# [如何学习一门新的语言](http://www.cnblogs.com/chgaowei/archive/2011/05/29/2061902.html)

这两天在看《编程人生》，这本书确实非常不错。而且看得也特别的轻松。其中有几个 人都谈到了如何学习新的语言，但是给我最深刻的是google的首席java架构师joshua bloch。正好最近我也在学习python，所以顺便总结一下如何学习一门新的语言。希望你能补充一些。

**心态**

这不但是学习一门新的语言最重要的，而是对任何的学习都是最重要的。下面是书中的描述，非常的精彩，特别是那个比喻：

“学 习一门新的语言的时候，要利用以前所学的语言的功底，但是也要保持开放的心态。有些人执着于一种理念：“这就是写所有程序必须遵循的方法”。我不是说那种 语言，但是某些语言，令人执着于这样的理念。当开始学习新语言的时候，他们会批评这种语言跟真正神的语言的所有的不同之处。当使用新语言时，他们极力使用 神的语言的方法去写。这样，你就会错过这个新语言真正的独特之处。   
这就像你本来只有一个榔头，有人给了你一个螺丝刀，你说“哎，这不是一把好榔头，但是我应该可以倒着拿螺丝刀，用螺丝刀来砸东西。”你得到了一个很烂的榔头，但事实上它确实一把很不错的螺丝刀。所以你应该对所有的事物保持开放和积极的心态。”

如果你的杯子满了，那他永远再也装不进水了。如果你认为你找到了银弹，那么你可能就要固步自封了。

对新的事物，方法保持一个开发而积极的心态，才能真正了解他，了解他的独特之处。

**了解他的历史，哲学观**

选择一门语言，往往选择了一种思维方式和哲学观。所以，了解一门语言的历史和哲学观非常重要。你要知道这门语言是谁创建的，为什么创建，如何发展起来的，适合那些领域，以及解决问题的哲学是什么。

那python来说，他的设计哲学是“用一种方法，最好是只有一种方法来做一件事”，而perl的设计哲学是“总有多种方法来做同一件事”。所以，我选择的是python。

了解这方面的知识的一个非常好的来源是百科网站。

**代码，代码，还是代码**

代码是学习一门语言的必经之路，可能也是最快的一种方法。

你不但要找一些优秀的代码来阅读，还要亲自动手来写代码。这个过程对学习语言来说是非常快的。另外，你一定要用语言去解决实际的问题，而不仅仅是写代码来验证语法。在解决问题的过程中，你可以学习它是如何解决问题的，而且会积累语言的经验。

**社区**

多去这个语言的社区逛逛吧，这里有很多人在讨论这种语言，和他们一起讨论你能够学到更多。

# [Python入门版](http://www.cnblogs.com/BeginMan/p/3192695.html)

一、前言

陆陆续续学习Python已经近半年时间了，感觉到Python的强大之外，也深刻体会到Python的艺术、哲学。曾经的约定，到现在才兑现，其中不乏有很多懈怠，狼狈。

Python入门关于Python的一些基础知识，大部分源自《Python核心编程2》。做这个博客系列一是学习总结，二是自我鞭挞，自我鼓励。

再此愿与所有同仁共勉之。

二、教程目录

1、[Python入门笔记：学习目录](http://www.cnblogs.com/BeginMan/archive/2013/03/11/2954819.html)

2、[Python入门笔记(1)：Python概述](http://www.cnblogs.com/BeginMan/archive/2013/03/11/2954830.html)

3、[Python入门笔记(2)：基础(上)](http://www.cnblogs.com/BeginMan/archive/2013/04/11/3015028.html)

4、[Python入门笔记(3)：基础(下)](http://www.cnblogs.com/BeginMan/archive/2013/04/12/3016323.html)

5、[Python入门笔记(4)：基础快餐版](http://www.cnblogs.com/BeginMan/archive/2013/06/03/3114974.html)

6、[Python入门笔记(5):对象](http://www.cnblogs.com/BeginMan/p/3160044.html)

7、[Python入门笔记(6)：关于数值类型](http://www.cnblogs.com/BeginMan/archive/2013/06/08/3125876.html)

8、[Python入门笔记(7)：序列类型（字符串）](http://www.cnblogs.com/BeginMan/archive/2013/06/08/3125502.html)

9、[Python入门笔记(8):列表](http://www.cnblogs.com/BeginMan/p/3153842.html)

10、[Python入门笔记(9):元组](http://www.cnblogs.com/BeginMan/p/3156235.html)

11、[Python入门笔记(10):字典](http://www.cnblogs.com/BeginMan/p/3156960.html)

12、[Python入门笔记(11):集合](http://www.cnblogs.com/BeginMan/p/3160565.html)

13、[Python入门笔记(12):表达式和语句](http://www.cnblogs.com/BeginMan/p/3164600.html)

14、[Python入门笔记(13):列表解析](http://www.cnblogs.com/BeginMan/p/3164937.html)

15、[Python入门笔记(14):Python的字符编码](http://www.cnblogs.com/BeginMan/p/3166363.html)

16、[Python入门笔记(15):对文件的操作(1)](http://www.cnblogs.com/BeginMan/p/3166644.html)

17、[Python入门笔记(16):对文件的操作(2)](http://www.cnblogs.com/BeginMan/p/3169020.html)

18、[Python入门笔记(17):错误、异常](http://www.cnblogs.com/BeginMan/p/3171445.html)

19、[Python入门笔记(18):Python函数(1):基础部分](http://www.cnblogs.com/BeginMan/p/3171977.html)

20、[Python入门笔记(19):Python函数(2):函数/方法装饰器](http://www.cnblogs.com/BeginMan/p/3173328.html)

21、[Python入门笔记(20):Python函数(3):关于lambda](http://www.cnblogs.com/BeginMan/p/3178103.html)

22、[Python入门笔记(21):Python函数(4):关于函数式编程的内建函数](http://www.cnblogs.com/BeginMan/p/3178363.html)

23、[Python入门笔记(22):Python函数(5):变量作用域与闭包](http://www.cnblogs.com/BeginMan/p/3179040.html)

24、[Python入门笔记(23)：模块](http://www.cnblogs.com/BeginMan/p/3183656.html)

25、[Python入门笔记(24)：Python面向对象(1)速成](http://www.cnblogs.com/BeginMan/p/3190776.html)

26、[Python入门笔记(25)：Python面向对象(2)](http://www.cnblogs.com/BeginMan/p/3191037.html)

27、[Python入门笔记(26)：Python执行环境](http://www.cnblogs.com/BeginMan/p/3191856.html)

# [Python入门笔记：学习目录](http://www.cnblogs.com/BeginMan/archive/2013/03/11/2954819.html)

一、我为什么要学Python?

1.兴趣、好奇心

俗话说：“兴趣是最好的老师！”，有兴趣，好奇心，才能支撑着我从简到难地去学习Python语言。

2.Python的发展前景

2012--2013年编程语言排行榜上地位显赫，且年纪轻轻就如此登峰造极，可见非同小可，何况在国外她已经很火了……而国内经常性的啃别人剩下的馍，对于Python在国内的发展，不可小觑啊。

二、学习参考

1.《Dive Into Python》

2.《Core Python Programming》

3. Python官方文档

三、学习目录

[ 一、Python初级 ]

1.Python概述

2.Python基础(上篇)

3.Python基础(下篇)

4.Python对象(上篇)

5.Python对象(下篇)

6.Python数字

7.Python序列(上篇)

8.Python序列(中篇)

9.Python序列(下篇)

10.Python映像和集合(上篇)

11.Python映像和集合(下篇)

12.Python条件和循环(上篇)

13.Python条件和循环(下篇)

14.Python文件输入输出

15.Python错误和异常

16.Python函数

17.Python模块

18.PythonOOP

19.Python小结

[ 二、Python高级 ]

待续……

# [Python入门笔记(1)：Python概述](http://www.cnblogs.com/BeginMan/archive/2013/03/11/2954830.html)

一、Python特点

Python发展了十五年了，现在正是少年得意之时。它的特点如下：

1.高级编程语言

2.面向对象

3.可升级

所谓的可升级也就是说重用、提升、增强和完善你的代码，远远凌驾于Unix的shell或批处理。在《core python programming》解释：“可升级”这个术语用于衡量硬件的负载，为系统添加新硬件后带来性能的提升；我们试图用于软件，指的是Python提供了 基本的开发模块，你可以在她上面开发你的软件，而且这些需要扩展和增长时，Python的可插入性和模块化架构则是你的项目生机盎然，便于管理。

4.可扩展

怎么扩展：分多文件或多模块，分部处理，同时便于管理；c/c++编写(因为Python的标准实现是使用C语言完成，也称CPython，故用c /c++编写Python扩展)编写Python扩展。Python的java实现被称作Jython,还有IronPython针对.net或Mono 平台的C#实现。

5.可移植性

以为它是C写的，因此具有C的可移植性。

6.简单易学

Python的总体感觉是：关键字少、结构简单、便于阅读

7.健壮性和易维护性

异常处理机制，内存管理由Python解释器负责，这样程序员才不会纠结于内存管理问题，而是致力开发任务。

8.丰富性

各种API、标准库、扩展库和开发插件、工具等，使得Python开发高效、丰富

9.解释性和编译性

Python是一种解释型语言，没有编译这一项。Python实际上是字节编译，其结果生成一个近似机器语言的中间形式；

Python源文件用.py扩展名，当源文件被解释器加载或者显式地进行字节码编译时会被编译成字节码，由于调用解释器的方式不同，源文件会被编译成带有.pyc或pyo扩展名文件。

二、Python和Django环境搭建

见另一篇博文讲解:<http://www.cnblogs.com/BeginMan/archive/2013/03/11/2954346.html>

三、运行Python的三种方法：

1.最简单 的一种就是通过交互式的启动解释器，每输入一行便执行一行如(Python command line)

2.运行Python脚本，调用相关的脚本解释器

3.集成开发环境的图形用户界面，如：Python GUI、eclipse pydev

图形用户[界面](http://baike.baidu.com/view/240704.htm)（Graphical User Interface，简称 GUI，又称图形[用户接口](http://baike.baidu.com/view/1309799.htm)）是指采用图形方式显示的计算机操作用户[界面](http://baike.baidu.com/view/240704.htm)。与早期计算机使用的[命令行界面](http://baike.baidu.com/view/5483485.htm)相比，图形界面对于用户来说在视觉上更易于接受。

**动态类型与静态类型的编程语言：**

编程语言分为动态(类型)语言和静态(类型)语言。

动态语言是指：在运行期间才去做数据类型检查的语言，也就是说，在用动态类型的语言编程时，永远也不用给任何变量指定数据类型，该语言会在第一次赋值给变量时，在内部将数据类型记录下来。Python、Ruby、js就是典型的动态类型语言。

静态类型语言与动态类型语言刚刚相反：它的数据类型是在编译期间检查的，也就是说写程序时要声明所有变量的数据类型，C/C++是静态类型语言的典型代表，其他的静态类型语言有C#、Java等。

# [Python入门笔记(2)：基础(上)](http://www.cnblogs.com/BeginMan/archive/2013/04/11/3015028.html)

一、欲练此功，必先自宫

从现在起，就开始真正地接触她了，或许很多人像我一样，不知道如何去快速掌握一门新的编程语言，今天贴出来一些建议，方便大家看看，这也是学习之前 的十分重要的事情。虽然“欲练此功，必先自宫”太夸张，但是在学习之前，肯定要做好准备了。由于工作中使用Python，在工作前一个月就开始学习并投入 到实际项目中，时隔一个月的学习，今天重温一遍。我做的准备如下：

**1.工具准备**

这个在上一章学习笔记中已经做好了

**2.资料准备**

《Python 核心编程》  
 Python官方文档  
博客园、CSDN等社区

**3.入门教程准备**

大约在六月份之前完成Python入门笔记。

**4.心理准备**

关于如何学好一门新的编程语言，这里推荐阅读：http://www.cnblogs.com/chgaowei/archive/2011/05/29/2061902.html

**5.知识框架拆分准备**

通过对整个Python学习过程中的框架进行拆分，利用《Python核心编程》一书中，几乎涵盖了所有Python知识点，但是对于像我一样的新 手来说，我还是难以掌握她的整个学习框架、流程。如果我们大致看了这一本书，就能细分她的框架，分成模块来学，这样才能做到心中有数。下面是我拆分的框 架：(由于我们先从Python基础学习，所以这里不考虑高级部分)

[1]、Python基础部分：

(1).语法基础、(2)、数据类型、(3).Python对象、(4).字符串、元祖和列表、(5).映射和集合、(6).函数

[2]、流程控制

条件和循环

[3]、文件操作

[4]、模块

[5]、面向对象

[6]、异常处理

[7]、环境部署

好了，这些准备都到位了，现在就开始学习吧，没事的时候建议看看Python哲学

二、Python语法

我不知道自己之前是如何写第一个Python程序的，貌似是照着《Dive Into Python》这本令人在之前学起来蛋疼的教程中写的。一如既往地，咱们从"helloworld"开始看吧，哈哈，我们学的很多语言都是从helloworld开始的。

[复制代码](javascript:void(0);)

>>> 1 + 1

2

>>> print 'hello world'

hello world

>>> x = 1

>>> y = 2

>>> x + y

3

[复制代码](javascript:void(0);)

这里例子并不能说明什么语法，但是能让我们熟悉一些，比如，**Python 既是 动态类型定义语言 (因为它不使用显示数据类型声明) , 又是 强类型定义语言 (因为一旦一个变量具有一个数据类型, 它实际上就一直是这个类型了) 、一句程序结束后并不不以分号结尾、print 可以输出字符串等。那下面就细致讲解下python语法吧**

**1、注释**

和很多Unix脚本类似，Python的注释语句从#字符开始，如：

# print "hello world"

注意Python的#字符注释只能实现单行注释，或许很多人和我一样好奇，如果要注释掉一百多行代码，那么是不是敲一百多次#呢？？毋庸置疑，如果我们这样做，只能说明还是打酱油的雇的帮打酱油的人。

我们可以有如下小技巧：

(1).程序控制法：

在不影响整个程序的情况下，我们可以把一大堆要注释掉的代码段，利用if条件句进行间接注释，也就是if屏蔽了要注释的代码块的执行、可以使用 if False：、if None：、if 0：进行屏蔽处理，如下：

[复制代码](javascript:void(0);)

1 在eclipse中:

2 print 'BeginMan' #输出

3 if False: #屏蔽，if 后还可以是 0、None等一切布尔值为假的条件

4 print 'hello world'

5 print 'good man'

6 print 'sssss'

7 print 'ok!' #输出

[复制代码](javascript:void(0);)

则if 下的代码不会执行，也就没输出了。但是这个技巧呢，最好不要用，可能在很多代码存在的情况下存在潜伏的异常。

(2).编辑器技巧

如果在eclipse下，选中多行，可以使用快捷键ctrl+/注释或取消注释，这个强烈推荐，真是“工欲善其事，必先利其器”啊，对于编辑器，最好是掌握常用的快捷键，才能提高开发效率。

(3).doc string (文档字符串)

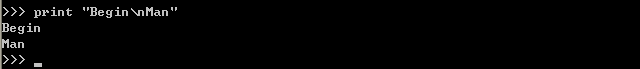
1 def foo(params):

2 """print a,b,c and return a """

三重引号(包含单引号又包含双引号)表示一个多行字符串。在开始与结束引号间的所有东西都被视为单个字符串的一部分, 包括硬回车和其它的引号字符。在三重引号中的任何东西都是这个函数的 doc string, 它们用来说明函数可以做什么。 如果存在 doc string, 它必须是一个函数要定义的第一个内容( 也就是说, 在冒号后面的第一个内容 )。 在技术上不要求给出函数的 doc string, 但应该这样做。 Python 带来一些额外的动机： doc string 在运行时可作为函数的属性。

**2、换行(\n)**

常见的换行符，在学其他编程语言中经常看到。



**3、反斜线(\)继续上一行**

也就是对一行一个语句的隔行，针对一些过长的语句可使用反斜杠\来分解成多行，如：

1 if a>0 and \

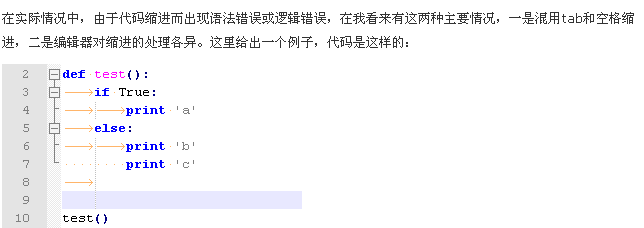
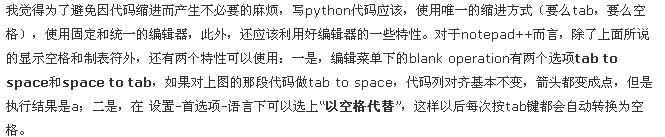
2 b<0 :

3 print 'ok!'

有两种例外情况可以不用反斜杠，一种是：小括号()、中括号[]、花括号{}；另外一种是三引号。针对可读性，不建议使用反斜杠。

**4、代码缩进**

Python 函数没有明显的 begin 和 end, 没有花括号, 用于标函数的开始和结束。 唯一的分隔符是一个冒号 (:), 接着代码本身是缩进的。例子可以参考上面的。注意，码缩进十分严格，如果不按规律办事，不小心的话就会出现语法错误，比如unexpected indent之类的。甚至有时也会出现逻辑错误。

参考：http://www.cnblogs.com/tt-0411/archive/2011/11/11/2245693.html  
  


**5、使用分号(;)可以在同一行书写多个语句**

如；import os; a=123; print a

注意句子末尾不要带;建议不要这样写，因为会大大降低代码的可读性。

**6、模块**

每个python脚本看做一个模块，磁盘的形式存放，可多次拆分成多个模块。使用import 导入。

三、Python的变量

注意，与C#、Java、C/C++等编程语言不一样，Python的变量没有类型，它的类型是在内存中根据对象所确定的。由于涉及到Python函数部分，在本节先不讲，在后面的部分会有讲解。这一节就首先学学变量赋值。

**Python是弱类型，无需明确指定变量类型，赋值的同时会自动声明类型**。如：

[复制代码](javascript:void(0);)

1 >>> anInt=10

2 >>> astr="Car"

3 >>> aFloat=-1.2+3

4 >>> aList=[1,2,3]

5 >>> aTuple=('aa','ss')

6 >>> print anInt,astr,aFloat,aList,aTuple

7 10 Car 1.8 [1, 2, 3] ('aa', 'ss')

[复制代码](javascript:void(0);)

我们需要注意以下几点：

1、在Python中是通过对象的引用而不是值来赋值给变量的。

2、赋值操作符主要是"="，同时也可使用增量赋值，如 x+=1。但是没有自增、自减操作符。

3、在C语言中，赋值语句可以当作表达式(可以返回值)，但是在Python中赋值语句不会返回值，如下面的就是非法的：

>>> x=1

>>> y=(x=x+1)

SyntaxError: invalid syntax

4、Python支持链式赋值、多重赋值、多元赋值

(1).链式赋值:

>>> y=x=x+1

>>> x,y

(2, 2)

(2).多重赋值

>>> x=y=z=1

>>> x,y,z

(1, 1, 1)

值为1的整型对象被创建，该对象的同一引用被赋值给了x,y,z

(3).多元赋值

为什么叫做多元赋值，在Python核心编程书中说，是作者自己其的名称，因为等号两边对象都是元祖。

>>> x,y,z=1,'ssss',True

>>> x,y,z

(1, 'ssss', True)

x,y,z=1,'as',3

(x,y,z)=1,'as',3

(x,y,z)=(1,'as',3) #推荐

其中，Python变量交换实在太优雅了，如果交换x和y的值，在其他语言，如C，我们或许用一个临时变量来完成工作。但是在Python中，你只需这样做就行了

>>> x,y=y,x

>>> x,y

('ssss', 1)

四、标识符

首先变量的命名规则跟其他主流语言如C一样，其次，Python关键字、标识符在Python文档中都有，必要时可翻阅。

 五、编码风格

[复制代码](javascript:void(0);)

1 #coding=utf-8 #编码格式

2 '''

3 Created on 2013-3-11

4 @author: BeginMan

5 '''

6 "this is a test module" #模块文档

7 import os #导入模块

8 import sys

9 s=True #全局变量(globalvariable)

10 class FooClass(object): #类的定义

11 "fooclass test" #类的文档对象

12 classAttribute=10 #类属性

13 def myFunction(self): #函数定义

14 "function test" #函数的文档对象

15 dataAttribute="BeginMan" #数据属性(实例属性)

16 pass

17

18 if \_\_name\_\_=="\_\_main\_\_": #主程序

19 inst=FooClass()

20 pass

21

[复制代码](javascript:void(0);)

 这里我们重点是Python的主程序。那么什么是Python的主程序呢，就是说无论这个模块是被别的模块导入还是作为脚本直接执行，都会执行这部分代码。通常这里不会有太多功能性代码，而是执行模式调用不同函数。

时刻记住一个事实，所有模块都有能力来执行代码，最高级别的Python语句，即没有缩进的代码，在模块被导入时就会执行。

其他：

如：垃圾回收机制，使我们更加专注于代码。

# [Python入门笔记(3)：基础(下)](http://www.cnblogs.com/BeginMan/archive/2013/04/12/3016323.html)

一、数字

在看《Python 核心编程》的时候，我就有点疑问，为什么没有把Python数字放在基础范围之内呢，可能作者先要通过Python对象，然后在深入到数字，这样会更加容易理解。这里有一个观点就是**“Python的一切皆对象”原则**，通过上一节的学习变量赋值，就能深刻体现这个原则，具体见下节的学习入门笔记。

由于数字在Python中用到的并不很多，就像学js的数字类型一样，我觉得最好放在基础篇之内，这样对于下节的对象有个铺垫的作用。

本节主要是学习Python的数字类型、操作、与数字有关的内建函数。

**1、数字类型**

 在上节中说，Python变量无须定义类型，这是由于它是动态类型的语言，在内存中根据所赋值动态地确定其类型。如果偏偏要在python中给变量分类的话，那么可分：**可变类型和不可变类型**。

这里整理了可变类型(mutable)和不可变类型(immutable)的相关知识：  
(或许我们在 学习了下一节才能对此有深刻理解，如果曾学习过C，它像C指针的艺术；如果学了C#，它像C#值类型与引用类型的性质；如果学了js，它的性质基本上同 Python；总的来说，一切的编程语言都围绕着基础核心，只是各自体现不一样而已，我们要学会融会贯通，举一反三。。。不过感觉好难哦。)

这里推荐一篇文章：[《如何学会600多种编程语言》](http://www.csdn.net/article/2013-03-20/2814575-programming-languages) -----------------------------------------------------------------------------------

不 可变类型(immutable):就是内存中的那可内容(value)是否可以被改变，如果是不可变则，在对对象本身操作的时候，必须在内存的另外地方再 申请一块区域(因为老区域#不可变#)，老的区域就丢弃了(如果还有其他ref，则ref数字减1，类似unix下的hard-link)。  
  
可变类型(mutable):对对象操作的时候，不需要再在其他地方申请内存，只需要在此对象后面连续申请(+/-)即可，也就是它的address会保持不变，但区域会变长或者变短。  
  
不可变类型有:string、interger、tuple：

[复制代码](javascript:void(0);)

1 >>> var1 = 1

2 >>> var2 = var1

3 >>> var1,var2

4 (1, 1)

5 >>> id(var1),id(var2) #同指同一内存区域，id相同

6 (21200672, 21200672)

7 >>> var1 += 1

8 >>> var1,var2

9 (2, 1)

10 >>> id(var1),id(var2)  
#由于var1,var2数据不可变的，对var1+1后，申请另外一块区域，id(var1)变了，id(var2)不变,对此区域ref数字-1

11 (21200660, 21200672)

12 >>>

[复制代码](javascript:void(0);)

可变类型：list,dict

[复制代码](javascript:void(0);)

1 >>> lst1 = [1,2,3]

2 >>> lst2 = lst1

3 >>> lst1,lst2

4 ([1, 2, 3], [1, 2, 3])

5 >>> id(lst1),id(lst2)

6 (28497360, 28497360)

7 >>> lst1.append(4)#可变类型的，append后，address会保持不变

8 >>> lst1,lst2

9 ([1, 2, 3, 4], [1, 2, 3, 4])

10 >>> id(lst1),id(lst2)#lst1,lst2一直指向同一区域

11 (28497360, 28497360)

12 >>>

13 在操作这种类型的时候要特别小心，dict的key不用可变类型的，函数传参的时候不用可变类型的。

14 eg:

15 >>> def dis(arg=[]):

16 ... arg.append('1')

17 ... print arg

18 ...

19 >>> dis()

20 ['1']

21 >>> dis()

22 ['1', '1']

23 >>>id(dis())

24 ['1', '1', '1']

25 505246615

26 >>>id(dis())

27 ['1','1','1','1']

28 505246615

[复制代码](javascript:void(0);)

参考：http://blog.chinaunix.net/uid-26249349-id-3080279.html

数字就是不可变类型，也就是说更改数字的值就会产生新的对象，当然这个我们无须考虑。

python数字类型可分为：整型、长整型、布尔型、双精度浮点型、十进制浮点型和复数

**(1)、整型：**

如：0101、8、-3、0X80、-0X78

**(2)、布尔型**

True、False

**(3)、长整型**

在整型后面加L(或l)，目前整型和长整型在慢慢统一。只有对它调用repr()函数才有机会看到L，调用str()函数则看不到L：

[复制代码](javascript:void(0);)

>>> aLong=11111111111111111111

>>> aLong (在命令行中，直接输入变量名显示值，这种情况实际上调用了repr()函数)

11111111111111111111L

>>> print aLong （在命令行中，如果在变量前面使用print 则调用str()函数）

11111111111111111111

>>>

[复制代码](javascript:void(0);)

**(4)、双精度浮点型**

类似C中的double。实际精度依赖Python解释器的编译器。

**(5)、复数**

[复制代码](javascript:void(0);)

1 >>> aComplex=2.1+3j

2 >>> aComplex

3 (2.1000000000000001+3j)

4 >>> aComplex.real

5 2.1000000000000001

6 >>> aComplex.imag

7 3.0

8 >>> aComplex.conjugate()

9 (2.1000000000000001-3j)

10 >>>

[复制代码](javascript:void(0);)

更多见python core programming

**2、操作符**

分别是：+、-、\*、/、%、\*\*、//（地板除）

很有意思的是，python的除法运算。

在js中实现以下代码：

[复制代码](javascript:void(0);)

1 <script type="text/javascript">

2 var a=1,b=2;

3 var c=0.1,d=0.2;

4 var result1=a/b;

5 var result2=c/d;

6 alert(result1);#0.5

7 alert(result2);#0.5

8 </script>

[复制代码](javascript:void(0);)

如果我们在Python中实现，则如下:

>>>1/2

0

>>>0.1/0.2

0.5

有些奇怪吧，Python是这样处理的；

**传统除法**：如果是整型除法，则传统除法会舍弃小数点部分，返回一个整型(地板除)；如果操作数之一是浮点型，则执行真正地除法。

**真正除法**：返回真实的商，在未来Python版本中会成为统一标准，不过现在我们通过from \_\_future\_\_ import division可以实现。

[复制代码](javascript:void(0);)

>>> from \_\_future\_\_ import division

>>> 1/2

0.5

>>> 1.0/2.0

0.5

[复制代码](javascript:void(0);)

**地板除**：从Python2.2开始，新操作符//，被称为地板除、即不管操作数是什么类型，总会舍弃小数部分，返回整数。

[复制代码](javascript:void(0);)

>>> 1//2

0

>>> 1.0//2.0

0.0

>>> -1//2

-1

[复制代码](javascript:void(0);)

其他操作符有兴趣可以在《Python 核心编程》和官网文档看看。

**3、与数字有关的内建函数**

关于更多内建函数，将会在后面学习中具体说明，今天先学习关于数字的内建函数。它们有：

**(1)、标准内建函数：**

cmp()、str()、type()对于任何类型都可以使用

[复制代码](javascript:void(0);)

>>> cmp(1,2) #比较大小

-1

>>> cmp(2,1)

1

>>> cmp(2,2)

0

>>> str(0xFF) #转换成字符串

'255'

>>> type(0xFF) #返回对象类型

<type 'int'>

>>> type('str')

<type 'str'>

[复制代码](javascript:void(0);)

**（2）、转换工厂函数：**

int()、long()、float()、bool()、complex()，它们都是转换成跟数字有关的函数。如：

[复制代码](javascript:void(0);)

>>> '111'

'111'

>>> int('222')

222

>>> long('222')

222L

>>> float('222')

222.0

>>> bool(1)

True

>>> complex(2)

(2+0j)

>>> bool(True)

True

>>> bool(454)

True

>>> bool(0)

False

[复制代码](javascript:void(0);)

**(3)、功能函数**

有abs()返回给定参数的绝对值：

>>> abs(-3)

3

coerce()数据类型对两个参数进行转换函数：

[复制代码](javascript:void(0);)

>>> coerce(1,2)

(1, 2)

>>> coerce(1,-2)

(1, -2)

>>> coerce(1,123L)

(1L, 123L)

>>> coerce(1.3,123L)

(1.3, 123.0)

>>> coerce(1j,123L)

(1j, (123+0j))

[复制代码](javascript:void(0);)

divmod()通过取余，返回商和余数的元祖：

>>> divmod(10,2)

(5, 0)

>>> divmod(10,3)

(3, 1)

pow()和\*\*都可以进行指数运算：

>>> 5\*\*2

25

>>> pow(5,2)

25

round()四舍五入：

[复制代码](javascript:void(0);)

>>> round(3)

3.0

>>> round(3.45)

3.0

>>> round(3.999999,1)

4.0

[复制代码](javascript:void(0);)

更多内建函数见Python官网文档。

# [Python入门笔记(4)：基础快餐版](http://www.cnblogs.com/BeginMan/archive/2013/06/03/3114974.html)

**1、print语句调用str()函数显示，交互式解释器调用repr()函数来显示对象**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> s='python'  >>> s  'python'         #repr()，显示结果呈现单引号  >>> print s    #str().没有单引号  python  >>> repr(s)  "'python'"  >>> str(s)  'python' |

 str()主要显示给人看，repr()显示个机器和畜生看。  
print语句会默认给每一行加上换行符，只要在print语句的最后添加一个逗号(,)就可让结果排列在一行。

**2、raw\_input():**

读取标准输入，并把结果给指定变量，如:name=raw\_input('your name:')

**3、一些语句**

(1)、if、if .. else ..、if ..elif..else..

elif即‘else if ’，注意在Django中不存在 elif 模板标签

（2）、while循环  
循环控制，最好依赖 ..True..Flase，如下：(《DjangoBook第八章例子》)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | #coding=utf-8  '''  Created on 2013-4-17  @author: BeginMan  '''  db={}  def newuser():      prompt='login desired:'      while True:          name=raw\_input(prompt)          if db.has\_key(name):              prompt='name taken,try another'              continue          else:              break      pwd=raw\_input('password:')      db[name]=pwd    def olduser():      name=raw\_input('name:')      pwd=raw\_input('password:')      if pwd==db.get(name):          print 'welecom back ',name      else:          print 'login error'    def showmenu():      prompt="""      -----------------      (N) new user login      (E) existing user login      (Q) quit      -----------------      Enter choice:      """      done=False      while not done:          chosen=False          while not chosen:              try:                  choice=raw\_input(prompt).strip()[0].lower()              except(EOFError,KeyboardInterrupt):                  choice='q'              print '\n you picked:[%s]' %choice              if choice not in 'neq':                  print 'invalid option,try again'              else:                  chosen=True          if choice=='q':done=True          if choice=='n':newuser()          if choice=='e':olduser()    if \_\_name\_\_=='\_\_main\_\_':      showmenu() |

 (3)、for循环

不同C#、java、C、等编程语言，如js中：for(var i=0;i<s.length;i++){....};python中它更像C#中的foreach()：

|  |
| --- |
| >>> dic={'name':'BeginMan','job':'pythoner','age':22}  >>> for obj in dic.items():      print obj    ('age', 22)  ('job', 'pythoner')  ('name', 'BeginMan') |

 （4）、range()/len()使用

这两个方法用的很多，如：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> for obj in range(5):      print obj,    0 1 2 3 4  >>> for obj in [0,1,2,3,4]:      print obj,    0 1 2 3 4 |

 首先了解下range()。它很像JavaScript里面随机函数，在python里也这样称呼。

|  |
| --- |
| >>> help(range)  Help on built-in function range in module \_\_builtin\_\_:    range(...)      range([start,] stop[, step]) -> list of integers        Return a list containing an arithmetic progression of integers.      range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.      When step is given, it specifies the increment (or decrement).      For example, range(4) returns [0, 1, 2, 3].  The end point is omitted!      These are exactly the valid indices for a list of 4 elements. |

 当然，我们也可以这样：

|  |
| --- |
| >>> for obj in range(5,10):      print obj,    5 6 7 8 9 |

 range()经常和len()函数一起使用用于字符串索引，如：

|  |
| --- |
| >>> name='BeginMan'  >>> for obj in range(len(name)):      print '(%d)' %obj,name[obj]    (0) B  (1) e  (2) g  (3) i  (4) n  (5) M  (6) a  (7) n |

 enumerate()的强势围攻，  
上面的例子循环有些约束，Python2.3推出了enumerate()函数来解决这一问题，enumerate:枚举 的意思：

|  |
| --- |
| >>> for i,j in enumerate(name):      print i,j      0 B  1 e  2 g  3 i  4 n  5 M  6 a  7 n  >>> |

**4、列表解析**

**5、文件操作**

打开文件：handle=open(file\_name,access\_mode='r')  
如果打开成功，一个文件对象的句柄将会被返回，就可以通过它的句柄进行一系列的操作。

|  |
| --- |
| filename=raw\_input('Enter file name:')  #filename：对应文件完整路径，这里创建一个py.txt与.py文件同级  fobj=open(filename,'r') #获得文件对象的句柄 fobj  for eachLine in fobj:      print eachLine  fobj.close()    #一次读入文件的所有行然后关闭文件，再迭代每一行的输出 |

# [Python入门笔记(5):对象](http://www.cnblogs.com/BeginMan/p/3160044.html)

一、学习目录

1、pyhton对象

2、python类型

3、类型操作符与内建函数

4、类型工厂函数

5、不支持类型

二、思考

1、Python解释执行原理?

2、Python对象机制?

3、Python哪些常用类型和内建函数?

4、什么是工厂函数?与内建函数的区别?

三、学习重点总结   
资源参考：

1.《Python核心编程 2》

2.互联网

--------------------------------------------------------------------------------------------------------------------------------------------

我承认我的C语言基础比较薄弱，承认自己计算机原理并没有好好学，这才导致了在编程上的一知半解，其实了解底层的东西更加有利于编程的理解。

四、Python中一切皆对象

**1、思考1：Python解释执行原理?**

参考：<http://l62s.iteye.com/blog/1481421>

Python使用对象模型来存储数据，构造任何类型的值都是一个对象。所以说在Python的世界中，一切皆对象，一个整数是一个对象，一个字符串也是一个对象，而且类型也是一个对象，整数类型是一个对象，字符串类型也是一个对象。

**2、思考2：什么是对象模型？**

#### 对象模型技术简介：

对象模型技术（object modeling technique,OMT）是美国通用电气对象模型技术提出的一套系统开发技术。它以[面向对象](http://baike.baidu.com/view/125370.htm)的思想为基础，通过对问题进行抽象，构造出一组相关的模型，这些模型描述了现实世界中“类与对象”以及它们之间的关系，表示了目标系统的静态[数据结构](http://baike.baidu.com/view/9900.htm)，从而能够全面地捕捉问题空间的信息。

#### 对象模型技术特点：

对象模型技术把分析时收到的信息构造在三类模型中，即对象模型、[功能模型](http://baike.baidu.com/view/2135656.htm)和[动态模型](http://baike.baidu.com/view/1465324.htm)。三个模型从不同的角度对系统进行描述，分别着重于系统的一个方面，组合起来构成对系统的完整描述。形象地说，[功能模型](http://baike.baidu.com/view/2135656.htm)定义“做什么”，状态模型定义“何时做”，对象模型定义“对谁做”。

**面向对象三种模型：**

对象模型：描述系统数据结构   
动态模型：描述系统控制结构   
功能模型：描述系统功能

**3、Python对象**

**身份：**唯一性身份标志，是该对象的内存地址，可用内建函数id()获得

**类型：**对象的类型决定了该对象可以保存什么类型的值，可进行什么样的操作，以及遵循什么样的规则。可以type()获得

**值：**对象表示的数据项

[复制代码](javascript:void(0);)

1 >>> s='abc'

2 >>> id(s) #身份

3 13434272

4 >>> type(s) #类型

5 <type 'str'>

6 >>> s #值

7 'abc'

[复制代码](javascript:void(0);)

在对象被创建时就已被赋值，除了值之外，其他两个都是只读的。

对象的可改变性(mutability)：指对象的值是否可以更改。

面向对象：  
**类**：具有同种属性的对象称为类，是个抽象的概念。比如说：汽车、人、狗、神；  
**对象**：日常生活中的所有东西都是对象，是类的实例化。比如说：推土车是汽车的实例化；姚明是人的实例化；小白（狗的名字）是狗的实例化；二郎神是神的实例化；  
**属性**：用来描述具体某个对象的特征的是属性，是静态的。比如：姚明身高2.6米多；小白的毛发是棕色的；二郎神额头上有只眼睛；  
**方法**：每个对象有它们自己的行为或者是使用它们的方法，比如说一只狗会跑会叫等，我们把这些行为称之为方法，是动态的，可以使用这些方法来操作一个对象；  
**类的成员**：属性和方法称为这个对象的成员，因为它们是构成一个对象的主要部分，没有了这两样东西，那么对象也没什么存在意义了。  
参考：http://liuyuru.iteye.com/blog/806705

五、Python类型

**1、标准类型(基本数据类型)**

数字（含子类型）  
Integer:整型  
Boolean：布尔型  
Long integer：长整型  
Floating point real number:浮点型  
String：字符串  
List:列表  
Tuple：元组  
Dictionary:字典

**2、其他类型**

类型  
[Null对象（None）](http://www.cnblogs.com/BeginMan/p/3153983.html)  
文件  
集合/固定集合  
函数/方法  
模块  
类

>>> type(type('abc'))

<type 'type'>

这行代码告诉我们：“所有类型对象的类型都是type，它是python类型的根，所有Python标准类的默认元类”

要弄明白：对象类型(对象的类型)和类型对象(类型本身是对象)。

3、简单了解下内部类型

六、内建函数与工厂行数

1、所谓内建函数（Built-in Functions），个人觉得就是自带的、标准的函数，如：

|  |  | **Built-in Functions** |  |  |
| --- | --- | --- | --- | --- |
| [abs()](http://docs.python.org/2/library/functions.html#abs) | [divmod()](http://docs.python.org/2/library/functions.html#divmod) | [input()](http://docs.python.org/2/library/functions.html#input) | [open()](http://docs.python.org/2/library/functions.html#open) | [staticmethod()](http://docs.python.org/2/library/functions.html#staticmethod) |
| [all()](http://docs.python.org/2/library/functions.html#all) | [enumerate()](http://docs.python.org/2/library/functions.html#enumerate) | [int()](http://docs.python.org/2/library/functions.html#int) | [ord()](http://docs.python.org/2/library/functions.html#ord) | [str()](http://docs.python.org/2/library/functions.html#str) |
| [any()](http://docs.python.org/2/library/functions.html#any) | [eval()](http://docs.python.org/2/library/functions.html#eval) | [isinstance()](http://docs.python.org/2/library/functions.html#isinstance) | [pow()](http://docs.python.org/2/library/functions.html#pow) | [sum()](http://docs.python.org/2/library/functions.html#sum) |
| [basestring()](http://docs.python.org/2/library/functions.html#basestring) | [execfile()](http://docs.python.org/2/library/functions.html#execfile) | [issubclass()](http://docs.python.org/2/library/functions.html#issubclass) | [print()](http://docs.python.org/2/library/functions.html#print) | [super()](http://docs.python.org/2/library/functions.html#super) |
| [bin()](http://docs.python.org/2/library/functions.html#bin) | [file()](http://docs.python.org/2/library/functions.html#file) | [iter()](http://docs.python.org/2/library/functions.html#iter) | [property()](http://docs.python.org/2/library/functions.html#property) | [tuple()](http://docs.python.org/2/library/functions.html#tuple) |
| [bool()](http://docs.python.org/2/library/functions.html#bool) | [filter()](http://docs.python.org/2/library/functions.html#filter) | [len()](http://docs.python.org/2/library/functions.html#len) | [range()](http://docs.python.org/2/library/functions.html#range) | [type()](http://docs.python.org/2/library/functions.html#type) |
| [bytearray()](http://docs.python.org/2/library/functions.html#bytearray) | [float()](http://docs.python.org/2/library/functions.html#float) | [list()](http://docs.python.org/2/library/functions.html#list) | [raw\_input()](http://docs.python.org/2/library/functions.html#raw_input) | [unichr()](http://docs.python.org/2/library/functions.html#unichr) |
| [callable()](http://docs.python.org/2/library/functions.html#callable) | [format()](http://docs.python.org/2/library/functions.html#format) | [locals()](http://docs.python.org/2/library/functions.html#locals) | [reduce()](http://docs.python.org/2/library/functions.html#reduce) | [unicode()](http://docs.python.org/2/library/functions.html#unicode) |
| [chr()](http://docs.python.org/2/library/functions.html#chr) | [frozenset()](http://docs.python.org/2/library/functions.html#func-frozenset) | [long()](http://docs.python.org/2/library/functions.html#long) | [reload()](http://docs.python.org/2/library/functions.html#reload) | [vars()](http://docs.python.org/2/library/functions.html#vars) |
| [classmethod()](http://docs.python.org/2/library/functions.html#classmethod) | [getattr()](http://docs.python.org/2/library/functions.html#getattr) | [map()](http://docs.python.org/2/library/functions.html#map) | [repr()](http://docs.python.org/2/library/functions.html#func-repr) | [xrange()](http://docs.python.org/2/library/functions.html#xrange) |
| [cmp()](http://docs.python.org/2/library/functions.html#cmp) | [globals()](http://docs.python.org/2/library/functions.html#globals) | [max()](http://docs.python.org/2/library/functions.html#max) | [reversed()](http://docs.python.org/2/library/functions.html#reversed) | [zip()](http://docs.python.org/2/library/functions.html#zip) |
| [compile()](http://docs.python.org/2/library/functions.html#compile) | [hasattr()](http://docs.python.org/2/library/functions.html#hasattr) | [memoryview()](http://docs.python.org/2/library/functions.html#func-memoryview) | [round()](http://docs.python.org/2/library/functions.html#round) | [\_\_import\_\_()](http://docs.python.org/2/library/functions.html#__import__) |
| [complex()](http://docs.python.org/2/library/functions.html#complex) | [hash()](http://docs.python.org/2/library/functions.html#hash) | [min()](http://docs.python.org/2/library/functions.html#min) | [set()](http://docs.python.org/2/library/functions.html#func-set) | [apply()](http://docs.python.org/2/library/functions.html#apply) |
| [delattr()](http://docs.python.org/2/library/functions.html#delattr) | [help()](http://docs.python.org/2/library/functions.html#help) | [next()](http://docs.python.org/2/library/functions.html#next) | [setattr()](http://docs.python.org/2/library/functions.html#setattr) | [buffer()](http://docs.python.org/2/library/functions.html#buffer) |
| [dict()](http://docs.python.org/2/library/functions.html#func-dict) | [hex()](http://docs.python.org/2/library/functions.html#hex) | [object()](http://docs.python.org/2/library/functions.html#object) | [slice()](http://docs.python.org/2/library/functions.html#slice) | [coerce()](http://docs.python.org/2/library/functions.html#coerce) |
| [dir()](http://docs.python.org/2/library/functions.html#dir) | [id()](http://docs.python.org/2/library/functions.html#id) | [oct()](http://docs.python.org/2/library/functions.html#oct) | [sorted()](http://docs.python.org/2/library/functions.html#sorted) | [intern()](http://docs.python.org/2/library/functions.html#intern) |

官网参考：http://docs.python.org/2/library/functions.html

2、工厂函数(factory function)

Python2.2统一了类型和类，所有的内建函数现在都也是类，在这基础上，原来的所谓的内建转换函数像int()、type()、list() 等，现在都成了工厂函数。虽然看上去像函数，但实质上它们是类，当你在调用它们时，实际上生成了该类的一个实例，就像工厂生产货物一样。

int()、long()、float()、complex()、str()、unicode()、basestring()、list()、tuple()、type()  
dict()、bool()、set()、frozenset()、object()、classmethod()、staticmethod()、super()、property()、file()

七、不支持类型

1、char与type

2、指针

3、int vs short vs long

4、float vs double

# [Python入门笔记(6)：关于数值类型](http://www.cnblogs.com/BeginMan/archive/2013/06/08/3125876.html)

一、标准类型函数

cmp()：比较大小

str():转换为字符串

type():类型

|  |
| --- |
| cmp(...)      cmp(x, y) -> integer      Return negative(负数) if x<y, zero(0) if x==y, positive（正数） if x>y. |

 如下：

|  |
| --- |
| >>> cmp(5,3.2)  1  >>> cmp(3.5,8)  -1 |

 二、转换工厂函数

存在精度损失

|  |
| --- |
| >>> int(1.847)  1  >>> long(42)  42L  >>> float(42)  42.0  >>> complex(42)  (42+0j)  >>> complex(2.4,-8)  (2.3999999999999999-8j) |

 三、功能函数

用于数值运算：asb()、coerce()、divmod()、pow()、round()

asb():absolute:绝对的；完全的；专制的;n:绝对值

|  |
| --- |
| >>> abs(-1)  1 |

 coerce():vt. 强制，迫使,  
类型转换，但是提供了不依赖python解释器而是通过自定义两个数值类型转换。返回一个元祖，存在强制行为。  
coerce(...)  
    coerce(x, y) -> (x1, y1)  
      
    Return a tuple consisting of the two numeric arguments converted to  
    a common type, using the same rules as used by arithmetic operations.  
    If coercion is not possible, raise TypeError.

|  |
| --- |
| >>> coerce(1,2)  (1, 2)  >>> coerce(1.2,2l)  (1.2, 2.0)  >>> coerce(1.2,2)  (1.2, 2.0)  >>> coerce(1,2.3)  (1.0, 2.2999999999999998)  >>> coerce(1j,123)  (1j, (123+0j)) |

 divmod():.divmod 整除求余、返回包含商和余数的元祖

|  |
| --- |
| >>> divmod(10,3)  (3, 1)  >>> divmod(3,10)  (0, 3)  >>> divmod(10,2.5)  (4.0, 0.0) |

 pow()：power of a number：指数的意思

pow()与\*\*都可以实现指数运算，pow()先出生些。

|  |
| --- |
| >>> pow(2,5)  32  >>> 2\*\*5  32 |

 round（）：四舍五入

round(...)  
    round(number[, ndigits]) -> floating point number  
  
    Round a number to a given precision in decimal digits (default 0 digits).  
    This always returns a floating point number.  Precision may be negative.

|  |
| --- |
| >>> round(1.234,2)  1.23  >>> round(3.14)  3.0  >>> for each in range(10):           print round(math.pi,each)    3.0  3.1  3.14  3.142  3.1416  3.14159  3.141593  3.1415927  3.14159265  3.141592654 |

 四、仅用于整数的函数

oct():octonary number system 八进制字符串形式

|  |
| --- |
| >>> oct(255)  '0377' |

 hex()：hexadecimal number system十六进制字符串形式

|  |
| --- |
| >>> hex(255)  '0xff' |

 ASCII码转换函数

ord（）:ordinal：序数，将字符转换成对应整数值

|  |
| --- |
| >>> ord('A')  65 |

 chr()：char: 单个字符，数字对应当个ASCII字符

|  |
| --- |
| >>> chr(65)  'A' |

 五、操作符

|  |
| --- |
| >>> x>=80 and x<=100  True  >>> 80<=x<=100  True  -----------------------------  总是写错：  >>> 80=<span style="color: #ff0000;"><</span>x<=100  <span style="color: #ff0000;">SyntaxError: invalid syntax  </span> |

 六、致用

1、分数等级

|  |
| --- |
| def result(x):     dic={9:'A',8:'B',7:'C',6:'D'}     myre=x/10     for obj in sorted(dic.keys(),reverse=True):         if myre>= obj:             out=dic[obj]             break         else:             out='F'     return out    if \_\_name\_\_=="\_\_main\_\_":      sorce = input('Enter your sorce:')      print 'level:%s' %result(sorce) |

# [Python入门笔记(7)：序列类型（字符串）](http://www.cnblogs.com/BeginMan/archive/2013/06/08/3125502.html)

一、题外话

之前信誓旦旦说本系列笔记要六月份完成，但是现在几乎不可能了。为什么出现这种状况呢？我这里反思了以下：

1、学习路线不坚定：

一会儿Python、一会儿Django、mysql、jquery等，使得原本的计划被打乱。

2、目标不明确：

我想达到什么样的程度、我想实现什么功能等并不明确，导致涣散、敷衍等不好的行为。

那么接下来就是改正的时候了，还是要多管住自己同时也勉励自己。

二、序列类型

包含字符串、列表、元祖。模式都一样，举一反三即可。如：

1、成员关系操作符(in / not in )

2、关于切片

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | s=[1,2,3,4]  print s[::-1]     #下标范围[0,0],步长是-1，则从后(4,包括4)往前切取所有，输出：[4, 3, 2, 1]  print s[::-2]     #下标范围[0,0],步长是-2，则从后(4，包括4)往前跳过2位切取，输出：[4, 2]  print s[::]        #下标范围[0,0],步长是0，则从前(1,包括1)往后切取所有，输出：[1, 2, 3, 4]  print s[::2]       #下标范围[0,0],步长是2，则从前(1,包括1)往后跳过2位切取，输出：[1, 3]  print s[1:4:2]  #下标范围[1,4],步长是2，则从下标为1(2)到下标为4(4)跳过2位切取，输出：[2, 4] |

 要灵活运用。

三、关于序列类型的内建函数

如list()、tuple()、str()类型转换，实际上是工厂函数，浅copy的结果而并非真正的改头换面(转换)。

注意在string类型上应用list()、tuple()往往并不能得到我们想要的结果。

序列类型的内建函数一览表：

 cmp()、len()、max()、min()、enumerate()、zip()、

四、Unicode字符串

1 >>> 'hello'+u' '+'world'

2 u'hello world'

五、字符串类型的内建方法

1、不常用的string模块

[复制代码](javascript:void(0);)

1 >>> import string

2 >>> string.uppercase

3 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

4 >>> string.lowercase

5 'abcdefghijklmnopqrstuvwxyz'

6 >>> string.whitespace

7 '\t\n\x0b\x0c\r '

8 >>> string.digits

9 '0123456789'

10 >>> string.punctuation

11 '!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~'

[复制代码](javascript:void(0);)

打开这个string.py模块，如下：

[复制代码](javascript:void(0);)

.......................................

# Some strings for ctype-style character classification

whitespace = ' \t\n\r\v\f'

lowercase = 'abcdefghijklmnopqrstuvwxyz'

uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

letters = lowercase + uppercase

ascii\_lowercase = lowercase

ascii\_uppercase = uppercase

ascii\_letters = ascii\_lowercase + ascii\_uppercase

digits = '0123456789'

hexdigits = digits + 'abcdef' + 'ABCDEF'

octdigits = '01234567'

punctuation = """!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~"""

printable = digits + letters + punctuation + whitespace

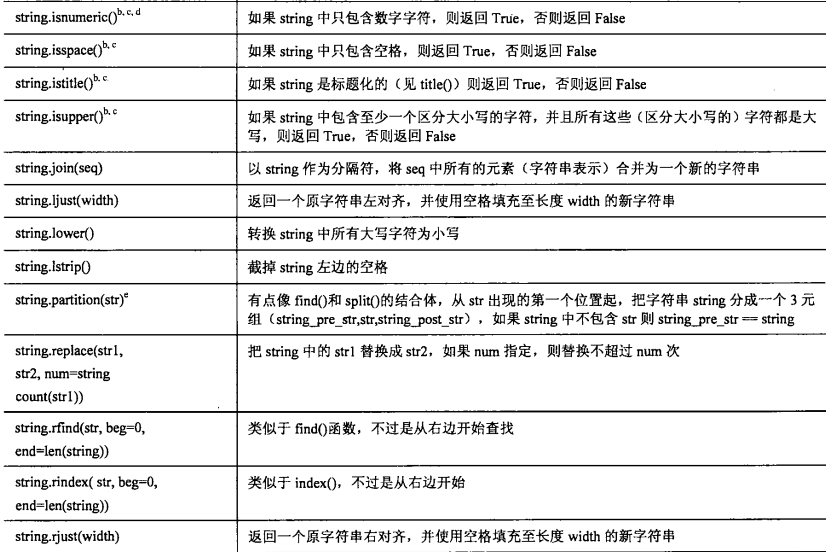
# Case conversion helpers

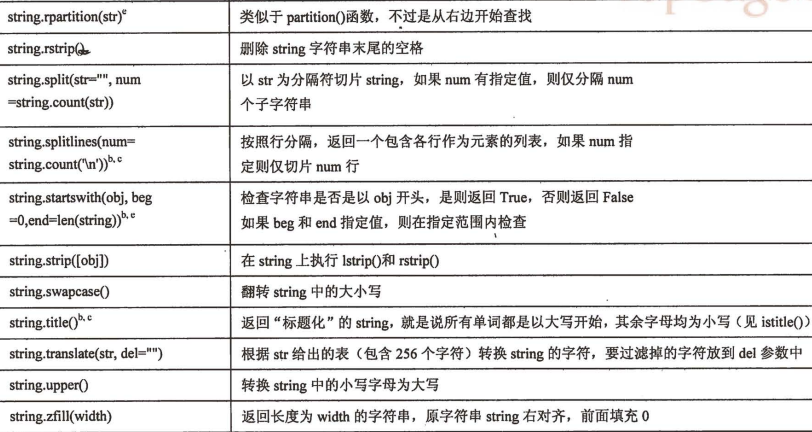
.......................................

[复制代码](javascript:void(0);)

不常用，很多功能可以自己模拟。

2、内建函数  



[join/split:http://www.cnblogs.com/BeginMan/archive/2013/03/21/2972857.html](http://www.cnblogs.com/BeginMan/archive/2013/03/21/2972857.html)

# [Python入门笔记(8):列表](http://www.cnblogs.com/BeginMan/p/3153842.html)

一、序列类型操作符

**1、切片[]和[:]**

**2、成员关系操作符(in ,not in )**

1: s1 = [1,2,3,4,5,6,7]

2: s2 = [2,3,6]

3: s3 = []

4: for obj in s1:

5: if obj not in s2:

6: s3.append(obj)

7:

8: print s3

9: s4 = set(s1)-set(s2)

10: print s4 #set([1, 4, 5, 7])

**3、连接操作符 (+)**

+:   两边连接同类型的，并且连接之后创建一个新的对象，不能将一个元素添加进去，只能是一个list对象+list对象。

1: print s1+s2 #[1, 2, 3, 4, 5, 6, 7, 2, 3, 6]

1: 注意：

2: li=[1,2,3]

3: li.extend('good'),

4: print li 则输出：[1,2,3,'g','o''o','d']

5: 如果我们换成li.append('good')之后，则输出[1,2,3,'good'].

6: 这是因为extend()添加整个列表对象，append()添加元素

注意：

1: >>> li = [1,2,3]

2: >>> li+'new item'

3:

4: Traceback (most recent call last):

5: File "<pyshell#15>", line 1, in <module>

6: li+'new item'

7: TypeError: can only concatenate list (not "str") to list

8: >>> li.extend('item')

9: >>> li

10: [1, 2, 3, 'i', 't', 'e', 'm']

11: >>> li.append('item')

12: >>> li

13: [1, 2, 3, 'i', 't', 'e', 'm', 'item']

14: >>>

**4、重复操作符(\*)**

二、列表类型内建函数

**1、list.append(obj)：向列表中添加一个对象obj**

1: lis = [1,2,3,4,'a','b','c','a','aaa','e','great']

2: lis.append('object')

3: print lis #[1, 2, 3, 4, 'a', 'b', 'c', 'a', 'aaa', 'e', 'great', 'object']

**2、list.count(obj)：返回一个对象obj在列表中出现的次数**

1: #print lis.count('a') #2

**3、list.extend(seq)：把序列seq的内容添加到列表中**

1: lis.extend('seq') #[1, 2, 3, 4, 'a', 'b', 'c', 'a', 'aaa', 'e', 'great', 's', 'e', 'q']

**4、list.index(obj,i=0,j=len(list))：返回list[k]==obj的k值，并且k的范围在 i<=k<j，否则引发ValueError异常**

1: #print lis.index('aaa') #8

**5、list.insert(index,obj)：在索引量为index的位置插入对象obj（必须两个参数）**

1: lis.insert(1,'Z')

2: print lis #[1, 'Z', 2, 3, 4, 'a', 'b', 'c', 'a', 'aaa', 'e', 'great']

**6、list.pop(index = -1)：删除并返回指定位置的对象，默认是最后一个**

1: print lis.pop() #great

2: print lis.pop(1) #2

**7、list.remove(obj)：从列表中删除对象obj**

1: lis = [1,2,3,4,'a','b','c','a','aaa','e','great']

2: #lis.remove('value')

3: #如果不存在则发生异常：

4: #Traceback (most recent call last):

5: # File "E:\project\Python\_test\src\test\dic.py", line 8, in <module>

6: # lis.remove('value')

7: #ValueError: list.remove(x): x not in list

8: lis.remove('a')

9: #如果有多个相同的值，则只删除最前面的一个：[1, 2, 3, 4, 'b', 'c', 'a', 'aaa', 'e', 'great']

10: lis.remove('great') #[1, 2, 3, 4, 'b', 'c', 'a', 'aaa', 'e']

11:

12: print lis

**8、list.reverse()：原地翻转列表**

1: lis = [1,2,3,4,'a','b','c','a','aaa','e','great']

2: lis.reverse()

3: print lis #['great', 'e', 'aaa', 'a', 'c', 'b', 'a', 4, 3, 2, 1]

**9、list.sort()**

1: lis = [1,2,3,4,'a','b','c','a','aaa','e','great']

2: #lis.sort(cmp=None, key=None, reverse=False)

3: lis.sort(cmp=None, key=None, reverse=True)

4: print lis #['great', 'e', 'c', 'b', 'aaa', 'a', 'a', 4, 3, 2, 1]

**注意：那些可以改变对象值的可变对象的方法是没有返回值的**

如：

1: lis = [1,2,3,4,'a','b','c','a','aaa','e','great']

2: print lis.extend('MM') #None

3: result = lis.extend('MM')

4: print result #None

这些操作会在列表中原地执行操作，也就是说现有的列表内容会被改变，但是没有返回值。与之相反，字符串方法有返回值：

1: print 'abc'.upper() #ABC

其中与序列有关的内建函数，则可有返回值：

1: print reversed(lis) #<listreverseiterator object at 0x0128DAD0>

2: print sorted(lis) #[1, 2, 3, 4, 'M', 'M', 'M', 'M', 'a', 'a', 'aaa', 'b', 'c', 'e', 'great']

三、序列类型函数

1、len()

2、max() min()

3、sorted()、reversed()：注意字符串排序时使用字典序，而不是字母序

4、enumerate()、zip()

见：<http://www.cnblogs.com/BeginMan/archive/2013/03/14/2959447.html>

5、sum()

6、list()、tuple()

# [Python入门笔记(9):元组](http://www.cnblogs.com/BeginMan/p/3156235.html)

一、元组特性

**1**、类似列表，但不可变类型，正因如此，它可以做一个字典的key  
**2、**当处理一组对象时，这个组默认是元组类型（老写错"元祖"）  
**3、**所有的多对象，逗号分隔的，没有明确用符号定义的这些都默认为元组类型

[复制代码](javascript:void(0);)

1 >>> 'abc',1,2,'x',True

2 ('abc', 1, 2, 'x', True)

3 >>> x,y =1,2

4 >>> x,y

5 (1, 2)

6 -----------------------------

7 def foo1():

8 return obj1,obj2,obj3

9

10

11 def foo2():

12 return [obj1,obj2,obj3]

13

14 def foo3():

15 return (obj1,obj2,obj3)

[复制代码](javascript:void(0);)

**4、**逗号不要少，尽量都写，即使只有一个元素

[复制代码](javascript:void(0);)

>>> type(('x'))

<type 'str'>

>>> type(('x',))

<type 'tuple'>

>>> tup = ('x')

>>> type(tup)

<type 'str'>

>>> tup = ('x',)

>>> type(tup)

<type 'tuple'>

[复制代码](javascript:void(0);)

觉得不错：hi.baidu.com/wuxinzy/item/c2c3cd428c99aa01896d10a7  
  
1.   无法向元组添加元素。元组没有  append() 或 extend() 方法。    
2.   不能从元组中删除元素。元组没有 remove() 或 pop()  方法。    
3.   可以 在元组中查找元素，由于该操作不改变元组。    
4.   还可以使用 in 运算符检查某元素是否存在于元组中。 那么元组有什么好处呢？   
  
•   元组的速度比列表更快。如果定义了一系列常量值，而所需做的仅是对它进行遍历，那么请使用元组替代列表。   
 •   对不需要改变的数据进行“ 写保护” 将使得代码更加安全。使用元组替代列表就像是有一条隐含的 assert 语句显示该数据是  
•   一些元组可用作字典键（特别是包含字符串、数值和其它元组这样的不可变数据的元组）。列表永远不能当做字典键使用，因为列表不是不可变的。

# [Python入门笔记(10):字典](http://www.cnblogs.com/BeginMan/p/3156960.html)

一、映射类型

我理解中的映射类型是：键值对的关系，键(key)映射值(value),且它们是一对多的关系。  
字典是Python唯一的映射类型。

**扩展1：哈希表**  
一种数据结构，值是根据相关的键进行数据存储的，形成"键-值对"(key-value pairs),哈希表中的值是没有顺序的。

**扩展2：映射类型与序列类型的区别**

1):访问方式不同，序列类型用数字类型的键，而映射类型可以用其他对象类型做键(一般式字符串)

[复制代码](javascript:void(0);)

>>> lis = ['a','b','c']

>>> lis[2]

'c'

>>> dic = {'name':'a','father':'b','mother':'c'}

>>> dic['mother']

'c'

[复制代码](javascript:void(0);)

2):存储类型不同，映射类型的键，直接或间接地与值相关。  
3):序列类型，有序之列；映射类型则无序之列

二、字典

**1、工厂方法dict()：**

[复制代码](javascript:void(0);)

>>> tu=(['a','b'],['xx','yy'])

>>> tu

(['a', 'b'], ['xx', 'yy'])

>>> fdict = dict(tu)

>>> fdict

{'a': 'b', 'xx': 'yy'}

[复制代码](javascript:void(0);)

**2、访问形式：**

[复制代码](javascript:void(0);)

>>> dic

{'father': 'b', 'name': 'a', 'mother': 'c'}

>>> for obj in dic:

print obj

father

name

mother

>>> for obj in dic.keys():

print obj

father

name

mother

>>> for obj in dic.values():

print obj

b

a

c

>>> for obj in dic.items():

print obj

('father', 'b')

('name', 'a')

('mother', 'c')

>>>

[复制代码](javascript:void(0);)

**3、has\_key()、in、not in来检查是否有某个键，has\_key()已慢慢弃用了。**

**4、有则更新，无则添**

**5、删除**

del dic['name'] #删除元素

del dic #删除整个

**6、操作符**  
[]、和 in、not in

>>> if 'name' in dic:

print dic['name']

a

**7、相关函数**  
dict()：创建字典  
len():返回键值对数目  
hash(obj):返回obj的哈希值

**8、内建方法**  
dict.clear()：删除字典中所有元素  
dict.copy():浅copy  
dict.formkeys()：创建字典  
dict.get(key,default=None):返回对应键值  
dict.has\_key():键是否存在  
dict.items()：返回字典中键值对元祖的列表  
dict.keys():键的列表dict.values():值的列表  
......

[复制代码](javascript:void(0);)

>>> dic.keys()

['father', 'name', 'mother']

>>> dic.values()

['b', 'a', 'c']

>>> dic.get('name')

'a'

[复制代码](javascript:void(0);)

 三、注意

**1、**不允许一个键对应多个值，一个键只能对应一项  
**2、**当键发生冲突时，取最后一个。  
**3、**Python不会检查键的冲突，也不会因为键的冲突而产生错误，如果检查每个键是否冲突势必会占用很多内存。

>>> dic={'a':'ss','a':"xxxx"}

>>> dic

{'a': 'xxxx'}

**4、**键必须是可哈希的。  
所有不可变类型都是可哈希的，不可变类型如（列表、字典）则不能。  
不可变类型：string,integer,tuple、  
可变类型：list,dict  
  
**5、**值相同的数字表示相同的键，如1和1.0的哈希值是相同的，因此它们是相同的键。

>>> dic={1:'a',1.0:'b'}

>>> dic

{1: 'b'}

# [Python入门笔记(11):集合](http://www.cnblogs.com/BeginMan/p/3160565.html)

一、目录

1、集合概述

2、关于集合的操作符、关系符号

3、集合的一系列操作(添加、更新、访问、删除)

4、关于集合的内建函数、内建方法

5、小结

二、集合概述

集合(set)：把不同的元素组成一起形成集合，是python基本的数据类型。

集合元素(set elements):组成集合的成员

1 >>> li=['a','b','c','a']

2 >>> se =set(li)

3 >>> se

4 set(['a', 'c', 'b'])

集合对象是一组无序排列的可哈希的值：集合成员可以做字典的键

[复制代码](javascript:void(0);)

>>> li=[['a','b','c'],['a','c']]

>>> se = set(li)

Traceback (most recent call last):

File "<pyshell#29>", line 1, in <module>

se = set(li)

TypeError: list objects are unhashable

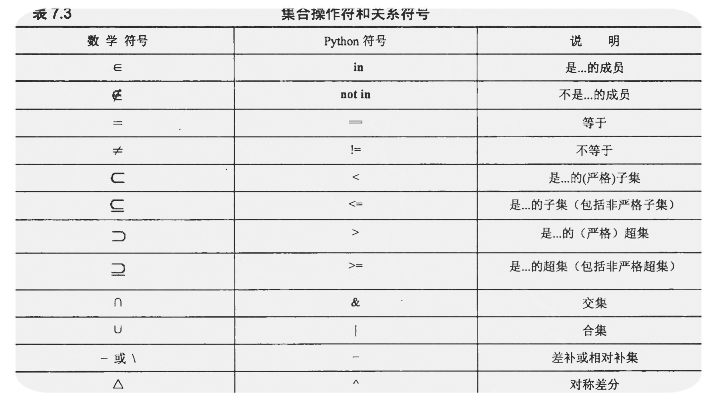
[复制代码](javascript:void(0);)

集合分类：可变集合、不可变集合

可变集合(set)：可添加和删除元素，非可哈希的，不能用作字典的键，也不能做其他集合的元素

不可变集合(frozenset)：与上面恰恰相反

集合操作符与关系符号：（忘完了!）



三、集合的相关操作

**1、创建集合**

由于集合没有自己的语法格式，只能通过集合的工厂方法set()和frozenset()创建

[复制代码](javascript:void(0);)

>>> s = set('beginman')

>>> s

set(['a', 'b', 'e', 'g', 'i', 'm', 'n'])

>>> t = frozenset('pythonman')

>>> t

frozenset(['a', 'h', 'm', 'o', 'n', 'p', 't', 'y'])

>>> type(s),type(t)

(<type 'set'>, <type 'frozenset'>)

>>> len(s),len(t)

(7, 8)

>>> s==t

False

>>> s=t

>>> s==t

True

>>>

[复制代码](javascript:void(0);)

**2、访问集合**

由于集合本身是无序的，所以不能为集合创建索引或切片操作，只能循环遍历或使用in、not in来访问或判断集合元素。

[复制代码](javascript:void(0);)

>>> 'a' in s

True

>>> 'z' in s

False

>>> for i in s:

print i

a

h

m

o

n

p

t

y

>>>

[复制代码](javascript:void(0);)

**3、更新集合**

可使用以下内建方法来更新：

s.add()  
s.update()  
s.remove()

注意只有可变集合才能更新：

[复制代码](javascript:void(0);)

>>> s.add(0)

Traceback (most recent call last):

File "<pyshell#46>", line 1, in <module>

s.add(0)

AttributeError: 'frozenset' object has no attribute 'add'

>>> type(s)

<type 'frozenset'>

>>> se = set(s)

>>> se

set(['a', 'h', 'm', 'o', 'n', 'p', 't', 'y'])

>>> type(se)

<type 'set'>

>>> se.add(0)

>>> se

set(['a', 0, 'h', 'm', 'o', 'n', 'p', 't', 'y'])

>>> se.update('MM')

>>> se

set(['a', 0, 'h', 'm', 'o', 'n', 'p', 'M', 't', 'y'])

>>> se.update('Django')

>>> se

set(['a', 0, 'D', 'g', 'h', 'j', 'm', 'o', 'n', 'p', 'M', 't', 'y'])

>>> se.remove('D')

>>> se

set(['a', 0, 'g', 'h', 'j', 'm', 'o', 'n', 'p', 'M', 't', 'y'])

>>>

[复制代码](javascript:void(0);)

del：删除集合本身

 四、集合类型操作符

**1、in ,not in**  
**2、集合等价与不等价(==, !=)**  
**3、子集、超集(见上表)**

>>> set('shop')<set('cheeshop')

True

>>> set('bookshop')>=set('shop')

True

**4、联合(|)**

联合(union)操作与集合的OR操作其实等价的，联合符号有个等价的方法，union()。

[复制代码](javascript:void(0);)

>>> s1=set('begin')

>>> s2=set('man')

>>> s3=s1|s2

>>> s3

set(['a', 'b', 'e', 'g', 'i', 'm', 'n'])

[复制代码](javascript:void(0);)

>>> s1.union(s2)

set(['a', 'b', 'e', 'g', 'i', 'm', 'n'])

**但+ 运算则不适合：**

[复制代码](javascript:void(0);)

>>> s3New = s1+s2

Traceback (most recent call last):

File "<pyshell#68>", line 1, in <module>

s3New = s1+s2

TypeError: unsupported operand type(s) for +: 'set' and 'set'

[复制代码](javascript:void(0);)

**5、交集(&)**

与集合AND等价，交集符号的等价方法是intersection()

>>> s1&s2

set(['n'])

>>> s1.intersection(s2)

set(['n'])

**6、查补(-)**  
等价方法是difference()

>>> s1-s2

set(['i', 'b', 'e', 'g'])

>>> s1.difference(s2)

set(['i', 'b', 'e', 'g'])

**7、对称差分(^)**

对称差分是集合的XOR(‘异或’)，取得的元素属于s1,s2但不同时属于s1和s2.其等价方法symmetric\_difference()

>>> s1^s2

set(['a', 'b', 'e', 'g', 'i', 'm'])

>>> s1.symmetric\_difference(s2)

set(['a', 'b', 'e', 'g', 'i', 'm'])

**注意：集合之间and,or**

[复制代码](javascript:void(0);)

>>> s1 and s2

set(['a', 'm', 'n']) #取 s2

>>> s1 or s2

set(['i', 'b', 'e', 'g', 'n']) #取 s1

>>>

[复制代码](javascript:void(0);)

五、集合、列表、元组、字符串之间转换

[复制代码](javascript:void(0);)

>>> list(s1)

['i', 'b', 'e', 'g', 'n']

>>> str(s1)

"set(['i', 'b', 'e', 'g', 'n'])"

>>> tuple(s1)

('i', 'b', 'e', 'g', 'n')

[复制代码](javascript:void(0);)

**应用：**

'''最简单的去重方式'''

lis = [1,2,3,4,1,2,3,4]

print list(set(lis)) #[1, 2, 3, 4]

六、关于集合的内建函数、内建方法

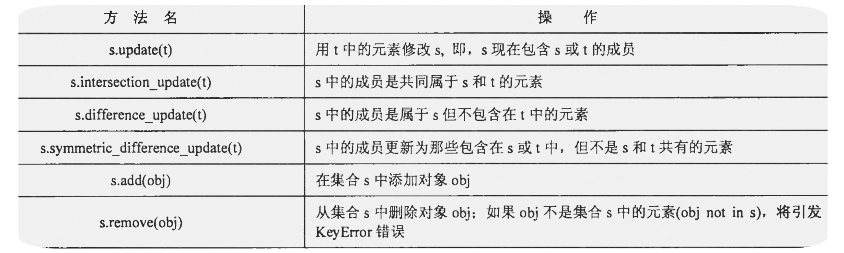
**1、len():返回集合元素个数**

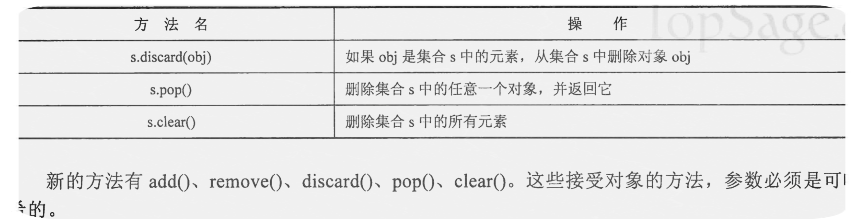
**2、set()、frozenset()工厂函数**

**3、所有集合方法：**



**4、仅适合可变集合**





# [Python入门笔记(12):表达式和语句](http://www.cnblogs.com/BeginMan/p/3164600.html)

一、Python语句

if语句、else语句、elif语句、条件表达式、while语句、for语句、break语句、continue语句、pass语句、Iterators(迭代器)、列表解析

二、常用语句小结

**1、if语句可以通过布尔操作符and、or和not实现多重判断条件或否定判断条件**

**2、if...elif...elif...else..**

**3、三元操作符：Python2.5后：X if C else Y**

[复制代码](javascript:void(0);)

x,y=4,3

if x>y:

s = y

else:

s= x

print s

[复制代码](javascript:void(0);)

**或许更加简洁的是：**

x,y = 4,3

s = (x<y and [x] or [y])[0]

**在Python2.5更新后：**

x,y = 4,3

**s = x if x<y else y**

**4、while：**

[复制代码](javascript:void(0);)

count = 0

while(count<9):

.....

或者:

while True:

.....

#用的挺多的。

[复制代码](javascript:void(0);)

**5、for循环**

[复制代码](javascript:void(0);)

s = ['a','b','c','d','e']

#No1：项

for obj in s:

print obj, #a b c d e

print range(len(s)) #[0, 1, 2, 3, 4]

#No2：索引

for obj in range(len(s)):

print s[obj], #a b c d e

[复制代码](javascript:void(0);)

配合len()、range()使用，使用range()能得到迭代对象的索引数的列表。

[复制代码](javascript:void(0);)

#No3:使用项和索引迭代

for i,eachline in enumerate(s):

print i,eachline

#0 a

#1 b

#2 c

#3 d

#4 e

[复制代码](javascript:void(0);)

enumrate()：返回两个对象，一个是下标索引(%d),另一个是下标索引对应的元素(%s)

**6、break、continue**

break：结束当前循环，跳到下一条语句。  
continue：终止当前循环，忽略剩余的部分，然后回到循环的顶端，在开始执行下一次迭代前，验证是否通过，然后再进行下一次的迭代。

[复制代码](javascript:void(0);)

pwd='abc'

count = 0

while count<3:

inp =raw\_input('Please input password:')

if inp == pwd:

print 'ok'

break

else:

count += 1

# if count<3:

# print 'you have %d times to input password' %(3-count)

# else:

# print 'game over'

#精简：x if ..else y

s = 'you have %d times to input password' %(3-count) if count<3 else 'game over'

print s

[复制代码](javascript:void(0);)

**7、pass**

pass:是一个很好的占位符，不做任何事情。

注意：编写代码时，最好先别结构定下来，如果不想让一些代码干扰，那么最好的方法就是使用pass

**8、for、while与else的联合使用**

其他语言中，else只能用于if条件句，但是Python不同其他语言，**else还能与for、while一起使用。在循环后处理，并且如果遇到break，则也会跳过else的。**

[复制代码](javascript:void(0);)

def showMaxFactor(num):

count = num / 2

while count > 1:

if num % count == 0:

print u'%d的最大公约数是: %d' %(num,count)

break

count -= 1

else:

print num,u'是素数'

for eachNum in range(10,21):

showMaxFactor(eachNum)

[复制代码](javascript:void(0);)

# [Python入门笔记(13):列表解析](http://www.cnblogs.com/BeginMan/p/3164937.html)

一、列表解析

列表解析来自函数式编程语言(haskell),语法如下：

[expr for iter\_var in iterable]

[expr for iter\_var in iterable if cond\_expr]

第一种语法：首先迭代iterable里所有内容，每一次迭代，都把iterable里相应内容放到iter\_var中，再在表达式中应用该iter\_var的内容，最后用表达式的计算值生成一个列表。

第二种语法：加入了判断语句，只有满足条件的内容才把iterable里相应内容放到iter\_var中，再在表达式中应用该iter\_var的内容，最后用表达式的计算值生成一个列表。

例子如下：

[复制代码](javascript:void(0);)

>>> seq = [11,10,9,9,5,35,8,20,31,72,54,53]

>>> filter(lambda x:x%2,seq)

[11, 9, 9, 5, 35, 31, 53]

>>> [x for x in seq if x%2]

[11, 9, 9, 5, 35, 31, 53]

[复制代码](javascript:void(0);)

更复杂的如求矩阵：

[复制代码](javascript:void(0);)

#3行5列的矩阵

>>> [(x+1,y+1) for x in range(3) for y in range(5)]

[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5)]

#简单的3行3列

>>> [(x,y) for x in range(3) for y in range(3)]

[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]

#模拟

>>> for x in range(3):

for y in range(3):

print x,y

0 0

0 1

0 2

1 0

1 1

1 2

2 0

2 1

[复制代码](javascript:void(0);)

计算一个文件的字数和大小：

[复制代码](javascript:void(0);)

>>> import os

>>> f = open('demo.txt','r')

>>> #计算单词的个数

>>> len([word for line in f for word in line.split()])

1942

>>> #计算文件大小

>>> os.stat('demo.txt').st\_size

14051L

>>> f.seek(0) #seek()函数回到文件头部，因为迭代器已经访问完了文件的所有行

>>> sum([len(word) for line in f for word in line.split()])

10806

>>> f.close()

>>>

[复制代码](javascript:void(0);)

 经典例子：

[复制代码](javascript:void(0);)

books=[

{"name":u"C#从入门到精通","price":23.7,"store":u"卓越"},

{"name":u"ASP.NET高级编程","price":44.5,"store":u"卓越"},

{"name":u"Python核心编程","price":24.7,"store":u"当当"},

{"name":u"JavaScript大全","price":45.7,"store":u"当当"},

{"name":u"Django简明教程","price":26.7,"store":u"新华书店"},

{"name":u"深入Python","price":55.7,"store":u"新华书店"},

]

[复制代码](javascript:void(0);)

1、书籍信息中价格最低的

[复制代码](javascript:void(0);)

#No1:原始方式：

>>> price=[]

>>> for item in books:

for p in item:

if p == 'price':

price.append(item[p])

>>> min(price)

23.699999999999999

#No2:列表解析：

>>> min([item[p] for item in books for p in item if p=='price'])

23.699999999999999

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

#Python相关书籍检索

for item in books:

for p in item:

if item['name'].find('Python')>=0:

print item[p], #24.7 Python核心编程 当当 55.7 深入Python 新华书店

total = [item[p] for item in books for p in item if item['name'].find('Python')>=0]

print total #[24.699999999999999, u'Python\u6838\u5fc3\u7f16\u7a0b', u'\u5f53\u5f53', 55.700000000000003, u'\u6df1\u5165Python', u'\u65b0\u534e\u4e66\u5e97']

[复制代码](javascript:void(0);)

# [Python入门笔记(14):Python的字符编码](http://www.cnblogs.com/BeginMan/p/3166363.html)

一、字符编码中ASCII、Unicode和UTF-8的区别

[点击阅读：http://www.cnblogs.com/kingstarspe/p/ASCII.html](http://www.cnblogs.com/kingstarspe/p/ASCII.html)

[再推荐一篇相关博文：http://www.cnblogs.com/huxi/archive/2010/12/05/1897271.html](http://www.cnblogs.com/huxi/archive/2010/12/05/1897271.html)

二、Unicode与ASCII

Python能处理Unicode和ASCII编码，为了让这两者看起来尽可能的相似，Python字符串从原来简单的类型改成了真正的对象。ASCII字符串成了StringType、Unicode字符串成了UnicodeType。使用如下：

[复制代码](javascript:void(0);)

>>> "hello world" #ASCII string

'hello world'

>>> u"hello world" #Unicode string

u'hello world'

>>>

[复制代码](javascript:void(0);)

1、str()、chr()只能以0~255作为参数，也即是说只处理ASCII字符串。如果有Unicode字符串，则会先自动转换成ASCII的然后在传入这些函数中。

原因：Unicode支持的字符多，如果在str()、chr()中有ASCII不存在的字符，则会发生异常。

2、unicode()、unichar()可以看做是Unicode版本的str()和chr()。

>>> unicode('hello world')

u'hello world'

三、编码与解码

它们解决的问题就是编码(encode())、解码(decode())问题，不至于出现乱码。

Codec表示编码方式。

[复制代码](javascript:void(0);)

""" 把一个Unicode字符串写入到磁盘文件，然后再把它读出并显示;

写入的时候用UTF-8，读也一样用UTF-8。"""

CODEC = 'utf-8'

FILE = 'demo.txt'

strIn = u'BeginMan will be a great coder'

byte\_strIn = strIn.**encode**(CODEC) #以uft-8进行编码

f = open(FILE,'w')

f.write(byte\_strIn)

f.close()

f = open(FILE,'r')

str = f.read()

f.close()

str\_out = str.**decode**(CODEC) #以utf-8进行解码

print str\_out #输出：BeginMan will be a great coder

[复制代码](javascript:void(0);)

注意：

1、程序中出现字符串时一定要在前面加上前缀u

s= '博客园Cnblog' #不要这样写,这样容易乱码如：鍗氬鍥瑿nblog

s = u'博客园Cnblog'#正确

2、不要使用str()函数，尽量用unicode()代替

3、不要使用过时的string 模块

4、没必要在程序中编码或解码unicode字符串，编码解码一般用于操作文件、数据库、网络等才使用。

5、字符串格式化

[复制代码](javascript:void(0);)

>>> '%s %s' %('Begin','man')

'Begin man'

#还记得上次的关于字符串的博客中说到的：“普通字符串与unicode字符串能转换成unicode字符串”

>>> u'%s %s' %(u'Begin',u'Man')

u'Begin Man'

>>> u'%s %s' %('Begin','Man')

u'Begin Man'

>>> '%s %s' %(u'Begin','man')

u'Begin man'

>>> '%s %s' %('Begin',u'man')

u'Begin man'

[复制代码](javascript:void(0);)

# [Python入门笔记(15):对文件的操作(1)](http://www.cnblogs.com/BeginMan/p/3166644.html)

一、文件对象

我理解的文件对象就是一个接口，通过这个接口对文件进行相关操作。

《Python 核心编程》上说的很晦涩，这里没有深刻理解到，希望有人能解释给我听。

[复制代码](javascript:void(0);)

>>> f = open('demo.txt','r')

>>> f

<open file 'demo.txt', mode 'r' at 0x00CCCEC0>

>>> type(f)

<type 'file'>

[复制代码](javascript:void(0);)

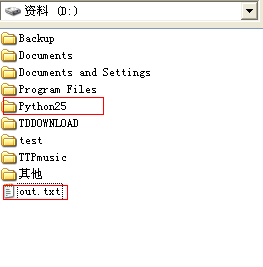
二、相关函数

**[1]、内建函数:open()**

提供了初始化输入/输出(I/O)操作的通用接口，成功打开一个文件后会返回一个文件对象，否则发生IOError异常。语法如下：

file\_object = open(file\_name,access\_mode = 'r',buffering = -1)

分析如下：

**1):file\_name:表示要打开文件名字的字符串，可以是相对路径也可以是绝对路径。**  
如：我在D盘新建一个空白名为out的文本文件，然后进行相关操作。  


>>> f = open('../out.txt','w')

>>> f.write('good')

>>> f.close()

则打开该文件后，发现'good'已经写入进去了。

接下来使用绝对路径进行访问，如下：

>>> f = open('D:/out.txt','r')

>>> f.read()

'good'

>>> f.close()

**2、可选变量access\_mode也是字符串，代表文件打开模式。**



注意：

1):使用'r'或'U'模式打开的文件必须已经存在

2):'w'模式打开的文件若存在则先清空，然后重新创建

3):'a'模式打开的文件是为了追加数据做准备的，所有写入的数据将追加到文件末尾。如果文件不存在，则自动被创建。

4):'+'代表可读可写

5):'b'代表二进制模式访问

6):默认情况下是 'r'

3、buffering可选参数，用于指示访问文件所采用的缓存方式。0表示不缓存；1表示只缓存一行，n代表缓存n行。如果不提供或为负数，则代表使用系统默认的缓存机制。

用处？这个麻烦有人能告诉我！

**关闭文件：**

打开文件消耗系统资源, 并且依赖于文件模式, 其它程序或许不能访问它们。这就是一旦操作完毕就该关闭文件的重要所在。

接上面的操作，加入还没有关闭：

[复制代码](javascript:void(0);)

>>> f.closed #判断是否关闭

False

>>> f.close() #关闭方法

>>> f.closed

True

>>> f.tell() #如果已关闭，则一切针对文件操作都无效

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

f.tell()

ValueError: I/O operation on closed file

>>>

[复制代码](javascript:void(0);)

**[2]、工厂函数file()**

与open()一样，可相互替换

>>> f = file('../out.txt','r')

>>> f.read()

'good'

>>> f.close()

三、相关方法

返回一个文件对象后（句柄），其实我也不清楚到底什么是句柄，这里摘自百度百科：

句柄，是整个windows编程的基础。一个句柄是指使用的一个唯一的整数值，即一 个四字节长的数值，来标志应用程序中的不同对象和同类对象中的不同的实 例，诸如，一个窗口，按钮，图标，滚动条，输出设备，控件或者文件等。应用程序能够通过句柄访问相应的对象的信息，但是句柄不是一个[指针](http://baike.baidu.com/view/159417.htm)，程序不能利用句柄来直接阅读文件中的信息。如果句柄不用在I/O文件中，它是毫无用处的。 句柄是windows用来标志应用程序中建立的或是使用的唯一整数，windows使用了大量的句柄来标志很多对象。

一般与文件相关的方法无非是:输入、输出、文件内移动、杂项

**[1]、输入**

1、read():读取字节到字符串中，有可选参数size，默认是-1，如果为-1或复数则文件将会被读取到文件末尾。

>>> f.read()

'good'

>>> f.read(2)

'go'

2、readline():读取文件的一行，包括行结束符。同read()也有个可选参数size。

3、readlines()：读取所有（剩余的）然后将它们作为字符串列表返回，它有个可选参数sizhint代表返回的最大字大小。

>>> f = file('../out.txt','r')

>>> f.readlines()

['good']

**[2]、输出**

1、write():把含有文本数据或二进制数据块的字符串写入到文件中去。

2、writelines()：针对列表操作，接受一个字符串列表作为参数，将它们写入文件，行结束符并不会被自动加入，如果需要的话必须在调用writelines方法前给每一行结尾加上结束符。

[复制代码](javascript:void(0);)

>>> f = open('d:/out.txt','w')

>>> f.writelines(['man'])

>>> f.close()

>>> f = open('d:/out.txt','r')

>>> f.read()

'man'

[复制代码](javascript:void(0);)

**【注意：】**

当使用read()或者readlines()从文件中读取行时，Python并不会删除行结束符，这个操作留给了程序员。

[复制代码](javascript:void(0);)

>>> f.readlines()

['game is your life,game is your life,game is your life,game is your life,game is your life,[\*]**\n**', 'game is your life,game is your life,game is your life[]']

---------------------------

>>> f = open('demo.txt','r')

>>> data = [line.strip() for line in f.readlines()]

>>> data

['game is your life,game is your life,game is your life,game is your life,game is your life,[\*]', 'game is your life,game is your life,game is your life[]']

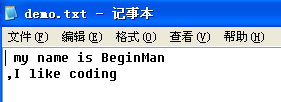
[复制代码](javascript:void(0);)

类似的，write()和writelines()也不会自动加入行结束符，应该自己添加。

>>> f = open('demo.txt','w')

>>> f.writelines([' my name is BeginMan\n,I like coding'])

>>> f.close()

打开该文件可查看：  


**[3]、文件内移动**

tell()、seek()方法

[复制代码](javascript:void(0);)

f = open('d:/out.txt','rb')

print f.tell() #[1]

f.seek(10,1) #[2]

print f.tell() #10

print f.read(10) # BeginMan,

print f.read(18) #I like coding.

print f.tell() #38

"""

【1】：tell():

tell 方法确认了已经移到当前文件位置

一个文件对象维护它所打开文件的状态。文件对象的 tell 方法告诉你在打开文件中的当前位置。

因为我们还没有对这个文件做任何事，当前位置为 0，它是文件的开始处。

"""

"""

【2】：seek():

文件对象的 seek 方法在打开文件中移动到另一个位置。第二个参数指出第一个参数是什么意思：

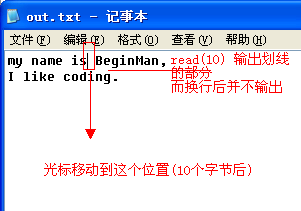
0 表示移动到一个绝对位置 （从文件开始算起），

1 表示移到一个相对位置 （从当前位置算起），

还有 2 表示文件末尾

"""

[复制代码](javascript:void(0);)



**[4]、文件迭代**

[复制代码](javascript:void(0);)

>>> f = open('d:/out.txt','rb')

>>> for eachline in f:

eachline

'my name is BeginMan,\r\n'

'I like coding.\r\n'

[复制代码](javascript:void(0);)

 四、相关属性

file.closed:文件已被关闭，否则为False

file.mode:文件访问模式

file.name:文件名称

>>> f.mode

'r'

>>> f.name

'demo.txt'

# [Python入门笔记(16):对文件的操作(2)](http://www.cnblogs.com/BeginMan/p/3169020.html)

一、文件系统

从系统角度来看，文件系统是对文件[存储器](http://baike.baidu.com/view/87697.htm)空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。

**Python对文件系统的操作是通过os模块实现**。提供了访问操作系统功能的接口，但是这个os模块只是一个伪文件系统操作，真正的“模块”如，dos，os2等，这些是直接的底层工作，而Python的os模块只是一种借用操作(个人理解)

os.path模块可对路径进行相关操作。

os、os.path这两个模块与平台、操作系统无关。

 二、os模块的文件/目录访问函数

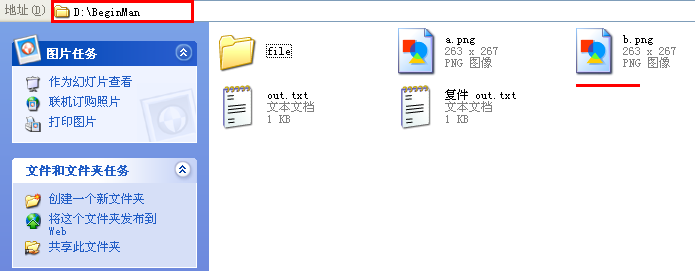
|  |  |
| --- | --- |
| **函数** | **描述** |
| **文件处理** |  |
| mkfifo()/mknod() | 创建命名管道/创建文件系统节点 |
| remove(path)/unlink(path) | 删除文件 如：os.remove('d:\\b.png') |
| rename(old, new)/renames(old, new) | 重命名文件：如：os.rename('d:\\b.png','d:\\a.png') |
| stat(path) | 返回文件信息：  >>> os.stat('d:\\a.png')  (33206, 0L, 0, 0, 0, 0, 4088L, 1372746141, 1372734171, 1372746141)  >>> |
| symlink() | 创建符号链接 |
| utime() | 更新时间戳 |
| tmpfile() | 创建并打开('w+b')一个新的临时文件，注意不含参数，一旦执行则创建一个临时文件，类型未知  >>> os.tmpfile()#创建临时文件  <open file '<tmpfile>', mode 'w+b' at 0x011D4BF0>  >>> os.rename('d:\\t58g.2','d:\\abc.txt')#改成txt |
| walk(top, topdown, onerror) | 生成一个目录树下所有文件名 |
| **目录/文件夹** |  |
| chdir(path) | 改变当前工作目录 |
| listdir(path) | 列出指定目录文件  >>> os.listdir('d:\\')  ['a.png', 'abc.txt', 'out.txt', 'Program Files', 'Python25', 'RECYCLER',]>>> |
| getcwd()/getcwdu() | 返回当前目录：  >>> os.getcwd()  'D:\\Python25' >>> os.getcwdu() u'D:\\Python25 |
| mkdir(path)/makedirs(name, mode) | 创建目录/创建多层目录  >>> os.mkdir('D:\\file')  http://images.cnitblog.com/blog/476998/201307/03114657-fc463944f89c40a29938a1ead2d7c84a.png |
| rmdir(path)/removedirs(name) | 删除目录/删除多层目录 |
| **访问/权限** |  |
| access(path, mode) | 验证权限目录 |
| chmod(path, mode) | 更改权限模式 |
| **文件描述符操作** |  |
| open()、read()、write() | 见上一节 |
| dup()/dup2() | 复制 |

 其他的见《Python 核心编程2》9.7节

**应用：**

（1）：检查下磁盘中(D)是否有文件"BeginMan"的文件夹，如果存在则新建"SuperMan",  
            在该目录下创建一个名为"readme.txt"的文本文件，写入数据，然后输出文件内容  
（2）：如果文件夹BeginMan存在，则遍历该目录，并输入目录文件，  
            如果该目录下存在文件，且如果有文件名为b.png，则重命名为boy.png，移到D盘根目录下  
  (3)：删除BeginMan目录下所有文件，只保留子目录

文件系统图如下：



D盘中并没有SuperMan的文件夹，这里程序如下：

[复制代码](javascript:void(0);)

#coding=utf-8

import os,shutil

import datetime

DIR = 'D:\\'

FILENAME = 'BeginMan'

PATH = 'D:\\%s' %FILENAME

CONTENT = """

（1）：检查下磁盘中(D)是否有文件"BeginMan"的文件夹，如果存在则新建"SuperMan",\n

在该目录下创建一个名为"readme.txt"的文本文件，写入数据，然后输出文件内容\n

（2）：如果文件夹BeginMan存在，则遍历该目录，并输入目录文件\n，

如果该目录下存在文件，且如果有文件名为b.png，则重命名为boy.png，移到D盘根目录下\n

(3)：删除BeginMan目录下所有文件，只保留子目录

==============================================\n

时间：%s\n

作者：%s\n

""" %(datetime.datetime.now(),FILENAME)

if FILENAME in os.listdir(DIR):

print u'已经有同名文件：%s' %FILENAME

for obj in os.listdir(PATH):

print u'%s\\%s\n' %(PATH,obj)

try:

if os.path.isfile(u'%s\\%s' %(PATH,obj)):

if obj == 'b.png': #如果含有文件为b.png，则重命名为boy.png,移动到该目录上级

os.rename(u'%s\\%s' %(PATH,obj),u'%s\\%s' %(PATH,'boy.png'))

shutil.move(u'%s\\%s' %(PATH,'boy.png'), 'D:\\')

os.remove(u'%s\\%s' %(PATH,obj))#删除文件,只保留子目录

except Exception,e:

print 'The error:%s' %e

FILENAME = 'SuperMan'

os.mkdir('D:\\%s' %FILENAME) #创建目录

f = open('D:\\%s\\readme.txt'%FILENAME,'w' ) #在该目录下创建文件

f.write(CONTENT) #写入数据

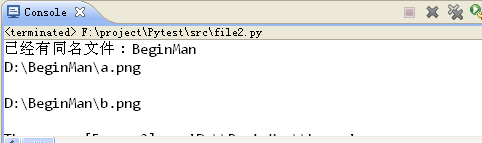
f.close()

f = open('D:\\%s\\readme.txt'%FILENAME,'r' )

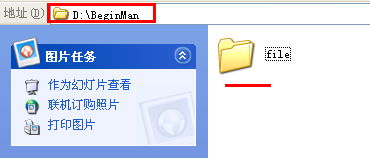
print [lines.strip() for lines in f] #输出

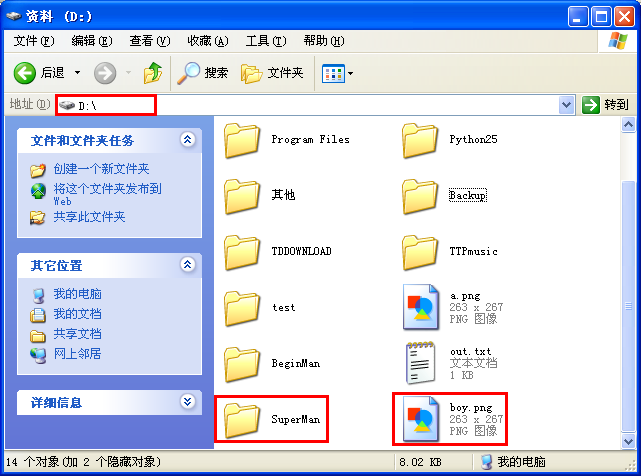
[复制代码](javascript:void(0);)

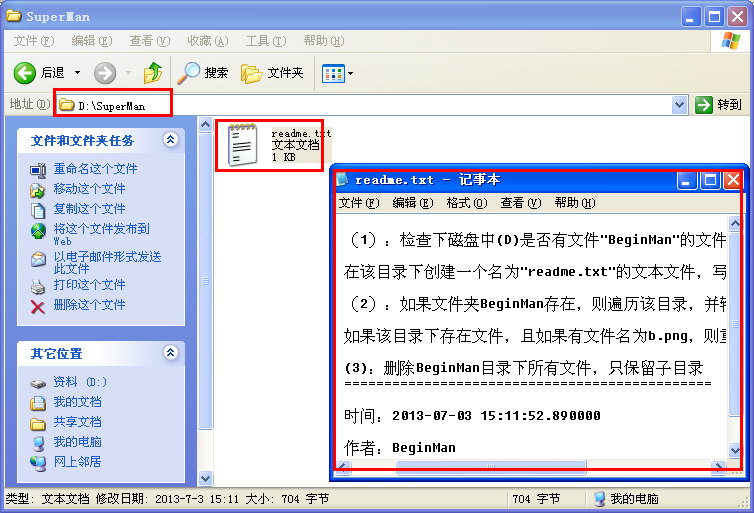
运行如下：



效果如下：







**Python有关文件、文件夹操作小结：**

import os,shutil #要用到这两个模块

**创建文件：**

1) os.mknod("test.txt") 创建空文件

2) open("test.txt",w) 直接打开一个文件，如果文件不存在则创建文件

**创建目录：**

os.mkdir("file") 创建目录

**复制文件：**

shutil.copyfile("oldfile","newfile") oldfile和newfile都只能是文件

shutil.copy("oldfile","newfile") oldfile只能是文件夹，newfile可以是文件，也可以是目标目录

**复制文件夹：**

shutil.copytree("olddir","newdir") olddir和newdir都只能是目录，且newdir必须不存在

**重命名文件（目录）**

os.rename("oldname","newname") 文件或目录都是使用这条命令

**移动文件（目录）**

shutil.move("oldpos","newpos")

**删除文件**

os.remove("file")

**删除目录**

os.rmdir("dir") 只能删除空目录

shutil.rmtree("dir") 空目录、有内容的目录都可以删

**转换目录**

os.chdir("path") 换路径

**判断目标**

os.path.exists("goal") 判断目标是否存在

os.path.isdir("goal") 判断目标是否目录

os.path.isfile("goal") 判断目标是否文件   
参考：http://www.cnblogs.com/phoebus0501/archive/2011/01/19/1939646.html

 三、os.path

如下表：

|  |  |
| --- | --- |
| **函数** | **描述** |
| 分隔 |  |
| basename(p) | 去掉目录路径, 返回文件名:  os.path.basename('D:\\Project\\python25\\Lib\\a.txt') #a.txt |
| dirname(p) | 去掉文件名, 返回目录路径:  os.path.dirname(D:\\Project\\python25\\Lib\\a.txt)  #D:\Project\python25\Lib |
| join() | 将分离的各部分组合成一个路径名:Django项目中常用，  [复制代码](javascript:void(0);)  settings.py  DIRNAME = os.path.dirname(\_\_file\_\_)  TEMPLATE\_DIRS = (  os.path.join(DIRNAME, "themes"),  )  [复制代码](javascript:void(0);) |
| split() | 返回 (dirname(), basename()) 元组:  >>>DIRNAME = 'D:\\Project\\python25\\Lib\\a.txt'  >>>os.path.split(DIRNAME)  ('D:\\Project\\python25\\Lib', 'a.txt') |
| splitdrive() | 返回 (drivename, pathname) 元组:  os.path.splitdrive(DIRNAME)  #('D:', '\\Project\\python25\\Lib\\a.txt') |
| splitext() | 返回 (filename, extension) 元组  os.path.splitext(DIRNAME)  #('D:\\Project\\python25\\Lib\\a', '.txt') |
| 信息 |  |
| getatime(filename) | 返回最近访问时间：  [复制代码](javascript:void(0);)  gTime = os.path.getatime(os.getcwd())  print gTime #1372836299.77  tTime = time.localtime(gTime)  print time.strftime('%Y-%m-%d %H:%M:%S',tTime) #2013-07-03 15:24:59  [复制代码](javascript:void(0);) |
| getctime(filename) | 返回文件创建时间：  cTime = os.path.getctime(os.getcwd())  c\_time = time.localtime(cTime)  print time.strftime('%Y-%m-%d %H:%M:%S',c\_time) #2013-04-17 17:27:46 |
| getmtime(filename) | 返回最近文件修改时间 |
| getsize(filename) | 返回文件大小(以字节为单位) |
| 查询 |  |
| exists() | 指定路径(文件或目录)是否存在 |
| isabs() | 指定路径是否为绝对路径 |
| isdir() | 指定路径是否存在且为一个目录 |
| isfile() | 指定路径是否存在且为一个文件 |
| islink() | 指定路径是否存在且为一个符号链接 |
| ismount()  samefile() | 指定路径是否存在且为一个挂载点  两个路径名是否指向同个文件 |

# [Python入门笔记(17):错误、异常](http://www.cnblogs.com/BeginMan/p/3171445.html)

一、什么是错误，什么是异常，它们两者区别

这里解释如下：个人觉得很通俗易懂

错误是指在执行代码过程中发生的事件，它中断或干扰代码的正常流程并创建异常对象。当错误中断流程时，该程序将尝试寻找异常处理程序(一段告诉程序如何对错误做出响应的代码)，以帮助程序恢复流程。换句话说，错误是一个事件，而异常是该事件创建的对象。

当使用短语“产生异常”时，表示存在问题的方法发生错误，并创建异常对象(包含该错误的信息及发生的时间和位置)来响应该错误。导致出现错误和随后异常的因素包括用户错误、资源失败和编程逻辑失败。这些错误与代码实现特定任务的方法有关，而与该任务的目的无关。

如果不进行异常处理，即不对错误做出响应，程序的健壮性就会大打折扣，甚至无法保证正常运行，所以必须要进行异常处理。  
原文：http://blog.163.com/zhaojinyong2@126/blog/static/279729562008719384580/

Python的异常处理能力是很强大的，可向用户准确反馈出错信息。在Python中，异常也是对象，可对它进行操作。所有异常都是基类 Exception的成员。所有异常都从基类Exception继承，而且都在exceptions模块中定义。Python自动将所有异常名称放在内建 命名空间中，所以程序不必导入exceptions模块即可使用异常。一旦引发而且没有捕捉SystemExit异常，程序执行就会终止。如果交互式会话 遇到一个未被捕捉的SystemExit异常，会话就会终止。  
Python采用的是”尝试(try)块“、和”捕获(catch)块“。

二、Python中的异常

**1、NameError：尝试访问一个未声明的变量**

[复制代码](javascript:void(0);)

>>>foo

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

foo

NameError: name 'foo' is not defined

[复制代码](javascript:void(0);)

如果请求的名字没有在任何名称空间里找到，则产生一个NameError异常。

**2、ZeroDivisionError：除数为0**

[复制代码](javascript:void(0);)

>>> 1/0

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

1/0

ZeroDivisionError: integer division or modulo by zero

[复制代码](javascript:void(0);)

**3、SyntaxError:Python解释器语法错误**

>>> for

SyntaxError: invalid syntax

>>>

SyntaxError异常是唯一一个不是在程序执行时发生的异常，代表一个不正常的结构，一般在编译时发生。  
  
**注意**：Python是解释型非编译型，这里说的编译是字节编译。

**4、IndexError:超出索引（ out of range）**

[复制代码](javascript:void(0);)

>>> lis = []

>>> lis[0]

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

lis[0]

IndexError: list index out of range

[复制代码](javascript:void(0);)

**5、KeyError：请求一个不存在的字典关键字**

[复制代码](javascript:void(0);)

>>> dic = {'name':'BeginMan'}

>>> dic['name']

'BeginMan'

>>> dic['age']

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

dic['age']

KeyError: 'age'

>>>

[复制代码](javascript:void(0);)

**6、IOError：输入输出错误**

[复制代码](javascript:void(0);)

>>> f = open('s.txt','r')

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

f = open('s.txt','r')

IOError: [Errno 2] No such file or directory: 's.txt'

[复制代码](javascript:void(0);)

**7、AttributeError：尝试访问未知的属性**

[复制代码](javascript:void(0);)

>>> class myClass(object):

pass

>>> myIn = myClass()

>>> myIn.bar

Traceback (most recent call last):

File "<pyshell#17>", line 1, in <module>

myIn.bar

AttributeError: 'myClass' object has no attribute 'bar'

>>>

[复制代码](javascript:void(0);)

**8、ValueError：传给函数的参数类型不正确，比如给int()函数传入字符串形**

 三、异常处理

try-except/try-finally只選其一，可以一个try多个except；但只能一个try，一个finally  
try-except-finally组合

[复制代码](javascript:void(0);)

try:

s = 1/0

#except Exception,e:

except ZeroDivisionError, e:

print 'Error:%s' %e

finally:

print 'ok'

[复制代码](javascript:void(0);)

**断言assert:**

[复制代码](javascript:void(0);)

try:

assert 1 == 2,'1 is not equal 2'

except Exception,e:

print '%s:%s' %(e.\_\_class\_\_.\_\_name\_\_,e)

#输出：AssertionError:1 is not equal 2

[复制代码](javascript:void(0);)

 注意：  
如果上面的with代码块没有使用from \_\_future\_\_ import with\_statement, 代码将会报错, 提示你这个功能在2.6中实现.  
Warning: 'with' will become a reserved keyword in Python 2.6

[强烈推荐：http://blog.csdn.net/sinchb/article/details/8392827#t1](http://blog.csdn.net/sinchb/article/details/8392827#t1)

# [Python入门笔记(18):Python函数(1):基础部分](http://www.cnblogs.com/BeginMan/p/3171977.html)

一、什么是函数、方法、过程

推荐阅读：http://www.cnblogs.com/snandy/archive/2011/08/29/2153871.html

一般程序设计语言包含两种基本的抽象：**过程抽象**和**数据抽象**。过程抽象有时也称**控制抽象**。

 子程序在1950年以前就发明了，作为一种抽象那时候并未被完全接受。相反，最初它被看做是一种节省代码的机制，但很快子程序就被认可为过程抽象的一种方式。意识到子程序可以作为一种抽象机制，这产生了三个重要结果。

1. 人们发明了一些语言，支持各种参数传递机制
2. 奠定了「结构化程序设计」的基础，语言开始支持嵌套的子程序（如JS的function，Java的inner class）
3. 诞生了「结构化程序设计」，为试图构建大型系统提供了指导，利用子程序作为基本构建块

**子程序**是最主要过程抽象机制。面向对象语言中的**方法**与子程序的概念十分相似，不同在于它们的调用以及它们与类或对象关联的方式。

[**点击继续阅读**](http://www.cnblogs.com/snandy/archive/2011/08/29/2153871.html)

二、Python函数基础

**1、基础一：**

函数是对程序逻辑进行结构化和过程化的一种编程方式。

过程是简单、特殊、没有返回值的函数

从后面的内容可知Python的过程就是函数，因为解释器会隐式的返回默认值None

**2、基础二：**

**2.1、关于没有返回值：**  
如果什么也不返回的话，在C中默认的是void返回类型，而在Python中是none返回类型。

[复制代码](javascript:void(0);)

>>> def hello():

print 'hello world'

>>> res = hello()

hello world

>>> print res

None

>>> type(res)

<type 'NoneType'>

>>>

[复制代码](javascript:void(0);)

**2.2、关于有返回值：**

**函数的返回值只有一个。**那为什么有的会出现多个呢?这里的”多个“并不是多个返回值而已，比如返回一列表，里面包含很多值。举个例子就是：”只能带走一个东西，但是允许把一些东西装到塑料袋里面看做一个东西带走，这是允许的。“

如在Js中：

[复制代码](javascript:void(0);)

<script type="text/javascript">

function foo(){

return 'xya','abc',123,true;

}

function bar(){

return ['xya','abc',123,true];

}

var s1 = foo();

var s2 = bar();

alert(s1); //输出true

alert(s2); //输出xya,abc,123,true

</script>

[复制代码](javascript:void(0);)

同理，Python中也一样。

[复制代码](javascript:void(0);)

>>> def foo():

return ['xyz',123,True]

>>> def bar():

return 'abc',456,False

[复制代码](javascript:void(0);)

**foo()返回一个列表；bar()返回一个元祖(元祖语法上不需要一定要带上圆括号)，**所以让人真以为可以返回多个对象。当然，我们为了方便可读性，最好加上圆括号。

[复制代码](javascript:void(0);)

>>> aTuple = bar()

>>> x,y,z = bar()

>>> (x,y,z) = bar()

>>> aTuple

('abc', 456, False)

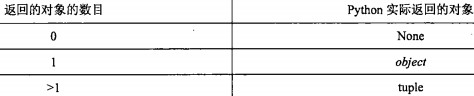
>>> x,y,z

('abc', 456, False)

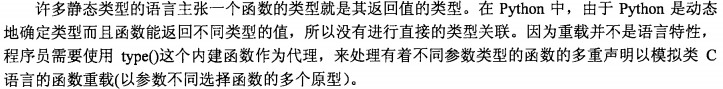
>>> (x,y,z)

('abc', 456, False)

[复制代码](javascript:void(0);)



**注意：**



三、函数调用

[复制代码](javascript:void(0);)

>>> def foo(x,y):

return x,y

>>> foo(1,2)#标准调用

(1, 2)

>>> foo('a','b')

('a', 'b')

>>> foo(x=1,y=2)#参数化调用

(1, 2)

>>>

[复制代码](javascript:void(0);)

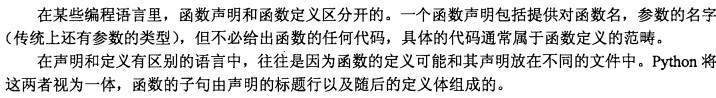
四、创建函数

def function\_name(arguments):

"""function documention\_string"""

function\_body\_suite

**1、声明与定义：**



**2、前向引用**

和其他高级语言类似，Python不允许在函数未声明之前对它引用或调用。

**3、函数的属性**

比如创建两个Py文件,test1.py；test2.py，如下：

[复制代码](javascript:void(0);)

test1.py

--------------------------

ST = 'test1'

def bar():

'''This is bar'''

print 'ok'

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

test2.py

--------------------------------------

import test1

print test1.ST

print test1.bar()

#-----------输出

#test1

#ok

#None

#-----------记得过程是返回值None的函数

def foo():

'''This is foo'''

print 'foo'

print foo.\_\_doc\_\_ #This is foo

[复制代码](javascript:void(0);)

**4、嵌套函数**

[复制代码](javascript:void(0);)

>>> def foo():

def bar():

print 'bar() is called'

print 'foo() is called'

bar()

>>> foo()

foo() is called

bar() is called

>>> bar()

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

bar()

NameError: name 'bar' is not defined

[复制代码](javascript:void(0);)

因为bar()的作用域在foo()里面，在外面不能引用bar。

**5、函数(方法)的装饰器**

[见：http://www.cnblogs.com/BeginMan/p/3173328.html](http://www.cnblogs.com/BeginMan/p/3173328.html)

五、传递函数

**1、其他变量可做函数的别名，因为对象的引用**

所有对象都是通过引用来传递的，函数也不例外，当对一个变量赋值时，实际上是将相同对象的引用赋值给了这个变量。如果对象是函数的话，这个对象的所有别名都是可调用的。

[复制代码](javascript:void(0);)

def foo():

print 'foo'

bar = foo #函数对象的引用

bar() #函数对象的调用

#输出foo

[复制代码](javascript:void(0);)

注意：foo:函数对象的引用；foo():函数对象的调用

**2、函数作为参数传入其他函数来调用**

时刻要记住：函数对象、引用与调用的区别，这几个关键字眼。

[复制代码](javascript:void(0);)

def foo():

print 'foo'

def bar(argfunc):

print 'in bar()'

argfunc()

bar(foo) #把函数作为参数传入其他函数来调用

#输出：

#in bar()

#foo

[复制代码](javascript:void(0);)

局部变量argfunc就相当于函数对象foo的别名

在如下：

[复制代码](javascript:void(0);)

#内建函数int()/float()/long()转换、实现函数的传递与调用

def convert(argfunc,seq):

return [argfunc(obj) for obj in seq]

lis = [123,15.23,-6.2e5,99999999L]

print convert(int, lis) #[123, 15, -620000, 99999999]

print convert(float,lis)#[123.0, 15.23, -620000.0, 99999999.0]

print convert(long, lis)#[123L, 15L, -620000L, 99999999L]

[复制代码](javascript:void(0);)

六、Formal Arguments

**1、位置参数**

以函数定义的准确顺序来传递，如果没有任何默认参数的下，传入参数个数也要一致。

常见错误如下：

takes no arguments (1 given)

takes exactly 2 arguments (1 given)等等

**2、默认参数**

def foo(arg,default1 = 'Python',default2 = 'Django'):

pass

每一个默认参数后面都紧跟一个默认值的赋值语句，如果在函数调用时没有给出值，则使用默认的值。

默认参数最好写在最后

**3、关键字参数**

[复制代码](javascript:void(0);)

def foo(arg,obj1='good',obj2=10):

return arg,obj1,obj2

print foo(arg='BeginMan') #('BeginMan', 'good', 10)

print foo('BeginMan') #('BeginMan', 'good', 10)

print foo('BeginMan','Python') #('BeginMan', 'Python', 10)

#print foo(arg='BeginMan',100) #出错

print foo(arg='BeginMan',obj2=100) #('BeginMan', 'good', 100)

print foo(obj2=100,arg='BeginMan',obj1='Ok') #('BeginMan', 'Ok', 100)

[复制代码](javascript:void(0);)

七、可变长度的参数

1、变长参数与非变长参数

[强烈推荐：http://blog.csdn.net/qinyilang/article/details/5484415](http://blog.csdn.net/qinyilang/article/details/5484415)

[复制代码](javascript:void(0);)

"""在参数名之前使用一个星号，就是让函数接受任意多的位置参数"""

def multiply(\*args):

total = 1

for arg in args:

total+=arg

return total

print multiply() #1

print multiply(1,2) #4

print multiply(4,5,6) #16

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

"""python在参数名之前使用2个星号来支持任意多的关键字参数"""

def multiply2(\*\*kwargs):

for key,value in kwargs.items():

print '%s=>%s' %(key,value)

multiply2()

multiply2(name='beginman') #name=>beginman

multiply2(name='beginman',age='22',tel=110)

[复制代码](javascript:void(0);)

|  |
| --- |
| #顺序  def complex\_function(a,b=None,\*args,\*\*kwarg):      pass  #测试  def add(a,b,c):      return a+b+c    print add(a=10,b=10,c=10)  args=(2,3)  print add(1,\*args)  #6  kwargs={'b':100,'c':200}  print add(100,\*\*kwargs) #400  print add(a=100,\*\*kwargs)   #400  #print add(a=100,\*args)      #TypeError: add() got multiple values for keyword argument 'a'  print add(1,2,c=3)  #6  print add(1,2,b=3)  # add() got multiple values for keyword argument 'b' |

 引用如下：

**过量的参数**在运行时知道一个函数有什么参数，通常是不可能的。另一个情况是一个函数能操作很多对象。更有甚者，调用自身的函数变成一种api提供给可用的应用。  
  
对于这些情况，python提供了两种特别的方法来定义函数的参数，允许函数接受过量的参数，不用显式声明参数。这些“额外”的参数下一步再解释。  
  
注意args和kwargs只是python的约定。任何函数参数，你可以自己喜欢的方式命名，但是最好和python标准的惯用法一致，以便你的代码，其他的程序员也能轻松读懂。  
 **位置参数**  
  
在参数名之前使用一个星号，就是让函数接受任意多的位置参数。

|  |
| --- |
| >>> def multiply(\*args):  ...     total = 1  ...     for arg in args:  ...         total \*= arg  ...     return total  ...  >>> multiply(2, 3)  6  >>> multiply(2, 3, 4, 5, 6)  720 |

python把参数收集到一个元组中，作为变量args。显式声明的参数之外如果没有位置参数，这个参数就作为一个空元组。  
  
**关键字参数**python在参数名之前使用2个星号来支持任意多的关键字参数。

|  |
| --- |
| >>> def accept(\*\*kwargs):  ...     for keyword, value in kwargs.items():  ...         print "%s => %r" % (keyword, value)  ...  >>> accept(foo='bar', spam='eggs')  foo => 'bar'  spam => 'eggs' |

注意：kwargs是一个正常的python字典类型，包含参数名和值。如果没有更多的关键字参数，kwargs就是一个空字典。  
  
**混合参数类型**  
  
任意的位置参数和关键字参数可以和其他标准的参数声明一起使用。混合使用时要加些小心，因为python中他们的次序是重要的。参数归为4类，不是所有的类别都需要。他们必须按下面的次序定义，不用的可以跳过。  
  
1）必须的参数  
2）可选的参数  
3）过量的位置参数  
4）过量的关键字参数  
  
def complex\_function(a, b=None, \*c, \*\*d):  
  
这个次序是必须的，因为\*args和\*\*kwargs只接受那些没有放进来的其他任何参数。没有这个次序，当你调用一个带有位置参数的函数，python就不知道哪个值是已声明参数想要的，也不知道哪个被作为过量参数对待。  
  
也要注意的是，当函数能接受许多必须的参数和可选的参数，那它只要定义一个过量的参数类型即可。  
  
**传递参数集合**  
  
除了函数能接受任意参数集合，python代码也可以调用带有任意多数量的函数，像前面说过的用星号。这种方式传递的参数由python扩展成为参数列表。以便被调用的函数  
不需要为了这样调用而去使用过量参数。python中任何可调用的，都能用这种技法来调用。并且用相同的次序规则和标准参数一起使用。

|  |
| --- |
| >>> def add(a, b, c):  ...     return a + b + c  ...  >>> add(1, 2, 3)  6  >>> add(a=4, b=5, c=6)  15  >>> args = (2, 3)  >>> add(1, \*args)  6  >>> kwargs={'b': 8, 'c': 9}  >>> add(a=7, \*\*kwargs)  24  >>> add(a=7, \*args)  Traceback (most recent call last):    File "<stdin>", line 1, in <module>  TypeError: add() got multiple values for keyword argument 'a'  >>> add(1, 2, a=7)  Traceback (most recent call last):    File "<stdin>", line 1, in <module>  TypeError: add() got multiple values for keyword argument 'a' |

注意这个例子的最后几行，特别留意当传递一个元组作为过量的位置参数时，是否要显式的传递关键字参数。因为python使用次序规则来扩展过量的参数，那位置参数要放在前面。这个例子中，最后两个调用是相同的，python不能决定那个值是给a的。

# [Python入门笔记(19):Python函数(2):函数/方法装饰器](http://www.cnblogs.com/BeginMan/p/3173328.html)

一、装饰器(decorators)

装饰器的语法以@开头，接着是装饰器函数的名字、可选参数。

紧跟装饰器声明的是被装饰的函数和被装饰的函数的可选参数，如下：

@decorator(dec\_opt\_args)

def func(func\_args):

....

其实总体说起来，装饰器其实也就是一个函数，一个用来包装函数的函数，装饰器在函数声明完成的时候被调用，调用之后声明的函数被换成一个被装饰器装饰过后的函数。

如：

[复制代码](javascript:void(0);)

def deco(func):

...

return func

@deco

def foo():

print 'foo'

#-----------------------------------

#等价如下：

def deco(func):

...

return func

def foo():

print 'foo'

foo = deco(foo)

[复制代码](javascript:void(0);)

如下例子：

[复制代码](javascript:void(0);)

def deco1(func):

print 'ok'

return func

@deco1

def foo():

print 'foo'

foo()

#输出--------------

#ok

#foo

#------------------

[复制代码](javascript:void(0);)

如果不使用装饰器，则可如下：

[复制代码](javascript:void(0);)

def deco1(func):

print 'ok'

return func

def foo():

print 'foo'

print foo #<function foo at 0x00AFE6F0>

foo = deco1(foo)

foo()

#输出--------------

#ok

#foo

#------------------

[复制代码](javascript:void(0);)

两者对比下，可发现使用装饰器是那么简便、灵活。特别是在企业级的开发上。

同时也可以多个装饰器重叠使用：

[复制代码](javascript:void(0);)

def deco1(func):

print 'deco1'

return func

def deco2(func):

print 'deco2'

return func

@deco1

@deco2

def foo():

print 'foo'

foo()

#输出如下：-----------

#deco2

#deco1

#foo

#---------------------

[复制代码](javascript:void(0);)

等效于：

[复制代码](javascript:void(0);)

@deco1

@deco2

def foo(arg):pass

-----------与下等效----------

foo = deco1(deco2(foo()))

[复制代码](javascript:void(0);)

二、有参、无参的装饰器

上面的例子基本上都是有参数的，无参数更为简单。

1、无参

[复制代码](javascript:void(0);)

@deco1

@deco2

def foo(arg):pass

---------------------

foo = deco1(deco2(foo()))

[复制代码](javascript:void(0);)

2、有参

[复制代码](javascript:void(0);)

@deco1(deco\_arg)

@deco2

def foo(arg):pass

---------------------

foo = deco1(deco\_arg)(deco2(foo()))

[复制代码](javascript:void(0);)

返回以函数作为参数的装饰器

三、用处

1、引用日志

2、增加计时逻辑来检测性能

3、给函数加入事务的能力

四、实例

[复制代码](javascript:void(0);)

from time import ctime,sleep

def deco(func):

def decoIn():

print '[%s]:%s called' %(ctime(),func.\_\_name\_\_)

return func

return decoIn

@deco

def foo():

pass

foo()

sleep(4)

for i in range(2):

sleep(1)

foo()

#输出如下：--------

#[Fri Jul 05 10:45:04 2013]:foo called

#[Fri Jul 05 10:45:09 2013]:foo called

#[Fri Jul 05 10:45:10 2013]:foo called

#------------------

[复制代码](javascript:void(0);)

五、强烈推荐阅读：

<http://www.cnblogs.com/huxi/archive/2011/03/01/1967600.html>

# [Python入门笔记(20):Python函数(3):关于lambda](http://www.cnblogs.com/BeginMan/p/3178103.html)

一、lambda函数

**1、lambda函数基础：**

lambda函数也叫匿名函数，即，函数没有具体的名称,而用def创建的方法是有名称的。如下：

"""命名的foo函数"""

def foo():return 'beginman' #Python中单行参数可以和标题写在一行

"""lambda关键字创建匿名函数,该表达式同以上函数"""

lambda:'beginman'

上面的只是简单的用lambda创建一个函数对象，并没有保存它也没有调用它，时刻会被回收了。这里我们保存并调用：

bar = lambda:'beginman'

print bar() #beginman

从上面几个例子中，可易理解Python lambda语法：

lambda [arg1[,arg2,arg3....argN]]:expression

lambda语句中，冒号前是参数，可以有多个，用逗号隔开，冒号右边的返回值。lambda语句构建的其实是一个函数对象。

print lambda:'beginman' #<function <lambda> at 0x00B00A30>

**2、无参数**

如果没有参数，则lambda冒号前面就没有，如以上例子。

**3、有参数**

[复制代码](javascript:void(0);)

def add(x,y):return x+y

add2 = lambda x,y:x+y

print add2(1,2) #3

def sum(x,y=10):return x+y

sum2 = lambda x,y=10:x+y

print sum2(1) #11

print sum2(1,100) #101

[复制代码](javascript:void(0);)

二、lambda与def

上面的例子中，可知lambda函数只是创建简单的函数对象，是一个函数的单行版本，但是这种语句由于性能的原因，调用的时候绕过函数的栈分配。python lambda还有哪些和def不一样呢？

1 、python lambda会创建一个函数对象，但不会把这个函数对象赋给一个标识符，而def则会把函数对象赋值给一个变量。

如：

>>> def foo():return 'foo()'

>>> foo

<function foo at 0x011A34F0>

2、python lambda它只是一个表达式，而def则是一个语句。lambda表达式运行起来像一个函数，当被调用时创建一个框架对象。

三、lambda函数的用途

个人认为有以下：

1、对于单行函数，使用lambda可以省去定义函数的过程，让代码更加精简。

2、在非多次调用的函数的情况下，lambda表达式即用既得，提高性能

注意：如果for..in..if能做的，最好不要选择lambda

四、参考

http://www.cnblogs.com/coderzh/archive/2010/04/30/python-cookbook-lambda.html

http://www.cnblogs.com/wanpython/archive/2010/11/01/1865919.html

# [Python入门笔记(20):Python函数(3):关于lambda](http://www.cnblogs.com/BeginMan/p/3178103.html)

一、lambda函数

**1、lambda函数基础：**

lambda函数也叫匿名函数，即，函数没有具体的名称,而用def创建的方法是有名称的。如下：

"""命名的foo函数"""

def foo():return 'beginman' #Python中单行参数可以和标题写在一行

"""lambda关键字创建匿名函数,该表达式同以上函数"""

lambda:'beginman'

上面的只是简单的用lambda创建一个函数对象，并没有保存它也没有调用它，时刻会被回收了。这里我们保存并调用：

bar = lambda:'beginman'

print bar() #beginman

从上面几个例子中，可易理解Python lambda语法：

lambda [arg1[,arg2,arg3....argN]]:expression

lambda语句中，冒号前是参数，可以有多个，用逗号隔开，冒号右边的返回值。lambda语句构建的其实是一个函数对象。

print lambda:'beginman' #<function <lambda> at 0x00B00A30>

**2、无参数**

如果没有参数，则lambda冒号前面就没有，如以上例子。

**3、有参数**

[复制代码](javascript:void(0);)

def add(x,y):return x+y

add2 = lambda x,y:x+y

print add2(1,2) #3

def sum(x,y=10):return x+y

sum2 = lambda x,y=10:x+y

print sum2(1) #11

print sum2(1,100) #101

[复制代码](javascript:void(0);)

二、lambda与def

上面的例子中，可知lambda函数只是创建简单的函数对象，是一个函数的单行版本，但是这种语句由于性能的原因，调用的时候绕过函数的栈分配。python lambda还有哪些和def不一样呢？

1 、python lambda会创建一个函数对象，但不会把这个函数对象赋给一个标识符，而def则会把函数对象赋值给一个变量。

如：

>>> def foo():return 'foo()'

>>> foo

<function foo at 0x011A34F0>

2、python lambda它只是一个表达式，而def则是一个语句。lambda表达式运行起来像一个函数，当被调用时创建一个框架对象。

三、lambda函数的用途

个人认为有以下：

1、对于单行函数，使用lambda可以省去定义函数的过程，让代码更加精简。

2、在非多次调用的函数的情况下，lambda表达式即用既得，提高性能

注意：如果for..in..if能做的，最好不要选择lambda

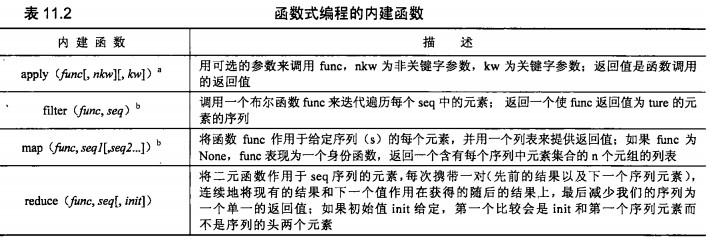
四、参考

http://www.cnblogs.com/coderzh/archive/2010/04/30/python-cookbook-lambda.html

http://www.cnblogs.com/wanpython/archive/2010/11/01/1865919.html

# [Python入门笔记(21):Python函数(4):关于函数式编程的内建函数](http://www.cnblogs.com/BeginMan/p/3178363.html)

一、关于函数式编程的内建函数



apply()逐渐被舍弃，这里不讨论

**1、filter()**

[复制代码](javascript:void(0);)

#filter(func,seq)

"""纯Python描述filter函数"""

def Myfilter(bool\_func,seq):

filtered\_seq = []

for obj in seq:

if bool\_func(obj):

filtered\_seq.append(obj)

return filtered\_seq

print Myfilter(lambda x:x%3==0, [1,2,3,4,5,6,7,8,9]) #[3, 6, 9]

"""Build-In function filter()"""

print filter(lambda x:x%3==0, [1,2,3,4,5,6,7,8,9]) #[3, 6, 9]

"""非函数下的实现"""

print [x for x in [1,2,3,4,5,6,7,8,9] if x%3==0] #[3, 6, 9]

#还记得上一节写的，如果能有for..in..if（或列表解析）能处理的最好用这个而放弃lambda

[复制代码](javascript:void(0);)

**试验：列表解析和lambda性能比较**

lambda:

[复制代码](javascript:void(0);)

import time

start = time.clock()

filter(lambda x:x%3==0, [i for i in range(10000000)])

end = time.clock()

print end-start #耗时4.9441799282

[复制代码](javascript:void(0);)

列表解析：

[复制代码](javascript:void(0);)

import time

start = time.clock()

[x for x in range(10000000) if x%3==0]

end = time.clock()

print end-start #耗时2.95589058109

[复制代码](javascript:void(0);)

从上可见，最好使用列表解析。

**2、map()**

[复制代码](javascript:void(0);)

"""纯Python模拟map()"""

def Mymap(func,seq):

mapped\_seq = []

for obj in seq:

mapped\_seq.append(func(obj))

return mapped\_seq

print Mymap(lambda x:x\*10,[i for i in range(10)]) #[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

"""内建map()函数"""

print map(lambda x:x\*10,[i for i in range(10)]) #[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

"""非函数式编程"""

print [x\*10 for x in range(10)] #[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

"""多个序列的map()"""

print map(lambda x,y:x+y,[1,2,3],[4,5,6]) #[5, 7, 9]

"""None与map()"""

print map(None,[1,2,3],[4,5,6]) #[(1, 4), (2, 5), (3, 6)]

"""None在map()中使用效果同zip()，将不相干的序列归并在一起"""

print zip([1,2,3],[4,5,6]) #[(1, 4), (2, 5), (3, 6)]

[复制代码](javascript:void(0);)

**3、reduce()**

[复制代码](javascript:void(0);)

"""纯Python模拟reduce()"""

def Myreduce(bin\_func,seq,init=None):

Iseq = list(seq)

if init is None:

res = Iseq.pop()

else:

res = init

for obj in Iseq:

res = bin\_func(res,obj)

return res

print Myreduce(lambda x,y:x+y, [i for i in range(10)]) #45

print Myreduce(lambda x,y:x+y, [i for i in range(10)],100) #145

"""reduce()"""

print reduce(lambda x,y:x+y,[i for i in range(10)]) #45

print reduce(lambda x,y:x+y,[i for i in range(10)],100) #145

[复制代码](javascript:void(0);)

综上，尽量用最简便的方式去实现。

# [Python入门笔记(22):Python函数(5):变量作用域与闭包](http://www.cnblogs.com/BeginMan/p/3179040.html)

一、全局变量与局部变量

一个模块中，最高级别的变量有全局作用域。

全局变量一个特征就是：除非被删除，否则他们存活到脚本运行结束，且对于所有的函数都可访问。

当搜索一个标识符（也称变量、名字等），Python是先从局部作用域开始搜索，如果局部作用域内没有找到那个名字，那么就在全局域找，如果还没有则抛出一个NameError异常。

一个变量的作用域和它寄存的名称空间相关，注意，局部变量会覆盖全局变量。如：

[复制代码](javascript:void(0);)

bar = 10

def foo():

bar = 100

return bar

print foo() #100

[复制代码](javascript:void(0);)

**global语句：**

为了明确地引用一个已命名的全局变量，必须使用global语句，语法如下：

global var1[,val2...]

[复制代码](javascript:void(0);)

bar = 10

def foo():

global bar

return bar

bar = 10000

print foo() #10000

[复制代码](javascript:void(0);)

下面的例子更为详细：

[复制代码](javascript:void(0);)

bar = 10 #声明一个全局变量bar

def foo():

global bar #指定bar是全局变量，谁都不能替换！但是能修改原值！

bar2 = 0.1 #声明一个局部变量bar2

bar = 0.0001 #是对全局变量bar的修改，而不是在又声明一个局部变量将全局变量覆盖。

return bar+bar2

bar = 10000 #又对全局变量bar进行修改，但是为时已晚了，因为在foo函数里已经 return bar+bar2了

print foo() #综上，输出：0.1001

[复制代码](javascript:void(0);)

**嵌套函数作用域：**

[复制代码](javascript:void(0);)

def foo():

m = 1

def bar():

n = 2

return m+n

# m +=n #NameError: global name 'n' is not defined

return bar()

print foo() #3

[复制代码](javascript:void(0);)

二、闭包

如果在一个内部函数里，对在外部函数内（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是**闭包(closure)。**

定义在外部函数内但由内部函数引用或者使用的变量称为**自由变量**。

闭包(closure)是函数式编程的重要的语法结构。函数式编程是一种编程范式 (而面向过程编程和面向对象编程也都是编程范式)。在面向过程编程中，我们见到过函数(function)；在面向对象编程中，我们见过对象(object)。函数和对象的根本目的是以某种逻辑方式组织代码，并提高代码的可重复使用性(reusability)。闭包也是一种组织代码的结构，它同样提高了代码的可重复使用性。

不同的语言实现闭包的方式不同。Python以函数对象为基础，为闭包这一语法结构提供支持的 (我们在[特殊方法与多范式](http://www.cnblogs.com/vamei/archive/2012/11/19/2772441.html)中，已经多次看到Python使用对象来实现一些特殊的语法)。Python一切皆对象，函数这一语法结构也是一个对象。在[函数对象](http://www.cnblogs.com/vamei/archive/2012/07/10/2582772.html)中，我们像使用一个普通对象一样使用函数对象，比如更改函数对象的名字，或者将函数对象作为参数进行传递。

参考：http://www.cnblogs.com/vamei/archive/2012/12/15/2772451.html

如实现一个计数的闭包例子：

[复制代码](javascript:void(0);)

def counter(start\_at=0):

count = [start\_at]

#内部函数incr()实现计数

def incr(): #定义一个内部函数,在内部使用变量count,创建一个闭包

count[0]+=1

return count[0]

return incr #返回一个可调用的函数对象

#看起来和实例化一个counter对象并执行这个实例是多么相似

count = counter(5)

for i in range(10):

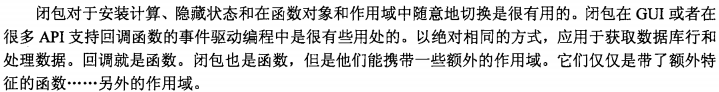
print count(), #6 7 8 9 10 11 12 13 14 15

count = counter(100)

print count() #101

[复制代码](javascript:void(0);)

**闭包的作用**：



**推荐阅读**：[Python深入04 闭包](http://www.cnblogs.com/vamei/archive/2012/12/15/2772451.html)

# [Python入门笔记(23)：模块](http://www.cnblogs.com/BeginMan/p/3183656.html)

一、模块基础

**1、模块**

自我包含，且有组织的代码片段就是模块

模块是Pyhon最高级别的程序组织单元，它将程序代码和数据封装起来以便重用。实际的角度，模块往往对应Python程序文件。  
每个文件都是一个模块，并且模块导入其他模块之后就可以使用导入模块定义的变量名。模块可以由两个语句和一个重要的内置函数进行处理。  
import: 使客户端（导入者）以一个整体获取一个模块。  
from:容许客户端从一个模块文件中获取特定的变量名。  
reload:在不中止Python程序的情况下，提供了一个重新载入模块文件代码的方法。

参考：http://ipseek.blog.51cto.com/1041109/795782

**2、名称空间**

一个名称空间就是从名称到对象的关系隐射集合。

每一个模块都定义的自己唯一的名称空间，如：

import s1,s2#s1.py,s2.py

s1.foo()

s2.bar()

s1.bar() #由于每个模块都有唯一的名称空间，即使属性相同也不会发生冲突

向名称空间添加名称的操作过程涉及绑定标识符到指定对象的操作（以及给对象的引用计数加1）。

**3、搜索路径和路径搜索**

模块的导入需要叫做“路径搜索”的过程。

**搜索路径：查找一组目录**

**路径搜索：查找某个文件的操作**

ImportError: No module named myModule

这种错误就是说：模块不在**搜索路径**里，从而导致**路径搜索**失败！

导入模块时，不带模块的后缀名，比如.py  
Python搜索模块的路径：  
1)、程序的主目录  
2)、PTYHONPATH目录（如果已经进行了设置）  
3)、标准连接库目录（一般在/usr/local/lib/python2.X/）  
4)、任何的.pth文件的内容（如果存在的话）.新功能，允许用户把有效果的目录添加到模块搜索路径中去  
.pth后缀的文本文件中一行一行的地列出目录。  
这四个组建组合起来就变成了sys.path了，

>>> import sys

>>> sys.path

导入时，Python会自动由左到右搜索这个列表中每个目录。

二、模块导入

**1、import语句**

推荐：

import放在程序前面，且按照Python标准库模块、Python第三方模块、自定义模块的顺序从上到下排开。

**2、from-import语句**

可以导入模块中指定属性:from module import name1[,nam2[,...nameN]]

注意：

(1)、导入可使用\进行换行，

如：from module import nam1,name2,\  
name3,name4....

(2)、不推荐from module import \*

(3)、使用as更换名称：

from module import name1 as name2

**3、模块导入的特征**

(1)、载入时执行模块，也就是导入模块的顶层代码将会被执行，通常包括该模块内**全局变量**、**类和函数的声明**。我们尽可能多的把代码封装到函数中，这是良好的模块编程习惯。

(2)、一个模块可被导入(import)多次,但是只加载(load)一次.

**4、模块内建函数**

(1)、\_\_import\_\_():导入模块的函数

import语句实际上调用\_\_import\_\_()函数

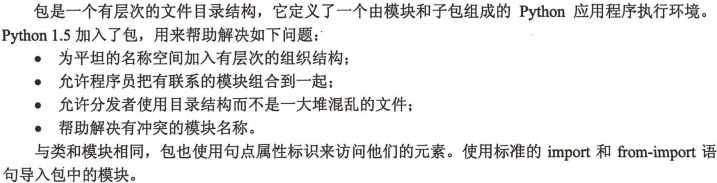
如果我们有一个testa.py文件，我们可以用module=\_\_import\_\_('testa')来导入它。这样可以用module.属性来使用testa.py中的属性

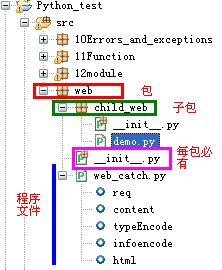
其他见书

三、包

**1、包的概念**

包是一个有层次的文件目录结构，由模块和子包组成。





**2、解析包机制：**

每一个.py文件称为一个module,module之间可以互相导入

module可以定义在包里面.Python定义包的方式稍微有点古怪,假设我们有一个parent文件夹,该文件夹有一个child子文件夹.child中有一个module a.py . 如何让Python知道这个文件层次结构?很简单,每个目录都放一个名为\_init

\_.py 的文件.该文件内容可以为空.这个层次结构同上:

参考：http://huangliangfeixu.blog.163.com/blog/static/18974706220112741040442/

**3、python中\_\_init\_\_.py文件的作用**

python的每个模块的包中，都有一个\_\_init\_\_.py文件，有了这个文件，我们才能导入这个目录下的module。  
那么，\_\_init\_\_.py还有什么别的功能呢？  
其实，\_\_init\_\_.py里面还是可以有内容的，我们在导入一个包时，实际上导入了它的\_\_init\_\_.py文件。

我们可以再\_\_init\_\_.py文件中再导入其他的包，或者模块。  
[python]  
import readers   
import writers   
import commands   
import users   
import meta   
import auth   
import admin

这样，当我们导入这个包的时候，\_\_init\_\_.py文件自动运行。帮我们导入了这么多个模块，我们就不需要将所有的import语句写在一个文件里了，也可以减少代码量。  
不需要一个个去导入module了。  
\_\_init\_\_.py 中还有一个重要的变量，叫做 \_\_all\_\_。我们有时会使出一招“全部导入”，也就是这样：  
from PackageName import \*  
这时 import 就会把注册在包 \_\_init\_\_.py 文件中 \_\_all\_\_ 列表中的子模块和子包导入到当前作用域中来。比如：  
#文件 \_\_init\_\_.py

\_\_all\_\_ = ["Module1", "Module2", "subPackage1", "subPackage2"]

如：在一个包里有foo.py、\_\_init\_\_.py

#\_\_init\_\_.py

import os

import datetime

#foo.py

from \_\_init\_\_ import \*

print datetime.datetime.now()

则输出：2013-07-11 11:34:41.250000

**4、导入**

参考上面的图片

import web.web\_catch

import web.child\_web.demo

from web.child\_web import demo

# [Python入门笔记(24)：Python面向对象(1)速成](http://www.cnblogs.com/BeginMan/p/3190776.html)

一、Python经典类与新类

经典类：如果没有直接或间接的子类化一个对象，也就是说如果没有指定一个父类，或者是如果子类化的基本类没有父类，那么就定义了经典类：

class classics:

'define a classics Class'

pass

新式类：与经典类相反。object是“所有类之母”，也就是基类，如果你的类没有继承任何父类，object将作为默认的父类：

class newClass(object):

'define a classics Class'

pass

二、Python的实例化

创建一个实例的过程叫实例化，在其他编程语言中一般用关键字 new ,但是在Python中没有这个关键字。而是类似函数调用的方式创建实例的：

[复制代码](javascript:void(0);)

class newClass(object):

pass

inst = newClass()

print inst #<\_\_main\_\_.newClass object at 0x00B71650>

[复制代码](javascript:void(0);)

三、类最简单的用法

类最简单的用法仅仅用作名称空间，意味着将数据保存到变量中，使用句点属性标识对它们按名称空间分组。

[复制代码](javascript:void(0);)

class newClass(object):

pass

bar = newClass() #创建实例

bar.x = 1 #创建实例属性

bar.y = 2

print bar.x+bar.y

[复制代码](javascript:void(0);)

注意：bar.x  ,bar.y  是实例对象的属性不是类的属性。

四、方法

方法就是类的功能

方法在类中定义，只能被实例调用。

在类中，方法有一个默认参数self，这个参数代表实例本身，当用实例调用方法时，由解释器悄悄传递给方法。

[复制代码](javascript:void(0);)

class newClass(object):

def method(self):

return self

bar = newClass()

print bar.method() #<\_\_main\_\_.newClass object at 0x00B71910>

[复制代码](javascript:void(0);)

在其他语言中，self一般称为this，一般方法会用到这个实例(self)，但是静态方法和类方法不需要。

五、创建类、方法、实例对象访问

[复制代码](javascript:void(0);)

class addPerson(object):

def \_\_init\_\_(self,nm,ph): #定义构造器

self.name = nm #设置name

self.phone = ph #设置phone

print 'Created instance for',self.name

def updatePhone(self,newph):

self.phone = newph

print 'Updated phone for',self.phone

josn = addPerson('JSON','1565208444') #创建实例对象josn

ben = addPerson('Ben','15249638892') #创建实例对象ben

print 'Instance property:[%s][%s]' %(ben.name,ben.phone) #访问实例属性

ben.updatePhone('110')

print ben.phone

[复制代码](javascript:void(0);)

六、子类

创建子类:靠继承来进行子类化，在不影响父类（基类）的基础上改进自己的功能。

如:继承上面的类

[复制代码](javascript:void(0);)

class addInfo(addPerson):

def \_\_init\_\_(self,nm,ph,id,em): #定义自己的构造器

addPerson.\_\_init\_\_(self, nm, ph) #继承父类构造器（这个很重要）

self.empid = id

self.email = em

def updateEmail(self,newem):

self.email = newem

print 'Updated e-mail address for:',self.name

zhangsan = addInfo('ZhangSan','123456789','01','ZhangSan@gmail') #创建addInfo实例,输出Created instance for ZhangSan

print zhangsan #<\_\_main\_\_.addInfo object at 0x00B71BD0>

print 'Instance property:[%s],[%s],[%s],[%s]' %(zhangsan.name,zhangsan.phone,zhangsan.empid,zhangsan.email)

#输出：Instance property:[ZhangSan],[123456789],[01],[ZhangSan@gmail]

zhangsan.updatePhone('250') #Updated phone for 250

zhangsan.updateEmail('Zhang@163.com') #Updated e-mail address for: ZhangSan

[复制代码](javascript:void(0);)

**注意：**

如果需要，每个子类最好定义自己的构造器，不然的话，基类的构造器将会被调用。然而如果子类重写基类的构造器，那么基类的构造器就不会被自动调用，必须显式写出来才行，如上的：addPerson.\_\_init\_\_

# [Python入门笔记(25)：Python面向对象(2)](http://www.cnblogs.com/BeginMan/p/3191037.html)

一、类

类就是一个数据结构，封装了数据和操作。

类的声明与函数的声明十分类似：

class newClass(object):

"""class documentation string"""#类文档字符串

class\_suite #类体

注意：类是对象（在Python中，一切皆对象），但是类在定义的时候，还不是对象的实现。

二、类属性

在[面向对象](http://baike.baidu.com/view/125370.htm)的编程和思想中，出现了类属性的概念。

在java中，类的实例[变量](http://baike.baidu.com/view/296689.htm)和[静态变量](http://baike.baidu.com/view/675642.htm)称为类属变量（class‘s variables，常称为[类变量](http://baike.baidu.com/view/2893749.htm)）或数据域（java语言程序设计（基础篇）》，机械工业出版社，p185）。

在[面向对象](http://baike.baidu.com/view/125370.htm)的编程语言Python中，类属性就是在类中定义的[变量](http://baike.baidu.com/view/296689.htm)。

在C++中，类属性是描述类的所有对象共同特征的一个[数据项](http://baike.baidu.com/view/178581.htm)，对于任何对象实例，它的属性值是相同的。

不同的编程语言有不同的定义。而有些语言是不区分static类型的，所以并不能用static来表示这个属性是否为类属性。只不过用static修饰的属性可以实现对象间的数据共享而已。

属性：就是属于另一个对象的数据或函数元素。

在Python中一个有趣的现象是：当访问一个属性时，它同时也是一个对象。拥有它自己的属性可以访问，这就造成了一个属性链。

1、类的数据属性：

就是定义类的变量，即静态变量，或静态数据。它们与所属的类对象绑定，不依赖任何类实例。这里类型的数据相当于在变量前面加上static。

[复制代码](javascript:void(0);)

class newClass(object):

foo = 100

print newClass.foo #100

newClass.foo+=100

print newClass.foo #200

[复制代码](javascript:void(0);)

类属性与实例属性不一样。

2、方法

方法也成为类属性。

3、查看类属性

使用内建函数dir()或类的字典属性\_\_dict\_\_都可以。

[复制代码](javascript:void(0);)

class newClass(object):

def foo(self):

pass

print dir(newClass)

#['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_str\_\_', '\_\_weakref\_\_', 'foo']

print newClass.\_\_dict\_\_

#{'\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'newClass' objects>, '\_\_module\_\_': '\_\_main\_\_', 'foo': <function foo at 0x00B00A30>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'newClass' objects>, '\_\_doc\_\_': None}

[复制代码](javascript:void(0);)



[复制代码](javascript:void(0);)

class newClass(object):

"""Python Class"""

def foo(self):

pass

print newClass.\_\_name\_\_ #newClass

print newClass.\_\_doc\_\_ #Python Class

print newClass.\_\_bases\_\_ #(<type 'object'>,)

print newClass.\_\_base\_\_ #<type 'object'>

print newClass.\_\_module\_\_ #\_\_main\_\_

print newClass.\_\_class\_\_ #<type 'type'>

[复制代码](javascript:void(0);)

三、实例

1、关于类的实例

类是一种数据结构定义类型，那么实例就是声明这种类型的变量。

[复制代码](javascript:void(0);)

class newClass(object):

"""Python Class"""

def foo(self):

pass

c = newClass()

print type(c) #<class '\_\_main\_\_.newClass'>,而在Python2.2版本之前，实例是”实例类型“从不考虑从哪个类来的

print type(0) #<type 'int'>

print type(newClass) #<type 'type'>

print type(int) #<type 'type'>两者都是类型

[复制代码](javascript:void(0);)

注意，在2.2版本之后，当定义一个新式类后，你已创建一个新的类型。

2、\_\_init\_\_() “构造器”方法

不通过new来创建实例，你也没有定义一个构造器，是Python为你创建对象

\_\_init\_\_()是在解释器为你创建一个实例后调用的第一个方法

如果没有定义(或覆盖)特殊方法\_\_init\_\_().则对实例不会施加任何特殊的操作。

3、\_\_new\_\_()“构造器”方法

与\_\_init\_\_()相比，\_\_new\_\_()则更像一个真正的构造器。

4、\_\_del\_\_()"解构器"方法

四、类属性与实例属性

[参考：](http://www.pythonfan.org/thread-9827-1-1.html)http://www.pythonfan.org/thread-9827-1-1.html

python中的类属性只是存储与类相关 的数据，和该类的实例无关。类属性和java中的静态成员变量类似。访问python的类属性可使用类名+“.”+属性名的方式，如果类的实例没有同名变 量也可以使用实例来访问。如果实例含有与类属性同名的属性，则用该实例访问属性时，访问的是实例中的属性。

[复制代码](javascript:void(0);)

class Test():

myVersion="1.0" #声明一个类属性，并赋值为1.0

t=Test() #生成一个实例

Test.myVersion #使用类空间来访问类属性 ，输出为 1.0

t.myVersion #使用实例空间来访问类属性，输出为1.0

Test.myVersion="2.0" #使用类空间来更新类属性

Test.myVersion #使用类空间来访问类属性 ，输出为 2.0

t.myVersion #使用实例空间来访问类属性，输出为2.0

[复制代码](javascript:void(0);)

只有使用类空间引用类属性时，才能设定和更新类属性。如果尝试使用实例空间来引用类属性来进行更新，则该实例（如果没有同名属性的话）会创建一个与该类属性同名的实例属性。该实例属性会阻止实例对类属性的访问，直到该实例的同名属性被清除掉。

[复制代码](javascript:void(0);)

t.myVersion='3.0' #实例t会创建一名为‘myVersion’的实例属性

Test.myVersion #输出2.0，上一条语句并不会对类属性产生任何影响

t.myVersion #输出3.0，t访问的是自身的实例属性

del t.myVersion #清除t的实例属性

t.myVersion #输出2.0，此时访问的是类属性

[复制代码](javascript:void(0);)

但是，在类属性可变的情况下，事情又不一样了

[复制代码](javascript:void(0);)

Test.x={'myVersion':'1.0'} #给Test类添加一个新的类属性

Test.x #用类空间访问该属性 输出 {'myVersion':'1.0'}

t.x #用实例空间访问该属性 输出 {'myVersion':'1.0'}

t.x['myVersion']='2.0'

t.x #输出{'myVersion':'2.0'}

Test.x #输出{'myVersion':'2.0'} # 实例t的更新操作对类属性生效了

del t.x #出错 ：t instance has no attribute 'x'

[复制代码](javascript:void(0);)

# [Python入门笔记(26)：Python执行环境](http://www.cnblogs.com/BeginMan/p/3191856.html)

一、python特定的执行环境

在当前脚本继续进行

创建和管理子进程

执行外部命令或程序

执行需要输入的命令

通过网络来调用命令

执行命令来创建需要处理的输出

动态生成Python语句

导入Python模块

这些都可通过内建或外部模块实现。

二、可调用对象

调用方式：可调用对象后紧跟操作符()

4种可调用对象：函数、方法、类以及一些类的实例

**1、函数**

分三种不同类型的函数对象

(1)、内建函数(BIF)，是Built-in Function缩写，由C/C++编写导入Python编辑器的。

(2)、用户自定义函数(UDF)

(3)、lambda函数

**2、方法**

有类方法，还有内建方法(BIM)，如列表，字典的。

class C(object):  
    def foo(self):pass  #定义类方法，或叫用户自定义方法（UDM）  
      
c=C()  
print type(C)   #类的类别，输出<type 'type'>  
print type(c)   #实例的类别,输出<class '\_\_main\_\_.C'>  
print type(C.foo)   #非绑定方法的类别，输出<type 'instancemethod'>  
print type(c.foo)   #绑定方法的类别,输出<type 'instancemethod'>  
print C.foo         #非绑定方法对象，输出<unbound method C.foo>  
print c.foo         #绑定方法对象，输出<bound method C.foo of <\_\_main\_\_.C object at 0x0130D830>>

**3、类**

**4、类的实例**

通过类的\_\_call\_\_()特殊方法，必须在类中定义这个方法才行。

[复制代码](javascript:void(0);)

class C(object):

def \_\_call\_\_(self,\*args):

return u'可调用实例'

c = C()

print callable(c) #True

print c() #调用实例输出，"可调用实例"

[复制代码](javascript:void(0);)

三、代码对象

代码对象是编译过的Python 源代码片段，它是可执行对象。通过调用内建函数compile()可以得到代码对象。代码对象可以被 exec 命令或 eval\_r()内建函数来执行。

[参考](http://www.cnblogs.com/btchenguang/archive/2012/08/29/2662571.html)：http://www.cnblogs.com/btchenguang/archive/2012/08/29/2662571.html

四、可执行对象的声明与内建函数

其他见《Python核心编程2》

# [Python生成器](http://www.cnblogs.com/BeginMan/p/3509983.html)

## Python生成器

什么是python生成器，这点在核心编程中说的很清楚，意思是带有一个yield语句的**函数**，既然它是个函数，那么与普通的函数有什么关系呢？

### 与普通函数的关系：

首先确定一点就是，生成器就是函数，是一个特殊的函数，生成器和函数的主要区别在于函数 return a value，生成器 yield a value同时标记或记忆 point of the yield 以便于在下次调用时从标记点恢复执行。 yield 使函数转换成生成器，而生成器反过来又返回迭代器。

要弄懂上述关系，除了看核心编程之外，我推荐：

[**AstralWind的“Python函数式编程指南（四）：生成器”**](http://www.cnblogs.com/huxi/archive/2011/07/14/2106863.html)

那么它与迭代器有什么区别呢。

### 与迭代器的区别：

生成器就是一种迭代器。生成器拥有next方法并且行为与迭代器完全相同，这意味着生成器也可以用于Python的for循环中。

Python的for循环中，有next()调用和对StopIteration的处理，这样使得生成器和迭代器运行表象上看似相同，如下：

iterator = [i for i in range(5)]

for obj in iterator:

print obj,

# 0 1 2 3 4

generator = (i for i in range(5))

for obj in generator:

print obj,

#0 1 2 3 4

当你使用一个列表生成式来建立一个列表的时候，就建立了一个可迭代的对象， 看起来除了把 [] 换成 () 外没什么不同。但是，你不可以再次使用 for i in mygenerator , 因为生成器只能被跌代一次:先计算出0,然后继续计算1,然后计算4,一个跟一个的。

**注意：** 所有你可以使用 for .. in .. 语法的叫做一个迭代器:链表，字符串，文件... 你经常使用它们是因为你可以如你所愿的读取其中的元素，但是你把所有的值都存储到了内存中，如果你有大量数据的话这个方式并不是你想要的。

生成器的初级研究就先到这，等运用实际项目中再深入！

**参考资料：**

[Python函数式编程指南（四）：生成器](http://www.cnblogs.com/huxi/archive/2011/07/14/2106863.html)

[(译)Python关键字yield的解释(stackoverflow)](https://pyzh.readthedocs.org/en/latest/the-python-yield-keyword-explained.html)

# [Python 函数式编程](http://www.cnblogs.com/BeginMan/p/3509985.html)

## 前言

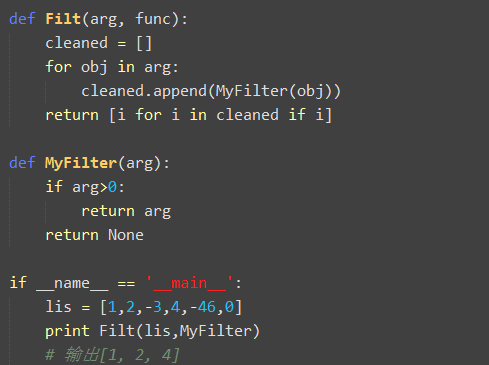
《core python programming 2》说：

Python不大可能会成为一种函数式编程语言，但它支持许多有价值的函数式编程语言构建。

然后就开始介绍lambda与四种内建函数。 对于新手来说，就有点迷惑了，到底什么是函数式编程了？？ 于是google如下：

函数式编程是一种编程模型，他将计算机运算看做是数学中函数的计算，并且避免了状态以及变量的概念。[wiki](http://en.wikipedia.org/wiki/Functional_programming)

我们知道，对象是面向对象的第一型，那么函数式编程也是一样，函数是函数式编程的第一型。在面向对象编程中，我们把对象传来传去，那在函数式编程中，我们要做的是把函数传来传去,而这个，说成术语，我们把他叫做高阶函数。[飞林沙](http://www.cnblogs.com/kym/archive/2011/03/07/1976519.html)

在[飞林沙](http://www.cnblogs.com/kym/archive/2011/03/07/1976519.html)的一篇博客中（[函数式编程扫盲篇](http://www.cnblogs.com/kym/archive/2011/03/07/1976519.html)）,我们探求高阶函数：  


该文说： 在数学和计算机科学中，高阶函数是至少满足下列一个条件的函数:

* 接受一个或多个函数作为输入
* 输出一个函数

重点：

**在函数式编程中，函数是基本单位，是第一型，他几乎被用作一切，包括最简单的计算，甚至连变量都被计算所取代。在函数式编程中，变量只是一个名称，而不是一个存储单元，这是函数式编程与传统的命令式编程最典型的不同之处。**

在上面的代码中，我们可以简写如下：

lis = [1,2,-3,4,-46,0]

print map(lambda x:0 if x<0 else x, lis)

该文又介绍了函数式编程与递归，递归是函数式编程的一个重要的概念，循环可以没有，但是递归对于函数式编程却是不可或缺的。

递归其实就是将大问题无限地分解，直到问题足够小。 而递归与循环在编程模型和思维模型上最大的区别则在于：

**循环是在描述我们该如何地去解决问题。**

**递归是在描述这个问题的定义。**

## 学习

[阮一峰的网络日志中说](http://www.ruanyifeng.com/blog/2012/04/functional_programming.html)：也许继"面向对象编程"之后，"函数式编程"会成为下一个编程的主流范式（paradigm）。

**并且总结了函数式编程具有五个鲜明的特点。：**

1. 函数是"第一等公民"
2. 只用"表达式"，不用"语句"
3. 没有"副作用"
4. 不修改状态
5. 引用透明

**意义如下：**

1. 代码简洁，开发快速

函数式编程大量使用函数，减少了代码的重复，因此程序比较短，开发速度较快。

1. 接近自然语言，易于理解
2. 更方便的代码管理
3. 易于"并发编程" 函数式编程不需要考虑"死锁"（deadlock），因为它不修改变量，所以根本不存在"锁"线程的问题。不必担心一个线程的数据，被另一个线程修改，所以可以很放心地把工作分摊到多个线程，部署"并发编程"（concurrency）。

5.代码的热升级

函数式编程没有副作用，只要保证接口不变，内部实现是外部无关的。所以，可以在运行状态下直接升级代码

## Python下函数式编程

在理解了上面的文字后，不难发现前言部分所言的Python不大可能会成为一种函数式编程语言，但它支持许多有价值的函数式编程语言构建。

因为python一切皆对象的原则，但是python也提供了函数式编程的支持，就如函数式语言Lisp兴起之后各大热门编程语言都凑热闹纷纷提供对函数式编程的支持一样，Python当然也会分得一杯羹啊。

那下面就学习Python体现函数式编程的方面吧：

### 方面1：lambda表达式

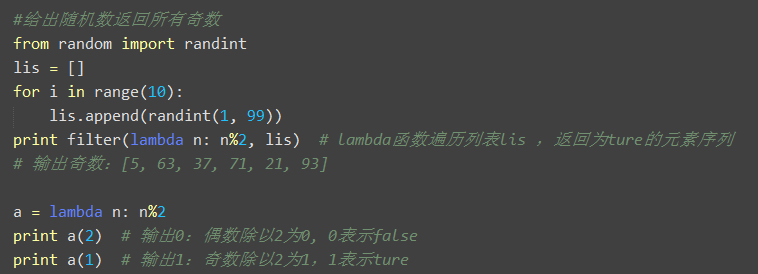
体现原理就是python允许lambda关键字创建匿名函数，所谓匿名就是python不会创建名称空间，lambda表达式返回可调用的函数对象，这些都符合函数式编程的思想。

### 方面2：函数式编程的内建函数：apply()、filter()、map()、reduce()

它们都会将函数作为参数。这里重点学习后面三个。

**filter(func,seq):**

调用一个布尔类型的函数func来迭代遍历每一个seq中的元素，返回一个是func返回值为ture的元素的序列。

示例如下：  


当然还可以重构如下，更加简洁：

# 重构

from random import randint as r

print [i for i in [r(1, 99) for j in range(10)] if i%2]

**map(func, seq1[,seq2...]):**

将函数func作用于给定序列的每一个元素，并用一个列表来提供返回值；如果func为None，func表现为一个身份函数，返回一个含有每个序列中元素集合的n个元祖的列表。

一个简单的例子：

# map 针对一个序列

print map(lambda a:a+2, [4,5,6])

# [6, 7, 8]

# 重构

print [i+2 for i in [4,5,6]]

#map 针对多个序列

#在第一次调用时，map()会将每个序列对应位置的元素捆绑到一个元祖中，将func函数作用到map()上

#当map()已经完成执行时，将元祖的结果返回到mapped\_seq映射的，最终以整体返回的序列上。

print map(lambda a,b:a+b, [1,2,3],[4,5,6])

# [5, 7, 9]

# print map(lambda a,b:a+b, [1,2,3],[4,5])

# 出错：TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'

#None与map()

print map(None,[1,2,3],[4,5,6]) #[(1, 4), (2, 5), (3, 6)]

#重构

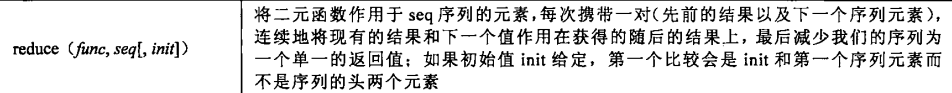
print zip([1,2,3], [4,5,6])

#[(1, 4), (2, 5), (3, 6)]

对于上述的zip:zip([seql, ...])接受一系列可迭代对象作为参数，将对象中对应的元素打包成一个个tuple（元组），然后返回由这些tuples组成的list（列表）。若传入参数的长度不等，则返回list的长度和参数中长度最短的对象相同。 可看这篇博客[Python零碎知识(2):强大的zip](http://www.cnblogs.com/BeginMan/archive/2013/03/14/2959447.html)

print zip([1,2],[1,2,3,4,5]) #[(1, 1), (2, 2)]

print map(None,[1,2],[1,2,3,4,5]) # [(1, 1), (2, 2), (None, 3), (None, 4), (None, 5)]

**reduce(func,seq[,init]):**  


理解方式可以用下面Python代码实现：

"""纯Python模拟reduce()"""

def Myreduce(bin\_func,seq,init=None):

Iseq = list(seq)

if init is None:

res = Iseq.pop(0)

else:

res = init

for obj in Iseq:

res = bin\_func(res,obj)

return res

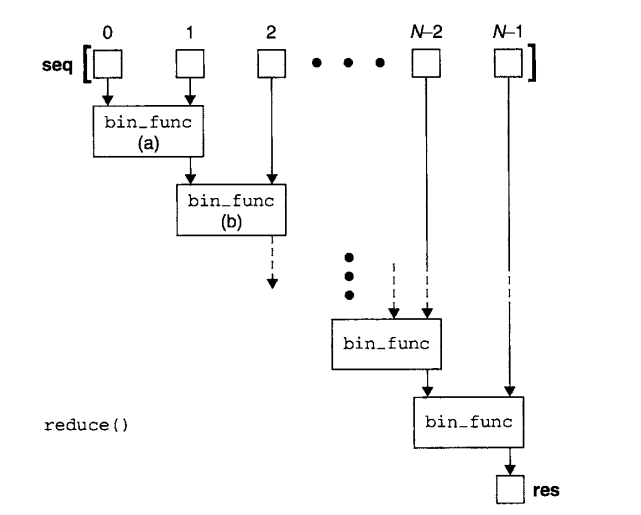
print Myreduce(lambda x,y:x+y, [i for i in range(10)]) #45

print Myreduce(lambda x,y:x+y, [i for i in range(10)],100) #145

"""reduce()"""

print reduce(lambda x,y:x+y,[i for i in range(10)]) #45

print reduce(lambda x,y:x+y,[i for i in range(10)],100) #145

其工作方式如下图：  


参考资料：

[函数式编程初探（阮一峰）](http://www.ruanyifeng.com/blog/2012/04/functional_programming.html)

[函数式编程扫盲篇](http://www.cnblogs.com/BeginMan/admin/%E9%A3%9E%E6%9E%97%E6%B2%99)[3](http://www.cnblogs.com/kym/archive/2011/03/07/1976519.html)