# Python基础教程

**计算机革命起源于机器，因此，编程语言的产生也始于对机器的模仿。**

**所有编程语言都提供抽象机制。**

Python定义：

Python是一种面向对象、直译式的计算机程序语言。它包含了一组功能完备的标准库，能轻松完成很多常见的任务。它的语法简单，与其它大多数程序设计语言使用大括号不一样，Python使用缩进来定义语句的块。（来自维基百科的定义）

Python设计哲学：

Python的设计哲学是“优雅”、“明确”、“简单”。

关于Python的一些方法技巧：

**help()**

**inspect模块**

inspect用途：

来自doc的定义：The [inspect](mk:@MSITStore:C:\Users\asus\AppData\Local\Programs\Python\Python35\Doc\python352.chm::/library/inspect.html" \l "module-inspect) module provides several useful functions to help get information about live objects such as modules, classes, methods, functions, tracebacks, frame objects, and code objects.

**文档 documentation**

## 学习Python语法前要弄清楚的问题

### 为什么使用Python

软件质量

优雅、简单、明确

Python更注重可读性、一致性、和软件质量，从而与脚本语言世界中的其他工具区别开来。

Python支持面向对象。即支持软件开发的高级重用机制。

提高开发者效率

更少的代码实现,Python程序立即运行，无需传统编译。

程序的可移植性

跨平台

标准库的支持

Python内置了众多预编译并可移植的功能模块，这些功能模块叫做标准库。标准库支持一系列应用级的编程任务。

组件集成

Python脚本可通过灵活的集成机制轻松地与应用程序的其他部分进行通信。

Python技术上的优点

面向对象

免费

可移植

功能强大

动态类型

自动内存管理

大型程序支持

内置对象类型

内置工具

库工具

第三方工具

简单易用

简单易学

### Python程序是如何运行的？

#### 字节码编译

字节码：字节码是源代码底层的、与平台无关的表现形式。Python字节码不是机器的二进制代码。

编译：编译是一个简单的翻译步骤，当程序执行时，Python内部会先将源代码编译成所谓字节码的形式。字节码编译是自动完成的。

#### Python虚拟机PVM

PVM是Python运行的引擎，字节码要在PVM中执行。

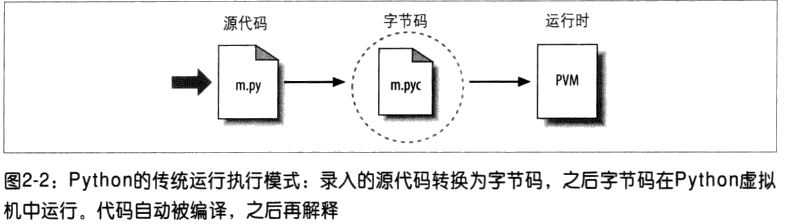
### Python解释器简介

Python运行时结构

了解一些Python运行时的结构，可以从宏观上掌握程序的执行。

我们编写的Python代码必须在解释器中运行。

第一步是编译成字节码，第二步将字节码转发到虚拟机中。



## 第一章 基础知识

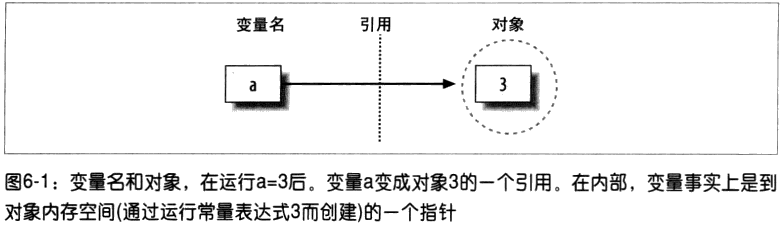
### 1.1 基础概念

#### 1.表达式:

表达式用于表示值

#### 2.变量:

变量其实是存储在内存中的值，变量就是引用或者说是代表某值的名字。创建一个变量时会在内存中开辟一个空间。Python中的变量不需要声明。



Python中，变量在赋值的时候才创建，它可以引用任何类型的对象，并且必须在引用之前赋值。

#### 3.语句:

print语句

赋值语句

#### 4.函数:

函数是组织好的，可重复利用，用来实现特定功能的小程序。

#### 5.字符串:

字符串就是字符的序列

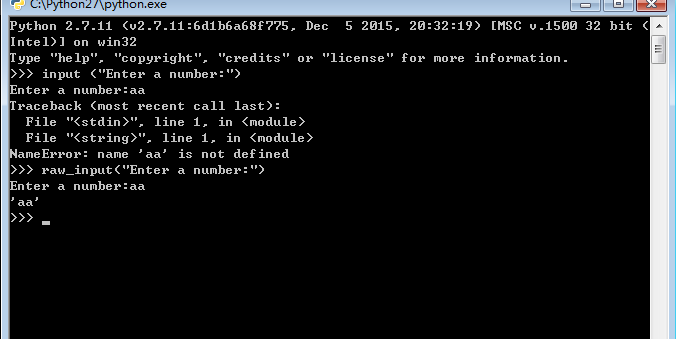
字符串表示：str和repr，是值被转换为字符串的两种机制。

str函数，它会把值转换为合理形式的字符串；

repr函数，它会创建一个字符串，以合法的Python表达式的形式来表示值。

input和raw\_input的比较：

input会假设用户输入的是合法的Python表达式，要求用户输入字符串，也就是输入的内容要用引号引起来，否则报错，如下图所示。raw\_input 则没有这样的问题。



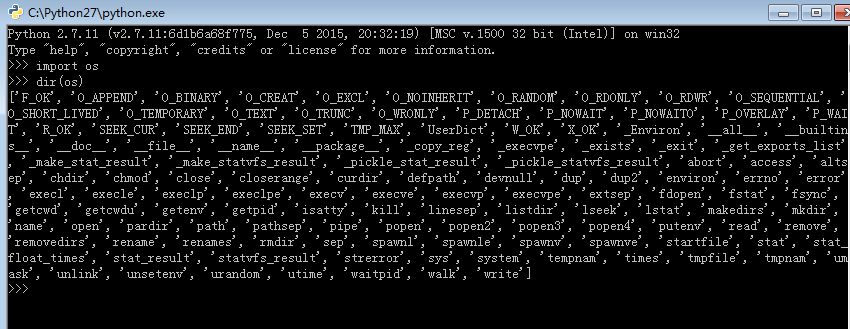
#### 6.模块

学习参考：<http://python.jobbole.com/87330/>

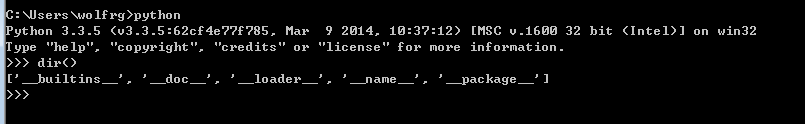
模块是处理某一类问题的函数和类的集合，扩展程序的相关功能。

使用内建的dir函数来列出模块定义的标识符。标识符有函数、类和变量。来自doc的定义：Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object.它返回传递给它的任何对象的属性名称经过排序的列表。如果不指定对象，则 dir() 返回当前作用域中的名称。

os模块的属性



当前作用域中的名称



dir() 函数适用于所有对象类型，包括字符串、整数、列表、元组、字典、函数、定制类、类实例和类方法。

参考文档:<http://www.ibm.com/developerworks/cn/linux/l-pyint/index1.html>

#### 7.Pthon3标准数据类型

Python3 有六个标准数据类型：

* Number（数字）
* String（字符串）
* List（列表）
* Tuple（元组）
* Dictionary（字典）
* Set（集合）

这六个标准类型又可以划分为三种主要类型：

数字

序列 （字符串、列表、元组）

支持索引、分片、合并等。

映射 （字典）

支持通过键的索引。

集合是自成一体的一个分类

这些主要类型又可归纳为如下两类：

不可变类型（数字、字符串、元组、不可变集合）

可变类型（列表、字典、可变集合）

## 第二章 列表和元组

### 2.1 序列概览

Python中的序列：列表、元组、字符串、Unicode字符串、buffer对象、xrange对象

列表和元组很类似，主要区别是列表可以修改，元组不能修改。

通用序列操作：索引、分片、加、乘以及成员资格。

索引：序列中的所有元素都是有编号的从0开始递增。可以通过索引获取元素。

分片：使用分片操作来访问一定范围内的元素。分片通过冒号相隔的两个索引来实现。

索引的编号在分片操作中很重要，第1个索引是需要提取部分的第1个元素的编号，第一个索引的元素是包含在分片内的；而第2个索引则是分片之后剩下部分的第一个元素的编号，也就是说

### 2.2 通用序列操作

#### 2.2.2 分片

分片原理：分片操作有两个索引作为边界，第一个索引的元素包含在分片内，第二个则不包含在分片内。

在进行分片的时候，分片的开始和结束点需要指定，还有一个参数是分片的步长，在普通的分片中，步长是1，但是步长也可以是负数，即从右到左提取元素：

例子：



偏移和分片

位置偏移从左至右，偏移0为第一个元素，而负偏移是从右至左提取元素，-1是最后一个元素。

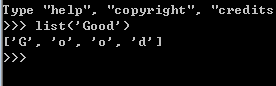
### 2.3  列表：Python的苦力

列表是可变的有序的集合

Python散列表

### 2.3.1  list() 函数：

list函数将字符串创建为列表：



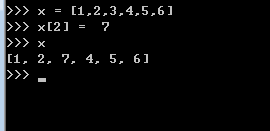
### 2.3.2  基本的列表操作

适用于序列的标准操作同样也适用于列表；

列表可以修改；

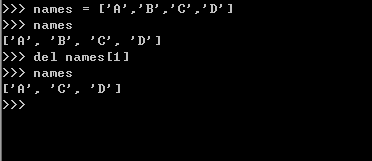
改变列表的方法：元素赋值、元素删除、分片赋值以及列表的方法

1.改变列表：元素赋值



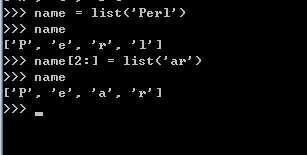
2. 删除元素

使用del语句来删除列表中的元素：

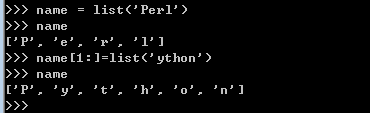


3. 分片赋值

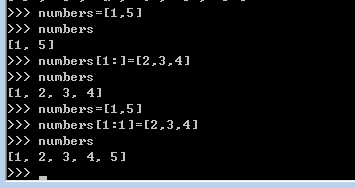
1）分片赋值是个非常强大的特性。



2）不等长分片赋值：

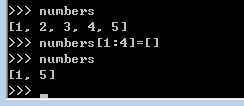


3）分片赋值插入新的元素：



观察上面两种结果的差别

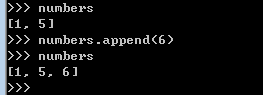
4）分片删除元素：



### 2.3.3 列表方法

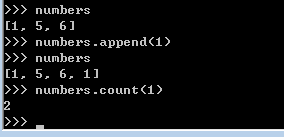
#### 1. append

 append方法用于在列表末尾追加一个新的元素：



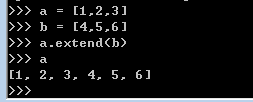
#### 2. count

 count方法用来统计某个元素在列表中出现的次数：



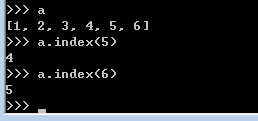
#### 3.extend

  extend方法可以用新列表扩展原有的列表：



#### 4. index

 index方法用于从列表中找出某个值第一个匹配项的索引位置：



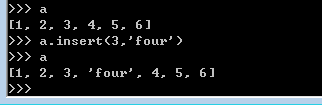
#### 5. insert

list.insert(index, obj)

ndex -- 对象 obj 需要插入的索引位置。

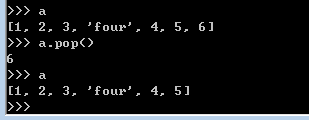
obj -- 要插入列表中的对象。

insert方法用于将对象插入到列表中：



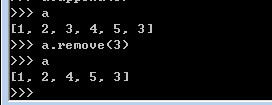
#### 6. pop

pop方法会移除列表中的一个元素（默认是最后一个），并且返回该元素的值：



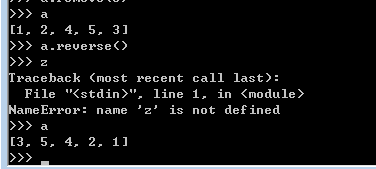
#### 7. remove

remove方法用于移除列表中某个值的第一个匹配项：



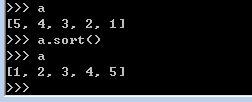
#### 8. reverse

reverse方法用于将列表中的元素反向存放：



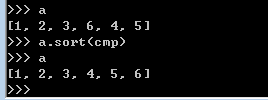
#### 9. sort

sort方法用于在原位置对列表进行排序，原位置意味着改变原来的列表：



#### 10. 高级排序

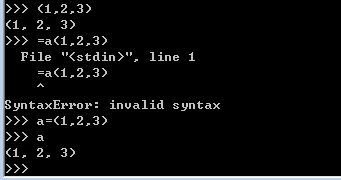
使用cmp(x,y)函数



### 2.4 元组：不可变序列

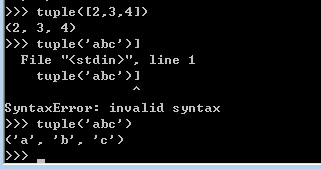
元组也是一种序列但不能修改

元组通过圆括号括起来



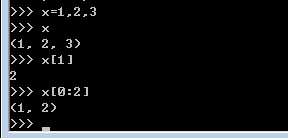
### 2.4.1 tuple 函数

tuple函数的功能与list函数基本一样：以一个序列作为参数并把它转化为元组，如果参数是元组，原样返回。

s

### 2.4.2  基本元组操作

创建和访问元组



### 2.4.3  元组的意义

元组可以在映射中当作键使用而列表则不行

元组作为很多内建函数和方法的返回值存在

## 第三章 使用字符串

### 3.1  基本字符串操作

所有标准的序列操作对字符串同样适用，但是字符串不是不可变的，因此，对字符串分片赋值是不合法的。

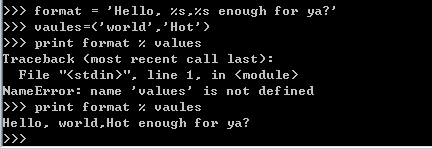
### 3.2  字符串格式化：

%是字符串格式化操作符

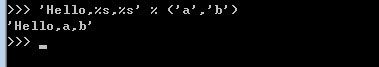
%的用法：%左侧放置一个字符串，右侧放置希望格式化的值，可以使用一个值，如一个字符串或者是数字，也可以使用多个值的元组或者字典。

%s称为转换说明符，标记了需要插入转换值的位置，s表示值会被格式化为字符串。

例1



例2



### 3.3  字符串格式化：完整版

格式化操作符的右操作数可以是任何东西，如果右操作数是元组的话，则其中的每一个元素都会被单独格式化

### 3.3.1 简单转换

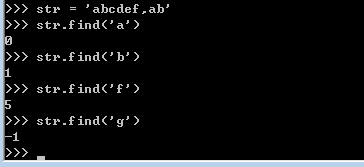
C:\Users\wolfrg\AppData\Local\YNote\data\qqF81001B3D1AE6661D6ADA11171A5F28E\179a307282de4bc5851970c7563ebafc\clipboard.png

### 3.3.2  字段宽度和精度 （暂时略过）

### 3.4 字符串方法

#### 3.4.1 find

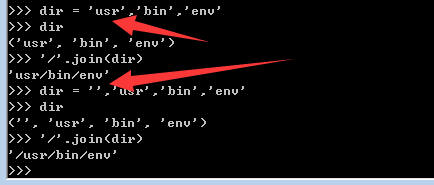
find方法可以在一个较长的字符串中查找子字符串。它返回子串所在位置的最左端索引。



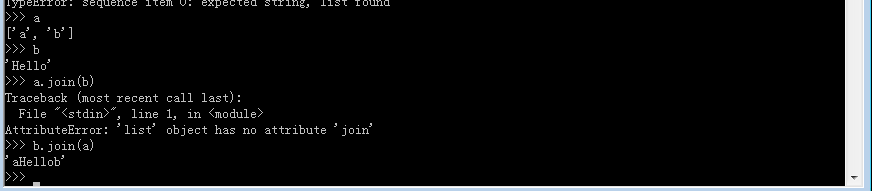
#### 3.4.2 join

join方法是非常重要的字符串方法，它是split方法的逆方法，用来连接序列中的元素，将序列中的元素以指定的字符连接生成一个新的字符串。

例1：用"/"把字符串连接起来

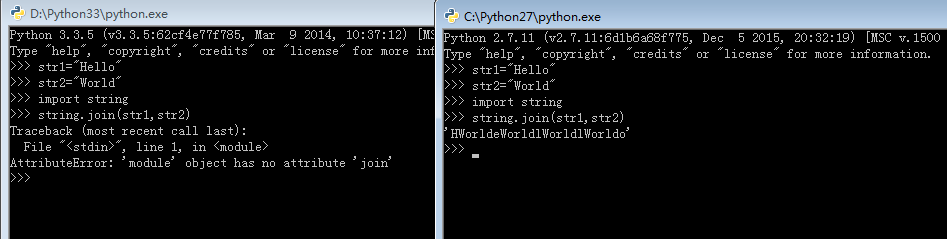


列2：将字符串b用列表a连接生成一个新的字符串。



列3：string.join

在python3中import string string.join(str1,str2)不能那么用



#### 3.4.3 lower

     lower方法返回字符串的小写字母版

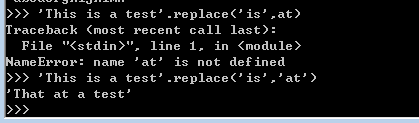
C:\Users\wolfrg\AppData\Local\YNote\data\qqF81001B3D1AE6661D6ADA11171A5F28E\8a55374011f342fda2ad67a3f83d27fb\clipboard.png

如果想编写“不区分大小写”的代码的话，就可以用lower方法

#### 3.4.4 Python字符串替换

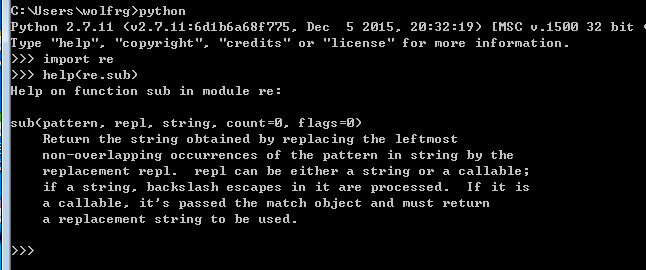
##### str.replace()

replace方法返回某个字符串的所有匹配项均被替换完之后得到的字符串，两个参数，被替换和替换



##### re.sub()

help一下



Python 的re模块提供了re.sub用于替换字符串中的匹配项。

语法：

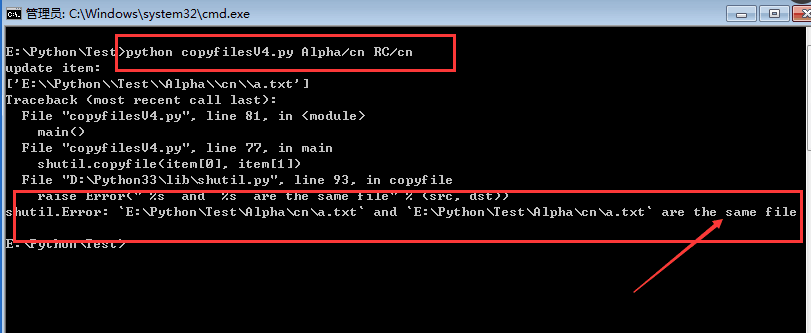
re.sub(pattern, repl, string, count=0)

参数：

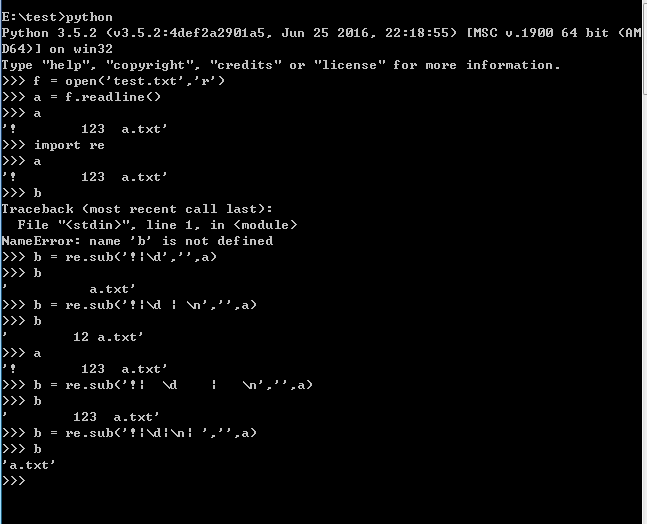
* pattern : 正则中的模式字符串。
* repl : 替换的字符串，也可为一个函数。
* string : 要被查找替换的原始字符串。
* count : 模式匹配后替换的最大次数，默认 0 表示替换所有的匹配。

比replace高级的替换方法是sub，sub用正则替换字符串。

sub存在夸平台的问题，在windows下sub对路径识别有问题。replace在linux 和windows下都没有这样的问题。具体脚本请见本人有道云笔记Python实践脚本：copyfilesV4.py



re.sub的用法



#### 3.4.5 split

这是一个非常重要的字符串方法，它是join的逆方法，用来将字符串分割成序列。

C:\Users\wolfrg\AppData\Local\YNote\data\qqF81001B3D1AE6661D6ADA11171A5F28E\fe503e16552d40439631c0ad461f6762\clipboard.png

#### string.splitlines()

按照行分隔，返回一个包含各行作为元素的列表，如果 num 指定则仅切片 num 个行。

#### 3.4.6 strip

strip方法返回去除两侧空格的字符串

C:\Users\wolfrg\AppData\Local\YNote\data\qqF81001B3D1AE6661D6ADA11171A5F28E\e262499c7e9349b09d952afc0f99e661\clipboard.png

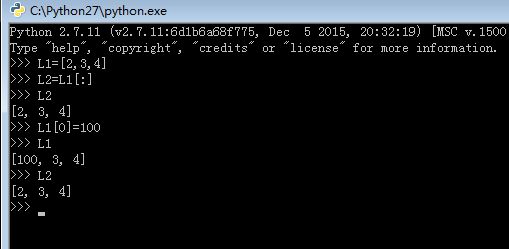
#### 3.4.7  translate

translate方法和replace方法一样，可以替换字符串中的某些部分，translate方法只处理单个字符。

复制列表的方法，从头到尾的分片是复制列表的常用方法。

例如下面的例子，L2复制了L1，L1做了修改，L2并没有变化。

这种分片技术不会应用在其他的可变的核心类型（字典和集合，因为它们不是序列）。



## 第四章 字典

字典的概念：

1，字典是通过名字引用值的一种数据结构；

2，字典是Python唯一内建的映射数据类型；

3，字典中的值并没有特殊的顺序，但是都存储在一个特定的键里。键可以是数字、字符串或者是元组。

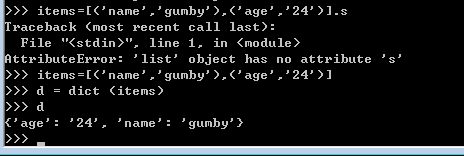
### 4.1 创建和使用字典：

字典的结构：字典由多个键值对组成。每个键和它的值之间用冒号隔开，各项之间用逗号隔开。整个字典是由一对大括号括起来。

创建字典的方法：

直接创建：pb= {'A':'123','B':'456','C':'789'}

dict函数：dict函数可以指定两个元组或列表来对应创建字典



创建字典的几种方法：

第一种直接创建：

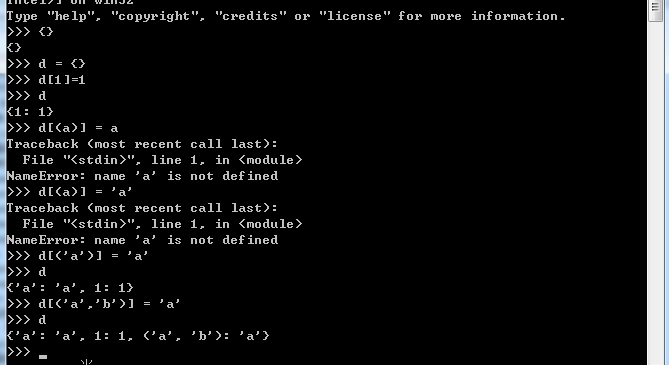
{'name':'mel','age':45}

第二种 动态的创建

D = {}

D['name'] = 'mel'

D['age'] = 45



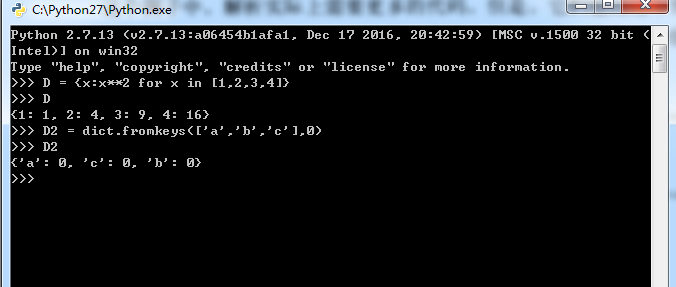
第三种 dict 函数，键为字符串

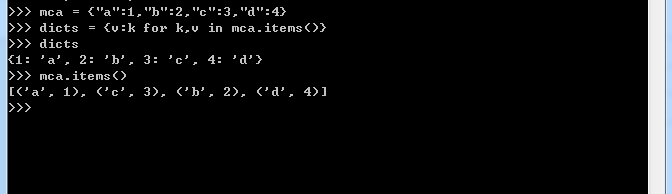
dict(name='mel',age=45)

第四种 键和值，形成序列

dict([('name','mel'),('age',45)])

第五种：字典解析的方式





### 知识点补充

推导式分为列表推导式（list），字典推导式(dict)，集合推导式(set)三种

### 4.2.2 基本字典操作：

len(d) 返回字典d中项的数量；

d[k]返回关联到键k上的值；

d[k]=v 将值v关联到键k上；

del d[k] 删除键为k的项；

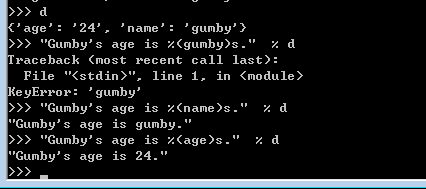
字典不同于列表的一些特性：

键类型：键可以是数字、字符串设置是元组

自动添加：即使那个键初始在字典中并不存在，也可以为它分配一个值，这样字典就会建立新的项。

成员资格：k   in  d

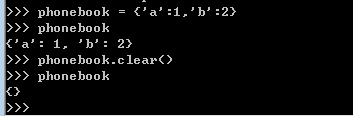
### 4.2.3 字典的格式化字符串



### 4.2.4 字典方法

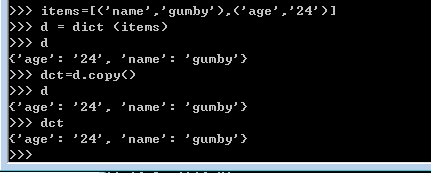
#### clear：

clear方法清除字典中所有的项



#### 2. copy：

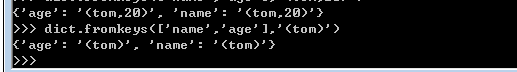
copy方法返回一个具有相同键值对的新字典；分为浅复制和深复制



http://www.runoob.com/w3cnote/python-understanding-dict-copy-shallow-or-deep.html

#### 3. fromkeys：

fromkeys方法使用给定的键建立新的字典，每个键默认对应的值为None，同时这也是初始化字典的一种方法



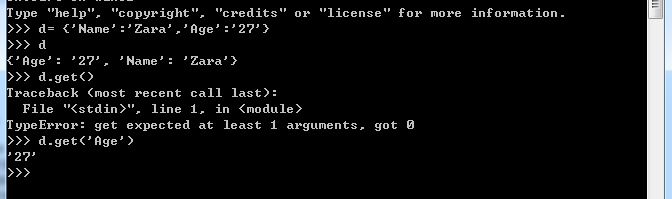
#### 4. get：

Python 字典(Dictionary) get() 函数返回指定键的值，如果值不在字典中返回默认值。

get()方法语法：

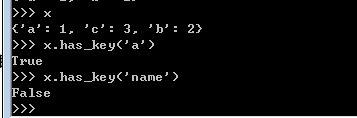
dict.get(key, default=None)

get方法是个更宽松的访问字典的方法。一般来说，如果试图访问字典中不存在的项时会出错：



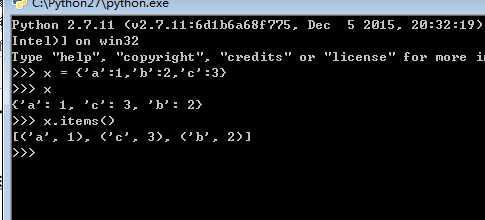
#### 5. has\_key:

has\_key 方法可以检查字典中是否含有给出的键

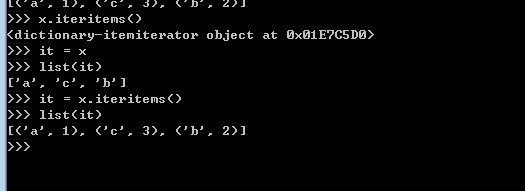


#### 6. items和iteritems

items方法将所有的字典项以列表方式返回，这些列表项中的每一项都来自于（键,值），返回项并没有顺序。



 iteritems方法的作用大致相同，但是会返回一个迭代器对象而不是列表：



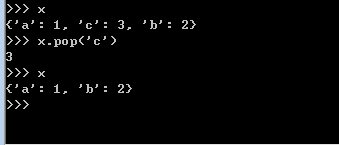
 在很多情况下使用iteritems更高效。

#### 7. keys和iterkeys

keys方法将字典中的键以列表形式返回，而iterkeys则返回针对键的迭代器。

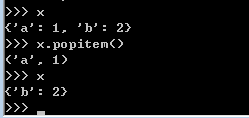
#### 8. pop

pop方法用来获得对应于给定键的值，然后将这个键-值对从字典中移除。



#### 9. popitem

popitem方法类似于list.pop,用于随机移除字典中的项，不用考虑顺序的问题，因为字典没有顺序的概念。

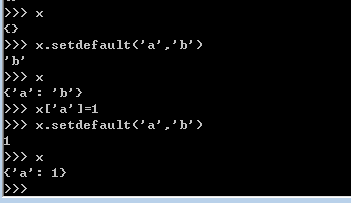


#### 10. setdefault

setdefault方法在某种程度上类似于get方法，setdefault方法的两个作用：

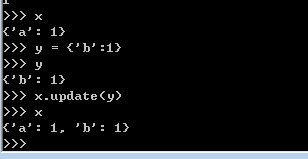
获得与给定键相关联的值

在字典中不含有给定键的情况下设定相应的键值。



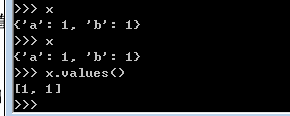
#### 11. update

update方法可以利用一个字典项更新另外一个字典：



#### 12. vaules和itervalues

values方法以列表的形式返回字典中的值，intervalues返回值的迭代器。

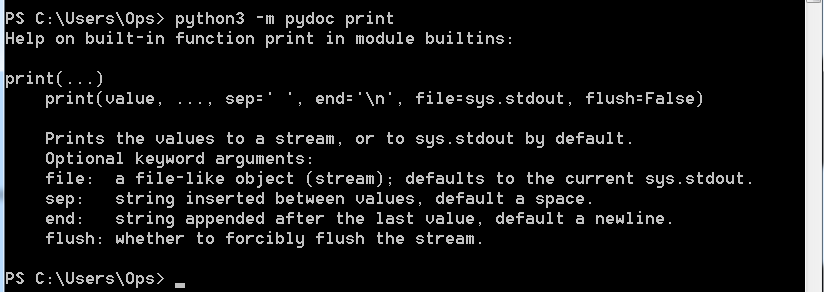


## 第五章 条件、循环和其他语句

### 5.1 print 和import的更多信息

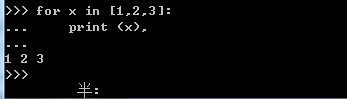
The pydoc module automatically generates documentation from Python modules.

Windows中python -m pydoc print

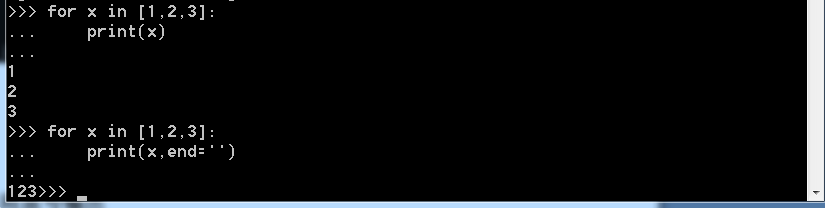


print不换行在python3和python2中打印的不同：

python2.7交互模式下



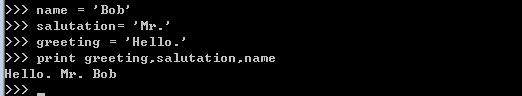
python3中：



#### 5.1.1  使用逗号输出

用逗号隔开表达式，print 打印多个表达式

例子：不同于之前的输出方式



逗号在python2.x中处理换行的使用方法请参考上一小节的例子。

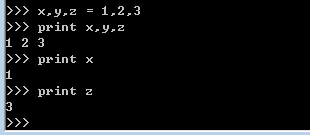
#### 5.1.2  （暂略）

### 5.2 赋值魔法

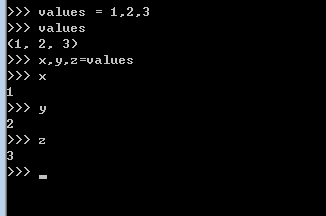
就算最不起眼的赋值语句也有一些特殊的技巧。

#### 5.2.1   序列解包

多个赋值操作同时进行：

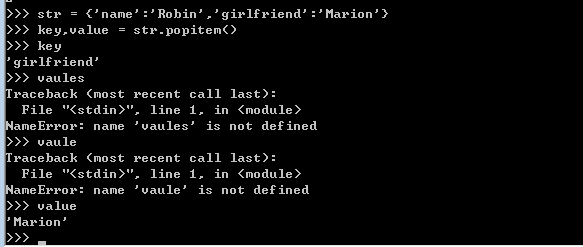


更加形象的表示：



这就是序列解包，当函数或者是方法返回元组时，这个特性尤其有用。

例子：将键值作为元组返回，这个元组可以直接赋值到两个变量中：



#### 5.2.2  链式赋值

       暂略

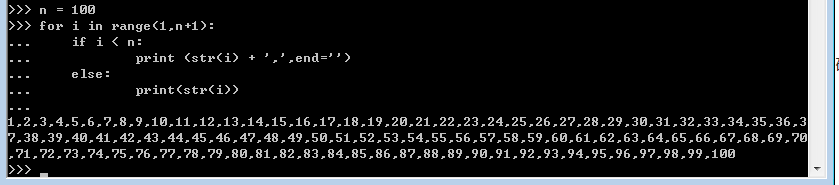
#### 5.2.3  增量赋值

### 5.3  语句块：缩排的乐趣

 语句块是在条件为真时执行或者执行多次的一组语句，在Python中用冒号(:)来标示语句块的开始。

### 5.4  条件和条件语句

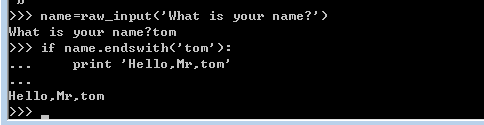
 条件语句让程序选择是否执行语句块的方法



#### 5.4.1 布尔变量的作用

    真值

#### 5.4.2   条件执行和if语句

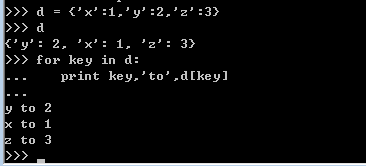


#### 5.4.3  elif 子句

如果需要检查多个条件，就可以用elif，它是 “else if”的简写，是具有条件的else子句。

### 5.5.3 循环遍历字典元素

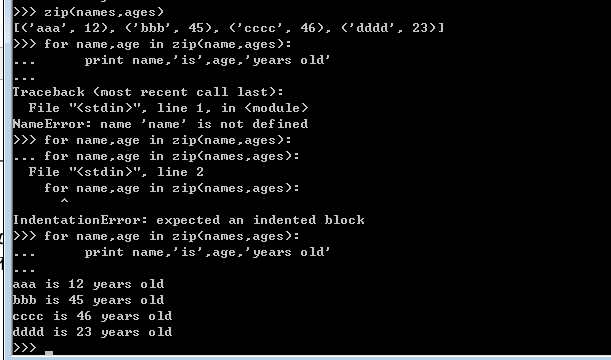
一个简单的for语句就能循环字典的所有键



### 5.5.4 一些迭代工具

1.并行迭代

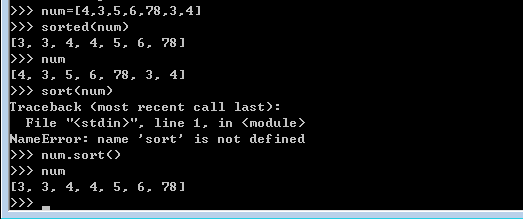
zip函数可以用来进行并行迭代



2.  编号迭代（暂略）

3. 翻转和排序迭代

两个常用的函数：reversed和sorted

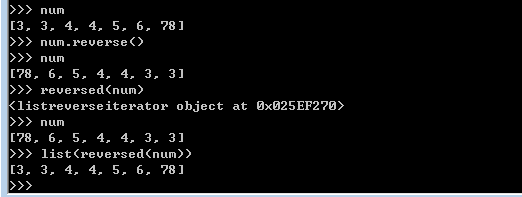


从上面的例子可以看出sort和sorted用法的区别：

sort函数是序列的内部函数，用法是list.sort()，作用是把序列原地排序，原来的序列并没有副本，改变了序列。

sorted函数是Python的内建函数，它作用于任何序列或可迭代对象上，不是原地修改对象，而是返回排序后的版本。

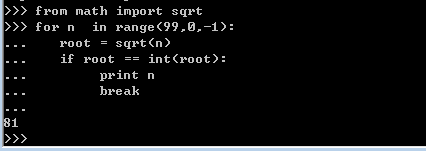
reversed函数跟sorted类似



### 5.5.5 跳出循环

1. break

结束循环可以使用break语句



2. continue

continue最基本的意思是“跳过剩余的循环体，但是不结束循环”，结束当前的迭代，跳到下一轮循环的开始

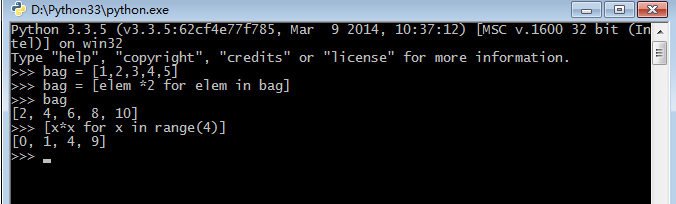
3. while True/breaek语句

### 5.5.6 循环中的else 子句

### 5.6 列表推导式——轻量级循环

列表推导式是利用其他列表创建新列表的一种方法，工作方式类似于for循环。

例子：有一个list 让list中的元素翻倍。



### 5.7  三人行 （暂略）

3个语句：pass、del、exec

#### 5.7.3  使用exec和eval执行和求值字符串

## 《Python学习手册》第六章 动态类型简介

### 本章理解的重点：

1、类型属于对象，而不属于变量

2、引用是自动形成的从变量到对象的指针

3、理解共享引用的关键是看对象的类型是否可变

4、每个对象都有一个计数器，计数器记录了当前指向该对象的引用的数目

5、Python垃圾回收机制主要基于对象内部的计数器

## 第六章抽象-创建函数

函数就是组织好的，可重复使用的，用来实现相关功能的代码段。提高了应用的模块行和代码的重复利用率。本章介绍如何将语句组织成函数。

函数可以被调用

执行并返回一个值

### 6.3  创建函数

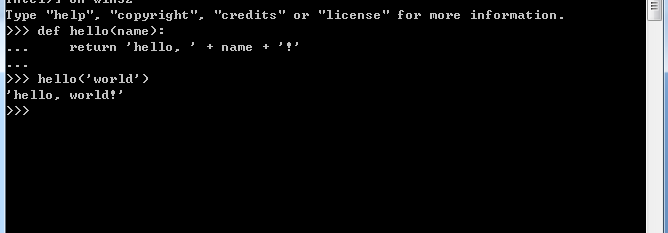
函数是组织代码的关键：

def hello(name):

       return 'Hello, ' + name + '!'

**定义一个函数的规则：**

1. 使用def定义函数，后接函数标识符名称和圆括号()，以冒号结束；
2. 任何传入参数和自变量必须放在圆括号中。圆括号之间用于定义参数；
3. 函数的第一行语句可以选择性地使用温度字符串-用于存放函数说明；
4. 函数内容以冒号开始，并且缩进；
5. return [表达式] 结束函数，选择性地返回一个值给调用方，不带表达式的return相当于返回None；
6. 函数可以被调用，它执行某种行为并且返回一个值。内建的callable函数可以用来判断函数是否可用调用。



#### 6.3.1 记录函数

### 6.4  参数魔法

参数就是变量

#### 6.4.1  值从哪里来

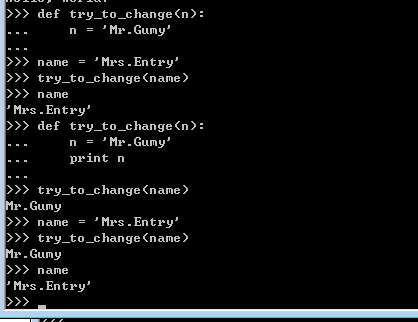
  值从调用的地方传递过来

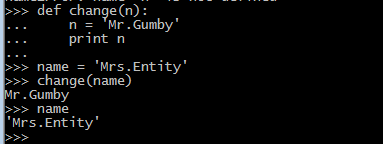
#### 6.4.2  函数作用域

       在函数内部为参数赋新值不会改变外部任何变量的值。

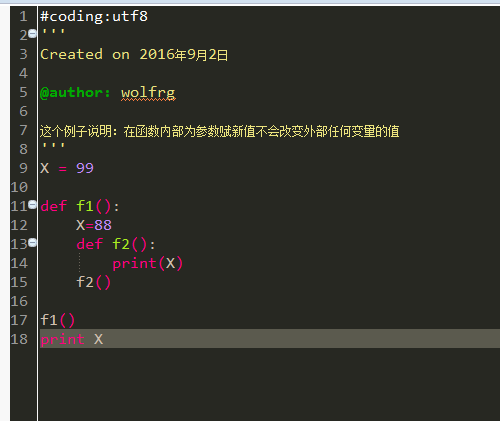
       在函数内部把参数重绑定的时候，函数外的变量不会受到影响。

例1:





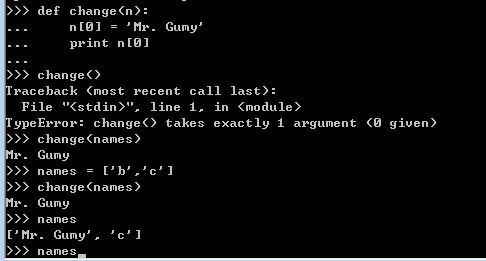
更加清晰的一个例子：



参数存储在局部作用域内（local scope）

思考问题：如果将可变的数据结构如列表用作参数时：

例2:



比较例1和例2：例子1中 字符串变量n，没有改变，原因是字符串是不可变的；例2中n是可变的列表。

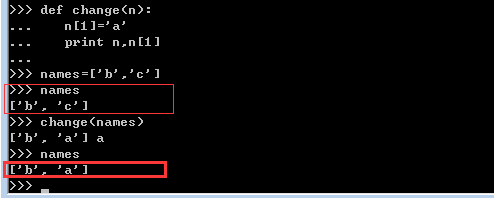
1. 为什么要修改参数

使用函数改变数据结构是将程序抽象化的好方法。

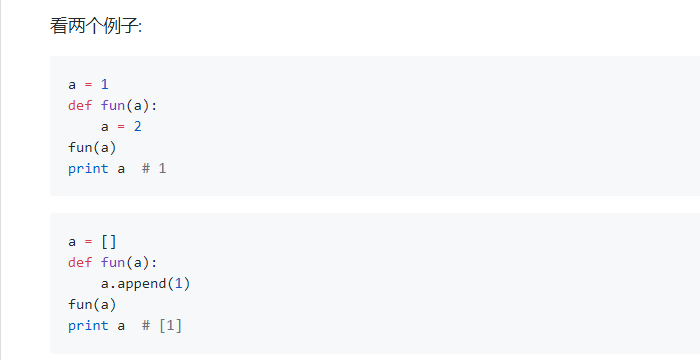
2. 参数不可变的情况：

Python中函数只能修改参数对象本身，如果参数不可变，应该从函数中返回所有需要的值

例3：元素n[1]没变，变量n改变了

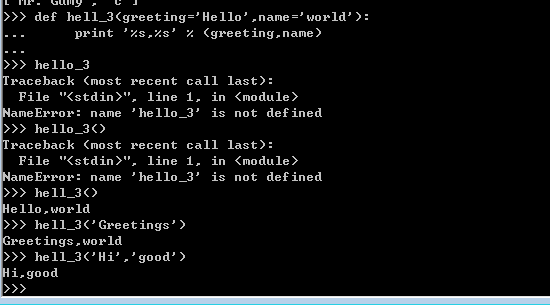


**总结：参数传递本质上和变量复制一样，只是两个变量分别为形参a和实参b。那么a=b后，a变了，b的值是否跟着变化取决于对象类型是否可变，当函数内部定义的变量指向列表这种可变的数据类型时，它会对同名的全局变量产生作用。上面的例子中names得到了改变。**



#### 6.4.3  关键字参数和默认值

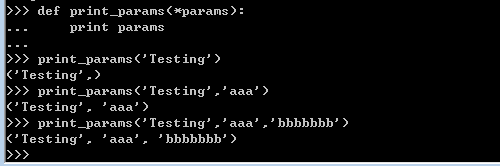
使用参数名提供的参数叫做关键字参数。它的主要作用在于可以明确每个参数的作用。关键字参数最厉害的地方在于可以在函数中给参数提供默认值：



#### 6.4.4  收集参数

1.提供任意多参数

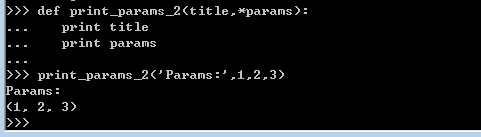
例1：你可以提供任意数量的参数，结果以元组打印出来。



返回结果是元组

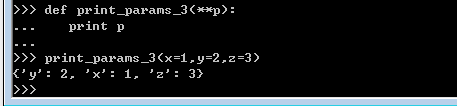
2. 联合普通参数使用

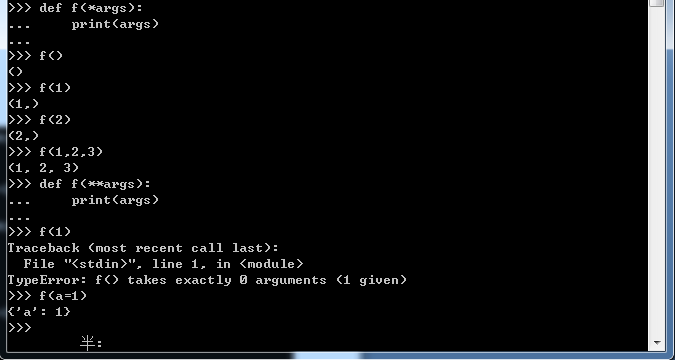
例2：



所以\*星号的意思就是"收集其余的位置参数"

3. 收集关键字参数的“收集”操作：使用\*\*

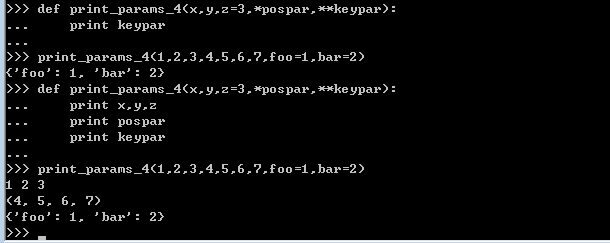




返回的结果是字典而不是元组。

总结：\*收集参数为元组，\*\*收集参数为字典

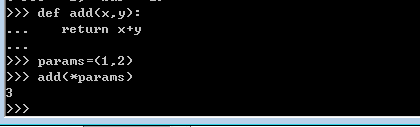
4. 放在一起用：



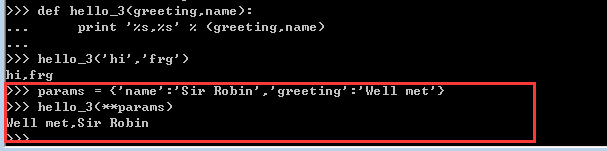
#### 6.4.5 反转过程

函数收集的逆过程：在调用函数的时候使用\*运算符

例1：



例2：



变量的作用域：分为全局变量和局部变量

定义在函数内部的变量拥有一个局部作用域，定义在函数外部的变量拥有全局作用域。

局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内部声明的变量名称都将被加入到作用域中。

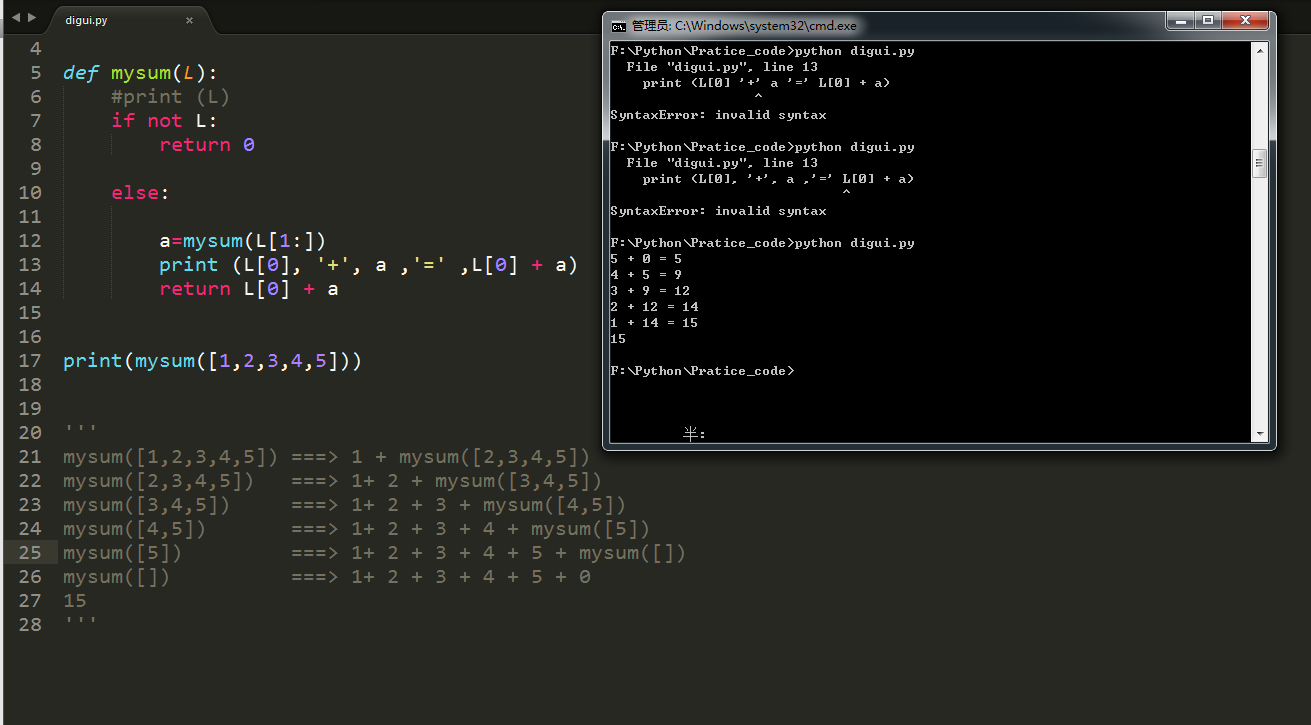
从第六章开始使用eclipse练习代码：

### 6.6 递归

在函数内部可以调用其他函数。如果一个函数在内部调用自身，这个函数就是递归函数。

练习例子：

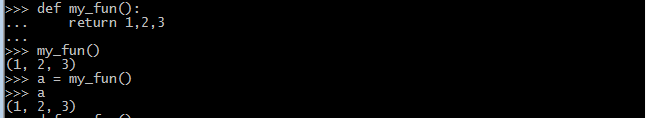
下面的这个函数递归求和，在每一层都递归调用自己来计算列表剩余的值的和，这个和随后加到前面的一项中。整个过程是先调用后计算



### 6.7函数返回值

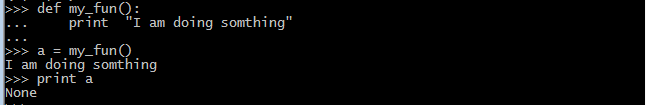
两个例子的区别：

例子1：



有返回值，函数调用类似上面

例子2：



返回值为None，函数用my\_fun()调用

### 高阶函数

lambda map filter reduce

**lambda函数**是一种匿名函数，就是说，函数没有具体的名称，是一个单行函数，在编程中，要选择使用。

语法格式：lambda 参数:操作(参数)

lambda 后面是 变量，变量后面是冒号，冒号后面是表达式。



lambda 是一个表达式，而不是一个语句

为什么使用lambda？为了代码简洁

map

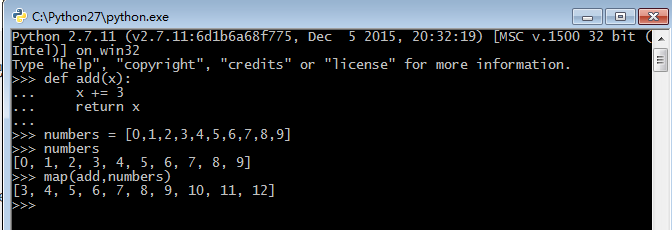
map(function, iterable, ...)

Return an iterator that applies function to every item of iterable, yielding the results. If additional iterable arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see itertools.starmap().

语法格式：map(func,seq)

map是内置函数，它期待传入一个函数。

func是一个函数，seq是一个序列对象。

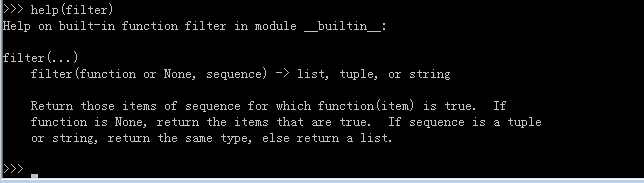


大多数时候，匿名函数lambda和map函数配合一起使用：

例如：list(map(lambda x:x+1,[100,200,300]))

filter函数

Python内置的filter()函数用于过滤序列，语法如图：



## 第七章 更加抽象 创建自己的对象

本章主要内容

### 理解几个重要概念

### 什么是类以及类的相关概念

类，用来描述具有相同的属性和方法的对象的集合。类定义了该集合中每个对象所共有的属性和方法。类是一个抽象的概念，是创建对象的蓝图，描述了所创建的对象的共同的属性和方法。

**类相关的概念:**

类变量：

类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量也可以称为静态变量，可以通过类名直接访问，也可以通过实例名访问。不能作为实例变量使用。

In [14]: class MemberCounter():

...:

...: members = 0 #类变量，静态变量

...: def count(self):

...: MemberCounter.members += 1

...:

In [15]: m1 = MemberCounter()

In [16]: m1.count()

In [17]: m1.members

Out[17]: 1

In [18]: m1.count()

In [19]: m1.members

Out[19]: 2

In [20]: m1.count()

In [21]: m1.members

Out[21]: 3

In [22]: MemberCounter.members

Out[22]: 3

类的定义取决于它所支持的方法。

实例化：创建一个类的实例，类的具体对象。

方法：类中定义和某个对象进行绑定的函数称作方法。

#### 什么是对象

两句话理解：

对象是通过类定义的数据结构的实例，对象包括两个数据成员（类变量和实例变量）和方法。方法可以操作数据。

面向对象程序设计中的术语对象（object）基本上可以看做数据（特性）以及由一系列可以存取、操作这些数据的方法所组成的集合。

#### 7.1. 对象的魔力

对象，基本上可以看做是数据（特性）以及由一系列可以存取、操作这些数据的方法所组成的集合。

使用对象替代全局变化和函数的原因有很多，其中对象最重要的优点包括以下几个方面：

多态：

封装：对外部世界隐藏对象的工作细节。

继承：

#### 7.1.3 多态

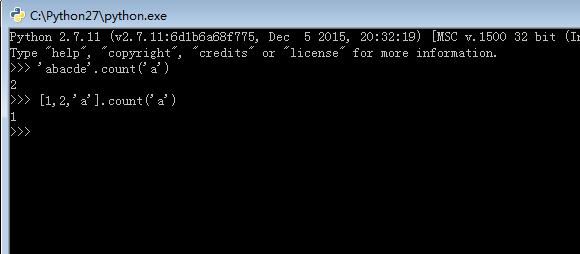
多态就是有多种形态。Python中多态是针对对象行为的。

多态意味着可以对不同类的对象使用同样的操作。例如，有Cat和Dog两个类，都继承了类Animal，他们都有talk()方法，输入不同的动物名称，会得到相同的结果；就算不知道变量所引用的对象类型是什么，还是能对他进行操作，而它也会根据对象类型的不同而表现出不同的行为。

1.多态和方法

绑定到对象特性上面的函数称为方法

例子：



例子说明：不管对象类型是什么，只要它支持count方法

2.多态的多种形式

任何不知道对象到底是什么类型，但是又要对对象“做点什么”的时候，都会用到多态。

#### 7.1.4 封装

是指向程序中的其他部分隐藏对象的具体实现细节的原则。封装不用关心对象是如何构建的，可以直接进行使用。

#### 7.1.5 继承

什么是继承？

继承就是一个派生类继承基类的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。

### 7.2 类和类型

所有编程语言都提供抽象机制。所谓的类和类型，其实是指“所抽象的是什么”。

#### 7.2.1 类到底是什么

类：用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所有共有的属性和方法。对象是类的实例。

#### 7.2.2 创建自己的类

self的理解：

类的方法与普通的函数只有一个特殊的区别，必须有一个额外的第一个参数名称，但是在调用这个方法的时候不用为这个参数赋值。Python会提供这个值，这个特别的变量指对象本身，按照惯例它的名称是self。

方法，更专业一点可以称为绑定方法，将他们的第一个参数绑定到所属的实例上。

|  |
| --- |
| #创建一个类,设置和获取名字，并打招呼。  class Person:  #定义类方法  def setName(self,name):  self.name = name  def getName(self):  return self.name  def greet(self):  print "Hello,world! I'm %s" % self.name  t = Person()  d = Person()  t.setName('王二妮')  d.setName('冯瑞钢')  t.getName()  t.greet()  d.greet() |

|  |
| --- |
| In [21]: class Person(object):  ...:  ...: def setName(self,name):  ...: self.name = name  ...: def getName(self):  ...: return self.name  ...: def greet(self):  ...: print "Hello,I'm %s" % self.name  ...:  ...:  ...:  In [22]: p1 = Person()  In [23]: p1.setName('bob')  In [24]: p1.getName()  Out[24]: 'bob'  In [25]: p1.greet()  Hello,I'm bob  In [26]: p1.name |

特性可以在外部访问

#### 7.2.3 Python中私有化的问题

Python并不直接支持私有方式，使用一些小技巧可以达到私有特性的效果。为了让方法或者特性变为私有（即无法从外部访问），只要在它的名称前面加上双下划线（\_\_）即可：

#### 代码练习及总结：

代码1：

|  |
| --- |
| class People:  name = 'jack' #定义类的属性，name和age是公有的，可以直接在类外通过对象名称访问。  age =12  p = People()  print "%s's age is %d" % (People.name,p.age)  输出结果：jack's age is 12 |

代码2：

|  |
| --- |
| class people:  \_\_name = 'jack'  \_\_age = 12  p = people()  print p.\_\_name,p.\_\_age  这段代码运行报错，提示找不到属性。 |

总结：私有属性是不能够在类外通过对象名来进行访问的。

代码3：我们可以定义一些方法来存取内部私有属性

|  |
| --- |
| class People:  \_\_name='jack'  \_\_age=20  def getName(self):  return self.\_\_name  def getAge(self):  return self.\_\_age  p = People()  print p.getName(),p.getAge()  输出结果：jack 20 |

#### 7.2.4 类的命名空间

类命名空间：所有位于class语句中的代码都在特殊的命名空间中执行；

像方法一样，类作用域内的变量也可以被所有实例访问；

类的定义其实就是执行代码块

In [28]: class C:

...: print 'Class C being defined...'

...:

Class C being defined...

In [29]:

#### 7.2.5 指定超类

子类可以扩展超类。扩展方法：将其类名写在class语句后的圆括号内就可以指定超类了。

## 《Python学习手册》第十七章 作用域

命名空间

作用域

## 第八章（先跳过）

## 第九章 魔法方法、属性和迭代器

### 9.1 Python中的魔法方法

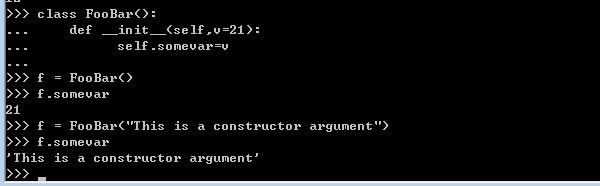
在Python中方法名有一种很特别的写法就是在方法名称的前后都加上两个下划线：\_\_classmethod\_\_，由这些名字组成的集合所包含的方法称为魔法方法。如果对象实现了魔法方法，这个特殊的方法会在特殊的情况下调用。最重要的是\_\_init\_\_方法。

### 9.2 构造方法

构造方法能简化你的工作，在类的定义中加入构造方法，当我们创建该类的实例时，会自动执行该方法。构造方法是面向对象语言共有的。

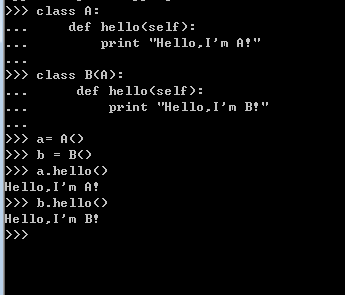
构造方法就是一种魔法方法。当一个对象被创建后，会立即调用它的构造方法。这也是它和一般方法的区别。

可以给构造方法传递参数。



#### 9.2.1 重写一般方法和构造方法

重写一般的方法：



这里用到了继承的知识。对于一般的方法：如果一个方法被B类的一个实例调用，但是在B类中并没有该方法，那么就会自动去它的超类A里面找，如上例；

在子类中增加功能最基本的办法就是增加方法。但也可以通过重写一些超类的方法来自定义继承的行为；

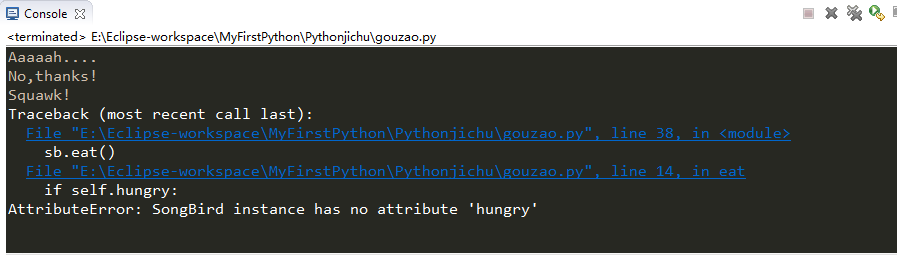
重写是继承机制中的一个重要内容，对于构造方法尤其重要。构造方法用来初始化新创建对象的状态；

对于构造方法：如果一个类的构造方法被重写，那么就需要调用超类的构造方法。

先看一段代码：

|  |
| --- |
| class Bird:  #这个类定义鸟类都具有的一些最基本的功能：吃。  def \_\_init\_\_(self):  self.hungry = True  def eat(self):  if self.hungry:  print 'Aaaaah....'  self.hungry =False  else:  print 'No,thanks!'  b = Bird()  b.eat()  b.eat()  #添加子类SongBird  class SongBird(Bird):  def \_\_init\_\_(self):  self.sound = 'Squawk!' #构造方法被重写。  def sing(self):  print self.sound  sb = SongBird()  sb.sing()  sb.eat() |

存在的问题：



解决办法：

调用超类的构造函数和使用super函数

|  |
| --- |
| \_\_metaclass\_\_ =type  class Bird:  #这个类定义鸟都具有的一些最基本的功能：吃。  def \_\_init\_\_(self):  self.hungry = True  def eat(self):  if self.hungry:  print 'Aaaaah....'  self.hungry =False  else:  print 'No,thanks!'  b = Bird()  b.eat()  b.eat()  #添加子类SongBird  class SongBird(Bird):  def \_\_init\_\_(self):  #Bird.\_\_init\_\_(self) #调用父类的构造函数  super(SongBird,self).\_\_init\_\_()  self.sound = 'Squawk!' #构造方法被重写。  def sing(self):  print self.sound  sb = SongBird()  sb.sing()  sb.eat() |

### 9.3 成员访问

魔法方法的集合，它可以创建行为类似于序列或映射的对象。

### 9.5 属性

什么是属性？通过访问器定义的特性被称为属性。

示例代码：

class Rectangle:

def \_\_init\_\_(self):

self.width = 0

self.height = 0

def setSize(self,size):

self.width,self.height = size

def getSize(self):

return self.width,self.height

r = Rectangle()

r.width = 10

r.height = 5

print r.getSize()

r.setSize((20,40))

print r.width

#### 9.5.1 property函数

Property 是一种创建属性的机制之一。

Property使用很简单，它有四个参数：fget、fset、fdel、doc，根据自己需要定义相应的函数即可。

实例代码：

\_\_metaclass\_\_ =type

class Rectangle:

def \_\_init\_\_(self):

self.width = 0

self.height = 0

def setSize(self,size):

self.width,self.height=size

def getSize(self):

return self.width,self.height

size = property(getSize,setSize)

r = Rectangle()

r.width = 20

r.height = 30

print r.size

r.size=150,100 #赋值

print r.width

#### 9.5.2 静态方法和类成员方法

静态方法：静态方法的定义没有self参数，且能够被类本身直接调用。创建时装入Staticmethod类型的对象中。

类成员方法：类方法在定义时需要名为cls的参数，类成员方法可以直接用类的具体对象调用。但是cls参数是自动绑定到类的。创建时装入Classmethod类型的对象中。

静态方法和类成员方法在Python中向来不是很重要，主要原因是大部分情况下可以使用函数或者绑定方法代替。

静态函数, 类函数, 成员函数的区别：

#### 9.5.3 \_\_getattr\_\_、\_\_setattr\_\_和它的朋友们

暂时略过

### 9.6 迭代器

迭代器是访问集合内元素的一种方式。迭代器对象从集合的第一个元素开始访问，直到所有的元素都被访问一遍结束。

迭代器不能回退，只能往前进行迭代。

深入学习迭代器：

什么是迭代，什么是迭代器

迭代就是重复做一件事很多次，\_\_iter\_\_方法会返回一个迭代器，迭代器就是具有next方法的对象。

迭代器的规则

一个实现了\_\_iter\_\_方法的对象是可迭代的，一个实现了next方法的对象则是迭代器。

3.迭代器都有一个next方法

总结：

迭代器（iterator）是访问集合内元素的一种方式，提供了一种遍历类序列对象的方法。对于一般的序列，利用索引从0一直迭代到序列的最后一个元素。对象从集合的第一个元素开始访问，直到所有的元素都被访问一遍后结束。对于字典、文件、自定义对象类型等，可以自定义迭代方式，从而实现对这些对象的遍历。总之，迭起器就是定义了对对象进行遍历的方式。

### 9.7 生成器

生成器的概念

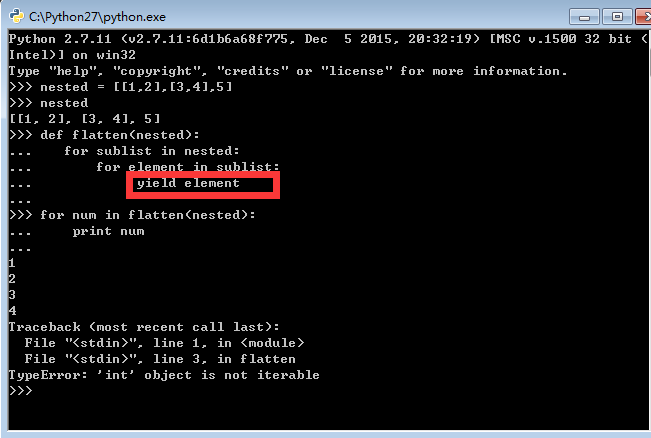
首先明确，生成器是一种迭代器，是一种用普通的函数语法定义的迭代器。生成器由两部分组成：生成器的函数和生成器的迭代器。生成器的函数是用def语句定义的，包含yield的部分，生成器的迭代器是这个函数返回的部分。两部分合起来叫做生成器。生成器必须是可迭代的。

生成器是如何工作的

生成器的用处

#### 9.7.1 创建生成器

在Python中，任何包含yield语句的函数称之为生成器。



#### 9.7.2 递归生成器

上小节的例子用了两个for循环，也就说上面的生成器只能处理两层嵌套。如果要处理任意层嵌套就要用递归生成器。处理递归时大部分都有两种情况：基本情况和需要递归的情况。

## 第十章 标准库和有用的模块

**本章学习目标：**

模块的使用

### 10.1 模块

#### 10.1.1模块是程序

任何Python程序都可以作为模块导入，模块就是Python文件。

## 第十一章 文件和素材

文件和文件夹：

文件：文本文件、二进制文件

a+

r+

w+

学习本章目标：

1. 会遍历目录和文件，进行相应的操作。
2. Python文件I/O操作

### 一、来自网上文档的总结

#### 判断文件和目录：

#heck if a path exists:

if os.path.exists("C:\\file\_mo"):

print "Yes - found it"

#Check if a path is a file

if os.path.isfile("C:\\file\_mo\\123.txt"):

print "That is a file alright"

#### os.listdir() 方法

os.listdir() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表。这个列表以字母顺序。 它不包括 '.' 和'..' 即使它在文件夹中。只支持在 Unix, Windows 下使用。但是Python os.listdir 不包括子目录中的内容。os.walk可以遍历所有目录，包括子目录。

#### 文件状态：时间和大小

import os

import datetime

def modification\_date(filename):

t = os.path.getmtime(filename)

return datetime.datetime.fromtimestamp(t)

d = modification\_date('c:/file\_mo/123.txt')

print d

#### 复制文件的操作

通过判断时间复制文件# If more than 1 second difference

if os.stat(src).st\_mtime - os.stat(dest).st\_mtime > 1:

shutil.copy2 (src, dst)

练习代码：

|  |
| --- |
| import os,shutil  def copyfiles\_bytime(src,dst):  if os.stat(src).st\_mtime-os.stat(dst).st\_mtime >1:  shutil.copy2(src, dst)  print "复制成功!"  else:  print "文件没有变化!"  copyfiles\_bytime("c:/1/1.txt", "c:/2/2.txt") |

继续思考，如何复制目录下的子目录和文件：

<http://mattsnider.com/python-script-for-automatically-copying-modified-files/>

总结思路：

根据文件的修改时间去复制文件

把源文件和目标文件都存储在列表里面

把修改的文件放到一个字典里面

Copyfil\_by\_time\_v1:

|  |
| --- |
| #coding:utf8  '''  Created on 2016年4月14日  @author: wolfrg  '''  import os  import shutil  def deleteFile(dirPath):  files = os.listdir(dirPath)  for file in files:  filePath = os.path.join(dirPath,file)  print filePath  if os.path.isfile(filePath):  os.remove(filePath)  elif os.path.isdir(filePath):  deleteFile(filePath)  os.rmdir(filePath)  print("删除成功")  #调用删除函数  deleteFile("E:\\Temp\\2")  '''  deleteFile("E:\\update-work\\zhangmen\\zhangmen-reader\\Alpha\\trunk\\WEB-INF\\classes\\cn")  removeDir("E:\\update-work\\zhangmen\\zhangmen-reader\\Alpha\\trunk\\WEB-INF\\classes\\cn")  removeDir("E:\\update-work\\zhangmen\\zhangmen-reader\\Alpha\\trunk\\WEB-INF\\lib\\")  '''  #复制的函数  def copyFiles(src, dst):  srcFiles = os.listdir(src)  filesCopiedNum = 0  #  for file in srcFiles:  src\_path = os.path.join(src, file) #把路径和路径下的文件连在一起  dst\_path = os.path.join(dst, file)  # 若源路径为文件夹，若存在于目标文件夹，则递归调用本函数；否则先创建再递归。  if os.path.isdir(src\_path):  if not os.path.isdir(dst\_path):  os.makedirs(dst\_path)  filesCopiedNum += copyFiles(src\_path, dst\_path)  print('复制静态文件夹成功')  #新增了源文件，目标文件不存在就去先复制，已经存在的文件就根据变化去时间复制。如果源文件或目录删除了，目标文件和目录还存在，那就删除目标文件和目录。  elif os.path.isfile(src\_path):  if not os.path.isfile(dst\_path):  shutil.copyfile(src\_path, dst\_path)  print "目标文件复制成功"  if os.stat(src\_path).st\_mtime - os.stat(dst\_path).st\_mtime >1:  shutil.copyfile(src\_path, dst\_path)  filesCopiedNum += 1  print('复制最新文件成功')  return filesCopiedNum  #调用函数  copyFiles("c:/1/","c:/2/") |

复制文件脚本V2

|  |
| --- |
| #coding:utf8  '''  Created on 2016年4月14日  @author: wolfrg  '''  import os  import shutil  def deleteFile(dirPath):  files = os.listdir(dirPath)  for file in files:  filePath = os.path.join(dirPath,file)  print filePath  if os.path.isfile(filePath):  os.remove(filePath)  elif os.path.isdir(filePath):  deleteFile(filePath)  os.rmdir(filePath)  print("删除成功")  #调用删除函数  #deleteFile("E:\\Temp\\2")  #判断文件类型的函数  #复制的函数  def copyFiles(src, dst):  srcFiles = os.listdir(src)  filesCopiedNum = 0  for file in srcFiles:  src\_path = os.path.join(src, file) #把路径和路径下的文件连在一起  dst\_path = os.path.join(dst, file)  # 若源路径为文件夹，若存在于目标文件夹，则递归调用本函数；否则先创建再递归。  if os.path.isdir(src\_path):  #如果目标文件夹不存在就创建文件夹  if not os.path.isdir(dst\_path):  os.makedirs(dst\_path)  filesCopiedNum += copyFiles(src\_path, dst\_path)  #print('复制文件夹成功') 在测试的时候开启这条语句  #新增了源文件，目标文件不存在就去先复制，已经存在的文件就根据时间变化去复制。如果源文件或目录删除了，目标文件和目录还存在，那就删除目标文件和目录。  elif os.path.isfile(src\_path):  if not os.path.isfile(dst\_path):  shutil.copyfile(src\_path, dst\_path)  print "目标文件复制成功"  elif os.stat(src\_path).st\_mtime - os.stat(dst\_path).st\_mtime >1:  shutil.copyfile(src\_path, dst\_path)  filesCopiedNum += 1  print('复制最新文件成功')  return filesCopiedNum  #调用函数  copyFiles("c:/1/","c:/2/")  #copyFiles("E:\\update-maven\\workspace\\zhangmen-live\\zm-live-service\\target\\zm-live-service-0.0.1\\WEB-INF","E:\\update-work\\zhangmen\\zhangmen-live\\Alpha\\trunk\\WEB-INF") |

### 文件I/O操作

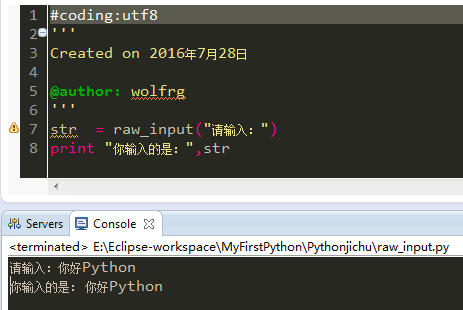
#### 1.打印到屏幕

print语句，可以打印多个表达式，用逗号隔开。

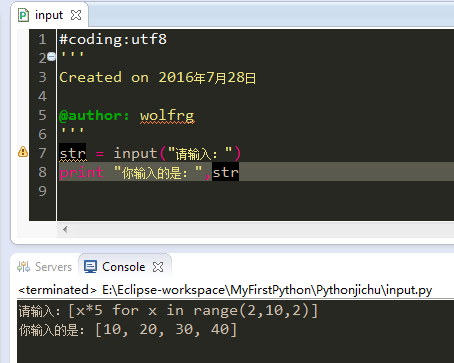
#### 2.读取键盘输入

读取键盘输入：raw\_input和input

raw\_input()函数从标准输入读取一个行，并返回一个字符串（去掉结尾的换行符）：



input可以接收一个Python表达式作为输入，并将运算结果返回。



#### 3.打开和关闭文件

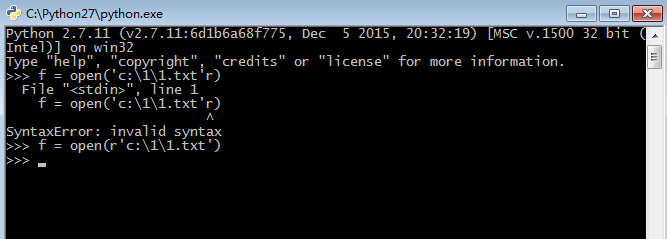
Python提供了必要的函数和方法进行默认情况下的文件的基本操作。可以用file对象做大部分的文件操作。

用open()函数打开一个文件，创建一个file对象。

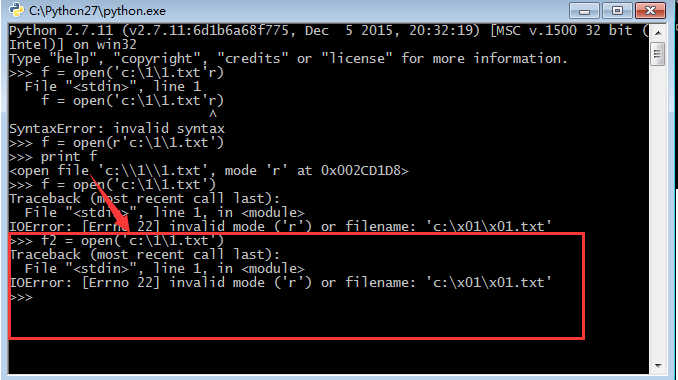
语法：file object = open(file\_name [, access\_mode][, buffering])

### 11.1 打开文件

open函数用来打开文件



### 11.1.1 文件模式



如上图所示，必须加上读写文件模式，否则会报错。这和《Python基础教程》中本章节的意思并不相符。

|  |  |
| --- | --- |
| **模式** | **描述** |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。 |
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。 |
| w | 打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| wb+ | 以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| a+ | 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。 |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 |

### 11.1.2 缓冲

open函数的第3个参数控制文件的缓冲，参数是0就是无缓冲（所有的读写操作都直接对硬盘），如果是1，就是有缓冲（意味着Python使用内存来代替硬盘，让程序更快）。

https://blog.linuxeye.com/365.html

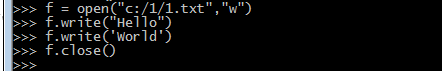
### 11.2 基本文件方法

文件对象的一些基本方法

### 11.2.1 读和写

文件（或流）最重要的能力是提供或者接受数据。如果有一个名为f的类文件对象，那么就可以用f.write 方法和f.read方法（以字符串的方式）写入和读取数据。

write()方法



read()方法

Read up to *n* bytes. If *n* is not provided, or set to -1, read until EOF and return all read bytes.

读取的方法很简单，读取指定的字节或是读到文件结尾。完成对一个文件的操作时，调用close()方法。



File对象方法：file对象提供了操作文件的一系列方法。

OS对象方法：提供了处理文件及目录的一系列方法。

### 11.2.2 管道输出

在UNIX的shell中，使用管道可以在一个命令后面继续写其他的多个命令。

### 11.2.3 读写行

readlines()和writelines()方法用于读写文件的全部的行，并返回列表。

语法格式：fileObject.readlines(sizehint);

sizehint -- 从文件中读取的字节数

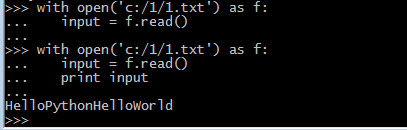
### 11.2.4关闭文件

应该牢记使用close方法关闭文件，有专门的语句来确保文件的关闭，即with语句：

语法格式：

with open(filename) as f:

input = f.read()



### 11.2.5 使用基本文件方法

read(n) 带参数

read() 不带参数

readline() 读取单独的一行，不使用任何参数

readlines() 读取一个文件中所有行并将其作为列表返回

write(string)

writelines(list) 传给它一个字符串的列表（任何序列或者是可迭代的对象都行）

### 11.3 对文件内容进行迭代

对文件内容进行迭代以及重复执行一些操作，是最常见的文件操作之一。

### 11.3.1 按字节处理

### 11.3.2 按行操作

### 11.3.3 读取所有内容

如果文件不是很大，比如一个txt的文件，可以使用readlines方法，把文件内容读入到一个字符串列表，在列表中每个字符串就是一行。

注意，将文件的内容读入到一个字符串或者是读入到列表在有些时候也很有用。比如在读取后，就可以对字符串使用正则表达式操作，也可以将行列表存入一些数据结构中，以备将来使用。

*实践代码例子：*

import pysvn

import os

import re

os.chdir('E:\\GitSVN\\example4') #切换到svn目录

#强制add没有受版本控制的文件

#把状态为delete的文件重定向到一个txt文件,每次提交完成后清空文件

os.system('svn add --force \*')

a = os.system('svn st -u . | findstr ! > E:\\GitSVN\\notdelete.txt')

#接下来就是对notdelete.txt文件的操作

#首先读取文件的每一行

lines = open("E:\\GitSVN\\notdelete.txt","r").readlines()

#使用for循环把替换后的内容写入一个新文件

fp = open("E:\\GitSVN\\delete.txt","w")

for s in lines:

fp.write(re.sub('!|\d| ','',s)) #把！符号、数字和空格替换掉

fp.close() #关闭文件

#第二步操作替换好的文件

deletefiles = open("E:\\GitSVN\\delete.txt","r").readlines()

for d in deletefiles:

os.system('svn delete %s ' % d)

#提交全部文件

os.system('svn commit -m "frg test log"')

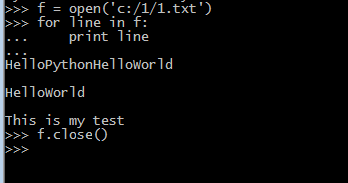
print '文件提交成功'

### 11.3.4 使用fileinput 进行迭代

对大文件进行迭代

### 11.3.5 文件迭代器

文件对象是可迭代的，也就是可以直接在for循环中使用。



### 文件操作实践总结：

#### 比较文件的操作：

## 第二章章 图形用户界面

## 第十三章 数据库支持

教程中没有以MySQL为例，根据实践，在这里总结Python操作MySQL数据库，当然Python数据库接口支持非常多的数据库。

不同的数据库需要下载不同的DB API模块，DB-API是一个规范，它定义了一系列必须的对象和数据库存取方式，以便为各种各样的底层数据库系统和多种多样的数据库接口程序提供一致的访问接口。

Python的DB-API，为大多数的数据库实现了接口，使用它连接各种数据库后，就可以用相同的方式操作各数据库。

Python DB-API使用流程：

* 引入API模块
* 获取与数据库的连接
* 执行SQL语句和存储过程
* 关闭数据库连接

### 13.1 什么是MySQLdb

MySQLdb 是Python连接MySQL数据库的模块

### 13.2 安装MySQLdb

#cd /usr/local/src/

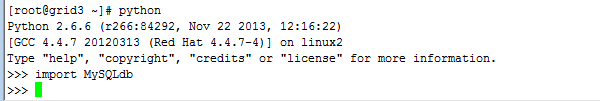
上传源码包到/usr/local/src

解压文件 #unzip MySQL-python-1.2.5.zip

#cd MySQL-python-1.2.5

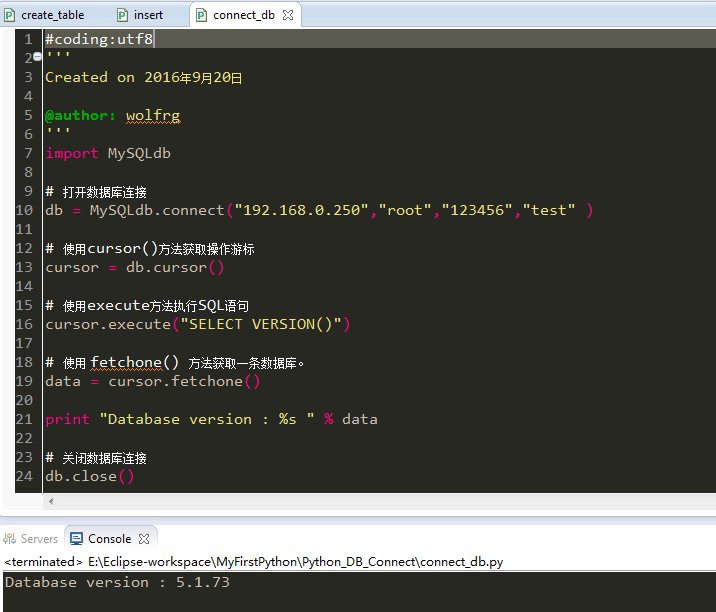
#python setup.py build

#python setup.py install



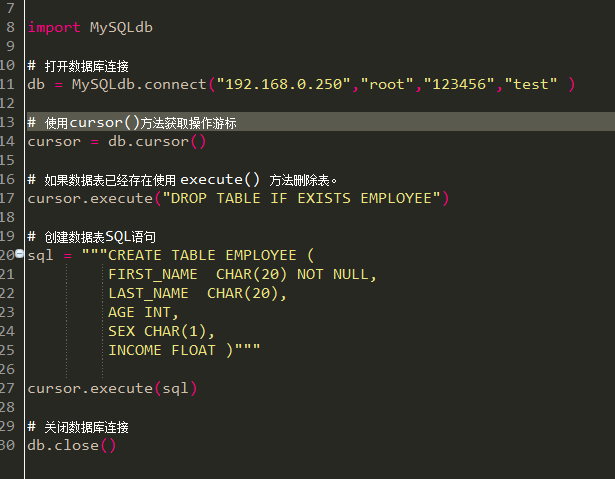
说明导入成功，否则报错。

### 13.3 数据库连接操作

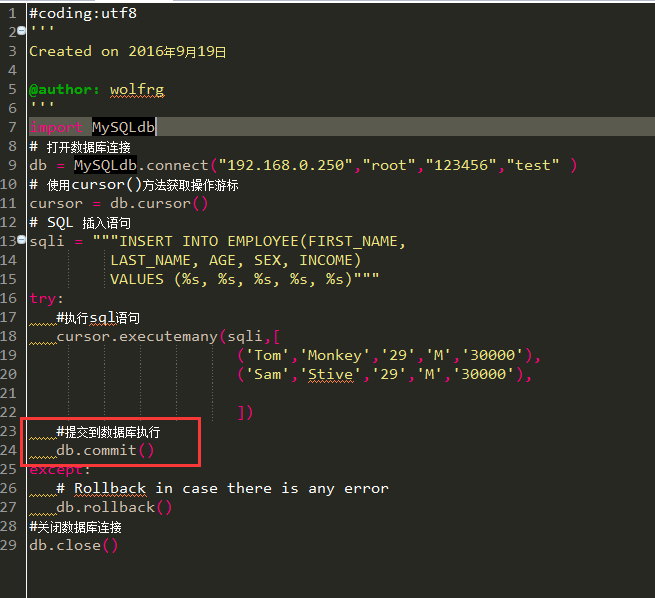


### 13.4 创建表的操作

实例：



### 13.5 插入操作



参考文档：<http://www.runoob.com/python/python-mysql.html>

## 第十四章 网络编程

### socket的知识：

套接字主要是两个程序之间的“信息通道“，是网络编程的基本组件。程序可能分部在不同的计算机上，通过套接字相互发送信息。

套接字包括两个：服务器套接字和客户机套接字。创建一个服务器套接字后，让它等待连接。

服务器套接字工作：准备随时处理客户端的连接，并处理多个连接。

客户机套接字工作：简单连接，完成事务，断开连接。

### 14.1.1 Python 中的socket模块

在网络编程中的一个基本组件就是套接字(socket)。Python标准库中有很多网络设计模块，socket模块就是其中之一。一个套接字就是一个socket模块中的socket类的实例。它的实例化需要3个参数

网络编程分为两种：阻塞或者是同步网络编程和非阻塞或者是异步网络编程

套接字有两个方法：send和recv

### 14.1.2 urllib和urllib2模块

urllib和urllib2很强大 ，它们能通过网络访问文件。

1.打开远程文件

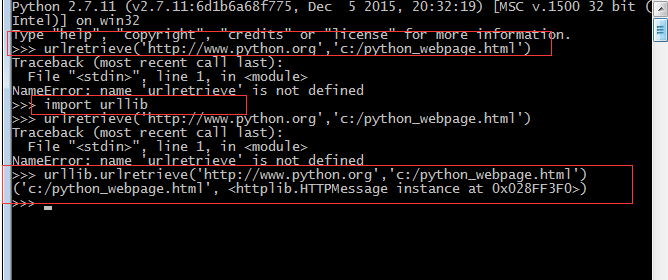
使用urlopen函数打开远程文件。

爬一个网页下来：



1. 获取远程文件

urlretrieve直接将远程数据下载保存到本地。



### 14.1.3 其他模块（暂略）

### 14.2 SocketServer和它的朋友们

SocketServer模块是标准库中很多服务器框架的基础。

SocketServer包含了4个基本的类：

针对TCP套接字流的TCPServer；

针对UDP数据报套接字的UDPServer；

以及针对性不强的UnixStremServer和UnixDatagramServer。

### 14.3 多连接、多线程、多进程

前面讨论的服务器解决方案都是同步的：即一次只能同步一个客户机并处理它的请求。能同时处理多个连接很重要。有3种主要的实现方法：分叉（forking）、线程（threading）、以及异步IO（asynchronous I/O）。

通过对SocketServer服务器使用混入类，派生进程和线程很容易处理。

分叉的理解：一个UNIX术语，分叉一个进程时，基本上是复制了它。父进程和子进程

进程的理解：对于操作系统来说，一个任务就是一个进程(Process)。

线程的理解：进程有时不止同时干一件事。在一个进程内部，要同时干多件事，就需要同时运行多个“子任务”，把进程内的这些“子任务”称为线程（Thread）。

进程和线程的关系：进程是由若干线程组成的，一个进程至少有一个线程。

多线程和多进程区别：

多进程中，同一个变量，各自有一份拷贝存在于每个进程中，互不影响；

多线程中，所有变量都有线程共享，所以，任何一个变量都可以被任何一个线程修改。

因此，线程之间共享数据最大的危险在于多个线程同时修改一个变量，把内容改乱了。

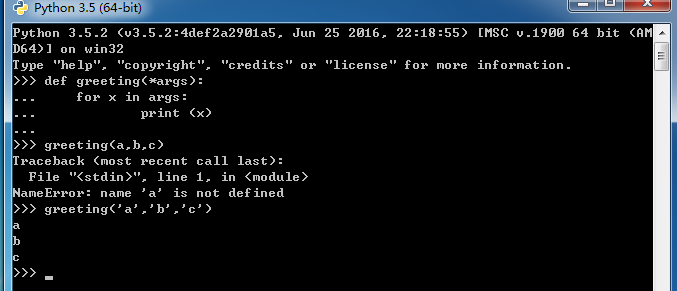
## 第十五章Python和万维网

# Python进阶

学习教程：<http://docs.pythontab.com/interpy/>

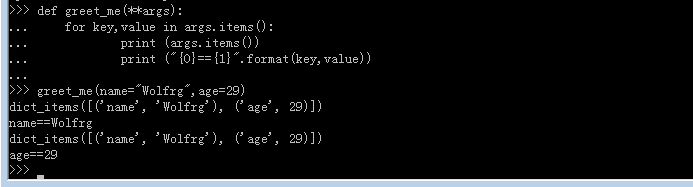
## 1.两个魔法变量：\*args 和 \*\*kwargs

\*args 是用来发送一个非键值对的可变数量的参数列表给一个函数。



\*\*kwargs 允许你将不定长度的键值对, 作为参数传递给一个函数。 也就是说如果你想要在一个函数里处理带名字的参数, 你应该使用\*\*kwargs。

一个有趣的实验：



从输入结果可以看到，dict\_items输出了两次

## 2.生成器（Generators）：

可迭代对象

迭代器

迭代

## 正则表达式

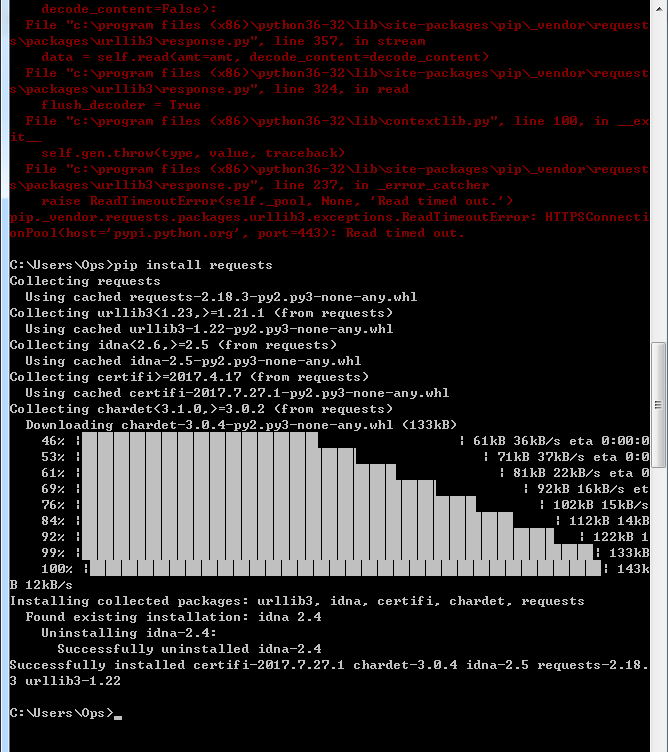
参考文档：http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

在python批量提交

# Python爬虫

安装一些库的时候，少用easy\_install，多用pip方式安装，因为esay\_install只能安装不能卸载。

有时候在网络不好的时候 pip install xx module 会报下面这样的错误，再执行一下这个命令就行。



第三方库的安装

# 运维自动化

## Python 虚拟环境及管理

[virtualenv](http://pypi.python.org/pypi/virtualenv) is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

创建独立的环境，比如在python virtualenv下安装fabric，你只有切到虚拟环境下才能执行fab命令，这样就起到一定的隔离作用。

1、安装：pip install virtualenv

2、创建：virtualenv myproject

3、激活：source myproject/bin/activate

4、退出：deactivate

Virtaulenvwrapper是virtualenv的扩展包，用于更方便管理虚拟环境:

安装：pip install virtualenvwrapper

创建目录用来存放虚拟环境

mkdir $HOME/.virtualenvs

在~/.bashrc中添加行： export WORKON\_HOME=$HOME/.virtualenvs

在~/.bashrc中添加行：source /usr/local/bin/virtualenvwrapper.sh

运行： source ~/.bashrc

此时virtualenvwrapper就可以使用了。

列出虚拟环境列表

workon

也可以使用

lsvirtualenv

进入虚拟环境：workon [虚拟环境名称]

新建虚拟环境

mkvirtualenv [虚拟环境名称]

启动/切换虚拟环境

workon [虚拟环境名称]

删除虚拟环境

rmvirtualenv [虚拟环境名称]

离开虚拟环境

deactivate

python3下命令启动虚拟环境：

实例：

在python虚拟环境下安装jumpserv v4，

cd /opt

python3 -m venv py3

source /opt/py3/bin/activate

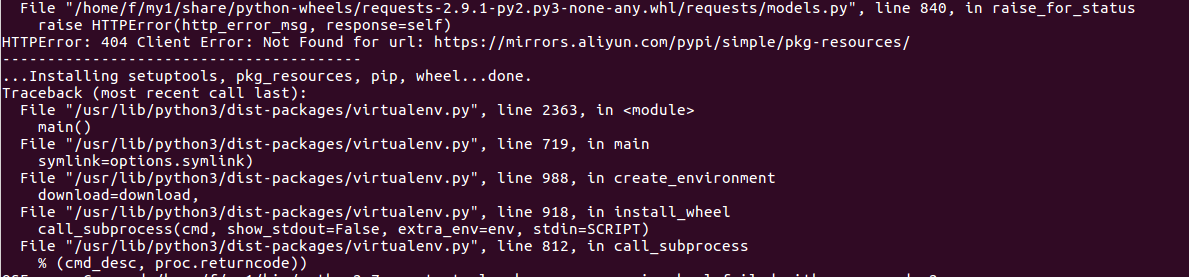
参考文档：

[https://github.com/jumpserver/jumpserver/wiki/v0.4.0-%E5%9F%BA%E4%BA%8E-CenOS7-%E7%9A%84%E7%B3%BB%E7%BB%9F](https://github.com/jumpserver/jumpserver/wiki/v0.4.0-基于-CenOS7-的系统)

Ubuntu+python虚拟环境+flask+mysql

ubuntu 16 中python2.7 创建虚拟环境:

在之前给pip做过加速，导致创建虚拟环境的创建报错如下：

解决办法，先去掉加速配置:mv ~/.pip/pip.conf pip.conf.bak 问题解决。

但是创建速度有些慢：

mkdir myvirtualenv #创建一个目录，专门存放

virtualenv my\_flask #创建你的虚拟环境

source my\_flask/bin/activate #激活虚拟环境

pip install flask

先看一个小例子，目录结构如下：

tree

.

├── hello\_name.py

├── pymysql.py

└── templates

└── index.html

1 directory, 3 files

在虚拟环境下启动python连接数据库的脚本需要前面加sudo。

连接mysql代码：

from flask import Flask, request, render\_template

import MySQLdb

db = MySQLdb.connect("localhost", "root", "123321", "python01")

app = Flask(\_\_name\_\_)

@app.route('/')

def someName():

cursor = db.cursor()

sql = "SELECT \* FROM userinfo"

cursor.execute(sql)

results = cursor.fetchall()

return render\_template('index.html', results=results)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True

html代码，html代码存放在templates目录下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>测试flask</title>

</head>

<body>

<div>

{% for row in results %}

<tr>

<td>{{ row[0] }}</td>

<td>{{ row[1] }}</td>

<td>{{ row[2] }}</td>

<td>{{ row[3] }}</td>

</tr>

{% endfor %}

</div>

</body>

</html

## 1、paramiko模块

基于Python实现的SSH2远程安全连接，支持认证及秘钥方式。

### paramiko安装

命令pip install paramiko

练习代码见GitHub：

https://github.com/wolfrg/Python/blob/master/test/test\_paramiko.py

https://github.com/wolfrg/Python/blob/master/test/paramiko\_keyfile.py

https://github.com/wolfrg/Python/blob/master/test/func\_para.py

Python有个读取配置的模块方法：ConfigParser

## 2、Fabric入门

系统版本：Centos7

Python版本：Python 2.7

pip版本：pip 8.1.2 from /usr/lib/python2.7/site-packages (python 2.7)

## 3、Ansible

### 安装Ansible：

#### 安装环境：

系统环境：Centos7、Python2.7、Pyhton virtualenv

安装方式：

yum install ansible

pip install ansible

Ansible配置：

Ansible任务执行模式：

ad-hoc模式：命令管理模式

playbook模式：Ansible主要管理模式，脚本管理模式

# 实战篇之网站架设

### 一、Django介绍

学Django学什么，Django学习目标：

1.目录结构规范

2.urls路由方式

3.settings配置

4.ORM操作

5.模板渲染

6.其它

官网：

https://www.djangoproject.com/

官网下载地址：

https://www.djangoproject.com/download/

官网教程：

<https://docs.djangoproject.com/en/1.10/intro/tutorial01/>

伯乐在线Django教程：

简约而不简单的Django新手图文教程：

http://python.jobbole.com/87444/

***什么是Django？Django是干什么的？它的原理是什么？***

Django是一个开放源代码的Web应用框架，由Python写成。采用了MVC的软件设计模式，即模型M，视图V和控制器C。

MVC用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

Django的主要目标是使得开发复杂的、数据库驱动的网站变得简单。Django注重组件的重用性和“可插拔性”。

模型（model）：定义数据库相关的内容，一般放在models.py文件中。

视图（view）：定义HTML等静态网页文件相关，html、css、js等前端的东西。

控制器（controller）：定义业务逻辑相关，就是你的主要代码。

### 二Django安装

#### 1.1本地开发环境的安装：

Jdk+eclipse+pydev先前已经安装

需要安装的服务：setuptools Django

命令行方式创建和启动服务：

创建项目：

django-admin.py startproject projectname

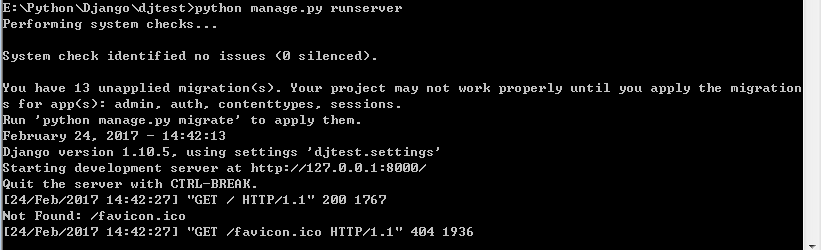
创建APP应用，每个：

django-admin startapp appname

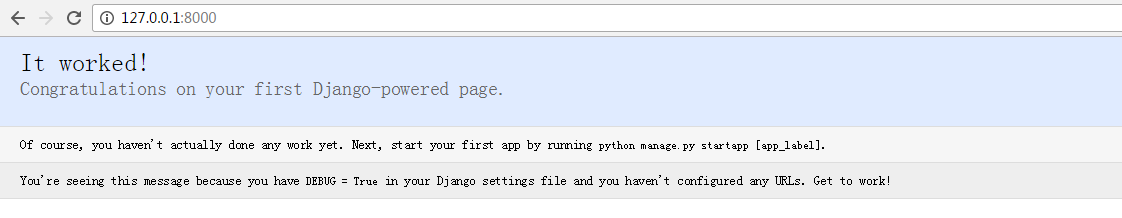
cd 到app目录

启动服务：

python manage.py runserver



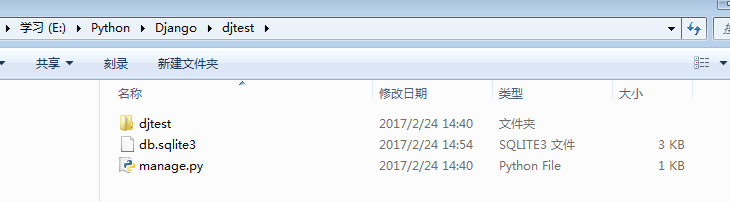
在浏览器输入:127.0.0.1:8000，得到如图所示结果。

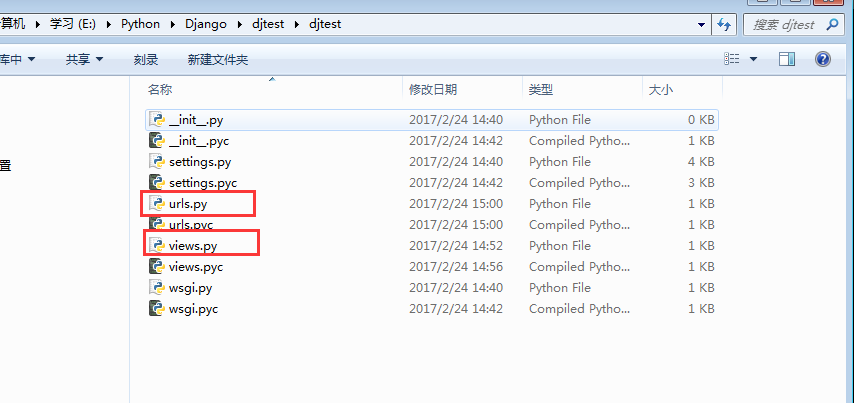


例子：

在页面上显示Hello world!的

项目的文件目录结构如图：





新建V视图views.py文件，写入代码：

#coding:utf8

#这是一个视图文件

from django.http import HttpResponse

def hello(request):

return HttpResponse("Hello world!")

修改urls.py文件：

#coding:utf8

from django.conf.urls import url

from djtest.views import hello

urlpatterns = [

url(r'^hello$', hello),

]

#### 1.2 服务器上的安装：Nginx+Django+uwsgi

系统版本：CentOS 6.5

Python版本：python2.6

安装方式：源码安装

Django版本：Django-1.6.11

uwsgi版本：uwsgi-2.0.15

uwsgi官网下载地址：

https://projects.unbit.it/downloads/uwsgi-2.0.15.tar.gz

参考文档：

http://tchuairen.blog.51cto.com/3848118/1831281 https://www.w3cschool.cn/django/django-nginx-uwsgi.html

#### 1.3 Django模板：

上面列子没有采用Django的MVC思想。

模板是一个文件，用来分离文档的表现形式和内容。html文件内容：

<!--这是模板文件 -->

<!DOCTYPE html>

<html>

<head>

<title>Django MVC</title>

</head>

<body>

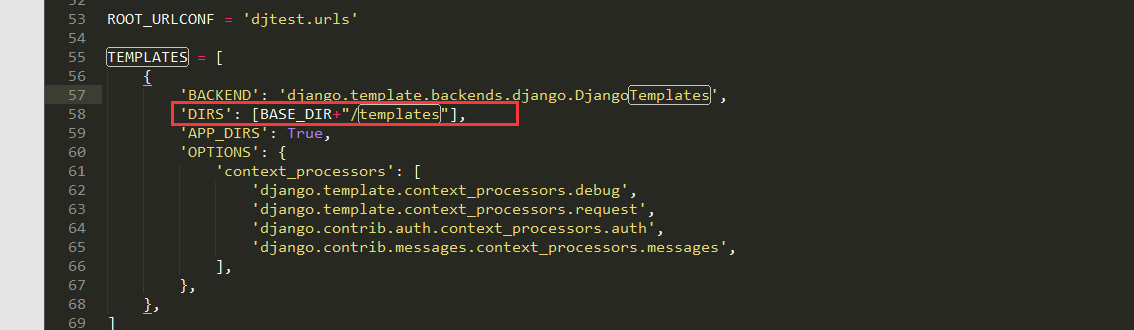
<h1>{{str1}}</h1>

</body>

</html>

模板中变量使用{{}}括起来。

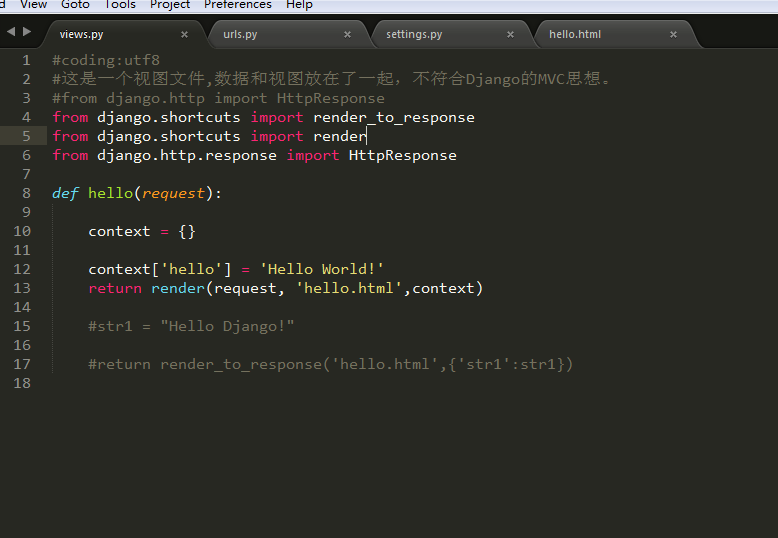
settings.py配置文件:修改TEMPLATES中的DIRS为[BASE\_DIR+”templates”]，你也可以不叫templates，叫别的也行。



来自官方的定义：Templates engines are configured with the **[TEMPLATES](https://docs.djangoproject.com/en/1.10/ref/settings/" \l "std:setting-TEMPLATES)** setting. It’s a list of configurations, one for each engine. The default value is empty. The **settings.py** generated by the **[startproject](https://docs.djangoproject.com/en/1.10/ref/django-admin/" \l "django-admin-startproject)** command defines a more useful value:



views.py:



注意上图中注释部分和未注释部分的区别以及对应模板文件中变量的不同。

#### 1.4 Django模板标签

### Tornado框架

概念：Tornado全称Tornado Web Server，是一个用Python语言写成的Web服务器兼Web应用框架

安装：

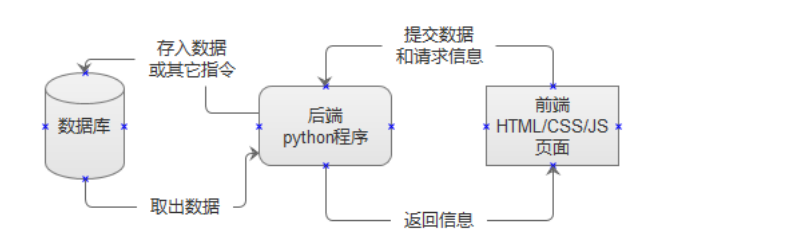
### 网站后台登录验证的知识

jQuery是一套跨浏览器的JavaScript库，简化HTML与JavaScript之间的操作。在HTML文件中引入jQuery的方法有很多种。

第一种：直接从CDN上直接引用：由于防火墙的限制，不可行。

第二种：直接将jQuery下载下来。

### 网站的基本结构



### 连接数据库