

Python教程

33602次阅读

这是小白的Python新手教程。

Python是一种计算机程序设计语言。你可能已经听说过很多种流行的编程语言，比如非常难学的C语言，非常流行的Java语言，适合初学者的Basic语言，适合网页编程的JavaScript语言，等等。

那Python是一种什么语言？

首选，我们普及一下编程语言的基础知识。用任何编程语言来开发程序，都是为了让计算机干活，比如下载一个MP3，编写一个文档等等，而计算机干活的CPU只认识机器指令，所以，尽管不同的编程语言差异极大，最后都得“翻译”成CPU可以执行的机器指令。而不同的编程语言，干同一个活，编写的代码量，差距也很大。

比如，完成同一个任务，C语言要写1000行代码，Java只需要写100行，而Python可能只要20行。

所以Python是一种相当高级的语言。

你也许会问，代码少还不好？代码少的代价是运行速度慢，C程序运行1秒钟，Java程序可能需要2秒，而Python程序可能就需要10秒。

那是不是越低级的程序越难学，越高级的程序越简单？表面上来说，是的，但是，在非常高的抽象计算中，高级的Python程序设计也是非常难学的，所以，高级程序语言不等于简单。

但是，对于初学者和完成普通任务，Python语言是非常简单易用的。连Google都在大规模使用Python，你就不用担心学了会没用。

用Python可以做什么？可以做日常任务，比如自动备份你的MP3；可以做网站，很多著名的网站包括YouTube就是Python写的；可以做网络游戏的后台，很多在线游戏的后台都是Python开发的。总之就是能干很多很多事啦。

Python当然也有不能干的事情，比如写操作系统，这个只能用C语言写；写手机应用，只能用Objective-C（针对iPhone）和Java（针对Android）；写3D游戏，最好用C或C++。

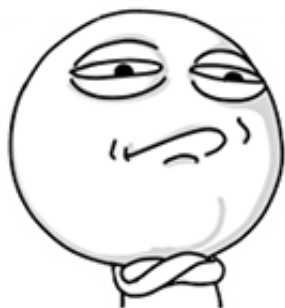
如果你是小白用户，满足以下条件：

- 会使用电脑，但从来没写过程序；
- 还记得初中数学学的方程式和一点点代数知识；
- 想从编程小白变成专业的软件架构师；
- 每天能抽出半个小时学习。

不要再犹豫了，这个教程就是为你准备的！

准备好了吗？

CHALLENGE ACCEPTED !



关于作者

[廖雪峰](#)，十年软件开发经验，业余产品经理，精通Java/Python/Ruby/Visual Basic/Objective C等，对开源框架有深入研究，著有《Spring 2.0核心技术与最佳实践》一书，多个业余开源项目托管在[GitHub](#)，欢迎微博交流：

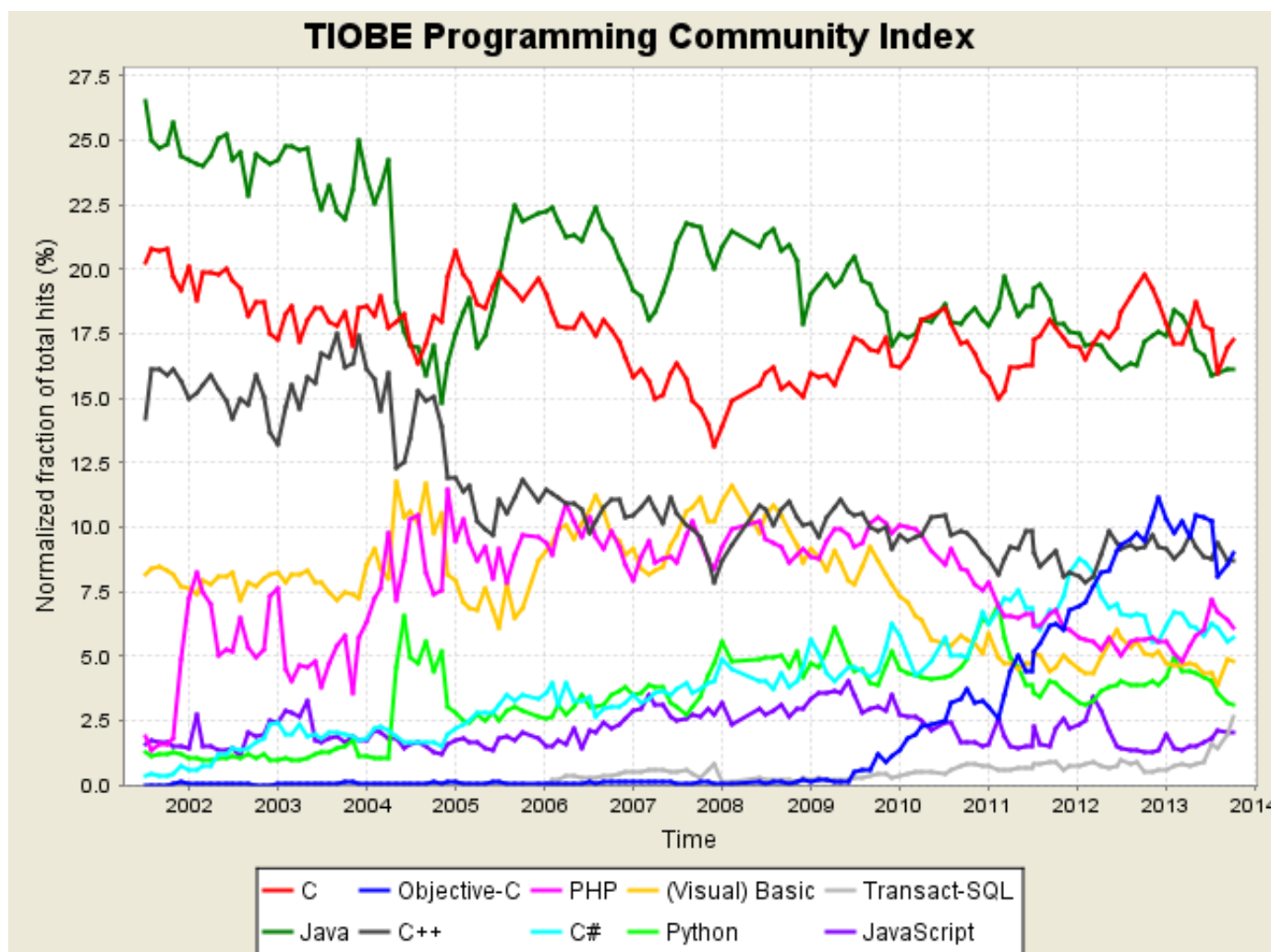


Python简介

6348次阅读

Python是著名的“龟叔”Guido van Rossum在1989年圣诞节期间，为了打发无聊的圣诞节而编写的一个编程语言。

现在，全世界差不多有600多种编程语言，但流行的编程语言也就那么20来种。如果你听说过TIOBE排行榜，你就能知道编程语言的大致流行程度。这是最近10年最常用的10种编程语言的变化图：



总的来说，这几种编程语言各有千秋。C语言是可以用来编写操作系统的贴近硬件的语言，所以，C语言适合开发那些追求运行速度、充分发挥硬件性能的程序。而Python是用来编写应用程序的高级编程语言。

当你用一种语言开始作真正的软件开发时，你除了编写代码外，还需要很多基本的已经写好的现成的东西，来帮助你加快开发进度。比如说，要编写一个电子邮件客户端，如果先从最底层开始编写网络协议相关的代码，那估计一年半载也开发不出来。高级编程语言通常都会提供一个比较完善的基础代码库，让你能直接调用，比如，针对电子邮件协议的SMTP库，针对桌面环境的GUI库，在这些已有的代码库的基础上开发，一个电子邮件客户端几天就能开发出来。

Python就为我们提供了非常完善的基础代码库，覆盖了网络、文件、GUI、数据库、文本等大量内容，被形象地称作“内置电池（batteries included）”。用Python开发，许多功能不必从零编写，直接使用现成的即可。

除了内置的库外，Python还有大量的第三方库，也就是别人开发的，供你直接使用的东西。当

然，如果你开发的代码通过很好的封装，也可以作为第三方库给别人使用。

许多大型网站就是用Python开发的，例如YouTube、[Instagram](#)，还有国内的[豆瓣](#)。很多大公司，包括Google、Yahoo等，甚至[NASA](#)（美国航空航天局）都大量地使用Python。

龟叔给Python的定位是“优雅”、“明确”、“简单”，所以Python程序看上去总是简单易懂，初学者学Python，不但入门容易，而且将来深入下去，可以编写那些非常非常复杂的程序。

总的来说，Python的哲学就是简单优雅，尽量写容易看明白的代码，尽量写少的代码。如果一个资深程序员向你炫耀他写的晦涩难懂、动不动就几万行的代码，你可以尽情地嘲笑他。

那Python适合开发哪些类型的应用呢？

首选是[网络应用](#)，包括网站、后台服务等等；

其次是许多日常需要的[小工具](#)，包括系统管理员需要的脚本任务等等；

另外就是把[其他语言开发的程序再包装起来](#)，方便使用。

最后说说Python的缺点。

任何编程语言都有缺点，Python也不例外。优点说过了，那Python有哪些缺点呢？

第一个缺点就是运行[速度慢](#)，和C程序相比非常慢，[因为Python是解释型语言，你的代码在执行时会一行一行地翻译成CPU能理解的机器码，这个翻译过程非常耗时](#)，所以很慢。而C程序是运行前直接编译成CPU能执行的机器码，所以非常快。

但是大量的应用程序不需要这么快的运行速度，因为用户根本感觉不出来。例如开发一个下载MP3的网络应用程序，C程序的运行时间需要0.001秒，而Python程序的运行时间需要0.1秒，慢了100倍，但由于网络更慢，需要等待1秒，你想，用户能感觉到1.001秒和1.1秒的区别吗？这就好比F1赛车和普通的出租车在北京三环路上行驶的道理一样，虽然F1赛车理论时速高达400公里，但由于三环路堵车的时速只有20公里，因此，作为乘客，你感觉的时速永远是20公里。

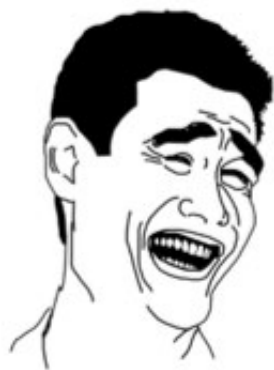


第二个缺点就是[代码不能加密](#)。如果要发布你的Python程序，实际上就是发布源代码，这一点跟C语言不同，C语言不用发布源代码，只需要把编译后的机器码（也就是你在Windows上常见的xxx.exe文件）发布出去。要从机器码反推出C代码是不可能的，所以，凡是编译型的语言，都没有这个问题，而解释型的语言，则必须把源码发布出去。

这个缺点仅限于你要编写的软件需要卖给别人挣钱的时候。好消息是目前的互联网时代，靠卖软件授权的商业模式越来越少了，靠网站和移动应用卖服务的模式越来越多了，后一种模式不需要把源码给别人。

再说了，现在如火如荼的开源运动和互联网自由开放的精神是一致的，互联网上有无数非常优秀的像Linux一样的开源代码，我们千万不要高估自己写的代码真的有非常大的“商业价

值”。那些大公司的代码不愿意开放的更重要的原因是代码写得太烂了，一旦开源，就没人敢用他们的产品了。



大家都那么忙，
哪有闲功夫破解你的烂代码

当然，Python还有其他若干小缺点，请自行忽略，就不一一列举了。

安装Python

6838次阅读

因为Python是跨平台的，它可以运行在Windows、Mac和各种Linux/Unix系统上。在Windows上写Python程序，放到Linux上也是能够运行的。

要开始学习Python编程，首先就得把Python安装到你的电脑里。安装后，你会得到Python解释器（就是负责运行Python程序的），一个命令行交互环境，还有一个简单的集成开发环境。

2. x还是3. x

目前，Python有两个版本，一个是2. x版，一个是3. x版，这两个版本是不兼容的，因为现在Python正在朝着3. x版本进化，在进化过程中，大量的针对2. x版本的代码要修改后才能运行，所以，目前有许多第三方库还暂时无法在3. x上使用。

为了保证你的程序能用到大量的第三方库，我们的教程仍以2. x版本为基础，确切地说，是2. 7版本。请确保你的电脑上安装的Python版本是2. 7. x，这样，你才能无痛学习这个教程。

在Mac上安装Python

如果你正在使用Mac，系统是OS X 10. 8或者最新的10. 9 Mavericks，恭喜你，系统自带了Python 2. 7。如果你的系统版本低于10. 8，请自行备份系统并免费升级到最新的10. 9，就可以获得Python 2. 7。

查看系统版本的办法是点击左上角的苹果图标，选择“关于本机”：



在Linux上安装Python

如果你正在使用Linux，那我可以假定你有Linux系统管理经验，自行安装Python 2. 7应该没有

问题，否则，请换回Windows系统。

对于大量的目前仍在使用Windows的同学，如果短期内没有打算换Mac，就可以继续阅读以下内容。

在Windows上安装Python

首先，从Python的官方网站www.python.org下载最新的2.7.6版本，地址是这个：

<http://www.python.org/ftp/python/2.7.6/python-2.7.6.msi>

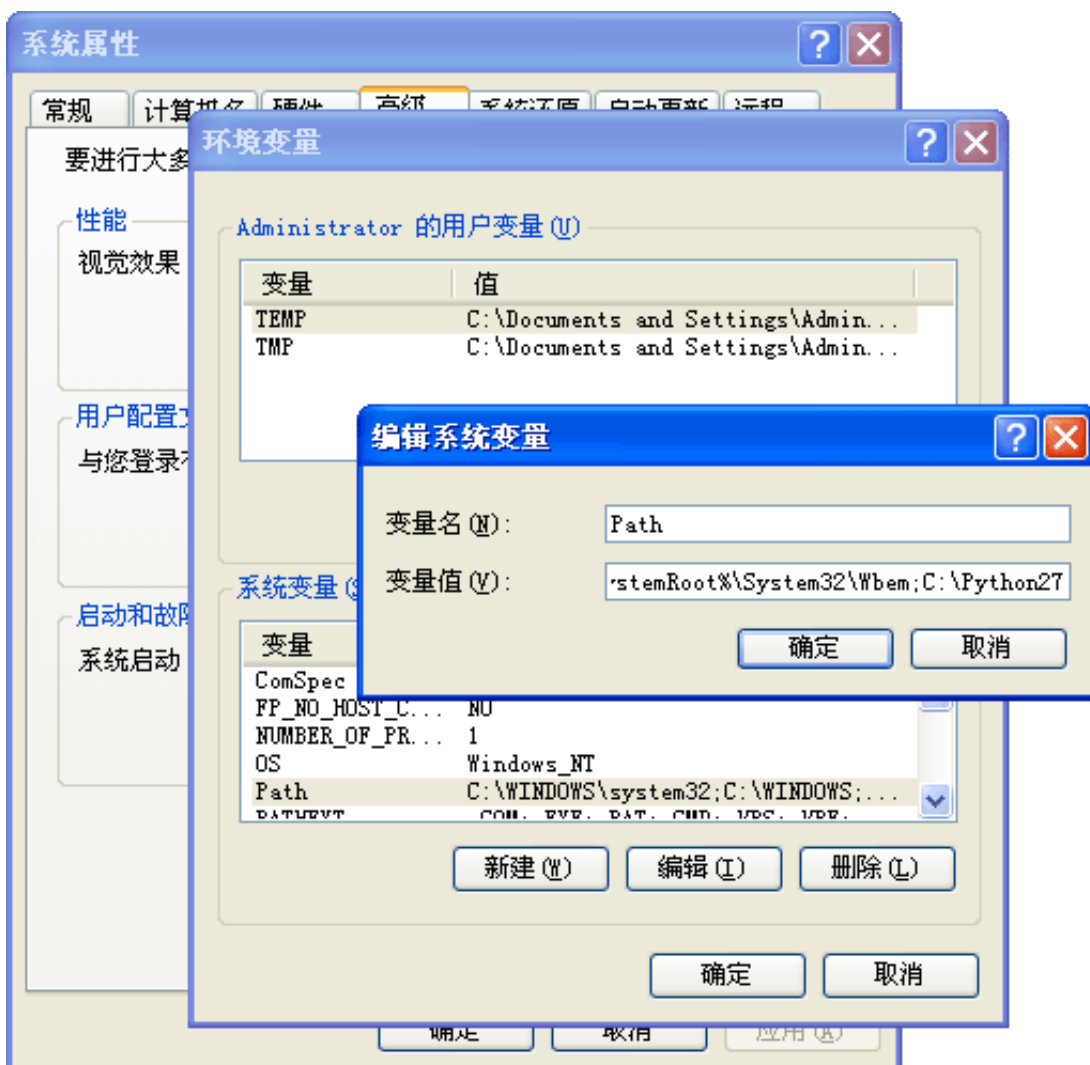
然后，运行下载的MSI安装包，不需要更改任何默认设置，直接一路点“Next”即可完成安装：

默认会安装到C:\Python27目录下，但是当你兴致勃勃地打开命令提示符窗口，敲入python后，会得到：

‘python’不是内部或外部命令，也不是可运行的程序或批处理文件。

这是因为Windows会根据一个Path的环境变量设定的路径去查找python.exe，如果没找到，就会报错。解决办法是把python.exe所在的路径C:\Python27添加到Path中。

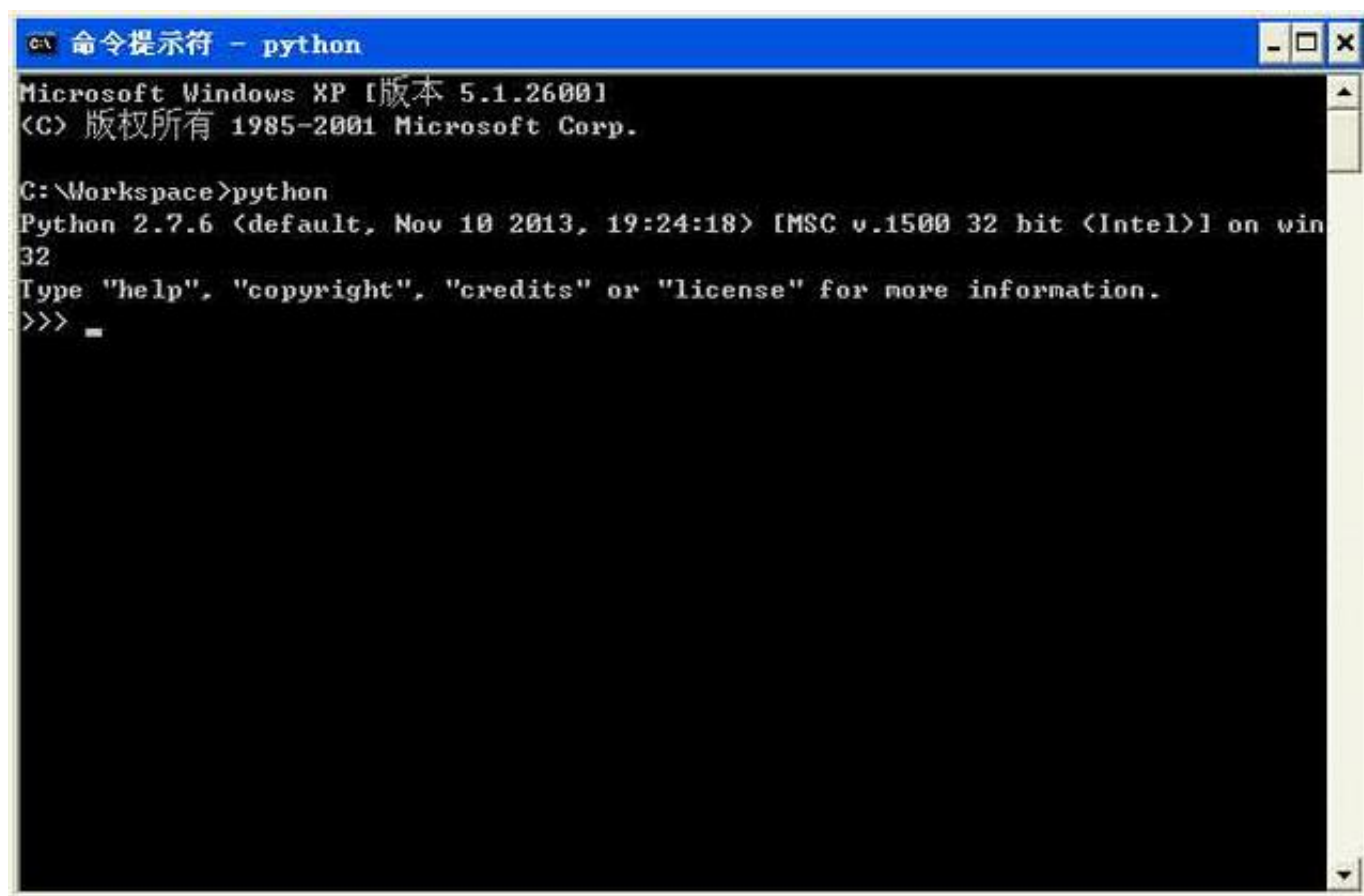
在控制面板中打开“系统属性”，点击“高级”，“环境变量”，打开“环境变量”窗口，在系统变量中，找到“Path”变量，然后点击“编辑”：



在“编辑系统变量”的窗口中，可以看到，变量名是Path，在变量值的最后面，先添加一个分

号“;”（注意用英文输入法，千万不要输入中文分号），再写上C:\Python27（如果安装的时候没有更改过安装目录），然后连续点“确定”，“确定”，“确定”把所有窗口都关掉。

现在，再打开一个新的命令行窗口（一定要关掉原来的命令行窗口，再新开一个），输入python：



```
命令提示符 - python
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Workspace>python
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

看到上面的画面，就说明Python安装成功！

你看到提示符>>>就表示我们已经在Python交互式环境中了，可以输入任何Python代码，回车后会立刻得到执行结果。现在，输入exit()并回车，就可以退出Python交互式环境（直接关掉命令行窗口也可以！）。

小结

学会如何把Python安装到计算机中，并且熟练打开和退出Python交互式环境。

Python解释器

4050次阅读

当我们编写Python代码时，我们得到的是一个包含Python代码的以.py为扩展名的文本文件。要运行代码，就需要Python解释器去执行.py文件。

由于整个Python语言从规范到解释器都是开源的，所以理论上，只要水平够高，任何人都可以编写Python解释器来执行Python代码（当然难度很大）。事实上，确实存在多种Python解释器。

CPython

当我们从[Python官方网站](#)下载并安装好Python 2.7后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。

CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行。

IPython

IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。好比很多国产浏览器虽然外观不同，但内核其实都是调用了IE。

CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。

PyPy

PyPy是另一个Python解释器，它的目标是执行速度。PyPy采用[JIT技术](#)，对Python代码进行动态编译（注意不是解释），所以可以显著提高Python代码的执行速度。

绝大部分Python代码都可以在PyPy下运行，但是PyPy和CPython有一些是不同的，这就导致相同的Python代码在两种解释器下执行可能会有不同的结果。如果你的代码要放到PyPy下执行，就需要了解[PyPy和CPython的不同点](#)。

Jython

Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。

IronPython

IronPython和Jython类似，只不过IronPython是运行在微软.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码。

小结

Python的解释器很多，但使用最广泛的还是CPython。如果要和Java或.Net平台交互，最好的办法不是用Jython或IronPython，而是通过网络调用来交互，确保各程序之间的独立性。

本教程的所有代码只确保在CPython 2.7版本下运行。请务必在本地安装CPython（也就是从Python官方网站下载的安装程序）。

此外，教程还内嵌一个IPython的Web版本，用来在浏览器内练习执行一些Python代码。要注意两者功能一样，输入的代码一样，但是提示符有所不同。另外，不是所有代码都能在Web版本的IPython中执行，出于安全原因，很多操作（比如文件操作）是受限的，所以有些代码必须在本地环境执行代码。

第一个Python程序

6402次阅读

现在，了解了如何启动和退出Python的交互式环境，我们就可以正式开始编写Python代码了。

在写代码之前，请千万不要用“复制”-“粘贴”把代码从页面粘贴到你自己的电脑上。写程序也讲究一个感觉，你需要一个字母一个字母地把代码自己敲进去，在敲代码的过程中，初学者经常会敲错代码，所以，你需要仔细地检查、对照，才能以最快的速度掌握如何写程序。

在交互式环境的提示符`>>>`下，直接输入代码，按回车，就可以立刻得到代码执行结果。现在，试试输入`100+200`，看看计算结果是不是300：

```
>>> 100+200
300
```

很简单吧，任何有效的数学计算都可以算出来。

如果要让Python打印出指定的文字，可以用`print`语句，然后把希望打印的文字用单引号或者双引号括起来，但不能混用单引号和双引号：

```
>>> print 'hello, world'
hello, world
```

这种用单引号或者双引号括起来的文本在程序中叫字符串，今后我们还会经常遇到。

最后，用`exit()`退出Python，我们的第一个Python程序完成！唯一的缺憾是没有保存下来，下次运行时还要再输入一遍代码。

小结

在Python交互式命令行下，可以直接输入代码，然后执行，并立刻得到结果。



使用文本编辑器

6524次阅读

在Python的交互式命令行写程序，好处是一下就能得到结果，坏处是没法保存，下次还想运行的时候，还得再敲一遍。

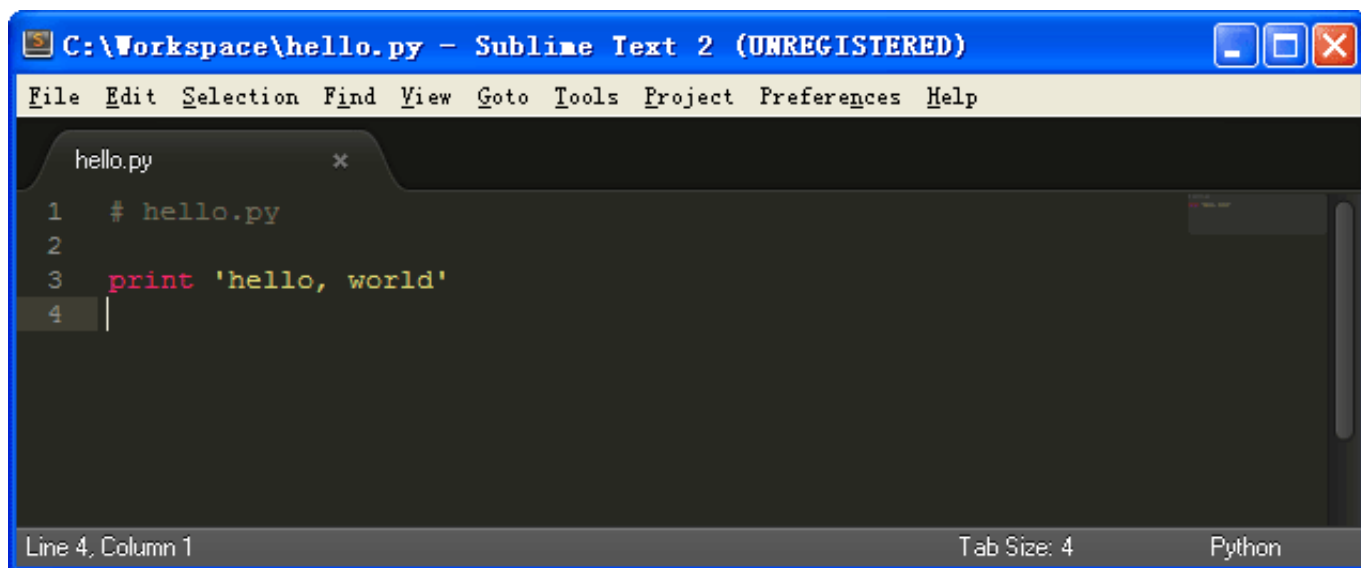
所以，实际开发的时候，我们总是使用一个文本编辑器来写代码，写完了，保存为一个文件，这样，程序就可以反复运行了。

现在，我们就把上次的'hello, world'程序用文本编辑器写出来，保存下来。

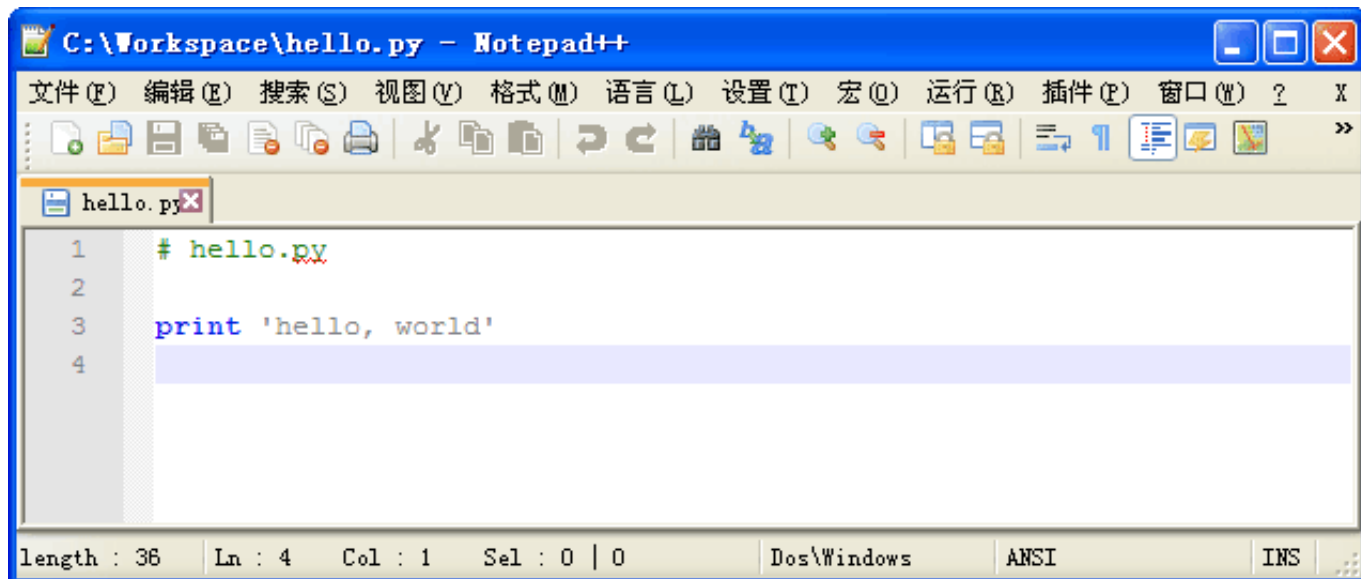
所以问题又变成了：用什么文本编辑器？

推荐两款文本编辑器：

一个是[Sublime Text](#)，免费使用，但是不付费会弹出提示框：



一个是[Notepad++](#)，免费使用，有中文界面：



请注意，用哪个都行，但是**绝对不能用Word和Windows自带的记事本**。Word保存的不是纯文本文件，而记事本会自作聪明地在文件开始的地方加上几个特殊字符（UTF-8 BOM），结果会导

致程序运行出现莫名其妙的错误。

安装好文本编辑器后，输入以下代码：

```
print 'hello, world'
```

注意print前面不要有任何空格。然后，选择一个目录，例如C:\Workspace，把文件保存为hello.py，就可以打开命令行窗口，把当前目录切换到hello.py所在目录，就可以运行这个程序了：

```
C:\Workspace>python hello.py  
hello, world
```

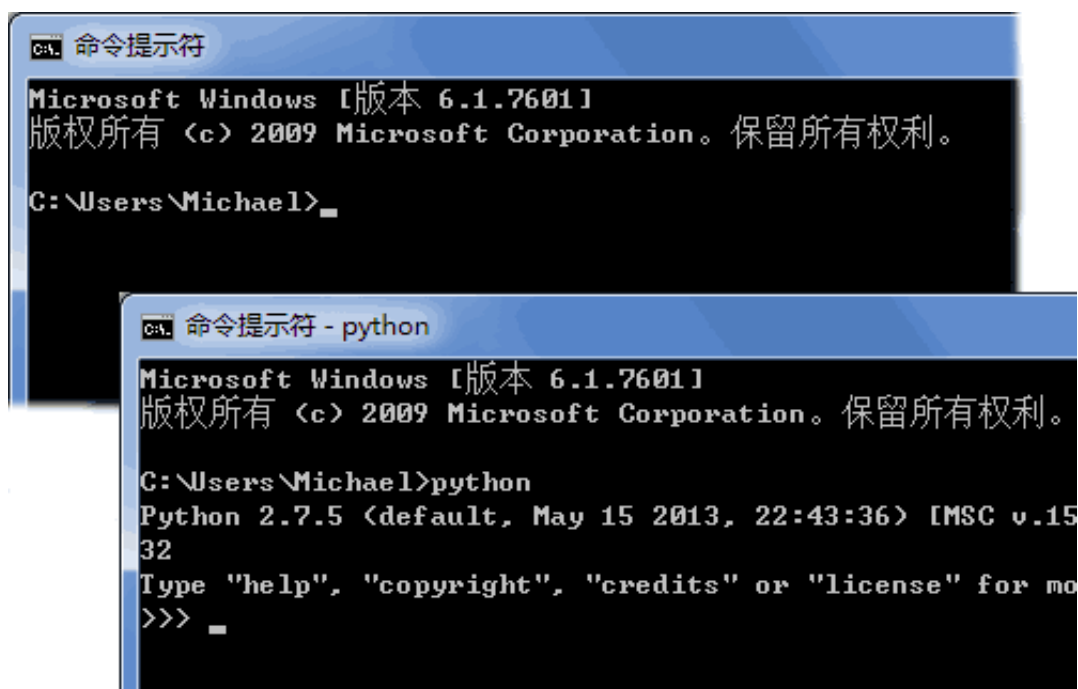
也可以保存为别的名字，比如abc.py，但是必须要以.py结尾，其他的都不行。此外，文件名只能是英文字母、数字和下划线的组合。

如果当前目录下没有hello.py这个文件，运行python hello.py就会报错：

```
python hello.py  
python: can't open file 'hello.py': [Errno 2] No such file or directory
```

报错的意思就是，无法打开hello.py这个文件，因为文件不存在。这个时候，就要检查一下当前目录下是否有这个文件了。

请注意区分命令行模式和Python交互模式：



看到类似C:\>是在Windows提供的命令行模式，看到>>>是在Python交互式环境下。

在命令行模式下，可以执行python进入Python交互式环境，也可以执行python hello.py运行一个.py文件，但是在Python交互式环境下，只能输入Python代码执行。

直接运行py文件

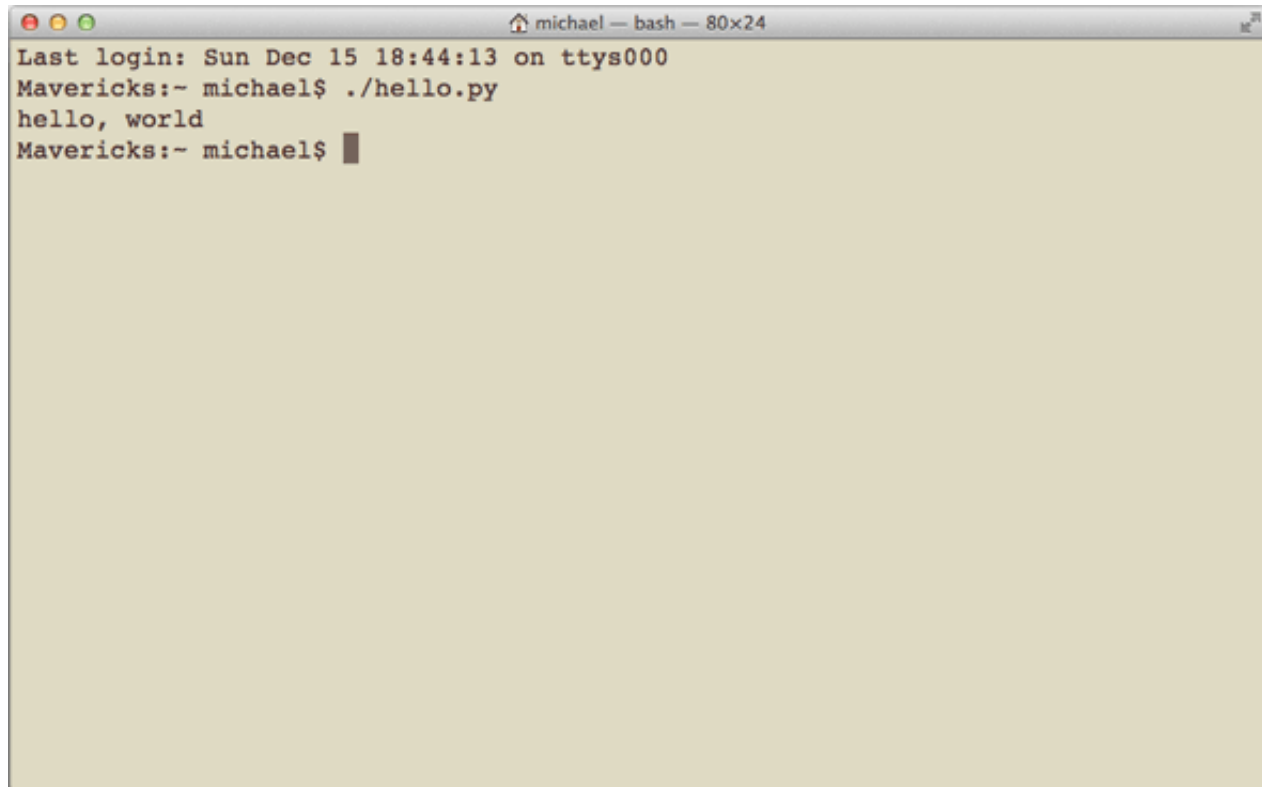
还有同学问，能不能像.exe文件那样直接运行.py文件呢？在Windows上是不行的，但是，在Mac和Linux上是可以的，方法是在.py文件的第一行加上：

```
#!/usr/bin/env python
```

然后，通过命令：

```
$ chmod a+x hello.py
```

就可以直接运行hello.py了，比如在Mac下运行：

A screenshot of a macOS terminal window titled "michael - bash - 80x24". The terminal shows the following text: "Last login: Sun Dec 15 18:44:13 on ttys000", "Mavericks:~ michael\$./hello.py", "hello, world", and "Mavericks:~ michael\$". The prompt "Mavericks:~ michael\$" is followed by a cursor. The terminal has a light beige background and a dark grey title bar with standard macOS window controls (red, yellow, green buttons) on the left and a close button on the right.

```
michael - bash - 80x24
Last login: Sun Dec 15 18:44:13 on ttys000
Mavericks:~ michael$ ./hello.py
hello, world
Mavericks:~ michael$
```

小结

用文本编辑器写Python程序，然后保存为后缀为.py的文件，就可以用Python直接运行这个程序了。

用Python开发程序，完全可以一边在文本编辑器里写代码，一边开一个交互式命令窗口，在写代码的过程中，把部分代码粘到命令行去验证，事半功倍！前提是得有个27'的超大显示器！

输入和输出

5905次阅读

输出

用`print`加上字符串，就可以向屏幕上输出指定的文字。比如输出'hello, world'，用代码实现如下：

```
>>> print 'hello, world'
```

`print`语句也可以跟上多个字符串，用逗号“,”隔开，就可以连成一串输出：

```
>>> print 'The quick brown fox', 'jumps over', 'the lazy dog'
The quick brown fox jumps over the lazy dog
```

`print`会依次打印每个字符串，遇到逗号“,”会输出一个空格，因此，输出的字符串是这样拼起来的：

```
print 'The quick brown fox' , 'jumps over' , 'the lazy dog'

      ↓           ↓           ↓           ↓           ↓
The quick brown fox jumps over the lazy dog
```

`print`也可以打印整数，或者计算结果：

```
>>> print 300
300
>>> print 100 + 200
300
```

因此，我们可以把计算`100 + 200`的结果打印得更漂亮一点：

```
>>> print '100 + 200 = ', 100 + 200
100 + 200 = 300
```

注意，对于`100 + 200`，Python解释器自动计算出结果300，但是，`'100 + 200 ='`是字符串而非数学公式，Python把它视为字符串，请自行解释上述打印结果。

输入

现在，你已经可以用`print`输出你想要的结果了。但是，如果要从用户从电脑输入一些字符怎么办？Python提供了一个`raw_input`，可以让用户输入字符串，并存放到一个变量里。比如输入用户的名字：

```
>>> name = raw_input()
Michael
```

当你输入`name = raw_input()`并按下回车后，Python交互式命令行就在等待你的输入了。这时，你可以输入任意字符，然后按回车后完成输入。

输入完成后，不会有任何提示，Python交互式命令行又回到`>>>`状态了。那我们刚才输入的内容到哪去了？答案是存放到`name`变量里了。可以直接输入`name`查看变量内容：

```
>>> name
```

```
'Michael'
```

什么是变量？请回忆初中数学所学的代数基础知识：

设正方形的边长为 a ，则正方形的面积为 $a \times a$ 。把边长 a 看做一个变量，我们就可以根据 a 的值计算正方形的面积，比如：

若 $a=2$ ，则面积为 $a \times a = 2 \times 2 = 4$ ；

若 $a=3.5$ ，则面积为 $a \times a = 3.5 \times 3.5 = 12.25$ 。

在计算机程序中，变量不仅可以为整数或浮点数，还可以是字符串，因此，`name`作为一个变量就是一个字符串。

要打印出`name`变量的内容，除了直接写`name`然后按回车外，还可以用`print`语句：

```
>>> print name
Michael
```

有了输入和输出，我们就可以把上次打印‘hello, world’的程序改成有点意义的程序了：

```
name = raw_input()
print 'hello,', name
```

运行上面的程序，第一行代码会让用户输入任意字符作为自己的名字，然后存入`name`变量中；第二行代码会根据用户的名字向用户说hello，比如输入Michael：

```
C:\Workspace> python hello.py
Michael
hello, Michael
```

但是程序运行的时候，没有任何提示信息告诉用户：“嘿，赶紧输入你的名字”，这样显得很不好。幸好，`raw_input`可以让你显示一个字符串来提示用户，于是我们把代码改成：

```
name = raw_input('please enter your name: ')
print 'hello,', name
```

再次运行这个程序，你会发现，程序一运行，会首先打印出`please enter your name:`，这样，用户就可以根据提示，输入名字后，得到`hello, xxx`的输出：

```
C:\Workspace> python hello.py
please enter your name: Michael
hello, Michael
```

每次运行该程序，根据用户输入的不同，输出结果也会不同。

在命令行下，输入和输出就是这么简单。

小结

任何计算机程序都是为了执行一个特定的任务，有了输入，用户才能告诉计算机程序所需的信息，有了输出，程序运行后才能告诉用户任务的结果。

输入是Input，输出是Output，因此，我们把输入输出统称为Input/Output，或者简写为IO。

`raw_input`和`print`是在命令行下面最基本的输入和输出，但是，用户也可以通过其他更高级的图形界面完成输入和输出，比如，在网页上的一个文本框输入自己的名字，点击“确定”后在网页上看到输出信息。

Python基础

4280次阅读

Python是一种计算机编程语言。计算机编程语言和我们日常使用的自然语言有所不同，最大的区别就是，自然语言在不同的语境下有不同的理解，而计算机要根据编程语言执行任务，就必须保证编程语言写出的程序决不能有歧义，所以，任何一种编程语言都有自己的一套语法，编译器或者解释器就是负责把符合语法的程序代码转换成CPU能够执行的机器码，然后执行。Python也不例外。

Python的语法比较简单，采用缩进方式，写出来的代码就像下面的样子：

```
# print absolute value of an integer:
a = 100
if a >= 0:
    print a
else:
    print -a
```

以#开头的语句是注释，注释是给人看的，可以是任意内容，解释器会忽略掉注释。其他每一行都是一个语句，当语句以冒号“:”结尾时，缩进的语句视为代码块。

缩进有利有弊。好处是强迫你写出格式化的代码，但没有规定缩进是几个空格还是Tab。按照约定俗成的管理，应该始终坚持使用4个空格的缩进。

缩进的另一个好处是强迫你写出缩进较少的代码，你会倾向于把一段很长的代码拆分成若干函数，从而得到缩进较少的代码。

缩进的坏处就是“复制—粘贴”功能失效了，这是最坑爹的地方。当你重构代码时，粘贴过去的代码必须重新检查缩进是否正确。此外，IDE很难像格式化Java代码那样格式化Python代码。

最后，请务必注意，Python程序是大小写敏感的，如果写错了大小写，程序会报错。

数据类型和变量

5796次阅读

数据类型

计算机顾名思义就是可以做数学计算的机器，因此，计算机程序理所当然地可以处理各种数值。但是，计算机能处理的远不止数值，还可以处理文本、图形、音频、视频、网页等各种各样的数据，不同的数据，需要定义不同的数据类型。在Python中，能够直接处理的数据类型有以下几种：

整数

Python可以处理任意大小的整数，当然包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：1，100，-8080，0，等等。

计算机由于使用二进制，所以，有时候用十六进制表示整数比较方便，十六进制用0x前缀和0-9，a-f表示，例如：0xff00，0xa5b4c3d2，等等。

浮点数

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，比如， 1.23×10^9 和 12.3×10^8 是相等的。浮点数可以用数学写法，如1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把10用e替代， 1.23×10^9 就是1.23e9，或者 12.3×10^8 ，0.000012可以写成1.2e-5，等等。

整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的（除法难道也是精确的？是的！），而浮点数运算则可能会有四舍五入的误差。

字符串

字符串是以''或""括起来的任意文本，比如'abc'，"xyz"等等。请注意，''或""本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有a，b，c这3个字符。如果'本身也是一个字符，那就可以用""括起来，比如"I'm OK"包含的字符是I，'，m，空格，O，K这6个字符。

如果字符串内部既包含'又包含"怎么办？可以用转义字符\来标识，比如：

```
'I\'m \'OK\'!'
```

表示的字符串内容是：

```
I'm "OK"!
```

转义字符\可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\\表示的字符就是\，可以在Python的交互式命令行用print打印字符串看看：

```
>>> print 'I\'m ok.'
I'm ok.
>>> print 'I\'m learning\nPython.'
I'm learning
Python.
>>> print '\\n\\'
\
\
```

如果字符串里面有很多字符都需要转义，就需要加很多\，为了简化，Python还允许用r''表示''内部的字符串默认不转义，可以自己试试：

```
>>> print '\\t\\'
\
>>> print r'\\t\\'
\\t\\
```

如果字符串内部有很多换行，用\n写在一行里不好阅读，为了简化，Python允许用'''...'''的格式表示多行内容，可以自己试试：

```
>>> print '''line1
... line2
... line3'''
line1
line2
line3
```

上面是在交互式命令行内输入，如果写成程序，就是：

```
print '''line1
line2
line3'''
```

多行字符串'''...'''还可以在前面加上r使用，请自行测试。

布尔值

布尔值和布尔代数的表示完全一致，一个布尔值只有True、False两种值，要么是True，要么是False，在Python中，可以直接用True、False表示布尔值（请注意大小写），也可以通过布尔运算计算出来：

```
>>> True
True
>>> False
False
>>> 3 > 2
True
>>> 3 > 5
False
```

布尔值可以用and、or和not运算。

and运算是与运算，只有所有都为True，and运算结果才是True：

```
>>> True and True
True
>>> True and False
False
>>> False and False
False
```

or运算是或运算，只要其中有一个为True，or运算结果就是True：

```
>>> True or True
True
>>> True or False
True
>>> False or False
False
```


not运算是非运算，它是一个单目运算符，把True变成False，False变成True：

```
>>> not True
False
>>> not False
True
```

布尔值经常用在条件判断中，比如：

```
if age >= 18:
    print 'adult'
else:
    print 'teenager'
```

空值

空值是Python里一个特殊的值，用None表示。None不能理解为0，因为0是有意义的，而None是一个特殊的空值。

此外，Python还提供了列表、字典等多种数据类型，还允许创建自定义数据类型，我们后面会继续讲到。

变量

变量的概念基本上和初中代数的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意数据类型。

变量在程序中就是用一个变量名表示了，变量名必须是大小写英文、数字和_的组合，且不能用数字开头，比如：

```
a = 1
```

变量a是一个整数。

```
t_007 = 'T007'
```

变量t_007是一个字符串。

```
Answer = True
```

变量Answer是一个布尔值True。

在Python中，等号=是赋值语句，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，例如：

```
a = 123 # a是整数
print a
a = 'ABC' # a变为字符串
print a
```

这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。例如Java是静态语言，赋值语句如下（// 表示注释）：

```
int a = 123; // a是整数类型变量
a = "ABC"; // 错误：不能把字符串赋给整型变量
```

和静态语言相比，动态语言更灵活，就是这个原因。

请不要把赋值语句的等号等同于数学的等号。比如下面的代码：

```
x = 10
x = x + 2
```

如果从数学上理解 $x = x + 2$ 那无论如何是不成立的，在程序中，赋值语句先计算右侧的表达式 $x + 2$ ，得到结果12，再赋给变量 x 。由于 x 之前的值是10，重新赋值后， x 的值变成12。

最后，理解变量在计算机内存中的表示也非常重要。当我们写：

```
a = 'ABC'
```

时，Python解释器干了两件事情：

1. 在内存中创建了一个'ABC'的字符串；
2. 在内存中创建了一个名为a的变量，并把它指向'ABC'。

也可以把一个变量a赋值给另一个变量b，这个操作实际上是把变量b指向变量a所指向的数据，例如下面的代码：

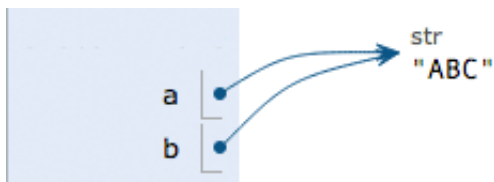
```
a = 'ABC'
b = a
a = 'XYZ'
print b
```

最后一行打印出变量b的内容到底是'ABC'呢还是'XYZ'？如果从数学意义上理解，就会错误地得出b和a相同，也应该是'XYZ'，但实际上b的值是'ABC'，让我们一行一行地执行代码，就可以看到到底发生了什么事：

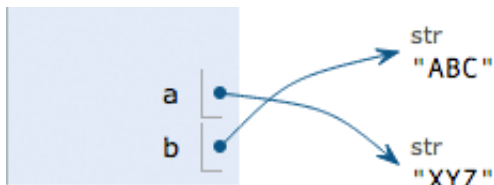
执行`a = 'ABC'`，解释器创建了字符串'ABC'和变量a，并把a指向'ABC'：



执行`b = a`，解释器创建了变量b，并把b指向a指向的字符串'ABC'：



执行`a = 'XYZ'`，解释器创建了字符串'XYZ'，并把a的指向改为'XYZ'，但b并没有更改：



所以，最后打印变量b的结果自然是'ABC'了。

常量

所谓常量就是不能变的变量，比如常用的数学常数 π 就是一个常量。在Python中，通常用全部

大写的变量名表示常量：

```
PI = 3.14159265359
```

但事实上PI仍然是一个变量，Python根本没有任何机制保证PI不会被改变，所以，用全部大写的变量名表示常量只是一个习惯上的用法，如果你一定要改变变量PI的值，也没人能拦住你。

最后解释一下整数的除法为什么也是精确的，可以试试：

```
>>> 10 / 3
3
```

你没有看错，整数除法永远是整数，即使除不尽。要做精确的除法，只需把其中一个整数换成浮点数做除法就可以：

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

因为整数除法只取结果的整数部分，所以Python还提供一个余数运算，可以得到两个整数相除的余数：

```
>>> 10 % 3
1
```

无论整数做除法还是取余数，结果永远是整数，所以，整数运算结果永远是精确的。

小结

Python支持多种数据类型，在计算机内部，可以把任何数据都看成一个“对象”，而变量就是在程序中用来指向这些数据对象的，对变量赋值就是把数据和变量给关联起来。

字符串和编码

4839次阅读

字符编码

我们已经讲过了，字符串也是一种数据类型，但是，字符串比较特殊的是还有一个编码问题。

因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），所以，一个字节能表示的最大的整数就是255（二进制11111111=十进制255），如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是65535，4个字节可以表示的最大整数是4294967295。

由于计算机是美国人发明的，因此，最早只有127个字母被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母z的编码是122。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了GB2312编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到Shift_JIS里，韩国把韩文编到Euc-kr里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。



因此，Unicode应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母A用ASCII编码是十进制的65，二进制的01000001；

字符0用ASCII编码是十进制的48，二进制的00110000，注意字符'0'和整数0是不同的；

汉字中已经超出了ASCII编码的范围，用Unicode编码是十进制的20013，二进制的01001110 00101101。

你可以猜测，如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

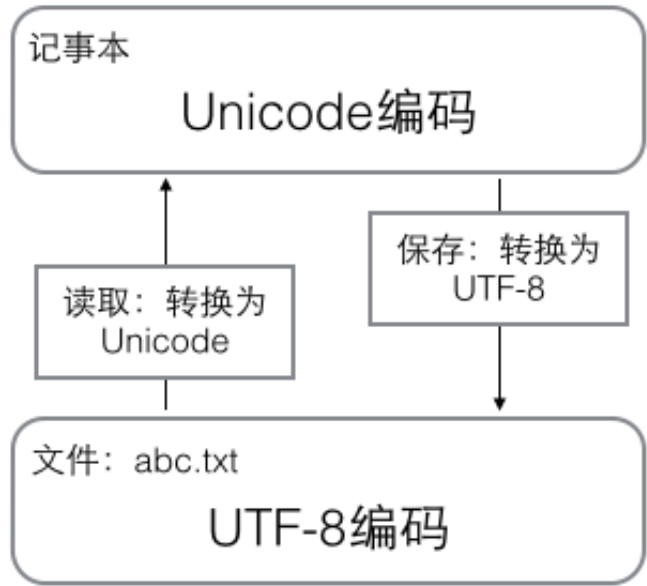
字符	ASCII	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中	x	01001110 00101101 11100100	10111000 10101101

从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

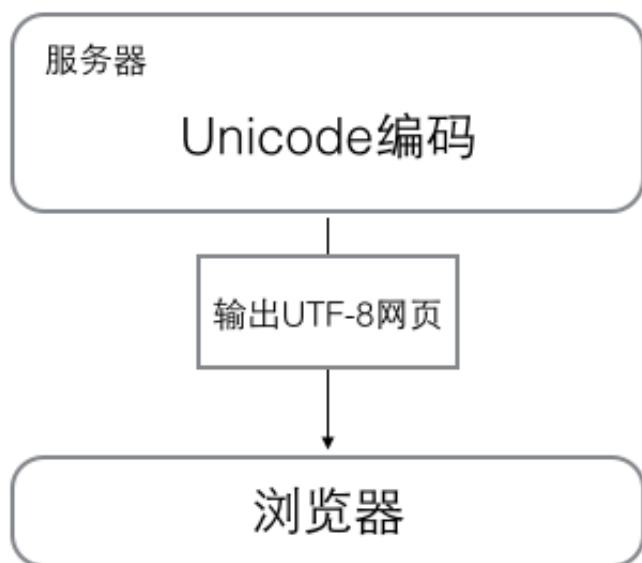
搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：



浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器：



所以你看很多网页的源码上会有类似`<meta charset="UTF-8" />`的信息，表示该网页正是用的UTF-8编码。

Python的字符串

搞清楚了令人头疼的字符编码问题后，我们再来研究Python对Unicode的支持。

因为Python的诞生比Unicode标准发布的时间还要早，所以最早的Python只支持ASCII编码，普通的字符串'ABC'在Python内部都是ASCII编码的。Python提供了`ord()`和`chr()`函数，可以把字母和对应的数字相互转换：

```
>>> ord('A')
65
>>> chr(65)
'A'
```

Python在后来添加了对Unicode的支持，以Unicode表示的字符串用`u'...'`表示，比如：

```
>>> print u'中文'
中文
>>> u'中'
u'\u4e2d'
```

写`u'中'`和`u'\u4e2d'`是一样的，`\u`后面是十六进制的Unicode码。因此，`u'A'`和`u'\u0041'`也是一样的。

两种字符串如何相互转换？字符串'xxx'虽然是ASCII编码，但也可以看成是UTF-8编码，而`u'xxx'`则只能是Unicode编码。

把`u'xxx'`转换为UTF-8编码的'xxx'用`encode('utf-8')`方法：

```
>>> u'ABC'.encode('utf-8')
'ABC'
>>> u'中文'.encode('utf-8')
'\xe4\xb8\xad\xe6\x96\x87'
```

英文字符转换后表示的UTF-8的值和Unicode值相等（但占用的存储空间不同），而中文字符转换后1个Unicode字符将变为3个UTF-8字符，你看到的`\xe4`就是其中一个字节，因为它的值是228，没有对应的字母可以显示，所以以十六进制显示字节的数值。`len()`函数可以返回字符串的长度：


```
>>> len(u'ABC')
3
>>> len('ABC')
3
>>> len(u'中文')
2
>>> len('\xe4\xb8\xad\xe6\x96\x87')
6
```

反过来，把UTF-8编码表示的字符串'xxx'转换为Unicode字符串u'xxx'用decode('utf-8')方法：

```
>>> 'abc'.decode('utf-8')
u'abc'
>>> '\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
u'\u4e2d\u6587'
>>> print '\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
中文
```

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

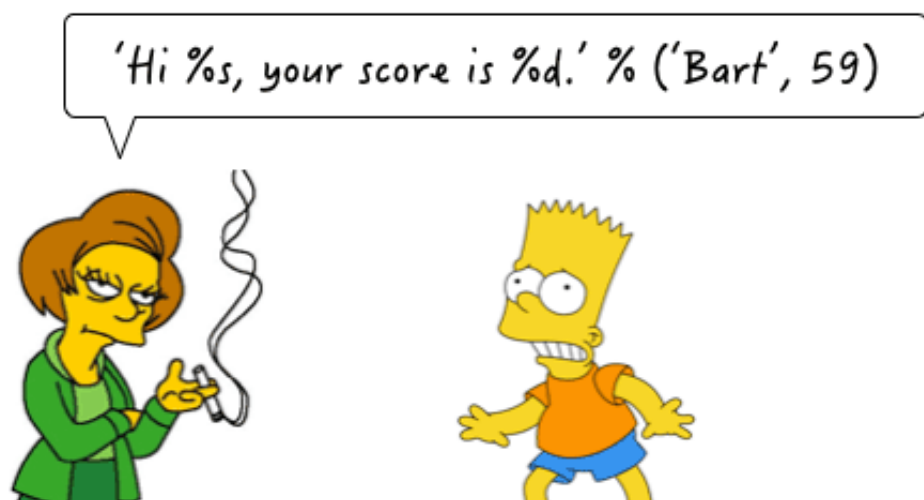
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释；

第二行注释是为了告诉Python解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。

格式化

最后一个常见的问题是如何输出格式化的字符串。我们经常会输出类似'亲爱的xxx你好！你xx月的话费是xx，余额是xx'之类的字符串，而xxx的内容都是根据变量变化的，所以，需要一种简便的格式化字符串的方式。



在Python中，采用的格式化方式和C语言是一致的，用%实现，举例如下：

```
>>> 'Hello, %s' % 'world'
'Hello, world'
>>> 'Hi, %s, you have $%d.' % ('Michael', 1000000)
'Hi, Michael, you have $1000000.'
```

你可能猜到了，%运算符就是用来格式化字符串的。在字符串内部，%s表示用字符串替换，%d表示用整数替换，有几个%?占位符，后面就跟几个变量或者值，顺序要对应好。如果只有一个%?，括号可以省略。

常见的占位符有：

```
%d 整数
%f 浮点数
%s 字符串
%x 十六进制整数
```

其中，格式化整数和浮点数还可以指定是否补0和整数与小数的位数：

```
>>> '%2d-%02d' % (3, 1)
' 3-01'
>>> '%.2f' % 3.1415926
'3.14'
```

如果你不太确定应该用什么，%s永远起作用，它会把任何数据类型转换为字符串：

```
>>> 'Age: %s. Gender: %s' % (25, True)
'Age: 25. Gender: True'
```

对于Unicode字符串，用法完全一样，但最好确保替换的字符串也是Unicode字符串：

```
>>> u'Hi, %s' % u'Michael'
u'Hi, Michael'
```

有些时候，字符串里面的%是一个普通字符怎么办？这个时候就需要转义，用%%来表示一个%：

```
>>> 'growth rate: %d %%' % 7
'growth rate: 7 %'
```

小结

由于历史遗留问题，Python 2.x版本虽然支持Unicode，但在语法上需要'xxx'和u'xxx'两种字符串表示方式。

Python当然也支持其他编码方式，比如把Unicode编码成GB2312：

```
>>> u'中文'.encode('gb2312')
'\xd6\xd0\xce\xca'
```

但这种方式纯属自找麻烦，如果没有特殊业务要求，请牢记仅使用Unicode和UTF-8这两种编码方式。

在Python 3.x版本中，把'xxx'和u'xxx'统一成Unicode编码，即写不写前缀u都是一样的，而以字节形式表示的字符串则必须加上b前缀：b'xxx'。

格式化字符串的时候，可以用Python的交互式命令行测试，方便快捷。
