《Python基础教程》(一)

公益分享，仅供参考

智普教育

www.jeapedu.com

Writen in LaTeX

2013.07.23

目 录

第 1 章. Python概述

Python是一种面向对象的解释性的计算机程序设计语言，也是一种功能强大而完善的通用型语言，已经具有十多年的发展历史，成熟且稳定。Python 具有脚本语言中最丰富和强大的类库，足以支持绝大多数日常应用。[1]

这种语言具有非常简捷而清晰的语法特点，适合完成各种高层任务，几乎可以在所有的操作系统中运行。

目前，基于这种语言的相关技术正在飞速的发展，用户数量急剧扩大，相关的资源非常多。

1.1 现代计算机系统简介

这里我们首先简要了解一下现代计算机的体系结构。现代计算机通常是由计算机**硬件**和**软件**两部分组成,计算机硬件设备用于执行程序指令，软件是计算机的系统核心，为运行应用程序提供环境和平台。

1.1.1 计算机硬件

一般的计算机的硬件系统通常包含以下几个基本部分：内部存储器(Memory)、包含有运算器和控制器的中央处理器(CPU)、输入设备(Input)、输出设备(Output) 等。

输入输出设备

输入、输出设备主要用于计算机运行程序和计算机使用者之间的交互，输入设备常见的有键盘和鼠标，而键盘又是常用于获取用户键盘输入信息。输出设备在一般的计算机系统上有显示器、打印机、声音等，通常显示器作为程序运行结果的输出的主要设备。

内部存储器

计算机的内存主要以电的方式存储数字(二进制)数据，分随机存储器和只读存储器。

中央处理器

中央处理器在计算机世界里称之CPU，主要由两部分组成：运算器和控制器。**运算器**又称算术逻辑单元，它是完成计算机对各种算术运算和逻辑运算的装置，能进行

[1] http://baike.soso.com/ Python

加、减、乘、除等数学运算，也能作比较、判断、查找、逻辑运算等。而**控制器**则是计算机的指挥中心,负责决定执行程序的顺序,给出执行指令时机器各部件需要的操作控制命令。由程序计数器、指令寄存器、指令译码器、时序产生器和操作控制器组成，它是发布命令的“决策机构”，即完成协调和指挥整个计算机系统的操作。[2]

1.1.2 计算机软件

软件是用户与硬件之间的接口界面。用户主要是通过软件与计算机进行交流。软件是计算机系统设计的重要依据。为了方便用户，为了使计算机系统具有较高的总体效用，在设计计算机系统时，必须通盘考虑软件与硬件的结合，以及用户的要求和软件的要求。[3]

计算机软件分类

计算机软件总体分为系统软件和应用软件两大类：系统软件是各类操作系统，如windows、Linux、UNIX 等，还包括操作系统的补丁程序及硬件驱动程序，都是系统软件类。应用软件可以细分的种类就更多了，如工具软件、游戏软件、管理软件等都属于应用软件类。

- **系统软件**, 系统软件是负责管理计算机系统中各种独立的硬件，使得它们可以协调工作。系统软件使得计算机使用者和其他软件将计算机当作一个整体而不需要顾及到底层每个硬件是如何工作的。一般来讲，系统软件包括操作系统和一系列基本的工具（比如编译器，数据库管理，存储器格式化，文件系统管理，用户身份验证，驱动管理，网络连接等方面的工具）。
- **应用软件**, 应用软件是为了某种特定的用途而被开发的软件。它可以是一个特定的程序，比如一个图像浏览器。也可以是一组功能联系紧密，可以互相协作的程序的集合，比如微软的Office软件。也可以是一个由众多独立程序组成的庞大的软件系统，比如数据库管理系统。

*1.2 Python*简介

Python是一种解释型、面向对象、动态数据类型的高级程序设计语言，属于应用层软件。自从20 世纪90 年代初Python语言诞生至今，它逐渐被广泛应用于处理系统管理任务、自动化运维、图像处理游戏和Web 网站开发等领域。Python已经成为最受欢迎的程序设计语言之一。[4]

由于Python语言的简洁、易读以及可扩展性，在国外用Python做科学计算的研究机构日益增多，一些知名大学已经采用Python教授程序设计课程。例如麻省理工学院的计算机科学及编程导论课程就使用Python语言讲授。众多开源的科学计算软件包都提供了Python 的调用接口，例如著名的计算机视觉库OpenCV、三维可视化库VTK、医学图像处理库ITK。而Python专用的科学计算扩展库就更多了，例如如下3 个十分经典的科学计算扩展库：NumPy、SciPy 和matplotlib，它们分别为Python提供了快速数组处理、数值运算以及绘图功能。因此Python语言及其众多的扩展库所构成的开发

[2]http://baike.baidu.com/ 控制器
[3]http://baike.baidu.com/ 计算机软件
[4]http://baike.baidu.com/ Python 简介

环境十分适合工程技术、科研人员处理实验数据、制作图表，甚至开发科学计算应用程序。

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”。

*1.2.1 Python*的历史

Python的创始人为Guido van Rossum。1989年圣诞节期间，在阿姆斯特丹，Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，做为ABC 语言的一种继承。之所以选中Python （大蟒蛇的意思）作为程序的名字，是因为他是一个Monty Python 的飞行马戏团的爱好者。

ABC是由Guido参加设计的一种教学语言。就Guido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，Guido 认为是由其非开放性使用原则所造成的。Guido 决心在Python 中避免这一错误（的确如此，Python 与其它的语言如C、C++和Java结合的非常好）。同时，他还想实现在ABC 中闪现过但未曾实现的东西。

就这样，Python就在Guido手中诞生了。实际上，Python 首先是在Mac机上实现的。可以说，Python 是从ABC发展起来，主要受到了Modula-3 （另一种相当优美且强大的语言，为小型团体所设计的）的影响。并且结合了Unix shell和C的习惯。成为一门为众多UNIX 和Linux开发者所青睐的开发语言。[5]

*1.2.2 Python*语言特色

可扩充性可以说是Python能作为一种编程语言的一大特色。扩展的新的模块（module）可以用C 或C++ 写成。而我们也可为现成的模块加上Python的接口。Python 可以使用户避免过分的语法的羁绊而将精力主要集中到所要实现的程序任务（逻辑）上。

Python也被称为是一门清晰的语言。因为它的作者在设计它的时候，总的指导思想是，对于一个特定的问题，只要有一种最好的方法来解决就好了。

Python语言是一种清晰的语言的另一个意思是，它的作者有意的设计限制性很强的语法，使得不好的编程习惯（例如if 语句的下一行不向右缩进）都不能通过编译。这样有意的强制程序员养成良好的编程习惯。其中很重要的一项就是Python的缩进规则。

*1.3 Python*前景

Python在编程领域的占有率一直处于稳步上升之中，根据最新的数据，Python排名第七。前六名分别是Java，C，VB，C++，PHP和Perl。随着微软将Python纳入.Net平台，相信Python的将来会更加强劲发展。Python 很可能会成为.Net平台快速开发的主流语言。欲了解这方面情况，请参考Iron Python的相关信息。

著名的搜索引擎Google 也大量使用Python。更加令人吃惊的是，在Nokia 智能手机所采用的Symbian操作系统上，Python同样也可以运行在Android手机操作系统上，

[5]http://baike.baidu.com/ Python简介

有很多Python爱好者通过Android平台来学习Python语言，无理由不相信Python将成为继C++,Java 之后的第三个编程语言！可见Python 的影响力之巨大。

*1.4 Python*的语言地位

通常认为，Python是一种解释性的语言，但是这种说法是不正确的，实际上Python在执行时，首先会将.py 文件中的源代码编译成Python的byte code（字节码），然后再由Python Virtual Machine来执行这些编译好的byte code。这种机制的基本思想跟Java，.NET是一致的。然而，Python Virtual Machine与Java 或.NET的Virtual Machine不同的是，Python的Virtual Machine 是一种更高级的Virtual Machine。这里的高级并不是通常意义上的高级，不是说Python 的Virtual Machine比Java 或.NET 的功能更强大，而是说和Java 或.NET相比，Python 的Virtual Machine距离真实机器的距离更远。或者可以这么说，Python的Virtual Machine 是一种抽象层次更高的Virtual Machine。

*1.5 Python*的应用领域

Python 不仅仅是一个设计优秀的程序语言，它能够完成现实中的各种任务，包括开发者们日复一日所做的事情。作为编制其他组件、实现独立程序的工具，它通常应用于各种领域。实际上，作为一种通用语言，Python 的应用角色几乎是无限的：你可以在任何场合应用Python, 从网站和游戏开发到机器人和航天飞机控制。

尽管如此，Python 的应用领域分为如下几类。下文将介绍一些Python 如今最常见的应用领域，以及每个应用领域内所用的一些工具。我们不会对各个工具进行深入探讨，如果你对这些话题感兴趣，请从Python 网站或其他一些资源中获取更多的信息。[6]

1.5.1 系统编程

Python 对操作系统服务的内置接口，使其成为编写可移植的维护操作系统的管理工具和部件（有时也被称为Shell 工具）的理想工具。Python 程序可以搜索文件和目录树，可以运行其他程序，用进程或线程进行并行处理等等。

Python 的标准库绑定了POSIX 以及其他常规操作系统（OS ）工具：环境变量、文件、套接字、管道、进程、多线程、正则表达式模式匹配、命令行参数、标准流接口、Shell 命令启动器、文件名扩展等。此外，很多Python 的系统工具设计时都考虑了其可移植性。例如，复制目录树的脚本无需做任何修改就可以在几乎所有的Python平台上运行。

1.5.2 用户图形接口

Python 的简洁以及快速的开发周期十分适合开发GUI 程序。Python 内置了TKinter的标准面向对象接口Tk GUI API，使Python 程序可以生成可移植的本地观感的GUI应用程序。Python/Tkinter GUI 不做任何改变就可以运行在微软Windows 、X Windows（UNIX 和Linux ）以及Mac OS（Classic 和OS X 都支持）等平台上。一个免费

[6]http://baike.baidu.com/ Python 应用

的扩展包PMW ，为Tkinter 工具包增加了一些高级部件。此外，基于C++ 平台的工具包wxPython GUI API 可以使用Python 构建可移植的GUI应用程序。

诸如PythonCard 和Dabo 等一些高级工具包均是构建在wxPython 和Tkinter 的基础API 之上的。对于运行于浏览器中的应用程序，Jython（Java 版本的Python ，我们将会在第2章中进行介绍）和Python 服务器端CGI 脚本提供了其他一些用户界面的选择。

1.5.3 数据库编程

对于传统的数据库需求，Python 提供了对所有主流关系数据库系统的接口，例如，Sybase 、Oracle 、Informix 、ODBC 、MySQL 、PostgreSQL 、SQLite等常用的数据库Python均有相应的接口函数库访问这些数据库。Python 定义了一种通过Python脚本存取SQL数据库系统的且可移植的数据库API 接口函数，这个API对于各种底层应用的数据库系统都是统一的。例如，因为厂商的接口实现为可移植的API ，所以一个写给自由软件MySQL数据库访问应用脚本在很大程度上不需改变就可以工作在其他数据库系统上（例如，Oracle），仅仅需要将底层的厂商接口替换掉就可以实现。

1.5.4 数值计算和科学计算编程

我们之前提到过的Python数值编程方面的扩展包NumPy包括很多高级工具，例如，矩阵对象、标准数学库的接口等。通过NumPy将Python 变成一个缜密严谨并简单易用的数值计算工具，其他一些数值计算工具为Python 提供了动画、3D 可视化、并行处理等功能的支持。

1.5.5 游戏、图像、人工智能等

Python可以利用pygame系统进行图像图形处理和游戏编程；用PIL和其他的一些工具进行图像处理；用PyRo 工具包进行机器人控制编程；用xml 库、xmlrpclib 模块和其他一些第三方扩展进行XML 解析；使用神经网络仿真器和专业的系统shell进行AI编程；使用NLTK 包进行自然语言分析：甚至可以使用PySol 程序下棋娱乐。可以从Vaults of Parnassus 以及新的PyPI 网站（请在Google 或http://www.python.org 上获得具体链接）找到这些领域的更多支持。

1.5.6 组件集成

在介绍Python 作为控制语言时，曾涉及它的组件集成的角色。Python 可以通过C/C++ 系统进行扩展，并能够嵌套C/C++ 系统的特性，使其能够作为一种灵活的粘合语言，脚本化处理其他系统和组件的行为。例如，将一个C 库集成到Python 中，能够利用Python 进行测试并调用库中的其他组件；将Python 嵌入到产品中，在不需要重新编译整个产品或分发源代码的情况下，能够进行产品的单独定制。

为了在脚本中使用，在Python 连接编译好组件时，例如，SWIG 和SIP 这样的代码生成工具可以让这部分工作自动完成。更大一些的框架，例如，Python 的微软Windows 所支持的COM 、基于Java 实现的Jython 、基于.NET 实现的IronPython 和

各种CORBA 工具包，提供了多种不同的脚本组件。例如，在Windows 中，Python 脚本可利用框架对微软Word 和Excel 文件进行脚本处理。

*1.5.7 Internet*脚本

Python 提供了标准Internet 模块，使Python 能够广泛地在多种网络任务中发挥作用，无论是在服务器端还是在客户端都是如此。脚本可以通过套接字进行通信；从发给服务器端的CGI 脚本的表单中解析信息；通过URL 获取网页；从获取的网页中解析HTML 和XML 文件；通过XML-RPC 、SOAP 和Telnet 通信等。Python 的库使这一切变得相当简单。

*1.6 Python*开发环境的安装与配置

Python是开源自由软件，所有Python的开发环境基本都是可以从网络上免费下载安装的，多数的Python 相关的软件和书籍均可在Python官网上找寻到，Python的官方网站是：*www.python.org*。使用Windows用户可以从http://www.python.org/getit/ 网页里下载适合自己使用计算机硬件平台版本的Python安装包。例如，如果您使用的计算机安装了WindowsXP 操作系统，计算机的CPU 是Intel的处理器，那你可以选择点击下载“Python 2.7.5 Windows x86 MSI Installer”到您计算机本地安装Python2.7.5，而使用Linux-Ubuntu操作系统的用户则不需要下载、安装Python开发环境，在安装完Ubuntu操作系统时Python开发环境一并安装完毕。

1.7 前辈寄语

读一下像ESR这样的超级电脑高手谈Python的话，你会感到十分有意思[7]:

- Eric S. Raymond是《The Cathedral and the Bazaar》的作者、“开放源码”一词的提出人。他说Python已经成为了他最喜爱的编程语言。这篇文章也是促使我第一次接触Python 的真正原动力。
- Bruce Eckel著名的《Thinking in Java》和《Thinking in C++》的作者。他说没有一种语言比得上Python使他的工作效率如此之高。同时他说Python可能是唯一一种旨在帮助程序员把事情弄得更加简单的语言。请阅读完整的采访以获得更详细的内容。
- Peter Norvig是著名的Lisp语言书籍的作者和Google公司的搜索质量主任（感谢Guido van Rossum告诉我这一点）。他说Python始终是Google的主要部分。事实上你看一下Google 招聘的网页就可以验证这一点。在那个网页上，Python 知识是对软件工程师的一个必需要求。
- Bruce Perens是OpenSource.org和UserLinux项目的一位共同创始人。UserLinux 旨在创造一个可以被多家发行商支持标准的Linux发行版。Python 击败了其它竞争对手如Perl 和Ruby 成为UserLinux支持的主要编程语言。

思考与练习

① 下载安装setuptools工具包，使用easy_install安装urllib2模块

[7] http://sebug.net/paper/python/ch01s04.html

```
https://pypi.python.org/pypi/setuptools
```

② 安装pip工具包，学习pip的search、install等功能

```
https://pypi.python.org/pypi/pip
```

③ 安装ipython工具包

```
https://pypi.python.org/pypi/ipython
```

④ 学习使用ipython notebook –pylib=inline

⑤ 配置Sublime及Notepad++

```
Notpad++: http://www.douban.com/note/311837575/
Sublime : http://www.douban.com/note/311834881/
```

第 2 章. Hello Python

2.1 初识Python语言

Python（英语发音：/'paɪθən/），是一种面向对象、直译式计算机程序设计语言，由Guido van Rossum 于1989年底发明，第一个公开发行版发行于1991 年。Py- thon 语法简捷而清晰，具有丰富和强大的类库。它常被昵称为胶水语言，它能够很轻松的把用其他语言制作的各种模块（尤其是C/C++）轻松地联结在一起。Python语言的发明者Guido Van Rossum 在90 年代初发明了Python 语言，Python是目前运行在现代计算机上的为数不多的几款高性能、通用的计算机编程语言，Python语言支持多种自由免费的工具包，可以运行在目前主流的操作系统和平台下，可以从http://www.python.org网站下载各类Python相关技术文档和软件。[1]

2.1.1 Python-Shell里运行Python语句

Python是交互式解释性语言，简单功能Python语句可以直接在Python自带的***Sh-ell***里直接运行。Python-Shell工具非常适合初学者在学习语法时使用。

进入Python-Shell， Windows 操作系统可以从开始菜单里找到并点击Python2.x.y下的IDLE(Python GUI)，而Linux用户则只需在**终端**里键入**Python**便可进入Python 的交互式**Shell** 环境。

退出Python-Shell， 退出Python-Shell 可以键入quit()或者Ctlr+d也可以退出Python-Shell。

在Python-Shell里依次键入下列语句，可以体验一下Python基本语法。

```
1 >>>x = 12
2 >>>y = 13
3 >>>z = x + y
4 >>>print z
5 25
6 >>>print 'hello the cruel world!'
7 hello the cruel world!
```

[1] http://baike.baidu.com/ Python

注1: ">>>"是***Python-Shell***下的命令**提示符**，无">>>"的行则表示上一条语句的执行结果输出。

注2: 等号'='前面的字母x、y、z我们在Python 里称之为**变量**，等号右边的表达式的值赋值给左边的变量。

现在我们暂且这样去理解上边的代码，代码第1行是创建一个变量x 并且赋值12，第2 行查看x变量的值，第3行是输出x的值，第4行代码则是创建变量y 并且被赋值13，第5行查看y变量的值，第6 行是输出y的值，第7行则是做了一个算术运算操作，将x和y 两个变量的值求和并将结果赋值给变量z，代码的第8行是打印变量z 的值，也就是说将x 加y 之和打印输出来，代码第9行是输出结果，代码第10行是打印字符串'hello the cruel world!'，第11 行结果输出在第七行。

我们可以在***Python-Shell***里直接键入x回车，y回车、z 回车直接看刚刚被赋值了的x、y以及z 变量的值。

```
1 >>>x
2 12
3 >>>y
4 13
5 >>>z
6 25
```

*2.1.2 IDLE-Editor*里编写*Python*程序

在Windows平台下安装了Python2.7.5以后，可以从**开始**菜单里找到**Python2.7** 文件夹，点击其下的**IDLE(Python GUI)** 后启动进入了***Python-Shell***，在IDLE(Python GUI) 的菜单栏上选择"File" → "New Windows"进入IDLE 的***Python-Editor***程序代码文本编辑器，在IDLE的***Python-Editor*** 编辑器里可以编写多行语句程序如下所示。

在源程序编写完后便可运行、测试程序。测试运行程序之前需要保存源代码，可通过菜单栏"File" → "save" 保存编写好的Python 程序为x.py，接着在IDLE 的***Editor***里按F5键或者选择***Editor*** 菜单"Run" → "Run Module" 或者直接按**F5**键来运行刚刚编写好的x.py程序。

```
1 x = 12
2 y = 13
3 z = x + y
4 print z
```

程序x.py运行结果如下:

```
25
```

2.1.3 IPython里运行Python语句

IPython 是一个python 的交互式shell，比默认的python shell 好用得多，支持变量自动补全，自动缩进，支持bash shell命令，内置了许多很有用的功能和函数。通过ipython 命令启动。

ubuntu下安装IPython 在ubuntu下只需要在终端里键入命令***sudo apt-get install ipython***就装好了。

Windows下安装IPython 运行→ cmd → easy_install ipython 即可安装IPython 软件。

2.2 Python程序基本组成

Python和其他高级语言一样，几乎都是首先从某些地方接收一些数据(如键盘或文件或者赋值)，接着对数据进行必要的处理，之后把处理的结果传到某个地方去(输出到文件、数据库或者打印到屏幕上)。

Python程序基本架构如下:

① 程序初始化部分
② 程序数据的输入部分
③ 程序数据的处理部分
④ 程序数据的输出部分
⑤ 程序结束部分

2.2.1 数据输出

在Python语言里可以通过print函数实现数据的输出操作，print 函数的语法结构如下所示。

print < *expressions* >

Python在执行print语句时，首先是计算一下print函数后边的expressions 表达式的值，之后再将表达式的值打印输出。

print < *expression* >,< *expression* >,..< *expression* >

我们注意到print语法结构里的< *expressions* >单词后边有s 的，复数，什么意思？其含义是表达式可以是多个，多个< *expression* >间用逗号间隔。

```
1 >>>print 'hello'
2 hello
3 >>>print 'world'
4 world
5 >>>print 'hello'+'world'
6 helloworld
7 >>>print 'hello','world'
8 hello world
```

语句1、3是分别打印'hello'和'world'两个字符串，我们可以看到每条print语句输出后都自动换了一行，第5行的语句用'+' 连接了两个字符串，打印输出时字母'o'和'w'是紧挨着的，而第7 行语句是用逗号','连接了两个字符串，我们发现在输出的时候字母'o'和'w' 并没有挨着而是中间有了空格，因此可以这样去理解print函数里的逗号，逗号在print语句里可以不输出回车键换行，使得下一次的print 数据和本次print 语句输出在同一行。

这里有了新的数据类型**字符串**，用引号将一些字符引起来，我们称之为这样的数据为字符串。在Python 里字符串的数据的定义有三种形式:

① 用单引号引起来一些字符

② 用双引号引起来的一些字符

③ 三个双引号引起来的字符都是字符串数据。

为何出现三个双引号字符串呢？这个是为了可在字符串数据里使用双引号。

```
1 >>>print 'hello'
2 hello
3 >>>print "world"
4 world
5 >>>print """He said "hello" to God"""
6 He said "hello" to God
```

请注意在said后边的hello字符串前后的双引号在输出时被打印出来了。那如何实现字符串和数字型数据同时输出呢?

逗号运算 我们可以采用逗号运算连接字符串和数字型数据，从而实现同时打印字符串和数字型数据，当然也可以用两条print语句。

```
1 >>> x = 12
2 >>> print 'hello x = ', x
3 hello x =  12
```

格式化控制字运算 在字符串里引用格式控制字%d可以实现打印整形数据，%f 则可以打印浮点型数据，而%s则是字符串数据。

```
1 >>> x = 12
2 >>> print 'hello x = ', x
3 hello x =  12
4 >>> y = 12.45
5 >>> s = "world"
6 >>> print "output %d %f %s"%(x, y ,s)
7 output 12 12.450000 world
8 >>>
```

这里需要注意的是代码的第6行，%前双引号引起来的字符串称之为print函数的格式化控制字，%后圆括号括起来的是格式化控制字里各个%所对应的值序列。

2.2.2 数据输入

在Python里可以通过***raw_input***函数从键盘获得用户的数据输入，其语法结构如下所示。

raw_input(< *prompt* >)

raw_input函数的形参prompt是一个字符串用于提示用户输入，当用户输入数据以后***raw_input***函数会把输入数据传给等号左边的变量来保存输入数据，raw_input 函数的返回值是字符串型的。

```
1 >>>name = raw_input("plz input your name:")
2 plz input your name:hello
3 >>>print name
4 hello
```

语句代码第1行使用了***raw_input***函数用于接收用户数据，从***raw_input*** 函数返回值赋值给了name变量，第2行输入了'hello' 字符串，第3 行调用print打印name变量的值。我们再看看下面的程序，任务是想从键盘输入两个整数给x和y，计算x+y之和并打印结果。

```
1 >>> x = raw_input("plz input x : ")
2 plz input x : 12
3 >>> y = raw_input("plz input y : ")
4 plz input y : 12
5 >>> z = x + y
6 >>> print z
7 '1213'
```

为何如此？为何不是结果25呢?我们再看一个程序好了。

```
1 >>> x = raw_input("plz input x : ")
2 plz input x : 12
3 >>>print x
4 '12'
5 >>> x = 12
6 >>>print x
7 12
```

第4行和第7行都是在执行'print x'以后打印的结果，两次打印结果有不同的地方么？第4 行的输出在12 的前后有单引号(由此可知raw_input返回值是字符串型的)，而第7 行输出结果12前后无单引号，由此可知***raw_input***函数的返回值是字符串，那我们的任务该如何完成呢？这里介绍一下类型转换函数***int*** 函数,***int*** 函数可以将纯数字字

符组成的字符串转换成整形数据，因此任务程序可以改成下面这个样子。

```
1 >>> x = int(raw_input("plz input x : "))
2 plz input x : 12
3 >>> y = int(raw_input("plz input y : "))
4 plz input y : 13
5 >>> z = x + y
6 >>> print z
7 25
```

语句第1行和第3行通过***raw_input***函数从键盘输入字符串数据，经int函数将数字字符串转换成整形数据之后赋值给左边的变量x 和y，此时x和y变量里就存储着整形数据12和13。

实际上，Python里有另外一个函数叫***input*** 也可以从键盘捕获数据，它的返回值直接就是数值数据，但键盘输入的必须是数(整形或者浮点型)才行。

```
1 >>> x = input("plz input x : ")
2 plz input x : 12
3 >>> y = input("plz input y : ")
4 plz input y : 13.5
5 >>> z = x + y
6 >>> print z
7 25.5
```

可以带有负号输入么？这里介绍***float***函数可以将带有点的和全数字的字符串转为浮点型数据。

```
1 >>> z = raw_input('plz input a float: ')
2 plz input a float: -12.45
3 >>> print z
4 '-12.45'
5 >>> w = float(z)
6 >>> print w
7 -12.45
```

我们在看看***input***函数的表现如何。

```
1 >>> x = input('plz input an int: ')
2 plz input an int: 1234
3 >>> print x
4 1234
5 >>> y = input('plz input a float: ')
6 plz input a float: -9.8765
7 >>> print y
```

```
8 -9.8765
```

2.3 Python-Shell反馈常见错误

初学者通常会使用***Python-Shell***来学习Python基础及语法知识，在使用***Python-Shell*** 时会遇到这样或者那样的错误，有的是语法错误，有的是键入的函数或者变量名字拼写错误，现就初学者常出现的错误做一个总结。

2.3.1 变量、函数未定义

下面我们简单总结一下在使用Python-Shell时常见的错误提示。

```
1 >>>len = 12
2 >>>le
3
4 Traceback (most recent call last):
5   File "<pyshell#36>", line 1, in <module>
6     le
7 NameError: name 'le' is not defined
8 >>>
```

上边Python-Shell反馈***NameError: name 'le' is not defined***，是说'le'变量未定义，的确如此，因为之前我们赋值的是len变量等于12，le没有赋值就没有被创建故报错没有被定义。

```
1 >>>len = "www.jeapedu.com"
2 >>> pint(len)
3
4 Traceback (most recent call last):
5   File "<pyshell#38>", line 1, in <module>
6     pint(len)
7 NameError: name 'pint' is not defined
8 >>>
```

从上边IDLE-Shell反馈***NameError: name 'pint' is not defined***,可以看出pint 函数没有定义，应该是函数print少了个r字母。

2.3.2 非语句字符

在python语句指令里放入了一些非语句的字符，怎么理解？比如在print函数前敲了一个(多个)空格或者按了TAB 键，都会导致在Python-Shell里运行语句时出现错误。

```
1 >>> s = 'www.jeapedu.com'
2 >>> print s
```

```
3  www.jeapedu.com
4  >>>     print s
5
6    File "<pyshell#45>", line 1
7      print s
8      ^
9  IndentationError: unexpected indent
10 >>>
```

第4行错误的原因在于，print函数前有一个TAB或者若干个空格，导致在Shell里语法不合规而报错误。

思考与练习

① 有多少种运行Python程序的方法？
② Python标准库所在的目录？
③ Dos退出Python-Shell的函数？
④ 编写脚本：打印姓名、学号、性别、数学成绩等信息
⑤ 利用 ** 运算求 8^5 的值
⑥ 利用raw_input输入12、34、56并求和(int函数可将字符串转为整形数据)

第 3 章. 变量与函数

3.1 变量基础

变量是编程语言里重要的基本概念，在各类高级语言里都用变量来代表一块内存区域，某一时刻这块区域里存储了 ‘a’ ，又一时刻可能又被存储成了’c’,正是由于该块内存里的值可以随时发生变化，我们称之这个代表内存区域的**符号**为变量。

3.1.1 变量本质

Python程序也是通过变量来访问某块内存里的数据，但Python里的变量的概念却和C语言里的变量有些不同，首先从变量的定义上来区分一下不同之处，在C语言里定义一个变量时，需要指定变量的数据类型，变量初始化时，还需依据等号左边变量的数据类型进行相应赋值，否则会出现数据的转换操作，造成不可想象的错误，而Python没有这方面的要求，在Python里定义变量不需要指定变量的数据类型，可以将各类数据直接赋值给等号左边的变量，比较自由，举例说明一下。

```
1 >>>x = 12
2 >>>print x
3 12
4 >>>x = 13.09
5 >>>print x
6 13.09
7 >>>x = "a1309b"
8 >>>print x
9 'a1309b'
```

从上边的程序可以看出赋值时Python不关心变量x的数据类型是什么，学习过C语言的开发者都知道，C规定变量的类型是为了分配一定大小的数据内存区域，而Python定义变量时不指定变量类型，那么对于较长的数据怎么能存储的下呢？下面我们再看一段小程序说明一下为何Python不需要指定变量的类型也可以”存储”下任意长度数据的。

```
1 >>> x = 12
2 >>> type(x)
3 <type 'int'>
```

```
>>> x = '12'
>>> type(x)
<type 'str'>
>>> x = 12.03
>>> type(x)
<type 'float'>
>>> x = '12.0300203000404'
>>> type(x)
<type 'str'>
```

注： type函数是返回括号里表达式值的数据类型

上边代码x被分别被赋值整数12，字符串 '12' ，浮点型数据12.03，但是都没有问题，都能正确执行，十分奇怪！如果数据很大能存下么(超出了4、8 字节)？我们看一下下面的程序，来分析一下Python的变量和C 语言的变量到底有哪些不同？

```
>>> x = 12
>>> id(x)
10416516
>>> x = 13
>>> id(x)
10416504
>>> x = '12'
>>> id(x)
19428056
>>> y = '12'
>>> id(y)
19428056
>>> y = 13
>>> id(y)
10416504
>>> y = 12
>>> id(y)
10416516
```

注： id函数是返回括号里Object在内存里的地址，读者请自行用id输出值作为内存地址，等号右边的值作为内存里的存储内容来画一个内存分布图分析一下。

10416516和10416516、10416504出现多次。

```
>>> help(id)
Help on built-in function id in module __builtin__:
id(...)
    id(object) -> integer
```

```
5       Return the identity of an object.  This is
            guaranteed to be unique among
6       simultaneously existing objects.  (Hint: it's the
            object's memory address.)
7   >>>
```

当x和y被赋值12的时候id返回值相同，当x和y被赋值13的时候id 返回值也相同，当x 和y被赋值'12' 的时候id的返回值也相同；但x 或者y被赋值12、13、'12'的候id的返回值不同！由此可见Python是在给等号右边的数据分配内存空间，可以通过不同的变量来指向这个数据。也可通过不同变量可以指向相同的内存空间获得相同的值。

```
1   >>> x = 12
2   >>> y = 13
3   >>> x = y
4   >>> x
5   13
6   >>> id(x)
7   10416504
8   >>> id(y)
9   10416504
10  >>>
```

可以通过变量来改变它正指向的这块内存空间里的数据么？不可以！

```
1   >>> x = 12
2   >>> id(x)
3   10416516
4   >>> x = 15
5   >>> id(x)
6   10416480
```

由上边结果可见，并未改变12所在内存区里的值，而是又分配了一个内存空间给数据15，也就是说变量在被重新赋值以后指向的是另外内存一个地方，和C语言变量的区别在于C里的变量一直指向某地，C语言可以通过变量来改变它指向内存里的值，而Python 里变量变化的是它"指向"，可以直接认为Python里的变量等价于C 语言里的指针概念。因此C和Python里都有变量的概念，但两个语言的变量的“变”的含义不同！

3.1.2 局部变量

要谈局部变量和全局变量的技术前提是对函数有一定的理解，我们先在这里简单说明一下Python的函数，Python 有自带的函数也可以使用第三方工具包了外部引用函数，有的时候用户自己可以自主开发一些特定功能的函数我们称之为自定义函数。用户怎么才能自己定义函数呢？Python 规定的自定义函数语法结构如下:

```
1 def function_name(parameters):
2 (TAB)statement1
3 (TAB)statement2
4 (TAB)statement3
5 (TAB)etc.
```

第1行是定义函数名及函数的参数，请注意函数右括号后有一个冒号！第2-5 行是函数语句块。需要注意的是函数名下的每条语句前都要用TAB键缩进一下，否则会认为是非本函数的语句块里的语句，而是与函数同级的程序的某条语句。Python函数体不用花括号将语句块括起来而是用TAB来区分语句是函数里的还是不是函数的。函数无返回值类型，行参也无数据类型说明。

自定义函数可以像C语言一样在其他地方被其他程序或语句调用，调用自定函数时不需要使用def这个关键字。我们举例说明一下。

```
1  #define function: add
2  def add(x, y):
3      z = x + y
4      return z
5  #define main function
6  def main():
7      a = 12
8      b = 13
9      #function 'add' called
10     c = add(a, b)
11     print c
12 #programme entry
13 main()
14 print 'End!'
```

程序代码从第13行开始执行，从第2行到第11行分别定义了add和main两个函数。add函数有两个形参x 和y，而main 函数无形参也无返回值。程序从第13行开始执行然后跳转到自定义main函数的的第7 行、8行和第10行，当程序执行到第10行的时候又跳转到add函数的第3 行、第4行将和返回给调用函数main，当add 执行完以后返回到调用函数main函数，同时会把返回值赋值给第10行的等号左边的c 这个变量，接下来执行第11行，打印求和结果，执行完11 行后main被全部执行完毕返回到第13行的下一行第14 行打印出字符串'End!',至此整个程序结束。

有了函数的基本概念以后，我们再来聊聊Python变量的局部变量问题。顾名思义局部变量肯定生活在某个局部地方，那什么叫局部变量呢？定义在函数体内的变量叫局部变量，因此我们可以很快得到全局变量的定义，那就是定义在函数体外的变量时全局变量。局部变量只可被本函数使用，全局变量可以被所有语句访问使用。

```
1 def f1():
```

```
2       x = 12
3       print x
4   def f2():
5       y = 13
6       print y
7   def f3():
8       print x
9       print y
10  def main():
11      f1()
12      f2()
13      %f3()
14  main()
15  print 'End!'
```

此时第13行代码是被注释掉的，如果打开第13行会报错如下:

```
1  >>>
2  12
3  13
4
5  Traceback (most recent call last):
6    File "F:/Python27/t2.py", line 14, in <module>
7      main()
8    File "F:/Python27/t2.py", line 13, in main
9      f3()
10   File "F:/Python27/t2.py", line 8, in f3
11     print x
12 NameError: global name 'x' is not defined
13 >>>
```

错误是说main函数在调用函数f3时发现f3函数某一行有问题(的第8行)"print x" 里的x 没有定义。我猜设计者是想打印f1定义的x 想打印出12，对么？但x定义在自定义函数f1 里，x此时是局部变量不能被除f1 函数之外的函数或者语句使用，故报错。由于局部变量定义在函数体里，故多个函数可以在各自函数体内使用相同的变量名作为自己的变量。

3.1.3 全局变量

全局变量没有定义在任何函数体内，或者可以这么说和main函数属于同一级，全局变量可以被自定义的函数访问使用，如果函数想读去某个全局变量没有特别的要求，直接使用即可。但是如果某函数想在函数里修改某个全局变量，Python语言要求在修改前用global 语句声明一下这个变量是全局的才能修改这个全局变量。

```
1  def printLocalx():
2      x = 12
3      print 'f1 local x = ',
4      print x
5  def printLocaly():
6      y = 13
7      print 'f2 local y = ',
8      print y
9  def readGlobal():
10     print 'f3 read global x = ',
11     print x
12     print 'f3 read global y = ',
13     print y
14 def modifyGlobal():
15     global x
16     print 'f4 write x = -1'
17     x = -1
18 def main():
19     printLocalx()
20     printLocaly()
21     readGlobal()
22     modifyGlobal()
23 x = 200
24 y = 100
25 main()
26 print 'after modified global x = ',
27 print x
28 print 'End!'
```

此时由于程序定义了全局变量x和y，故readGlobal函数的两条打印语句是不会报语法错误的。全局变量可被任何子函数访问使用，在readGlobal函数里(第11行和第13行)读x和y变量的值，没有问题。而在modifyGlobal函数体里(第17行)将x赋值为-1，如果自定义函数体修改全局变量，首先要像第15行那样声明一下这个x是全局的变量，只有这样才能修改全局的x变量。如果去掉第15和16行，第17 行的x将被看做modifyGlobal 函数的局部变量。

程序执行结果如下所示:

```
1  f1 local x =  12
2  f2 local y =  13
3  f3 read global x =  200
4  f3 read global y =  200
```

```
5 f4 write x = -1
6 after f4 modified global x =  -1
7 End!
```

3.2 函数的形参与实参

在函数定义时的变量称作函数的**形参**，形参主要是函数接收函数外部值传入函数体内去处理，是函数和外部程序或者语句的接口。函数调用时的变量称之为**实参**。现在我们举个例子来说明一下形参和实参。

```
1  #define fun: multi
2  def multi(x, y):
3      z = x * y
4      return z
5  #define main function
6  def main():
7      a = 12
8      b = 13
9      #function 'add' called
10     c = multi(a, b)
11     print c
12 #programme entry
13 main()
14 print 'End!'
```

定义在main函数里的a和b是main函数的两个局部变量，第10行main调用了multi函数，变量a和b被放在multi函数定义时的x和y的位置上了，那么a的值就传给了x，b变量的值就传给了y变量，函数调用时括号里的a和b变量(第10行)我们称之为函数调用时的实参变量，而函数定义(第2行)里的x 和y我们称之形参。

思考与练习

① Python下如何定义函数及变量？与C语言定义函数和变量的主要区别？
② 在修改全局变量值时需要注意什么？
③ 如何查sqrt函数的帮助文档？
④ 赋值语句x, y, z = 1, 2, 3结果是？
⑤ 编写求两数的max和min函数，要求函数带形参。
⑥ 编写求两数之和函数sum，要求函数带形参。
⑦ 基于递归编写求和函数sum,要求函数带形参。

第 4 章. 表达式与分支控制语句if

4.1 表达式

表达式，是由数字、算符、数字分组符号括号、自由变量和约束变量等以能求得数值的有意义排列方法所得的组合,故表示通常是由操作数和操作符两部分组成，如果操作符前后均有操作数，我们称此类操作符是双目运算符，例如加法、减法、取模、赋值运算等运算符均是双面运算符。如果操作符要么前边有操作数，要么后边有操作数，我们称之这类操作符为单目运算符，例如C 语言里的++、–以及取负运算均属于单目运算符，而Python 里的单目运算符比较少。

4.1.1 算术表达式

常见的算术运表达式由加减乘除、取模取余、取负以及幂次方(**)等运算符组成。

```
1  >>> x = 12
2  >>> y = 13
3  >>> z = 2
4  >>> su = x + y
5  >>> sm = x - y
6  >>> sc = x * y
7  >>> sd = x / y
8  >>> sq = x % y
9  >>> sf = -x
10 >>> xz = x ** z
11 >>> print su,sm,sc,sd,sq,sf,xz
12 25 -1 156 0 12 -12 144
```

算术运算比较简单，上边语句里第10行代码需要注意一下幂次方(**)运算符，x(**)y的意思是x^y。

4.1.2 关系表达式

关系表达式实际上是一种布尔表达式，简单的布尔表达式只有True(1)和False(0)两个值，稍微复杂一点的布尔表达式是由大于、小于、等于等比较运算符组成的表达式，表达式的运算结果也是只有True(1)和False(0)两个值。用于构建布尔表达式的比较运算符有：大于>、小于<、等于==、大于等于>=、小于等于<=、不等于!= 等。

```
>>>4 == 4
True
>>>4 != 4
False
>>>4 < 5
True
>>>4 >= 3
True
>>>"A" < "B"
True
```

4.1.3 逻辑表达式

用逻辑运算符**and**、**or**和**not**可以将若干个表达式组合成一个更加复杂的布尔表达式，逻辑与**and**的意思是**and**前后的表达式都为真的情况下这个复杂的布尔表示的结果才为真；而逻辑或**or** 的意思则是，**or**前后有一个为真则整体为真；逻辑非**not**的作用则是如果后边的表达式值为真，则结果为假，如果**not**后边的表达式运算结果为假则结果为真。

```
>>> A = True
>>> B = False
>>> A and B
False
>>> A or B
True
>>>not A
False
>>> A and (not B)
True
```

4.2 if分支语句

分支语句的作用是在某些条件控制下有选择的执行实现一定功能语句块。if 分支语句则是当if后的条件满足时，if 下的语句块被执行，语法格式如下所示:

```
if <condition>:
    statements
```

让我们看看代码吧。

```
>>> sex = 'male'
>>> if sex == 'male':
```

```
3       print 'Man!'
4  #此处有两次回车键
5
6  Man!
7  >>> if sex == 'female':
8       print 'Woman'
9  #此处有两次回车键
10
11 >>>
```

4.3 if_else语句

if语句下的语句块是在<***condition***>条件满足时执行，***else*** 语句下的语句块则是在<***condition***>条件不满足的情况下执行，使用***if_else*** 语句需要注意的是if的¡condition¿判定条件后有冒号，***else*** 语句后无<***condition***>判定表达式，但有冒号。if 和else下的语句块不用左右花括号。

```
if <condition>:
    statements
else:
    statements
```

举个例子来说明一下if_else的使用。

```
1  sex = raw_input("plz input your sex: ")
2  if sex == 'male' or sex == 'm':
3          print 'Man!'
4  else:
5          print 'Woman!'
```

程序运行结果如下:

```
plz input your sex: female
Woman
```

做个小练习，输入数学成绩(整形)，0~60打印"No Pass!"，60~70打印"Just Pass！"，70~80 打印"Good"，70~80打印"Wonderful!"，80~90打印"Excellent!"，90~100打印"Best!"，请用***if_else***嵌套来完成。

```
1  #coding:utf-8
2  x = input("plz input math record: ")
3  if x >= 60:
4      if x >= 70:
5          if x >= 80:
```

```
6               if x >= 90:
7                   print "Best!"
8               else:
9                   print "Excellent!"
10          else:
11              print "Good!"
12      else:
13          print "Just ! Pass"
14  else:
15      print "No Pass!"
```

程序运行结果如下

```
>>> ========= RESTART =========
plz input math record: 27
No Pass!
>>> ========= RESTART =========
plz input math record: 67
Just ! Pass
>>> ========= RESTART =========
plz input math record: 77
Good!
>>> ========= RESTART =========
plz input math record: 80
Excellent!
>>> ========= RESTART =========
plz input math record: 98
Best!
>>>
```

4.4 if_elif_else多分支语句

上边的程序如果写不好，很有可能无法完成对成绩的分类打印，诸如用**if_else** 嵌套完成的程序可以用**if_elif_elif_.....elif_else**结构来完成,其语法结构如下所示:

```
if <condition1>:
    statements_1
elif <condition2>:
    statements_2
elif <condition3>:
    statements_3
...
```

```
...
...
elif <conditionN>:
    statements_n
else:
    statements_else
```

这种结构称之为多分支结构，从上***if*** 至下***elif*** 逐一检查判定条件表达式上<***conditionX***>，看那个条件满足就执行其下的语句块上***statements_X***，所有条件均不满足才执行***else*** 下的语句块***statements_else***。整个结构只有一个语句块被执行。由此上一小节的分类打印成绩的程序可以改成下面这个样子了。

```
1  if x >= 90:
2      print "Best!"
3  elif x >= 80:
4      print "Excellent!"
5  elif x >= 70:
6      print "Good!"
7  elif x >= 60:
8      print "Just ! Pass"
9  else:
10     print "No Pass!"
```

小练习：写一个完备的判断性别的程序。(键入'male','Male','m','M','man','Man' 均代表男性)

第 5 章. while循环体

5.1 while基础语法

在程序设计时，经常会遇到一些语句被不断的重复执行，这样的代码即长又低效，可以考虑将重复执行的语句用循环体来实现。***Python***下的循环体有两种结构，一是***while*** 循环体，一是***for*** 循环体。***while*** 循环体常用于多次重复运算，而***for*** 循环体常用于遍历序列型数据体，这个之后再说。我们先看看***while*** 循环体的语法结构。

```
while <condition>:
    statements
else:
    statements
```

通常在while上有一个循环体变量用于***while*** 条件判断，***while*** 结束前有个修正循环体变量的操作，通过循环体变量的修正来控制循环次数。举例如何打印某字符串10遍？说明如下:

```
1 i = 0
2 while i <= 10:
3     print "hello jeapedu.com"
4     i = i + 1
```

while之上的i = 0是对循环体变量i 进行初始化，i = i + 1 是对循环体变量进行修正，通过i <= 10 这个条件控制就可以打印”hello jeapedu.com” 这个字符串10次了。

小练习：如何求1~100之和？

```
1 i = 1
2 s = 0
3 while i <= 100:
4     s = s + i
5     i = i + 1
6 print "s = %d"%(s)
```

思考题：打印出1~100内的素数[1]

[1]http://baike.baidu.com/ 搜索”素数”

5.2 while体的退出

有的时候可能不希望**while**完全执行完n次，当某条件满足时退出**while**循环体，这时候可以在**while**循环体里使用if语句来设定退出条件，用**break**语句退出**while** 循环体。

小程序：猜数练习，随机产生一个1~100以内的数，用户来猜，猜中结束。

```
1  import random
2  r = random.randint(1, 100)
3  i = 0
4  while i < 100:
5      g = input("guess ")
6      if g == r:
7          break
8      elif g > r:
9          print 'plz smaller ! than %d'%(g)
10     else:
11         print 'plz bigger ! than %d'%(g)
12 print "Bingo!"
```

5.3 continue语句

while循环体语句体里出现**continue**执行时，本次循环**continue** 之下的语句不被执行，直接进入下一次循环。

思考与练习

① 求素数，设计成函数，并调用这个函数。
② 编写N的阶乘函数。
③ 编写打印整数二进制、八进制、十六进制函数
④ 因式分解，输入20得到2、2、5
⑤ 利用 ** 运算求 8^5 的值
⑥ 利用raw_input输入12、34、56并求和(int函数可将字符串转为整形数据)

第 6 章. 字符串

6.1 字符串基础

用引号引起来的字符集合称之为字符串，这里的引号可以是一对单引号、双引号、三引号(单、双)。单引号和双引号字符串基本没啥区别，特殊的是用三引号引起来的字符串为何？举个例子来说明一下：比如打算输出Hello ‘Jack’ ...这个字符串该如何用print 打印输出呢？很简单！

```
1 s = "hello 'jack'..."
2 print s
```

输出结果如下:

```
hello 'jack'...
```

要输出Hello "Jack"...能直接把s里的单引号改成双引号么？

```
1 s = "hello "jack"..."
2 print s
```

不行Python会报错的!为何错了？原因在于Python在处理s 等号右边时会从左至右找到与h 前边的左双引号匹配的右双引号，两个双引号间的字符集为字符串，当找到j前的双引号结束，那余下的jack “... 该如何理解呢？Python 理解不了就报错了！

咋办呢？解决办法有二:

(1)在jack字符串的j前边和k后边加上\号就行,我们称之这种表达方式为转义字符，诸如C 语言里的\n和\t 均是转义字符。Python 会解析双引号里的各个字符，如遇\ 会将\和其后边的字符组合成转义字符去理解的(最终按一字符长度处理)。

```
1 s = "hello \"jack\"..."
2 print s
```

输出结果如下:

```
hello "jack"...
```

(2)用三引号引起来的字符串Python是不关心字符串里的双引号和单引号的，直接输出"'hello "jack"..."'。

```
1 s = '''hello "jack"...'''
2 print s
3 s = """hello "jack"..."""
4 print s
```

输出结果如下:

```
hello "jack"...
hello "jack"...
```

一般单引号里可以有双引号，双引号里可以有单引号，三引号里可以有单引号和双引号。

```
1 print 'hello "jeapedu.com" !'
2 print 'hello """jeapedu.com""" !'
3
4 print "hello 'jeapedu.com' !"
5 print "hello '''jeapedu.com''' !"
6
7 print '''hello 'jeapedu.com' !'''
8 print '''hello "jeapedu.com" !'''
9 print '''hello """jeapedu.com""" !'''
10
11 print """hello 'jeapedu.com' !"""
12 print """hello '''jeapedu.com''' !"""
13 print """hello "jeapedu.com" !"""
```

以下会出错

```
1 print "hello """jeapedu.com""" !"
2 print "hello "jeapedu.com" !"
3 print 'hello 'jeapedu.com' !'
```

太复杂了，怎么办呢？这样去理解好了，Python解释器从左向右找到第一个匹配的引号，如果找到的这个匹配在最后，解释器判定这个字符串合法，如果后边还有字符判定字符串语法错误。”hello """jeapedu.com""" !”这个字符串Python解释器找到o后边的双引号即认为“hello ”为打印输出的字符串，而之后的""jeapedu.com""" !”就不知道如何理解了，便语法报错。

6.2 几种特殊字符串

6.2.1 转义字符串

上边程序中语句print ”hello ”jeapedu.com” !”会报错，原因在于Python 解释器在处理字母j前的双引号时把这个双引号当作字符串尾部的双引号了，程序设计者是想输

出字符串j前有一个双引号，那有办法实现么？这里可在j前边的双引号前加上\符号，这是Python语句解释器就不会再报错了，原因在于：Python发现字符串里有\ 会将\和后边的字符组合成一个一字节字符来输出。如果\ 后是双引号，则Python解释器会处理成字符"，输出，这个操作称之为转义。'\+ **字母**'称之为转义字符。

```
1 print "hello \"\"\"jeapedu.com\"\"\" !"
2 print "hello \"jeapedu.com\" !"
3 print 'hello \'jeapedu.com\' !'
```

常见的转义字符有:

\n 回车换行
\t 下一制表位
\" 双引号
\' 单引号
\\ 输出斜杠

6.2.2 原字符串

如果字符串本身就是有\的，输出打印也要求\，或者说字符串里的\ 不和其后的字符被看成转义字符处理。那么需要在字符串前加上字母r，此时的字符串称之为原字符串，特点是原样打印，\不被处理成转义字符。

```
1 print "c:\temp\node\jeapedu.py"
2 print r"c:\temp\node\jeapedu.py"
```

看看结果就明白了，呵呵。

```
c:  emp
ode\jeapedu.py
c:\temp\node\jeapedu.py
```

6.3 字符串的访问

*6.3.1 index*索引

字符串是Python下一种序列型的数据类型，字符串里的每个字符都有一个标号可以标识其在字符串里的位置，从左至右依次是$0....,n-1$, 从右至左依次是$-1,-2,....-n$(n 为字符串的长度)。因此就有方法可以随意访问字符串里的某个字符了。

```
1 s = "www.jeapedu.com"
2 print s[4]
```

6.3.2 slice切片

也可描述一定的范围来访问字符串里的子串，可以通过切片操作来完成访问子串，“切片”顾名思义，要切两刀，start是切的第一刀的位置，end是切的第二刀的位置，切片访问子串的语法如下:

```
str_name[start : end : step]
```

start是访问字串的起点，end为终点，step为步长，得到的子串由start到end-1这些字符组成。

```
1 s = "www.jeapedu.com"
2 print s[4 : 11]
```

打印结果为:

```
jeapedu
```

start不指定则从字符串第一个字符开始；end不指定，则默认到字符串结束；step不指定默认为+1。

正切片， step为正数，方向从左至右。

① 不指定start

```
1 s = "www.jeapedu.com"
2 print s[ : 11]
```

打印结果为:

```
www.jeapedu
```

② 不指定end

```
1 s = "www.jeapedu.com"
2 print s[4 : ]
```

打印结果为:

```
jeapedu.com
```

③ start和end都不指定

```
1 s = "www.jeapedu.com"
2 print s[ : ]
```

打印结果为:

```
www.jeapedu.com
```

④ step不为1

```
1 s = "www.jeapedu.com"
2 print s[ : : 2]
```

打印结果为:

```
wwjaeucm
```

负切片， step为负数，方向从右至左。

① start正，end负，step负

```
1 s = "www.jeapedu.com"
2 print s[ 10:-12:-1]
```

打印结果为:

```
udepaej
```

请分析结果为何如此?

② start负，end负，step负

```
1 s = "www.jeapedu.com"
2 print s[ -5:-12:-1]
```

打印结果为:

```
udepaej
```

请分析结果为何如此?

③ start和end不指定，step为-1

```
1 s = "www.jeapedu.com"
2 print s[ ::-1]
```

打印结果为:

```
moc.udepaej.www
```

6.4 字符串运算

6.4.1 字符串加法

两个字符串做加法操作是将两个字符串连接在一起，请对比print 函数里两个字符串逗号连接的区别。

```
1 s = "www.jeapedu.com"
2 s1 = "python education"
```

```
3 print s, s1
4 print s + s1
```

程序运行结果如下所示:

```
www.jeapedu.com python education
www.jeapedu.compython education
```

请注意m和p之间有无空格，如果是逗号连接s和s1则有空格，如果是用字符串加法连接两个字符串则无空格，这样可以构造出一个更长的字符串了。

6.4.2 字符串乘法

字符串的乘法运算相当于同一字符串n次相加，举例说明如下:

```
1 s = "www.jeapedu.com"
2 s1 = s * 3
3 print s1
```

程序运行结果如下所示:

```
www.jeapedu.comwww.jeapedu.comwww.jeapedu.com
```

6.4.3 字符串*in*运算

可以判断一个字符或者子串是否在某个字符串里，如果字符串里含有这个字符或者子串，in运算结果为真，否则为假。

```
1 s = "www.jeapedu.com"
2 print 'j' in s
3 print 'jeapedu' in s
4 print 'a1' in s
```

程序运行结果如下所示:

```
True
True
False
```

6.4.4 字符串*not in*运算

与in运算意思基本一致，判断单个字符或者子串是否不在字符串里。

```
1 s = "www.jeapedu.com"
2 print 'j' not in s
3 print 'jeapedu' not in s
4 print 'a1' not in s
```

程序运行结果如下所示:

```
False
False
True
```

6.5 字符串函数

和其他高级语言一样，Python也提供大量的字符串函数用于字符串的操作和处理，常用的字符串处理函数有:

函数名	功能
s.index(sub[,start[,end]])	定位子串sub在s里第一次出现的位置，找不到会报错
s.find(sub[,start[,end]])	与index函数一样,但如果找不到会返回-1
s.replace(old, new[,count])	替换s里所有old子串为new子串,count 指定多少个可被替换
s.count(sub[, start[, end]])	统计s里有多少个sub 子串
s.split([sep [,maxsplit]])	用sep分割s字符串,默认空格分隔,返回由单词组成的列表

举个例子来展示一下这些函数的用法.

```
1 s = "www.jeapedu.com" * 5
2 n = s.count("jeapedu")
3 i = 0
4 pos = -1
5 while  i < n:
6   pos = s.index("jeapedu", pos + 1)
7   print 'find \"jeapedu\" in s: ',pos
8   i += 1
```

程序运行结果如下所示:

```
find "jeapedu" in s:  4
find "jeapedu" in s:  19
find "jeapedu" in s:  34
find "jeapedu" in s:  49
find "jeapedu" in s:  64
```

为何index能查出所有的"jeapedu"而没有报错？因为count函数统计出了字符串里有多少个，故不会多查找一次而报错的。如果将i ¡ n改成i ¡= n就会多循环查找一次而报错。

```
1 s = "www.jeapedu.com" * 5
2 n = s.count("jeapedu")
3 i = 0
4 pos = -1
```

```
5 while  i <= n:
6   pos = s.index("jeapedu", pos + 1)
7   print 'find \"jeapedu\" in s: ',pos
8   i += 1
```

程序运行结果如下所示:

```
find "jeapedu" in s:  4
find "jeapedu" in s:  19
find "jeapedu" in s:  34
find "jeapedu" in s:  49
find "jeapedu" in s:  64

Traceback (most recent call last):
  File "C:/Python27/code/mytest.py", line 17, in <
     module>
    pos = s.index("jeapedu", pos + 1)
ValueError: substring not found
```

下面是replace函数的使用示例。

```
1 s = "www.jeapedu.com" * 3
2 print s
3 s  = s.replace('j', 'J', 2)
4 print s
```

程序运行结果如下所示:

```
www.jeapedu.comwww.jeapedu.comwww.jeapedu.com
www.Jeapedu.comwww.Jeapedu.comwww.jeapedu.com
```

6.6 字符串基础练习

6.6.1 输出字符串

① 打印 '$x = 12$'

```
1 s = "x = 12"
2 print s
```

或者字符串逗号连接整形

```
1 x = 12
2 print "x =", x
```

或则用字符串加法

```
1 x = "12"
2 s = "x = " + x
3 print s
```

或则用字符串格式化输出

```
1 x = 12
2 print "x = %d"%(x)
```

② 打印 ‘***www.jeapedu.comwww.jeapedu.com.....www.jeapedu.com***’
其中 ‘***www.jeapedu.com***’重复一百次。

```
1 s = "www.jeapedu.com"
2 w = ""
3 i = 0
4 while i < 100:
5     w += s
6     i += 1
7 print w
```

或者字符串乘法

```
1 s = "www.jeapedu.com" * 100
2 print s
```

③ 打印从
”http://www.baidu.com/s?wd=latex&pn=10&tn=baiduhome_pg&ie=utf-8&usm=1”
到
”http://www.baidu.com/s?wd=latex&pn=300&tn=baiduhome_pg&ie=utf-8&usm=1”
变化的是pn后的数字从10~300。

```
1 x = 10
2 while x <= 300:
3     x = x + 10
4     sb = "http://www.baidu.com/s?wd=latex&pn="+str(
          x)+"&tn=baiduhome_pg&ie=utf-8&usm=1"
5     print sb
```

6.6.2 字符串输入

① input函数输入数字***123***

```
1 a = input("plz input your age:")
2 print a
```

② raw_input函数输入字符串 ***'jeapedu.com'***

```
1 name = raw_input("plz input your name:")
2 print name
```

③ raw_input函数输入数字型字符串 ***'123'*** 处理成整形123

```
1 age = input("plz input your age:")
2 t = int(age)
3 print t
```

6.6.3 字符串索引

① 通过index索引访问字符串里字母 ***'j'***

```
1 s = "hello 'jack'..."
2 print s[7]
```

或则

```
1 s = "hello 'jack'..."
2 print s[s.find('j')]
```

6.6.4 字符串切片

① 输出字串***'jeapedu'***

```
1 s = "www.jeapedu.com"
2 print s[4 : 11]
```

或者用find和index函数找到***'j'*** 和***'.'*** 的位置

```
1 s = "www.jeapedu.com"
2 b = s.find('j')
3 e = s.find('.', b)
4 print s[b : e]
```

或则

```
1 s = "www.jeapedu.com"
2 sub = 'jeapedu'
3 n = len(sub)
4 print s[s.find('j'): s.find('j') + n]
```

或则

```
1 s = "www.jeapedu.com"
2 print s[4]+s[5]+s[6]+s[7]+s[8]+s[9]+s[10]
```

或则通过查找*'ed'*来输出*'jeapedu'*字符串

```
1 s = "www.jeapedu.com"
2 ed = s.find('ed')
3 print s[ed - 4 : ed + 3]
```

② 将字符串里的小写*'j'*换成大写的*'J'*

```
1 s = "www.jeapedu.com"
2 b = s.find('j')
3 s = s[:b] + 'J' + s[b + 1:]
4 print s
```

或则

```
1 s = "www.jeapedu.com"
2 s = s[:s.find('j')] + 'J' + s[s.find('j') + 1:]
```

或则

```
1 s = "www.jeapedu.com"
2 s = s[:s.index('j')] + 'J' + s[s.index('j') + 1:]
```

或则

```
1 s = "www.jeapedu.com"
2 s = s.replace('j', 'J', 1)
```

③ 替换掉多个小写*'j'*换成大写的*'J'*

```
1 s = "www.jeapedu.com" * 10
2 i = 0
3 b = -1
4 while i < 10:
5     b = s.find('j', b + 1)
6     s = s[:b] + 'J' + s[b + 1:]
7     print s
8     i = i + 1
```

或则

```
1 s = "www.jeapedu.com" * 10
2 s = s.replace('j', 'J', 10)
```



```
3 print s
```

6.6.5 字符串转义字符

① 打印输出这样一个字符串*hello "jeapedu" !*

```
1 print "hello \"jeapdu\" !"
```

或则

```
1 print """hello "jeapedu" !"""
```

或则

```
1 print '''hello "jeapedu" !'''
```

或者

```
1 print 'hello "jeapedu" !'
```

② 打印输出*c:\test\node\readme.txt* 这样一个字符串

```
1 print r"c:\test\node\readme.txt"
```

或则

```
1 print "c:\\test\\node\\readme.txt"
```

6.7 字符串进阶

① 替换*"abc"*为*"ABC"*

```
1 s = "11abc22abc33abc44abc55"
2 s = s.replace("abc", "ABC")
```

或者用字符串切片来完成

```
1 s = "12abc22abc33abc44abc55"
2 while s.find("abc") != -1:
3     pos = s.find("abc" )
4     s = s[: pos]+"ABC"+s[pos + 3:]
5 print s
```

② 去除多余的*"abc"*

```
1 s = "11abc22abc33abc44abc55"
2 first = s.find("abc")
```

```
3 s1 = s[first + 3:]
4 s1 = s1.replace("abc", "")
5 s = s[0:first + 3] + s1
6 print s
```

或者用字符串切片来完成

```
1 s = "12abc22abc33abc44abc55"
2 first = s.find("abc")
3 s1 = s[first + 3:]
4 while s1.find("abc") != -1:
5     pos = s1.find("abc" )
6     s1 = s1[: pos]+s1[pos + 3:]
7 s = s[0:first + 3] + s1
8 print s
```

如果s = "12abcabc22ababcc33aabcbc44abc55"，上边的程序正确么？

③ 切分单词,字符串里的单词以"*::*" 分割

```
1 s = "::xyz::abc::bcd::cde::def"
2 #s = "qrt::xyz::abc::bcd::cde::def::"
3 while s.find("::") != -1:
4     if s.find("::") == 0:
5         st = "empty"
6     else:
7         st = s[:s.find("::")]
8     print "st"+str(i),st
9     i += 1
10    s = s[s.find("::") + 2:]
11 if len(s):
12         print "st"+str(i),s
```

6.8 字符串高级: 自动刷视频

6.8.1 提取网页里的超级链接地址

① 浏览网页***http://www.tudou.com/home/jeapedu***

② "查看全部"***http://www.tudou.com/home/item_u121551255s0p1.html***

③ 鼠标移至某图片或某图片下方的文字之后鼠标右键，选"审查元素",可见如下文字

```
<div class = "pic"
    <div class = "quick"....
    <a class = "inner" ....
```

```
        <img width = .....
<div class = "txt"
```

④ 点开<***div class = "txt"*** 之下的"<***h1 class = "caption"***>" 可见如下字符

```
<a target="new" title="智普教育06替换操
    作vim" href="http://www.tudou.com/programs/view/
    myLEA7IrHx4/"> 智普教
    育06vim 替..</a>
```

⑤ 鼠标点中<***a target="new" title***=这行，右键"***Copy as Html***"，然后赋值给变量s。

```
s = "<a target="new" title="智普教育培
    训Python 06 vim 替换操作" href="http://www.tudou.
    com/programs/view/myLEA7IrHx4/"> 智普教育Python 培
    训 06 vim 替..</a>"
```

⑥ 向上滚动鼠标，找到<html>,点击向下三角块，右键"***Copy as Html***"，然后赋值给变量sa。

```
s = """<a target="new" title智普教育="培
    训Python 06 替换操作vim" href="http://www.tudou.
    com/programs/view/myLEA7IrHx4/"> 智普教育Python 培
    训 06 vim 替..</a>"""
sa = """<html><head><meta http-equiv="Content-Type"
     content="text/html; charset=GBK"><title>k
.....中间部分省略

.....
frameborder="0" src="http://ui.tudou.com/js/embed/
    xstorage/storage.html" style="visibility:hidden;
    position: absolute"></iframe></body></html>"""
```

⑦ 先从s里提取一条地址***http://www.tudou.com/programs/view/myLEA7IrHx4/***

```
s = """<a target="new" title智普教育="培
    训Python 06 替换操作vim" href="http://www.tudou.
    com/programs/view/myLEA7IrHx4/"> 智普教育Python 培
    训 06 vim 替..</a>"""
sa = """<html><head><meta http-equiv="Content-Type"
     content="text/html; charset=GBK"><title>k
.....中间部分省略
.....
frameborder="0" src="http://ui.tudou.com/js/embed/
    xstorage/storage.html" style="visibility:hidden;
```

```
    position: absolute"></iframe></body></html>"""

url = s[s.find("href") + 6 : s.find("\">")]
print url
```

⑧ 从sa里提取一条网络地址

```
s = """<a target="new" title智普教育="培
    训Python 06 替换操作vim" href="http://www.tudou.
    com/programs/view/myLEA7IrHx4/"> 智普教育Python 培
    训 06 vim 替..</a>"""
sa = """<html><head><meta http-equiv="Content-Type"
     content="text/html; charset=GBK"><title>k
.....中间部分省略
.....
frameborder="0" src="http://ui.tudou.com/js/embed/
    xstorage/storage.html" style="visibility:hidden;
    position: absolute"></iframe></body></html>"""

url = s[s.find("href") + 6 : s.find("\">")]
print url

urla = 0
urlatarget = sa.find("<a target",urla)
urla = sa.find("</a",urlatarger)
s = sa[urlatarger : urla]
urlb = s.find("href")
urle = s.find(""\">"")
url = s[urlb + 6 : urle]
```

⑨ 从sa里提取40条地址

```
s = """<a target="new" title智普教育="培
    训Python 06 替换操作vim" href="http://www.tudou.
    com/programs/view/myLEA7IrHx4/"> 智普教育Python 培
    训 06 vim 替..</a>"""
sa = """<html><head><meta http-equiv="Content-Type"
     content="text/html; charset=GBK"><title>k
.....中间部分省略
.....
frameborder="0" src="http://ui.tudou.com/js/embed/
    xstorage/storage.html" style="visibility:hidden;
    position: absolute"></iframe></body></html>"""
```

```
6
7  url = s[s.find("href") + 6 : s.find("\">")]
8  print url
9
10 urla = -1
11 i = 0
12 while i < 40:
13         urlatarget = sa.find("<a target",urla + 1)
14         urla = sa.find("</a",urlatarget)
15         s = sa[urlatarget : urla]
16         urlb = s.find("href")
17         urle = s.find("\">")
18         url = s[urlb + 6 : urle]
19         i += 1
20         print i,url
```

6.8.2 网址加载到浏览器里

```
1 # easy_install webbrowser
2 import webbrowser as web
3 url = "http://www.tudou.com/programs/view/myLEA7IrHx4/"
4 web.open_new_tab(url)
```

6.8.3 关闭浏览器

```
1 import os
2 import time
3 time.sleep(1)
4 os.system("taskkill /F /IM chrome.exe")
```

6.8.4 给浏览器发送*Ctrl + F5*

① 安装*"SendKeys"*方法一

```
http://pan.baidu.com/share/link?uk=1462801323&
   shareid
=2664151388#dir/path=%2Fpywinauto 下
   载SendKeys-0.3_py27.exe
```

② 安装*"SendKeys"*方法二

```
pip install SendKeys
```

③ 发送***F5***刷新浏览器

```
import webbrowser as web
import SendKeys
import time
url = "http://www.tudou.com/programs/view/
   myLEA7IrHx4/"
web.open_new_tab(url)
SendKeys.SendKeys("^{F5}")
time.sleep(5)
SendKeys.SendKeys("^{F5}")
time.sleep(5)
SendKeys.SendKeys("^{F5}")
time.sleep(5)
```

6.8.5 改进自动刷视频程序

① 实现在线(修改***sa***赋值方式)

```
import urllib2
f = urllib2.urlopen("http://www.tudou.com/home/
   item_u121551255s0p1.html")
sa = f.read()
print sa
```

② 提取网址

a). 安装BeautifulSoup方法一

```
pip install BeautifulSoup4
```

b). 安装BeautifulSoup方法二

```
easy_install BeautifulSoup4
```

c). BeautifulSoup官方文档

```
www.crummy.com/software/BeautifulSoup/bs4/doc/
```

d). BeautifulSoup提取网址

```
from bs4 import BeautifulSoup
import urllib2
f = urllib2.urlopen("http://www.tudou.com/home/
   item_u121551255s0p1.html")
sa = f.read()
soup = BeautifulSoup(sa)
```

```
6  tag = soup.findAll('a', {'class':'inner'})
7  print tag
8  i = 0
9  for hr in tag:
10     url = hr.get('href')
11     print i, url
12     i += 1
```

③ 每个视频刷随机次数

```
1  import webbrowser as web
2  import SendKeys
3  import time
4  import random
5  url = "http://www.tudou.com/programs/view/
       myLEA7IrHx4/"
6  web.open_new_tab(url)
7  n = random.randint(0, 10)
8  i = 0
9  while i < n:
10     time.sleep(20)
11     SendKeys.SendKeys("^{F5}")
12     i += 1
```

第 7 章. 列表及列表函数

7.1 列表基础

列表也是一种序列型数据类型，一有序数据集合用逗号间隔用方括号括起来，这就是列表数据。既然是序列型的数据也就和字符串一样可以通过索引index和切片来访问某个元素或子列表了。

7.1.1 列表的索引访问

想访问列表里的某个元素可以指定该元素所在的位置(起始依然为0)来访问。语法结构如下所示:

```
list_name[index]
```

示例如下所示:

```
1 li = [1, 2, 3, 4, 5]
2 print li[2]
```

可以用del list_name[index]来删除列表里的第index项数据项。

```
1 li = [1, 2, 3, 4, 5]
2 del li[2]
3 print li
```

输出结果如下:

```
[1, 2, 4, 5]
```

7.1.2 列表的切片访问

想访问列表里的一些连续的元素想可以指定起始位置和终止位置来访问某一块子列表。语法结构如下所示:

```
list_name[start : end: step]
```

示例如下所示:

```
1 li = [1, 2, 3, 4, 5]
```

```
2 print li[2: 4]
```

可以用del list_name[start:end]来删除列表里的start到end-1所有数据项。

```
1 li = [1, 2, 3, 4, 5]
2 del li[1:3]
3 print li
```

输出结果如下:

```
[1, 4, 5]
```

7.1.3 列表的基本运算

列表的加法运算 列表的加法和字符串加法一样可以将多个列表组合成一个新的更长的列表。

示例如下所示:

```
1 li1 = [1, 2, 3, 4, 5]
2 li2 = range(6, 9)
3 li = li1 + li2
4 print li
```

输出结果如下:

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

列表的乘法运算 列表的乘法相当于同一个列表相加了n次。示例如下所示:

```
1 li1 = [1, 2, 3, 4, 5]
2 li = li1 * 3
3 print li
```

输出结果如下:

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

列表的in和not in运算 in运算可以判定某个值是否为列表的元素值，或者说判定一下列表里是否包含了某个元素。而not in 运算则是可以判断出是否不包含有这个值。示例如下所示:

```
1 li = [1, 2, 3, 4, 5]
2 print li
3 if 2 in li:
4     print 'Yes 2 in li'
```

```
5 if 'x' not in li:
6     print "Yes 'x' not in li"
```

输出结果如下:

```
[1, 2, 3, 4, 5]
Yes 2 in li
Yes 'x' not in li
```

列表的遍历 想逐一访问列表里的各个元素可以像字符串那样通过for循环来依次取出列表的各个元素项的值，在for迭代器的帮助下依次取出第0、第1、第2、...第n个元素。示例如下所示:

```
1 li1 = [1, 2, 3, 4, 5]
2 li = li1 * 3
3 i = 0
4 for val in li:
5     print "li[%d]"%(i),val
6     i += 1
```

输出结果如下:

```
li[0] 1
li[1] 2
....
li[13] 4
li[14] 5
```

小练习:剔除列表里的相同多余元素项。小练习:用列表实现求1~9的立方x^3 。

```
1 li2 = range(1,10)
2 print li2
3 li = [x ** 3 for x in li2]
4 print li
```

输出结果如下:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

下面的这个表达式称之为**列表的解析**。

```
li = [x ** 3 for x in li2]
```

其语法结构如下所示:

```
[val_expr for val in list_name]
```

val_expr是变量val的运算表达式，val用于存储for每次从list_name列表里取出的元素的值，用每一个val_expr的值作为构建新列表的元素项。

7.2 列表相关函数

列表的相关函数很多，可以通过help函数查看list列表到底提供了哪些函数可以使用，help(list)或help(list.function_name)来查看帮助文档。

7.2.1 len函数

len函数是Python内建函数不属于任何数据类型，一般序列型的数据可以通过len函数测得数据的长度。

```
1 li = range(1,11)
2 print li
3 print len(li)
```

输出结果如下:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
10
```

7.2.2 count函数

列表的count函数方法可以统计列表里某元素项相同的一共有几个。

```
1 li = range(1, 3)
2 li3 = li * 3
3 print li3
4 print li3.count(li3[2])
5 print li3.count('a')
```

输出结果如下:

```
[1, 2, 1, 2, 1, 2]
3
0
```

7.2.3 insert函数

列表的insert函数可以将对象添加到列表的指定位置，列表里的元素顺序后移。

```
1 li = []
2 li = range(1, 11)
3 print li
4 li.insert(5, 10)
```

```
5 print li
```

输出结果如下:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 10, 6, 7, 8, 9, 10]
```

7.2.4 append函数

```
1 li = []
2 li1 = range(1, 5)
3 for x in li1:
4     li.append(x)
5 print li
```

输出结果如下:

```
[1, 2, 3, 4]
```

如果添加的对象是列表，列表作为整体添加到列表的尾部。

```
1 li = []
2 li1 = range(1, 5)
3 for x in li1:
4     li.append(x)
5 print li
6 li1 = range(5, 10)
7 li.append(li1)
8 print li
```

输出结果如下:

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

7.2.5 extend函数

extend函数可以将一个列表的所有元素以个体的方式添加到列表的尾部。

```
1 li1 = range(1, 6)
2 print li1
3 li2 = range(6, 11)
4 print li2
5 li1.extend(li2)
6 print li1
```

输出结果如下:

```
[1, 2, 3, 4, 5]
[6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

7.2.6 remove函数

remove函数可以将列表的第一次出现的指定元素删除。

```
1 li = range(1, 4) * 3
2 print 'li', li
3 li.remove(2)
4 print 'li',li
```

输出结果如下:

```
li [1, 2, 3, 1, 2, 3, 1, 2, 3]
li [1, 3, 1, 2, 3, 1, 2, 3]
```

7.2.7 pop函数

pop函数可以将列表指定位置的元素从列表删除或者尾部的元素从列表里删除。

```
1 li = range(1, 4) * 3
2 print 'li    ',li
3 li.pop()
4 print 'pop   ', li
5 li.pop(3)
6 print 'pop 3', li
```

输出结果如下:

```
li     [1, 2, 3, 1, 2, 3, 1, 2, 3]
pop    [1, 2, 3, 1, 2, 3, 1, 2]
pop 3 [1, 2, 3, 2, 3, 1, 2]
```

练习 用函数实现剔除列表里重复的元素。

① remove实现

```
1 li = "jeapedu" * 100
2 li = list(li)
3 li.sort()
4 print li
5 i = 0
```

```
6  for s in li:
7      print i, s
8      while li.count(s) > 1:
9          li.remove(s)
10     i += 1
11     print li
```

② pop实现

```
1  li = [1, 1, 1, 2, 2, 3, 3, 3, 4]
2  length = len(li)
3  print li
4  pos = length - 1
5  while pos >=0:
6    r = li.count(li[pos])
7    print 'pos ',pos,'r ',r
8    if r > 1:
9      i = 0
10     print 'del :'
11     while i < r - 1:
12       print li.pop(pos)
13       pos = pos - 1
14       i = i + 1
15   else:
16     print '=>not repeat!'
17   pos -= 1
18 print li
```

③ 请用自己编写程序：extend和append实现

练习 去除二维列表[ ]，二维变一维。

① 方法一,isinstance函数可以判定数据的类型

```
1  li = [1, 2, 4,range(1,4),5,range(1,4),6]
2  print li
3  k = 0
4  for li1 in li:
5      if isinstance(li1, list):
6          j = 0
7          print len(li1)
8          for li2 in li1:
9              li.insert(k + j, li2)
```

```
10                li.remove
11                print li
12                j = j + 1
13            del li[k + j]
14        k = k + 1
15    print li
```

② 方法二

```
1  li = [1, 2, 4,range(1,4),5,range(1,4),6]
2  b = []
3  for i in li:
4      if isinstance(i, list):
5          b.extend(i)
6      else:
7          b.append(i)
```

练习 合并两个列表(相同不要)。

7.3 元组 *Tuple*

元组的结构和访问、使用方法和列表基本一致，区别有二：使用圆括号将各个数据项括起来的、元组里的元素值不可修改。看用过索引来访问元组里的某个元素值，通过切片来访问子元组。

```
1  >>> t = tuple(range(1,11))
2  >>> print t
3  (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
4  >>> print t[3]
5  4
6  >>> print t[2:8]
7  (3, 4, 5, 6, 7, 8)
8  >>>
```

第 8 章. 文件操作

8.1 文件基础

文件读写是Python编程基础内容，在Python下通常先以某种方式打开文件，之后再进行相应的读写操作，读写完成以后一定要关闭打开的文件。打开文件可以用open函数，将指定的文件(第一个参数)以某种模式(第二个参数)打开。

8.1.1 读文件

在以'r'读模式打开文件以后可以调用read函数一次性将文件里的内容全部读取回来，也可以指定每次read 读多少个字节，示例如下所示:

```
1  #coding:utf-8
2  fn = "a.txt"
3  fp = open(fn , 'r')#以读的方式打开文件文件必须首先存在
       和,.文件在同一目录下py
4  print "reading position: ", fp.tell()
5  r = fp.read(20)#读文件里的内容返回是字符串,
6  print "have read: \""+r+"\""
7  print "reading position ",fp.tell()
8  r = fp.read(20)
9  print "have read: \""+r+"\""
10 print "reading position: ",fp.tell()
11 fp.close()
```

输出结果如下:

```
reading position:  0
have read: "welcome to Jeapedu.c"
reading position  20
have read: "om Python education "
reading position:  40
```

需要说明的是tell函数可以返回当前即将要读的位置。读者可以试试在运行程序之前不创建a.txt看看运行程序时会有什么样的结果?

8.1.2 写文件

如果想将某些内容写入文件，可以以'w'写的方式打开文件，当文件以'w'方式打开时，会遇到这样的问题：文件可能不存在，怎么办？没有就创建呗！如果存在且文件里有内容又该怎么办呢？打开且清空文件里的内容，就这个逻辑！

```
1 fw = 'c.txt'
2 fp = open(fw , 'w')
3 fp.write('jeapedu.com\n')
4 fp.close()
```

读者可以自行查看文件里是否被写入了上边的内容。为了更好的理解'w'写模式，有两个小实验可以去做一下：

① 和.py文件在同一目录下没有c.txt文件，运行程序看是否产生了c.txt文件，里面有刚写入的文本么？

② 在.py目录下有c.txt文件，里面有内容(随便写点儿别的)，运行程序，看原来的内容是否被冲掉了？

8.1.3 读写文件r+ w+

'r'是以读的模式打开文件，之后只能用read函数读取文件里的内，'w'是以写的方式打开文件之后也只能用write 函数向文件里写些新的内容，规矩太多了，谁记得住?!好了，又有两个模式出现了即'r+'和'w+'模式，'r+'和'w+'都是以读写模式(是说文件可以被读也可以被写入内容，个人建议一次打开到文件关闭之间要么读要么写，不要两件事同时做！)打开文件，区别在于'r+' 必须是针对已存在的文件而'w+' 可以创建未存在的文件。

```
1 fn = "rplus.txt"
2 fp = open(fn , 'r+')
3 r = fp.read(12)
4 print r
5 fp.close()
```

如果当前目录下无rplus.txt文件会报错：

```
>>>

Traceback (most recent call last):
  File "C:/Python27/test/fra.py", line 20, in <module>
    fp = open(fn , 'r+')
IOError: [Errno 2] No such file or directory: 'rplus.
   txt'
>>>
```

如果上边r+改成w+则不会报错！

```
1 fn = "rplus.txt"
2 fp = open(fn , 'w+')
3 r = fp.read(12)
4 print 'have read ' + r
5 fp.close()
```

由于rplus.txt文件不存在，'w+'则会创建之，对新文件rplus.txt去读当然啥都读不回来了

```
>>>
have read
```

8.1.4 追加写入文件a

'w'写模式打开的文件只能把内容写入文件，文件里的原有内容被清除掉，如果想保留原内容，只是把新内容追加到文件的尾部或者说新增新写入内容而不影响原内容，那么用'a'追加写模式吧。

```
1 fn = "rplus.txt"
2
3 fp = open(fn , 'w+')
4 fp.write('aaaa\n')
5 fp.close()
6
7 fa = open("rplus.txt", 'a')
8 fa.write("bbbb\n")
9 fa.close()
```

文件rplus.txt里的内容如下:

```
aaaa
bbbb
```

字串'bbbb\n'被追加到了文件尾部(在'aaaa\n'的'\n'之后)，保留了原有内容'aaaa\n'，为何文件里不见了\n啊？，呵呵。

8.2 格式化读写文件

8.2.1 格式化写文件

如果想写入文本文件的数据最终在文件里以下面的形式展示出来，需要在调用write函数时使用格式化控制字来格式化写入字符串。

下面的数据表现出数据是右对齐，且每列数据间有制表位。第一行是字段名，从第二行起是一条条的数据，和Excel唯一的区别是没有边框。

```
name    age         sex
 lwk     56        male
 lwk     56        male
 lwk     56        male
 lli     56        male
 lwk     56      female
```

实现程序如下所示:

```
1 fn = "wformat.txt"
2 fw = open(fn, 'w')
3 fw.write("%10s\t %3s\t %6s\n"%("name", "age", 'sex'))
4 fw.write("%10s\t %3d\t %6s\n"%("lwk", 56, 'male'))
5 fw.write("%10s\t %3d\t %6s\n"%("lwk", 56, 'male'))
6 fw.write("%10s\t %3d\t %6s\n"%("lwk", 56, 'male'))
7 fw.write("%10s\t %3d\t %6s\n"%("lli", 56, 'male'))
8 fw.write("%10s\t %3d\t %6s\n"%("lwk", 56, 'female'))
9 fw.close()
```

8.2.2 读成列表

文件的readlines函数可以将文本文件的若干行文本一一映射成列表的若干项，即文本文件的每一行映射成列表的一个数据项，每个数据项都是字符串。假设文本文件a.txt里的内容如下:

```
aaaa
bbbb
cccc
dddd
```

通过readlines函数将四行文本编程列表的四个元素项。

```
1 fr = open("a.txt",'r')
2 print fr.readlines()
3 fr.close()
```

输出结果如下:

```
['aaaa\n', 'bbbb\n', 'cccc\n', 'dddd']
```

8.2.3 读一行文件

读模式打开的文件可以用read函数一次性将文件的内容全部读回，这是read的特点，有的时候想读一行回来行不？可以用readline函数读一行内容，假设文本文件a.txt里的内容如下:

```
    aaaa
   bbbb
  cccc
 dddd
```

可以看出前三行文字前有若干个空格，每行尾部还有\n。

```
1 fr = open("a.txt",'r')
2 print fr.readlines()
3 fr.close()
```

输出结果如下:

```
['   aaaa\n', '  bbbb\n', ' cccc\n', 'dddd']
```

怎样得到后无\n和前无空格的纯字符串呢？这里得用到字符串函数strip来去掉\n和空格。

```
1 fr = open("c.txt",'r')#only read, if r+ then can read
    and write
2 print fr.readline().strip().strip('\n')
3 print fr.readline().strip().strip('\n')
4 print fr.readline().strip().strip('\n')
5 print fr.readline().strip().strip('\n')
6 fr.close()
```

输出结果如下:

```
aaaa
bbbb
cccc
dddd
```

8.2.4 split格式化数据

在格式化写入章节里，通过格式化控制字写入的数据如何读回呢？

```
name    age        sex
 lwk     56       male
 lwk     56       male
 lwk     56       male
 lli     56       male
 lwk     56     female
```

上边数据每一行是一条记录，读回时可以把整行当作一个数据，当有的时候我们想最好这是编程三个单独的数据分别存储当前行的姓名、年龄和性别数据。可以

用split函数来把这一行字符串分割成三个数据项。

```
1 fr = open("wformat.txt",'r')
2 line1 = fr.readline()
3 print line1
4 line2 = fr.readline()
5 print line2
6 print line2.split('\t')
7 fw.close()
```

输出结果如下:

```
        name    age         sex

         lwk     56        male

['        lwk', '  56', '    male\n']
```

输出的结果很诡异! 好像有多余的空行?

```
1 fr = open("wformat.txt",'r')
2 line1 = fr.readline().strip().strip('\n')
3 print line1
4 line2 = fr.readline().strip().strip('\n')
5 print line2
6 print line2.split('\t')
7 fw.close()
```

输出结果如下:

```
name      age         sex
lwk    56        male
['lwk', '  56', '    male']
```

漂亮多了，呵呵。思考一下strip()和strip('\n')能换一下位置么?

8.2.5 如何读写子目录文件呢?

这很简单，只需要指定文件时描述好路径即可，但需要注意的是转义字符问题。

```
1 #coding:utf-8
2 fn = 'c:\\python27\\a.txt'
3 fp = open(fn, 'w')
4 fp.write('hello jeapedu.com')
5 fp.close()
6
```

```
7  fn = 'sample\b.txt'
8  fp = open(fn, 'w')
9  fp.write('hello jeapedu.com')
10 fp.close()
11
12 fn = 'sample\tmp\a.txt'
13 fp = open(fn, 'w')
14 fp.write('hello jeapedu.com')
15 fp.close()
```

为何代码的第七行会报错？转义字符的啦！解决办法是在前面加个字母r即可或b前面加个\。修改完以后第12行也错了，一样加个r字母(原字符串，不转义)。

```
1  fn = 'c:\\python27\\a.txt'
2  fp = open(fn, 'w')
3  fp.write('hello jeapedu.com')
4  fp.close()
5
6  fn = 'sample\\b.txt'
7  fp = open(fn, 'w')
8  fp.write('hello jeapedu.com')
9  fp.close()
10
11 fn = r'sample\tmp\a.txt'
12 fp = open(fn, 'w')
13 fp.write('hello jeapedu.com')
14 fp.close()
```

练习：设计个人通信录程序，通信录存储在文本文件里

第 9 章. 字典

9.1 字典定义

字典的元素是由一对称之为键和值构成，用逗号间隔起来、用花括号括起来就构成了字典。从字典的定义可以得出：字典里的每个元素的键和值之间用冒号间隔，元素项之间用逗号间隔，整体用花括号括起来。语法结构如下所示：

```
dict_name = {key:value, key:value,.....}
```

举例说明如下：

```
1 dict1 = {'a' : 12, 'b' : 14, 'c': 18, 'ff' : 25}
```

字典的数据项的值可以是字典、列表等数据类型。

```
1 dict2 = {"jeapedu":{"domain":"www.jeapedu.com", "qq":
      1941847311}, "cpython":{"domain": "www.cpython.org",
       "qq" : 1941847311}}
```

9.2 字典基础操作

9.2.1 字典长度

len函数可以测得字典的数据项个数。

```
1 >>> dict1 = {'a': 'b', "name" : "jeapedu", 12 : 34}
2 >>> len(dict1)
3 3
```

9.2.2 字典元素值的访问

Python的字典可以通过键获取其所对应的值，而序列型数据字符串、列表则是通过index 索引来获取其所对应的值。正是因为如此字典是非序列型数据类型，字典里的元素间的关系比较稀松，无序。

```
1 >>> dict1 = {'a': 'b', "name" : "jeapedu", 12 : 34}
2 >>> print dict1['a']
```

```
3 'b'
4 >>> print dict1["name"]
5 "jeapedu"
```

9.2.3 字典元素值的修改

Python的字典可以通过键获修改所对应的值。

```
1 >>> dict1 = {'a': 'b', "name" : "jeapedu", 12 : 34}
2 >>> dict1['a'] = "hello"
3 >>> print dict1
4 {'a': 'hello', 12: 34, 'name': 'jeapedu'}
```

9.2.4 字典元素项的删除

Python的字典可以通过*del* **字典名**[**键**]来删除字典里的元素。

```
1 >>> dict1 = {'a': 'hello', 12: 34, 'name': 'jeapedu'}
2 >>> del dict1[12]
3 >>> print dict1
4 {'a': 'hello', 'name': 'jeapedu'}
```

9.2.5 字典元素项的增加

如果字典里无某个键，可以通过**字典名**[**新键**]赋值的方式在字典里新增一个数据项。

```
1 >>> dict1 = {'a': 'hello', 'name': 'jeapedu', 'sex': 'm
     '}
2 >>> dict1["qq"] = 1941847311
3 >>> print dict1
4 {'a': 'hello', 'qq': 1941847311, 'name': 'jeapedu', '
     sex': 'm'}
5 >>>
```

9.2.6 字典的*in*运算

字典里的in运算可以判断某键是否在字典里。Python的字典可以通过键字典里的元素。

```
1 >>> dict1 = {'a': 'hello', 'name': 'jeapedu'}
2 >>> 'name' in dict1
3 True
```

主要in运算查找的是key键，而不是value值。

9.2.7 字典小练习

用姓名做key键，用性别、年龄、电话做组成的value值(也是字典)，构造一个通信录字典数据结构。

9.3 字典相关函数

9.3.1 清空字典数据项clear

字典的clear方法可以清空字典里的数据项。

```
1 >>> print dict1
2 {'a': 'hello', 'qq': 1941847311, 'name': 'jeapedu', '
     sex': 'm'}
3 >>> dict1.clear()
4 >>> print dict1
5 {}
```

9.3.2 字典复制copy函数

字典的copy方法可以新建一字典与拷贝对象里的数据项一样，与源对象间的关系是备份关系。

```
1 >>> dict1['sex'] = 'male'
2 >>> dict1['age'] = 9
3 >>> dict1['name'] = "jeapedu.com"
4 >>> print dict1
5 {'age': 9, 'name': 'jeapedu.com', 'sex': 'male'}
6 >>> dict2 = dict1.copy()
7 >>> print dict2
8 {'age': 9, 'name': 'jeapedu.com', 'sex': 'male'}
9 >>> del dict2['sex']
10 >>> print dict2
11 {'age': 9, 'name': 'jeapedu.com'}
12 >>> print dict1
13 {'age': 9, 'name': 'jeapedu.com', 'sex': 'male'}
14 >>>
```

代码第9行删除了dict2里的键为"sex"的数据项，但dict1 并未受影响。

9.3.3 获得字典数据函数get

字典可以通过get函数获取某键所对应的值，等价于dict_name[键]这样的操作。

```
1 >>> print dict1
2 {'a': 'hello', 'qq': 1941847311, 'name': 'jeapedu', '
    sex': 'm'}
3 >>> dict1.get("qq")
4 1941847311
```

9.3.4 获得字典所有的key

字典提供了一个函数keys可以获取字典的所有的keys键。

```
1 >>> print dict1
2 {'po': 100110, 'age': 9, 'name': 'jeapedu.com', 'sex':
    'male'}
3 >>> dict1.keys()
4 ['po', 'age', 'name', 'sex']
```

9.3.5 获得字典所有的value

字典提供了一个函数values可以获取字典的所有的values 值。

```
1 >>> print dict1
2 {'po': 100110, 'age': 9, 'name': 'jeapedu.com', 'sex':
    'male'}
3 >>> dict1.values()
4 [100110, 9, 'jeapedu.com', 'male']
```

9.3.6 获得字典所有的key-value

函数items可以获取字典的所有的keys和values值。

```
1 >>> print dict1
2 {'po': 100110, 'age': 9, 'name': 'jeapedu.com', 'sex':
    'male'}
3 >>> dict1.items()
4 [('po', 100110), ('age', 9), ('name', 'jeapedu.com'), (
    'sex', 'male')]
```

items函数的返回值是一个列表。

9.3.7 修改某键的值update

函数update可以更新字典里某key的value。

```
>>> print dict1
{'po': 100110, 'age': 9, 'name': 'jeapedu.com', 'sex':
    'male'}
>>> new = {'age' : 33}
>>> dict1.update(new)
>>> print dict1
{'po': 100110, 'age': 33, 'name': 'jeapedu.com', 'sex':
    'male'}
```

如果更新的key原字典里没有，则update就向字典里添加一项数据。

```
>>> print dict1
{'po': 100110, 'age': 33, 'name': 'jeapedu.com', 'sex':
    'male'}
>>> add = {'Phone':18010096001}
>>> dict1.update(add)
>>> dict1
{'Phone': 18010096001L, 'po': 100110, 'age': 33, 'name'
    : 'jeapedu.com', 'sex': 'male'}
```

9.3.8 dict函数

函数dict可以创建字典。

```
>>> d0 = dict() #创建空字典
>>> print d0
{}
>>> d1 = dict(a=12,dns="jeapedu",c=13) #通过赋值创建字典
>>> print d1
{'a': 12, 'c': 13, 'dns': 'jeapedu'}
>>> value = ['Tom', 'Jack', 'Rose', 'John', 'Micheal']
>>> print value
['Tom', 'Jack', 'Rose', 'John', 'Micheal']
>>> key = range(1,6)
>>> d2 = dict(zip(key, value))#使用一对列表创建字典
>>> print d2
{1: 'Tom', 2: 'Jack', 3: 'Rose', 4: 'John', 5: 'Mark'}
```

9.3.9 pop和popitem函数

pop方法可以通过键key来获取其值并从字典里删除该数据项，而popitem函数则是随机的移除一个数据项，返回值是元组。

```
>>> print d2
{1: 'Tom', 2: 'Jack', 3: 'Rose', 4: 'John', 5: 'Mark'}
>>> d2.pop(2)
'Jack'
>>> d2.popitem()
(1, 'Tom')
>>> print d2
{3: 'Rose', 4: 'John', 5: 'Mark'}
```

9.4 字典基础练习

9.4.1 字典和for循环遍历字典

通过in运算和键，来访问字典的值。

```
>>> d1 = {"a" : 12, "b" : 13, "c" : 14, 15 : 16}
>>> for x in d1:
        print d1[x]
```

输出结果为:

```
12
14
13
16
```

通过items函数返回值为(key, value)元组组成的列表来访问字典里各个元素的key和value值。

```
>>>d1 = {"a" : 12, "b" : 13, "c" : 14, 15 : 16}
>>>print d1.items()
[('a', 12), ('c', 14), ('b', 13), (15, 16)]
>>>for (k, v) in d1.items():
      print "dict[", k, "] = ",v
```

输出结果为:

```
dict[ a ] =  12
dict[ c ] =  14
dict[ b ] =  13
dict[ 15 ] =  16
```

第 10 章. 模块

*10.1 import*引入其他标准模块

是否所有的代码都需要自己写呢？不，Python提供了很多第三方的模块供大家选用，Python标准安装包里的模块称为标准库。引入模块的方式有三:

① 引入模块

```
import moduleName
```

② 引入模块下的函数

```
from moduleName import function1, function2,...
```

③ 引入模块的所有函数

```
from moduleName import *
```

引入模块以后，就可以使用模块里的函数了，使用方法如下:

```
modulename.function(args)
```

以计算16平方根为例说明一下模块的引入和模块函数的使用。

```
1 >>> import math
2 >>> r = math.sqrt(16)
3 >>> print r
4 4.0
```

如果模块或者函数名字过长可以在import后使用as给该模块取个假名。之后可以通过“假名.函数”来使用模块里的函数。

```
1 >>> import webbrowser as web
2 >>> web.open_new_tab("http://www.jeapedu.com")
```

10.2 使用自定义模块

任何Python程序hello.py(无主函数)都可以作为一个模块被另外一个Python程序test.py引入使用。

Listing 10.1: 'hello.py'

```
1 def hello():
2   print "imported me, hello!"
3 def logo():
4   print "www.jeapedu.com"
```

Listing 10.2: 'test.py'

```
1 import hello #文件名作为模块名去掉文件的扩展名(.py)
2 hello.hello()
3 hello.logo()
```

执行结果如下所示:

Listing 10.3: "test.py 程序运行结果输出"

```
imported me, hello!
www.jeapedu.com
```

以上引入模块的程序和模块程序需要在同一目录下，这个有点局限性，想模块文件可被任何程序文件引用还需做些处理步骤，添加模块文件所在目录到系统里去，让Python 解释器能够找到模块文件，需调用os.path.append(模块文件所在目录)，这样其他程序文件在使用这个模块时就可以找到该模块文件了。

Listing 10.4: " 层次结构"

```
~/|
  |_test.py            #调用"hello模块的程序文件"test.py
  |_test               #目录test
    |__hello.py        #模块文件hello.py
    |__hello.pyc       #模块字节码文件hello.pyc
```

注: *.pyc*是**模块字节码文件**，在使用模块时*Python*解释器需要把模块加载到系统里去，如果发现*.py*比*.pyc* 新，则需要重新编译生成*.pyc* 文件，如果*.pyc* 比*.py*新则不需要编译了，文件*.pyc* 是在第一次加载模块文件时完成的。

需要在程序文件test.py里先把搜索路径找到，这里用过sys模块来添加加载模块hello 的搜索路径。

Listing 10.5: 'test.py'

```
1 import sys
2 sys.path.append('test/')
3 import hello
4 hello.hello()
5 hello.logo()
```

这种方法不好！通用的告知Python解释器到到那去找模块文件的方法有二:
① PYTHONPATH环境变量里增加模块文件所在路径
② .pth文件,文件里列出所有模块文件所在路径(放在安装目录即可)
③ 模块包

一个大的模块不可能就只有一文件、一个大目录，应该是有多个、多层子目录，每个目录下有若干个文件，这样在使用模块时可以通过目录描述来使用其子目录下的模块文件。如打算使用mod.py模块文件里的某函数。

```
dir0\dir\dir2\mod.py
```

可以这样去用

```
import dir0.dir1.dir2.mod as mod
mod.function(args)
```

一条死规则：除包的容器目录dir0外的所有目录下均有一个__init__.py文件，此文件内容可为空。

10.3 使用模块示例Json模块

以使用json模块为例展示Python下如何使用模块的。

10.3.1 Python数据转Json数据，编码dumps

Python下使用dumps函数可以将***Python***数据字典转换成***Json***数据结构。

Python	***Json***
dict	object
list,tuple	array
unicode str	str
int, long	number(int)
float	number(real)
True	true
False	false

Listing 10.6: 'Python 数据编码成Json数据'

```
1 import json
2 s = [{"f":(1, 3, 5, "a"), "x" : None}]
3 d = json.dumps(s)
4 print d
```

程序运行结果如下所示:

```
[{"x": null, "f": [1, 3, 5, "a"]}]
```

从结果中可以看出***Python***的元组数据转换成了***Json***的array数组型数据，***None***转换成了***Null***。

10.3.2 Json数据转Python数据，解码loads

Json模块库里的loads函数可以将***Json***字符串转换成***Python***字典，***Json***字符串里的数组转换为***Python***列表，object转换为***Python*** 字典，其他数据类型基本不发生变化。转化关系如下所示:

Json	***Python***
object	dict
array	list
str	unicode str
number(int)	int, long
number(real)	float
true	True
false	False

从下列网址在线下载一条Json数据。

```
http://developer.usa.gov/1usagov
```

把第一条数据用单引号引起来，并赋值给s字符串变量。

Listing 10.7: 'Json 数据转换成Python数据'

```
1 import json
2 s1 = '{ "a": "Mozilla\/5.0 (Linux; U; Android 4.0.4; zh
    -cn; GT-I8258 Build\/IMM76D) AppleWebKit\/534.30 (
    KHTML, like Gecko) Version\/4.0 Mobile Safari
    \/534.30 baiduboxapp\/4.7.1 (Baidu; P1 4.0.4)", "c":
     "CN", "nk": 0, "tz": "Asia\/Chongqing", "gr": "21",
     "g": "1cLkbYk", "h": "1cLkbYk", "l": "bitly", "al":
     "zh-CN, en-US", "hh": "1.usa.gov", "r": "http:\/\/
    h2w.iask.cn\/jump.php?url=http%3A%2F%2F1.usa.gov%2
    F1cLkbYk", "u": "https:\/\/petitions.whitehouse.gov
    \/petition\/investigate-jimmy-kimmel-kids-table-
    government-shutdown-show-abc-network\/tLxzbBjg", "t
    ": 1383118836, "hc": 1382308785, "cy": "Yinchuan", "
    ll": [ 38.468102, 106.273102 ] }'
3 d = json.loads(s1)
4 print d
5 print "d['t']", d['t']
6 print "d['r']", d['r']
7 print "d['tz']",d['tz']
```

程序运行结果如下所示:

```
{u'a': u'Mozilla/5.0 (Linux; U; Android 4.0.4; zh-cn;
   GT-I8258 Build/IMM76D) AppleWebKit/534.30 (KHTML,
   like Gecko) Version/4.0 Mobile Safari/534.30
   baiduboxapp/4.7.1 (Baidu; P1 4.0.4)', u'c': u'CN', u
   'nk': 0, u'tz': u'Asia/Chongqing', u'gr': u'21', u'g
   ': u'1cLkbYk', u'h': u'1cLkbYk', u'cy': u'Yinchuan',
    u'l': u'bitly', u'al': u'zh-CN, en-US', u'hh': u'1.
   usa.gov', u'r': u'http://h2w.iask.cn/jump.php?url=
   http%3A%2F%2F1.usa.gov%2F1cLkbYk', u'u': u'https://
   petitions.whitehouse.gov/petition/investigate-jimmy-
   kimmel-kids-table-government-shutdown-show-abc-
   network/tLxzbBjg', u't': 1383118836, u'hc':
   1382308785, u'll': [38.468102, 106.273102]}
d['t']1383118836
d['r']http://h2w.iask.cn/jump.php?url=http%3A%2F%2F1.
   usa.gov%2F1cLkbYk
d['tz'] Asia/Chongqing
```

10.4 正则模块*re*

10.4.1 正则表达式的应用

正则表达式常见的应用如下:

- 验证字符串，即验证给定的字符串或子字符串是否符合指定特征，譬如验证是否是合法的邮件地址、验证是否为合法的HTTP地址等。
- 查找字符串，从给定的文本中查找符合指定特征的字符串，比查找固定字符串更加灵活方便。
- 替换字符串，即把给定的字符串中的符合指定特征的子字符串替换为其他字符串，比普通的替换更强大。
- 提取字符串，即从给定的字符串中提取符合指定特征的子字符串。

10.4.2 正则表达式的测试工具

创建一个正则表达式之后，需要测试该正则表达式是否正确。本书中使用正则表达式测试工具“Notepad++”来测试正则表达式。操作步骤：打开Notepad++软件，点击"搜索"菜单下的查找，在弹出的对话框的左下角"查找模式"选项里点选"正则表达式"，在正上方的输入文本框里输入正则表达式即可搜索匹配字符串了。

10.4.3 正则里一些基本概念

正则表达式

正则表达式可以匹配文本片段的模式。正则表达式在程序设计语言中存在着广泛的应用，特别是用来处理字符串。如匹配字符串、查找字符串、替换字符串等。可以

说，正则表达式是一段文本或一个公式，它是用来描述用某种模式去匹配一类字符串的公式，并且该公式具有一定的模式。正则表达式就是用某种模式去匹配一类字符串的公式，主要用来描述字符串匹配的工具。正则表达式描述了一种字符串匹配的模式。它可以用来检查字符串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。正则表达式是由普通字符（如字符a到z）以及特殊字符（称为元字符）组成的文字模式。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。正则表达式就是用于描述某些规则的工具。这些规则经常用于处理字符串中的查找或替换字符串。换句话说，正则表达式就是记录文本规则的代码。正则表达式就是用一个“字符串”来描述一个特征，然后去验证另一个“字符串”是否符合这个特征。[1]

匹配

在正则表达式中，匹配是最常用的一个词语，它描述了正则表达式动作结果。给定一段文本或字符串，使用正则表达式从文本或字符串中查找出符合正则表达式的字符串。有可能文本或字符存在不止一个部分满足给定的正则表达式，这时每一个这样的部分被称为一个匹配。

元字符

一次只能匹配一个字符或者一个位置。故元字符分：匹配字符的元字符和匹配位置的元字符。

① 匹配字符的元字符

- ^string, 这个正则表达式可以匹配所有以string字符串开始的行
- string$, 这个正则表达式可以匹配所有以string 字符串结尾的行
- ^$,则可以匹配空行
- \bStr, \b可以实现匹配以Str开始的单词(等价于\ <)
- Str\b, \b可以实现匹配以Str结尾的单词(等价于\ >)

② 匹配位置的元字符

- \w,可以匹配单词里字符(字母、数字和下划线)
- \W,可以匹配单词里非字符
- \d,可以匹配数字
- \D,可以匹配非数字字符
- \s,可以匹配空格字符
- \S,可以匹配非空格字符

匹配以jea开始

```
1 >>> import re
2 >>> s = 'hello www jeapedu com world'
3 >>> res = r'\bjea'
4 >>> print re.findall(res, s)
5 ['jea']
```

[1] http://blog.csdn.net/wlzhengzebiaodashi/

```
>>> s = 'help jeap jeep jeapedu jeapedu.com '
>>> print re.findall(res, s)
['jea', 'jea', 'jea']
```

匹配以eap结尾

```
>>> import re
>>> s = 'help jeap jeep jeapedu jeapedu.com '
>>> res = r'eap\b'
>>> print re.findall(res, s)
['eap']
```

匹配含hello的行

```
import re
s = """
ahello
www jeapedu com hellob world

helloc world
nice hellod world
piece of helloe world jeapedu
"""
res = r'\hello'
print 'hello ',re.findall(res, s)
```

输出结果如下所示:

```
hello  ['hello', 'hello', 'hello', 'hello', 'hello']
```

匹配含字母和数字

```
import re
s = "a1b2c3d"
res = '\w\d'
print re.findall(res, s)
res = '\w\d\w'
print re.findall(res, s)
```

输出结果如下所示:

```
['a1', 'b2', 'c3']
['a1b', 'c3d']
```

匹配电话

```
1 import re
2 s = 'telephone to 110-123-114119 or call
     4008-6688-9988, 3ks'
3 res = '\d\d\d\D\d\d\d\D\d\d\d\d\d\d'
4 print re.findall(res, s)
5 res = '\d\d\d\d\D\d\d\d\d\D\d\d\d\d'
6 print re.findall(res, s)
```

输出结果如下所示:

```
['110-123-114119']
['4008-6688-9988']
```

太费劲了！有没有好办法，有！可以直接学习使用限定符来简化这类正则。

字符集

用方括号括起来的字符集合，如果其中的任何一个字符被匹配，则它就会找到该匹配项，反字符集可在字符前加^。例如[a-f]匹配a、b、c...f任意某字符，[^b-t]则能匹配除b 到t外的任意字符。字符集也只能匹配一个字符。

假设想匹配Html网页源文件里的<h1>~<h7>字符串，这个匹配正则表达式可以这样去写：<h[1234567]>或者<h[1-7]> 再或者h\d。

再举一例，在匹配电子邮件地址时的正则可以这样去写：匹配Email地址的正则表达式:

```
res = r'[\w\.-]+@[\w\.-]+\.\w{2,4}'
```

，这里的*表示可以匹配0次及以上, +表示可以匹配1次级以上。

```
1 import re
2 s = "Plz write a email to jeap@jeapedu.com or
     service@jeapedu.com, thanks!"
3 res = r'\w[\w\.-]+@[\w\.-]+\.\w{2,4}'
4 print re.findall(res, s)
```

输出结果如下所示:

```
['jeap@jeapedu.com', 'service@jeapedu.com']
```

当然这个正则还是不太准确，不太符合邮件网站对电子邮件名的规定，但能基本匹配出一般字符串里的邮件地址来，也就够用了。

分组或子模式

把一个正则表达式的全部或部分分成一个或多个组。其中，分组使用的字符为“(”和“)”，即左圆括号和右圆括号。分组之后，可以将字符为“(”和“)”之中

的表达式看成一个整体来处理。'P(ython|erl)'可以匹配Python 或者Perl这个字符串。竖线(替换或者可选)表示分组里可以选择选字符"|" 的左边或者右边的规则进行匹配。

例如(t|T)h(e|ere|eir)可以匹配以下字符串:

- the
- The
- their
- Their
- there
- There

```
1 import re
2 s = "www.jeapedu.comwww.JeapedU.comwww.jEAPEDU.comwww.
    heapedu.comwww.HeapedU.com"
3 res = '(j|J)\w*(u|U)'
4 print re.findall(res, s)
5 res = '[jJ]\w*(u|U)'
6 print re.findall(res, s)
7 res = '[jJ]\w*[uU]'
8 print re.findall(res, s)
```

输出结果如下所示:

```
[('j', 'u'), ('J', 'U'), ('j', 'U')]
['u', 'U', 'U']
['jeapedu', 'JeapedU', 'jEAPEDU']
```

请注意分析分组输出结果，只输出匹配括号内的内容。

限定符

正则表达式的元字符一次一般只能匹配一个位置或一个字符，如果要匹配零个或一个或多个字符时，则需要使用限定符。限定符用于指定允许特定字符或字符集自身重复出现的次数。如{n}表示重复n 次、{n,}表示重复至少n 次、{n,m}表示重复至少n次，最多m次。常用限定符的说明如下表所示。

- (pattern)?, 问号?重复0次或1次，等同于{0,1}
- (pattern)*, 星号*重复至少0次，等同于{0,}
- (pattern)+, 加号+重复至少1次，等同于{1,}
- (pattern){m:n}, 重复至少n次，最多m次
- (pattern)??, 使用零次重复（如有可能）或一次重复
- (pattern)*?, 尽可能少地使用重复的第一个匹配
- (pattern)+?, 尽可能少地使用重复但至少使用一次

例如: "(http://)?(www.)?jeapedu.com"会匹配这四个字符串:

- 'http://www.jeapedu.com'

- 'http://jeapedu.com'
- 'www.jeapedu.com'
- 'jeapedu.com'

举例说明，'w{3:4}.jeapedu.com'能匹配:

- 'www.jeapedu.com'
- 'wwww.jeapedu.com'

colou?r能匹配colour或者color, 简化的匹配电话正则

```
1 import re
2 s = 'telephone to 110-123-114119 or call
     4008-6688-9988, 3ks'
3 res = '\d+\D\d+\D\d+'
4 print re.findall(res, s)
```

输出结果如下所示:

```
['110-123-114119', '4008-6688-9988']
```

这里的d可以换成w么？D能换成W么？

通配符

运用于正则表达式里的一些特殊符号，它可以匹配限定长度的字符串，例如点号可以匹配任意一个字符。

转义字符

正则表达式定义了一些特殊的元字符，如^、$和 . 等。由于这些字符在正则表达式中被解释成其他的指定的意义，如果需要匹配这些字符，则需要使用字符转义来解决这一个问题。转义字符为“\”（反斜杠），它可以取消这些字符（如^、$、. 等）在表达式中的具有的特殊意义。

转义字符	含义
\b	在正则表达式中，表示单词的边界；如果在字符集中，则表示退格符。
\t	制表位
\n、\r、\f	回车、换行、换页
\e	回退(Esc)

*10.4.4 Python*里使用正则

现在我们已经看了一些简单的正则表达式，那么我们实际在Python中是如何使用它们的呢？re 模块提供了一个正则表达式引擎的接口，可以让你将正则表达式(字符串)编译(re.compile 函数)成re模块可以使用、处理的实例对象并用它调用各种函数来进行正则匹配。

一旦有了已经编译了的正则表达式的实例对象,就可以实例对象调用match、search、findall、split 等函数进行正则匹配了。

```
1  import re
2  s = "hello www.jeapedu.com"
3  print '--------------------------'
4  res = "[\s\.]"
5  pat = re.compile(res)
6  print  pat.findall(s)
7  print  pat.split(s)
8  print '--------------------------'
9  s = "hello www.jeapedu.com"
10 res = "[\s\.]"
11 print  re.findall(res, s)
12 print  re.split(res, s)
13 print '--------------------------'
```

运行结果如下所示:

```
--------------------------
[' ', '.', '.']
['hello', 'www', 'jeapedu', 'com']
--------------------------
[' ', '.', '.']
['hello', 'www', 'jeapedu', 'com']
--------------------------
```

也可以不编译res直接用re模块调用findall函数。

第 11 章. MySQL for Python

11.1 MySQL-Python安装

11.1.1 MySQL-Python插件

Python里操作MySQL数据库，需要Python下安装访问MySQL数据库接口API 包，即插件，从而使得Python2.7能访问操作MySQL 数据库。读者可以从下边的网址获取插件并安装。

```
http://sourceforge.net/projects/mysql-python/files/
```

11.1.2 MySQL数据库

安装MySQL数据库。

11.1.3 Python里访问MySQL数据库

① fetchall访问表里数据示例如下:

```
1  import MySQLdb
2  #connect to a database 'test'
3  conn=MySQLdb.connect(host="localhost",user="root",
       passwd="123456",db="test")
4  cursor = conn.cursor()
5  #fetch datas
6  n = cursor.execute("select * from tbtest")
7  r = cursor.fetchall()
8  print , rn
9  #closing database
10 cursor.close()
11 conn.close()
```

② insert向表里插入数据示例如下:

```
1  import MySQLdb
2  #connect to a database 'test'
```

```
3  conn=MySQLdb.connect(host="localhost",user="root",
       passwd="123456",db="test")
4  cursor = conn.cursor()
5  #insert data into table 'tbtest'
6  sql = """insert into tbtest(id, name) values(6, "
       xjeapedud")"""
7  n = cursor.execute(sql)
8  conn.commit()#below mysql 5.0 needded
9  #fetch datas
10 n = cursor.execute("select * from tbtest")
11 r = cursor.fetchall()
12 print r
13 #closing database
14 cursor.close()
15 conn.close()
```

输出结果如下所示:

```
((1L, 'jeapedu'), (4L, 'ajeapedu'), (2L, 'bjeapedu'
   ), (3L, 'cjeapedu'), (5L, 'cjeapedud'), (6L, '
   xjeapedud'))
```

③ update修改表里的数据示例如下:

```
1  import MySQLdb
2  #connect to a database 'test'
3  conn=MySQLdb.connect(host="localhost",user="root",
       passwd="123456",db="test")
4  cursor = conn.cursor()
5  #insert data into table 'tbtest'
6  sql = """update tbtest set name = 'Jeapedu' where
       id = 6"""
7  n = cursor.execute(sql)
8  conn.commit()#below mysql5.0 need this statement
9  #fetch datas
10 n = cursor.execute("select * from tbtest")
11 r = cursor.fetchall()
12 print r
13 #closing database
14 cursor.close()
15 conn.close()
```

输出结果如下

```
((1L, 'jeapedu'), (4L, 'ajeapedu'), (2L, 'bjeapedu'
   ), (3L, 'cjeapedu'), (5L, 'cjeapedud'), (6L, '
   Jeapedu'))
```

④ delete删除表里记录示例如下:

```
1  import MySQLdb
2  #connect to a database 'test'
3  conn=MySQLdb.connect(host="localhost",user="root",
      passwd="123456",db="test")
4  cursor = conn.cursor()
5  #insert data into table 'tbtest'
6  sql = """delete from tbtest where id = 5"""
7  n = cursor.execute(sql)
8  conn.commit()#below mysql5.0 need this statement
9  #fetch datas
10 n = cursor.execute("select * from tbtest")
11 r = cursor.fetchall()
12 print r
13 print n
14 #closing database
15 cursor.close()
16 conn.close()
```

输出结果如下

```
((1L, 'jeapedu'), (4L, 'ajeapedu'), (2L, 'bjeapedu'
   ), (3L, 'cjeapedu'), (6L, 'Jeapedu'))
```

第 12 章. Python网络编程

12.1 urllib编程

12.1.1 urlpen方法

urllib模块使我们可以像读取本地文件一样读取www和ftp上的数据，可以对URL字符串进行格式化处理。先看一个例子，这个例子把Google首页的html抓取下来并显示在控制台上:

```
1 import urllib
2 response = urllib.urlopen('http://www.google.com.hk/')
3 html = response.read()
4 print html
```

别惊讶，整个程序确实只用了两行代码就可以把Google网页所有内容下载下来。!

12.1.2 urlencode方法

urllib库里面有个urlencode函数，可以把keyvalue这样的键值对转换成我们想要的格式，返回的是a=1&b=2这样的字符串。如果只想对一个字符串进行urlencode 转换，怎么办？urllib提供另外一个函数: quote()

```
1 #coding:utf-8
2 import urllib
3 response = urllib.urlopen('http://www.google.com.hk/')
4 html = response.read()
5 print html
6 import urllib,urllib2
7 url = 'http://www.baidu.com/s?'
8 values = {'wd' : 'jeapedu',
9 'pn' : '30',
10 'tn' : 'baiduhome_pg',
11 'ie' : 'utf-8'}
12 data = urllib.urlencode(values)
13 print 'data = ',data
14 print url + data
```

输出结果如下所示:

```
data =  tn=baiduhome_pg&ie=utf-8&wd=jeapedu&pn=30
http://www.baidu.com/s?tn=baiduhome_pg&ie=utf-8&wd=
   jeapedu&pn=30
```

```
1 #coding:utf-8
2 import urllib
3 s = "智普教育"
4 print "encode, "urllib.quote(s)
```

输出结果如下所示:

```
encode,  %E6%99%BA%E6%99%AE%E6%95%99%E8%82%B2
```

相反的操作可以用urldecode函数来解码

```
1 #coding:utf-8
2 #on windows
3 import urllib
4 s = "%E6%99%BA%E6%99%AE%E6%95%99%E8%82%B2"
5 su = urllib.unquote(s)
6 su = su.decode('UTF-8').encode('GBK')
7 print 'decode, ', su
```

输出结果如下所示:

```
decode, 智普教育
```

12.1.3 urlretrieve方法

urlretrieve方法直接将远程数据下载到本地。参数filename指定了保存到本地的路径（如果未指定该参数，urllib会生成一个临时文件来保存数据）；参数reporthook是一个回调函数，当连接上服务器、以及相应的数据块传输完毕的时候会触发该回调。我们可以利用这个回调函数来显示当前的下载进度，下面的例子会展示。参数data指post到服务器的数据。该方法返回一个包含两个元素的元组(filename, headers)，filename表示保存到本地的路径，header表示服务器的响应头。

12.1.4 使用urllib模块下载mp3

```
1 import urllib2,urllib
2 url = 'http://zhangmenshiting.baidu.com/data2/music
   /44148422/730073318000256.mp3?xcode=6
   d5d6ca1b1604d012b576fb4854dfb585e770a7610358a8a'
3 r = urllib2.urlopen(url)
```

```
4 data = r.read()
5 with open("1234t_urllib2.mp3", "wb") as code:
6     code.write(data)
7 urllib.urlretrieve(url, 'test.mp3')
```

*12.2 urllib2*编程

HTTP的访问过程就是一来一回的，python提供的urllib2很方便发起访问请求获得相应的内容，request可以向指定url发出请求而urlopen可以获取返回信息再用read就可把内容读到字符串里处理了。

*12.2.1 HTTP*请求报文格式

向web服务器发送访问请求，实际上就是把HTTP请求报文发送给Web服务器，HTTP请求报文格式如下[1]:

- 请求行请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如，GET /index.html HTTP/1.1。
- 请求头部请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端请求的信息，典型的请求头有：User-Agent：产生请求的浏览器类型。Accept：客户端可识别的内容类型列表。Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。
- 空行最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。
- 请求数据请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

*12.2.2 HTTP*响应报文格式

- 状态行状态行格式如下:
  HTTP-Version Status-Code Reason-Phrase CRLF
  其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase 表示状态代码的文本描述。状态代码由三位数字组成，第一个数字定义了响应的类别，且有五种可能取值。
  ↪ 1xx：指示信息–表示请求已接收，继续处理。
  ↪ 2xx：成功–表示请求已被成功接收、理解、接受。
  ↪ 3xx：重定向–要完成请求必须进行更进一步的操作。
  ↪ 4xx：客户端错误–请求有语法错误或请求无法实现。
  ↪ 5xx：服务器端错误–服务器未能实现合法的请求。
  常见状态代码、状态描述的说明如下。
  ↪ 200 OK：客户端请求成功。

[1] http://hi.baidu.com/our_poll/item/2492adab346e49f115329bf4

↪ 400 Bad Request：客户端请求有语法错误，不能被服务器所理解。
↪ 401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
↪ 403 Forbidden：服务器收到请求，但是拒绝提供服务。
↪ 404 Not Found：请求资源不存在，举个例子：输入了错误的URL。
↪ 500 Internal Server Error：服务器发生不可预期的错误。
↪ 503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常，举个例子：HTTP/1.1 200 OK（CRLF）。

- 响应头部请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端请求的信息，典型的请求头有：User-Agent：产生请求的浏览器类型。Accept：客户端可识别的内容类型列表。Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。
- 空行最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。
- 响应数据请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

下面是一个响应报文示例

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 122< >

html< >
head< >
titleWrox <Homepage/>title<
/>head< >
body<
!-- body goes here >--<
/>body<
/>html
```

12.2.3 urlopen

urllib2模块里也有urlopen函数，可对指定的Url发出请求，urlopen把从服务器返回信息存储在一个response对象里。这个response是一个类似像file的对象，这意味着你能用.read() 函数操作这个response 对象的内容。

```
1 import urllib2
2 response = urllib2.urlopen('http://python.org/')
3 html = response.read()
```

```
4 print html
```

12.2.4 request

Http是基于请求request、响应response的网络协议(客户端请求，服务器端相应)，用requset函数可向构建向某url提出请求响应的对象req,urlopen函数使用构建的request对象从url获得response对象resp。urllib2.Request()的功能是构造一个请求信息，返回的req就是一个构造好的请求。urllib2.urlopen()的功能是发送刚刚构造好的请求req，并返回一个file类型的对象resp，resp包括了所有从服务器响应并返回信息。

```
1 #coding:utf-8
2 import urllib2
3 req = urllib2.Request('http://www.voidspace.org.uk')
4 resp = urllib2.urlopen(req)
5 the_page = resp.read()
6 print '----------------------',the_page
```

12.2.5 request + metadata

客户端在向服务器端提出请求访问时是可以携带一些数据data，例如日常我们上网填写表单的数据就和请求服务器响应时一并传给了服务器，这些请求时携带的数据称之为metadata, 在Python可以构造字典数据结构来实现携带数据模拟。

```
1 import urllib,urllib2
2 url = 'http://www.baidu.com/s?'
3 values = {'wd' : 'jeapedu',
4 'pn' : '30',
5 'tn' : 'baiduhome_pg',
6 'ie' : 'utf-8'}
7 data = urllib.urlencode(values)
8 print 'data',data
9 response = urllib2.urlopen(url+data)
10 print response.read()
```

12.2.6 request+header

上边的数据是可以放在请求的头里的，下面就是将携带数据放入请求头的基本操作。

```
1 import urllib2
2 url = 'http://blog.csdn.net'
3 request = urllib2.Request(url)
4 request.add_header('User-Agent', 'fake-client')
```

```
5 content = urllib2.urlopen(request)
6 print content.read()
7 print content.geturl()
8 headers = content.info()
9 print headers
```

代码第3行构建一个req对象，代码第4行填写请求头(携带数据), 代码第5行向服务器发送携带数据的请求，服务器响应后会返回一些信息urlopen会把服务器响应内容保存在content对象里。

*12.3 httplib2*编程

*12.4 BeautifulSoup4*编程

12.4.1 下载*mm.taobao.com*图片

```
1 import urllib2, urllib
2 from bs4 import BeautifulSoup
3 import time
4 '''
5 #test open a url
6 li = range(1, 10)
7 print li
8 i = 1
9 while i < len(li):
10   url = "http://mm.taobao.com/json/request_top_list.htm
       ?type=2&page="+str(li[i])
11   ret = urllib2.urlopen(url)
12   #print url,i,li[i]
13   i += 1
14 '''
15
16 #test get all_modelers's url in mm.taobao.com
17 url = []
18 def geturl(depth):
19   j = 0
20   global url
21   while j < depth:
22     taobaourl = "http://mm.taobao.com/json/
         request_top_list.htm?type=2&page="+str(j)
23     print taobaourl
24     ret = urllib2.urlopen(taobaourl)
```

```
25      ret = ret.read()
26      soup = BeautifulSoup(ret)
27      tag = soup.findAll('a', {'class':'lady-avatar'})
28      print tag
29      for hr in tag :
30        url.append(hr.get('href'))
31      j += 1
32    #print url
33
34
35  #get a pic
36  def getpic(urlr):
37    urlpic = []
38    picurl = "http://mm.taobao.com/139108254.htm"
39    print 'in getpic', urlr
40    ret = urllib2.urlopen(urlr)
41
42    ret = ret.read()
43    #print ret
44    soup = BeautifulSoup(ret)
45    tag = soup.findAll('img', {'style':'float: none;
         margin: 10.0px;'})
46    if len(tag) == 0:
47      tag = soup.findAll('img', {'style':'margin: 10.0px;
           float: none;'})
48    #tag = soup.findAll('img')
49    print 'tag ',tag
50    uu = "http://img04.taobaocdn.com/imgextra/i4
         /18254030107239481/T1GBD3FiBeXXXXXXXX_!!"
51    for hr in tag :
52        urlpic.append(hr.get('src'))
53    i = 0
54    for urlx in urlpic:
55        un = urlx[len(uu):]
56        #print un
57        un = un[:-4]+str(i)+".jpg"
58        print 'from ',urlr,
59        print ' downloading...',un
60        if len(urlpic[i]):
61          urllib.urlretrieve(urlpic[i], 'pic/' + un)
```

```
62          time.sleep(0.2)
63          i += 1
64
65  def main():
66  #'''
67    d = 10
68    geturl(d)
69    print len(url), url
70  #'''
71  #   getpic("http://mm.taobao.com/276439081.htm")
72  #'''
73    i = 1
74    while i < len(url) :
75      print 'in main', url[i]
76      getpic(url[i])
77      i += 1
78  #'''
79  main()
```