



零基础学 Python

—— 老齐 (qiwsir) 的 Python 基础教程

老齐 (qiwsir) 著

路小磊 整理

目錄

第零部分 独上高楼，望尽天涯路	0
唠叨一些关于python的事情	0.1
第一部分 积小流，至江海	1
Python环境安装	1.1
集成开发环境（IDE）	1.2
数的类型和四则运算	1.3
啰嗦的除法	1.4
开始真正编程	1.5
初识永远强大的函数	1.6
玩转字符串(1)：基本概念、字符转义、字符串连接、变量与字符串关系	1.7
玩转字符串(2)	1.8
玩转字符串(3)	1.9
眼花缭乱的运算符	1.10
从if开始语句的征程	1.11
一个免费的实验室	1.12
有容乃大的list(1)	1.13
有容乃大的list(2)	1.14
有容乃大的list(3)	1.15
有容乃大的list(4)	1.16
list和str比较	1.17
画圈还不简单吗	1.18
再深点，更懂list	1.19
字典，你还记得吗？	1.20
字典的操作方法	1.21
有点简约的元组	1.22
一二三，集合了	1.23
集合的关系	1.24
Python数据类型总结	1.25
深入变量和引用对象	1.26
赋值，简单也不简单	1.27

坑爹的字符编码	1.28
做一个小游戏	1.29
不要红头文件(1): <code>open</code> , <code>write</code> , <code>close</code>	1.30
不要红头文件(2): <code>os.stat</code> , <code>closed</code> , <code>mode</code> , <code>read</code> , <code>readlines</code> , <code>readline</code>	1.31



老齐 (qiwsir) 著
路小磊 整理

声明

这个“零基础学Python”并不是我写的，原内容来自于[这里](#)，我只是将其Github的MarkDown内容整理成了Gitbook格式方便阅读。

我已经联系作者qiwsir，同意我整理，欢迎大家多多支持原作者。

原作者：老齐（qiwsir）

原作者Github：<https://github.com/qiwsir>

我：[Looly](#)

我的Github：<https://github.com/looly>

我的邮箱：loolly@gmail.com

为什么要开设此栏目

这个栏目的名称叫做“零基础学Python”。

现在网上已经有不少学习python的课程，其中也不乏精品。按理说，不缺少我这个基础类型的课程了。但是，我注意到一个问题，不管是课程还是出版的书，大多数是面向已经有一定编程经验的人写的或者讲的，也就是对这些朋友来讲，python已经不是他们的第一门高级编程语言。据我所知，目前国内很多大学都是将C之类的做为学生的第一门语言。

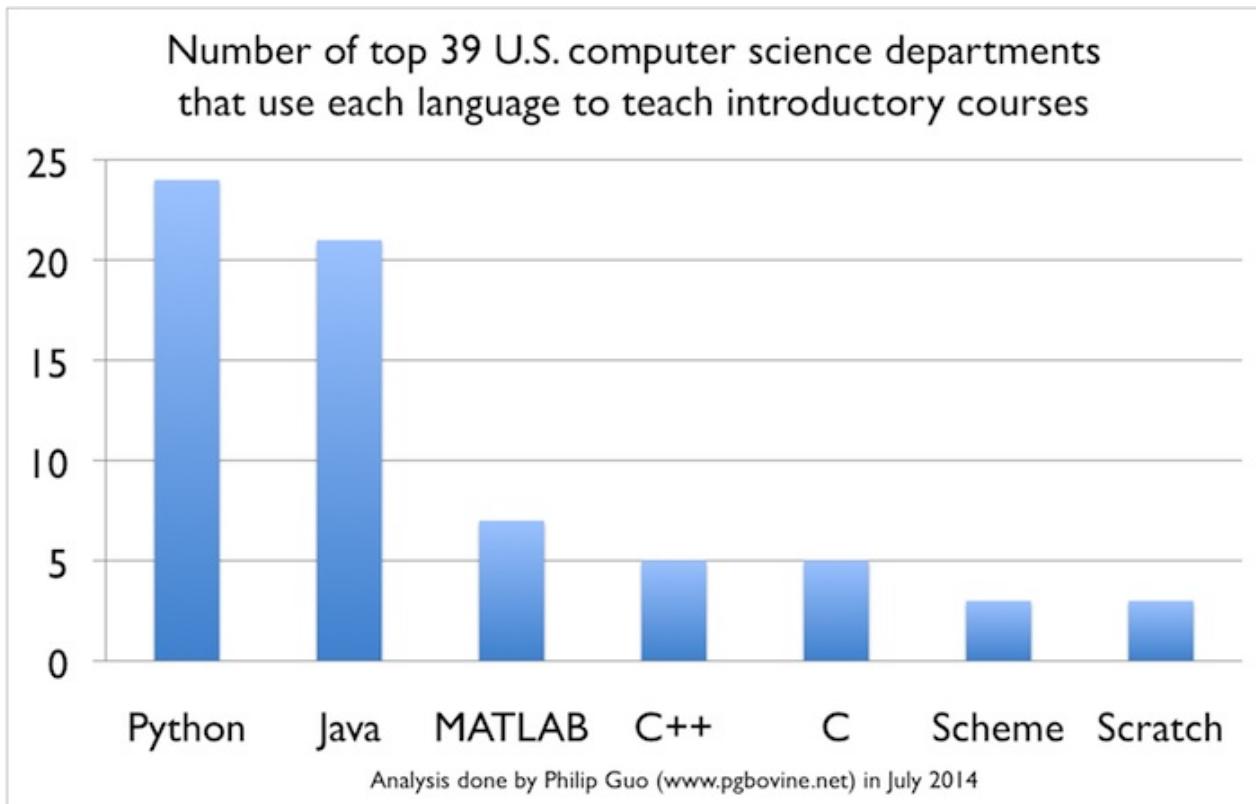
然而，在我看来，python是非常适合做为学习高级语言编程的第一门语言的。有一本书，名字叫《与孩子一起学编程》，这本书的定位，是将python定位为学习者学习的第一门高级编程语言。然而，由于读者对象是孩子，很多“成年人”不屑一顾，当然，里面的讲法与“实战”有

点距离，导致以“找工作”、“工作需要”为目标的学习者，认为这本书跟自己要学的方向相差甚远。

为了弥补那本书的缺憾，我在这里推出面向成年人——大学生、或者其他想学习程序但是没有任何编程基础的朋友——学习第一门编程高级语言的教程。将Python做为学习高级语言编程的第一门语言，其优势在于：

- 入门容易，避免了其它语言的繁琐。
- 更接近我们的自然语言和平常的思维方法。
- 学习完这门语言之后，能够直接“实战”——用在工作上。
- 学习完这门语言之后，能够顺利理解并学习其它语言。
- python本身功能强大，一门语言也可以打天下，省却了以后的学习成本。

下面的图示统计显示：Python现在成为美国名校中最流行的编程入门语言。



[点击这里看上图来源](#)

综上，我有了这样一个冲动，做一个栏目，面对零基础要学习Python的朋友，面对将python做为第一门高级语言的朋友。这就是开始本栏目的初衷。

Then God said: "Let there be light"; and there was light. And God saw that the light was good; and God separated the light from the darkness.

唠叨一些关于Python的事情

如同学习任何一种自然语言比如英语、或者其它编程语言比如汇编（这个我喜欢，可惜多年之后，已经好久没有用过了）一样，总要说一说有关这种语言的事情，有的可能就是八卦，越八卦的越容易传播。当然，以下的所有说法中，难免充满了自恋，因为你看不到说Python的坏话。这也好理解，如果要挑缺点是比较容易的事情，但是找优点，不管是对人还是对其它事务，都是困难的。这也许是人的劣根之所在吧，喜欢挑别人的刺儿，从而彰显自己在那方面高于对方。特别是在我们这个麻将文化充斥的神奇地方，更多了。

废话少说点（已经不少了），进入有关python的话题。

Python的昨天今天和明天

这个题目有点大了，似乎回顾过去、考察现在、张望未来，都是那些掌握方向的大人物（司机吗？）做的。那就让我们每个人都成为大人物吧。因为如果不回顾一下历史，似乎无法满足学习者的好奇心；如果不考察一下现在，学习者不放心（担心学了之后没有什么用途）；如果不张望一下未来，怎么能吸引（也算是一种忽悠吧）学习者或者未来的开发者呢？

Python的历史

Python的创始人为吉多·范罗苏姆（Guido van Rossum）。关于这个人开发这种语言的过程，很多资料里面都要记录下面的故事：

1989年的圣诞节期间，吉多·范罗苏姆为了在阿姆斯特丹打发时间，决心开发一个新的脚本解释程序，作为ABC语言的一种继承。之所以选中Python作为程序的名字，是因为他是一个蒙提·派森的飞行马戏团的爱好者。ABC是由吉多参加设计的一种教学语言。就吉多本人看来，ABC这种语言非常优美和强大，是专门为非专业程序员设计的。但是ABC语言并没有成功，究其原因，吉多认为是非开放造成的。吉多决心在Python中避免这一错误，并取得了非常好的效果，完美结合了C和其他一些语言。

这个故事我是从维基百科里面直接复制过来的，很多讲python历史的资料里面，也都转载这段。但是，在我来看，这段故事有点忽悠人的味道。其实，上面这段中提到的，吉多为了打发时间而决定开发python的说法，来自他自己的这样一段自述：

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office (a government-run research lab in Amsterdam) would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

(原文地址：<https://www.python.org/doc/essays/foreword/>)

首先，必须承认，这个哥们儿是一个牛人，非常牛的人。此处献上我的崇拜。

其次，做为刚刚开始学习python的朋友，可千万别认为python就是一个随随便便就做出来的东西，就是一个牛人一冲动搞出来的东西。人家也是站在巨人的肩膀上的。

第三，牛人在成功之后，往往把奋斗的过程描绘的比较简单，或者是谦虚？或者是让人听起来他更牛？反正，我们看最后结果的时候，很难感受过程中的酸甜苦辣。

不管怎么样，牛人在那时刻开始创立了python，而且，他更牛的在于具有现代化的思维：开放。通过Python社区，吸引来自世界各地的开发者，参与python的建设。在这里，请读者一定要联想到Linux和它的创始人芬兰人林纳斯·托瓦兹。两者都秉承“开放”思想，得到了来自世界各地开发者和应用者的欢呼和尊敬。也请大家再联想到另外一个在另外领域秉承开放思想的人——邓小平先生，他让一个封闭的破旧老水车有了更新。

请列位多所有倡导“开放”的牛人们表示敬意，是他们让这个世界更美好了。他们以行动诠释了热力学第二定律——“熵增原理”。

Python的现在

有一次与某软件公司一个号称是CTO的人谈话，他问我用什么语言开发，我说用Python，估计是我的英语发音不好吧（我这回真的谦虚了一把），他居然听成了Pascal（也是一种高级语言，现在很少用了，曾经是比较流行的教学语言）。呜呼，Python是小众吗？不是，是那家伙眼界不开阔！接触过不少号称CTO的，多数是有几年经验的程序员，并没有以国际视野来看待技术，当然，大牛的CTO还是不少的。总之，不要被外表忽悠了，“不看广告，看疗效”。

首先看一张最近一期的编程语言排行

2014年6月	2013年6月	变动	编程语言
1	1		C
2	2		Java
3	3		Objective-C
4	4		C++
5	6	▲	C#
6	7	▲	(Visual) Basic
7	5	▼	PHP
8	8		Python
9	10	▲	JavaScript

python在这个榜单中第8，也许看官心理在想：为什么我不去学那个排第一呢？如果您是一个零基础的学习者，我以多年的工作和教学经验正告：还是从入门比较容易的开始吧，python是这样的。等以后，完全可以拓展到其它语言。或许你又问了，php和vb是不是可以呢？他们排名比python靠前。回答是：当然可以。但是，学习一种入门的语言，要多方考虑，或许以后你就不想学别的，想用这个包打天下了，那就只有python。并且，还得看下面的信息：

根据Dice.com一项网上对20000名IT专业人士进行调查的结果：java类平均工资：91060美元；python类平均工资：90208美元；

不错，python程序员平均来讲，比java平均工资低，但看看差距，再看看两者的入门门槛，就知道，学习python绝对是一个性价比非常高的投资啦。

Python就是这样，默默地拓展着它的领域。

Python的未来

未来，要靠列为来做了，你学好了，用好了，未来它就光明了。它的未来在你手里。

Python的特点

很多高级语言都宣称自己是简单的、入门容易的，并且具有普适性的。真正做到这些的，不忽悠的，只有Python。有朋友做了一件衬衫，上面写着“生命有限，我用Python”，这说明什么？它有着简单、开发速度快，节省时间和精力的特点。因为它是开放的，有很多可爱的开发者（为开放社区做贡献的开发者，是最可爱的人），将常用的功能做好了，放在网上，谁都可以拿过来使用。这就是Python，这就是开放。

抄一段严格的描述，来自维基百科：

Python是完全面向对象的语言。函数、模块、数字、字符串都是对象。并且完全支持继承、重载、派生、多继承，有益于增强源代码的复用性。Python支持重载运算符，因此Python也支持泛型设计。相对于Lisp这种传统的函数式编程语言，Python对函数式设计只提供了有限的支持。有两个标准库（functools, itertools）提供了Haskell和Standard ML中久经考验的函数式程序设计工具。

虽然Python可能被粗略地分类为“脚本语言”（script language），但实际上一些大规模软件开发项目例如Zope、Mnet及BitTorrent，Google也广泛地使用它。Python的支持者较喜欢称它为一种高级动态编程语言，原因是“脚本语言”泛指仅作简单程序设计任务的语言，如shell script、VBScript等只能处理简单任务的编程语言，并不能与Python相提并论。

Python本身被设计为可扩充的。并非所有的特性和功能都集成到语言核心。Python提供了丰富的API和工具，以便程序员能够轻松地使用C、C++、Cython来编写扩充模块。Python编译器本身也可以被集成到其它需要脚本语言的程序内。因此，很多人还把Python作为一种“胶水语言”（glue language）使用。使用Python将其它语言编写的程序进行集成和封装。在Google内部的很多项目，例如Google Engine使用C++编写性能要求极高的部分，然后用Python或Java/Go调用相应的模块。

《Python技术手册》的作者马特利（Alex Martelli）说：“这很难讲，不过，2004年，Python已在Google内部使用，Google招募许多Python高手，但在这之前就已决定使用Python。他们的目的是尽量使用Python，在不得已时改用C++；在操控硬件的场合使用C++，在快速开发时候使用Python。”

可能里面有一些术语还不是很理解，没关系，只要明白：Python是一种很牛的语言，应用简单，功能强大，google都在使用。这就足够了，足够让你下决心学习了。

python哲学

Python之所以与众不同，还在于它强调一种哲学理念，用黑字表示强调吧：

Python的设计哲学是“优雅”、“明确”、“简单”。

Python开发者的哲学是“用一种方法，最好是只有一种方法来做一件事。在设计Python语言时，如果面临多种选择，Python开发者一般会拒绝花俏的语法，而选择明确没有或者很少有歧义的语法。由于这种设计观念的差异，Python源代码通常具备更好的可读性，并且能够支撑大规模的软件开发。这些准则被称为Python格言。

The Zen of Python

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!
```

上面的诗来自[Python官方](#)，已经把前面唠叨的东西做了精美的概括。有中译本，[看这里](#)，本文摘抄一种中文翻译：

优美胜于丑陋，明晰胜于隐晦
简单胜于复杂，复杂胜于繁芜
扁平胜于嵌套，稀疏胜于密集
可读性很重要。
虽然实用性比纯粹性更重要，
但特例并不足以把规则破坏掉。

错误状态永远不要忽略，
除非你明确地保持沉默，
直面多义，永不臆断。

最佳的途径只有一条，然而他并非显而易见——谁叫你不是荷兰人？

置之不理或许会比慌忙应对要好，
然而现在动手远比束手无策更好。

难以解读的实现不会是个好主意，
容易解读的或许才是。

名字空间就是个顶呱呱好的主意。

让我们想出更多的好主意！

准备

已经描述了python的美好，开始学啦，做好如下准备：

- 电脑，必须的。不管是什么操作系统。
- 上网，必须的。没有为什么。

除了这些，还有一条，非常非常重要，写在最后：这是自己的兴趣。

Python安装

任何高级语言都是需要一个自己的编程环境的，这就好比写字一样，需要有纸和笔，在计算机上写东西，也需要有文字处理软件，比如各种名称的OFFICE。笔和纸以及office软件，就是写东西的硬件或软件，总之，那些文字只能写在那个上边，才能最后成为一篇文章。那么编程也是，要有个什么程序之类的东西，要把程序写到那个上面，才能形成最后类似文章那样的东西。

刚才又有了一个术语——“程序”，什么是程序？本文就不讲了。如果列为观众不是很理解这个词语，请上网google一下。

注：推荐一种非常重要的学习方法

在我这里看文章的零基础朋友，乃至非零基础的朋友，不要希望在这里学到很多高深的python语言技巧。

“靠，那看你胡扯吗？”

非也。重要的是学会一些方法。比如刚才给大家推荐的“上网google一下”，就是非常好的学习方法。互联网的伟大之处，不仅仅在于打打游戏、看看养眼的照片或者各种视频之类的，当然，在某国很长时间互联网等于娱乐网，我衷心希望从读本文的朋友开始，互联网不仅仅是娱乐网，还是知识网和创造网。扯远了，拉回来。在学习过程中，如果遇到一点点疑问，都不要放过，思考一下、尝试一下之后，不管有没有结果，还都要google一下。

列为看好了，我上面写的很清楚，是google一下，不是让大家去用那个什么度来搜索，那个搜索是专用搜索八卦、假药、以及各种穿的很节俭的女孩子照片的。如果你真的要提高自己的技术视野并且专心研究技术问题，请用google。当然，我知道你在用的时候时候困难的，做为一个要在技术上有点成就的人，一定要学点上网的技术的，你懂得。

什么？你不懂？你的确是我的读者：零基础。那就具体来问我吧，不管是加入QQ群还是微博，都可以。

欲练神功，挥刀自宫。神功是有前提地。

要学python，不用自宫。python不用那么残忍的前提，但是，也需要安装点东西才能用。

所需要安装的东西，都在这个页面里面：www.python.org/downloads/

www.python.org是python的官方网站，如果你的英语足够使用，那么自己在这里阅读，可以获得非常多的收获。

在python的下载页面里面，显示出python目前有两大类，一类是python3.x.x，另外一类是python2.7.x。可以说，python3是未来，它比python2.7有进步。但是，现在，还有很多东西没有完全兼容python3。更何况，如果学了python2.7，对于python3，也只是某些地方的小变化了。

所以，我这里是用python2.7为例子来讲解的。

Linux系统的安装

看官所用的计算机是什么操作系统的？自己先弄懂。如果是Linux某个发行版，就跟我同道了。并且我恭喜你，因为以后会安装更多的一些python库（模块），在这种操作系统下，操作非常简单，当然，如果是iOS，也一样，因为都是UNIX下的蛋。只是widows有点另类了。

不过，没关系，python就是跨平台的。

我以ubutu 12.04为例，所有用这个操作系统的朋友（肯定很少啦），你们肯定会在shell中输入python，如果看到了>>>，并且显示出python的版本信息，恭喜你，因为你的系统已经自带了python的环境。的确，ubuntu内置了python环境。

我非要自己安装一遍不可。那就这么操作吧：

```
#下载源码，目前最新版本是2.7.8，如果以后换了，可以在下面的命令中换版本号
#源码也可以在网站上下载，具体见前述下载页面
wget http://www.python.org/ftp/python/2.7.8/Python-2.7.8.tgz

#解压源码包
tar -zxvf Python-2.7.8.tgz

#编译
cd Python-2.7.8
./configure --prefix=/usr/local      #指定了目录
make&&make install
```

以上步骤，是我从网上找来的，供参考。因为我的机器早就安装了，不想折腾。安装好之后，进入shell，输入python，会看到如下：

```
qw@qw-Latitude-E4300:~$ python
Python 2.7.6 (default, Nov 13 2013, 19:24:16)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

恭喜你，安装成功了。我用的是python2.7.6，或许你的版本号更高。

windows系统的安装

到[下载页面里面](#)找到windows安装包，下载之，比如下载了这个文件：python-2.7.8.msi。然后就是不断的“下一步”，即可完成安装。

特别注意，安装完之后，需要检查一下，在环境变量是否有python。

如果还不知道什么是windows环境变量，以及如何设置。不用担心，请google一下，搜索：“windows 环境变量”就能找到如何设置了。

以上搞定，在cmd中，输入python，得到跟上面类似的结果，就说明已经安装好了。

Mac OS X系统的安装

其实根本就不用再写怎么安装了，因为用Mac OS X的朋友，肯定是高手中的高高手了，至少我一直很敬佩那些用Mac OS X并坚持没有更换为windows的。麻烦用Mac OS X的朋友自己网上搜吧，跟前面unbutu差不多。

如果按照以上方法，顺利安装成功，只能说明幸运，无它。如果没有安装成功，这是提高自己的绝佳机会，因为只有遇到问题才能解决问题，才能知道更深刻的道理，不要怕，有google，它能帮助列为看官解决所有问题。当然，加入QQ群或者通过微博，问我也可以。

就一般情况而言，Linux和Mac OS x系统都已经安装了某种python的版本，打开就可以使用。但是windows是肯定不安装的。除了可以用上面所说的方法安装，还有一个更省事的方法，就是安装：ActivePython

用ActivePython安装

这个ActivePython是一个面向好多种操作系统的Python 套件,它包含了一个完整的 Python 发布、一个适用于 Python 编程的 IDE 以及一些 Python的。有兴趣的看官可以到其官网浏览：<http://www.activestate.com>

用源码安装

python是开源的，它的源码都在网上。有高手朋友，如果愿意用源码来安装，亦可，请到：<https://www.python.org/ftp/python/>，下载源码安装。

简单记录一下我的安装方法（我是在linux系统中做的）：

1. 获得root权限
2. 到上述地址下载某种版本的python: wget <https://www.python.org/ftp/python/2.7.8/Python-2.7.8.tgz>

2.7.8.tgz

3. 解压缩：tar xfz Python-2.7.8.tgz
4. 进入该目录：cd Python-2.7.8
5. 配置： ./configure
6. 在上述文件夹内运行：make，然后运行：make install
7. 祝你幸运
8. 安装完毕

OK!已经安装好之后，马上就可以开始编程了。

最后喊一句在一个编程视频课程广告里面看到的口号，很有启发：“我们程序员，不求通过，但求报错”。

集成开发环境(IDE)

当安装好python之后，其实就已经可以进行开发了。下面我们开始写第一行python代码。

值得纪念的时刻：Hello world

如果是用windows，请打开CMD，并执行python。

如果是UNIX类的，就运行shell，并执行python。

都会出现如下内容：

```
Python 2.7.6 (default, Nov 13 2013, 19:24:16)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

在>>>后面输入下面内容，并按回车。这就是见证奇迹的时刻。从这一刻开始，一个从来不懂编程的你，就跨入了程序员行列，不管你的工作是不是编程，你都已经是程序员了，其标志就是你已经用代码向这个世界打招呼了。

```
>>> print "Hello, World"
Hello, World
```

每个程序员，都曾经经历过这个伟大时刻，不经历这个伟大时刻的程序员不是伟大的程序员。为了纪念这个伟大时刻，理解其伟大之所在，下面执行分解动作：

说明：在下面的分解动作中，用到了一个符号：#，就是键盘上数字3上面的那个井号，通过按下shift，然后按3，就得到了。这个符号，在python编程中，表示注释。所谓注释，就是在计算机不执行，只是为了说明某行语句表达什么意思。

```
#看到“>>>”符号，表示python做好了准备，当代你向她发出指令，让她做什么事情
>>>

#print，意思是打印。在这里也是这个意思，是要求python打印什么东西
>>> print

#"Hello,World"是打印的内容，注意，量变的双引号，都是英文状态下的。引号不是打印内容，它相当于一个包裹，把
>>> print "Hello, World"

#上面命令执行的结果。python接收到你要求她做的事情：打印Hello,World，于是她就老老实实地执行这个命令，输出
Hello, World
```

祝贺，伟大的程序员。

笑一笑：有一个程序员，自己感觉书法太烂了，于是立志继承光荣文化传统，购买了笔墨纸砚。在某天，开始练字。将纸铺好，拿起笔蘸足墨水，挥毫在纸上写下了两个打字：Hello World

从此，进入了程序员行列，但是，看官有没有感觉，程序员用的这个工具，就是刚才打印Hello,World的那个cmd或者shell，是不是太简陋了？你看美工妹妹用的Photoshop，行政妹妹用的word，出纳妹妹用的Excel，就连坐在老板桌后面的那个家伙还用一个PPT播放自己都不相信的新理念呢，难道我们伟大的程序员，就用这么简陋的工具写出旷世代码吗？

当然不是。软件是谁开发的？程序员。程序员肯定会先为自己打造好用的工具，这也叫做近水楼台先得月。

IDE就是程序员的工具。

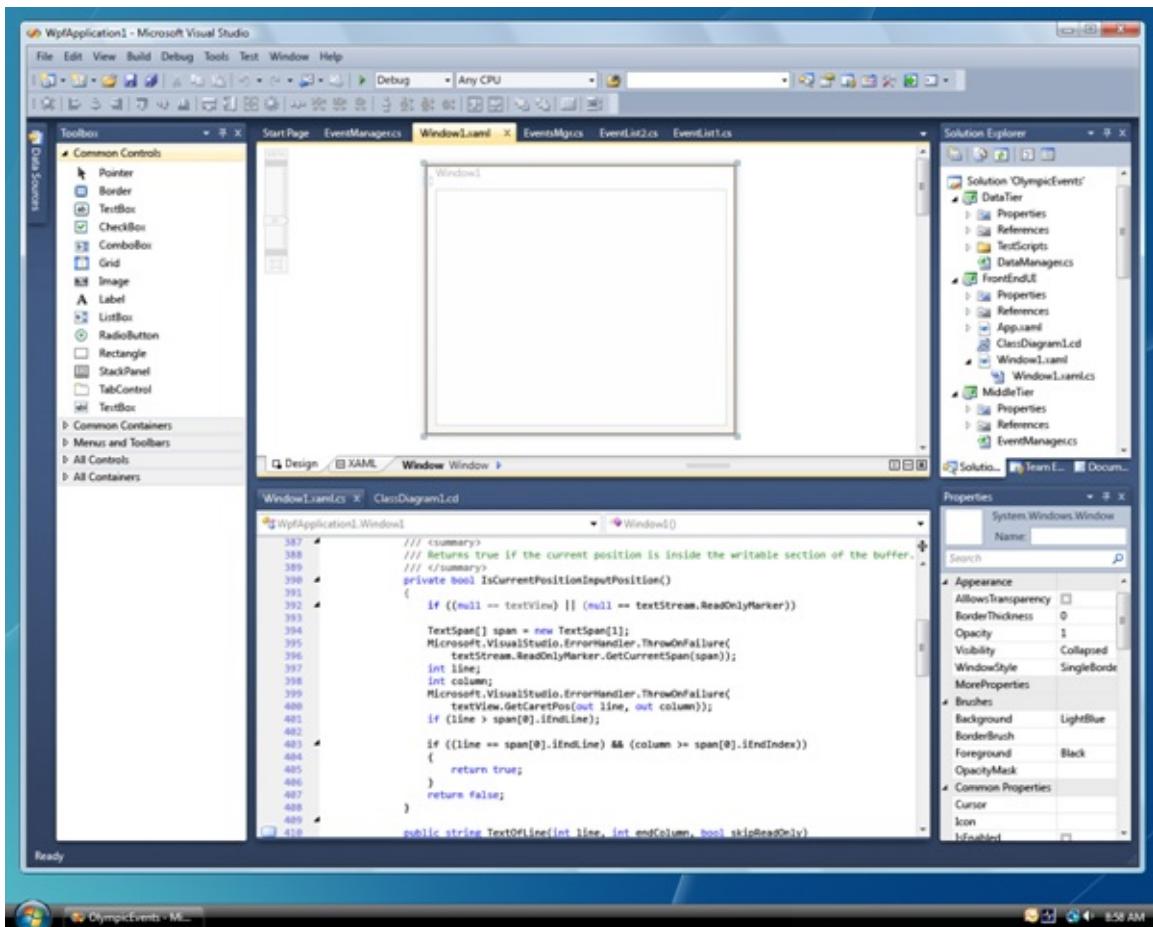
集成开发环境

IDE的全称是：Integrated Development Environment，简称IDE，也稱為Integration Design Environment、Integration Debugging Environment，翻译成中文叫做“集成开发环境”，在台湾那边叫做“整合開發環境”。它是一種輔助程式開發人員開發軟體的應用軟體。

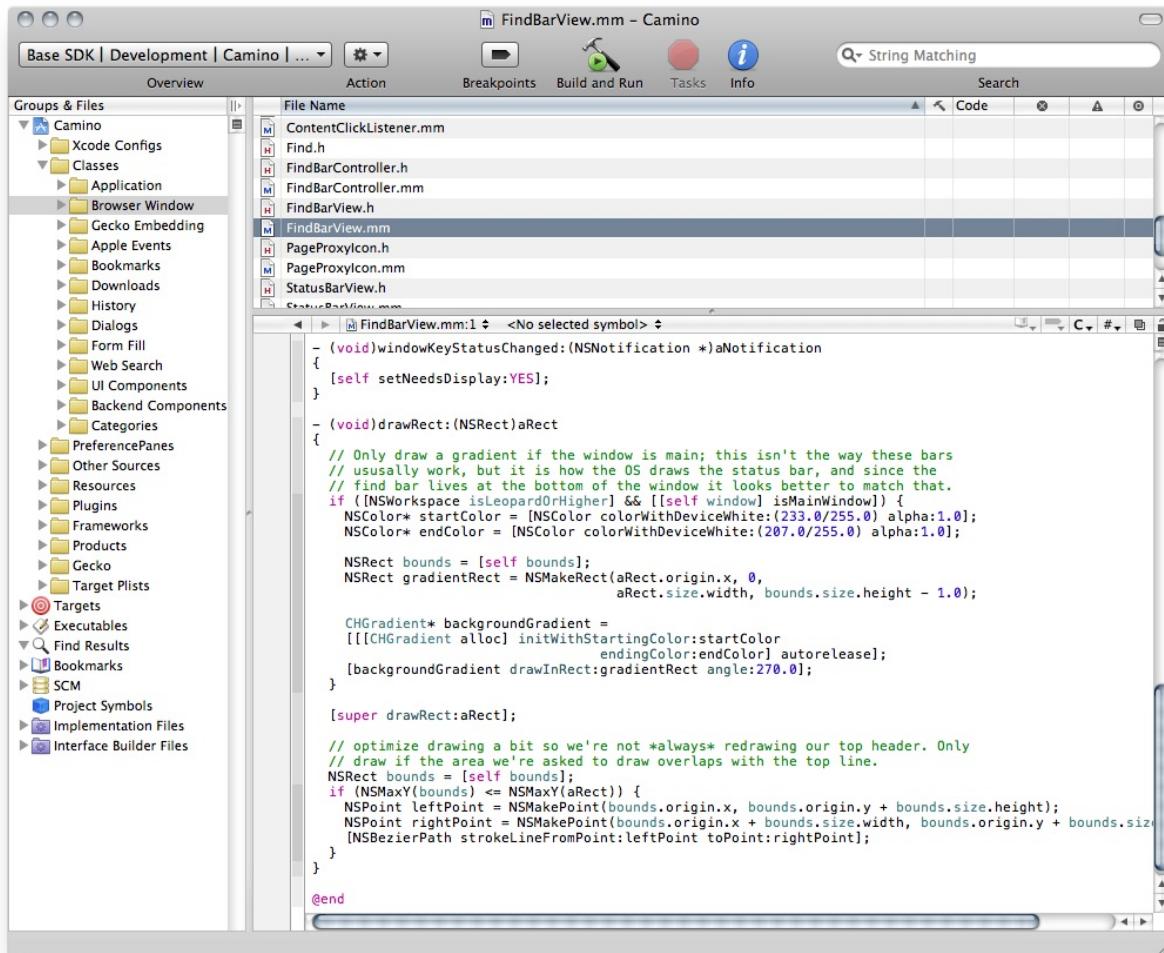
下面就直接抄维基百科上的说明了：

IDE通常包括程式語言編輯器、自動建立工具、通常還包括除錯器。有些IDE包含編譯器／直譯器，如微软的Microsoft Visual Studio，有些则不包含，如Eclipse、SharpDevelop等，这些IDE是通过调用第三方编译器来实现代码的编译工作的。有时IDE还会包含版本控制系统和一些可以设计图形用户界面的工具。许多支援物件导向的现代化IDE还包括了类别浏览器、物件检视器、物件結構圖。虽然目前有一些IDE支援多種程式語言（例如Eclipse、NetBeans、Microsoft Visual Studio），但是，一般而言，IDE主要还是針對特定的程式語言而量身打造（例如Visual Basic）。

看不懂，没关系，看图，认识一下，混个脸熟就好了。所谓有图有真相。



上面的图显示的是微软的提供的名字叫做Microsoft Visual Studio的IDE。用C#进行编程的程序员都用它。



上图是在苹果电脑中出现的名叫XCode的IDE。

要想了解更多IDE的信息，推荐阅读维基百科中的词条

- 英文词条：[Integrated development environment](#)
- 中文词条：[集成开发环境](#)

Python的IDE

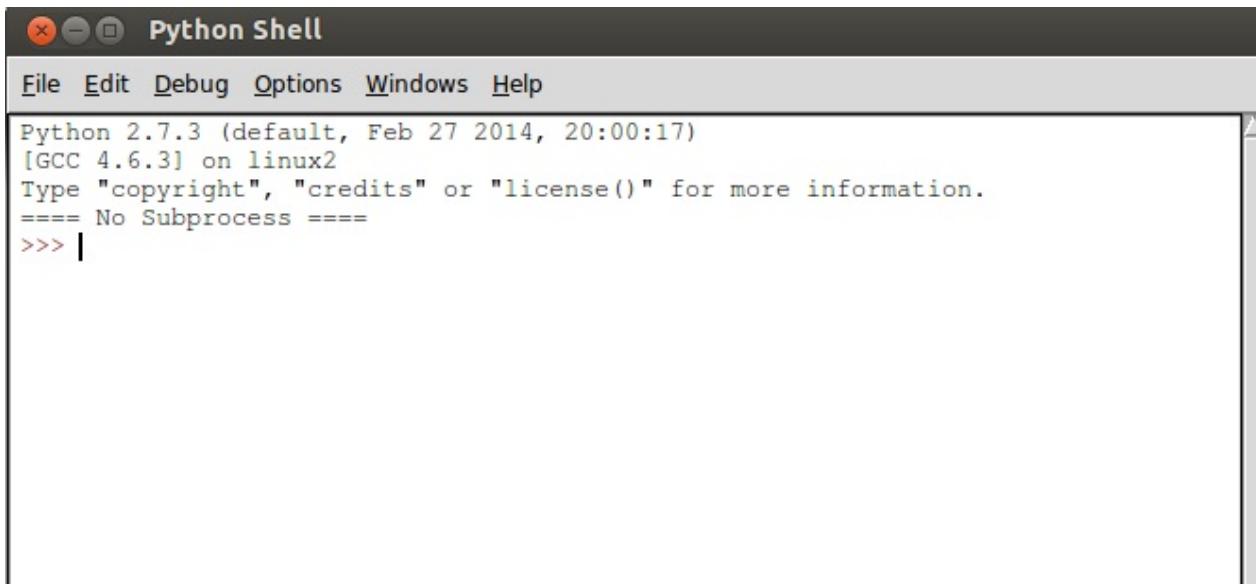
google一下：python IDE，会发现，能够进行python编程的IDE还真的不少。东西一多，就开始无所适从了。所有，有不少人都问用哪个IDE好。可以看看[这个提问](#)，还列出了众多IDE的比较。

顺便向列为看客推荐一个非常好的开发相关网站：[stackoverflow.com](#) 在这里可以提问，可以查看答案。一般如果有问题，先在这里查找，多能找到非常满意的结果，至少有很大启发。在某国有时候有地方可能不能访问，需要科学上网。好东西，一定不会让你容易得到，也不会让任何人都得到。

那么做为零基础的学习者，用什么好呢？

既然是零基础，就别瞎折腾了，就用Python自带的IDLE。原因就是：简单。

Windows的朋友操作：“开始”菜单->“所有程序”->“Python 2.x”->“IDLE (Python GUI)”来启动IDLE。启动之后，大概看到这样一个图



注意：看官所看到的界面中显示版本跟这个图不同，因为安装的版本区别。大致模样差不多。

其它操作系统的用户，也都能在找到idle这个程序，启动之后，跟上面一样的图。

后面我们所有的编程，就在这里完成了。这就是伟大程序员用的第一个IDE。

磨刀不误砍柴工。IDE已经有了，伟大程序员就要开始从事伟大的编程工作了。且看下回分解。

For I am not ashamed of the gospel; it is the power of God for salvation to everyone who has faith, to the Jew first and also to the Greek. For in it the righteousness of God is revealed through faith for faith; so it is written, "The one who is righteous will live by faith"

用 Python 计算

一提到计算机，当然现在更多人把她叫做电脑，这两个词都是指computer。不管什么，只要提到她，普遍都会想到她能够比较快地做加减乘除，甚至乘方开方等。乃至于，有的人在口语中区分不开计算机和计算器。

那么，做为零基础学习这，也就从计算小学数学题目开始吧。因为从这里开始，数学的基础知识列为肯定过关了。

复习

还是先来重温一下伟大时刻，打印hello world.

打开电脑，让python idle运行起来，然后输入：

```
>>> print 'Hello, World'  
Hello, World
```

细心的看官，是否注意到，我在这里用的是单引号，[上次](#)用的是双引号。两者效果一样，也就是在这种情况下，单引号和双引号是一样的效果，一定要是成对出现的，不能一半是单引号，另外一半是双引号。

四则运算

按照下面要求，在ide中运行，看看得到的结果和用小学数学知识运算之后得到的结果是否一致

```

>>> 2+5
7
>>> 5-2
3
>>> 10/2
5
>>> 5*2
10
>>> 10/5+1
3
>>> 2*3-4
2

```

上面的运算中，分别涉及到了四个运算符号：加(+)、减(-)、乘(*)、除(/)

另外，我相信看官已经发现了一个重要的公理：

在计算机中，四则运算和小学数学中学习过的四则运算规则是一样的

要不说人是高等动物呢，自己发明的东西，一定要继承自己已经掌握的知识，别跟自己的历史过不去。伟大的科学家们，在当初设计计算机的时候就想到列为现在学习的需要了，一定不能让后世子孙再学新的运算规则，就用小学数学里面的好了。感谢那些科学家先驱者，泽被后世。

下面计算三个算术题，看看结果是什么

- 4 + 2
- 4.0 + 2
- 4.0 + 2.0

看官可能愤怒了，这么简单的题目，就不要劳驾计算机了，太浪费了。

别着急，还是要在ide中运算一下，然后看看结果，有没有不一样？要仔细观察哦。

```

>>> 4+2
6
>>> 4.0+2
6.0
>>> 4.0+2.0
6.0

```

不一样的地方是：第一个式子结果是6，后面两个是6.0。

现在我们就要引入两个数据类型：整数和浮点数

对这两个的定义，不用死记硬背，google一下。记住爱因斯坦说的那句话：书上有的我都不记忆（是这么的说？好像是，大概意思，反正我也不记忆）。后半句他没说，我补充一下：忘了就google。

定义1：类似4、-2、129486655、-988654、0这样形式的数，称之为整数 定义2：

类似4.0、-2.0、2344.123、3.1415926这样形式的数，称之为浮点数

比较好理解，整数，就是小学学过的整数；浮点数，就是小数。如果整数写成小数形式，比如4写成4.0，也就变成了浮点数。

爱学习，就要有探索精神。看官在网上google一下整数，会发现还有另外一个词：长整数（型）。顾名思义，就是比较长的整数啦。在有的语言中，把这个做为单独一类区分开，但是，在python中，我们不用管这个了。只要是整数，就只是整数，不用区分长短（以前版本区分），因为区分没有什么意思，而且跟小学学过的数学知识不协调。

还有一个问题，需要向看官交代一下，眼前可能用不到，但是会总有一些人用这个来忽悠你，当他忽悠你的时候，下面的知识就用到了。

整数溢出问题

这里有一篇专门讨论这个问题的文章，推荐阅读：[整数溢出](#)

对于其它语言，整数溢出是必须正视的，但是，在python里面，看官就无忧愁了，原因就是python为我们解决了这个问题，请阅读拙文：[大整数相乘](#)

ok!看官可以在IDE中实验一下大整数相乘。

```
>>> 123456789870987654321122343445567678890098876*123345566778999009987654333238766544334
152278477193527562870044352587576277277562328362032444339019158937017801601677976183816L
```

看官是幸运的，python解忧愁，所以，选择学习python就是珍惜光阴了。

上面计算结果的数字最后有一个L，就表示这个数是一个长整数，不过，看官不用管这点，反正是python为我们搞定了。

在结束本节之前，有两个符号需要看官牢记（不记住也没关系，可以随时google，只不过记住后使用更方便）

- 整数，用int表示，来自单词：integer
- 浮点数，用float表示，就是单词：float

可以用一个命令：type(object)来检测一个数是什么类型。

```

>>> type(4)
<type 'int'>    #4是int，整数
>>> type(5.0)
<type 'float'>  #5.0是float，浮点数
type(988776544222112233445566778899887766544322113334445566677788998776543222344556678
<type 'long'>   #是长整数，也是一个整数

```

几个常见函数

在这里就提到函数，因为这个东西是经常用到的。什么是函数？如果看官不知道此定义，可以去google。貌似是初二数学讲的了。

有几个常用的函数，列一下，如果记不住也不要紧，知道有这些就好了，用的时候就google。

求绝对值

```

>>> abs(10)
10
>>> abs(-10)
10
>>> abs(-1.2)
1.2

```

四舍五入

```

>>> round(1.234)
1.0
>>> round(1.234, 2)
1.23

>>> #如果不清楚这个函数的用法，可以使用下面方法看帮助信息
>>> help(round)

Help on built-in function round in module __builtin__:

round(...)
    round(number[, ndigits]) -> floating point number

    Round a number to a given precision in decimal digits (default 0 digits).
    This always returns a floating point number. Precision may be negative.

```

幂函数

```
>>> pow(2, 3)      #2的3次方
8
```

math模块（对于模块可能还有点陌生，不过不要紧，先按照下面代码实验一下，慢慢就理解了）

```
>>> import math      #引入math模块
>>> math.floor(32.8) #取整，不是四舍五入
32.0
>>> math.sqrt(4)     #开平方
2.0
```

总结

- python里的加减乘除按照小学数学规则执行
- 不用担心大整数问题，python会自动处理
- `type(object)`是一个有用的东西

"I give you a new commandment, that you love one another. Just as I have loved you, you also should love one another. By this everyone will know that you are my disciples, if you have love for one another."(JOHN14:34-35)

啰嗦的除法

除法啰嗦，不仅是python。

整数除以整数

看官请在进入python交互模式之后（以后在本教程中，可能不再重复这类的叙述，只要看到
`>>>`，就说明是在交互模式下，这个交互模式，看官可以在ide中，也可以像我一样直接在
shell中运行python进入交互模式），练习下面的运算：

```
>>> 2/5
0
>>> 2.0/5
0.4
>>> 2/5.0
0.4
>>> 2.0/5.0
0.4
```

看到没有？麻烦出来了（这是在python2.x中），如果从小学数学知识除法，以上四个运算结果都应该是0.4。但我们看到的后三个符合，第一个居然结果是0。why?

因为，在python（严格说是python2.x中，python3会有所变化，具体看官要了解，可以去
google）里面有一个规定，像`2/5`中的除法这样，是要取整（就是去掉小数，但不是四舍五
入）。2除以5，商是0（整数），余数是2（整数）。那么如果用这种形式：`2/5`，计算结果就
是商那个整数。或者可以理解为：整数除以整数，结果是整数（商）。

继续实验，验证这个结论：

```
>>> 5/2
2
>>> 6/3
2
>>> 5/2
2
>>> 6/2
3
>>> 7/2
3
>>> 8/2
4
>>> 9/2
4
```

注意：这里是得到整数商，而不是得到含有小数位的结果后“四舍五入”。例如 $5/2$ ，得到的是商2，余数1，最终 $5/2=2$ 。并不是对2.5进行四舍五入。

浮点数与整数相除

列位看官注意，这个标题和上面的标题格式不一样，上面的标题是“整数除以整数”，如果按照风格一贯制的要求，本节标题应该是“浮点数除以整数”，但没有，现在是“浮点数与整数相除”，其含义是：

假设： x 除以 y 。其中 x 可能是整数，也可能是浮点数； y 可能是整数，也可能是浮点数。

出结论之前，还是先做实验：

```
>>> 9.0/2
4.5
>>> 9/2.0
4.5
>>> 9.0/2.0
4.5
>>> 8.0/2
4.0
>>> 8/2.0
4.0
>>> 8.0/2.0
4.0
```

归纳，得到规律：不管是被除数还是除数，只要有一个数是浮点数，结果就是浮点数。所以，如果相除的结果有余数，也不会像前面一样了，而是要返回一个浮点数，这就跟在数学上学习的结果一样了。

```
>>> 10.0/3
3.3333333333333335
```

这个是不是就有点搞怪了，按照数学知识，应该是 $3.33333\dots$ ，后面是3的循环了。那么你的计算机就停不下来了，满屏都是3。为了避免这个，python武断终结了循环，但是，可悲的是没有按照“四舍五入”的原则终止。

关于无限循环小数问题，小学都学习了，但是这可不是一个简单问题，看看[维基百科的词条：0.999...](#)，会不会有深入体会呢？

总之，要用python，就得遵循她的规定，前面两条规定已经明确了。

补充一个资料，供有兴趣的朋友阅读：[浮点数算法：争议和限制](#)

说明：以上除法规则，是针对python2，在python3中，将 $5/2$ 和 $5.0/2$ 等同起来了。不过，如果要得到那个整数部分的上，可以用另外一种方式：地板除。

```
>>> 9/2
4
>>> 9//2
4
```

python总会要提供多种解决问题的方案的，这是她的风格。

引用模块解决除法--启用轮子

python之所以受人欢迎，一个很重要的原因，就是轮子多。这是比喻啦。就好比你要跑的快，怎么办？光天天练习跑步是不行滴，要用轮子。找辆自行车，就快了很多。还嫌不够快，再换电瓶车，再换汽车，再换高铁...反正你可以选择的很多。但是，这些让你跑的快的东西，多数不是你自己造的，是别人造好了，你来用。甚至两条腿也是感谢父母恩赐。正是因为轮子多，可以选择的多，就可以以各种不同速度享受了。

python就是这样，有各种各样别人造好的轮子，我们只需要用。只不过那些轮子在python里面的名字不叫自行车、汽车，叫做“模块”，有人承接别的语言的名称，叫做“类库”、“类”。不管叫什么名字吧。就是别人造好的东西我们拿过来使用。

怎么用？可以通过两种形式用：

- 形式1：`import module-name`。import后面跟空格，然后是模块名称，例如：`import os`
- 形式2：`from module1 import module11`。module1是一个大模块，里面还有子模块module11，只想用module11，就这么写了。比如下面的例子：

不啰嗦了，实验一个：

```

>>> from __future__ import division
>>> 5/2
2.5
>>> 9/2
4.5
>>> 9.0/2
4.5
>>> 9/2.0
4.5

```

注意了，引用了一个模块之后，再做除法，就不管什么情况，都是得到浮点数的结果了。

这就是轮子的力量。

关于余数

前面计算 $5/2$ 的时候，商是2，余数是1

余数怎么得到？在python中（其实大多数语言也都是），用% 符号来取得两个数相除的余数。

实验下面的操作：

```

>>> 5%2
1
>>> 9%2
1
>>> 7%3
1
>>> 6%4
2
>>> 5.0%2
1.0

```

符号：% ，就是要得到两个数（可以是整数，也可以是浮点数）相除的余数。

前面说python有很多人见人爱的轮子（模块），她还有丰富的内建函数，也会帮我们做不少事情。例如函数 divmod()

```

>>> divmod(5, 2) #表示5除以2，返回了商和余数
(2, 1)
>>> divmod(9, 2)
(4, 1)
>>> divmod(5.0, 2)
(2.0, 1.0)

```

四舍五入

最后一个了，一定要坚持，今天的确有点啰嗦了。要实现四舍五入，很简单，就是内建函数：`round()`

动手试试：

```
>>> round(1.234567, 2)
1.23
>>> round(1.234567, 3)
1.235
>>> round(10.0/3, 4)
3.3333
```

简单吧。越简单的时候，越要小心，当你遇到下面的情况，就有点怀疑了：

```
>>> round(1.2345, 3)
1.234          #应该是：1.235
>>> round(2.235, 2)
2.23          #应该是：2.24
```

哈哈，我发现了python的一个bug，太激动了。

别那么激动，如果真的是bug，这么明显，是轮不到我的。为什么？具体解释看这里，下面摘录官方文档中的一段话：

Note: The behavior of `round()` for floats can be surprising: for example, `round(2.675, 2)` gives 2.67 instead of the expected 2.68. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See [Floating Point Arithmetic: Issues and Limitations](#) for more information.

原来真的轮不到我。（垂头丧气状。）

似乎除法的问题到此要结束了，其实远远没有，不过，做为初学者，至此即可。还留下了很多话题，比如如何处理循环小数问题，我肯定不会让有探索精神的朋友失望的，在我的github中有这样一个轮子，如果要深入研究，[可以来这里尝试](#)。

开始真正编程

通过对四则运算的学习，已经初步接触了Python中内容，如果看官是零基础的学习者，可能有点迷惑了。难道在IDE里面敲几个命令，然后看到结果，就算编程了？这也不是那些能够自动运行的程序呀？

的确。到目前位置，还不能算编程，只能算是会用一些指令（或者叫做命令）来做点简单的工作。并且看官所在的那个IDE界面，也是输入指令用的。

列位稍安勿躁，下面我们就学习如何编写一个真正的程序。工具还是那个IDLE，但是，请大家谨记，对于一个真正的程序来讲，用什么工具是无所谓的，只要能够把指令写进去，比如用记事本也可以。

我去倒杯茶，列位先认真读一读下面一段，关于程序的概念，内容来自维基百科：

- 先阅读一段英文的：[computer program and source code](#)，看不懂不要紧，可以跳过去，直接看下一条。

A computer program, or just a program, is a sequence of instructions, written to perform a specified task with a computer.[1] A computer requires programs to function, typically executing the program's instructions in a central processor.[2] The program has an executable form that the computer can use directly to execute the instructions. The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled), enables a programmer to study and develop its algorithms. A collection of computer programs and related data is referred to as the software.

Computer source code is typically written by computer programmers.[3] Source code is written in a programming language that usually follows one of two main paradigms: imperative or declarative programming. Source code may be converted into an executable file (sometimes called an executable program or a binary) by a compiler and later executed by a central processing unit. Alternatively, computer programs may be executed with the aid of an interpreter, or may be embedded directly into hardware.

Computer programs may be ranked along functional lines: system software and application software. Two or more computer programs may run simultaneously on one computer from the perspective of the user, this process being known as multitasking.

- [计算机程序](#)

计算机程序（Computer Program）是指一组指示计算机或其他具有信息处理能力装置每一步动作的指令，通常用某种程序设计语言编写，运行于某种目标体系结构上。打个比方，一个程序就像一个用汉语（程序设计语言）写下的红烧肉菜谱（程序），用于指导懂汉语和烹饪手法的人（体系结构）来做这个菜。通常，计算机程序要经过编译和链接而成为一种人们不易看清而计算机可解读的格式，然后运行。未经编译就可运行的程序，通常称之为脚本程序（script）。

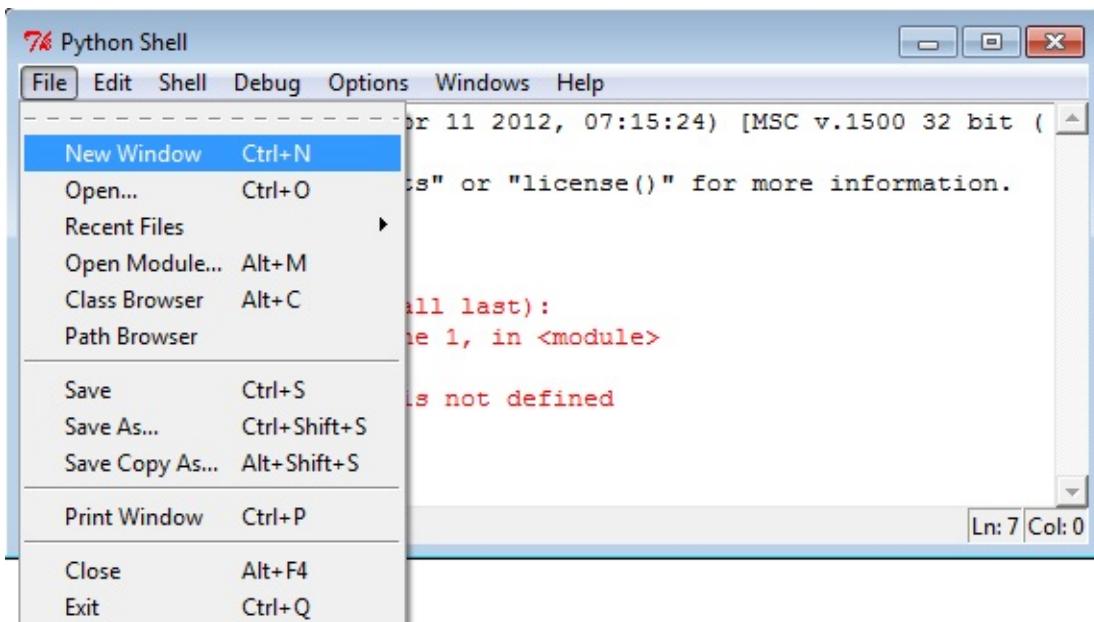
碧螺春，是我最喜欢的了。有人要送礼给我，请别忘记了。难道我期望列位看官会送吗？哈哈哈

废话少说，开始说程序。程序，简而言之，就是指令的集合。但是，有的程序需要编译，有的不需要。python编写的程序就不需要，因此她也被称之为脚本程序。特别提醒列位，不要认为编译的好，不编译的就不好；也不要认为编译的就“高端”，不编译的就属于“低端”。有一些做了很多年程序的程序员或者其它什么人，可能会有这样的想法，这是毫无根据的。

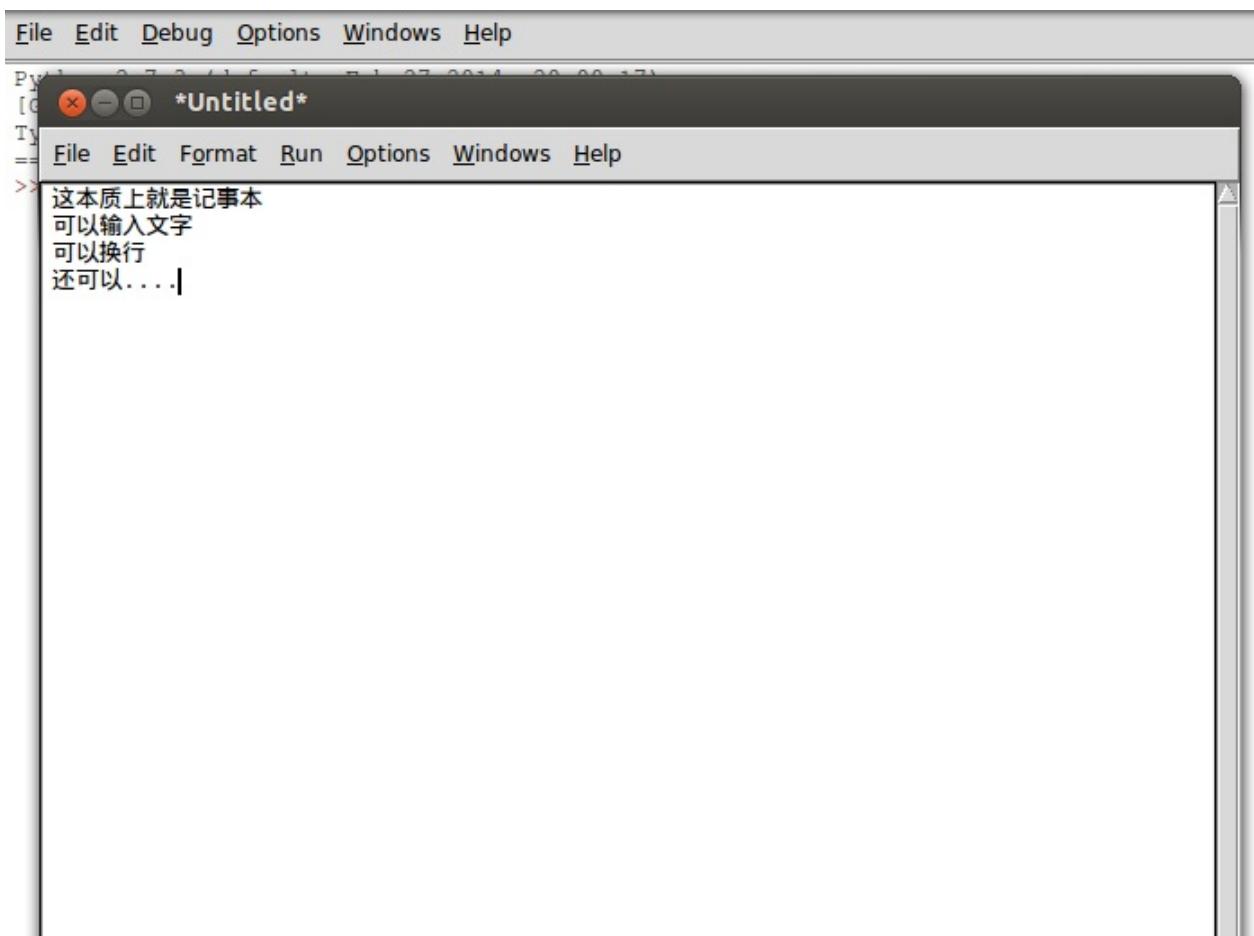
不争论。用得妙就是好。

用 IDLE 的编程环境

操作：File->New window



这样，就出现了一个新的操作界面，在这个界面里面，看不到用于输入指令的提示符：`>>>`，这个界面有点像记事本。说对了，本质上就是一个记事本，只能输入文本，不能直接在里面贴图片。



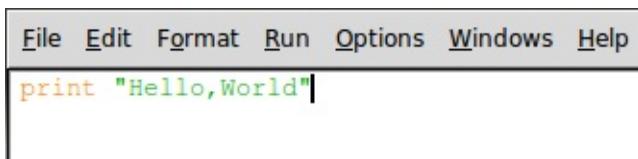
写两个大字：Hello,World

Hello,World.是面向世界的标志，所以，写任何程序，第一句一定要写这个，因为程序员是面向世界的，绝对不畏缩在某个局域网内，所以，所以看官要会科学上网，才能真正与世界Hello。

直接上代码，就这么一行即可。

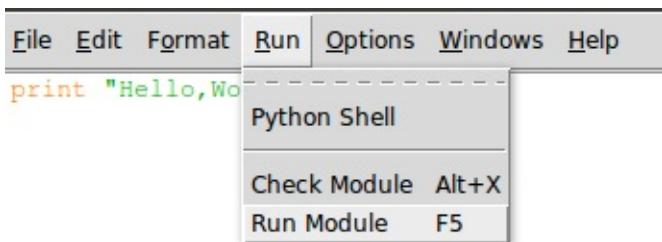
```
print "Hello,World"
```

如下图的样式



前面说过了，程序就是指令的集合，现在，这个程序里面，就一条指令。一条指令也可以成为集合。

注意观察，菜单上有一个RUN，点击这个菜单，在下拉的里面选择Run Moudle



会弹出对话框，要求把这个文件保存，这就比较简单了，保存到一个位置，看官一定要记住这个位置，并且取个文件名，文件名是以.py为扩展名的。

都做好之后，点击确定按钮，就会发现在另外一个带有>>>的界面中，就自动出来了Hello,World两个大字。

成功了吗？成功了也别兴奋，因为还没有到庆祝的时候。

在这种情况下，我们依然是在IDLE的环境中实现了刚才那段程序的自动执行，如果脱离这个环境呢？

下面就关闭IDLE，打开shell(如果看官在使用苹果的 Mac OS 操作系统或者某种linux发行版的操作系统，比如我使用的是ubuntu)，或者打开cmd(windows操作系统的用户，特别提醒用windows的用户，使用windows不是你的错，错就错在你只会使用鼠标点来点去，而不想也不会使用命令，更不想也不会使用linux的命令，还梦想成为优秀程序员。)，通过命令的方式，进入到你保存刚才的文件目录。

下图是我保存那个文件的地址，我把那个文件命名为105.py，并保存在一个文件夹中。

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$
```

然后在这个shell里面，输入：python 105.py

上面这句话的含义就是告诉计算机，给我运行一个python语言编写的程序，那个程序文件的名称是105.py

我的计算机我做主。于是它给我乖乖地执行了这条命令。如下图：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python 105.py
Hello,World
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ |
```

还在沉默？可以欢呼了，德国队7:1胜巴西队，列看官中，不管是德国队还是巴西队的粉丝，都可以欢呼，因为你在程序员道路上迈出了伟大的第二步。顺便预测一下，本届世界杯最终冠军应该是：中国队。（还有这么扯的吗？）

解一道题目

请计算： $19+2^4-8/2$

代码如下：

```
#coding:utf-8

"""
请计算：19+2^4-8/2
"""

a = 19+2**4-8/2
print a
```

提醒初学者，别复制这段代码，而是要一个字一个字的敲进去。然后保存(我保存的文件名是:105-1.py)。

在shell或者cmd中，执行：python (文件名.py)

执行结果如下图：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python 105-1.py
23
```

上面代码中，第一行，不能少，本文件是能够输入汉字的，否则汉字无法输入。

好像还是比较简单。

别着急。复杂的在后面呢。

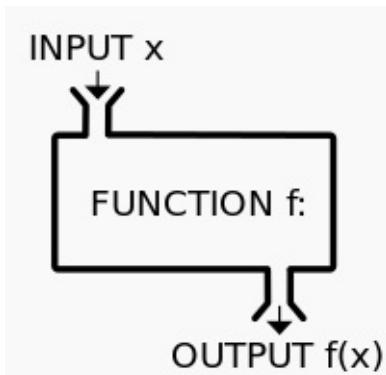
永远强大的函数

函数，对于人类来讲，能够发展到这个数学思维层次，是一个飞跃。可以说，它的提出，直接加快了现代科技和社会的发展，不论是现代的任何科技门类，乃至于经济学、政治学、社会学等，都已经普遍使用函数。

下面一段来自维基百科（在本教程中，大量的定义来自维基百科，因为它真的很百科）：[函数词条](#)

函数这个数学名词是莱布尼兹在1694年开始使用的，以描述曲线的一个相关量，如曲线的斜率或者曲线上的某一点。莱布尼兹所指的函数现在被称作可导函数，数学家之外的普通人一般接触到的函数即属此类。对于可导函数可以讨论它的极限和导数。此两者描述了函数输出值的变化同输入值变化的关系，是微积分学的基础。中文的“函数”一词由清朝数学家李善兰译出。其《代数学》书中解释：“凡此變數中函（包含）彼變數者，則此爲彼之函數”。

函数，从简单到复杂，各式各样。前面提供的维基百科中的函数词条，里面可以做一个概览。但不管什么样子的函数，都可以用下图概括：



有初中数学水平都能理解一个大概了。这里不赘述。

本讲重点说明用python怎么来做一个函数用一用。

深入理解函数

在中学数学中，可以用这样的方式定义函数： $y=4x+3$ ，这就是一个一次函数，当然，也可以写成： $f(x)=4x+3$ 。其中 x 是变量，它可以代表任何数。

当 $x=2$ 时，代入到上面的函数表达式：

$f(2) = 4*2+3 = 11$

所以： $f(2) = 11$

以上对函数的理解，是一般初中生都能打到的。但是，如果看官已经初中毕业了，或者是一个有追求的初中生，还不能局限在上面的理解，还要将函数的理解拓展。

变量不仅仅是数

变量x只能是任意数吗？其实，一个函数，就是一个对应关系。看官尝试着将上面表达式的x理解为馅饼， $4x+3$ ，就是4个馅饼在加上3（单位是什么，就不重要了），这个结果对应着另外一个东西，那个东西比如说是iphone。或者说可以理解为4个馅饼加3就对应一个iphone。这就是所谓映射关系。

所以，x，不仅仅是数，可以是你认为的任何东西。

变量本质——占位符

函数中为什么变量用x？这是一个有趣的问题，自己google一下，看能不能找到答案。

我也不清楚原因。不过，我清楚地知道，变量可以用x，也可以用别的符号，比如y,z,k,i,j...，甚至用alpha,beta,qiwei,qiwsir这样的字母组合也可以。

变量在本质上就是一个占位符。这是一针见血的理解。什么是占位符？就是先把那个位置用变量占上，表示这里有一个东西，至于这个位置放什么东西，以后再说，反正先用一个符号占着这个位置（占位符）。

其实在高级语言编程中，变量比我们在初中数学中学习的要复杂。但是，现在我们先不管那些，复杂东西放在以后再说了。现在，就按照初中数学来研究python中的变量

通常使小写字母来命名python中的变量，也可以在其中加上下划线什么的，表示区别。

比如：alpha,x,j,p_beta，这些都可以做为python的变量。

给变量赋值

打开IDLE，实验操作如下：

```

>>> a = 2    #注1
>>> a        #注2
2
>>> b = 3    #注3
>>> c = 3
>>> b
3
>>> c
3
>>>

```

说明：

- 注1：`a=2`的含义是将一个变量`a`指向了2这个数，就好比叫做`a`是的钓鱼的人，通过鱼线，跟一条叫做2的鱼连接者，`a`通过鱼线就可以导到2
- 注2：相当于要`a`这个钓鱼的人，顺着鱼线导出那条鱼，看看连接的是哪一条，发现是叫做2的那条傻鱼
- 注3：`b=3`，理解同上。那么`c=3`呢？就是这条叫做3的鱼被两个人同时钓到了。

建立简单函数

```

>>> a = 2
>>> y=3*a+2
>>> y
8

```

这种方式建立的函数，跟在初中数学中学习的没有什么区别。当然，这种方式的函数，在编程实践中的用途不大，一般是在学习阶段理解函数来使用的。

别急躁，你在输入`a=3`，然后输入`y`，看看得到什么结果呢？

```

>>> a=2
>>> y=3*a+2
>>> y
8
>>> a=3
>>> y
8

```

是不是很奇怪？为什么后面已经让`a`等于3了，结果`y`还是8。

用前面的钓鱼理论就可以解释了。`a`和2相连，经过计算，`y`和8相连了。后面`a`的连接对象修改了，但是`y`的连接对象还没有变，所以，还是8。再计算一次，`y`的连接对象就变了：

```
>>> a=3
>>> y
8
>>> y=3*a+2
>>> y
11
```

特别注意，如果没有先a=2，就直接下函数表达式了，像这样，就会报错。

```
>>> y=3*a+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    NameError: name 'a' is not defined
```

注意看错误提示，a是一个变量，提示中告诉我们这个变量没有定义。显然，如果函数中要使用某个变量，不得不提前定义出来。定义方法就是给这个变量赋值。

建立实用的函数

上面用命令方式建立函数，还不够“正规化”，那么就来写一个.py文件吧。

在IDLE中，File->New window

然后输入如下代码：

```
#coding:utf-8

def add_function(a,b):
    c = a+b
    print c

if __name__=="__main__":
    add_function(2,3)
```

然后将文件保存，我把它命名为106-1.py，你根据自己的喜好取个名字。

然后我就进入到那个文件夹，运行这个文件，出现下面的结果，如图：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105-1.py 105.py 106-1.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python 106-1.py
5
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ |
```

你运行的结果是什么？如果没有得到上面的结果，你就非常认真地检查代码，是否跟我写的完全一样，注意，包括冒号和空格，都得一样。冒号和空格很重要。

下面开始庖丁解牛：

- `#coding:utf-8` 声明本文件中代码的字符集类型是utf-8格式。初学者如果还不理解，一方面可以去google，另外还可放一放，就先这么抄写下来，以后会讲解。
- `def add_function(a,b):` 这里是函数的开始。在声明要建立一个函数的时候，一定要使用`def`（`def`就是英文`define`的前三个字母），意思就是告知计算机，这里要声明一个函数；`add_function`是这个函数名称，取名字是有讲究的，就好比你的名字一样。在`python`中取名字的讲究就是要有一定意义，能够从名字中看出这个函数是用来干什么的。从`add_function`这个名字中，是不是看出她是用来计算加法的呢？`(a,b)`这个括号里面的是这个函数的参数，也就是函数变量。冒号，这个冒号非常非常重要，如果少了，就报错了。冒号的意思就是下面好开始真正的函数内容了。
- `c=a+b` 特别注意，这一行比上一行要缩进四个空格。这是`python`的规定，要牢记，不可丢掉，丢了就报错。然后这句话就是将两个参数(变量)相加，结果赋值与另外一个变量`c`。
- `print c` 还是提醒看官注意，缩进四个空格。将得到的结果`c`的值打印出来。
- `if name=="main":` 这句话先照抄，不解释。注意就是不缩进了。
- `add_function(2,3)` 这才是真正调用前面建立的函数，并且传入两个参数：`a=2,b=3`。仔细观察传入参数的方法，就是把2放在`a`那个位置，3放在`b`那个位置（所以说，变量就是占位符）。

解牛完毕，做个总结：

声明函数的格式为：

```
def 函数名(参数1, 参数2, ..., 参数n):
    函数体
```

是不是样式很简单呢？

取名字的学问

有的大师，会通过某个人的名字来预测他/她的吉凶祸福等。看来名字这玩意太重要了。取个好名字，就有好兆头呀。所以孔丘先生说“名不正，言不顺”，歪解：名字不正规化，就不顺。这是歪解，希望不要影响看官正确理解。不知道大师们是不是能够通过外国人名字预测外国人的吉凶祸福呢？

不管怎样，某国人是很在意名字的，旁边有个国家似乎就不在乎。

python也很在乎名字问题，其实，所有高级语言对名字都有要求。为什么呢？因为如果命名乱了，计算机就有点不知所措了。看python对命名的一般要求。

- 文件名:全小写,可使用下划线
- 函数名:小写，可以用下划线风格单词以增加可读性。如：`myfunction`，`my_example_function`。注意：混合大小写仅被允许用于这种风格已经占据优势的时候，以便保持向后兼容。
- 函数的参数:如果一个函数的参数名称和保留的关键字(所谓保留关键字，就是python语言已经占用的名称，通常被用来做为已经有的函数等的命名了，你如果还用，就不行了。)冲突，通常使用一个后缀下划线好于使用缩写或奇怪的拼写。
- 变量:变量名全部小写，由下划线连接各个单词。如`color = WHITE`，`this_is_a_variable = 1`。

其实，关于命名的问题，还有不少争论呢？最典型的是所谓匈牙利命名法、驼峰命名等。如果你喜欢，可以google一下。以下内容供参考：

- [匈牙利命名法](#)
- [驼峰式大小写](#)
- [帕斯卡命名法](#)
- [python命名的官方要求](#)，如果看官的英文可以，一定要阅读。如果英文稍逊，可以来阅读[中文](#),不用梯子能行吗？看你命了。

And since they did not see fit to acknowledge God, God gave them up to a debased mind and things that should not be done. They were filled with every kind of wickedness, evil, covetousness, malice. Full of envy, murder, strife, deceit, craftiness, they are gossips, slanderers, God-haters, insolent, haughty, boastful, inventors of evil, rebellious toward parents, foolish, faithless, heartless, ruthless. They know God's decree, that those who practice such things deserve to die--yet they not only do them but even applaud others who practice them. (ROMANS 1:28-32)

玩转字符串(1)

如果对自然语言分类，有很多中分法，比如英语、法语、汉语等，这种分法是最常见的。在语言学里面，也有对语言的分类方法，比如什么什么语系之类的。我这里提出一种分法，这种分法尚未得到广大人民群众和研究者的广泛认同，但是，我相信那句“真理是掌握在少数人的手里”，至少在这里可以用来给自己壮壮胆。

我的分法：一种是语言中的两个元素（比如两个字）和在一起，出来一个新的元素（比如新的字）；另外一种是两个元素和在一起，知识两个元素并列。比如“好”和“人”，两个元素和在一起是“好人”，而3和5和在一起是8，如果你认为是35，那就属于第二类和法了。

把我的这种分法抽象一下：

- 一种是： $\Delta + \square = \circ$
- 另外一种是： $\Delta + \square = \Delta \square$

我们的语言中，离不开以上两类，不是第一类就是第二类。

太天才了。请鼓掌。

字符串

在我洋洋自得的时候，我google了一下，才发现，自己没那么高明，看维基百科的字符串词条是这么说的：

字符串（String），是由零个或多个字符组成的有限串行。一般记为 $s=a[1]a[2]...a[n]$ 。

看到维基百科的伟大了吧，它已经把我所设想的一种情况取了一个形象的名称，叫做字符串

根据这个定义，在前面两次让一个程序员感到伟大的"Hello,World"，就是一个字符串。或者说不管用英文还是中文还是别的某种问，写出来的文字都可以做为字符串对待，当然，里面的特殊符号，也是可以做为字符串的，比如空格等。

操练一下字符串吧。

```
>>> print "good good study, day day up"
good good study, day day up
>>> print "----good---study---day----up"
----good---study---day----up
```

在print后面，打印的都是字符串。注意，是双引号里面的，引号不是字符串的组成部分。它是在告诉计算机，它里面包裹着的是一个字符串。也就是在python中，通常用一对双引号、或者单引号来包裹一个字符串。或者说，要定义一个字符串，就用双引号或者单引号。

爱思考的看官肯定发现上面这句话有问题了。如果我要把下面这句话看做一个字符串，应该怎么做？

```
小明说"我没有烧圆明园"
```

或者这句

```
What's your name?
```

问题非常好，有道理。在python中有一种方法专门解决类似的问题。看下面的例子：

```
>>> print "小明说："\"我没有少圆明园\""
小明说"我没有少圆明园"
```

这个例子中，为了打印出那句含有双引号的字符串，也就是双引号是字符串的一部分了，使用了一个符号：\，在python中，将这个符号叫做转义符。本来双引号表示包括字符串，它不是字符串一部分，但是如果前面有转义符，那么它就失去了原来的含义，转化为字符串的一部分，相当于一个特殊字符了。

下面用转义符在打印第二句话：

```
>>> print 'what\'s your name?'
what's your name?
```

另外，双引号和单引号还可以嵌套，比如下面的句子中，单引号在双引号里面，虽然没有在单引号前面加转义符，但是它被认为是字符串一部分，而不是包裹字符串的符号

```
>>> print "what's your name?"  #双引号包裹单引号，单引号是字符
what's your name?
>>> print 'what "is your" name' #单引号包裹双引号，双引号是字符
what "is your" name
```

变量连接到字符串

前面讲过变量了，并且有一个钓鱼的比喻。如果忘记了，请看前一章内容。

其实，变量不仅可以跟数字连接，还能够跟字符串连接。

```
>>> a=5
>>> a
5
>>> print a
5
>>> b="hello,world"
>>> b
'hello,world'
>>> print b
hello,world
```

还记得我们曾经用过一个`type`命令吗？现在它还有用，就是检验一个变量，到底跟什么类型联系着，是字符串还是数字？

```
>>> type(a)
<type 'int'>
>>> type(b)
<type 'str'>
```

程序员们经常用一种简单的说法，把`a`称之为数字型变量，意思就是它能够或者已经跟数字连着呢；把`b`叫做字符（串）型变量，意思就是它能够或者已经跟字符串连着呢。

对字符串的简单操作

对数字，有一些简单操作，比如四则运算就是，如果`3+5`，就计算出为`8`。那么对字符串都能进行什么样的操作呢？试试吧：

```
>>> "py"+"thon"
'python'
```

跟我那个不为大多数人认可的发现是一样的，你还不认可吗？两个字符串相加，就相当于把两个字符串连接起来。（别的运算就别尝试了，没什么意义，肯定报错，不信就试试）

```
>>> "py"- "thon"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

以上就是对字符串的第一种操作。

连接字符串

在IDLE中按照下面方法操作

```
>>> a = "老齐"
>>> b = "教python"
>>> c = a+b
>>> print c
老齐教python
>>> c
'\xe8\x80\x81\xe9\xbd\x90\xe6\x95\x99python'
```

这是一种最简单连接两个字符串的方法。注意上面例子的最后一行，怎么出现乱码了？那不是乱码，是字符编码的问题。这个你权当没看见好了。不过的确是看见了。请看官[google](#)字符编码就知道了。这里推荐一篇非常好的文章：[字符集和字符编码](#)

提示：看官做为学习者，一定要对所学的对象有一种好奇心，比如上面例子中，如果你满足于print c，发现结果跟自己所预料一样，这还远远不够。如果你向下走了一行，就发现一个怪怪的结果了，这就让你在编程路上又前进一大步。所以，要有对世界好奇的心，不断探索、思考和尝试。反正在计算机上尝试，也没有多大成本。最坏的结果是关掉IDLE罢了。

用 + 号实现连接，的确比较简单，不过，有时候你会遇到这样的问题：

```
>>> a = 1989
>>> b = "free"
>>> print b+a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

抱错了，其错误原因已经打印出来了（一定要注意看打印出来的信息）：cannot concatenate 'str' and 'int' objects。原来 a 对应的对象是一个 int 类型的，不能将它和 str 对象连接起来。怎么办？

可以用下面三种方法中的任何一种：

```

>>> print b + `a`      #注意，` `是反引号，不是单引号，就是键盘中通常在数字1左边的那个，在英文半角状
free1989
>>> print b + str(a)  #str(a)实现将整数对象转换为字符串对象
free1989
>>> print b + repr(a) #repr(a)与上面的类似
free1989

```

可能看官看到这个，就要问它们三者之间的区别了。首先明确，`repr()`和````是一致的，就不用区别了。接下来需要区别的就是`repr()`和`str`，一个最简单的区别，`repr`是函数，`str`是跟`int`一样，一种对象类型。不过这么说是不能完全解惑的。幸亏有那好的google让我辈使用，你会找到不少人对这两者进行区分的内容，我推荐这个：

1. When should i use str() and when should i use repr() ?

Almost always use str when creating output for end users.

`repr` is mainly useful for debugging and exploring. For example, if you suspect a string has non printing characters in it, or a float has a small rounding error, `repr` will show you; `str` may not.

`repr` can also be useful for generating literals to paste into your source code. It can also be used for persistence (with `ast.literal_eval` or `eval`), but this is rarely a good idea—if you want editable persisted values, something like JSON or YAML is much better, and if you don't plan to edit them, use `pickle`.

2. In which cases i can use either of them ?

Well, you can use them almost anywhere. You shouldn't generally use them except as described above.

3. What can str() do which repr() can't ?

Give you output fit for end-user consumption--not always (e.g., `str(['spam', 'eggs'])` isn't likely to be anything you want to put in a GUI), but more often than `repr`.

4. What can repr() do which str() can't

Give you output that's useful for debugging--again, not always (the default for instances of user-created classes is rarely helpful), but whenever possible.

And sometimes give you output that's a valid Python literal or other expression--but you rarely want to rely on that except for interactive exploration.

以上英文内容来源：<http://stackoverflow.com/questions/19331404/str-vs-repr-functions-in-python-2-7-5>

Python转义字符

在字符串中，有时需要输入一些特殊的符号，但是，某些符号不能直接输出，就需要用转义符。所谓转义，就是不采用符号现在之前的含义，而采用另外一含义了。下面表格中列出常用的转义符：

转义字符	描述
\	(在行尾时) 续行符
\	反斜杠符号
'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy代表的字符，例如：\o12代表换行
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

以上所有转义符，都可以通过交互模式下print来测试一下，感受实际上是什么样子的。例如：

```

>>> print "hello.I am qiwsir.\n          #这里换行，下一行接续
... My website is 'http://qiwsir.github.io'."
hello.I am qiwsir.My website is 'http://qiwsir.github.io'.

>>> print "you can connect me by qq\\weibo\\gmail" #\\是为了要后面那个\
you can connect me by qq\weibo\gmail

```

看官自己试试吧。如果有问题，可以联系我解答。

玩转字符串(2)

上一章中已经讲到连接两个字符串的一种方法。复习一下：

```
>>> a= 'py'
>>> b= 'thon'
>>> a+b
'python'
```

既然这是一种方法，言外之意，还有另外一种方法。

连接字符串的方法2

在说方法2之前，先说明一下什么是占位符，此前在讲解变量（参数）的时候，提到了占位符，这里对占位符做一个比较严格的定义：

来自[百度百科](#)的定义：

顾名思义，占位符就是先占住一个固定的位置，等着你再往里面添加内容的符号。

根据这个定义，在[python](#)里面规定了一些占位符，通过这些占位符来说明那个位置应该填写什么类型的东西，这里暂且了解两个占位符：`%d`——表示那个位置是整数，`%s`——表示那个位置应该是字符串。下面看一个具体实例：

```
>>> print "one is %d"%1
one is 1
```

要求打印(`print`)的内容中，有一个`%d`占位符，就是说那个位置应该放一个整数。在第二个`%`后面，跟着的就是那个位置应该放的东西。这里是一个整数1。我们做下面的操作，就可以更清楚了解了：

```
>>> a=1
>>> type(a)
<type 'int'>    #a是整数
>>> b="1"
>>> type(b)
<type 'str'>    #b是字符串
>>> print "one is %d"%a
one is 1
>>> print "one is %d"%b      #报错了，这个占位符的位置应该放整数，不应该放字符串。
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: %d format: a number is required, not str
```

同样道理，%**s**对应的位置应该放字符串，但是，如果放了整数，也可以。只不过是已经转为字符串对待了。但是不赞成这么做。在将来，如果使用mysql（一种数据库）的时候，会要求都用%**s**做为占位符，这是后话，听听有这么回事即可。

```
>>> print "one is %s"%b
one is 1
>>> print "one is %s"%a      #字符串是包容的
one is 1
```

好了。啰嗦半天，占位符是不是理解了呢？下面我们就用占位符来连接字符串。是不是很有意思？

```
>>> a = "py"
>>> b = "thon"
>>> print "%s%s"%(a, b)  #注
python
```

注：仔细观察，如果两个占位符，要向这两个位置放东西，代表的东西要写在一个圆括号内，并且中间用逗号（半角）隔开。

字符串复制

有一个变量，连接某个字符串，也想让另外一个变量，也连接这个字符串。一种方法是把字符串再写一遍，这种方法有点笨拙，对于短的到无所谓了。但是长的就麻烦了。这里有一种字符串复制的方法：

```
>>> a = "My name is LaoQi. I like python and can teach you to learn it."
>>> print a
My name is LaoQi. I like python and can teach you to learn it.
>>> b = a
>>> print b
My name is LaoQi. I like python and can teach you to learn it.
>>> print a
My name is LaoQi. I like python and can teach you to learn it.
```

复制非常简单，类似与赋值一样。可以理解为那个字符串本来跟a连接着，通过b=a，a从自己手里分处一股绳子给了b，这样两者都可以指向那个字符串了。

字符串长度

要向知道一个字符串有多少个字符，一种方法是从头开始，盯着屏幕数一数。哦，这不是计算机在干活，是键客在干活。键客，不是剑客。剑客是以剑为武器的侠客；而键客是以键盘为武器的侠客。当然，还有贱客，那是贱人的最高境界，贱到大侠的程度，比如岳不群之流。

键客这样来数字符串长度：

```
>>> a="hello"
>>> len(a)
5
```

使用的是一个函数len(object)。得到的结果就是该字符串长度。

```
>>> m = len(a) #把结果返回后赋值给一个变量
>>> m
5
>>> type(m)      #这个返回值（变量）是一个整数型
<type 'int'>
```

字符大小写的转换

对于英文，有时候要用到大小写转换。最有名驼峰命名，里面就有一些大写和小写的参合。如果有兴趣，可以来这里看[自动将字符串转化为驼峰命名形式的方法](#)。

在python中有下面一堆内建函数，用来实现各种类型的大小写转化

- S.upper() #S中的字母大写
- S.lower() #S中的字母小写
- S.capitalize() #首字母大写
- S.istitle() #单词首字母是否大写的，且其它为小写，注网友白羽毛指出，这里表述不准确。非常感谢他。为了让看官对这些大小写问题有更深刻理解，我重新写下面的例子，请看官审查。再次感谢白羽毛。
- S.isupper() #S中的字母是否全是大写
- S.islower() #S中的字母是否全是小写

看例子：

```
>>> a = "qiwsir,python"
>>> a.upper()          #将小写字母完全变成大写字母
'QIWSIR,PYTHON'
>>> a                  #原数据对象并没有改变
'qiwsir,python'
>>> b = a.upper()
>>> b
'QIWSIR,PYTHON'
>>> c = b.lower()      #将所有的小写字母变成大写字母
>>> c
'qiwsir,python'

>>> a
'qiwsir,python'
>>> a.capitalize()    #把字符串的第一个字母变成大写
'Qiwsir,python'
>>> a                  #原数据对象没有改变
'qiwsir,python'
>>> b = a.capitalize() #新建立了一个
>>> b
'Qiwsir,python'

>>> a = "qiwsir,github"    #这里的问题就是网友白羽毛指出的，非常感谢他。
>>> a.istitle()
False
>>> a = "QIWSIR"          #当全是大写的时候，返回False
>>> a.istitle()
False
>>> a = "qIWSIR"
>>> a.istitle()
False
>>> a = "Qiwsir,github"  #如果这样，也返回False
>>> a.istitle()
False
>>> a = "Qiwsir"          #这样是True
>>> a.istitle()
True
>>> a = 'Qiwsir,Github' #这样也是True
>>> a.istitle()
True

>>> a = "Qiwsir"
>>> a.isupper()
False
>>> a.upper().isupper()
True
>>> a.islower()
False
>>> a.lower().islower()
True
```

顺着白羽毛网友指出的，再探究一下，可以这么做：

```
>>> a = "This is a Book"
>>> a.istitle()
False
>>> b = a.title()      #这样就把所有单词的第一个字母转化为大写
>>> b
'This Is A Book'
>>> a.istitle()        #判断每个单词的第一个字母是否为大写
False
```

字符串问题，看来本讲还不能结束。下一讲继续。有看官可能要问了，上面这些在实战中怎么用？我正想为你的，请键客设计一种实战情景，能不能用上所学。

玩转字符串(3)

字符串是一个很长的话题，纵然现在开始第三部分，但是也不能完全说尽。因为字符串是自然语言中最复杂的东西，也是承载功能最多的，计算机高级语言编程，要解决自然语言中的问题，让自然语言中完成的事情在计算机上完成，所以，也不得不有更多的话题。

字符串就是一个话题中心。

给字符串编号

在很多很多情况下，我们都要对字符串中的每个字符进行操作（具体看后面的内容），要准确进行操作，必须做的一个工作就是把字符进行编号。比如一个班里面有50名学生，如果这些学生都有学号，老师操作他们将简化很多。比如不用专门找每个人名字，直接通过学号知道谁有没有交作业。

在python中按照这样的顺序对字符串进行编号：从左边第一个开始是0号，向下依次按照整数增加，为1、2...，直到最后一个，在这个过程中，所有字符，包括空格，都进行变好。例如：

Hello,world

对于这个字符串，从左向右的变好依次是：

|0|1|2|3|4|5|6|7|8|9|10|11|

|H|e|l|l|o|,|w|o|r| || |d|

在班级了，老师只要喊出学生的学号，自动有对应的学生站起来。在python里面如何把某个编号所对应的字符调出来呢？看代码：

```

>>> a = "Hello,world"
>>> len(a)      #字符串的长度是12，说明公有12个字符，最后一个字符编号是11
12
>>> a[0]
'H'
>>> a[3]
'l'
>>> a[9]
'o'
>>> a[11]
'd'
>>> a[5]
'w'

```

特别说明，编号是从左边开始，第一个是0。

能不能从右边开始编号呢？可以。这么人见人爱的python难道这点小要求都不满足吗？

```
>>> a[-1]
'd'
>>> a[11]
'd'
>>> a[-12]
'H'
>>> a[-3]
'
```

看到了吗？如果从右边开始，第一个编号是-1,这样就跟从左边区分开了。也就是a[-1]和a[11]是指向同一个字符。

不管从左边开始还是从右边开始，都能准确找到某个字符。看官喜欢从哪边开始就从哪边开始，或者根据实际使用情况，需要从哪边开始就从哪边开始。

字符串截取

有了编号，不仅仅能够找出某个字符，还能在字符串中取出一部分来。比如，从“hello,world”里面取出“llo”。可以这样操作

```
>>> a[2:5]
'llo'
```

这就是截取字符串的一部分，注意：所截取部分的第一个字符 (l) 对应的编号是(2)，从这里开始；结束的字符是(o)，对应编号是 (4) ，但是结束的编号要增加1,不能是4,而是5.这样截取到的就是上面所要求的了。

试一试，怎么截取到",wor"

也就是说，截取a[n,m]，其中n<m，得到的字符是从a[n]开始到a[m-1]

有几个比较特殊的

```
>>> a[:]    #表示截取全部
'Hello,world'
>>> a[3:]  #表示从a[3]开始，一直到字符串的最后
'lo,world'
>>> a[:4]  #表示从字符串开头一直到a[4]前结束
'Hello'
```

去掉字符串两头的空格

这个功能，在让用户输入一些信息的时候非常有用。有的朋友喜欢输入结束的时候敲击空格，比如让他输入自己的名字，输完了，他来个空格。有的则喜欢先加一个空格，总做的输入的第一个字前面应该空两个格。

好吧，这些空格是没用的。python考虑到有不少人可能有这个习惯，因此就帮助程序员把这些空格去掉。

方法是：

- `S.strip()` 去掉字符串的左右空格
- `S.lstrip()` 去掉字符串的左边空格
- `S.rstrip()` 去掉字符串的右边空格

看官在看下面示例之前，请先自己用上面的内置函数，是否可以？

```
>>> b=" hello "
>>> b
' hello '
>>> b.strip()
'hello'
>>> b
' hello '
>>> b.lstrip()
'hello '
>>> b.rstrip()
' hello'
```

练习

学编程，必须做练习，通过练习熟悉各种情况下的使用。

下面共同做一个练习：输入用户名，计算机自动向这个用户打招呼。代码如下：

```
#coding:utf-8

print "please write your name:"
name=raw_input()
print "Hello,%s"%name
```

这段代码中的`raw_input()`的含义，就是要用户输入内容，所输入的内容是一个字符串。

其实，上面这段代码存在这改进的地方，比如，如果用户输入的是小写，是不是要将名字的首字母变成大写呢？如果有空格，是不是要去掉呢？等等。或许还有别的，看看能不能在这这个练习中，将以前学习过的东西综合应用一下？

眼花缭乱的运算符

在计算机高级中语言，运算符是比较多样化的。其实，也都源于我们日常的需要。

算术运算符

前面已经讲过了四则运算，其中涉及到一些运算符：加减乘除，对应的符号分别是：`+` `-` `*` `/`，此外，还有求余数的：`%`。这些都是算术运算符。其实，算术运算符不止这些。根据中学数学的知识，看官也应该想到，还应该有乘方、开方之类的。

下面列出一个表格，将所有的运算符表现出来。不用记，但是要认真地看一看，知道有那些，如果以后用到，但是不自信能够记住，可以来查。

运算符	描述	实例
<code>+</code>	加 - 两个对象相加	<code>10+20</code> 输出结果 30
<code>-</code>	减 - 得到负数或是一个数减去另一个数	<code>10-20</code> 输出结果 -10
<code>*</code>	乘 - 两个数相乘或是返回一个被重复若干次的字符串	<code>10 * 20</code> 输出结果 200
<code>/</code>	除 - x 除以 y	<code>20/10</code> 输出结果 2
<code>%</code>	取余 - 返回除法的余数	<code>20%10</code> 输出结果 0
<code>**</code>	幂 - 返回 x 的 y 次幂	<code>10**2</code> 输出结果 100
<code>//</code>	取整除 - 返回商的整数部分	<code>9//2</code> 输出结果 4, <code>9.0//2.0</code> 输出结果 4.0

是不是看着并不陌生呀。这里有一个建议给看官，请打开你的IDLE，依次将上面的运算符实验一下。

列为看官可以根据中学数学的知识，想想上面的运算符在混合运算中，应该按照什么顺序计算。并且亲自试试，是否与中学数学中的规律一致。（应该是一致的，计算机科学家不会另外搞一套让我们和他们一块受罪。）

比较运算符

所谓比较，就是比一比两个东西。这在某国是最常见的了，做家长的经常把自己的孩子跟别人的孩子比较，唯恐自己孩子在某方面差了；官员经常把自己的工资和银行比较，总觉得少了。

在计算机高级语言编程中，任何两个同一类型的量都可以比较，比如两个数字可以比较，两个字符串可以比较。注意，是两个同一类型的。不同类型的量可以比较吗？首先这种比较没有意义。就好比二两肉和三尺布进行比较，它们谁大呢？这种比较无意义。所以，在真正的编程中，我们要谨慎对待这种不同类型量的比较。

但是，在某些语言中，允许这种无意思的比较。因为它在比较的时候，都是将非数值的转化为了数值类型比较。这个后面我们会做个实验。

对于比较运算符，在小学数学中就学习了一些：大于、小于、等于、不等于。没有陌生的东西，python里面也是如此。且看下表：

以下假设变量a为10，变量b为20：

运算符	描述	实例
<code>==</code>	等于 - 比较对象是否相等	<code>(a == b)</code> 返回 <code>False</code> 。
<code>!=</code>	不等于 - 比较两个对象是否不相等	<code>(a != b)</code> 返回 <code>true</code> 。
<code>></code>	大于 - 返回x是否大于y	<code>(a > b)</code> 返回 <code>False</code> 。
<code><</code>	小于 - 返回x是否小于y	<code>(a < b)</code> 返回 <code>true</code> 。
<code>>=</code>	大于等于 - 返回x是否大于等于y。	<code>(a >= b)</code> 返回 <code>False</code> 。
<code><=</code>	小于等于 - 返回x是否小于等于y。	<code>(a <= b)</code> 返回 <code>true</code> 。

上面的表格实例中，显示比较的结果就是返回一个`true`或者`false`，这是什么意思呢。就是在告诉你，这个比较如果成立，就是为真，返回`True`，否则返回`False`，说明比较不成立。

请按照下面方式进行比较操作，然后再根据自己的想象，把比较操作熟练熟练。

```

>>> a=10
>>> b=20
>>> a>b
False
>>> a<b
True
>>> a==b
False
>>> a!=b
True
>>> a>=b
False
>>> a<=b
True
>>> c="5"  #a、c是两个不同类型的量，能比较，但是不提倡这么做。
>>> a>c
False
>>> a<c
True

```

逻辑运算符

首先谈谈什么是逻辑，韩寒先生对逻辑有一个分类：

逻辑分两种，一种是逻辑，另一种是中国人的逻辑。——韩寒

这种分类的确非常精准。在很多情况下，中国人是很有奇葩的逻辑的。但是，在python中，讲的是逻辑，不是中国人的逻辑。

逻辑（logic），又称理则、论理、推理、推论，是有效推论的哲学研究。逻辑被使用在大部份的智能活动中，但主要在哲学、数学、语义学和计算机科学等领域内被视为一门学科。在数学里，逻辑是指研究某个形式语言的有效推论。

关于逻辑问题，看官如有兴趣，可以听一听 [《国立台湾大学公开课：逻辑》](#)

下面简单理解一下逻辑问题。

布尔类型的变量

在所有的高级语言中，都有这么一类变量，被称之为布尔型。从这个名称，看官就知道了，这是用一个人的名字来命名的。

乔治·布尔（George Boole，1815年11月—1864年），英格兰数学家、哲学家。

乔治·布尔是一个皮匠的儿子，生于英格兰的林肯。由于家境贫寒，布尔不得不在协助养家的同时为自己能受教育而奋斗，不管怎么说，他成了19世纪最重要的数学家之一。尽管他考虑过以牧师为业，但最终还是决定从教，而且不久就开办了自己的学校。

在备课的时候，布尔不满意当时的数学课本，便决定阅读伟大数学家的论文。在阅读伟大的法国数学家拉格朗日的论文时，布尔有了变分法方面的新发现。变分法是数学分析的分支，它处理的是寻求优化某些参数的曲线和曲面。

1848年，布尔出版了《The Mathematical Analysis of Logic》，这是他对符号逻辑诸多贡献中的第一次。

1849年，他被任命位于爱尔兰科克的皇后学院（今科克大学或UCC）的数学教授。1854年，他出版了《The Laws of Thought》，这是他最著名的著作。在这本书中布尔介绍了现在以他的名字命名的布尔代数。布尔撰写了微分方程和差分方程的课本，这些课本在英国一直使用到19世纪末。由于其在符号逻辑运算中的特殊贡献，很多计算机语言中将逻辑运算称为布尔运算，将其结果称为布尔值。

请看官认真阅读布尔的生平，立志呀。

布尔所创立的这套逻辑被称之为“布尔代数”。其中规定只有两种值，True和False，正好对应这计算机上二进制数的1和0。所以，布尔代数和计算机是天然吻合的。

所谓布尔类型，就是返回结果为1(True)、0(False)的数据变量。

在python中（其它高级语言也类似，其实就是布尔代数的运算法则），有三种运算符，可以实现布尔类型的变量间的运算。

布尔运算

看下面的表格，对这种逻辑运算符比较容易理解：

（假设变量a为10，变量b为20）

运算符	描述	实例
and	布尔"与"- 如果x为False，x and y返回False，否则它返回y的计算值。	(a and b) 返回 true。
or	布尔"或"- 如果x是True，它返回True，否则它返回y的计算值。	(a or b) 返回 true。
not	布尔"非"- 如果x为True，返回False。如果x为False，它返回True。	not(a and b) 返回 false。

- and

and，翻译为“与”运算，但事实上，这种翻译容易引起望文生义的理解。先说一下正确的理解。A and B，含义是：首先运算A，如果A的值是true，就计算B，并将B的结果返回做为最终结果，如果B是False，那么A and B的最终结果就是False,如果B的结果是True，那么A and B的结果就是True；如果A的值是False,就不计算B了，直接返回A and B的结果为False.

比如：

`4>3 and 4<9`，首先看 `4>3` 的值，这个值是 `True`，再看 `4<9` 的值，是 `True`，那么最终这个表达式的结果为 `True`。

```
>>> 4>3 and 4<9
True
```

`4>3 and 4<2`，先看 `4>3`，返回 `True`，再看 `4<2`，返回的是 `False`，那么最终结果是 `False`。

```
>>> 4>3 and 4<2
False
```

`4<3 and 4<9`，先看 `4<3`，返回为 `False`，就不看后面的了，直接返回这个结果做为最终结果。

```
>>> 4<3 and 4<2
False
```

前面说容易引起望文生义的理解，就是有相当少的人认为无论什么时候都看`and`两边的值，都是`true`返回`true`，有一个是`false`就返回`false`。根据这种理解得到的结果，与前述理解得到的结果一样，但是，运算量不一样哦。

- `or`

`or`，翻译为“或”运算。在`A and B`中，它是这么运算的：

```
if A==True:
    return True
else:
    if B==True:
        return True
    else if B==False:
        return False
```

上面这段算是伪代码啦。所谓伪代码，就是不是真正的代码，无法运行。但是，伪代码也有用途，就是能够以类似代码的方式表达一种计算过程。

看官是不是能够看懂上面的伪代码呢？下面再增加上每行的注释。这个伪代码跟自然的英语差不多呀。

```
if A==True:          #如果A的值是True
    return True      #返回True, 表达式最终结果是True
else:                #否则，也就是A的值不是True
    if B==True:      #看B的值，然后就返回B的值做为最终结果。
        return True
    else if B==False:
        return False
```

举例，根据上面的运算过程，分析一下下面的例子，是不是与运算结果一致？

```
>>> 4<3 or 4<9
True
>>> 4<3 or 4>9
False
>>> 4>3 or 4>9
True
```

- `not`

`not`，翻译成“非”，窃以为非常好，不论面对什么，就是要否定它。

```
>>> not(4>3)
False
>>> not(4<3)
True
```

关于运算符问题，其实不止上面这些，还有呢，比如成员运算符`in`，在后面的学习中会逐渐遇到。

从if开始语句的征程

一般编程的教材，都是要把所有的变量类型讲完，然后才讲语句。这种讲法，其实不符合学习的特点。学习，就是要循序渐进的。在这点上，我可以很吹一通了，因为我做过教师，研究教育教学，算是有一点心得的。所以，我在这里就开始讲授语句。

什么是语句

在前面，我们已经写了一些.py的文件，这些文件可以用python来运行。那些文件，就是由语句组成的程序。

为了能够严谨地阐述这个概念，我还是要抄一段[维基百科中的词条：命令式编程](#)

命令式编程（英语：Imperative programming），是一种描述电脑所需作出的行为的编程范型。几乎所有电脑的硬件工作都是指令式的；几乎所有电脑的硬件都是设计来运行机器码，使用指令式的风格来写的。较高级的指令式编程语言使用变量和更复杂的语句，但仍依从相同的范型。

运算语句一般来说都表现了在存储器内的数据进行运算的行为，然后将结果存入存储器中以便日后使用。高级命令式编程语言更能处理复杂的表达式，可能会产生四则运算和函数计算的结合。

一般所有高级语言，都包含如下语句，Python也不例外：

- **循环语句**:容许一些语句反复运行数次。循环可依据一个默认的数目来决定运行这些语句的次数；或反复运行它们，直至某些条件改变。
- **条件语句**:容许仅当某些条件成立时才运行某个区块。否则，这个区块中的语句会略去，然后按区块后的语句继续运行。
- **无条件分支语句**容许运行顺序转移到程序的其他部分之中。包括跳跃（在很多语言中称为Goto）、副程序和Procedure等。

循环、条件分支和无条件分支都是控制流程。

if语句

谈到语句，不要被吓住。看下面的例子先：

```

if a==4:
    print "it is four"
else:
    print "it is no four"

```

逐句解释一番，注意看注释。在这里给列为看官提醒，在写程序的是由，一定要写必要的注释，同时在阅读程序的时候，也要注意看注释。

```

if a==4:          #如果变量a==4是真的，a==4为True，就
    print "it is four" #打印"it is four"。
else:            #否则，即a==4是假的，a==4为False，就
    print "it is not four" #打印"it is not four"。

```

以上几句话，就完成了一个条件判断，在不同条件下做不同的事情。因此，`if`语句，常被翻译成“条件语句”。

条件语句的基本样式结构：

```

if 条件1:
    执行的内容1
elif 条件2:
    执行的内容2
elif 条件3:
    执行的内容3
else:
    执行的内容4

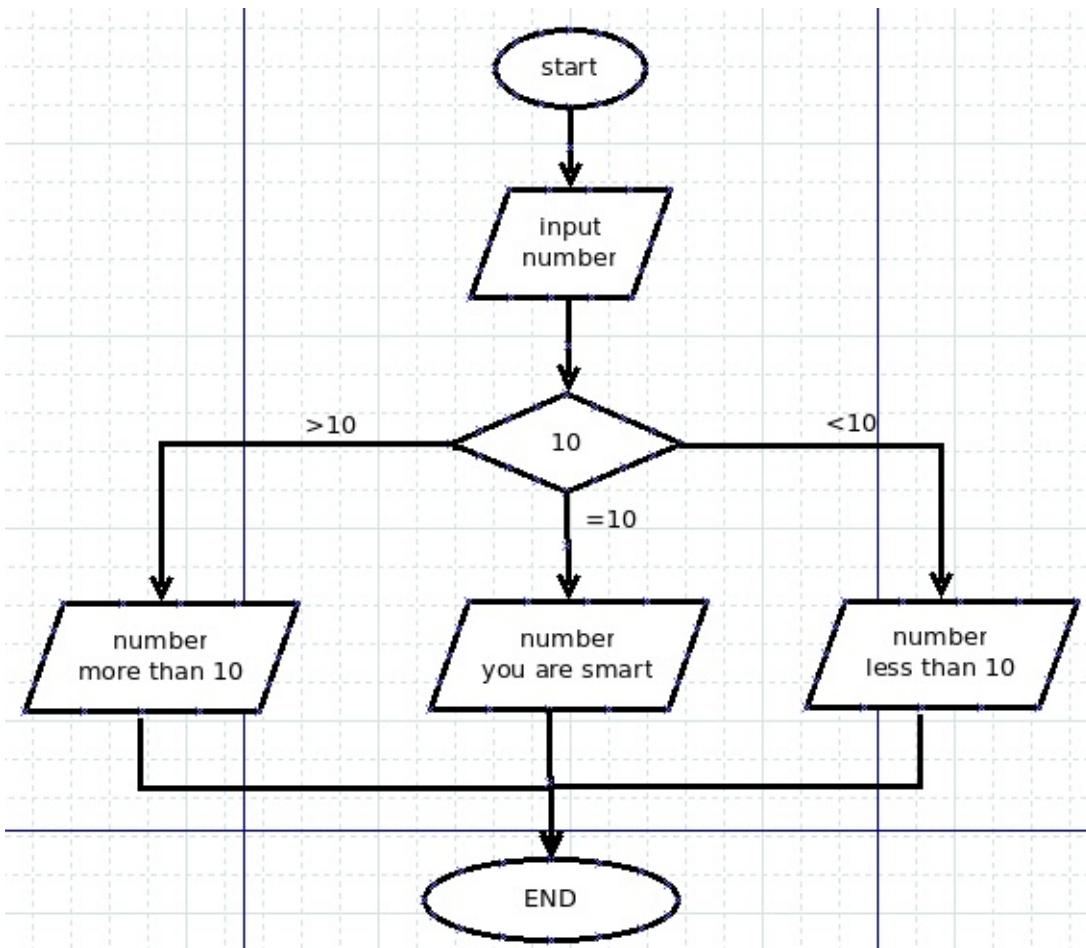
```

执行的内容1、内容2，等，称之为语句块。`elif`用于多个条件时使用，可以没有。另外，也可以只有`if`，而没有`else`。

提醒：每个执行的内容，均以缩进四个空格方式。

例1：输入一个数字，并输出输入的结果，如果这个数字大于10，那么同时输出大于10，如果小于10，同时输出提示小于10，如果等于10，就输出表扬的一句话。

从这里开始，我们的代码就要越来越接近于一个复杂的判断过程了。为了让我们的思维能够更明确上述问题的解决流程，在程序开发过程中，常常要画流程图。什么是流程图，我从另外一个角度讲，就是要让思维过程可视化，简称“思维可视化”。顺便自吹自擂一下，我从2004年就开始在我朝推广思维导图，这就是一种思维可视化工具。自吹到此结束。看这个问题的流程图：



理解了流程图中的含义，就开始写代码，代码实例如下：

```

#!/usr/bin/env python
#coding:utf-8

print "请输入任意一个整数数字 :"

number = int(raw_input())  #通过raw_input()输入的数字是字符串
                          #用int()将该字符串转化为整数

if number == 10:
    print "您输入的数字是 :%d"%number
    print "You are SMART."
elif number > 10:
    print "您输入的数字是 :%d"%number
    print "This number is more than 10."
elif number < 10:
    print "您输入的数字是 :%d"%number
    print "This number is less than 10."
else:
    print "Are you a human?"
  
```

特别提醒看官注意，前面我们已经用过`raw_input()`函数了，这个是获得用户在界面上输入的信息，而通过它得到的是字符串类型的数据。可以在IDLE中这样检验一下：

```

>>> a=raw_input()
10
>>> a
'10'
>>> type(a)
<type 'str'>
>>> a=int(a)
>>> a
10
>>> type(a)
<type 'int'>

```

刚刚得到的那个a就是str类型，如果用int()转换一下，就变成int类型了。

看来int()可以将字str类型的数字转换为int类型，类似，是不是有这样的结论呢：str()可以将int类型的数字转化为str类型.建议看官实验一下。

上述程序的后面，就是依据条件进行判断，不同条件下做不同的事情了。需要提醒的是在条件中：number == 10，为了阅读方便，在number和==之间有一个空格最好了，同理，后面也有一个。这里的10,是int类型，number也是int类型.

上面的程序不知道是不是搞懂了？如果没有，可以通过QQ跟我联系，我的QQ公布一下：26066913，或者登录我的微博，通过微博跟我联系，当然还可以发邮件啦。我看到您的问题，会答复的。在github上跟我互动，是我最欢迎的。

最后，给看官留一个练习题目：

课后练习：开发一个猜数字游戏的程序。即程序在某个范围内指定一个数字，比如在0到9范围内指定一个数字，用户猜测程序所指定的数字大小。

请看官自己编写。我们会在后面讨论这个问题。

小知识

不知道各位是否注意到，上面的那段代码，开始有一行：

```

#!/usr/bin/env python

```

这是什么意思呢？

这句话以#开头，表示本来不在程序中运行。这句话的用途是告诉机器寻找到该设备上的python解释器，操作系统使用它找到的解释器来运行文件中的程序代码。有的程序里写的是/usr/bin python，表示python解释器在/usr/bin里面。但是，如果写成/usr/bin/env，则表示

要通过系统搜索路径寻找python解释器。不同系统，可能解释器的位置不同，所以这种方式能够让代码更将拥有可移植性。对了，以上是对Unix系列操作系统而言。对与windows系统，这句话就当不存在。

一个免费的实验室

在学生时代，就羡慕实验室，老师在里面可以鼓捣各种有意思的东西。上大学的时候，终于有机会在实验室做大量实验了，因为我是物理系，并且，遇到了一位非常令我尊敬的老师——高老师，让他在他的实验室里面，把所有已经破旧损坏的实验仪器修理装配好，并且按照要求做好实验样例。经过一番折腾，才明白，要做好实验，不仅仅花费精力，还有不菲的设备成本呢。后来工作的时候，更感觉到实验设备费用之高昂，因此做实验的时候总要小心翼翼。

再后来，终于发现原来计算机是一个最好的实验室。在这里做实验成本真的很低呀。

扯的远了吧。不远，现在就扯回来。学习Python，也要做实验，也就是尝试性地看看某个命令到底什么含义。通过实验，研究清楚了，才能在编程实践中使用。

怎么做Python实验呢？

走进Python实验室

在《集成开发环境(IDE)》一章中，我们介绍了Python的IDE时，给大家推荐了IDLE，进入到IDLE中，看到>>>符号，可以在后面输入一行指令。其实，这就是一个非常好的实验室。

另外一个实验室就是UNIX操作系统（包含各种Linux和Mac OSx）的shell，在打开shell之后，输入python，出现如下图所示：

```
qw@qw-Latitude-E4300:~$ python
Python 2.7.6 (default, Nov 13 2013, 19:24:16)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

如果看官是用windows的，也能够通过cmd来获得上图类似的界面，依然是输入python，之后得到界面。

在上述任何一个环境中，都可以输入指令，敲回车键运行并输出结果。

在这里你可以随心所欲实验。

交互模式下进行实验

前面的各讲中，其实都使用了交互模式。本着循序渐进、循环上升的原则，本讲应该对交互模式进行一番深入和系统化了。

通过变量直接显示其内容

从例子开始：

```
>>> a="http://qiwsir.github.io"
>>> a
'http://qiwsir.github.io'
>>> print a
http://qiwsir.github.io
```

当给一个变量a赋值于一个字符串之后，输入变量名称，就能够打印出字符串，和print a具有同样的效果。这是交互模式下的一个特点，如果在文件模式中，则不能，只有通过print才能打印变量内容。

缩进

```
>>> if bool(a):
...     print "I like python"
...
I like python
```

对于if语句，在上一讲《从if开始语句的征程》中，已经注意到，if下面的执行语句要缩进四个空格。在有的python教材中，说在交互模式下不需要缩进，可能是针对python3或者其它版本，我使用的是python2.7，的确需要缩进。上面的例子就看出来了。

看官在自己的机器上测试一下，是不是需要缩进？

报错

在一个广告中看到过这样一句话：程序员的格言，“不求最好，只求报错”。报错，对编程不是坏事。如何对待报错呢？

一定要认真阅读所提示的错误信息。

还是上面那个例子，我如果这样写：

```
>>> if bool(a):
... print "I like python"
  File "<stdin>", line 2
    print "I like python"
      ^
IndentationError: expected an indented block
```

从错误信息中，我们可以知道，第二行错了。错在什么地方呢？python非常人性化就在这里，告诉你错误在什么地方：

IndentationError: expected an indented block

意思就是说需要一个缩进块。也就是我没有对第二行进行缩进，需要缩进。

另外，顺便还要提醒，>>>表示后面可以输入指令，...表示当前指令没有结束。要结束并执行，需要敲击两次回车键。

探索

如果看官对某个指令不了解，或者想试试某种操作是否可行，可以在交互模式下进行探索，这种探索的损失成本非常小，充其量就是报错。而且从报错信息中，我们还能得到更多有价值的内容。

例如，在《眼花缭乱的运算符》中，提到了布尔运算，其实，在变量的类型中，除了前面提到的整数型、字符串型，布尔型也是一种，那么布尔型的变量有什么特点呢？下面就探索一下：

```
>>> a
'http://qiwsir.github.io'
>>> bool(a)      #布尔型，用bool()表示，就类似int(),str()，是一个内置函数
True
>>> b=""
>>> bool(b)
False
>>> bool(4>3)
True
>>> bool(4<3)
False
>>> m=bool(b)
>>> m
False
>>> type(m)
<type 'bool'>
>>>
```

从上面的实验可以看出，如果对象是空，返回False，如果不是，则返回True；如果对象是False，返回False。上面探索，还可以扩展到其它情况。看官能不能通过探索，总结出bool()的特点呢？

有容乃大的list(1)

前面的学习中，我们已经知道了两种python的数据类型：`int`和`str`。再强调一下对数据类型的理解，这个世界是由数据组成的，数据可能是数字（注意，别搞混了，数字和数据是有区别的），也可能是文字、或者是声音、视频等。在python中（其它高级语言也类似）把状如`2,3`这样的数字划分为一个类型，把状如“你好”这样的文字划分一个类型，前者是`int`类型，后者是`str`类型（这里就不说翻译的名字了，请看官熟悉用英文的名称，对日后的编程大有好处，什么好处呢？谁用谁知道！）。

前面还学习了变量，如果某个变量跟一个`int`类型的数据用线连着（行话是：赋值），那么这个变量我们就把它叫做`int`类型的变量；有时候还没赋值呢，是准备让这个变量接收`int`类型的数据，我们也要将它声明为`int`类型的变量。不过，在python里面有一样好处，变量不用提前声明，随用随命名。

这一讲中的`list`类型，也是python的一种数据类型。翻译为：列表。下面的黑字，请看官注意了：

LIST在**python**中具有非常强大的功能。

定义

在python中，用方括号表示一个`list`，`[]`

在方括号里面，可以是`int`，也可以是`str`类型的数据，甚至也能够是`True/False`这种布尔值。看下面的例子，特别注意阅读注释。

```
>>> a=[]      #定义了一个变量a，它是list类型，并且是空的。
>>> type(a)
<type 'list'> #用内置函数type()查看变量a的类型，为list
>>> bool(a)    #用内置函数bool()看看list类型的变量a的布尔值，因为是空的，所以为False
False
>>> print a    #打印list类型的变量a
[]
```

不能总玩空的，来点实的吧。

```
>>> a=['2',3,'qiwsir.github.io']
>>> a
['2', 3, 'qiwsir.github.io']
>>> type(a)
<type 'list'>
>>> bool(a)
True
>>> print a
['2', 3, 'qiwsir.github.io']
```

用上述方法，定义一个list类型的变量和数据。

本讲的标题是“有容乃大的list”，就指明了list的一大特点：可以无限大，就是说list里面所能容纳的元素数量无限，当然这是在硬件设备理想的情况下。

list索引

尚记得在《玩转字符串(3)》中，曾经给字符串进行编号，然后根据编号来获取某个或者某部分字符，这样的过程，就是“索引”(index)。

```
>>> url = "qiwsir.github.io"
>>> url[2]
'w'
>>> url[:4]
'qiws'
>>> url[3:9]
'sir.gi'
```

在list中，也有类似的操作。只不过是以元素为单位，不是以字符为单位进行索引了。看例子就明白了。

```

>>> a
['2', 3, 'qiwsir.github.io']
>>> a[0]      #索引序号也是从0开始
'2'
>>> a[1]
3
>>> [2]
[2]
>>> a[:2]    #跟str中的类似，切片的范围是：包含开始位置，到结束位置之前
['2', 3]    #不包含结束位置
>>> a[1:]
[3, 'qiwsir.github.io']
>>> a[-1]    #负数编号从右边开始
'qiwsir.github.io'
>>> a[-2]
3
>>> a[:]
['2', 3, 'qiwsir.github.io']

```

对list的操作

任何一个行业都有自己的行话，如同古代的强盗，把撤退称之为“扯乎”一样，纵然是一个含义，但是强盗们愿意用他们自己的行业用语，俗称“黑话”。各行各业都如此。这样做的目的我理解有两个，一个是某种保密；另外一个是行外人士显示本行业的门槛，让别人感觉这个行业很高深，从业者有一定水平。

不管怎么，在python和很多高级语言中，都给本来数学角度就是函数的东西，又在不同情况下有不同的称呼，如方法、类等。当然，这种称呼，其实也是为了区分函数的不同功能。

前面在对str进行操作的时候，有一些内置函数，比如s.strip()，这是去掉左右空格的内置函数，也是str的方法。按照一贯制的对称法则，对list也会有一些操作方法。

追加元素

```

>>> a = ["good", "python", "I"]
>>> a
['good', 'python', 'I']
>>> a.append("like")          #向list中添加str类型"like"
>>> a
['good', 'python', 'I', 'like']
>>> a.append(100)            #向list中添加int类型100
>>> a
['good', 'python', 'I', 'like', 100]

```

[官方文档](#)这样描述list.append()方法

```
list.append(x)
```

Add an item to the end of the list; equivalent to `a[len(a):] = [x]`.

从以上描述中，以及本部分的标题“追加元素”，是不是能够理解`list.append(x)`的含义呢？即将新的元素`x`追加到`list`的尾部。

列位看官，如果您注意看上面官方文档中的那句话，应该注意到，还有后面半句： equivalent to `a[len(a):] = [x]`，意思是说`list.append(x)`等效于：`a[len(a):]=[x]`。这也相当于告诉我们了另外一种追加元素的方法，并且两种方法等效。

```
>>> a
['good', 'python', 'I', 'like', 100]
>>> a[len(a):]=[3]      #len(a), 即得到list的长度，这个长度是指list中的元素个数。
>>> a
['good', 'python', 'I', 'like', 100, 3]
>>> len(a)
6
>>> a[6:]=['xxoo']
>>> a
['good', 'python', 'I', 'like', 100, 3, 'xxoo']
```

顺便说一下`len()`，这个是用来获取`list, str`等类型的数据长度的。在字符串讲解的时候也提到了。

```
>>> name = 'yeashape'
>>> len(name)      #str的长度，是字符的个数
8
>>> a=[1,2,'a','b'] #list的长度，是元素的个数
>>> len(a)
4
>>> b=['yeashape']
>>> len(b)
1
```

下一讲继续`list`，有容乃大。

有容乃大的list(2)

对list的操作

list的长度

还记得str的长度怎么获得吗？其长度是什么含义呢？那种方法能不能用在list上面呢？效果如何？

做实验：

```
>>> name = 'qiwsir'
>>> type(name)
<type 'str'>
>>> len(name)
6
>>> lname = ['sir','qi']
>>> type(lname)
<type 'list'>
>>> len(lname)
2
>>> length = len(lname)
>>> length
2
>>> type(length)
<type 'int'>
```

实验结论：

- len(x)，对于list一样适用
- 得到的是list中元素个数
- 返回值是int类型

合并list

《有容乃大的list(1)》中，对list的操作提到了list.append(x)，也就是将某个元素x追加到已知的一个list后边。

除了将元素追加到list中，还能够将两个list合并，或者说将一个list追加到另外一个list中。按照前文的惯例，还是首先看[官方文档](#)中的描述：

list.extend(L)

Extend the list by appending all the items in the given list; equivalent to `a[len(a):] = L`.

向所有正在学习本内容的朋友提供一个成为优秀程序员的必备：看官方文档，是必须的。

官方文档的这句话翻译过来：

通过将所有元素追加到已知list来扩充它，相当于`a[len(a):]= L`

英语太烂，翻译太差。直接看例子，更明白

```
>>> la
[1, 2, 3]
>>> lb
['qiwsir', 'python']
>>> la.extend(lb)
>>> la
[1, 2, 3, 'qiwsir', 'python']
>>> lb
['qiwsir', 'python']
```

上面的例子，显示了如何将两个list，一个是la，另外一个lb，将lb追加到la的后面，也就是把lb中的所有元素加入到la中，即让la扩容。

学程序一定要有好奇心，我在交互环境中，经常实验一下自己的想法，有时候是比较愚蠢的想法。

```
>>> la = [1,2,3]
>>> b = "abc"
>>> la.extend(b)
>>> la
[1, 2, 3, 'a', 'b', 'c']
>>> c = 5
>>> la.extend(c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

从上面的实验中，看官能够有什么心得？原来，如果extend(str)的时候，str被以字符为单位拆开，然后追加到la里面。

如果extend的对象是数值型，则报错。

所以，extend的对象是一个list，如果是str，则python会先把它按照字符为单位转化为list再追加到已知list。

不过，别忘记了前面官方文档的后半句话，它的意思是：

```

>>> la
[1, 2, 3, 'a', 'b', 'c']
>>> lb
['qiwsir', 'python']
>>> la[len(la):]=lb
>>> la
[1, 2, 3, 'a', 'b', 'c', 'qiwsir', 'python']

```

`list.extend(L)` 等效于 `list[len(list):] = L`，`L`是待并入的list

联想到到[上一讲](#)中的一个list函数`list.append()`，这里的`extend`函数也是将另外的元素（只不过这个元素是列表）增加到一个已知列表中，那么两者有什么不一样呢？看下面例子：

```

>>> lst = [1,2,3]
>>> lst.append(["qiwsir","github"])
>>> lst
[1, 2, 3, ['qiwsir', 'github']]  #append的结果
>>> len(lst)
4

>>> lst2 = [1,2,3]
>>> lst2.extend(["qiwsir","github"])
>>> lst2
[1, 2, 3, 'qiwsir', 'github']  #extend的结果
>>> len(lst2)
5

```

`append`是整建制地追加，`extend`是个体化扩编。

list中某元素的个数

上面的`len(L)`，可得到list的长度，也就是list中有多少个元素。python的list还有一个操作，就是数一数某个元素在该list中出现多少次，也就是某个元素有多少个。官方文档是这么说的：

`list.count(x)`

Return the number of times x appears in the list.

一定要不断实验，才能理解文档中精炼的表达。

```

>>> la = [1,2,1,1,3]
>>> la.count(1)
3
>>> la.append('a')
>>> la.append('a')
>>> la
[1, 2, 1, 1, 3, 'a', 'a']
>>> la.count('a')
2
>>> la.count(2)
1
>>> la.count(5)      #NOTE:la中没有5,但是如果用这种方法找,不报错,返回的是数字0
0

```

元素在list中的位置

《有容乃大的list(1)》中已经提到，可以将list中的元素，从左向右依次从0开始编号，建立索引（如果从右向左，就从-1开始依次编号），通过索引能够提取出某个元素，或者某几个元素。就是如这样做：

```

>>> la
[1, 2, 3, 'a', 'b', 'c', 'qiwsir', 'python']
>>> la[2]
3
>>> la[2:5]
[3, 'a', 'b']
>>> la[:7]
[1, 2, 3, 'a', 'b', 'c', 'qiwsir']

```

如果考虑反过来的情况，能不能通过某个元素，找到它在list中的编号呢？

看官的需要就是python的方向，你想到，python就做到。

```

>>> la
[1, 2, 3, 'a', 'b', 'c', 'qiwsir', 'python']
>>> la.index(3)
2
>>> la.index('a')
3
>>> la.index(1)
0
>>> la.index('qi')      #如果不存在,就报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'qi' is not in list
>>> la.index('qiwsir')
6

```

`list.index(x)`，`x`是`list`中的一个元素，这样就能够检索到该元素在`list`中的位置了。这才是真正的索引，注意那个英文单词`index`。

依然是上一条官方解释：

`list.index(x)`

Return the index in the list of the first item whose value is `x`. It is an error if there is no such item.

是不是说的非常清楚明白了？

先到这里，下讲还继续有容乃大的`list`.

有容乃大的list(3)

现在是讲list的第三章了。俗话说，事不过三，不知道在开头，我也不知道这一讲是不是能够把基础的list知识讲完呢。哈哈。其实如果真正写文章，会在写完之后把这句话删掉的。而我则是完全像跟着官聊天一样，就不删除了。

继续。

对list的操作

向list中插入一个元素

前面有一个向list中追加元素的方法，那个追加是且只能是将新元素添加在list的最后一个。如：

```
>>> all_users = ["qiwsir", "github"]
>>> all_users.append("io")
>>> all_users
['qiwsir', 'github', 'io']
```

从这个操作，就可以说明list是可以随时改变的。这种改变的含义只它的大小即所容纳元素的个数以及元素内容，可以随时直接修改，而不用进行转换。这和str有着很大的不同。对于str，就不能进行字符的追加。请看官要注意比较，这也是str和list的重要区别。

与list.append(x)类似，list.insert(i,x)也是对list元素的增加。只不过是在任何位置增加一个元素。

我特别引导列为看官要通过[官方文档](#)来理解：

list.insert(i, x)

Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

这次就不翻译了。如果看不懂英语，怎么了解贵国呢？一定要硬着头皮看英语，不仅能够学好程序，更能...（此处省略两千字）

根据官方文档的说明，我们做下面的实验，请看官从实验中理解：

```

>>> all_users
['qiwsir', 'github', 'io']
>>> all_users.insert("python")      #list.insert(i,x),要求有两个参数，少了就报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: insert() takes exactly 2 arguments (1 given)

>>> all_users.insert(0,"python")
>>> all_users
['python', 'qiwsir', 'github', 'io']

>>> all_users.insert(1,"http://")
>>> all_users
['python', 'http://', 'qiwsir', 'github', 'io']

>>> length = len(all_users)
>>> length
5

>>> all_users.insert(length,"algorithm")
>>> all_users
['python', 'http://', 'qiwsir', 'github', 'io', 'algorithm']

```

小结：

- `list.insert(i,x)`, 将新的元素x 插入到原list中的list[i]前面
- 如果`i==len(list)`，意思是在后面追加，就等同于`list.append(x)`

删除list中的元素

list中的元素，不仅能增加，还能被删除。删除list元素的方法有两个，它们分别是：

`list.remove(x)`

Remove the first item from the list whose value is x. It is an error if there is no such item.

`list.pop([i])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

我这里讲授python，有一个习惯，就是用学习物理的方法。如果看官当初物理没有学好，那么一定是没有用这种方法，或者你的老师没有用这种教学法。这种方法就是：自己先实验，然后总结规律。

先实验`list.remove(x)`，注意看上面的描述。这是一个能够删除list元素的方法，同时上面说明告诉我们，如果x没有在list中，会报错。

```
>>> all_users
['python', 'http://', 'qiwsir', 'github', 'io', 'algorithm']
>>> all_users.remove("http://")
>>> all_users      #的确是把"http://"删除了
['python', 'qiwsir', 'github', 'io', 'algorithm']

>>> all_users.remove("tianchao")      #原list中没有"tianchao"，要删除，就报错。
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

注意两点：

- 如果正确删除，不会有任何反馈。没有消息就是好消息。
- 如果所删除的内容不在list中，就报错。注意阅读报错信息：`x not in list`

看官是不是想到一个问题？如果能够在删除之前，先判断一下这个元素是不是在list中，在就删，不在就不删，不是更智能吗？

如果看官想到这里，就是在编程的旅程上一进步。`python`的确让我们这么做。

```
>>> all_users
['python', 'qiwsir', 'github', 'io', 'algorithm']
>>> "python" in all_users      #这里用in来判断一个元素是否在list中，在则返回True，否则返回False
True

>>> if "python" in all_users:
...     all_users.remove("python")
...     print all_users
... else:
...     print "'python' is not in all_users"
...
['qiwsir', 'github', 'io', 'algorithm']      #删除了"python"元素

>>> if "python" in all_users:
...     all_users.remove("python")
...     print all_users
... else:
...     print "'python' is not in all_users"
...
'python' is not in all_users      #因为已经删除了，所以就没有了。
```

上述代码，就是两段小程序，我是在交互模式中运行的，相当于小实验。

另外一个删除`list.pop([i])`会怎么样呢？看看文档，做做实验。

```
>>> all_users
['qiwsir', 'github', 'io', 'algorithm']
>>> all_users.pop()      #list.pop([i]),圆括号里面是[i]，表示这个序号是可选的
'algorithm'              #如果不写，就如同这个操作，默认删除最后一个，并且将该结果返回

>>> all_users
['qiwsir', 'github', 'io']

>>> all_users.pop(1)      #指定删除编号为1的元素"github"
'github'

>>> all_users
['qiwsir', 'io']
>>> all_users.pop()
'io'

>>> all_users           #只有一个元素了，该元素编号是0
['qiwsir']
>>> all_users.pop(1)    #但是非要删除编号为1的元素，结果报错。注意看报错信息
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop index out of range      #删除索引超出范围，就是1不在list的编号范围之内
```

给看官留下一个思考题，如果要向前面那样，能不能事先判断一下要删除的编号是不是在list的长度范围（用len(list)获取长度）以内？然后进行删除或者不删除操作。

list是一个有意思的东西，内涵丰富。看来下一讲还要继续讲list。并且可能会做一个有意思的游戏。请期待。

有容乃大的list(4)

list的话题的确不少，而且，在编程中，用途也非常多。

有看官可能要问了，如果要生成一个list，除了要把元素一个一个写上之外，有没有能够让计算机自己按照某个规律生成list的方法呢？

如果你提出了这个问题，充分说明你是一个“懒人”，不过这不是什么坏事情，这个世界就是因为“懒人”的存在而进步。“懒人”其实不懒。

对list的操作

range(start,stop)生成数字list

range(start, stop[, step])是一个内置函数。

要研究清楚一些函数特别是内置函数的功能，建议看官首先要明白内置函数名称的含义。因为在python中，名称不是随便取的，是代表一定意义的。关于取名字问题，可以看参考本系列的：[永远强大的函数](#)中的《取名字的学问》部分内容。

range

n. 范围；幅度；排；山脉 vi. (在...内) 变动；平行，列为一行；延伸；漫游；射程达到
vt. 漫游；放牧；使并列；归类于；来回走动

在具体实验之前，还是按照管理，摘抄一段[官方文档的原话](#)，让我们能够深刻理解之：

This is a versatile function to create lists containing arithmetic progressions. It is most often used in for loops. The arguments must be plain integers. If the step argument is omitted, it defaults to 1. If the start argument is omitted, it defaults to 0. The full form returns a list of plain integers [start, start + step, start + 2 step, ...]. If step is positive, the last element is the largest start + i step less than stop; if step is negative, the last element is the smallest start + i * step greater than stop. step must not be zero (or else ValueError is raised).

从这段话，我们可以得出关于range()函数的以下几点：

- 这个函数可以创建一个数字元素组成的列表。
- 这个函数最常用于for循环（关于for循环，马上就要涉及到了）
- 函数的参数必须是整数，默认从0开始。返回值是类似[start, start + step, start + 2*step, ...]的列表。
- step默认值是1。如果不写，就是按照此值。

- 如果`step`是正数，返回`list`的最最后的值不包含`stop`值，即`start+istep`这个值小于`stop`；如果`step`是负数，`start+istep`的值大于`stop`。
- `step`不能等于零，如果等于零，就报错。

在实验开始之前，再解释`range(start,stop,[step])`的含义：

- `start`：开始数值，默認為0，也就是如果不写这项，就是认为`start=0`
- `stop`：结束的数值，必须要写的。
- `step`：变化的步长，默認是1，也就是不写，就是认为步长为1。坚决不能为0

实验开始，请以各项对照前面的讲述：

```
>>> range(9)           #stop=9，别的都没有写，含义就是range(0,9,1)
[0, 1, 2, 3, 4, 5, 6, 7, 8] #从0开始，步长为1，增加，直到小于9的那个数
>>> range(0,9)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> range(0,9,1)
[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> range(1,9)         #start=1
[1, 2, 3, 4, 5, 6, 7, 8]

>>> range(0,9,2)       #step=2, 每个元素等于start+i*step ,
[0, 2, 4, 6, 8]
```

仅仅解释一下`range(0,9,2)`

- 如果是从0开始，步长为1，可以写成`range(9)`的样子，但是，如果步长为2，写成`range(9,2)`的样子，计算机就有点糊涂了，它会认为`start=9,stop=2`。所以，在步长不为1的时候，切忌，要把`start`的值也写上。
- `start=0,step=2,stop=9`.`list`中的第一个值是`start=0`，第二个值是`start+1*step=2`（注意，这里是1，不是2，不要忘记，前面已经讲过，不论是`list`还是`str`，对元素进行编号的时候，都是从0开始的），第 n 个值就是`start+(n-1)*step`。直到小于`stop`前的那个值。

熟悉了上面的计算过程，看看下面的输入谁是什么结果？

```
>>> range(-9)
```

我本来期望给我返回`[0,-1,-2,-3,-4,-5,-6,-7,-8]`，我的期望能实现吗？

分析一下，这里`start=0,step=1,stop=-9`。

第一个值是0；第二个是`start+1*step`，将上面的数代入，应该是1，但是最后一个还是-9，显然出现问题了。但是，`python`在这里不报错，它返回的结果是：

```
>>> range(-9)
[]
>>> range(0, -9)
[]
>>> range(0)
[]
```

报错和返回结果，是两个含义，虽然返回的不是我们要的。应该如何修改呢？

```
>>> range(0, -9, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8]
>>> range(0, -9, -2)
[0, -2, -4, -6, -8]
```

有了这个内置函数，很多事情就简单了。比如：

```
>>> range(0, 100, 2)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96]
```

100以内的自然数中的偶数组成的list，就非常简单地搞定了。

思考一个问题，现在有一个列表，比如是

["I", "am", "a", "pythoner", "I", "am", "learning", "it", "with", "qiwsir"]，要得到这个list的所有序号组成的list，但是不能一个一个用手指头来数。怎么办？

请沉思两分钟之后，自己实验一下，然后看下面。

```
>>> pythoner
['I', 'am', 'a', 'pythoner', 'I', 'am', 'learning', 'it', 'with', 'qiwsir']
>>> py_index = range(len(pythoner))      #以len(pythoner)为stop的值
>>> py_index
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

再用手指头指着pythoner里面的元素，数一数，是不是跟结果一样。

排排坐，分果果

排序，不管在现实还是在网络上都是随处可见的。梁山好汉要从第一个排序到第108个，这是一个不容易搞定的活。

前面提到的内置函数range()得到的结果，就是一个排好序的。对于一个没有排好序的list，怎么排序呢？

有两个方法可以实现对list的排序：

- `list.sort(cmp=None, key=None, reverse=False)`
- `sorted(iterable[, cmp[, key[, reverse]]])`

通过下面的实验，可以理解如何排序的方法

```
>>> number = [1,4,6,2,9,7,3]
>>> number.sort()
>>> number
[1, 2, 3, 4, 6, 7, 9]

>>> number = [1,4,6,2,9,7,3]
>>> number
[1, 4, 6, 2, 9, 7, 3]
>>> sorted(number)
[1, 2, 3, 4, 6, 7, 9]

>>> number = [1,4,6,2,9,7,3]
>>> number
[1, 4, 6, 2, 9, 7, 3]
>>> number.sort(reverse=True)    #开始实现倒序
>>> number
[9, 7, 6, 4, 3, 2, 1]

>>> number = [1,4,6,2,9,7,3]
>>> number
[1, 4, 6, 2, 9, 7, 3]
>>> sorted(number,reverse=True)
[9, 7, 6, 4, 3, 2, 1]
```

其实，在高级语言中，排序是一个比较热门的话题，如果有兴趣的读者，可以到我写的[有关算法](#)中查看有关排序的话题。

至此，有关list的基本操作的内置函数，就差不多了。不过最后，还要告诉看官们一个学习方法。因为python的内置函数往往不少，有时候光凭教程，很难学到全部，那么，最关键地是要自己会查找都有哪些函数可以用。怎么查找呢？

一个非常重要的方法

假设有一个list，如何知道它所拥有的内置函数呢？请用`help()`，帮助我吧。

```
>>> help(list)
```

就能够看到所有的关于list的函数，以及该函数的使用方法。

list和str比较

list和str两种类型数据，有不少相似的地方，也有很大的区别。本讲对她们做个简要比较，同时也是对前面有关两者知识复习一下，所谓“温故而知新”。

相同点

都属于序列类型的数据

所谓序列类型的数据，就是说它的每一个元素都可以通过指定一个编号，行话叫做“偏移量”的方式得到，而要想一次得到多个元素，可以使用切片。偏移量从0开始，总元素数减1结束。

例如：

```
>>> welcome_str = "Welcome you"
>>> welcome_str[0]
'W'
>>> welcome_str[1]
'e'
>>> welcome_str[len(welcome_str)-1]
'u'
>>> welcome_str[:4]
'Welc'
>>> a = "python"
>>> a*3
'pythonpythonpython'

>>> git_list = ["qiwsir", "github", "io"]
>>> git_list[0]
'qiwsir'
>>> git_list[len(git_list)-1]
'io'
>>> git_list[0:2]
['qiwsir', 'github']
>>> b = ['qiwsir']
>>> b*7
['qiwsir', 'qiwsir', 'qiwsir', 'qiwsir', 'qiwsir', 'qiwsir', 'qiwsir']
```

对于此类数据，下面一些操作是类似的：

```
>>> first = "hello,world"
>>> welcome_str
'Welcome you'
>>> first+","+welcome_str    #用+号连接str
'hello,world,Welcome you'
>>> welcome_str              #原来的str没有受到影响，即上面的+号连接后从新生成了一个字符串
'Welcome you'
>>> first
'hello,world'

>>> language = ['python']
>>> git_list
['qiwsir', 'github', 'io']
>>> language + git_list      #用+号连接list，得到一个新的list
['python', 'qiwsir', 'github', 'io']
>>> git_list
['qiwsir', 'github', 'io']
>>> language
['python']

>>> len(welcome_str)        #得到字符数
11
>>> len(git_list)           #得到元素数
3
```

区别

list和str的最大区别是：list是可以改变的，str不可变。这个怎么理解呢？

首先看对list的这些操作，其特点是在原处将list进行了修改：

```

>>> git_list
['qiwsir', 'github', 'io']

>>> git_list.append("python")
>>> git_list
['qiwsir', 'github', 'io', 'python']

>>> git_list[1]
'github'
>>> git_list[1] = 'github.com'
>>> git_list
['qiwsir', 'github.com', 'io', 'python']

>>> git_list.insert(1,"algorithm")
>>> git_list
['qiwsir', 'algorithm', 'github.com', 'io', 'python']

>>> git_list.pop()
'python'

>>> del git_list[1]
>>> git_list
['qiwsir', 'github.com', 'io']

```

以上这些操作，如果用在str上，都会报错，比如：

```

>>> welcome_str
'Welcome you'

>>> welcome_str[1]='E'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>> del welcome_str[1]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object doesn't support item deletion

>>> welcome_str.append("E")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'append'

```

如果要修改一个str，不得不这样。

```

>>> welcome_str
'Welcome you'
>>> welcome_str[0]+"E"+welcome_str[2:] #从新生成一个str
'WEcome you'
>>> welcome_str #对原来的没有任何影响
'Welcome you'

```

其实，在这种做法中，相当于从新生成了一个str。

多维list

这个也应该算是两者的区别了，虽然有点牵强。在str中，里面的每个元素只能是字符，在list中，元素可以是任何类型的数据。前面见的多是数字或者字符，其实还可以这样：

```

>>> matrix = [[1,2,3],[4,5,6],[7,8,9]]
>>> matrix = [[1,2,3],[4,5,6],[7,8,9]]
>>> matrix[0][1]
2
>>> mult = [[1,2,3],['a','b','c'],'d','e']
>>> mult
[[1, 2, 3], ['a', 'b', 'c'], 'd', 'e']
>>> mult[1][1]
'b'
>>> mult[2]
'd'

```

以上显示了多维list以及访问方式。在多维的情况下，里面的list也跟一个前面元素一样对待。

list和str转化

str.split()

这个内置函数实现的是将str转化为list。其中str=""是分隔符。

在看例子之前，请看官在交互模式下做如下操作：

```
>>>help(str.split)
```

得到了对这个内置函数的完整说明。特别强调：这是一种非常好的学习方法

split(...) S.split([sep [,maxsplit]]) -> list of strings

Return a list of the words in the string S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

不管是否看懂上面这段话，都可以看例子。还是希望看官能够理解上面的内容。

```
>>> line = "Hello.I am qiwsi.Welcome you."
>>> line.split(".")      #以英文的句点为分隔符，得到list
['Hello', 'I am qiwsi', 'Welcome you', '']

>>> line.split(".",1)   #这个1，就是表达了上文中的：If maxsplit is given, at most maxsplit spl
['Hello', 'I am qiwsi.Welcome you.']

>>> name = "Albert Ainstain"    #也有可能用空格来做为分隔符
>>> name.split(" ")
['Albert', 'Ainstain']
```

下面的例子，让你更有点惊奇了。

```
>>> s = "I am, writing\npython\tbook on line"    #这个字符串中有空格，逗号，换行\n，tab缩进\t 符
>>> print s          #输出之后的样式
I am, writing
python  book on line
>>> s.split()        #用split()，但是括号中不输入任何参数
['I', 'am,', 'writing', 'python', 'book', 'on', 'line']
```

如果split()不输入任何参数，显示就是见到任何分割符号，就用其分割了。

"[sep]".join(list)

join可以说是split的逆运算，举例：

```
>>> name
['Albert', 'Ainstain']
>>> "".join(name)      #将list中的元素连接起来，但是没有连接符，表示一个一个紧邻着
'AlbertAinstain'
>>> ". ".join(name)    #以英文的句点做为连接分隔符
'Albert.Ainstain'
>>> " ".join(name)     #以空格做为连接的分隔符
'Albert Ainstain'
```

回到上面那个神奇的例子中，可以这么使用join.

```
>>> s = "I am, writing\npython\tbook on line"
>>> print s
I am, writing
python  book on line
>>> s.split()
['I', 'am', 'writing', 'python', 'book', 'on', 'line']
>>> ".join(s.split())          #重新连接，不过有一点遗憾，am后面逗号还是有的。怎么去掉？
'I am, writing python book on line'
```

公告：

有朋友愿意学习python, 恭请到我的github上follower我，并且可以给我发邮件，也可以在微博上关注我。更多有关信息请看：[易水禾](http://qiwsir.github.io)：<http://qiwsir.github.io>

画圈还不简单吗？

画圈？换一个说法就是循环。循环，是高级语言编程中重要的工作。现实生活中，很多事情都是在循环，日月更迭，斗转星移，无不是循环；王朝更迭，寻常百姓，也都是循环。

在python中，循环有一个语句：for语句。

简单的for循环例子

```
>>> hello = "world"
>>> for i in hello:
...     print i
...
w
o
r
l
d
```

上面这个for循环是怎么工作的呢？

1. hello这个变量引用的是"world"这个str类型的数据
2. 变量 i 通过hello找到它所引用的"world",然后从第一字符开始，依次获得该字符的引用。
3. 当 i="w"的时候，执行print i，打印出了字母w，结束之后循环第二次，让 i="e"，然后执行print i,打印出字母e，如此循环下去，一直到最后一个字符被打印出来，循环自动结束

顺便补充一个print的技巧，上面的打印结果是竖着排列，也就是每打印一个之后，就自动换行。如果要让打印的在一行，可以用下面的方法，在打印的后面加一个逗号（英文）

```
>>> for i in hello:
...     print i,
...
w o r l d

>>> for i in hello:
...     print i+",",    #为了美观，可以在每个字符后面加一个逗号分割
...
w, o, r, l, d,
>>>
```

因为可以通过使用索引编号（偏移量）做为下表，得到某个字符。所以，还可以通过下面的循环方式实现上面代码中同样功能：

```
>>> for i in range(len(hello)):
...     print hello[i]
...
w
o
r
l
d
```

其工作方式是：

1. `len(hello)`得到`hello`引用的字符串的长度，为5
2. `range(len(hello))`，就是`range(5)`，也就是`[0, 1, 2, 3, 4]`，对应这"world"每个字母的编号，即偏移量。
3. `for i in range(len(hello))`，就相当于`for i in [0,1,2,3,4]`，让`i`依次等于`list`中的各个值。当`i=0`时，打印`hello[0]`，也就是第一个字符。然后顺序循环下去，直到最后一个`i=4`为止。

以上的循环举例中，显示了对字`str`的字符依次获取，也涉及了`list`，感觉不过瘾呀。那好，看下面对`list`的循环：

```
>>> ls_line
['Hello', 'I am qiwsir', 'welcome you', '']
>>> for word in ls_line:
...     print word
...
Hello
I am qiwsir
welcome you

>>> for i in range(len(ls_line)):
...     print ls_line[i]
...
Hello
I am qiwsir
welcome you
```

上一个台阶

我们已经理解了`for`语句的基本工作流程，如果写一个一般化的公式，可以这么表示：

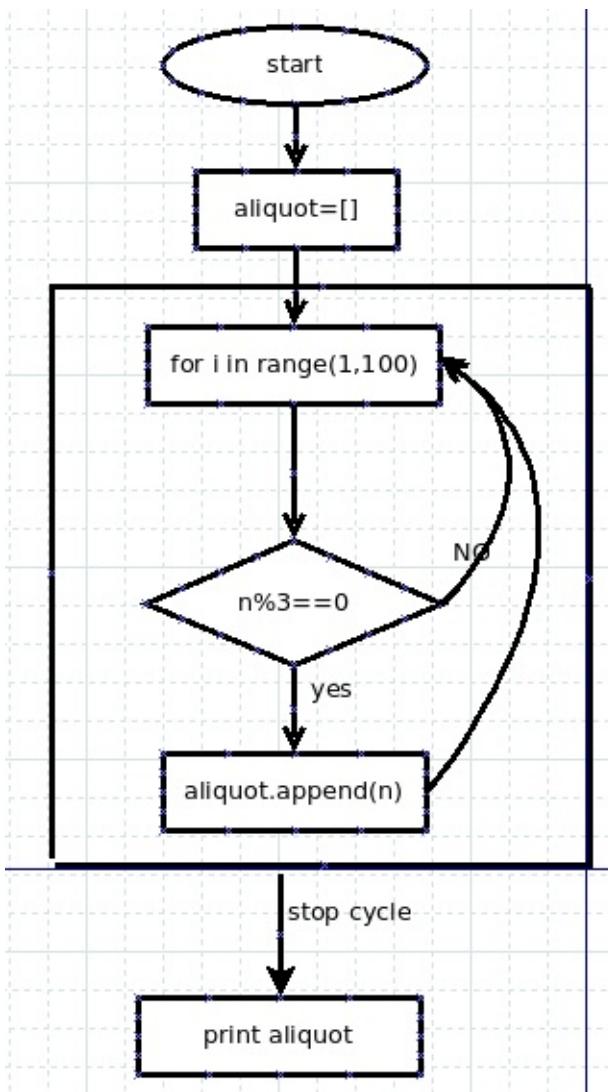
```
for 循环规则：
    操作语句
```

用`for`语句来解决一个实际问题。

例：找出100以内的能够被3整除的正整数。

分析：这个问题有两个限制条件，第一是100以内的正整数，根据前面所学，可以用`range(1,100)`来实现；第二个是要解决被3整除的问题，假设某个正整数n，这个数如果能够被3整除，也就是 $n \% 3 == 0$ 。那么如何得到n呢，就是要用`for`循环。

以上做了简单分析，要实现流程，还需要细化一下。按照前面曾经讲授过的一种方法，要画出问题解决的流程图。



下面写代码就是按图索骥了。

代码：

```
#!/usr/bin/env python
#coding:utf-8

aliquot = []

for n in range(1,100):
    if n%3 == 0:
        aliquot.append(n)

print aliquot
```

代码运行结果：

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69,
```



这里仅仅列举一个简单的例子，看官可以在这个例子基础上深入：打印某范围内的偶数/奇数等。

如果要对list的循环进行深入了解的，可以到我专门撰写的[python and algorithm](#)里面阅读有关文章

再深点，更懂list

对于list，由于她的确非常非常庞杂，在python中应用非常广泛，所以，虽然已经介绍完毕了基础内容，这里还要用一讲深入一点点，往往越深入越...

list解析

先看下面的例子，这个例子是想得到1到9的每个整数的平方，并且将结果放在list中打印出来

```
>>> power2 = []
>>> for i in range(1,10):
...     power2.append(i*i)
...
>>> power2
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

python有一个非常有意思的功能，就是list解析，就是这样的：

```
>>> squares = [x**2 for x in range(1,10)]
>>> squares
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

看到这个结果，看官还不惊叹吗？这就是python，追求简洁优雅的python！

其官方文档中有这样一段描述，道出了list解析的真谛：

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

还记得[前面一讲](#)中的那个问题吗？

找出100以内的能够被3整除的正整数。

我们用的方法是：

```

aliquot = []

for n in range(1,100):
    if n%3 == 0:
        aliquot.append(n)

print aliquot

```

好了。现在用list解析重写，会是这样的：

```

>>> aliquot = [n for n in range(1,100) if n%3==0]
>>> aliquot
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69,

```

震撼了。绝对牛X！

再来一个，是网友ccbikai提供的，比牛X还牛X。

```

>>> print range(3,100,3)
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69,

```

这就是python有意思的地方，也是计算机高级语言编程有意思的地方，你只要动脑筋，总能找到惊喜的东西。

其实，不仅仅对数字组成的list，所有的都可以如此操作。请在平复了激动的心之后，默默地看下面的代码，感悟一下list解析的魅力。

```

>>> mybag = [' glass',' apple','green leaf ']      #有的前面有空格，有的后面有空格
>>> [one.strip() for one in mybag]                 #去掉元素前后的空格
['glass', 'apple', 'green leaf']

```

enumerate

这是一个有意思的内置函数，本来我们可以通过for i in range(len(list))的方式得到一个list的每个元素编号，然后在用list[i]的方式得到该元素。如果要同时得到元素编号和元素怎么办？就是这样了：

```
>>> for i in range(len(week)):
...     print week[i] + ' is ' + str(i)      #注意，i是int类型，如果和前面的用+连接，必须是str类型
...
monday is 0
sunday is 1
friday is 2
```

python中提供了一个内置函数enumerate，能够实现类似的功能

```
>>> for (i,day) in enumerate(week):
...     print day + ' is ' + str(i)
...
monday is 0
sunday is 1
friday is 2
```

算是一个有意思的内置函数了，主要是提供一个简单快捷的方法。

官方文档是这么说的：

Return an enumerate object. sequence must be a sequence, an iterator, or some other object which supports iteration. The next() method of the iterator returned by enumerate() returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating over sequence:

顺便抄录几个例子，供看官欣赏，最好实验一下。

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

在这里有类似(0,'Spring')这样的东西，这是另外一种数据类型，待后面详解。

下面将enumerate函数和list解析联合起来，同时显示，在进行list解析的时候，也可以包含进函数（关于函数，可以参考的章节有：[初始强大的函数](#)，[重回函数](#)）。

```
>>> def treatment(pos, element):
...     return "%d: %s"%(pos,element)
...
>>> seq = ["qiwsir","qiwsir.github.io","python"]
>>> [ treatment(i, ele) for i,ele in enumerate(seq) ]
['0: qiwsir', '1: qiwsir.github.io', '2: python']
```

看官也可以用[小话题大函数](#)中的lambda函数来写上面的代码：

```
>>> seq = ["qiwsir", "qiwsir.github.io", "python"]
>>> foo = lambda i,ele:"%d:%s"%(i,ele)          #lambda函数，给代码带来了简介
>>> [foo(i,ele) for i,ele in enumerate(seq)]
['0:qiwsir', '1:qiwsir.github.io', '2:python']
```

字典，你还记得吗？

字典，这个东西你现在还用吗？随着网络的发展，用的人越来越少了。不少人习惯于在网上搜索，不仅有web版，以至于已经有手机版的各种字典了。。我曾经用过一本小小的《新华字典》。

《新华字典》是中国第一部现代汉语字典。最早的名字叫《伍记小字典》，但未能编纂完成。自1953年，开始重编，其凡例完全采用《伍记小字典》。从1953年开始出版，经过反复修订，但是以1957年商务印书馆出版的《新华字典》作为第一版。原由新华辞书社编写，1956年并入中科院语言研究所（现中国社科院语言研究所）词典编辑室。新华字典由商务印书馆出版。历经几代上百名专家学者10余次大规模的修订，重印200多次。成为迄今为止世界出版史上最高发行量的字典。

这里讲到字典，不是为了叙旧。而是提醒看官想想我们如何使用字典：先查索引（不管是拼音还是偏旁查字），然后通过索引找到相应内容。

这种方法能够快捷的找到目标。

在python中，也有一种数据与此相近，不仅相近，这种数据的名称就叫做**dictionary**，翻译过来是字典，类似于前面的**int/str/list**，这种类型数据名称是:**dict**

依据管理，要知道如何建立**dict**和它有关属性方法。

因为已经有了此前的基础，所以，学这个就可以加快了。

前面曾经建议看官一个很好的学习探究方法，比如想了解**str**的有关属性方法，可以在交互模式下使用：

```
>>>help(str)
```

将得到所有的有关内容。

现在换一个，使用**dir**，也能得到相同的结果。只是简单一些罢了。请在交互模式下：

```
>>> dir(dict)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__',

```

以`__`（双下划线）开头的先不管。看后面的。如果要想深入了解，可以这样：

```
>>> help(dict.values)
```

然后出现：

```
Help on method_descriptor:

values(...)

    D.values() -> list of D's values
(END)
```

也就是在这里显示出了values这个内置函数的使用方法。敲击键盘上的q键退回。

概述

python中的dict具有如下特点：

- dict是可变的
- dict可以存储任意数量的Python对象
- dict可以存储任何python数据类型
- dict以：key:value，即“键：值”对的形式存储数据，每个键是唯一的。
- dict也被称为关联数组或哈希表。

以上诸条，如果还不是很理解，也没有关系，通过下面的学习，特别是通过各种实验，就能理解了。

创建dict

话说创建dict的方法可是远远多于前面的int/str/list，为什么会多呢？一般规律是复杂点的东西都会有多种渠道生成，这也是从安全便捷角度考虑吧。

方法1：

创建一个空的dict，这个空dict，可以在以后向里面加东西用。

```
>>> mydict = {}
>>> mydict
{}
```

创建有内容的dict。

```
>>> person = {"name": "qiwsir", "site": "qiwsir.github.io", "language": "python"}
>>> person
{'name': 'qiwsir', 'language': 'python', 'site': 'qiwsir.github.io'}
```

"name":"qiwsir"就是一个键值对，前面的name叫做键（key），后面的qiwsir是前面的键所对应的值(value)。在一个dict中，键是唯一的，不能重复；值则是对应于键，值可以重复。键值之间用(:)英文的分号，每一对键值之间用英文的逗号(,)隔开。

```
>>> person['name2']="qiwsir"      #这是一种向dict中增加键值对的方法
>>> person
{'name2': 'qiwsir', 'name': 'qiwsir', 'language': 'python', 'site': 'qiwsir.github.io'}
```

如下，演示了从一个空的dict开始增加内容的过程：

```
>>> mydict = {}
>>> mydict
{}
>>> mydict["site"] = "qiwsir.github.io"
>>> mydict[1] = 80
>>> mydict[2] = "python"
>>> mydict["name"] = ["zhangsan", "lisi", "wangwu"]
>>> mydict
{1: 80, 2: 'python', 'site': 'qiwsir.github.io', 'name': ['zhangsan', 'lisi', 'wangwu']}

>>> mydict[1] = 90  #如果这样，则是修改这个键的值
>>> mydict
{1: 90, 2: 'python', 'site': 'qiwsir.github.io', 'name': ['zhangsan', 'lisi', 'wangwu']}
```

方法2：

```
>>> name = (["first", "Google"], ["second", "Yahoo"])      #这是另外一种数据类型，称之为元组，后面
>>> website = dict(name)
>>> website
{'second': 'Yahoo', 'first': 'Google'}
```

方法3：

这个方法，跟上面的不同在于使用fromkeys

```
>>> website = {}.fromkeys(("third", "forth"), "facebook")
>>> website
{'forth': 'facebook', 'third': 'facebook'}
```

需要提醒的是，这种方法是从新建立一个dict。

访问dict的值

因为dict是以键值对的形式存储数据的，所以，只要知道键，就能得到值。这本质上就是一种映射关系。

```
>>> person
{'name2': 'qiwsir', 'name': 'qiwsir', 'language': 'python', 'site': 'qiwsir.github.io'}
>>> person['name']
'qiwsir'
>>> person['language']
'python'
>>> site = person['site']
>>> print site
qiwsir.github.io
```

如同前面所讲，通过键能够增加dict中的值，通过键能够改变dict中的值，通过键也能够访问dict中的值。

看官可以跟list对比一下。如果我们访问list中的元素，可以通过索引值得到（list[i]），如果是让机器来巡回访问，就可以用for语句。复习一下：

```
>>> person_list = ["qiwsir", "Newton", "Boolean"]
>>> for name in person_list:
...     print name
...
qiwsir
Newton
Boolean
```

那么，dict是不是也可以用for语句来循环访问呢？当然可以，来看例子：

```
>>> person
{'name2': 'qiwsir', 'name': 'qiwsir', 'language': 'python', 'site': 'qiwsir.github.io'}
>>> for key in person:
...     print person[key]
...
qiwsir
qiwsir
python
qiwsir.github.io
```

知识

什么是关联数组？以下解释来自[维基百科](#)

在计算机科学中，关联数组（英语：Associative Array），又称映射（Map）、字典（Dictionary）是一个抽象的数据结构，它包含着类似于（键，值）的有序对。一个关联数组中的有序对可以重复（如C++中的multimap）也可以不重复（如C++中的map）。

这种数据结构包含以下几种常见的操作：

- 1.向关联数组添加配对
- 2.从关联数组内删除配对
- 3.修改关联数组内的配对
- 4.根据已知的键寻找配对

字典问题是设计一种能够具备关联数组特性的数据结构。解决字典问题的常用方法，是利用散列表，但有些情况下，也可以直接使用有地址的数组，或二叉树，和其他结构。

许多程序设计语言内置基本的数据类型，提供对关联数组的支持。而Content-addressable memory则是硬件层面上实现对关联数组的支持。

什么是哈希表？关于哈希表的叙述比较多，这里仅仅截取了概念描述，更多的可以到[维基百科](#)上阅读。

散列表（Hash table，也叫哈希表），是根据关键字（Key value）而直接访问在内存存储位置的数据结构。也就是说，它通过把键值通过一个函数的计算，映射到表中一个位置来访问记录，这加快了查找速度。这个映射函数称做散列函数，存放记录的数组称做散列表。

dict()的操作方法

dict的很多方法跟list有类似的地方，下面一一道来，并且会跟list做一个对比

嵌套

嵌套在list中也存在，就是元素是list，在dict中，也有类似的样式：

```
>>> a_list = [[1,2,3],[4,5],[6,7]]
>>> a_list[1][1]
5
>>> a_dict = {1:{"name":"qiwsir"},2:"python","email":"qiwsir@gmail.com"}
>>> a_dict
{1: {'name': 'qiwsir'}, 2: 'python', 'email': 'qiwsir@gmail.com'}
>>> a_dict[1]['name']      #一个嵌套的dict访问其值的方法：一层一层地写出键
'qiwsir'
```

获取键、值

在[上一讲](#)中，已经知道可以通过dict的键得到其值。例上面的例子。

还有别的方法得到键值吗？有！python一般不是只有一个方法实现某个操作的。

```
>>> website = {1:"google","second":"baidu",3:"facebook","twitter":4}

>>>#用d.keys()的方法得到dict的所有键，结果是list
>>> website.keys()
[1, 'second', 3, 'twitter']

>>>#用d.values()的方法得到dict的所有值，如果里面没有嵌套别的dict，结果是list
>>> website.values()
['google', 'baidu', 'facebook', 4]

>>>#用items()的方法得到了一组一组的键值对，
>>>#结果是list，只不过list里面的元素是元组
>>> website.items()
[(1, 'google'), ('second', 'baidu'), (3, 'facebook'), ('twitter', 4)]
```

从上面的结果中，我们就可以看出，还可以用for语句循环得到相应内容。例如：

```
>>> for key in website.keys():
...     print key,type(key)
...
1 <type 'int'>
second <type 'str'>
3 <type 'int'>
twitter <type 'str'>

>>>#下面的方法和上面的方法是一样的
>>> for key in website:
...     print key,type(key)
...
1 <type 'int'>
second <type 'str'>
3 <type 'int'>
twitter <type 'str'>
```

以下两种方法等效：

```
>>> for value in website.values():
...     print value
...
google
baidu
facebook
4

>>> for key in website:
...     print website[key]
...
google
baidu
facebook
4
```

下面的方法又是等效的：

```

>>> for k,v in website.items():
...     print str(k)+":"+str(v)
...
1:google
second:baidu
3:facebook
twitter:4

>>> for k in website:
...     print str(k)+":"+str(website[k])
...
1:google
second:baidu
3:facebook
twitter:4

```

下面的方法也能得到键值，不过似乎要多敲键盘

```

>>> website
{1: 'google', 'second': 'baidu', 3: 'facebook', 'twitter': 4}
>>> website.get(1)
'google'
>>> website.get("second")
'baidu'

```

其它几种常用方法

dict中的方法在这里不做过多的介绍，因为前面一节中已经列出来类，看官如果有兴趣可以一尝试。下面列出几种常用的

```

>>> len(website)
4
>>> website
{1: 'google', 'second': 'baidu', 3: 'facebook', 'twitter': 4}

>>> new_web = website.copy()      #拷贝一份，这个拷贝也叫做浅拷贝，对应着还有深拷贝。
>>> new_web                      #两者区别，可以google一下。
{1: 'google', 'second': 'baidu', 3: 'facebook', 'twitter': 4}

```

删除键值对的方法有两个，但是两者有一点区别

```

>>> #d.pop(key)，根据key删除相应的键值对，并返回该值
>>> new_web.pop('second')
'baidu'

>>> del new_web[3]      #没有返回值，如果删除键不存在，返回错误
>>> new_web
{1: 'google', 'twitter': 4}
>>> del new_web[9]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 9

```

用d.update(d2)可以把d2合并到d中。

```

>>> cnweb
{'qq': 'first in cn', 'python': 'qiwsir.github.io', 'alibaba': 'Business'}
>>> website
{1: 'google', 'second': 'baidu', 3: 'facebook', 'twitter': 4}

>>> website.update(cnweb)  #把cnweb合并到website内
>>> website
{'qq': 'first in cn', 1: 'google', 'second': 'baidu', 3: 'facebook', 'python': 'qiwsir.gi
>>> cnweb
#not changed
{'qq': 'first in cn', 'python': 'qiwsir.github.io', 'alibaba': 'Business'}

```

在本讲最后，要提醒看官，在python3中，dict有不少变化，比如能够进行字典解析，就类似列表解析那样，这可是非常有意思的东西哦。

有点简约的元组

关于元组，上一讲中涉及到了这个名词。本讲完整地讲述它。

先看一个例子：

```
>>> #变量引用str
>>> s = "abc"
>>> s
'abc'

>>> #如果这样写，就会是...
>>> t = 123, 'abc', ["come", "here"]
>>> t
(123, 'abc', ['come', 'here'])
```

上面例子中看到的变量t，并没有报错，也没有“最后一个有效”，而是将对象做为一个新的数据类型：tuple（元组），赋值给了变量t。

元组是用圆括号括起来的，其中的元素之间用逗号隔开。（都是英文半角）

tuple是一种序列类型的数据，这点上跟list/str类似。它的特点就是其中的元素不能更改，这点上跟list不同，倒是跟str类似；它的元素又可以是任何类型的数据，这点上跟list相同，但不同于str。

```
>>> t = 1, "23", [123, "abc"], ("python", "learn")    #元素多样性，近list
>>> t
(1, '23', [123, 'abc'], ('python', 'learn'))

>>> t[0] = 8                                         #不能原地修改，近str
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

>>> t.append("no")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

从上面的简单比较似乎可以认为，tuple就是一个融合了部分list和部分str属性的杂交产物。此言有理。

像list那样访问元素和切片

先复习**list**中的一点知识：

```
>>> one_list = ["python", "qiwsir", "github", "io"]
>>> one_list[2]
'github'
>>> one_list[1:]
['qiwsir', 'github', 'io']
>>> for word in one_list:
...     print word
...
python
qiwsir
github
io
>>> len(one_list)
4
```

下面再实验一下，上面的**list**如果换成**tuple**是否可行

```
>>> t
(1, '23', [123, 'abc'], ('python', 'learn'))
>>> t[2]
[123, 'abc']
>>> t[1:]
('23', [123, 'abc'], ('python', 'learn'))
>>> for every in t:
...     print every
...
1
23
[123, 'abc']
('python', 'learn')
>>> len(t)
4

>>> t[2][0]      #还能这样呀，哦对了，list中也能这样
123
>>> t[3][1]
'learn'
```

所有在**list**中可以修改**list**的方法，在**tuple**中，都失效。

分别用**list()**和**tuple()**能够实现两者的转化：

```

>>> t
(1, '23', [123, 'abc'], ('python', 'learn'))
>>> tls = list(t) #tuple-->list
>>> tls
[1, '23', [123, 'abc'], ('python', 'learn')]

>>> t_tuple = tuple(tls) #list-->tuple
>>> t_tuple
(1, '23', [123, 'abc'], ('python', 'learn'))

```

tuple用在哪里？

既然它是list和str的杂合，它有什么用途呢？不是用list和str都可以了吗？

在很多时候，的确是用list和str都可以了。但是，看官不要忘记，我们用计算机语言解决的问题不都是简单问题，就如同我们的自然语言一样，虽然有的词汇看似可有可无，用别的也能替换之，但是我们依然需要在某些情况下使用它们。

一般认为，tuple有这类特点，并且也是它使用的情景：

- Tuple 比 list 操作速度快。如果您定义了一个值的常量集，并且唯一要用它做的是不断地遍历它，请使用 tuple 代替 list。
- 如果对不需要修改的数据进行“写保护”，可以使代码更安全。使用 tuple 而不是 list 如同拥有一个隐含的 assert 语句，说明这一数据是常量。如果必须要改变这些值，则需要执行 tuple 到 list 的转换（需要使用一个特殊的函数）。
- Tuples 可以在 dictionary 中被用做 key，但是 list 不行。实际上，事情要比这更复杂。Dictionary key 必须是不可变的。Tuple 本身是不可改变的，但是如果您有一个 list 的 tuple，那就认为是可变的了，用做 dictionary key 就是不安全的。只有字符串、整数或其它对 dictionary 安全的 tuple 才可以用作 dictionary key。
- Tuples 可以用在字符串格式化中，后面会用到。

一二三,集合了

回顾一下已经了解的数据类型:int/str/bool/list/dict/tuple

还真不少了.

不过,python是一个发展的语言,没准以后还出别的呢.看官可能有疑问了,出了这么多的数据类型,我也记不住呀,特别是里面还有不少方法.

不要担心记不住,你只要记住爱因斯坦说的就好了.

爱因斯坦在美国演讲,有人问:“你可记得声音的速度是多少?你如何记下许多东西?”

爱因斯坦轻松答道:“声音的速度是多少,我必须查辞典才能回答.因为我从来不记在辞典上已经印着的东西,我的记忆力是用来记忆书本上没有的东西。”

多么霸气的回答,这回答不仅仅霸气,更是在告诉我们一种方法:只要能够通过某种方法查找到的,就不需要记忆.

那么,上面那么多数据类型的各种方法,都不需要记忆了,因为它们都可以通过下述方法但不限于这些方法查到(这句话的逻辑还是比较严密的,包括但不限于...)

- 交互模式下用dir()或者help()
- google(不推荐Xdu,原因自己体会啦)

为了能够在总体上对已经学习过的数据类型有了解,我们不妨做如下分类:

1. 是否为序列类型:即该数据的元素是否能够索引.其中序列类型的包括str/list/tuple
2. 是否可以原处修改:即该数据的元素是否能够原处修改(特别提醒看官,这里说的是原处修改问题,有的资料里面说str不能修改,也是指原处修改问题.为了避免误解,特别强调了原处).能够原处修改的是list/dict(特别说明,dict的键必须是不可修改的,dict的值可原处修改)

什么原处修改?看官能不能在交互模式下通过实例解释一下?

到这里,看官可千万不要以为本讲是复习课.本讲的主要内容不是复习,主要内容是要向看官介绍一种新的数据类型:集合(set).彻底晕倒了,到底python有多少个数据类型呢?又多出来了一个.

从基本道理上说,python中的数据类型可以很多,因为每个人都可以自己定义一种数据类型.但是,python官方认可或者说内置的数据类型,就那么几种了.基本上今天的set讲完,就差不多了.在以后的开发过程中,包括今天和以往介绍的数据类型,是常用的.当然,自己定义一个也可以,但是用原生的更好.

创建set

tuple算是list和str的杂合(杂交的都有自己的优势,上一节的末后已经显示了),那么set则可以堪称是list和dict的杂合.

set拥有类似dict的特点:可以用{}花括号来定义;其中的元素没有序列,也就是是非序列类型的数据;而且,set中的元素不可重复,这就类似dict的键.

set也有继承了一点list的特点:如可以原处修改(事实上是一种类别的set可以原处修改,另外一种不可以).

下面通过实验,进一步理解创建set的方法:

```
>>> s1 = set("qiwsir") #把str中的字符拆解开,形成set. 特别注意观察:qiwsir中有两个i
>>> s1
set(['q', 'i', 's', 'r', 'w'])

>>> s2 = set([123, "google", "face", "book", "facebook", "book"]) #通过list创建set. 不能有重复,
>>> s2
set(['facebook', 123, 'google', 'book', 'face']) #元素顺序排列不是按照指定顺序

>>> s3 = {"facebook", 123} #通过{}直接创建
>>> s3
set([123, 'facebook'])
```

再大胆做几个探究,请看官注意观察结果:

```
>>> s3 = {"facebook", [1, 2, 'a'], {"name": "python", "lang": "english"}, 123}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'

>>> s3 = {"facebook", [1, 2], 123}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

从上述实验中,可以看出,通过{}无法创建含有list/dict元素的set.

继续探索一个情况:

```

>>> s1
set(['q', 'i', 's', 'r', 'w'])
>>> s1[1] = "I"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support item assignment

>>> s1
set(['q', 'i', 's', 'r', 'w'])
>>> lst = list(s1)
>>> lst
['q', 'i', 's', 'r', 'w']
>>> lst[1] = "I"
>>> lst
['q', 'I', 's', 'r', 'w']

```

上面的探索中,将set和list做了一个对比,虽然说两者都能够做原处修改,但是,通过索引编号(偏移量)的方式,直接修改,list允许,但是set报错.

那么,set如何修改呢?

更改set

还是用前面已经介绍过多次的自学方法,把set的有关内置函数找出来,看看都可以对set做什么操作.

```

>>> dir(set)
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__', '__eq__', '__

```

为了看的清楚,我把双划线__开始的先删除掉(后面我们会有专题讲述这些):

```

'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection',
'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference', 'symmetric_difference_update', 'union', 'update'

```

然后用help()可以找到每个函数的具体使用方法,下面列几个例子:

增加元素

```
>>> help(set.add)

Help on method_descriptor:

add(...)

Add an element to a set.

This has no effect if the element is already present.
```

下面在交互模式这个最好的实验室里面做实验：

```
>>> a_set = {}           #我想当然地认为这样也可以建立一个set
>>> a_set.add("qiwsir")  #报错.看看错误信息,居然告诉我dict没有add.我分明建立的是set呀.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'add'
>>> type(a_set)        #type之后发现,计算机认为我建立的是一个dict
<type 'dict'>
```

特别说明一下,{}这个东西,在dict和set中都用.但是,如上面的方法建立的是dict,不是set.这是python规定的.要建立set,只能用前面介绍的方法了.

```
>>> a_set = {'a', 'i'}      #这回就是set了吧
>>> type(a_set)
<type 'set'>           #果然

>>> a_set.add("qiwsir")   #增加一个元素
>>> a_set                 #原处修改,即原来的a_set引用对象已经改变
set(['i', 'a', 'qiwsir'])

>>> b_set = set("python")
>>> type(b_set)
<type 'set'>
>>> b_set
set(['h', 'o', 'n', 'p', 't', 'y'])
>>> b_set.add("qiwsir")
>>> b_set
set(['h', 'o', 'n', 'p', 't', 'qiwsir', 'y'])

>>> b_set.add([1,2,3])    #这样做是不行滴,跟前面一样,报错.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'

>>> b_set.add('[1,2,3]')  #可以这样!
>>> b_set
set(['[1,2,3]', 'h', 'o', 'n', 'p', 't', 'qiwsir', 'y'])
```

除了上面的增加元素方法之外,还能够从另外一个set中合并过来元素,方法是set.update(s2)

```

>>> help(set.update)
update(...)
    Update a set with the union of itself and others.

>>> s1
set(['a', 'b'])
>>> s2
set(['github', 'qiwsir'])
>>> s1.update(s2)      #把s2的元素并入到s1中.
>>> s1                  #s1的引用对象修改
set(['a', 'qiwsir', 'b', 'github'])
>>> s2                  #s2的未变
set(['github', 'qiwsir'])

```

删除

```

>>> help(set.pop)
pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.

>>> b_set
set(['[1,2,3]', 'h', 'o', 'n', 'p', 't', 'qiwsir', 'y'])
>>> b_set.pop()      #从set中任意选一个删除,并返回该值
'[1,2,3]'
>>> b_set.pop()
'h'
>>> b_set.pop()
'o'
>>> b_set
set(['n', 'p', 't', 'qiwsir', 'y'])

>>> b_set.pop("n")  #如果要指定删除某个元素,报错了.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pop() takes no arguments (1 given)

```

set.pop()是从set中任意选一个元素,删除并将这个值返回.但是,不能指定删除某个元素.报错信息中就告诉我们了,pop()不能有参数.此外,如果set是空的了,也报错.这条是帮助信息告诉我们的,看官可以试试.

要删除指定的元素,怎么办?

```
>>> help(set.remove)

remove(...)
    Remove an element from a set; it must be a member.

    If the element is not a member, raise a KeyError.
```

set.remove(obj)中的obj,必须是set中的元素,否则就报错.试一试:

```
>>> a_set
set(['i', 'a', 'qiwsir'])
>>> a_set.remove("i")
>>> a_set
set(['a', 'qiwsir'])
>>> a_set.remove("w")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'w'
```

跟remove(obj)类似的一个还有discard(obj):

```
>>> help(set.discard)

discard(...)
    Remove an element from a set if it is a member.

    If the element is not a member, do nothing.
```

与help(set.remove)的信息对比,看看有什么不同.discard(obj)中的obj如果是set中的元素,就删除,如果不是,就什么也不做,do nothing.新闻就要对比着看才有意思呢.这里也一样.

```
>>> a_set.discard('a')
>>> a_set
set(['qiwsir'])
>>> a_set.discard('b')
>>>
```

在删除上还有一个绝杀,就是set.clear(),它的功能是:Remove all elements from this set.(看官自己在交互模式下help(set.clear))

```

>>> a_set
set(['qiwsir'])
>>> a_set.clear()
>>> a_set
set([])
>>> bool(a_set)      #空了, bool一下返回False.
False

```

知识

集合,也是一个数学概念(以下定义来自维基百科)

集合 (或简称集) 是基本的数学概念, 它是集合论的研究对象。最简单的说法, 即是在最原始的集合论—朴素集合论—中的定义, 集合就是“一堆东西”。集合里的“东西”, 叫作元素。若然 x 是集合 A 的元素, 记作 $x \in A$ 。

集合是现代数学中一个重要的基本概念。集合论的基本理论直到十九世纪末才被创立, 现在已经是数学教育中一个普遍存在的部分, 在小学时就开始学习了。这里对被数学家们称为“直观的”或“朴素的”集合论进行一个简短而基本的介绍; 更详细的分析可见朴素集合论。对集合进行严格的公理推导可见公理化集合论。

在计算机中,集合是什么呢?同样来自[维基百科](#),这么说的:

在计算机科学中,集合是一组可变数量的数据项 (也可能是0个) 的组合, 这些数据项可能共享某些特征, 需要以某种操作方式一起进行操作。一般来讲, 这些数据项的类型是相同的, 或基类相同 (若使用的语言支持继承)。列表 (或数组) 通常不被认为是集合, 因为其大小固定, 但事实上它常常在实现中作为某些形式的集合使用。

集合的种类包括列表, 集, 多重集, 树和图。枚举类型可以是列表或集。

不管是否明白,貌似很厉害呀.

是的,所以本讲仅仅是对集合有一个入门.关于集合的更多操作如运算/比较等,还没有涉及呢.

集合的关系

冻结的集合

前面一节讲述了集合的基本概念，注意，那里所涉及到的集合都是可原处修改的集合。还有一种集合，不能在原处修改。这种集合的创建方法是：

```
>>> f_set = frozenset("qiwsir")      #看这个名字就知道了frozen，冻结的set
>>> f_set
frozenset(['q', 'i', 's', 'r', 'w'])
>>> f_set.add("python")           #报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'

>>> a_set = set("github")        #对比看一看，这是一个可以原处修改的set
>>> a_set
set(['b', 'g', 'i', 'h', 'u', 't'])
>>> a_set.add("python")
>>> a_set
set(['b', 'g', 'i', 'h', 'python', 'u', 't'])
```

集合运算

先复习一下中学数学（准确说是高中数学中的一点知识）中关于集合的一点知识，主要是唤起那痛苦而青涩美丽的回忆吧，至少对我是。

元素与集合的关系

元素是否属于某个集合。

```
>>> aset
set(['h', 'o', 'n', 'p', 't', 'y'])
>>> "a" in aset
False
>>> "h" in aset
True
```

集合与集合的纠结

假设两个集合A、B

- A是否等于B，即两个集合的元素完全一样

在交互模式下实验

```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a == b
False
>>> a != b
True
```

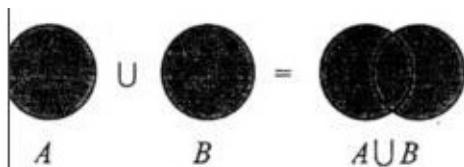
- A是否是B的子集，或者反过来，B是否是A的超集。即A的元素也都是B的元素，但是B的元素比A的元素数量多。

实验一下

```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> c
set(['q', 'i'])
>>> c < a      #c是a的子集
True
>>> c.issubset(a)    #或者用这种方法，判断c是否是a的子集
True
>>> a.issuperset(c) #判断a是否是c的超集
True

>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a < b      #a不是b的子集
False
>>> a.issubset(b)    #或者这样做
False
```

- A、B的并集，即A、B所有元素，如下图所示

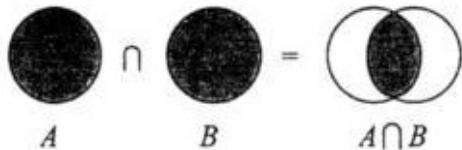


```

>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a | b          #可以有两种方式，结果一样
set(['a', 'i', 'l', 'o', 'q', 's', 'r', 'w'])
>>> a.union(b)
set(['a', 'i', 'l', 'o', 'q', 's', 'r', 'w'])

```

- A、B的交集，即A、B所公有的元素，如下图所示



```

>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a & b      #两种方式，等价
set(['q', 'i'])
>>> a.intersection(b)
set(['q', 'i'])

```

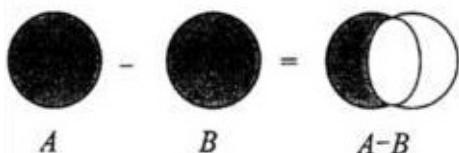
我在实验的时候，顺手敲了下面的代码，出现的结果如下，看官能解释一下吗？（思考题）

```

>>> a and b
set(['a', 'q', 'i', 'l', 'o'])

```

- A相对B的差（补），即A相对B不同的部分元素，如下图所示

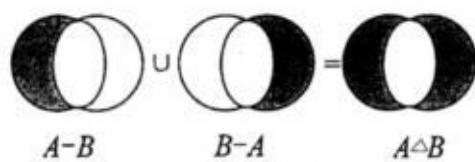


```

>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a - b
set(['s', 'r', 'w'])
>>> a.difference(b)
set(['s', 'r', 'w'])

```

-A、B的对称差集，如下图所示



```
>>> a
set(['q', 'i', 's', 'r', 'w'])
>>> b
set(['a', 'q', 'i', 'l', 'o'])
>>> a.symmetric_difference(b)
set(['a', 'l', 'o', 's', 'r', 'w'])
```

以上是集合的基本运算。在编程中，如果用到，可以用前面说的方法查找。不用死记硬背。

Python的数据类型总结

前面已经洋洋洒洒地介绍了不少数据类型。不能再不顾一切地向前冲了，应当总结一下。这样让看官能够从总体上对这些数据类型有所了解，如果能够有一览众山小的感觉，就太好了。

下面的表格中列出了已经学习过的数据类型，也是python的核心数据类型之一部分，这些都被称之为内置对象。

对象，就是你面对的所有东西都是对象，看官要逐渐熟悉这个称呼。所有的数据类型，就是一种对象。英文单词是object，直接的汉语意思是物体，这就好像我们在现实中一样，把很多我们看到和用到的都可以统称为“东西”一样。“东西”就是“对象”，就是object。在编程中，那个所谓面向对象，也可以说成“面向东西”，是吗？容易有歧义吧。

对象类型	举例
int/float	123, 3.14
str	'qiwsir.github.io'
list	[1, [2, 'three'], 4]
dict	{'name': "qiwsir", "lang": "python"}
tuple	(1, 2, "three")
set	set("qi"), {"q", "i"}

不论任何类型的数据，只要动用dir(object)或者help(obj)就能够在交互模式下查看到有关的函数，也就是这样能够查看相关帮助文档了。举例：

```
>>> dir(dict)
```

看官需要移动鼠标，就能够看全(下面的本质上就是一个list)：

```
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__',
]
```

先略过__双下划线开头的那些，看后面的，就是dict的内置函数。至于详细的操作方法，通过类似help(dict.pop)的方式获得。这是前面说过的，再说一遍，加深印象。

我的观点：学习，重要的是学习方法，不是按部就班的敲代码。

今天既然是复习，就要在原来基础上提高一点。所以，也要看看上面那些以双下划线`__`开头的东西，请看官找一下，有没有发现这个：`"__doc__"`。这是什么，它是一个文件，里面记录了对当前所查看的对象的详细解释。可以在交互模式下这样查看：

```
>>> dict.__doc__
```

显示应该是这样的：

```
"dict() -> new empty dictionary\ndict(mapping) -> new dictionary initialized from a
mapping object's\n (key, value) pairs\ndict(iterable) -> new dictionary initialized as if
via:\n d = {}\n for k, v in iterable:\n     d[k] = v\ndict(**kwargs) -> new dictionary initialized
with the name=value pairs\n in the keyword argument list. For example: dict(one=1,
two=2)"
```

注意看上面乱七八糟的英文中，是不是有`\n`符号，这是什么？前面在讲述字符串的时候提到了转义符号`\`，这是换一行。也就是说，如果上面的文字，按照排版要求，应该是这样的（当然，在文本中，如果打开，其实就是排好版的样子）。

```
"dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a
mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d
= {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the
name=value pairs in the keyword argument list. For example: dict(one=1, two=2)"
```

可能排版还是不符合愿意。不过，看官也大概能看明白了。我要说的不是排版，要说的是告诉看官一种查看某个数据类型含义的方法，就是通过`obj.__doc__`文件来看。

嘿嘿，其实有一种方法，可以看到排版的结果的：

```
>>> print dict.__doc__
dict() -> new empty dictionary
dict(mapping) -> new dictionary initialized from a mapping object's
 (key, value) pairs
dict(iterable) -> new dictionary initialized as if via:
 d = {}
 for k, v in iterable:
     d[k] = v
dict(**kwargs) -> new dictionary initialized with the name=value pairs
 in the keyword argument list. For example: dict(one=1, two=2)
```

上面那么折腾一下，就是为了凑篇幅，不然这个总结的东西太少了。

总之，只要用这种方法，你就能得到所有帮助文档，随时随地。如果可以上网，到官方网站，是另外一种方法。

还需要再解释别的吗？都多余了。唯一需要的是看官要能会点英语。不过我相信看官能够读懂，我这个二把刀都不如的英语水平，还能凑合看呢，何况看官呢？

总结不是意味着结束，是意味着继往开来。精彩还在后面，这里只是休息。今天还是周日。

主日崇拜

腓立比書 Philippians (3:13-14)

Brethren, I count not myself to have apprehended: but this one thing I do, forgetting those things which are behind, and reaching forth unto those things which are before, I press toward the mark for the prize of the high calling of God in Christ Jesus.

弟兄們、我不是以爲自己已經得著了。我只有一件事、就是忘記背後努力面前的，向著標竿直跑、要得神在基督耶穌裡從上面召我來得的獎賞。

忘记背后，努力面前，向着标杆直跑

深入变量和引用对象

今天是2014年8月4日，这段时间灾祸接连发生，显示不久前昆山的工厂爆炸，死伤不少，然后是云南地震，也有死伤。为所有在灾难中受伤害的人们献上祷告。

在《永远强大的函数》那一讲中，老齐我（<http://qiwsir.github.io>）已经向看官们简述了一下变量，之后我们就一直在使用变量，每次使用变量，都要有一个操作，就是赋值。本讲再次提及这个两个事情，就是要让看官对变量和赋值有一个知其然和知其所以然的认识。当然，最后能不能达到此目的，主要看我是不是说的通俗易懂了。如果您没有明白，就说明我说的还不够好，可以联系我，我再为您效劳。

变量和对象

在《learning python》那本书里面，作者对变量、对象和引用的关系阐述的非常明了。我这里在很大程度上是受他的启发。感谢作者Mark Lutz先生的巨著。

应用《learning python》中的一个观点：变量无类型，对象有类型

在python中，如果要使用一个变量，不需要提前声明，只需要在用的时候，给这个变量赋值即可。这里特别强调，只要用一个变量，就要给这个变量赋值。

所以，像这样是不行的。

```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

反复提醒：一定要注意看报错信息。如果光光地写一个变量，而没有赋值，那么python认为这个变量没有定义。赋值，不仅仅是给一个非空的值，也可以给一个空值，如下，都是允许的

```
>>> x = 3
>>> lst = []
>>> word = ""
>>> my_dict = {}
```

在前面讲述中，我提出了一个类比，就是变量通过一根线，连着对象（具体就可能是一个int/list等），这个类比被很多人接受了，算是我老齐的首创呀。那么，如果要用一种严格的语言来描述，变量可以理解为一个系统表的元素，它拥有过指向对象的命名空间。太严肃了，

不好理解，就理解我那个类比吧。变量就是存在系统中的一个东西，这个东西有一种能力，能够用一根线与某对象连接，它能够钓鱼。

对象呢？展开想象。在机器的内存中，系统分配一个空间，这里面就放着所谓的对象，有时候放数字，有时候放字符串。如果放数字，就是int类型，如果放字符串，就是str类型。

接下来的事情，就是前面说的变量用自己所拥有的能力，把对象和自己连接起来（指针连接对象空间），这就是引用。引用完成，就实现了赋值。



看到上面的图了吧，从图中就比较鲜明的表示了变量和对象的关系。所以，严格地讲，只有放在内存空间中的对象（也就是数据）才有类型，而变量是没有类型的。这么说如果还没有彻底明白，就再打一个比喻：变量就好比钓鱼的人，湖水里就好像内存，里面有好多鱼，有各种各样的鱼，它们就是对象。钓鱼的人（变量）的任务就是用某种方式（鱼儿引诱）把自己和鱼通过鱼线连接起来。那么，鱼是有类型的，有鲤鱼、鲫鱼、带鱼（带鱼也跑到湖水了，难道是淡水带鱼？呵呵，就这么扯淡吧，别较真），钓鱼的人（变量）没有这种类型，他钓到不同类型的鱼。

这个比喻太烂了。凑合着理解吧。看官有好的比喻，别忘记分享。

同一个变量可以同时指向两个对象吗？绝对不能脚踩两只船。如果这样呢？

```
>>> x = 4
>>> x = 5
>>> x
5
```

变量x先指向了对象4，然后指向对象5，当后者放生的时候，自动跟第一个对象4解除关系。再看x，引用的对象就是5了。那么4呢？一旦没有变量引用它了，它就变成了孤魂野鬼。python是很吝啬的，它绝对不允许在内存中存在孤魂野鬼。凡是这些东西都被看做垃圾，而对垃圾，python有一个自动的收回机制。

在网上找了一个图示说明，很好，引用过来（来源：<http://www.linuxidc.com/Linux/2012-09/69523.htm>）

```
>>> a = 100          #完成了变量a对内存空间中的对象100的引用
```

如下图所示：



然后，又操作了：

```
>>> a = "hello"
```

如下图所示：



原来内存中的那个100就做为垃圾被收集了。而且，这个收集过程是python自动完成的，不用我们操心。

那么，python是怎么进行垃圾收集的呢？在[Quora](#)上也有人问这个问题，我看那个回答很精彩，做个链接，有兴趣的读一读吧。[Python \(programming language\): How does garbage collection in Python work?](#)

is和==的效果

以上过程的原理搞清楚了，下面就可以深入一步了。

```
>>> l1 = [1, 2, 3]
>>> l2 = l1
```

这个操作中，l1和l2两个变量，引用的是一个对象，都是[1,2,3]。何以见得？如果通过l1来修改[1,2,3]，l2引用对象也修改了，那么就证实这个观点了。

```
>>> l1[0] = 99      #把对象变为[99, 2, 3]
>>> l1
[99, 2, 3]
>>> l2      #真的变了
[99, 2, 3]
```

再换一个方式：

```
>>> l1 = [1, 2, 3]
>>> l2 = [1, 2, 3]
>>> l1[0] = 99
>>> l1
[99, 2, 3]
>>> l2
[1, 2, 3]
```

l1和l2貌似指向了同样的一个对象[1,2,3]，其实，在内存中，这是两块东西，互不相关。只是在内容上一样。就好像是水里长的一样的两条鱼，两个人都钓到了，但不是同一条。所以，当通过l1修改引用对象的后，l2没有变化。

进一步还能这么检验：

```

>>> l1
[1, 2, 3]
>>> l2
[1, 2, 3]
>>> l1 == l2      #两个相等，是指内容一样
True
>>> l1 is l2      #is 是比较两个引用对象在内存中的地址是不是一样
False             #前面的检验已经说明，这是两个东东

>>> l3 = l1      #顺便看看如果这样，l3和l1应用同一个对象
>>> l3
[1, 2, 3]
>>> l3 == l1
True
>>> l3 is l1      #is的结果是True
True

```

某些对象，有copy函数，通过这个函数得到的对象，是一个新的还是引用到同一个对象呢？看官也可以做一下类似上面的实验，就晓得了。比如：

```

>>> l1
[1, 2, 3]
>>> l2 = l1[:]
>>> l2
[1, 2, 3]
>>> l1[0] = 22
>>> l1
[22, 2, 3]
>>> l2
[1, 2, 3]

>>> adict = {"name": "qiwsir", "web": "qiwsir.github.io"}
>>> bdict = adict.copy()
>>> bdict
{'web': 'qiwsir.github.io', 'name': 'qiwsir'}
>>> adict["email"] = "qiwsir@gmail.com"
>>> adict
{'web': 'qiwsir.github.io', 'name': 'qiwsir', 'email': 'qiwsir@gmail.com'}
>>> bdict
{'web': 'qiwsir.github.io', 'name': 'qiwsir'}

```

不过，看官还有小心有点，python不总按照前面说的方式出牌，比如小数字的时候

```
>>> x = 2
>>> y = 2
>>> x is y
True
>>> x = 200000
>>> y = 200000
>>> x is y      #什么道理呀，小数字的时候，就用缓存中的。
False

>>> x = 'hello'
>>> y = 'hello'
>>> x is y
True
>>> x = "what is you name?"
>>> y = "what is you name?"
>>> x is y      #不光小的数字，短的字符串也是
False
```

赋值是不是简单地就是等号呢？从上面得出来，=的作用就是让变量指针指向某个对象。不过，还可以再深入一些。走着瞧吧。

赋值，简单也不简单

变量命名

在《初识永远强大的函数》一文中，有一节专门讨论“取名字的学问”，就是有关变量名称的问题，本温故而知新的原则，这里要复习：

名称格式：（下划线或者字母）+（任意数目的字母，数字或下划线）

注意：

1. 区分大小写
2. 禁止使用保留字
3. 遵守通常习惯
4. 以单一下划线开头的变量名（_X）不会被from module import *语句导入的。
5. 前后有下划线的变量名（_X）是系统定义的变量名，对解释器有特殊意义。
6. 以两个下划线开头，但结尾没有两个下划线的变量名（__X）是类本地（压缩）变量。
7. 通过交互模式运行时，只有单个下划线变量(_)会保存最后的表达式结果。

需要解释一下保留字，就是python里面保留了一些单词，这些单词不能让用户来用作变量名称。都有哪些呢？(python2和python3少有差别，但是总体差不多)

and assert break class continue def del elif else except exec finally for from global if import in is lambda not or pass print raise return try while yield

需要都记住吗？当然不需要了。一方面，可以在网上随手查到，另外，还能这样：

```
>>> not = 3
File "<stdin>", line 1
    not = 3
    ^
SyntaxError: invalid syntax

>>> pass = "hello,world"
File "<stdin>", line 1
    pass = "hello,world"
    ^
SyntaxError: invalid syntax
```

在交互模式的实验室中，用保留字做变量，就报错了。当然，这时候就要换名字了。

以上原则，是基本原则。在实际编程中，大家通常还这样做，以便让程序更具有可读性：

- 名字具有一定的含义。比如写：`n = "qiwsir"`，就不如写：`name = "qiwsir"`更好。
- 名字不要误导别人。比如用`account_list`指一组账号，就会被人误解为是`list`类型的数据，事实上可能是也可能不是。所以这时候最好换个名称，比如直接用`accounts`。
- 名字要有意义的区分，有时候你可能会用到`a1, a2`之类的名字，最好不要这么做，换个别的方式，通过字面能够看出一定的区分来更好。
- 最好是名称能够读出来，千万别自己造英文单词，也别乱用所写什么的，特别是贵国的，还喜欢用汉语拼音缩写来做为名字，更麻烦了，还不如全拼呢。最好是用完整的单词或者公认的不会引起歧义的缩写。
- 单个字母和数字就少用了，不仅是显得你太懒惰，还会因为在一段代码中可能有很多个单个的字母和数字，为搜索带来麻烦，别人也更不知道你的`i`和他理解的`i`是不是一个含义。

总之，取名字，讲究不少。不论如何，要记住一个标准：明确

赋值语句

对于赋值语句，看官已经不陌生了。任何一个变量，在python中，只要想用它，就要首先赋值。

语句格式：变量名称 = 对象

[上一节](#)中也分析了赋值的本质。

还有一种赋值方式，叫做隐式赋值，通过`import`、`from`、`del`、`class`、`for`、函数参数。等模块导入，函数和类的定义，`for`循环变量以及函数参数都是隐式赋值运算。这方面的东西后面会徐徐道来。

```
>>> name = "qiwsir"

>>> name, website = "qiwsir", "qiwsir.github.io"      #多个变量，按照顺序依次赋值
>>> name
'qiwsir'
>>> website
'qiwsir.github.io'

>>> name, website = "qiwsir"      #有几个变量，就对应几个对象，不能少，也不能多
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
```

如果这样赋值，也得两边数目一致：

```
>>> one,two,three,four = "good"
>>> one
'g'
>>> two
'o'
>>> three
'o'
>>> four
'd'
```

这就相当于把good分拆为一个一个的字母，然后对应着赋值给左边的变量。

```
>>> [name,site] = ["qiwsir","qiwsir.github.io"]
>>> name
'qiwsir'
>>> site
'qiwsir.github.io'
>>> name,site = ("qiwsir","qiwsir.github.io")
>>> name
'qiwsir'
>>> site
'qiwsir.github.io'
```

这样也行呀。

其实，赋值的样式不少，核心就是将变量和某对象对应起来。对象，可以用上面的方式，也许是这样的

```
>>> site = "qiwsir.github.io"
>>> name, main = site.split(".")[0], site.split(".")[1]  #还记得str.split(<sep>)这个东东吗
>>> name
'qiwsir'
>>> main
'github'
```

增强赋值

这个东西听名字就是比赋值强的。

在python中，将下列的方式称为增强赋值：

增强赋值语句	等价于语句
$x+=y$	$x = x+y$
$x-=y$	$x = x-y$
$x^*=y$	$x = x^*y$
$x//=y$	$x = x//y$

其它类似结构： $x&=y$ $x|=y$ $x^{\wedge}=y$ $x\% =y$ $x>>=y$ $x<<=y$ $x**=y$ $x//=y$

看下面的例子，有一个list，想得到另外一个列表，其中每个数比原来list中的大2。可以用下面方式实现：

```
>>> number
[1, 2, 3, 4, 5]
>>> number2 = []
>>> for i in number:
...     i = i+2
...     number2.append(i)
...
>>> number2
[3, 4, 5, 6, 7]
```

如果用上面的增强赋值， $i = i+2$ 可以写成 $i += 2$ ，试一试吧：

```
>>> number
[1, 2, 3, 4, 5]
>>> number2 = []
>>> for i in number:
...     i +=2
...     number2.append(i)
...
>>> number2
[3, 4, 5, 6, 7]
```

这就是增强赋值。为什么用增强赋值？因为 $i += 2$ ，比 $i = i+2$ 计算更快，后者右边还要拷贝一个 i 。

上面的例子还能修改，别忘记了list解析的强大功能呀。

```
>>> [i+2 for i in number]
[3, 4, 5, 6, 7]
```

坑爹的字符编码

字符编码，在编程中，是一个让学习者比较郁闷的东西，比如一个str，如果都是英文，好说多了。但恰恰不是如此，中文是我们不得不的。所以，哪怕是初学者，都要了解并能够解决字符编码问题。

```
>>> name = '老齐'
>>> name
'\xe8\x80\x81\xe9\xbd\x90'
```

在你的编程中，你遇到过上面的情形吗？认识最下面一行打印出来的东西吗？看人家英文，就好多了

```
>>> name = "qiwsir"
>>> name
'qiwsir'
```

难道这是中文的错吗？看来投胎真的是一个技术活。是的，投胎是技术活，但上面的问题不是中文的错。

编码

什么是编码？这是一个比较玄乎的问题。也不好下一个普通定义。我看到有的教材中有定义，不敢说他的定义不对，至少可以说不容易理解。

古代打仗，击鼓进攻、鸣金收兵，这就是编码。吧要传达给士兵的命令对应为一定的其它形式，比如命令“进攻”，经过如此的信息传递：



1. 长官下达进攻命令，传令员将这个命令编码为鼓声（如果复杂点，是不是有几声鼓响，如何进攻呢？）。
2. 鼓声在空气中传播，比传令员的嗓子吼出来的声音传播的更远，士兵听到后也不会引起歧义，一般不会有士兵把鼓声当做打呼噜的声音。这就是“进攻”命令被编码成鼓声之后的优势所在。
3. 士兵听到鼓声，就是接收到信息之后，如果接受过训练或者有人告诉过他们，他们就知道这是让我进攻。这个过程就是解码。所以，编码方案要有两套。一套在信息发出者那里，另外一套在信息接受者这里。经过解码之后，士兵明白了，才行动。

以上过程比较简单。其实，真实的编码和解码过程，要复杂了。不过，原理都差不多的。

举一个似乎遥远，其实不久前人们都在使用的东西做例子：[电报](#)

电报是通信业务的一种，在19世纪初发明，是最早使用电进行通信的方法。电报大为加快了消息的流通，是工业社会的其中一项重要发明。早期的电报只能在陆地上通讯，后来使用了海底电缆，开展了越洋服务。到了20世纪初，开始使用无线电拨发电报，电报业务基本上已能抵达地球上大部份地区。电报主要是用作传递文字讯息，使用电报技术用作传送图片称为传真。

中国首条出现电报线路是1871年，由英国、俄国及丹麦敷设，从香港经上海至日本长崎的海底电缆。由于清政府的反对，电缆被禁止在上海登陆。后来丹麦公司不理清政府的禁令，将线路引至上海公共租界，并在6月3日起开始收发电报。至于首条自主敷设的线路，是由福建巡抚丁日昌在台湾所建，1877年10月完工，连接台南及高雄。1879年，北洋大臣李鸿章在天津、大沽及北塘之间架设电报线路，用作军事通讯。1880年，李鸿章奏准开办电报总局，由盛宣怀任总办。并在1881年12月开通天津至上海的电报服务。李鸿章說：“五年来，我国创设沿江沿海各省电线，总计一万多里，国家所费无多，巨款来自民间。当时正值法人挑衅，将帅报告军情，朝廷传达指示，均相机而动，无丝毫阻碍。中国自古用兵，从未如此神速。出使大臣往来问答，朝发夕至，相隔万里好似同居庭院。举设电报一举三得，既防止外敌侵略，又加强国防，亦有利于商务。”天津官电局于庚子遭乱全毁。1887年，台湾巡抚刘铭传敷设了福州至台湾的海底电缆，是中国首条海底电缆。1884年，北京电报开始建设，采用“安设双线，由通州展至京城，以一端引入署中，专递官信，以一端择地安置用便商民”，同年8月5日，电报线路开始建设，所有电线杆一律漆成红色。8月22日，位于北京崇文门外大街西的喜鹊胡同的外城商用电报局开业。同年8月30日，位于崇文门内泡子和以西的吕公堂开局，专门收发官方电报。

为了传达汉字，电报部门准备由4位数字或3位罗马字构成的代码，即中文电码，采用发送前将汉字改写成电码发出，收电报后再将电码改写成汉字的方法。

列位看官注意了，这里出现了电报中用的“[中文电码](#)”，这就是一种编码，将汉字对应成阿拉伯数字，从而能够用电报发送汉字。

1873年，法国驻华人员威基杰参照《康熙字典》的部首排列方法，挑选了常用汉字6800多个，编成了第一部汉字电码本《电报新书》。

电报中的编码被称为摩尔斯电码，英文是Morse Code

摩尔斯电码（英语：Morse Code）是一种时通时断的信号代码，通过不同的排列顺序来表达不同的英文字母、数字和标点符号。是由美国人萨缪尔·摩尔斯在1836年发明。

摩尔斯电码是一种早期的数字化通信形式，但是它不同于现代只使用0和1两种状态的二进制代码，它的代码包括五种：点（.） 、划（-） 、每个字符间短的停顿（在点和划之间的停顿） 、每个词之间中等的停顿、以及句子之间长的停顿

看来电报员是一个技术活，不同长短的停顿都代表了不同意思。哦，对了，有一个老片子《永不消逝的电波》，看完之后保证你才知道，里面根本就没有讲电报是怎么编码的。

摩尔斯电码在海事通讯中被作为国际标准一直使用到1999年。1997年，当法国海军停止使用摩尔斯电码时，发送的最后一条消息是：“所有人注意，这是我们在永远沉寂之前最后的一声呐喊！”

我瞪着眼看了老长时间，这两行不是一样的吗？

不管这个了，总之，这就是编码。

计算机中的字符编码

先抄一段维基百科对字符编码的解释：

字符编码（英语：Character encoding）、字集码是把字符集中的字符编码为指定集合中某一对象（例如：比特模式、自然数串行、8位组或者电脉冲），以便文本在计算机中存储和通过通信网络的传递。常见的例子包括将拉丁字母表编码成摩斯电码和ASCII。其中，ASCII将字母、数字和其它符号编号，并用7比特的二进制来表示这个整数。通常会额外使用一个扩充的比特，以便于以1个字节的方式存储。

在计算机技术发展的早期，如ASCII（1963年）和EBCDIC（1964年）这样的字符集逐渐成为标准。但这些字符集的局限很快就变得明显，于是人们开发了许多方法来扩展它们。对于支持包括东亚CJK字符家族在内的写作系统的要求能支持更大量的字符，并且需要一种系统而不是临时的方法实现这些字符的编码。

在这个世界上，有好多不同的字符编码。但是，它们不是自己随便搞搞的。而是要有一定的基础，往往是以名叫ASCII的编码为基础，这里边也应该包括北朝鲜吧（不知道他们用什么字符编码，瞎想的，别当真，不代表本教材立场，只代表瞎想）。

ASCII (pronunciation: 英语发音: /'æski/ ASS-kee[1], American Standard Code for Information Interchange, 美国信息交换标准代码) 是基于拉丁字母的一套电脑编码系统。它主要用于显示现代英语, 而其扩展版本EASCII则可以部分支持其他西欧语言, 并等同于国际标准ISO/IEC 646。由于万维网使得ASCII广为通用, 直到2007年12月, 逐渐被Unicode取代。

上面的引文中已经说了，现在我们用的编码标准，已经不是ASCII了，我上大学那时候老师讲的还是ASCII呢(最坑爹的是贵国的大学教育，前几天面试一个大学毕业生，计算机专业的，他告诉我他的老师给他们讲的就是ASCII为编码标准呢，我说你别埋汰老师了，你去看看教

材，今天这哥们真给我发短信了，告诉我教材上就是这么说的。)，时代变迁，现在已经变成了**Unicode**了，那么什么是**Unicode**编码呢？还是抄一段来自[维基百科的说明](#)（需要说明一下，本讲不是我qiwsir在讲，是维基百科在讲，我只是一个配角，哈哈）

Unicode（中文：万国码、国际码、统一码、单一码）是计算机科学领域里的一项业界标准。它对世界上大部分的文字系统进行了整理、编码，使得电脑可以用更为简单的方式来呈现和处理文字。

Unicode伴随着通用字符集的标准而发展，同时也以书本的形式对外发表。**Unicode**至今仍在不断增修，每个新版本都加入更多新的字符。目前最新的版本为7.0.0，已收入超过十万个字符（第十万个字符在2005年获采纳）。**Unicode**涵盖的数据除了视觉上的字形、编码方法、标准的字符编码外，还包含了字符特性，如大小写字母。

听这名字：万国码，那就一定包含了中文喽。的确是。但是，光有一个**Unicode**还不行，因为....（此处省略若干字，看官可以到上面给出的维基百科连接中看），还要有其它的一些编码实现方式，**Unicode**的实现方式称为**Unicode**转换格式（**Unicode Transformation Format**，简称为**UTF**），于是乎有了一个我们在很多时候都会看到的**utf-8**。

什么是**utf-8**，还是看[维基百科](#)上怎么说的吧

UTF-8（8-bit **Unicode Transformation Format**）是一种针对**Unicode**的可变长度字符编码，也是一种前缀码。它可以用来表示**Unicode**标准中的任何字符，且其编码中的第一个字节仍与**ASCII**兼容，这使得原来处理**ASCII**字符的软件无须或只须做少部份修改，即可继续使用。因此，它逐渐成为电子邮件、网页及其他存储或发送文字的应用中，优先采用的编码。

不再多引用了，如果要看更多，请到原文。

看官现在是不是就理解了，前面写程序的时候，曾经出现过：`coding:utf-8`的字样。就是在告诉**python**我们要用什么字符编码呢。

encode和decode

历史部分说完了，接下怎么讲？比较麻烦了。因为不管怎么讲，都不是三言两语说清楚的。姑且从**encode()**和**decode()**两个内置函数起吧。

`codecs.encode(obj[, encoding[, errors]])`:Encodes obj using the codec registered for encoding. `codecs.decode(obj[, encoding[, errors]])`:Decodes obj using the codec registered for encoding.

python2默认的编码是**ascii**，通过**encode**可以将对象的编码转换为指定编码格式，而**decode**是这个过程的逆过程。

做一个实验，才能理解：

```

>>> a = "中"
>>> type(a)
<type 'str'>
>>> a
'\xe4\xb8\xad'
>>> len(a)
3

>>> b = a.decode()
>>> b
u'\u4e2d'
>>> type(b)
<type 'unicode'>
>>> len(b)
1

```

这个实验不做之前，或许看官还不是很迷茫（因为不知道，知道的越多越迷茫），实验做完了，自己也迷茫了。别急躁，对编码问题的理解，要慢慢来，如果一时理解不了，也肯定理解不了，就先注意按照要求做，做着做着就豁然开朗了。

上面试验中，变量a引用了一个字符串(str)，严格地将是字节串，它是经过编码后的字节组成的序列。也就是你在上面的实验中，看到的是“中”这个字在计算机中编码之后的字节表示。（关于字节，看官可以google一下）。用len(a)来度量它的长度，它是由三个字节组成的。

然后通过decode函数，将字节串转变为字符串，并且这个字符串是按照unicode编码的。在unicode编码中，一个汉字对应一个字符，这时候度量它的长度就是1.

反过来，一个unicode编码的字符串，也可以转换为字节串。

```

>>> c = b.encode('utf-8')
>>> c
'\xe4\xb8\xad'
>>> type(c)
<type 'str'>
>>> c == a
True

```

关于编码问题，先到这里，点到为止吧。因为再扯，还会扯出问题来。看官肯定感到不满意，因为还没有知其所以然。没关系，请尽情google，即可解决。

python中如何避免中文是乱码

这个问题是一个具有很强操作性的问题。我这里有一个经验总结，分享一下，供参考：

首先，提倡使用utf-8编码方案，因为它跨平台不错。

经验一：在开头声明：

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

有朋友问我-*-有什么作用，那个就是为了好看，爱美之心人皆有，更何况程序员？当然，也可以写成：

```
# coding=utf-8
```

经验二：遇到字符（节）串，立刻转化为unicode，不要用str()，直接使用unicode()

```
unicode_str = unicode('中文', encoding='utf-8')
print unicode_str.encode('utf-8')
```

经验三：如果对文件操作，打开文件的时候，最好用codecs.open，替代open(这个后面会讲到，先放在这里)

```
import codecs
codecs.open('filename', encoding='utf8')
```

我还收集了网上的一篇文章，也挺好的，推荐给看官：[Python2.x的中文显示方法](#)

最后告诉给我，如果用python3，坑爹的编码问题就不烦恼了。

做一个小游戏

在讲述有关list的时候，提到做游戏的事情，后来这个事情一直没有接续。不是忘记了，是在想在哪个阶段做最合适。经过一段时间学习，看官已经不是纯粹小白了，已经属于python初级者了。现在就是开始做那个游戏的时候了。

游戏内容：猜数字游戏

太简单了吧。是的，游戏难度不大，不过这个游戏中蕴含的东西可是值得玩味的。

游戏过程描述

1. 程序运行起来，随机在某个范围内选择一个整数。
2. 提示用户输入数字，也就是猜程序随即选的那个数字。
3. 程序将用户输入的数字与自己选定的对比，一样则用户完成游戏，否则继续猜。
4. 使用次数少的用户得胜。

分析

在任何形式的程序开发之前，不管是大还是小，都要进行分析。即根据功能需求，将不同功能点进行分解。从而确定开发过程。我们现在做一个很小的程序，也是这样做。

随机选择一个数

要实现随机选择一个数字，可以使用python中的一个随机函数：random。下面对这个函数做简要介绍，除了针对本次应用之外，还扩展点，也许别处看官能用上。

还是要首先强化一种学习方法，就是要学会查看帮助文档。

```
>>> import random #这个是必须的，因为不是内置函数
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'SystemRandom',
>>> help(random.randint)

Help on method randint in module random:

randint(self, a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

耐心地看文档，就明白怎么用了。不过，还是把主要的东西列出来，但仍然建议看官在看每个函数的使用之前，在交互模式下通过**help**来查看文档。

随机整数：

```
>>> import random  
>>> random.randint(0,99)  
21
```

随机选取**0**到**100**间的偶数：

```
>>> import random  
>>> random.randrange(0, 101, 2)  
42
```

随机浮点数：

```
>>> import random  
>>> random.random()  
0.85415370477785668  
>>> random.uniform(1, 10)  
5.4221167969800881
```

随机字符：

```
>>> import random  
>>> random.choice('qiwsir.github.io')  
'g'
```

多个字符中选取特定数量的字符：

```
>>> import random  
random.sample('qiwsir.github.io',3)  
['w', 's', 'b']
```

随机选取字符串：

```
>>> import random  
>>> random.choice ( ['apple', 'pear', 'peach', 'orange', 'lemon'] )  
'lemon'
```

洗牌：把原有的顺序打乱，按照随机顺序排列

```
>>> import random
>>> items = [1, 2, 3, 4, 5, 6]
>>> random.shuffle(items)
>>> items
[3, 2, 5, 6, 4, 1]
```

有点多了。不过，本次实验中，值用到了`random.randint()`即可。多出来是买一送一的（哦。忘记了，没有人买呢，本课程全是白送的）。

关键技术点之一已经突破。可以编程了。再梳理一下流程。画个图展示：

（备注：这里我先懒惰一下吧，看官能不能画出这个程序的流程图呢？特别是如果是一个初学者，流程图一定要自己画哦。刚才看到网上一个朋友说自己学编程，但是逻辑思维差，所以没有学好。其实，画流程图就是帮助提高逻辑思维的一种好方式，请画图吧。）

图画好了，按照直观的理解，下面的代码是一个初学者常常写出来的（老鸟们不要喷，因为是代表初学者的）。

```
#!/usr/bin/env python
#coding:utf-8

import random

number = random.randint(1,100)

print "请输入一个100以内的自然数："

input_number = raw_input()

if number == int(input_number):
    print "猜对了，这个数是："
    print number
else:
    print "错了。"
```

上面的程序已经能够基本走通，但是，还有很多缺陷。

最明显的就是只能让人猜一次，不能多次。怎么修改，能够多次猜呢？动动脑筋之后看代码，或者看官在自己的代码上改改，能不能实现多次猜测？

另外，能不能增强一些友好性呢，让用户知道自己输入的数是大了，还是小了。

根据上述修改想法，新代码如下：

```
#!/usr/bin/env python
#coding:utf-8

import random

number = random.randint(1,100)

print "请输入一个100以内的自然数："

input_number = raw_input()

if number == int(input_number):
    print "猜对了，这个数是："
    print number
elif number > int(input_number):
    print "小了"
    input_number = raw_input()
elif number < int(input_number):
    print "大了"
    input_number = raw_input()
else:
    print "错了。"
```

嗯，似乎比原来进步一点点，因为允许用户输入第二次了。同时也告诉用户输入的是大还是小了。但，这也不行呀。应该能够输入很多次，直到正确为止。

是的。这就要用到一个新的东西：循环。如果看官心急，可以google一下while或者for循环，来进一步完善这个游戏，如果不着急，可以等等，随后我也会讲到这部分。

这个游戏还没有完呢，及时用了循环，后面还会继续。

不要红头文件(1)

这两天身体不给力，拖欠了每天发讲座的约定，看官见谅。

红头文件，是某国特别色的东西，在python里不需要，python里要处理的是计算机中的文件，包括文本的、图片的、音频的、视频的等等，还有不少没见过的扩展名的，在linux中，不是所有的东西都被保存到文件中吗？文件，在python中，是一种对象，就如同已经学习过的字符串、数字等一样。

先要在交互模式下查看一下文件都有哪些属性：

```
>>> dir(file)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__', '__getattr__']
```

然后对部分属性进行详细说明，就是看官学习了。

打开文件

在某个文件夹下面建立了一个文件，名曰：130.txt，并且在里面输入了如下内容：

```
learn python
http://qiwsir.github.io
qiwsir@gmail.com
```

此文件以供三行。

下图显示了这个文件的存储位置：



```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105-1.py  106-1.py  111-1.py  129-1.py  130.txt
105.py    109.py    118-1.py  129-2.py
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ python|
```

在上面截图中，我在当前位置输入了python（我已经设置了环境变量，如果你没有，需要写全启动python命令路径），进入到交互模式。在这个交互模式下，这样操作：

```
>>> f = open("130.txt")      #打开已经存在的文件
>>> for line in f:
...     print line
...
learn python

http://qiwsir.github.io

qiwsir@gmail.com
```

将打开的文件，赋值个变量f，这样也就是变量f跟对象文件130.txt用线连起来了（对象引用）。

接下来，用for来读取文件中的内容，就如同读取一个前面已经学过的序列对象一样，如list、str、tuple，把读到的文件中的每行，赋值给变量line。也可以理解为，for循环是一行一行地读取文件内容。每次扫描一行，遇到行结束符号\n表示本行结束，然后是下一行。

从打印的结果看出，每一样跟前面看到的文件内容中的每一行是一样的。只是行与行之间多了一空行，前面显示文章内容的时候，没有这个空行。或许这无关紧要，但是，还要深究一下，才能豁然。

在原文中，每行结束有本行结束符号\n，表示换行。在for语句汇总，print line表示每次打印完line的对象之后，就换行，也就是打印完line的对象之后会增加一个\n。这样看来，在每行末尾就有两个\n，即：\n\n，于是在打印中就出现了一个空行。

```
>>> f = open('130.txt')
>>> for line in f:
...     print line,      #后面加一个逗号，就去掉了原来默认增加的\n了，看看，少了空行。
...
learn python
http://qiwsir.github.io
qiwsir@gmail.com
```

在进行上述操作的时候，有没有遇到这样的情况呢？

```

>>> f = open('130.txt')
>>> for line in f:
...     print line,
...
learn python
http://qiwsir.github.io
qiwsir@gmail.com

>>> for line2 in f:      #在前面通过for循环读取了文件内容之后，再次读取，
...     print line2      #然后打印，结果就什么也显示，这是什么问题？
...
>>>

```

如果看官没有遇到上面问题，可以试试。遇到了，这就解惑。不是什么错误，是因为前一次已经读取了文件内容，并且到了文件的末尾了。再重复操作，就是从末尾开始继续读了。当然显示不了什么东西，但是python并不认为这是错误，因为后面就会讲到，或许在这次读取之前，已经又向文件中追加内容了。那么，如果要再次读取怎么办？就重新来一边好了。

特别提醒看官，因为当前的交互模式是在该文件所在目录启动的，所以，就相当于这个实验室和文件130.txt是同一个目录，这时候我们打开文件130.txt，就认为是在本目录中打开，如果文件不是在本目录中，需要写清楚路径。

比如：在上一级目录中（~/Documents/ITArticles/BasicPython），假如我进入到那个目录中，运行交互模式，然后试图打开130.txt文件。

```

~/Documents/ITArticles/BasicPython/codes$ ls
11-1.py 129-1.py 130.txt
18-1.py 129-2.py
~/Documents/ITArticles/BasicPython/codes$ cd ..
~/Documents/ITArticles/BasicPython$ python

```

```

>>> f = open("130.txt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: '130.txt'

>>> f = open("./codes/130.txt")      #必须得写上路径了（注意，windows的路径是\隔开，需要转义。对转
>>> for line in f:
...     print line
...
learn python

http://qiwsir.github.io

qiwsir@gmail.com

>>>

```

创建文件

上面的实验中，打开的是一个已经存在的文件。如何创建文件呢？

```
>>> nf = open("131.txt", "w")
>>> nf.write("This is a file")
```

就这样创建了一个文件？并写入了文件内容呢？看看再说：

```
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ ls
105-1.py 106-1.py 111-1.py 129-1.py 130.txt
105.py 109.py 118-1.py 129-2.py 131.txt
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$ cat 131.txt
This is a file
qw@qw-Latitude-E4300:~/Documents/ITArticles/BasicPython/codes$
```

真的就这样创建了新文件，并且里面有那句话呢。

看官注意了没有，这次我们同样是用open()这个函数，但是多了个"w"，这是在告诉python用什么样的模式打开文件。也就是说，用open()打开文件，可以有不同的模式打开。看下表：

模式	描述
r	以读方式打开文件，可读取文件信息。
w	以写方式打开文件，可向文件写入信息。如文件存在，则清空该文件，再写入新内容
a	以追加模式打开文件（即一打开文件，文件指针自动移到文件末尾），如果文件不存在则创建
r+	以读写方式打开文件，可对文件进行读和写操作。
w+	消除文件内容，然后以读写方式打开文件。
a+	以读写方式打开文件，并把文件指针移到文件尾。
b	以二进制模式打开文件，而不是以文本模式。该模式只对Windows或Dos有效，类Unix的文件是用二进制模式进行操作的。

从表中不难看出，不同模式下打开文件，可以进行相关的读写。那么，如果什么模式都不写，像前面那样呢？那样就是默认为r模式，只读的方式打开文件。

```
>>> f = open("130.txt")
>>> f
<open file '130.txt', mode 'r' at 0xb7530230>
>>> f = open("130.txt", "r")
>>> f
<open file '130.txt', mode 'r' at 0xb750a700>
```

可以用这种方式查看当前打开的文件是采用什么模式的，上面显示，两种模式是一样的效果。下面逐个对各种模式进行解释

"w":以写方式打开文件，可向文件写入信息。如文件存在，则清空该文件，再写入新内容

131.txt这个文件是存在的，前面建立的，并且在里面写了一句话：This is a file

```
>>> fp = open("131.txt")
>>> for line in fp:          #原来这个文件里面的内容
...     print line
...
This is a file
>>> fp = open("131.txt", "w")  #这时候再看看这个文件，里面还有什么呢？是不是空了呢？
>>> fp.write("My name is qiwsir.\nMy website is qiwsir.github.io") #再查看内容
>>> fp.close()
```

查看文件内容：

```
$ cat 131.txt #cat是linux下显示文件内容的命令，这里就是要显示131.txt内容
My name is qiwsir.
My website is qiwsir.github.io
```

"a":以追加模式打开文件（即一打开文件，文件指针自动移到文件末尾），如果文件不存在则创建

```
>>> fp = open("131.txt", "a")
>>> fp.write("\nAha, I like program\n")  #向文件中追加
>>> fp.close()                         #这是关闭文件，一定要养成一个习惯，写完内容之后就关闭
```

查看文件内容：

```
$ cat 131.txt
My name is qiwsir.
My website is qiwsir.github.io
Aha, I like program
```

其它项目就不一一讲述了。看官可以自己实验。

本讲先到这里，明天继续文件。感冒药吃了，昏昏欲睡。

不要红头文件(2)

在前面学习了基本的打开和建立文件之后，就可以对文件进行多种多样的操作了。请看官要注意，文件，不是什么特别的东西，就是一个对象，如同对待此前学习过的字符串、列表等一样。

文件的属性

所谓属性，就是能够通过一个文件对象得到的东西。

```
>>> f = open("131.txt", "a")
>>> f.name
'131.txt'
>>> f.mode      #显示当前文件打开的模式
'a'
>>> f.closed    #文件是否关闭，如果关闭，返回True；如果打开，返回False
False
>>> f.close()   #关闭文件的内置函数
>>> f.closed
True
```

文件的有关状态

很多时候，我们需要获取一个文件的有关状态（有时候成为属性，但是这里的文件属性和上面的文件属性是不一样的，可是，我觉得称之为文件状态更好一点），比如创建日期，访问日期，修改日期，大小，等等。在os模块中，有这样一个方法，能够解决此问题：

```
>>> import os
>>> file_stat = os.stat("131.txt")      #查看这个文件的状态
>>> file_stat
#文件状态是这样的。有不少从英文单词中可以猜测
posix.stat_result(st_mode=33204, st_ino=5772566L, st_dev=2049L, st_nlink=1, st_uid=1000,
                   st_gid=1000, st_size=0, st_atime=1407734600.0882277, st_mtime=1407734600.0882277, st_ctime=1407734600.0882277)
#这个是文件创建时间
#换一种方式查看这个时间
>>> import time
>>> time.localtime(file_stat.st_ctime)  #这回看清楚了。
time.struct_time(tm_year=2014, tm_mon=8, tm_mday=11, tm_hour=13, tm_min=23, tm_sec=20, tm_wday=5, tm_yday=223, tm_isdst=0)
```

以上关于文件状态和文件属性的内容，在对文件的某些方面进行判断和操作的时候或许会用到。特别是文件属性。比如在操作文件的时候，我们经常要首先判断这个文件是否已经关闭或者打开，就需要用到`file.closed`这个属性来判断了。

文件的内置函数

```
>>> dir(file)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__', '__getattr__',
>>>
```

这么多内置函数，不会都讲述，只能捡着重点的来实验了。

```
>>> f = open("131.txt", "r")
>>> f.read()
'My name is qiwsir.\nMy website is qiwsir.github.io\nAha, I like program\n'
>>>
```

`file.read()`能够将文件中的内容全部读取过来。特别注意，这是返回一个字符串，而且是将文件中的内容全部读到内存中。试想，如果内容太多是不是就有点惨了呢？的确是，千万不要去读大个的文件。

```
>>> content = f.read()
>>> type(content)
<type 'str'>
```

如果文件比较大了，就不要一次都读过来，可以转而一行一行地，用`readline`

```
>>> f = open("131.txt", "r")
>>> f.readline()          #每次返回一行，然后指针向下移动
'My name is qiwsir.\n'
>>> f.readline()          #再读，再返回一行
'My website is qiwsir.github.io\n'
>>> f.readline()
'Aha, I like program\n'
>>> f.readline()          #已经到最后一行了，再读，不报错，返回空
''
```

这个方法，看官是不是觉得太慢了呢？有没有痛快点的呢？有，请挥刀自宫，不用自宫，也能用`readlines`。注意区别，这个是复数，言外之意就是多行啦。

```

>>> f = open("131.txt", "r")
>>> cont = f.readlines()
>>> cont
['My name is qiwsir.\n', 'My website is qiwsir.github.io\n', 'Aha, I like program\n']
>>> type(cont)
<type 'list'>
>>> for line in cont:
...     print line
...
My name is qiwsir.

My website is qiwsir.github.io

Aha, I like program

```

从实验中我们可以看到，`readlines`和`read`有一样之处，都是将文件内容一次性读出来，存放在内存，但是两者也有区别，`read`返回的是`str`类型，`readlines`返回的是`list`，而且一行一个元素，因此，就可以通过`for`逐行打印出来了。

在`print line`中，注意观察`list`里面的每个元素，最后都是一个`\n`结尾，所以打印的结果会有空行。其原因前面已经介绍过了，忘了的朋友请回滚到[上一讲](#)

不过，还是要提醒列位，太大的文件不用都读到内存中。对付大点的文件，还是推荐这么做：

```

>>> f = open("131.txt", "r")
>>> f
<open file '131.txt', mode 'r' at 0xb757c230>
>>> type(f)
<type 'file'>
>>> for line in f:
...     print line
...
My name is qiwsir.

My website is qiwsir.github.io

Aha, I like program

```

以上都是读文件的内置函数和方法。除了读，就是要写。所谓写，就是将内容存入到文件中。用到的内置函数是`write`。但是，要写入文件，还要注意打开文件的模式，可以是`w`，也可以是`a`，看具体情况而定。

```

>>> f = open("131.txt", "a")      #因为这个文件已经存在，我又不想清空，用追加的模式
>>> f.write("There is a baby.") #这句话应该放到文件最后
>>> f.close()                  #请看官注意，写了之后，一定要及时关闭文件。才能代表真正写入

```

看看写的效果：

```
>>> f = open("131.txt","r")
>>> for line in f.readlines():
...     print line
...
My name is qiwsir.

My website is qiwsir.github.io

Aha, I like program

There is a baby.      #果然增加了这一行
```

以上是关于文件的基本操作。其实对文件远远不知这些，有兴趣的看官可以google一下pickle这个模块，是一个很好用的东西。