以及其他一些信息，这不是问题，重要的是你输入了命令，而且看到了相同的输出。

如果你看到类似如下的错误信息：

```
1 $ python ex/ex1.py
2   File "ex/ex1.py", line 3
3
```



```
4     print "I like typing this."
5                                     ^
SyntaxError: EOL while scanning string literal
```

这些内容你应该学会看懂的，这是很重要的一点，因为你以后还会犯类似的错误。就是我现在也会犯这样的错误。让我们一行一行来看。

1. 首先我们在命令行终端输入命令来运行 `ex1.py` 脚本。
2. Python 告诉我们 `ex1.py` 文件的第 3 行有一个错误。
3. 然后这一行的内容被打印了出来。
4. 然后 Python 打印出一个 `^` (井号, caret) 符号，用来指示出错的位置。注意到少了一个 `"` (双引号, double-quote) 符号了吗？
5. 最后，它打印出了一个“语法错误(SyntaxError)”告诉你究竟是什么样的错误。通常这些错误信息都非常难懂，不过你可以把错误信息的内容复制到搜索引擎里，然后你就能看到别人也遇到过这样的错误，而且你也许能找到如何解决这个问题。

Warning

如果你来自另外一个国家，而且你看到关于 ASCII 编码的错误，那就在你的 python 脚本的最上面加入这一行：

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

这样你就在脚本中使用了 unicode UTF-8 编码，这些错误就不会出现了。

加分习题

你还会有 加分习题 需要完成。加分习题里边的内容是供你尝试的。如果你觉得做不出来，你可以暂时跳过，过段时间再回来做。

在这个练习中，试试这些东西：

1. 让你的脚本再多打印一行。
2. 让你的脚本只打印一行。
3. 在一行的起始位置放一个 `#` (octothorpe) 符号。它的作用是什么？自己研究一下。

从现在开始，除非特殊情况，我将不再解释每个习题的工作原理了。

Note

井号有很多的英文名字，例如：`'octothorpe(八角帽)'`，`'pound(英镑符)'`，`'hash(电话的#键)'`，`'mesh(网)'` 等。

常见问题回答

我可不可以使用 IDLE？

不行。你应该使用 OSX 的 Terminal 或者 Windows 的 Powershell，和我这里演示的一样。如果你不知道如何使用它们，你可以去读一下《命令行快速入门》，网址是 <http://cli.learncodethehardway.org/book/>

怎样让编辑器显示不同颜色？

编辑之前先将文件保存为 `.py` 格式，例如 `ex1.py`，后面编辑时你就可以看到各种颜色了。

运行 `ex1.py` 时看到 `SyntaxError: invalid syntax`。

你也许已经运行了 `python`，然后又在 `python` 环境下运行了一遍 `python`。关掉并重启命令行终端，重来一遍，只键入 `python ex1.py` 就可以了。

我还是没法再 PowerShell 下运行 `python`。

那就给你个视频教程看看吧：<http://www.youtube.com/watch?v=ndNIFy-5GKA>

错误信息 *can't open file 'ex1.py': [Errno 2] No such file or directory*。

你需要在你创建文件的目录下运行命令。确认你事先使用 *cd* 命令进入了这层目录下。加入你的文件存在 *lpthw/ex1.py* 下面，那你需要先执行 *cd lpthw/* 再运行 *python ex1.py*，如果你不明白命令的意思，那就去看看问题 1 中提到的《命令行快速入门》吧。

怎样在代码中输入我们国家的语言文字？

确认在文件开头输入这行：*# -*- coding: utf-8 -*-*

我的文件无法运行，它直接回到了命令行，没有任何输出。

很有可能是你把代码做了字面理解，认为 *print "Hello World!"* 就是让你在文件中 *print "Hello World!"* 出来，于是你没有输入 *print*。你的代码应该和我的完全一模一样。我的每行里边有 *print*，你的也要确保都有，这样代码才能正常运行。

习题 2: 注释和井号

程序里的注释是很重要的。它们可以用自然语言告诉你某段代码的功能是什么。在你想要临时移除一段代码时，你还可以用注解的方式将这段代码临时禁用。接下来的练习将让你学会注释：

```
1 # A comment, this is so you can read your program later.
2 # Anything after the # is ignored by python.
3
4 print "I could have code like this." # and the comment after is ignored
5
6 # You can also use a comment to "disable" or comment out a piece of code:
7 # print "This won't run."
8
9 print "This will run."
```

从现在开始，我将用这样的方式来写代码。我一直在强调“完全相同”，不过你也不必按照字面意思理解。你的程序在屏幕上的显示可能会有些不同，不过重要的是你在文本编辑器中输入的文本的正确性。事实上，我可以用任何编辑器写出这段程序，而且内容是完全一样的。

你应该看到的结果

```
$ python ex2.py
I could have code like this.
This will run.
$
```

再次说明，我不会再贴各种屏幕截图了。你应该明白上面的内容是输出内容的字面翻译，而 `$ python ...` 和最后的 `$` 之间才是你应该关心的内容。

加分习题

1. 弄清楚”#”符号的作用。而且记住它的名字。(中文为井号，英文为 `octothorpe` 或者 `pound character`)。
2. 打开你的 `ex2.py` 文件，从后往前逐行检查。从最后一行开始，倒着逐个单词单词检查回去。
3. 有没有发现什么错误呢？有的话就改正过来。
4. 朗读你写的习题，把每个字符都读出来。有没有发现更多的错误呢？有的话也一样改正过来。

常见问题回答

你确定 # 符号的名称是 `pound character`？

我叫它 `octothorpe`，这个名字没有哪个国家在用，不过所有的人都能看懂它的意思。每个国家都觉得他们的叫法最正确最闪亮。对我来说这是自大狂的想法，而且你也没必要去关心这种细枝末节，学习编程才是更重要的事情。

如果 # 是注解的意思，那么为什么 # `-*- coding: utf-8 -*-` 能起作用呢？

Python 其实还是没把这行当做代码处理，这种用法只是让字符格式被识别的一个取巧的方案，或者说是一个没办法的办法吧。在编辑器设置里你还能看到一个类似的注解。

为什么 `print "Hi # there."` 里的 # 没被忽略掉？

这行代码里的 # 处于字符串内部，所以它就是引号结束前的字符串中的一部分，这时它只是一个普通字符，而不代表注解的意思。

怎样做多行注解？

每行前面放一个 # 就可以了。

我们国家的键盘上找不到 # 字符，怎么办？

有的国家要通过 **Alt** 键组合才能输入这个字符。你可以用搜索引擎找一下解决方案。

为什么要让我倒着阅读代码？

这样可以避免让你的大脑跟着每一段代码内容的意思走，这样可以让你精确处理每个片段，从而让你更容易地发现代码中的错误。这是一个很好使的查错技巧。

习题 3: 数字和数学计算

每一种编程语言都包含处理数字和进行数学计算的方法。不必担心，程序员经常撒谎说他们是多么牛的数学天才，其实他们根本不是。如果他们真是数学天才，他们早就去从事数学相关的行业了，而不是写广告程序和社交网络游戏，从人们身上偷赚点小钱而已。

这章练习里有很多的数学运算符号。我们来看一遍它们都叫什么名字。你要一边写一边念出它们的名字来，直到你念烦了为止。名字如下：

- + plus 加号
- - minus 减号
- / slash 斜杠
- * asterisk 星号
- % percent 百分号
- < less-than 小于号
- > greater-than 大于号
- <= less-than-equal 小于等于号
- >= greater-than-equal 大于等于号

有没有注意到以上只是些符号，没有运算操作呢？写完下面的练习代码后，再回到上面的列表，写出每个符号的作用。例如 + 是用来做加法运算的。

```
1
2 print "I will now count my chickens:"
3
4 print "Hens", 25 + 30 / 6
5 print "Roosters", 100 - 25 * 3 % 4
6
7 print "Now I will count the eggs:"
8
9 print 3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6
10
11 print "Is it true that 3 + 2 < 5 - 7?"
12
13 print "What is 3 + 2?", 3 + 2
14 print "What is 5 - 7?", 5 - 7
15
16 print "Oh, that's why it's False."
17
18 print "How about some more."
19
20 print "Is it greater?", 5 > -2
21 print "Is it greater or equal?", 5 >= -2
22 print "Is it less or equal?", 5 <= -2
23
```

你应该看到的结果

```
$ python ex3.py
I will now count my chickens:
Hens 30
Roosters 97
Now I will count the eggs:
7
```

```
Is it true that 3 + 2 < 5 - 7?
False
What is 3 + 2? 5
What is 5 - 7? -2
Oh, that's why it's False.
How about some more.
Is it greater? True
Is it greater or equal? True
Is it less or equal? False
$
```

加分习题

1. 使用 # 在代码每一行的前一行为自己写一个注解，说明一下这一行的作用。
2. 记得开始时的 <练习 0> 吧？用里边的方法把 Python 运行起来，然后使用刚才学到的运算符号，把 Python 当做计算器玩玩。
3. 自己找个想要计算的东西，写一个 .py 文件把它计算出来。
4. 有没有发现计算结果是“错”的呢？计算结果只有整数，没有小数部分。研究一下这是为什么，搜索一下“浮点数(floating point number)”是什么东西。
5. 使用浮点数重写一遍 ex3.py，让它的计算结果更准确(提示: 20.0 是一个浮点数)。

常见问题回答

为什么 % 是求余数符号，而不是百分号？

很大程度上只是因为涉及人员选择了这个符号而已。一般而言它是百分号没错，就跟 100% 表示百分之百一样。在编程中除法我们用了 /，而求余数又恰恰选择了 % 这个符号，仅此而已。

% 是怎么工作的？

换个说法就是“X 除以 Y 还剩余 J”，例如“100 除以 16 还剩 4”。% 运算的结果就是 J 这部分。

运算优先级是什么样子的？

美国我们用 PEMDAS 这个简称来辅助记忆，它的意思是“括号、指数、乘、除、加、减”——Parentheses Exponents Multiplication Division Addition Subtraction ——这也是 Python 里的运算优先级。

为什么 / 除法算出来的比实际小？

其实不是没算对，而是它将小数部分丢弃了，试试 $7.0/4.0$ 和 $7/4$ 比较一下，你就看出不同了。

习题 4: 变量(variable)和命名

你已经学会了 `print` 和算术运算。下一步你要学的是“变量”。在编程中，变量只不过是用来指代某个东西的名字。程序员通过使用变量名可以让他们的程序读起来更像英语。而且因为程序员的记性都不怎么地，变量名可以让他们更容易记住程序的内容。如果他们没有在写程序时使用好的变量名，在下次读到原来写的代码时他们会大为头疼的。

如果你被这章习题难住了的话，记得我们之前教过的：找到不同点、注意细节。

1. 在每一行的上面写一行注解，给自己解释一下这一行的作用。
2. 倒着读你的 `.py` 文件。
3. 朗读你的 `.py` 文件，将每个字符也朗读出来。

```
1
2 cars = 100
3 space_in_a_car = 4.0
4 drivers = 30
5 passengers = 90
6 cars_not_driven = cars - drivers
7 cars_driven = drivers
8 carpool_capacity = cars_driven * space_in_a_car
9 average_passengers_per_car = passengers / cars_driven
10
11 print "There are", cars, "cars available."
12 print "There are only", drivers, "drivers available."
13 print "There will be", cars_not_driven, "empty cars today."
14 print "We can transport", carpool_capacity, "people today."
15 print "We have", passengers, "to carpool today."
16 print "We need to put about", average_passengers_per_car, "in each car."
```

Note

`space_in_a_car` 中的 `_` 是下划线(underscore) 字符。你要自己学会怎样打出这个字符来。这个符号在变量里通常被用作假想的空格，用来隔开单词。

你应该看到的结果

```
$ python ex4.py
There are 100 cars available.
There are only 30 drivers available.
There will be 70 empty cars today.
We can transport 120.0 people today.
We have 90 to carpool today.
We need to put about 3 in each car.
$
```

加分习题

当我刚开始写这个程序时我犯了个错误，`python` 告诉我这样的错误信息：

```
Traceback (most recent call last):
  File "ex4.py", line 8, in <module>
    average_passengers_per_car = car_pool_capacity / passenger
NameError: name 'car_pool_capacity' is not defined
```

用你自己的话解释一下这个错误信息，解释时记得使用行号，而且要说明原因。

更多的加分习题：

1. 我在程序里用了 4.0 作为 `space_in_a_car` 的值，这样做有必要吗？如果只用 4 会有什么问题？
2. 记住 4.0 是一个“浮点数”，自己研究一下这是什么意思。
3. 在每一个变量赋值的上一行加上一行注解。
4. 记住 `=` 的名字是等于(equal)，它的作用是为东西取名。
5. 记住 `_` 是下划线字符(underscore)。
6. 将 `python` 作为计算器运行起来，就跟以前一样，不过这一次在计算过程中使用变量名来做计算，常见的变量名有 `i`, `x`, `j` 等等。

常见问题回答

`=` 和 `==` 有什么不同？

`=` (single-equal) 的作用是将右边的值赋予左边的变量名。`==` (double-equal) 的作用是检查左右离岸边是否相等。习题 27 中你会学到 `==` 的用法。

写成 `x=100` 而非 `x = 100` 也没关系吧？

是可以这样写，但这种写法不好。操作符两边加上空格会让代码更容易阅读。

`print` 时词语间的空格有没有办法不让打印出来？

你可以通过这样的方法实现：`print "Hey %s there." % "you"`，后面马上就会讲到。

怎样倒着读代码？

很简单，假如说你的代码有 16 行，你就从第 16 行开始，和我的第 16 行比对，接着比对第 15 行，以此类推，直到全部检查完。

为什么 `space` 用了 `4.0`？

这个主要就是为了让你见识一下浮点数，并且提出这个问题。看看加分习题吧。

习题 5: 更多的变量和打印

我们现在要键入更多的变量并且把它们打印出来。这次我们将使用一个叫“格式化字符串(format string)”的东西。每一次你使用 `"` 把一些文本引用起来，你就建立了一个字符串。字符串是程序将信息展示给人的方式。你可以打印它们，可以将它们写入文件，还可以将它们发送给网站服务器，很多事情都是通过字符串交流实现的。

字符串是非常好用的东西，所以再这个练习中你将学会如何创建包含变量内容的字符串。使用专门的格式和语法把变量的内容放到字符串里，相当于来告诉 `python`：“嘿，这是一个格式化字符串，把这些变量放到那几个位置。”

一样的，即使你读不懂这些内容，只要一字不差地键入就可以了。

```
1 my_name = 'Zed A. Shaw'
2 my_age = 35 # not a lie
3 my_height = 74 # inches
4 my_weight = 180 # lbs
5 my_eyes = 'Blue'
6 my_teeth = 'White'
7 my_hair = 'Brown'

8 print "Let's talk about %s." % my_name
9 print "He's %d inches tall." % my_height
10 print "He's %d pounds heavy." % my_weight
11 print "Actually that's not too heavy."
12 print "He's got %s eyes and %s hair." % (my_eyes, my_hair)
13 print "His teeth are usually %s depending on the coffee." % my_teeth
14
15 # this line is tricky, try to get it exactly right
16 print "If I add %d, %d, and %d I get %d." % (
17     my_age, my_height, my_weight, my_age + my_height + my_weight)
18
```

Warning

如果你使用了非 ASCII 字符而且碰到了编码错误，记得在最顶端加一行 `# -*- coding: utf-8 -*-`。

你应该看到的结果

```
$ python ex5.py
Let's talk about Zed A. Shaw.
He's 74 inches tall.
He's 180 pounds heavy.
Actually that's not too heavy.
He's got Blue eyes and Brown hair.
His teeth are usually White depending on the coffee.
If I add 35, 74, and 180 I get 289.
$
```

加分习题

1. 修改所有的变量名字，把它们前面的 `my_` 去掉。确认将每一个地方的都改掉，不只是你使用 `=` 赋值过的地方。
2. 试着使用更多的格式化字符。例如 `%r` 就是非常有用，它的含义是“不管什么都打印出

- 来”。
3. 在网上搜索所有的 Python 格式化字符。
 4. 试着使用变量将英寸和磅转换成厘米和千克。不要直接键入答案。使用 Python 的计算功能来完成。

常见问题回答

这样定义变量行不行: `l = 'Zed Shaw'`?

不行。`l` 不是一个有效的变量名称。变量名要以字母开头。所以 `al` 可以，但 `l` 不行。

`%s`, `%r`, `%d` 这些符号是啥意思?

后面你会详细学到更多，现在可以告诉你的是它们是一种“格式控制工具”。它们告诉 Python 把右边的变量带到字符串中，并且把变量值放到 `%s` 所在的位置上。

还是不懂，“格式控制工具”是啥?

要明白一些描述的意义，你得先学会编程才更容易理解，你可以把这样的问题记录下来，看后面的内容会不会向你解释这些东西。

如何将浮点数四舍五入?

你可以使用 `round()` 函数，例如: `round(1.7333)`

我碰到了错误: `TypeError: 'str' object is not callable`。

很有可能你是漏写了字符串和变量之间的 `%`。

这都是些什么玩意啊? 我还是很糊涂。

试着将脚本里的数字看作是你自己量出来的东西，这样会很奇怪，但是多少会让你有身临其境的感觉，从而帮助你理解一些东西。

习题 6: 字符串(string)和文本

虽然你已经在程序中写过字符串了，你还没学过它们的用处。在这章习题中我们将使用复杂的字符串来建立一系列的变量，从中你将学到它们的用途。首先我们解释一下字符串是什么东西。

字符串通常是指你想要展示给别人的、或者是你想要从程序里“导出”的一小段字符。Python 可以通过文本里的双引号 " 或者单引号 ' 识别出字符串来。这在你以前的 print 练习中你已经见过很多次了。如果你把单引号或者双引号括起来的文本放到 print 后面，它们就会被 python 打印出来。

字符串可以包含格式化字符 %s，这个你之前也见过的。你只要将格式化的变量放到字符串中，再紧跟着一个百分号 % (percent)，再紧跟着变量名即可。唯一要注意的地方，是如果你想要在字符串中通过格式化字符放入多个变量的时候，你需要将变量放到 () 圆括号(parenthesis)中，而且变量之间用 , 逗号 (comma)隔开。就像你逛商店说“我要买牛奶、鸡蛋、面包、清汤”一样，只不过程序员的语法是”(milk, eggs, bread, soup)”。

我们将键入大量的字符串、变量、和格式化字符，并且将它们打印出来。我们还将练习使用简写的变量名。程序员喜欢使用恼人的难度的简写来节约打字时间，所以我们现在就提早学会这个，这样你就能读懂并且写出这些东西了。

```
1
2 x = "There are %d types of people." % 10
3 binary = "binary"
4 do_not = "don't"
5 y = "Those who know %s and those who %s." % (binary, do_not)
6
7 print x
8 print y
9
10 print "I said: %r." % x
11 print "I also said: '%s'." % y
12
13 hilarious = False
14 joke_evaluation = "Isn't that joke so funny?! %r"
15
16 print joke_evaluation % hilarious
17
18 w = "This is the left side of..."
19 e = "a string with a right side."
20
21 print w + e
```

你应该看到的结果

```
1 $ python ex6.py
2 There are 10 types of people.
3 Those who know binary and those who don't.
4 I said: 'There are 10 types of people.'.
5 I also said: 'Those who know binary and those who don't.'.
6 Isn't that joke so funny?! False
7 This is the left side of...a string with a right side.
8 $
```

加分习题

1. 通读程序，在每一行的上面写一行注解，给自己解释一下这一行的作用。
2. 找到所有的”字符串包含字符串”的位置，总共有四个位置。
3. 你确定只有四个位置吗？你怎么知道的？没准我在骗你呢。
4. 解释一下为什么 `w` 和 `e` 用 `+` 连起来就可以生成一个更长的字符串。

常见问题回答

`%r` 和 `%s` 有什么不同？

`%r` 用来做 `debug` 比较好，因为它会显示变量的原始数据（raw data），而其它的符号则是用来向用户显示输出的。

既然有 `%r` 了，为什么还要用 `%s` 和 `%d`？

`%r` 用来 `debug` 最好，而其它格式符则是用来向用户显示输出的。

如果你觉得很好笑，可不可以写一句 `hilarious = True`？

可以。在习题 27 中你会学到关于布尔函数的更多知识。

为什么你在有些字符串上用了 `'`（单引号）而在别的上没有用？

很大程度上只是个风格问题，我的风格就是在双引号的字符串中使用单引号。看看第 10 行。`g` `that`.

错误 *`TypeError: not all arguments converted during string formatting`*。

确定每一行代码都完全正确。这里是因为你的字符串里的 `%` 格式化字符数量比后面给的变量多，仔细检查一下哪里写错了。

习题 7: 更多打印

现在我们将做一批练习，在练习的过程中你需要键入代码，并且让它们运行起来。我不会解释太多，因为这节的内容都是以前熟悉过的。这节练习的目的是巩固你学到的东西。我们几个练习后再见。不要跳过这些习题。不要复制粘贴！

```
1
2 print "Mary had a little lamb."
3 print "Its fleece was white as %s." % 'snow'
4 print "And everywhere that Mary went."
5
6 end1 = "C"
7 end2 = "h"
8 end3 = "e"
9 end4 = "e"
10 end5 = "s"
11 end6 = "e"
12 end7 = "B"
13 end8 = "u"
14 end9 = "r"
15 end10 = "g"
16 end11 = "e"
17 end12 = "r"
18 # watch that comma at the end. try removing it to see what happens
19 print end1 + end2 + end3 + end4 + end5 + end6,
20 print end7 + end8 + end9 + end10 + end11 + end12
21
```

你应该看到的结果

```
$ python ex7.py
Mary had a little lamb.
Its fleece was white as snow.
And everywhere that Mary went.
.....
Cheese Burger
$
```

加分习题

接下来几节的加分习题是一样的。

1. 逆向阅读，在每一行的上面加一行注解。
2. 倒着朗读出来，找出自己的错误。
3. 从现在开始，把你的错误记录下来，写在一张纸上。
4. 在开始下一节习题时，阅读一遍你记录下来的错误，并且尽量避免在下个练习中再犯同样的错误。
5. 记住，每个人都会犯错误。程序员和魔术师一样，他们希望大家认为他们从不犯错，不过这只是表象而已，他们每时每刻都在犯错。

常见问题回答

“end”语句是什么原理？

没有什么 `end` 语句，只是变量名里带了个 `end` 而已。

为什么要用一个叫 `'snow'` 的变量？

其实不是变量，而是一个带 `snow` 的字符串而已。变量时不会带引号的。

你在加分习题 1 里说在每行代码上面写注解，一定要这样做吗？

不是。一般情况下加注解只是为了解释难懂的代码，或者注明为什么代码要这么写。一般来说后者更为重要。碰到特殊情况你的代码的确每一行都很难懂的话，加注解也是正确的选择。在这里，我主要是为了让你逐渐学会将代码翻译成日常语言

创建字符串时是不是单引号和双引号都可以，它们有什么不同用途吗？

`Python` 里边两种都是可以的，不过一般单引号会被用来创建简短的字符串，例如 `'a'`、`'snow'` 这样的。

不可以用逗号，将最后两行写成一行输出吗？

当然可以，不过这样以来这行的长度就超过 80 个字符了，这样做不是好的 `Python` 代码风格。

习题 8: 打印, 打印

```
1 formatter = "%r %r %r %r"
2
3 print formatter % (1, 2, 3, 4)
4 print formatter % ("one", "two", "three", "four")
5 print formatter % (True, False, False, True)
6 print formatter % (formatter, formatter, formatter, formatter)
7
8 "I had this thing.",
9 "That you could type up right.",
10 "But it didn't sing.",
11 "So I said goodnight."
12 )
```

你应该看到的结果

```
$ python ex8.py
1 2 3 4
'one' 'two' 'three' 'four'
True False False True
'%r %r %r %r' '%r %r %r %r' '%r %r %r %r' '%r %r %r %r'
'I had this thing.' 'That you could type up right.' "But it didn't sing." 'So I
said goodnight.'
```

加分习题

1. 自己检查结果, 记录你犯过的错误, 并且在下个练习中尽量不犯同样的错误。
2. 注意最后一行程序中既有单引号又有双引号, 你觉得它是如何工作的?

常见问题回答

我应该使用 `%s` 还是 `%r`?

你应该使用 `%s`, 只有在想要获取某些东西的 `debug` 信息时才能用到 `%r`。 `%r` 给你的是变量的“程序员原始版本”, 又被称作“representation”。

为什么 “one” 要用引号, 而 `True` 和 `False` 不需要?

因为 `True` 和 `False` 是 Python 的关键字, 用来表示真假的含义。如果你加了引号, 它们就变成了字符串, 也就无法实现它们本来的功能了。习题 27 中会有详细说明。

我在字符串中包含了中文 (或者其它非 ASCII 字符), 可是 `%r` 打印出的是乱码?

使用 `%s` 就行了。

为什么 `%r` 有时打印出来的是单引号, 而我实际用的是双引号?

Python 会用最有效的方式打印出字符串, 而不是完全按照你写的方式来打印。这样做对于 `%r` 来说是接受的, 因为它是用作 `debug` 和排错, 没必要非打印出多好看的格式。

为什么 Python 3 里这些都不灵?

别使用 Python 3 系列。使用 Python 2.7 或更新的版本, 虽然 Python 2.6 应该也没问题。

可不可以使用 IDLE 运行代码?

不行。你应该学习使用命令行。命令行对学习编程很重要, 而且是一个学习编程的绝佳初始环境。IDLE 在本书后面的章节里会让你失望的。

习题 9: 打印，打印，打印

```
1
2 # Here's some new strange stuff, remember type it exactly.
3 days = "Mon Tue Wed Thu Fri Sat Sun"
4 months = "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"
5
6 print "Here are the days: ", days
7 print "Here are the months: ", months
8
9 print """
10 There's something going on here.
11 With the three double-quotes.
12 We'll be able to type as much as we like.
13 Even 4 lines if we want, or 5, or 6.
14 """
```

你应该看到的结果

```
$ python ex9.py
Here are the days:  Mon Tue Wed Thu Fri Sat Sun
Here are the months:  Jan
Feb
Mar
Apr
May
Jun
Jul
Aug

There's something going on here.
With the three double-quotes.
We'll be able to type as much as we like.
Even 4 lines if we want, or 5, or 6.

$
```

加分习题

1. 自己检查结果，记录你犯过的错误，并且在下个练习中尽量不犯同样的错误。

常见问题回答

怎样将月份显示在新的一行？

字符串以 `\n` 开始就可以了，像这样：

```
"\nJan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"
```

为什么使用 `%r` 时 `\n` 新行就不灵了？

`%r` 就是这个样子，它打印出的是你写出来的方式（或者近似方式）。它是用来 `debug` 的原始格式。

为什么在三引号之间加入空格就会出错？

你必须写成 `"""` 而不是 `" " "`，引号之间不能有空格。

为什么你打印时用了 `+` 而不是逗号？

因为我的目的是将两个字符串连接起来，组建成一个新的字符串。后面你会学到，`print` 里的逗号其实是分隔参数的一种方式。

我的大部分错误都是拼写错误，是不是我太笨了？

对于初学者甚至进阶学员来说，大部分编程中的错误都是拼写错误，或者别的一些简单错误。

习题 10: 那是什么？

在习题 9 中我让你接触了一些新东西。我让你看到两种让字符串扩展到多行的方法。第一种方法是在月份之间用 `\n` (back-slash n) 隔开。这两个字符的作用是在该位置上放入一个“新行(new line)”字符。

使用反斜杠 `\` (back-slash) 可以将难打印出来的字符放到字符串。针对不同的符号有很多这样的所谓“转义序列(escape sequences)”，但有一个特殊的转义序列，就是 双反斜杠 (double back-slash) `\\`。这两个字符组合会打印出一个反斜杠来。接下来我们做几个练习，然后你就知道这些转义序列的意义了。

另外一种重要的转义序列是用来将单引号 `'` 和双引号 `"` 转义。想象你有一个用双引号引用起来的字符串，你想要在字符串的内容里再添加一组双引号进去，比如你想说 `"I "understand" joe."`，Python 就会认为 `"understand"` 前后的两个引号是字符串的边界，从而把字符串弄错。你需要一种方法告诉 python 字符串里边的双引号不是真正的双引号。

要解决这个问题，你需要将双引号和单引号转义，让 Python 将引号也包含到字符串里边去。这里有一个例子：

```
"I am 6'2\" tall." # 将字符串中的双引号转义
'I am 6\'2" tall.' # 将字符串中的单引号转义
```

第二种方法是使用“三引号(triple-quotes)”，也就是 `"""`，你可以在一组三引号之间放入任意多行的文字。接下来你将看到用法。

```
1
2 tabby_cat = "\tI'm tabbed in."
3 persian_cat = "I'm split\non a line."
4 backslash_cat = "I'm \\ a \\ cat."
5
6 fat_cat = """
7 I'll do a list:
8 \t* Cat food
9 \t* Fishies
10 \t* Catnip\n\t* Grass
11 """
12
13 print tabby_cat
14 print persian_cat
15 print backslash_cat
16 print fat_cat
```

你应该看到的结果

注意你打印出来的制表符，这节练习中的文字间隔对于答案的正确性是很重要的。

```
$ python ex10.py
    I'm tabbed in.
I'm split
on a line.
I'm \ a \ cat.

I'll do a list:
    * Cat food
    * Fishies
    * Catnip
```

转义序列

下面列出了 Python 支持的所有转义序列。很多你也许不会用到，不过还是要记住它们的格式和功能。试着在字符串中应用它们，看看它们的功能。

转义符	功能
\\	Backslash () 反斜杠
\'	Single quote (') 单引号
\"	Double quote (") 双引号
\a	ASCII Bell (BEL) 响铃符
\b	ASCII Backspace (BS) 退格符
\f	ASCII Formfeed (FF) 进纸符
\n	ASCII Linefeed (LF) 换行符
\N{name}	Unicode 数据库中的字符名, 其中 name 就是它的名字 (Unicode only)
\r ASCII	Carriage Return (CR) 回车符
\t ASCII	Horizontal Tab (TAB) 水平制表符
\uxxxx	值为 16 位十六进制值 xxxx 的字符 (Unicode only)
\Uxxxxxxxx	值为 32 位十六进制值 xxxx 的字符 (Unicode

转义符	功能
	only)
\v	ASCII Vertical Tab (VT) 垂直制表符
\ooo	值为八进制值 ooo 的字符
\xhh	值为十六进制数 hh 的字符

试着运行下面一段代码看看结果：

```
while True:
    for i in ["/", "-", "|", "\\", "|"]:
        print "%s\r" % i,
```

加分习题

1. 把这些转义字符记录到卡片上，并记住它们的含义。
2. 使用 `'''` (三个单引号)取代三个双引号，看看效果是不是一样的？
3. 将转义序列和格式化字符串组合到一起，创建一种更复杂的格式。
4. 记得 `%r` 格式化字符串吗？使用 `%r` 搭配单引号和双引号转义字符打印一些字符串出来。将 `%r` 和 `%s` 比较一下。注意到了吗？`%r` 打印出来的是你作为程序员写在脚本里的东西，而 `%s` 打印的是你作为用户应该看到的东西。

常见问题回答

我还没完全搞明白上一习题，我可以继续吗？

可以，继续向下看，看完一部分后回头看自己以前在笔记本上记下来的不懂的知识点，看是不是已经明白了。有时你可能还需要回到前面的练习中重新复习一遍。

`\\` 和别符号相比的有什么特别之处吗？

并无特别，这样只是为了输出一个反斜杠 (`\`)，想想为什么要把它写成两杠。

`//` 和 `/n` 怎么不灵？

因为你用了斜杠 `/` 而不是反斜杠 `\`，它们是不一样的字符，功能也完全不同。

使用了 `%r` 后转义序列都不灵了。

因为 `%r` 打印出的是你写到代码里的原始字符串，其中会包含原始的转义字符。你应该使用 `%s`，记住这条：```%r``` 用作 `debug`，```%s``` 用作显示。

加分习题 #3 说是要组合什么的，是什么意思？

我想让你明白的一点是，所有这些习题中教你的东西都可以组合起来帮你解决问题。把你学过的格式化字符串的知识和你新学到的转义字符的只是组合起来，写一些代码。

`'''` 和 `"""` 哪个好？

风格问题。现在你就用 `'''` 吧，以后碰到再说。有时候用某一种可能会更美观，有时候你要遵循之前的写法从而让整个项目代码风格一致，看具体情况吧。

习题 11: 提问

我已经出过很多打印相关的练习，让你习惯写简单的东西，但简单的东西都有点无聊，现在该跟上脚步了。我们现在要做的是把数据读到你的程序里边去。这可能对你有点难度，你可能一下子不明白，不过你需要相信我，无论如何把习题做了再说。只要做几个练习你就明白了。

一般软件做的事情主要就是下面几条：

1. 接受人的输入。
2. 改变输入。
3. 打印出改变了的输入。

到目前为止你只做了打印，但还不会接受或者修改人的输入。你也许还不知道“输入(input)”是什么意思。所以闲话少说，我们还是开始做点练习看你能不能明白。下一个习题里边我们会给你更多的解释。

```
1 print "How old are you?",
2 age = raw_input()
3 print "How tall are you?",
4 height = raw_input()
5 print "How much do you weigh?",
6 weight = raw_input()
7
8 print "So, you're %r old, %r tall and %r heavy." % (
9     age, height, weight)
```

Note

注意到我在每行 `print` 后面加了个逗号(`comma`)，了吧？这样的话 `print` 就不会输出新行符而结束这一行跑到下一行去了。

你应该看到的结果

```
$ python ex11.py
How old are you? 35
How tall are you? 6'2"
How much do you weigh? 180lbs
So, you're '35' old, '6\'2"' tall and '180lbs' heavy.
$
```

加分习题

1. 上网查一下 Python 的 `raw_input` 实现的是什么功能。
2. 你能找到它的别的用法吗？测试一下你上网搜索到的例子。
3. 用类似的格式再写一段，把问题改成你自己的问题。
4. 和转义序列有关的，想想为什么最后一行 `'6\'2\"'` 里边有一个 `\'` 序列。单引号需要被转义，从而防止它被识别为字符串的结尾。有没有注意到这一点？

常见问题回答

如何读取用户输入的数字进行计算？

这个有点算高级话题了。试试 `x = int(raw_input())`，他会把用户输入的字符串用 `int()` 转换成整数。

我把身高写到 `raw input` 里 `raw_input("6'2")` 怎么不灵？

不应该写成这样，只有从命令行输入才可以。首先回去把代码写成和我的一模一样，然后运行脚本，当脚本暂停下来的时候，用键盘输入你的身高。这样做就可以了。

为什么第 8 行要新写一行而不是放在一整行里边？

这样是为了保持每行不多于 80 个字符，Python 程序员喜欢这样的风格。放在一整行里边也不行。

`input()` 和 `raw_input()` 有何不同？

`input()` 函数会把你输入的东西当做 Python 代码进行处理，这么做会有安全问题，你应该避开这个函数。

打印出来后我的字符串前面有个 `u`，像 `u'35'` 这样。

它表示 Python 告诉你你的字符串是 **unicode**。使用 `%s` 就一切正常了。

习题 12: 提示别人

当你键入 `raw_input()` 的时候，你需要键入 (和) 也就是“括号(parenthesis)”。这和你格式化输出两个以上变量时的情况有点类似，比如说 `"%s %s" % (x, y)` 里边就有括号。对于 `raw_input` 而言，你还可以让它显示出一个提示，从而告诉别人应该输入什么东西。你可以在 () 之间放入一个你想要作为提示的字符串，如下所示：

```
y = raw_input("Name? ")
```

这句话会用 “Name?” 提示用户，然后将用户输入的结果赋值给变量 `y`。这就是我们提问用户并且得到答案的方式。

也就是说，我们的上一个练习可以使用 `raw_input` 重写一次。所有的提示都可以通过 `raw_input` 实现。

```
1 age = raw_input("How old are you? ")
2 height = raw_input("How tall are you? ")
3 weight = raw_input("How much do you weigh? ")
4
5 print "So, you're %r old, %r tall and %r heavy." % (
6     age, height, weight)
```

你应该看到的结果

```
$ python ex12.py
How old are you? 35
How tall are you? 6'2"
How much do you weigh? 180lbs
So, you're '35' old, '6\'2"' tall and '180lbs' heavy.
$
```

加分习题

1. 在命令行界面下运行你的程序，然后在命令行输入 `pydoc raw_input` 看它说了些什么。如果你用的是 **Window**，那就试一下 `python -m pydoc raw_input`。
2. 输入 `q` 退出 `pydoc`。
3. 上网找一下 `pydoc` 命令是用来做什么的。
4. 使用 `pydoc` 再看一下 `open`, `file`, `os`, 和 `sys` 的含义。看不懂没关系，只要通读一下，记下你觉得有意思的点就行了。

常见问题回答

运行 `pydoc` 时显示 `SyntaxError: invalid syntax`。

你没有从命令行运行 `pydoc`，很可能是从 `python` 里边运行的。退出 `python` 试试。

我的 `pydoc` 为什么不会暂停？

有时文档很短，一页屏幕就显示完了，这时 `pydoc` 就不会暂停。

运行 `pydoc` 是看到 `more is not recognized as an internal`。

有的版本 **Windows** 中没有这个命令，也就是说你没法用 `pydoc` 了。跳过这些加分习题，上网去搜索 **Python** 文档吧。

`%r` 和 `%s` 该用哪个？

记住 `%r` 是 **debug** 专用，它显示的是原始表示出来的字符，而 `%s` 是为了显示给用户。这个问题以后我就不再回答了，你要牢牢记住。这个问题是人们重复问的最多的问题，如果同一个问题要问很多遍，那说明你没记住你该记住的东西。别问了，现在我要求你必须记住。

写成 `print "How old are you?" , raw_input()` 为什么不行？

你觉得可以，但 **Python** 不这么认为。我唯一能给你的答案是：这样就是不行。

习题 13: 参数、解包、变量

在这节练习中，我们将降到另外一种将变量传递给脚本的方法(所谓脚本，就是你写的 .py 程序)。你已经知道，如果要运行 ex13.py，只要在命令行运行 `python ex13.py` 就可以了。这句命令中的 `ex13.py` 部分就是所谓的“参数(argument)”，我们现在要做的就是写一个可以接受参数的脚本。

将下面的程序写下来，后面你将看到详细解释。

```
1 from sys import argv
2
3 script, first, second, third = argv
4
5 print "The script is called:", script
6 print "Your first variable is:", first
7 print "Your second variable is:", second
8 print "Your third variable is:", third
```

在第 1 行我们有一个“import”语句。这是你将 python 的功能引入你的脚本的方法。Python 不会一下子将它所有的功能给你，而是让你需要什么就调用什么。这样可以让你的程序保持精简，而后面的程序员看到你的代码的时候，这些“import”可以作为提示，让他们明白你的代码用到了哪些功能。

argv 是所谓的“参数变量(argument variable)”，是一个非常标准的编程术语。在其他的编程语言里你也可以看到它。这个变量包含了你传递给 Python 的参数。通过后面的练习你将对它有更多的了解。

第 3 行将 argv “解包(unpack)”，与其将所有参数放到同一个变量下面，我们将每个参数赋予一个变量名： `script`, `first`, `second`, 以及 `third`。这也许看上去有些奇怪，不过“解包”可能是最好的描述方式了。它的含义很简单：“把 argv 中的东西解包，将所有的参数依次赋予左边的变量名”。

接下来就是正常的打印了。

等一下！“功能”还有另外一个名字

前面我们使用 `import` 让你的程序实现更多的功能，但实际上没人吧 `import` 称为“功能”。我希望你可以在没接触到正式术语的时候就弄懂它的功能。在继续下去之前，你需要知道它们的真正名称：模组(modules)。

从现在开始我们将把这些我们导入(import)进来的功能称作模组。你将看到类似这样的说法：“你需要把 `sys` 模组 `import` 进来。”也有人将它们称作“库(libraries)”，不过我们还是叫它们模组吧。

你应该看到的结果

用下面的方法运行你的程序（注意你必须传递*三个参数）：

```
python ex13.py first 2nd 3rd
```

如果你每次使用不同的参数运行，你将看到下面的结果：

```
$ python ex13.py first 2nd 3rd
The script is called: ex13.py
Your first variable is: first
Your second variable is: 2nd
Your third variable is: 3rd
```

```
$ python ex13.py cheese apples bread
The script is called: ex13.py
Your first variable is: cheese
Your second variable is: apples
```

```
Your third variable is: bread

$ python ex13.py Zed A. Shaw
The script is called: ex13.py
Your first variable is: Zed
Your second variable is: A.
Your third variable is: Shaw
```

你其实可以将“first”、“2nd”、“3rd”替换成任意三样东西。你可以将它们换成任意你想要的东西。

```
python ex13.py stuff I like
python ex13.py anything 6 7
```

如果你没有运行对，你将看到如下错误：

```
python ex13.py first 2nd
Traceback (most recent call last):
  File "ex/ex13.py", line 3, in <module>
    script, first, second, third = argv
ValueError: need more than 3 values to unpack
```

当你运行脚本时提供的参数的个数不对的时候，你就会看到上述错误信息（这次我只用了 first 2nd 两个参数）。“need more than 3 values to unpack”这个错误信息告诉你参数数量不足。

加分习题

1. 给你的脚本三个以下的参数。看看会得到什么错误信息。试着解释一下。
2. 再写两个脚本，其中一个接受更少的参数，另一个接受更多的参数，在参数解包时给它们取一些有意义的变量名。
3. 将 `raw_input` 和 `argv` 一起使用，让你的脚本从用户手上得到更多的输入。
4. 记住“模组(modules)”为你提供额外功能。多读几遍把这个词记住，因为我们后面还会用到它。

常见问题回答

运行时错误信息 `ValueError: need more than 1 value to unpack`。

记住，有一个很重要的技能是注重细节。如果你仔细阅读并且完整重复了“你应该看到的结果”部分的命令参数，你就不会看到这样的错误信息。

`argv` 和 `raw_input()` 有什么不同？

不同点在于用户输入的时机。如果参数是在用户执行命令时就要输入，那就是 `argv`，如果是在脚本运行过程中需要用户输入，那就使用 `raw_input()`。

命令行参数是字符串吗？

是的，就算你在命令行输入数字，你也需要用 `int()` 把它先转成数字，和在 `raw_input()` 里一样。

命令行该怎么使用？

这个你应该已经学会了才对。如果你还没学会，就去读读我写的《命令行迫降式入门》吧
<http://cli.learncodethehardway.org/book/>

`argv` 和 `raw_input()` 不能合起来用。

别想太多了。在脚本结尾加两行 `raw_input()` 随便读取点用户输入就行了，然后再慢慢在脚本中玩玩这两个东东。

为什么 `raw_input('? ') = x` 不灵？

因为你写反了。照着我的写就没问题了。

习题 14: 提示和传递

让我们使用 `argv` 和 `raw_input` 一起来向用户提一些特别的问题。下一节习题你会学习如何读写文件，这节练习是下节的基础。在这道习题里我们将用略微不同的方法使用 `raw_input`，让它打出一个简单的 `>` 作为提示符。这和一些游戏中的方式类似，例如 **Zork** 或者 **Adventure** 这两款游戏。

```
1
2 from sys import argv
3
4 script, user_name = argv
5 prompt = '>'
6
7 print "Hi %s, I'm the %s script." % (user_name, script)
8 print "I'd like to ask you a few questions."
9 print "Do you like me %s?" % user_name
10 likes = raw_input(prompt)
11
12
13 print "Where do you live %s?" % user_name
14 lives = raw_input(prompt)
15
16
17 print "What kind of computer do you have?"
18 computer = raw_input(prompt)
19
20
21 print """
22 Alright, so you said %r about liking me.
23 You live in %r.  Not sure where that is.
24 And you have a %r computer.  Nice.
25 """ % (likes, lives, computer)
```

我们将用户提示符设置为变量 `prompt`，这样我们就不需要在每次用到 `raw_input` 时重复输入提示用户的字符了。而且如果你要将提示符修改成别的字串，你只要改一个位置就可以了。

非常顺手吧。

你应该看到的结果

当你运行这个脚本时，记住你需要把你的名字赋给这个脚本，让 `argv` 参数接收到你的名称。

```
$ python ex14.py Zed
Hi Zed, I'm the ex14.py script.
I'd like to ask you a few questions.
Do you like me Zed?
> yes
Where do you live Zed?
> America
What kind of computer do you have?
> Tandy

Alright, so you said 'yes' about liking me.
You live in 'America'.  Not sure where that is.
And you have a 'Tandy' computer.  Nice.
```

加分习题

1. 查一下 **Zork** 和 **Adventure** 是两个怎样的游戏。看看能不能下载到一版，然后玩玩看。
2. 将 `prompt` 变量改成完全不同的内容再运行一遍。

3. 给你的脚本再添加一个参数，让你的程序用到这个参数。
4. 确认你弄懂了三个引号 `"""` 可以定义多行字符串，而 `%` 是字符串的格式化工具。

常见问题回答

运行时出现 `SyntaxError: invalid syntax`

再次说明，你应该使用命令行，而不是 `python` 环境去运行脚本。如果你先输了 `python` 然后试图输入 `python ex14.py Zed` 就会出现这个错误，你这是在 `python` 里运行 `python`。关掉窗口，重新运行 `python ex14.py Zed`。

修改命令提示符是什么意思？

看这句变量定义 `prompt = '> '`，将它改成一个不同的值。这个应该难不倒你，只是修改一个字符串而已，前面的 13 个习题都是围绕字符串来的，自己花时间搞定。

发生错误 `ValueError: need more than 1 value to unpack`。

记得上次我说过，你应该到“你应该看到的结果”部分重复我的动作。集中精力到我的输入，以及为什么我提供了一个命令行参数。

我可以用双引号定义 `prompt` 变量的值吗？

当然可以，试试看就知道了。

你有台 `Tandy` 计算机？

我小时候有过。

运行时出现 `NameError: name 'prompt' is not defined`。

要么拼错了 `prompt` 要么漏写了这一行。回去比较你写的和我写的东西，从最后一行开始直至第一行。

怎样从 `IDLE` 中运行？

不要使用 `IDLE`。

习题 15: 读取文件

你已经学过了 `raw_input` 和 `argv`, 这些是你开始学习读取文件的必备基础。你可能需要多多实验才能明白它的工作原理, 所以你要细心做练习, 并且仔细检查结果。处理文件需要非常仔细, 如果不仔细的话, 你可能会把有用的文件弄坏或者清空。导致前功尽弃。

这节练习涉及到写两个文件。一个正常的 `ex15.py` 文件, 另外一个 `ex15_sample.txt`, 第二个文件并不是脚本, 而是供你的脚本读取的文本文件。以下是后者的内容:

```
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.
```

我们要做的是把该文件用我们的脚本“打开(`open`)”, 然后打印出来。然而把文件名 `ex15_sample.txt` 写死(`hardcode`)在代码中不是一个好主意, 这些信息应该是用户输入的才对。如果我们碰到其他文件要处理, 写死的文件名就会给你带来麻烦了。我们的解决方案是使用 `argv` 和 `raw_input` 来从用户获取信息, 从而知道哪些文件该被处理。

```
1
2 from sys import argv
3
4 script, filename = argv
5 txt = open(filename)
6
7 print "Here's your file %r:" % filename
8 print txt.read()
9
10 print "Type the filename again:"
11 file_again = raw_input("> ")
12 txt_again = open(file_again)
13
14 print txt_again.read()
15
```

这个脚本中有一些新奇的玩意, 我们来快速地过一遍:

代码的 1-3 行使用 `argv` 来获取文件名, 这个你应该已经熟悉了。接下来第 5 行我们看到 `open` 这个新命令。现在请在命令行运行 `pydoc open` 来读读它的说明。你可以看到它和你自己的脚本、或者 `raw_input` 命令类似, 它会接受一个参数, 并且返回一个值, 你可以将这个值赋予一个变量。这就是你打开文件的过程。

第 7 行我们打印了一小行, 但在第 8 行我们看到了新奇的东西。我们在 `txt` 上调用了函数。你从 `open` 获得的东西是一个 `file` (文件), 文件本身也支持一些命令。它接受命令的方式是使用句点 `.` (英文称作 `dot` 或者 `period`), 紧跟着你的命令, 然后是类似 `open` 和 `raw_input` 一样的参数。不同点是: 当你说 `txt.read` 时, 你的意思其实是: “嘿 `txt`! 执行你的 `read` 命令, 无需任何参数!”

脚本剩下的部分基本差不多, 不过我就把剩下的分析作为加分习题留给你自己了。

你应该看到的结果

我的脚本叫 “`ex15_sample.txt`”, 以下是执行结果:

```
$ python ex15.py ex15_sample.txt
Here's your file 'ex15_sample.txt':
```

```
This is stuff I typed into a file.  
It is really cool stuff.  
Lots and lots of fun to have in here.
```

```
Type the filename again:  
> ex15_sample.txt  
This is stuff I typed into a file.  
It is really cool stuff.  
Lots and lots of fun to have in here.
```

```
$
```

加分习题

这节的难度跨越有点大，所以你要尽量做好这节加分习题，然后再继续后面的章节。

1. 在每一行的上面用注解说明这一行的用途。
2. 如果你不确定答案，就问别人，或者上网搜索。大部分时候，只要搜索 “python” 加上你要搜的东西就能得到你要的答案。比如搜索一下 “python open”。
3. 我使用了 “命令” 这个词，不过实际上它们的名字是 “函数 (function)” 和 “方法 (method)”。上网搜索一下这两者的意义和区别。看不明白也没关系，迷失在别的程序员的知识海洋里是很正常的一件事情。
4. 删掉 10-15 行使用到 `raw_input` 的部分，再运行一遍脚本。
5. 只是用 `raw_input` 写这个脚本，想想那种得到文件名称的方法更好，以及为什么。
6. 运行 `pydoc file` 向下滚动直到看见 `read()` 命令 (函数/方法)。看到很多别的命令了吧，你可以找几条试试看。不需要看那些包含 `__` (两个下划线) 的命令，这些只是垃圾而已。
7. 再次运行 `python` 在命令行下使用 `open` 打开一个文件，这种 `open` 和 `read` 的方法也值得你一学。
8. 让你的脚本针对 `txt` and `txt_again` 变量执行一下 `close()`，处理完文件后你需要将其关闭，这是很重要的一点。

常见问题回答

`txt = open(filename)` 返回的是文件的内容吗？

不是，它返回的是一个叫做 “file object” 的东西，你可以把它想象成一个磁带机或者 DVD 机。你可以随意访问内容的任意位置，并且去读取这些内容，不过这个 `object` 本身并不是它的内容。

我没法再我的 Terminal/PowerShell 命令行下输入 `python` 代码。

首先，在命令行输入 `python` 然后敲回车。现在你就在 `python` 环境中了。接下来你就可以输入并运行一句一句的代码。试着玩玩，如果想退出就输入 `quit()` 再敲回车。

`from sys import argv` 是什么意思？

现在能告诉你的是，`sys` 是一个代码库，这句话的意思是从库里取出 `argv` 这个功能来，供我使用。后面你会学到更多相关知识。

我把文件名写进去写成 `script, ex15_sample.txt = argv` 不过这样不灵。

这么做是错的。把代码写成和我一模一样，然后从命令行运行，照着我的方式。你不需要把文件名放到代码中，而是让 `Python` 把文件名当做参数接纳进去。

为什么打开了两次文件没有报错？

`Python` 不会限制你打开文件的次数，事实上有时候多次打开同一个文件是一件必须的事情。

习题 16: 读写文件

如果你做了上一个练习的加分习题，你应该已经了解了各种文件相关的命令（方法/函数）。你应该记住的命令如下：

- `close` – 关闭文件。跟你编辑器的 文件->保存... 一个意思。
- `read` – 读取文件内容。你可以把结果赋给一个变量。
- `readline` – 读取文本文件中的一行。
- `truncate` – 清空文件，请小心使用该命令。
- `write(stuff)` – 将 `stuff` 写入文件。

这是你现在该知道的重要命令。有些命令需要接受参数，这对我们并不重要。你只要记住 `write` 的用法就可以了。 `write` 需要接收一个字符串作为参数，从而将该字符串写入文件。

让我们来使用这些命令做一个简单的文本编辑器吧：

```
1
2 from sys import argv
3
4 script, filename = argv
5
6 print "We're going to erase %r." % filename
7 print "If you don't want that, hit CTRL-C (^C)."
8 print "If you do want that, hit RETURN."
9
10 raw_input("?")
11
12 print "Opening the file..."
13 target = open(filename, 'w')
14
15 print "Truncating the file.  Goodbye!"
16 target.truncate()
17
18 print "Now I'm going to ask you for three lines."
19
20 line1 = raw_input("line 1: ")
21 line2 = raw_input("line 2: ")
22 line3 = raw_input("line 3: ")
23
24 print "I'm going to write these to the file."
25
26 target.write(line1)
27 target.write("\n")
28 target.write(line2)
29 target.write("\n")
30 target.write(line3)
31 target.write("\n")
32
33 print "And finally, we close it."
34 target.close()
35
```

这个文件是够大的，大概是你键入过的最大的文件。所以慢慢来，仔细检查，让它能运行起来。有一个小技巧就是你可以让你的脚本一部分一部分地运行起来。先写 1-8 行，让它运行起来，再多运行 5 行，再接着多运行几行，以此类推，直到整个脚本运行起来为止。

你应该看到的结果

你将看到两样东西，一样是你新脚本的输出：

```
$ python ex16.py test.txt
We're going to erase 'test.txt'.
If you don't want that, hit CTRL-C (^C).
If you do want that, hit RETURN.
?
Opening the file...
Truncating the file. Goodbye!
Now I'm going to ask you for three lines.
line 1: To all the people out there.
line 2: I say I don't like my hair.
line 3: I need to shave it off.
I'm going to write these to the file.
And finally, we close it.
$
```

接下来打开你新建的文件（我的是 `test.txt`）检查一下里边的内容，怎么样，不错吧？

加分习题

1. 如果你觉得自己没有弄懂的话，用我们的老办法，在每一行之前加上注解，为自己理清思路。就算不能理清思路，你也可以知道自己究竟具体哪里没弄明白。
2. 写一个和上一个练习类似的脚本，使用 `read` 和 `argv` 读取你刚才新建的文件。
3. 文件中重复的地方太多了。试着用一个 `target.write()` 将 `line1`, `line2`, `line3` 打印出来，你可以使用字符串、格式化字符、以及转义字符。
4. 找出为什么我们需要给 `open` 多赋予一个 `'w'` 参数。提示：`open` 对于文件的写入操作态度是安全第一，所以你只有特别指定以后，它才会进行写入操作。
5. 如果你用 `'w'` 模式打开文件，那么你是不是还要 `target.truncate()` 呢？阅读以下 Python 的 `open` 函数的文档找找答案。

常见问题回答

如果用了 `'w'` 参数，`truncate()` 是必须的吗？

看看加分习题 5。

`'w'` 是什么意思？

它只是一个特殊字符串，用来表示文件的访问模式。如果你用了 `'w'` 那么你的文件就是写入（write）模式。除了 `'w'` 以外，我们还有 `'r'` 表示读取（read），`'a'` 表示追加（append）。

还有哪些修饰符可以用来控制文件访问？

最重要的是 `+` 修饰符，写法就是 `'w+'`, `'r+'`, `'a+'` ——这样的话文件将以同时读写的方式打开，而对于文件位置的使用也有些不同。

如果只写 `open(filename)` 那就使用 `'r'` 模式打开的吗？

是的，这是 `open()` 函数的默认工作方式。

习题 17: 更多文件操作

现在让我们再学习几种文件操作。我们将编写一个 Python 脚本，将一个文件中的内容拷贝到另外一个文件中。这个脚本很短，不过它会让你对于文件操作有更多的了解。

```
1
2 from sys import argv
3 from os.path import exists
4
5 script, from_file, to_file = argv
6
7 print "Copying from %s to %s" % (from_file, to_file)
8
9 # we could do these two on one line too, how?
10 in_file = open(from_file)
11 indata = in_file.read()
12
13 print "The input file is %d bytes long" % len(indata)
14
15 print "Does the output file exist? %r" % exists(to_file)
16 print "Ready, hit RETURN to continue, CTRL-C to abort."
17 raw_input()
18
19 out_file = open(to_file, 'w')
20 out_file.write(indata)
21
22 print "Alright, all done."
23
24 out_file.close()
25 in_file.close()
26
```

你应该很快注意到了我们 import 了又一个很好用的命令 exists。这个命令将文件名字符串作为参数，如果文件存在的话，它将返回 True，否则将返回 False。在本书的下半部分，我们将使用这个函数做很多的事情，不过现在你应该学会怎样通过 import 调用它。

通过使用 import，你可以在自己代码中直接使用其他更厉害的（通常是这样，不过也不尽然）程序员写的大量免费代码，这样你就不需要重写一遍了。

你应该看到的结果

和你前面写的脚本一样，运行该脚本需要两个参数，一个是待拷贝的文件，一个是要拷贝至的文件。如果我们使用以前的 test.txt 我们将看到如下的结果：

```
$ python ex17.py test.txt copied.txt
Copying from test.txt to copied.txt
The input file is 81 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort.
```

```
Alright, all done.
```

```
$ cat copied.txt
To all the people out there.
I say I don't like my hair.
I need to shave it off.
$
```

该命令对于任何文件都应该有效的。试试操作一些别的文件看看结果。不过小心别把你的重要文件给弄坏了。

Warning

你看到我用 `cat` 这个命令了吧？它只能在 **Linux** 和 **OSX** 下面使用，使用 **Windows** 的就只好跟你说声抱歉了。

加分习题

1. 再多读读和 `import` 相关的材料，将 `python` 运行起来，试试这一条命令。试着看看自己能不能摸出点门道，当然了，即使弄不明白也没关系。
2. 这个脚本 实在是 有点烦人。没必要在拷贝之前问一遍把，没必要在屏幕上输出那么多东西。试着删掉脚本的一些功能，让它使用起来更加友好。
3. 看看你能把这个脚本改多短，我可以把它写成一行。
4. 我使用了一个叫 *cat* 的东西，这个古老的命令的用处是将两个文件“连接(`con*cat*enate`)”到一起，不过实际上它最大的用途是打印文件内容到屏幕上。你可以通过 `man cat` 命令了解到更多信息。
5. 使用 **Windows** 的同学，你们可以给自己找一个 `cat` 的替代品。关于 `man` 的东西就别想太多了，**Windows** 下没这个命令。
6. 找出为什么你需要在代码中写 `output.close()` 。

常见问题回答

为什么 `'w'` 要放在括号中？

因为这是一个字符串，你已经学过一阵子字符串了，确定自己真的学会了。

不可能把这写在一行里边！

取决于你的行是怎么定义的——例如这样： `That ; depends ; on ; how ; you ; define ; one ; line ; of ; code.`

`len()` 函数的功能是什么？

它会以数字的形式返回你传递的字符串的长度。试着玩玩吧。

当我把代码写短时，我在关闭文件时出现一个错误。

很可能是你写了 `indata = open(from_file).read()` 这意味着你无需再运行 `in_file.close()` 了，因为 `read()` 一旦运行，文件就会被读到结尾并且被 `close` 掉。

我觉得这个习题很难，这个是正常现象吗？

是的，再正常不过了。也许在你看到习题 36 之前，甚至读完本书，编程对你来说都还是一件很难理解的事情。每个人的情况都不一样，坚持读书做练习，有问题的地方多研究，总会弄明白的。慢工出细活。

Syntax:EOL while scanning string literal 错误。

字符串结尾忘记加引号了。仔细检查那行看看。

习题 18: 命名、变量、代码、函数

标题包含的内容够多的吧？接下来我要教你“函数(function)”了！咚咚锵！说到函数，不一样的人会对它有不一样的理解和使用方法，不过我只会教你现在能用到的最简单的使用方式。

函数可以做三样事情：

1. 它们给代码片段命名，就跟“变量”给字符串和数字命名一样。
2. 它们可以接受参数，就跟你的脚本接受 `argv` 一样。
3. 通过使用 `#1` 和 `#2`，它们可以让你创建“微型脚本”或者“小命令”。

你可以使用 `def` 新建函数。我将让你创建四个不同的函数，它们工作起来和你的脚本一样。然后我会演示给你各个函数之间的关系。

```
1 # this one is like your scripts with argv
2 def print_two(*args):
3     arg1, arg2 = args
4     print "arg1: %r, arg2: %r" % (arg1, arg2)
5
6 # ok, that *args is actually pointless, we can just do this
7 def print_two_again(arg1, arg2):
8     print "arg1: %r, arg2: %r" % (arg1, arg2)
9
10 # this just takes one argument
11 def print_one(arg1):
12     print "arg1: %r" % arg1
13
14 # this one takes no arguments
15 def print_none():
16     print "I got nothin'."
17
18 print_two("Zed", "Shaw")
19 print_two_again("Zed", "Shaw")
20 print_one("First!")
21 print_none()
```

让我们把你一个函数 `print_two` 肢解一下，这个函数和你写脚本的方式差不多，因此你看上去应该会觉着比较眼熟：

1. 首先我们告诉 **Python** 创建一个函数，我们使用到的命令是 `def`，也就是“定义(define)”的意思。
2. 紧接着 `def` 的是函数的名称。本例中它的名称是“`print_two`”，但名字可以随便取，就叫“`peanuts`”也没关系。但最好函数的名称能够体现出函数的功能来。
3. 然后我们告诉函数我们需要 `*args` (asterisk args)，这和脚本的 `argv` 非常相似，参数必须放在圆括号 `()` 中才能正常工作。
4. 接着我们用冒号 `:` 结束本行，然后开始下一行缩进。
5. 冒号以下，使用 4 个空格缩进的行都是属于 `print_two` 这个函数的内容。其中第一行的作用是将参数解包，这和脚本参数解包的原理差不多。
6. 为了演示它的工作原理，我们把解包后的每个参数都打印出来，这和我们在之前脚本练习中所作的类似。

函数 `print_two` 的问题是：它并不是创建函数最简单的方法。在 **Python** 函数中我们可以跳过整个参数解包的过程，直接使用 `()` 里边的名称作为变量名。这就是 `print_two_again` 实现的功能。

接下来的例子是 `print_one`，它向你演示了函数如何接受单个参数。

最后一个例子是 `print_none`，它向你演示了函数可以不接收任何参数。

Warning

如果你不太能看懂上面的内容也别气馁。后面我们还有更多的练习向你展示如何创建和使用函数。现在你只要把函数理解成“迷你脚本”就可以了。

你应该看到的结果

运行上面的脚本会看到如下结果：

```
$ python ex18.py
arg1: 'Zed', arg2: 'Shaw'
arg1: 'Zed', arg2: 'Shaw'
arg1: 'First!'
I got nothin'.
$
```

你应该已经看出函数是怎样工作的了。注意到函数的用法和你以前见过的 `exists`、`open`，以及别的“命令”有点类似了吧？其实我只是为了让你容易理解才叫它们“命令”，它们的本质其实就是函数。也就是说，你也可以在自己的脚本中创建你自己的“命令”。

加分习题

为自己写一个函数注意事项以供后续参考。你可以写在一个索引卡片上随时阅读，直到你记住所有的要点为止。注意事项如下：

1. 函数定义是以 `def` 开始的吗？
2. 函数名称是以字符和下划线 `_` 组成的吗？
3. 函数名称是不是紧跟着括号 `(`？
4. 括号里是否包含参数？多个参数是否以逗号隔开？
5. 参数名称是否有重复？（不能使用重复的参数名）
6. 紧跟着参数的是不是括号和冒号 `):`？
7. 紧跟着函数定义的代码是否使用了 4 个空格的缩进 (`indent`)？
8. 函数结束的位置是否取消了缩进 (`dedent`)？

当你运行（或者说“使用 `use`”或者“调用 `call`”）一个函数时，记得检查下面的要点：

1. 调运函数时是否使用了函数的名称？
2. 函数名称是否紧跟着 `(`？
3. 括号后有无参数？多个参数是否以逗号隔开？
4. 函数是否以 `)` 结尾？

按照这两份检查表里的内容检查你的练习，直到你不需要检查表为止。

最后，将下面这句话阅读几遍：

“‘运行函数(`run`)’、‘调用函数(`call`)’、和‘使用函数(`use`)’是同一个意思”

常见问题回答

函数名称有什么规则？

和变量名一样，只要以字母数字以及下划线组成，而且不是数字开始，就可以了。

`*args` 的 `*` 是什么意思？

它的功能是告诉 `python` 让它把函数的所有参数都接受进来，然后放到名字叫 `args` 的列表中去。和你一直在用的 `argv` 差不多，只不过前者是用在函数上面。没什么特殊情况，我们一般不会经

常用到这个东西。

这些任务好枯燥好无聊啊。

这就对了，你这么感觉，说明你有了进步，你能明白代码的功用，而且写错代码的情况在你身上很少发生了。为了让任务不那么无聊，你可以试着故意写错一些东西，看看会发生什么事情。

习题 19: 函数和变量

函数这个概念也许承载了太多的信息量，不过别担心。只要坚持做这些练习，对照上个练习中的检查点检查一遍这次的联系，你最终会明白这些内容的。

有一个你可能没有注意到的细节，我们现在强调一下：函数里边的变量和脚本里边的变量之间是没有连接的。下面的这个练习可以让你对这一点有更多的思考：

```
1 def cheese_and_crackers(cheese_count, boxes_of_crackers):
2     print "You have %d cheeses!" % cheese_count
3     print "You have %d boxes of crackers!" % boxes_of_crackers
4     print "Man that's enough for a party!"
5     print "Get a blanket.\n"
6
7
8 print "We can just give the function numbers directly:"
9 cheese_and_crackers(20, 30)
10
11 print "OR, we can use variables from our script:"
12 amount_of_cheese = 10
13 amount_of_crackers = 50
14
15 cheese_and_crackers(amount_of_cheese, amount_of_crackers)
16
17
18 print "We can even do math inside too:"
19 cheese_and_crackers(10 + 20, 5 + 6)
20
21
22 print "And we can combine the two, variables and math:"
23 cheese_and_crackers(amount_of_cheese + 100, amount_of_crackers + 1000)
24
```

通过这个练习，你看到我们给我们的函数 `cheese_and_crackers` 很多的参数，然后在函数里把它们打印出来。我们可以在函数里用变量名，我们可以在函数里做运算，我们甚至可以将变量和运算结合起来。

从一方面来说，函数的参数和我们的生成变量时用的 `=` 赋值符类似。事实上，如果一个物件你可以用 `=` 将其命名，你通常也可以将其作为参数传递给一个函数。

你应该看到的结果

你应该研究一下脚本的输出，和你想象的结果对比一下看有什么不同。

```
$ python ex19.py
We can just give the function numbers directly:
You have 20 cheeses!
You have 30 boxes of crackers!
Man that's enough for a party!
Get a blanket.
```

```
OR, we can use variables from our script:
You have 10 cheeses!
You have 50 boxes of crackers!
Man that's enough for a party!
Get a blanket.
```

```
We can even do math inside too:
You have 30 cheeses!
You have 11 boxes of crackers!
Man that's enough for a party!
Get a blanket.
```

```
And we can combine the two, variables and math:
You have 110 cheeses!
You have 1050 boxes of crackers!
Man that's enough for a party!
Get a blanket.
$
```

加分习题

1. 倒着将脚本读完，在每一行上面添加一行注解，说明这行的作用。
2. 从最后一行开始，倒着阅读每一行，读出所有的重要字符来。
3. 自己编至少一个函数出来，然后用 10 种方法运行这个函数。

常见问题回答

怎么能有 10 种不同的方式运行一个函数呢？

信不信由你，理论上有无穷多种方法运行一个函数。在这里，试着按我在 8-12 行的方式运行，当然你可以随意创新。

有没有办法可以分析这个函数的功能以便我理解？

有很多方法，最简单的一个是在每一行代码上面添加注解，另外一个方法是大声朗读代码，还有一个方法就是把代码打印出来，用笔画一些图示，并写一些注解说明。

怎样处理用户输入的数字，例如我想让用户输入克力架和奶酪的数量？

记住使用 `int()` 把 `raw_input()` 的值转为整数。

第 13 和 14 行创建的变量会不会改变函数中的变量？

不会。这些变量是在函数之外的，当它们被传递到函数中以后，函数会为这些变量创建一些临时的版本，当函数运行结束后，这些临时变量就被丢弃了，一切又回到了从前。继续阅读本书，后面你会更清楚这些概念。

把全局变量（如 13、14 行）的名称和函数变量的名称取成一样的，这样做是不是不好？

是的，因为这样的话你就无法确定哪个是哪个了。有时候你可能会必须使用同一个变量名，有时候你会不小心使用了一样的变量名，不论如何，只要有可能，还是尽量避免变量名称相同吧。

第 12-19 行是不是覆写了函数 `cheese_and_crackers`？

没有，完全没有。这只是函数调用而已。基本上就是这里会跳转到函数的第一行，然后等函数运行完后再回到先前的位置。并没有把原函数怎么地。

函数的参数个数有限制吗？

取决于 Python 的版本和你的操作系统，不过就算有限，限值也是很大的。实际应用中，5 个参数就不少了，再多就会让人头疼了。

可以在函数中调用函数吗？

可以。后面的习题中你会用这一技巧写一个游戏。

习题 20: 函数和文件

回忆一下函数的要点，然后一边做这节练习，一边注意一下函数和文件是如何在一起协作发挥作用的。

```
1 from sys import argv
2
3 script, input_file = argv
4
5 def print_all(f):
6     print f.read()
7
8 def rewind(f):
9     f.seek(0)
10
11 def print_a_line(line_count, f):
12     print line_count, f.readline()
13
14 current_file = open(input_file)
15 print "First let's print the whole file:\n"
16
17 print_all(current_file)
18
19 print "Now let's rewind, kind of like a tape."
20
21 rewind(current_file)
22
23 print "Let's print three lines:"
24
25 current_line = 1
26 print_a_line(current_line, current_file)
27
28 current_line = current_line + 1
29 print_a_line(current_line, current_file)
30
31 current_line = current_line + 1
32 print_a_line(current_line, current_file)
33
```

特别注意一下，每次运行 `print_a_line` 时，我们是怎样传递当前的行号信息的。

你应该看到的结果

```
$ python ex20.py test.txt
First let's print the whole file:
```

```
To all the people out there.
I say I don't like my hair.
I need to shave it off.
```

```
Now let's rewind, kind of like a tape.
```

```
Let's print three lines:
1 To all the people out there.
```

```
2 I say I don't like my hair.
```

```
3 I need to shave it off.
```

```
$
```


加分习题

1. 通读脚本，在每行之前加上注解，以理解脚本里发生的事情。
2. 每次 `print_a_line` 运行时，你都传递了一个叫 `current_line` 的变量。在每次调用函数时，打印出 `current_line` 的至，跟踪一下它在 `print_a_line` 中是怎样变成 `line_count` 的。
3. 找出脚本中每一个用到函数的地方。检查 `def` 一行，确认参数没有用错。
4. 上网研究一下 `file` 中的 `seek` 函数是做什么用的。试着运行 `pydoc file` 看看能不能学到更多。
5. 研究一下 `+=` 这个简写操作符的作用，写一个脚本，把这个操作符用在里边试一下。

常见问题回答

`print_all` 和其它函数里的 `f` 是什么？

和 Ex 18 里的一样，`f` 只是一个变量名而已，不过在这里它指的是一个文件。Python 里的文件就和老式磁带机，或者 DVD 播放机差不多。它有一个用来读取数据的“磁头”，你可以通过这个“磁头”来操作文件。每次你运行 `f.seek(0)` 你就回到了文件的开始，而运行 `f.readline()` 则会读取文件的一行，然后将“磁头”移动到 `\n` 后面。后面你会看到更详细的解释。

问什么文件里会有间隔空行？

`readline()` 函数返回的内容中包含文件本来就有的 `\n`，而 `print` 在打印时又会添加一个 `\n`，这样一来就会多出一个空行了。解决方法是在 `print` 语句结尾加一个逗号 `,`，这样 `print` 就不会把它自己的 `\n` 打印出来了。

为什么 `seek(0)` 没有把 `current_line` 设为 0？

首先 `seek()` 函数的处理对象是 字节 而非行，所以 `seek(0)` 只是转到文件的 0 byte，也就是第一个 byte 的位置。其次，`current_line` 只是一个独立变量，和文件本身没有任何关系，我们只能手动为其增值。

`+=` 是什么？

英语里边 “it is” 可以写成 “it’s”，“you are” 可以写成 “you’re”，这叫做简写。而这个操作符是吧 `=` 和 `+` 简写到一起了。`x += y` 的意思和 `x = x + y` 是一样的。

`readline()` 是怎么知道每一行在哪里的？

`readline()` 里边的代码会扫描文件的每一个字节，直到找到一个 `\n` 为止，然后它停止读取文件，并且返回此前的文件内容。文件 `f` 会记录每次调用 `readline()` 后的读取位置，这样它就可以在下次被调用时读取接下来的一行了。

习题 21: 函数可以返回东西

你已经学过使用 `=` 给变量命名，以及将变量定义为某个数字或者字符串。接下来我们将让你见证更多奇迹。我们要演示给你的是如何使用 `=` 以及一个新的 Python 词汇 `return` 来将变量设置为“一个函数的值”。有一点你需要及其注意，不过我们暂且不讲，先撰写下面的脚本吧：

```
1 def add(a, b):
2     print "ADDING %d + %d" % (a, b)
3     return a + b
4
5 def subtract(a, b):
6     print "SUBTRACTING %d - %d" % (a, b)
7     return a - b
8
9 def multiply(a, b):
10    print "MULTIPLYING %d * %d" % (a, b)
11    return a * b
12
13 def divide(a, b):
14    print "DIVIDING %d / %d" % (a, b)
15    return a / b
16
17
18 print "Let's do some math with just functions!"
19
20 age = add(30, 5)
21 height = subtract(78, 4)
22 weight = multiply(90, 2)
23 iq = divide(100, 2)
24
25 print "Age: %d, Height: %d, Weight: %d, IQ: %d" % (age, height, weight, iq)
26
27
28 # A puzzle for the extra credit, type it in anyway.
29 print "Here is a puzzle."
30
31 what = add(age, subtract(height, multiply(weight, divide(iq, 2))))
32
33 print "That becomes: ", what, "Can you do it by hand?"
```

现在我们创建了我们自己的加减乘除数学函数：`add`, `subtract`, `multiply`, 以及 `divide`。重要的是函数的最后一行，例如 `add` 的最后一行是 `return a + b`，它实现的功能是这样的：

1. 我们调用函数时使用了两个参数：`a` 和 `b`。
2. 我们打印出这个函数的功能，这里就是计算加法（**adding**）
3. 接下来我们告诉 **Python** 让它做某个回传的动作：我们将 `a + b` 的值返回(**return**)。或者你可以这么说：“我将 `a` 和 `b` 加起来，再把结果返回。”
4. **Python** 将两个数字相加，然后当函数结束的时候，它就可以将 `a + b` 的结果赋予一个变量。

和本书里的很多其他东西一样，你要慢慢消化这些内容，一步一步执行下去，追踪一下究竟发生了什么。为了帮助你理解，本节的加分习题将让你解决一个谜题，并且让你学到点比较酷的东西。

你应该看到的结果

```
$ python ex21.py
Let's do some math with just functions!
```

```
ADDING 30 + 5
SUBTRACTING 78 - 4
MULTIPLYING 90 * 2
DIVIDING 100 / 2
Age: 35, Height: 74, Weight: 180, IQ: 50
Here is a puzzle.
DIVIDING 50 / 2
MULTIPLYING 180 * 25
SUBTRACTING 74 - 4500
ADDING 35 + -4426
That becomes: -4391 Can you do it by hand?
$
```

加分习题

1. 如果你不是很确定 `return` 的功能，试着自己写几个函数出来，让它们返回一些值。你可以将任何可以放在 `=` 右边的东西作为一个函数的返回值。
2. 这个脚本的结尾是一个谜题。我将一个函数的返回值用作了另外一个函数的参数。我将它们链接到了一起，就跟写数学等式一样。这样可能有些难读，不过运行一下你就知道结果了。接下来，你需要试试看能不能用正常的方法实现和这个表达式一样的功能。
3. 一旦你解决了这个谜题，试着修改一下函数里的某些部分，然后看会有什么样的结果。你可以有目的地修改它，让它输出另外一个值。
4. 最后，颠倒过来做一次。写一个简单的等式，使用一样的函数来计算它。

这个习题可能会让你有些头大，不过还是慢慢来，把它当做一个游戏，解决这样的谜题正是编程的乐趣之一。后面你还会看到类似的小谜题。

常见问题回答

为什么 `Python` 会把函数或公式倒着打印出来？

其实不是倒着打印，而是自内而外打印。如果你把函数内容逐句看下去，你会发现这里的规律。试着搞清楚为什么说它是“自内而外”而不是“自下而上”。

怎样使用 `raw_input()` 输入自定义值？

记得 `int(raw_input())` 吧？不过这样也有一个问题，那就是你无法输入浮点数，所以你可以试着使用 `float(raw_input())`。

你说的“写一个公式”是什么意思？

来个简单的例子吧： $24 + 34 / 100 - 1023$ ——把它用函数的形式写出来。然后自己想一些数学式子，像公式一样用变量写出来。

习题 22: 到现在你学到了哪些东西?

这节以及下一节的习题中不会有任何代码，所以也不会有习题答案或者加分习题。其实这节习题可以说是一个巨型的加分习题。我将让你完成一个表格，让你回顾你到现在学到的所有东西。

首先，回到你的每一个习题的脚本里，把你碰到的每一个词和每一个符号（`symbol`，`character` 的别名）写下来。确保你的符号列表是完整的。

下一步，在每一个关键词和字符后面写出它的名字，并且说明它的作用。如果你在书里找不到符号的名字，就上网找一下。如果你不知道某个关键字或者符号的作用，就回到用到该字符的章节通读一下，并且在脚本中测试一下这个字符的用处。

你也许会碰到一些横竖找不到答案的东西，只要把这些记在列表里，它可以提示你让你知道还有哪些东西不懂，等下次碰到的时候，你就不会轻易跳过了。

你的列表做好以后，再花几天时间重写一遍这份列表，确认里边的东西都是正确的。你可能觉得这很无聊，不过你还是需要坚持完成任务。

等你记住了这份列表中的所有内容，就试着把这份列表默写一遍。如果发现自己漏掉或者忘记了某些内容，就回去再记一遍。

Warning

做这节练习没有失败，只有尝试，请牢记这一点。

你学到的东西

这种记忆练习是枯燥无味的，所以知道它的意义很重要。它会让你明确目标，让你知道你所有努力的目的。

在这节练习中你学会的是各种符号的名称，这样读代码对你来说会更加容易。这和学英语时记忆字母表和基本单词的意义是一样的，不同的是 `Python` 中会有一些你不熟悉的字符。

慢慢做，别让它成为你的负担。这些符号对你来说应该比较熟悉，所以记住它们应该不是很费力的事情。你可以一次花个 15 分钟，然后休息一下。作息结合可以让你学得更快，而且可以让你保持士气。

习题 23: 读代码

上一周你应该已经牢记了你的符号列表。现在你需要将这些运用起来，再花一周的时间，在网上阅读代码。这个任务初看会觉得很艰巨。我将直接把你丢到深水区呆几天，让你竭尽全力去读懂实实在在的项目里的代码。这节练习的目的不是让你读懂，而是让你学会下面的技能：

1. 找到你需要的 Python 代码。
2. 通读代码，找到文件。
3. 尝试理解你找到的代码。

以你现在的水平，你还不具备完全理解你找到的代码的能力，不过通过接触这些代码，你可以熟悉真正的编程项目会是什么样子。

当你做这节练习时，你可以把自己当成是一个人类学家来到了一片陌生的大陆，你只懂得一丁点本地语言，但你需要接触当地人并且生存下去。当然做练习不会碰到生存问题，这毕竟这不是荒野或者丛林。

你要做的事情如下：

1. 使用你的浏览器登录 bitbucket.org，搜索 “python”。
2. 忽略那些提到 “Python 3” 的项目，它们只会让你变迷糊。
3. 随便找一个项目，然后点进去。
4. 点击 Source 标签，浏览目录和文件列表，直到你看到以 .py 结尾的文件（`setup.py` 就别看了，这样的文件看了也没用）。
5. 从头开始阅读你找到的代码。把它的功能用笔记记下来。
6. 如果你看到一些有趣的符号或者奇怪的字串，你可以把它们记下来，日后再进行研究。

就是这样，你的任务是使用你目前学到的东西，看自己能不能读懂一些代码，看出它们的功能来。你可以先粗略地阅读，然后再细读。也许你还可以试试将难度比较大的部分一字不漏地朗读出来。

现在再试试其它三个站点：

- github.com
- launchpad.net
- koders.com

在这些网站你可能还会看到以 .c 结尾的奇怪文件，不过你只需要看 .py 结尾的文件就可以了。

最后一个有趣的事情是你可以在这四个网站搜索 “python” 以外的你感兴趣的话题，例如你可以搜索 “journalism（新闻）”，“cooking（厨艺）”，“physics（物理）”，或者任何你感兴趣的话题。你也许会找到一些你对你有用的，可以直接拿来用的代码。

常见问题回答

能不能推荐我几个项目链接？这样我就不用四处寻觅了。

看看我的这个项目 <https://github.com/zedshaw/lamson> 然后在附近搜索一下。

救命啊，我看不懂！

看不懂没关系，你还是初学者。这个练习的目的是直接把你丢到泳池里让你自己试着扑腾。等你适应了，后面你再碰到别人的代码时就不会那么头大了。

我真的需要每天去做，坚持一个星期吗？

如果你有时间的话就坚持一个星期，不过也别死守着这条。你可以花个半小时看看别人的代码，再花一个小时看后面的习题，这样也没关系，只要看足够多的代码就行了。

习题 24: 更多练习

你离这本书第一部分的结尾已经不远了，你应该已经具备了足够的 Python 基础知识，可以继续学习一些编程的原理了，但你应该做更多的练习。这个练习的内容比较长，它的目的是锻炼你的毅力，下一个习题也差不多是这样的，好好完成它们，做到完全正确，记得仔细检查。

```
1 print "Let's practice everything."
2 print 'You\'d need to know \'bout escapes with \\ that do \n newlines and \t tabs.'
3
4 poem = """
5 \tThe lovely world
6 with logic so firmly planted
7 cannot discern \n the needs of love
8 nor comprehend passion from intuition
9 and requires an explanation
10 \n\t\twhere there is none.
11 """
12 print "-----"
13 print poem
14 print "-----"
15
16 five = 10 - 2 + 3 - 6
17 print "This should be five: %s" % five
18
19 def secret_formula(started):
20     jelly_beans = started * 500
21     jars = jelly_beans / 1000
22     crates = jars / 100
23     return jelly_beans, jars, crates
24
25
26
27 start_point = 10000
28 beans, jars, crates = secret_formula(start_point)
29
30 print "With a starting point of: %d" % start_point
31 print "We'd have %d beans, %d jars, and %d crates." % (beans, jars, crates)
32
33 start_point = start_point / 10
34
35 print "We can also do that this way:"
36 print "We'd have %d beans, %d jars, and %d crates." % secret_formula(start_point)
37
```

你应该看到的结果

```
$ python ex24.py
Let's practice everything.
You'd need to know 'bout escapes with \ that do
newlines and    tabs.
-----
```

```

    The lovely world
with logic so firmly planted
cannot discern
the needs of love
```

```
nor comprehend passion from intuition
and requires an explanation
```

```
where there is none.
```

```
-----
This should be five: 5
With a starting point of: 10000
We'd have 5000000 beans, 5000 jars, and 50 crates.
We can also do that this way:
We'd have 500000 beans, 500 jars, and 5 crates.
$
```

加分习题

1. 记得仔细检查结果，从后往前倒着检查，把代码朗读出来，在不清楚的位置加上注释。
2. 故意把代码改错，运行并检查会发生什么样的错误，并且确认你有能力改正这些错误。

常见问题回答

为什么你在后面把 `jelly_beans` 这个变量名又叫成了 `beans`？

这是函数的工作原理。记住函数内部的变量都是临时的，当你的函数返回以后，返回值可以被赋予一个变量。我这里是创建了一个新变量，用来存放函数的返回值。

倒着阅读代码是什么意思？

从最后一行开始，把你写的和我写的代码进行比较。如果这一行完全一样，就接着比较上一行，直到全部比较完为止。

这首诗是谁写的？

我写的。我的诗作偶尔也不赖吧。

习题 25: 更多更多的练习

我们将做一些关于函数和变量的练习，以确认你真正掌握了这些知识。这节练习对你来说可以说是一本道：写程序，逐行研究，弄懂它。

不过这节练习还是有些不同，你不需要运行它，取而代之，你需要将它导入到 `python` 里通过自己执行函数的方式运行。

```
def break_words(stuff):
    """This function will break up words for us."""
    1     words = stuff.split(' ')
    2     return words
    3
    4 def sort_words(words):
    5     """Sorts the words."""
    6     return sorted(words)
    7
    8 def print_first_word(words):
    9     """Prints the first word after popping it off."""
    10    word = words.pop(0)
    11    print word
    12
    13 def print_last_word(words):
    14    """Prints the last word after popping it off."""
    15    word = words.pop(-1)
    16    print word
    17
    18
    19 def sort_sentence(sentence):
    20    """Takes in a full sentence and returns the sorted words."""
    21    words = break_words(sentence)
    22    return sort_words(words)
    23
    24
    25 def print_first_and_last(sentence):
    26    """Prints the first and last words of the sentence."""
    27    words = break_words(sentence)
    28    print_first_word(words)
    29    print_last_word(words)
    30
    31
    32 def print_first_and_last_sorted(sentence):
    33    """Sorts the words then prints the first and last one."""
    34    words = sort_sentence(sentence)
    35    print_first_word(words)
    print_last_word(words)
```

首先以正常的方式 `python ex25.py` 运行，找出里边的错误，并把它们都改正过来。然后你需要接着下面的答案章节完成这节练习。

你应该看到的结果

这节练习我们将在你之前用来做算术的 `python` 编译器里，用交互的方式和你的 `.py` 作交流。

这是我做出来的样子：

```
1 $ python
```



```

2 Python 2.5.1 (r251:54863, Feb  6 2009, 19:02:12)
3 [GCC 4.0.1 (Apple Inc. build 5465)] on darwin
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> import ex25
6 >>> sentence = "All good things come to those who wait."
7 >>> words = ex25.break_words(sentence)
8 >>> words
9 ['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
10 >>> sorted_words = ex25.sort_words(words)
11 >>> sorted_words
12 ['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
13 >>> ex25.print_first_word(words)
14 All
15 >>> ex25.print_last_word(words)
16 wait.
17 >>> wrods
18 Traceback (most recent call last):
19   File "<stdin>", line 1, in <module>
20 NameError: name 'wrods' is not defined
21 >>> words
22 ['good', 'things', 'come', 'to', 'those', 'who']
23 >>> ex25.print_first_word(sorted_words)
24 All
25 >>> ex25.print_last_word(sorted_words)
26 who
27 >>> sorted_words
28 ['come', 'good', 'things', 'those', 'to', 'wait.']
29 >>> sorted_words = ex25.sort_sentence(sentence)
30 >>> sorted_words
31 ['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
32 >>> ex25.print_first_and_last(sentence)
33 All
34 >>> ex25.print_first_and_last_sorted(sentence)
35 All
36 who
37 >>> ^D
38 $
39

```

我们来逐行分析一下每一步实现的是什么：

- 在第 5 行你将自己的 `ex25.py` 执行了 `import`，和你做过的其它 `import` 一样。在 `import` 的时候你不需要加 `.py` 后缀。这个过程里，你把 `ex25.py` 当做了一个“模组(module)”来使用，你在这个模组里定义的函数也可以直接调用出来。
- 第 6 行你创建了一个用来处理的“句子(sentence)”。
- 第 7 行你使用 `ex25` 调用你的第一个函数 `ex25.break_words`。其中的 `.` (dot, period) 符号可以告诉 **Python**：“嗨，我要运行 `ex25` 里的哪个叫 `break_words` 的函数！”
- 第 8 行我们只是输入 `words`，而 **python** 将在第 9 行打印出这个变量里边有什么。看上去可能会觉得奇怪，不过这其实是一个“列表(list)”，你会在后面的章节中学到它。
- 10-11 行我们使用 `ex25.sort_words` 来得到一个排序过的句子。
- 13-16 行我们使用 `ex25.print_first_word` 和 `ex25.print_last_word` 将第一个和最后一个词打印出来。
- 第 17 行比较有趣。我把 `words` 变量写错成了 `wrods`，所以 **python** 在 18-20 行给了一个错误信息。
- 21-22 行我们打印出了修改过的词汇列表。第一个和最后一个单词我们已经打印过了，所以在这里没有再次打印出来。

剩下的行你需要自己分析一下，就留作你的加分习题了。

加分习题

1. 研究答案中没有分析过的行，找出它们的来龙去脉。确认自己明白了自己使用的是模組 `ex25` 中定义的函数。
2. 试着执行 `help(ex25)` 和 `help(ex25.break_words)`。这是你得到模組帮助文档的方式。所谓帮助文档就是你定义函数时放在 `"""` 之间的东西，它们也被称作 `documentation comments`（文档注解），后面你还会看到更多类似的东西。
3. 重复键入 `ex25.` 是很烦的一件事情。有一个捷径就是用 `from ex25 import *` 的方式导入模組。这相当于说：“我要把 `ex25` 中所有的东西 `import` 进来。”程序员喜欢说这样的倒装句，开一个新的会话，看看你所有的函数是不是已经在那里了。
4. 把你脚本里的内容逐行通过 `python` 编译器执行，看看会是什么样子。你可以执行 `CTRL-D` (Windows 下是 `CTRL-Z`) 来关闭编译器。

常见问题回答

有的函数打印出来的结果是 `None`。

也许你的函数漏写了最后的 `return` 语句。回到代码中检查一下是不是每一行都写对了。

输入 `import ex15` 时显示 `-bash: import: command not found`。

注意看《你应该看到的结果》部分。我是在 `Python` 中写的这句，不是在命令行终端直接写的。你要先运行 `python` 再输入代码。

输入 `import ex25.py` 时显示 `ImportError: No module named ex25.py`。

`.py` 是不需要的。`Python` 知道文件是 `.py` 结尾，所以只要输入 `import ex25` 即可。

运行时提示 `SyntaxError: invalid syntax`。

这说明你在提示的那行有一个语法错误，可能是漏了半个括号或者引号，也可能识别的。一旦看到这种错误，你应该去对应的行检查你的代码，如果这行没问题，就倒着继续往上检查每一行，直到发现问题为止。

函数里的代码不是只在函数里有效吗？为什么 `words.pop(0)` 这个函数会改变 `words` 的内容？

这个问题有点复杂，不过在这里 `words` 是一个列表，你可以对它进行操作，操作结果也可以被保存下来。这和你操作文件时文件的 `f.readline()` 工作原理差不多。

函数里什么时候该用 `print`，什么时候该用 `return`？

`print` 只是屏幕输出而已，你可以让一个函数既 `print` 又返回值。当你明白这一点后，你就知道这个问题其实没什么意义。如果你想要打印到屏幕，那就使用 `print`，如果是想返回值，那就是用 `return`。

习题 26: 恭喜你，现在可以考试了！

你已经差不多完成这本书的前半部分了，不过后半部分才是更有趣的。你将学到逻辑，并通过条件判断实现有用的功能。

在你继续学习之前，你有一道试题要做。这道试题很难，因为它需要你修正别人写的代码。当你成为程序员以后，你将需要经常面对别的程序员的代码，也许还有他们的傲慢态度，他们会经常说自己的代码是完美的。

这样的程序员是自以为是不在乎别人的蠢货。优秀的科学家会对他们自己的工作持怀疑态度，同样，优秀的程序员也会认为自己的代码总有出错的可能，他们会先假设是自己的代码有问题，然后用排除法清查所有可能是自己有问题的地方，最后才会得出“这是别人的错误”这样的结论。

在这节练习中，你将面对一个水平糟糕的程序员，并改好他的代码。我将习题 24 和 25 胡乱拷贝到了一个文件中，随机地删掉了一些字符，然后添加了一些错误进去。大部分的错误是 Python 在执行时会告诉你的，还有一些算术错误是你要自己找出来的。再剩下的就是格式和拼写错误了。

所有这些错误都是程序员很容易犯的，就算有经验的程序员也不例外。

你的任务是将此文件修改正确，用你所有的技能改进这个脚本。你可以先分析这个文件，或者你还可以把它像学期论文一样打印出来，修正里边的每一个缺陷，重复修正和运行的动作，直到这个脚本可以完美地运行起来。在整个过程中不要寻求帮助，如果你卡在某个地方无法进行下去，那就休息一会晚点再做。

就算你需要几天才能完成，也不要放弃，直到完全改对为止。

最后要说的是，这个练习的目的不是写程序，而是修正现有的程序，你需要访问下面的网站：

- <http://learnpythonthehardway.org/exercise26.txt>

从那里把代码复制粘贴过来，命名为 `ex26.py`，这也是本书唯一一处允许你复制粘贴的地方。

常见问题回答

一定要 `import ex25.py` 吗？移除对它的引用也可以吧？

怎样都可以。不过这个文件里会用到 `ex25` 中的函数，你可以试着移除引用看看会怎样。

我可以边修正代码边运行吗？

当然可以。这样的事情就是要计算机帮忙，多多益善。

习题 27: 记住逻辑关系

到此为止你已经学会了读写文件，命令行处理，以及很多 Python 数学运算功能。今天，你将要开始学习逻辑了。你要学习的不是研究院里的高深逻辑理论，只是程序员每天都用到的让程序跑起来的基础逻辑知识。

学习逻辑之前你需要先记住一些东西。这个练习我要求你一个星期完成，不要擅自修改日程，就算你烦得不得了，也要坚持下去。这个练习会让你背下来一系列的逻辑表格，这会让你更容易地完成后面的习题。

需要事先警告你的是：这件事情一开始一点乐趣都没有，你会一开始就觉得它很无聊乏味，但它的目的是教你程序员必须的一个重要技能——一些重要的概念是必须记住的，一旦你明白了这些概念，你会获得相当的成就感，但是一开始你会觉得它们很难掌握，就跟和乌贼摔跤一样，而等到某一天，你会刷的一下豁然开朗。你会从这些基础的记忆学习中得到丰厚的回报。

这里告诉你一个记住某样东西，而不让自己抓狂的方法：在一整天里，每次记忆一小部分，把你最需要加强的部分标记起来。不要想着在两小时内连续不停地背诵，这不会有什么好的结果。不管你花多长时间，你的大脑也只会留住你在前 15 或者 30 分钟内看过的东西。

取而代之，你需要做的是创建一些索引卡片，卡片有两列内容，正面写下逻辑关系，反面写下答案。你需要做到的结果是：拿出一张卡片来，看到正面的表达式，例如 “True or False”，你可以立即说出背面的结果是 “True”！坚持练习，直到你能做到这一点为止。

一旦你能做到这一点了，接下来你需要每天晚上自己在笔记本上写一份真值表出来。不要只是抄写它们，试着默写真值表，如果发现哪里没记住的话，就飞快地撇一眼这里的答案。这样将训练你的大脑让它记住整个真值表。

不要在这上面花超过一周的时间，因为你在后面的应用过程中还会继续学习它们。

逻辑术语

在 python 中我们会用到下面的术语（字符或者词汇）来定义事物的真(True)或者假(False)。计算机的逻辑就是在程序的某个位置检查这些字符或者变量组合在一起表达的结果是真是假。

- and 与
- or 或
- not 非
- != (not equal) 不等于
- == (equal) 等于
- >= (greater-than-equal) 大于等于
- <= (less-than-equal) 小于等于
- True 真
- False 假

其实你已经见过这些字符了，但这些词汇你可能还没见过。这些词汇(and, or, not)和你期望的效果其实是一样的，跟英语里的意思一模一样。

真值表

我们将使用这些字符来创建你需要记住的真值表。

NOT	True?
not False	True