# 杂项

* Python注释：

单行：用#号。

多行：放在三个””” “””之间或者’’’ ‘’’之间。

* 目前版本 Python 中的 class 分为 **classical** 和 **new-style** 两大类。其中 classical 是 python 一直沿用的，而 new-style 是 2.2 才开始引入的东西。只要 class 继承于 object，或 bases class 里面任意一个继承于 object，这个 class 都是 new-style。

# 习题 3: 数字和数学计算

* Python打印字符串时，是以逗号为分隔符，来分割字符串和数字计算/变量的：

print "Roosters", 100 - 25 \* 3 % 4, "OK!", 123 / 5.0

结果：Roosters 97 OK! 24.6

* 浮点数的运算，貌似加上小数点就好。

print "Hens1", 25.0 + 31/6

print "Hens2", 25.0 + 31.0/6

结果：

Hens1 30.0

Hens2 30.1666666667

# 习题 5: 更多的变量和打印

* 格式化字符的用法。

my\_name = 'Zed A. Shaw'

my\_age = 35 # not a lie

my\_height = 74 # inches

my\_weight = 180 # lbs

my\_eyes = 'Blue'

my\_teeth = 'White'

print "He's **%d** inches tall." **%** \_height #注意百分号，前面没有逗号。

#逗号仍可拼接。

#此外注意，有多个变量的时候，多个变量放在括号里以逗号隔开。

print "He's got **%s** eyes and **%s** teeth." %(\_eyes, \_teeth), "Good", \_height + \_age

结果

He's 74 inches tall.

He's got Blue eyes and White teeth. Good 109

* Python打字符串的时候可以用单引号。如： print ‘ni hao’

# 习题 6: 字符串(string)和文本

* 格式化字符，%后面不仅可以是变量名，也可以是值。

x = "there are %d types of people." % 10

print x

结果：there are 10 types of people.

* %r 和 %s 有什么不同？  
  %r 用来做 debug 比较好，因为它会显示变量的原始数据（raw data），而其它的符号则是用来向用户显示输出的。

如：

print "I said: %r." % x

print "I also said: %s." % x

结果：

I said: 'there are 10 types of people.'.

I also said: there are 10 types of people..

注意第一个有引号。

* 格式化字符放在变量里，依然可以work。这点挺不习惯：

hilarious = False

joke\_evaluation = "Isn't that joke funny?! % r % s"

temp = joke\_evaluation % (hilarious,'wahaha! ')

print temp

结果：Isn't that joke funny?! False wahaha!

* 类似Java，加号也可以字符串拼接。但是数字不行。

w = "This is the left side of..."

e = "a string with right side."

z = 10

print w + e , z

结果：This is the left side of...a string with right side. 10

* 字符串除了可以双引号括起来，也可以用单引号。

# 习题 7: 更多打印

* \* N 可以打印字符串N次。如：

print "." \* 10

结果：

……….

* 打印字符串的结尾加上逗号，表示在一行继续打印。通常用于一行太长的情况，来分行。只是个比较好的编程习惯。

print "Mary had a little lamb.", #注意逗号

print "Its fleece was white as %s." % 'snow'

结果：

Mary had a little lamb. Its fleece was white as snow.

# 习题 8: 打印，打印

* 我应该使用 *%s* 还是 *%r*？

你应该使用 %s，只有在想要获取某些东西的 debug信息时才能用到 %r。 %r 给你的是变量的“程序员原始版本”，又被称作“representation”。

* formatter = "%r %r %r %r"

print formatter % (formatter, formatter, formatter, formatter)

print formatter % (

"I had this thing.",

"That you could type up right.",

"But it didn't sing.",

"So I said goodnight."

)

结果：

'%r %r %r %r' '%r %r %r %r' '%r %r %r %r' '%r %r %r %r'

'I had this thing.' 'That you could type up right.' "But it didn't sing." 'So I

said goodnight.'

注意最后的一行结果里，既有单引号，也有双引号。因为有双引号的String里包括了一个单引号。书中解释：

*为什么 %r 有时打印出来的是单引号，而我实际用的是双引号？*

*Python 会用最有效的方式打印出字符串，而不是完全按照你写的方式来打印。这样做对于 %r 来说是可以接受的，因为它是用作 debug 和排错，没必要非打印出多好看的格式。*

# 习题 10: 那是什么?

* 本章讲了字符转意符。

字符转移符和格式化字符%r：  
*记得 %r 格式化字符串吗？使用 %r 搭配单引号和双引号转义字符打印一些字符串出来，将 %r和 %s 比较一下。注意到了吗？****%r 打印出来的是你作为程序员写在脚本里的东西，而 %s 打印的是你作为用户应该看到的东西****。例子*：（不是很明白，双引号在%r为什么没有看到反斜杠。）

s1 = "It's \"Sparta!\""

s2 = 'It\'s "Sparta!"';

print "He said: %s %s" % (s1, s2)

print "He said: %r %r" % (s1, s2)

结果：

He said: It's "Sparta!" It's "Sparta!"

He said: 'It\'s "Sparta!"' 'It\'s "Sparta!"'

* “三引号(triple-quotes)”，也就是 """，你可以在一组三引号之间放入任意多行的文字。例子：

fat\_cat = """

I'll do a list:

\t\* Cat food

\t\* Fishies

\t\* Catnip\n\t\* Grass

"""

print fat\_cat

结果：

I'll do a list:

\* Cat food

\* Fishies

\* Catnip

\* Grass

用单引号代替双引号，效果也一样。

# 习题 10：提问

* 这节介绍了控制台输入函数raw\_input()。
  + x = int(raw\_input())，可以把用户输入的字符串转成整数。
  + x = float(raw\_input()) ，浮点数
* 换行输入的问题

print "So, you are %r old, %r tall and %r heavy" % (

age, height, weight) //可以

但如果在’(‘前面换行，不管在百分号前还是后，都会报错，执行不了。

# 习题 12: 提示别人

* raw\_input的参数问题。  
  age = raw\_input("How old are you? ")

等同于

print “How old are you?”

age = raw\_input()

* python帮助文档

pydoc [类名?] 如 pydoc raw\_input

Window下要输入 python –m pydoc raw\_input

# 习题 13: 参数、解包、变量

* import包的时候，python的术语是”Module，模组”。也可以叫库。
* 和Java一样，命令行中输入的参数是字符串。想变成数字要用int(xxx)来转。
* argv 和 raw\_input() 有什么不同？

不同点在于用户输入的时机。如果参数是在用户执行命令时就要输入，那就是 argv，如果是在脚本运行过程中需要用户输入，那就使用 raw\_input()。

思考：Java中有类似raw\_input()的功能吗？

* 例子：

|  |
| --- |
| #Author Zhu Xiuwei  #Date 2014-3-19  from sys import argv  print "This function is to perform add operation."  firstNum = int(raw\_input("The first number: "))  secondNum = int(raw\_input("The first number: "))  **ss, thirdNum = argv //貌似第一个参数永远是脚本的名字。**  print "The 3rd number is", thirdNum  print "The script name is: " , ss  print "The result is: %d" % (firstNum + secondNum + int(thirdNum))  结果：  注意红色框中的字是用户输入的字。 |
|  |

# 习题 14: 提示和传递

就是上一个习题的一个加强版。挺有意思的，有点“人机交互”的感觉。

|  |
| --- |
| from sys import argv  **script, user\_name = argv**  prompt = '> '  print "Hi %s, I'm the s% script." %(user\_name, script)  print "I'd like to ask you some questions."  print "%s, do you like me?" % user\_name  likes = raw\_input(prompt)  print "Where do you live?"  lives = raw\_input(prompt)  print "What kind of computer do you have?"  computer = raw\_input(prompt)  print"""  Well, so you said %r about like me.  You live in %r. Not sure where that it.  And you have a %r computer. Nice.  """ % (likes, lives, computer) //注意，格式化变量要写在三个双引号的外边。 |
| 结果： |

# 习题 15: 读取文件

* 本章介绍怎么open一个文件，以及用简单地读取里面的内容。现在看来，Python处理文件比Java简单一些。

示例：

txt = open(filename) //打开文件

print txt.read() //读取内容。

txt.close() //关闭文件

其中，Open方法返回的是一个file对象。

close – 关闭文件。跟你编辑器的 文件->保存.. 一个意思。

read – 读取文件内容。你可以把结果赋给一个变量。

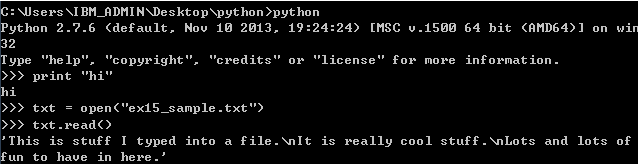
readline – 读取文本文件中的一行。

truncate – 清空文件，请小心使用该命令。

write(stuff) – 将stuff 写入文件

* Python不限制打开同一个文件的次数。
* 可以在Python环境下运行Python命令：

*首先，在命令行输入 python 然后敲回车。现在你就在 python 环境中了。接下来你就可以输入并运行一句一句的代码。试着玩玩，如果想退出就输入 quit()（或者ctrl+z） 再敲回车。*

**

# 习题 16: 读写文件

* 直接调用write函数去写文件。想调用write函数，打开文件时要用’w’模式。

target = open(filename, 'w')

target.write(line1)

target.truncate() --- 在读模式的时候不能用。会报错。  
 另外，’w’模式时会自动清空原来的文本，而不是追加。

* 除了’**w**’模式，还有’**r**’模式（默认的，只读）和’**a**’模式（append，追加写。）

最重要的是 **+ 修饰符**，写法就是 'w+', 'r+', 'a+' ——这样的话文件将以同时读写的方式打开，而对于文件位置的使用也有些不同。

* Python写文件操作，如果目标文件不存在，会自动创建文件。

# 习题 17: 更多文件操作

* 判断文件是否存在，用到的一个模组：

from os.path import exists  
exists(file\_name) //返回True or False。注意牛逼之处，传入的参数是文件名就行，不像Java必须先定义一个File对象。

* 获取字符串长度: len(str)

# 习题18：命名、变量、代码、函数

* 传递多个参数：

def print\_two**(\*args)**: //注意后面有冒号

arg1, arg2 = args //多个参数的，也需要解包。

print "arg1: %r, arg2: %r" % (arg1, arg2)

print\_two（”a”,”b”） //注意，只能传2个参数。因为函数体在unpack的时候是两个参数，传多、传少都不行。

* 感觉python**不支持函数重载。**

def a():

print "1"

def a(ss):

print ss

a()

a("1")

结果：

C:\Users\IBM\_ADMIN\Desktop\python>python ex18.py

Traceback (most recent call last):

File "ex18.py", line 5, in <module>

a()

TypeError: a() takes exactly 1 argument (0 given)

# 习题 20: 函数和文件

* 文件的一些新注意：

读文件的时候，如果想“倒带”读已经读过的内容，需要用到seek()函数。

例子：

|  |
| --- |
| ex16\_.txt内容：  111111111111  222222222222  333333333333333333333333333333333333333333333333333333 |

from sys import argv

script, file\_name = argv

file = open(file\_name)

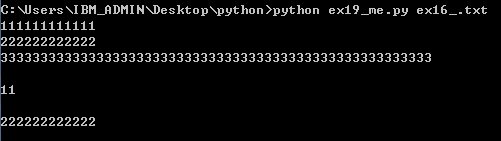
print file.read()

**file.seek(10)** //让“游标”回到第10+1，即第11个字符（下标从0开始）

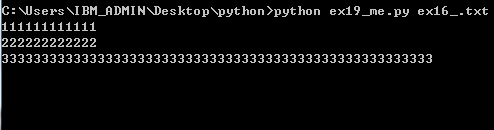
print file.readline()

print file.readline()

运行：



如果把**file.seek(10)**去掉，结果：



readline()就读不出内容了。

* readline() read()默认会在打印一行后面加入一个’\n’，不希望换行的话，可以在后面加个逗号：

**file.readline(),**

# 习题21：函数可以返回东西

def addF(a,b):

result = a + b

print "%d + %d = %d" % (a, b, result)

**return result**

result = addF(1,2)

print "return value: ", result

# 习题 24: 更多练习

**函数返回值可以有多个！！！**

def secret\_formula(started):

jelly\_beans = started \* 500

jars = jelly\_beans / 1000

crates = jars / 100

**return jelly\_beans, jars, crates**

start\_point = 10000

**beans, jars, crates** = secret\_formula(start\_point)

# 习题25：更多更多的练习

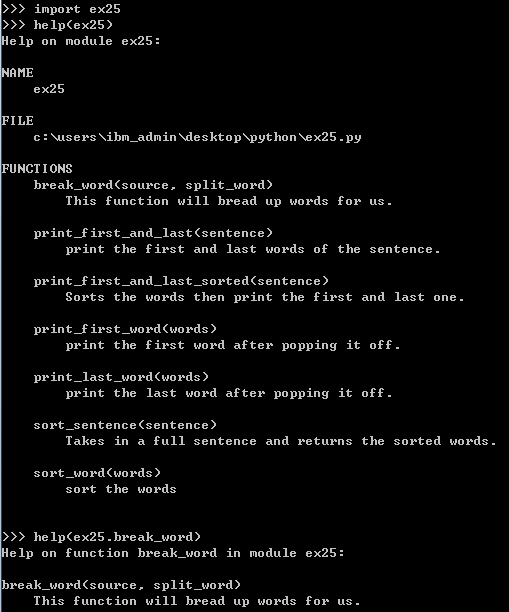
**这章比较有价值。介绍了一些有用的特性。**

* 关于文档注解（documentation comments）

在定义函数中，放在””” “””之间的东西。（貌似放在函数第一行才行？？？）

然后在python控制台中可以调用help()开查看这些注解。

**注意在调用help前，要先import这个模组**。如：



* 关于import命令

两种使用方式：

1. **import** ex25

…

**ex25**.methond1()

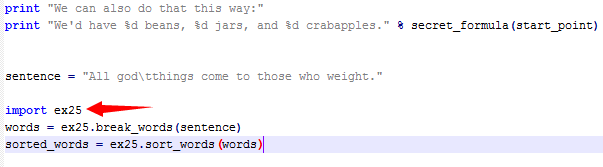
1. **from** ex25 **import** method1 //（或者from ex25 import \*）

…

method1() //前面就不用再写ex25了

# [习题 26: 恭喜你，现在可以考试了！](http://www.2cto.com/shouce/Pythonbbf/ex26.html)

* Import可以出现在代码的中间，而不是像Java那样必须出现在最前面。



# 习题 28: 布尔表达式练习

* Python布尔的规则和Java基本一样。一点区别：

|  |  |
| --- | --- |
| **Java** | **Python** |
| && | and |
| || | or |

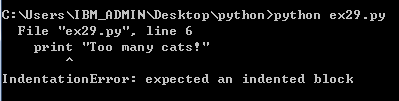
* 有没有**短路逻辑**？  
  有的。任何以 False 开头的 and 语句都会直接被处理成 False 并且不会继续检查后面语句了。任何包含 True 的 or 语句，只要处理到 True 这个字样，就不会继续向下推算，而是直接返回 True 了。不过还是要确保整个语句都能正常处理，以方便日后理解和使用代码。

# [习题 29: 如果(if)](http://www.2cto.com/shouce/Pythonbbf/ex29.html)

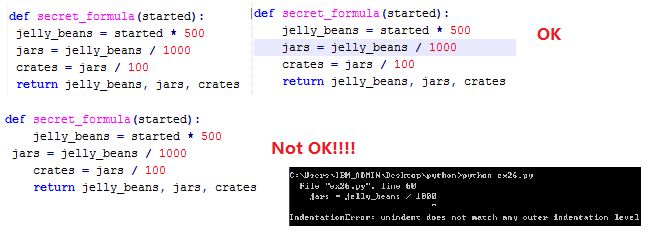
* 类似于函数，python中if后面也必须有一个冒号’:’，然后其内的语句也要有缩进。
  + 为什么 if 语句的下一行需要 4 个空格的缩进？**行尾的冒号的作用是告诉 Python 接下来你要创建一个新的代码区段**。这根你创建函数时的冒号是一个道理。
  + 如果不缩进, 会发生什么事情? 如果你没有缩进，你应该会看到 Python 报错。**Python 的规则里，只要一行以“冒号(colon)” : 结尾，它接下来的内容就应该有缩进**。

不缩进的结果：

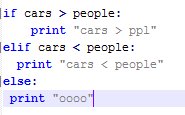




* + 还有一个有意思的事情，缩进的话，缩进几个空格没有强制要求，但是同一“级别”的语句要保证有相同的缩进。比如都是一个空格或一个tab符。但有的1个空格有的2个空格就不行。如：



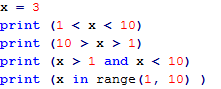
# 习题 30: Else 和 If

* Python中是**elif**关键字和**else**关键字。后面也是冒号和缩进。
* 如果有else、elif，其各子区块**不需要保持一样的缩进**。如：
* 如果多个 elif 区块都是 True 是 python 会如何处理？

Python 只会运行它碰到的是 True 的第一个区块，所以只有第一个为 True 的区块会被运行。（应该和java一样）

但如果有多个if else，那每个if都要判断一次。（还是和java一样）

* 怎样判断一个数字处于某个值域中？

|  |
| --- |
| 注意**range**函数。他是含首不含尾的。即，range(1,10)是从1到9。  **range**函数**返回的是一个integer的list**。 |

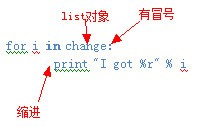
# 习题 32: 循环和列表

* Python中的list(列表)，在python中列表和数组是一个东西，统称列表。

change = [1, 'pennies', 2, 'dimes', 3, 'quarters']

其中的内容可以是混合的，如上，既有字符串也有数字。

* for in 循环：注意貌似一般都是针对一个list来做for。（例子中change就是个list）



类似Java的

for(String s: change)

System.out.print(s);

* for 循环中用**range函数**：(因为for一般都是在list中做，range函数返回的是个list，故当然可以对range函数返回的list做循环)

elements = []

for i **in** **range(0, 6, 2):**

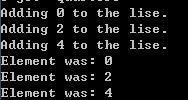
print "Adding %d to the list." % i

elements.**append(i)** #注意list的append函数。

for i in elements:

print "Element was: %d" % i

结果:



注意range的用法：

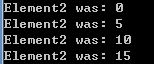
1. 是含首不含尾的。第一个参数是其实数字（包括），第二个参数是结束数字（不包括）。
2. 注意第三个参数是step，可选。

* 可以直接用range函数给list赋值：（因为range函数返回的是个list）

elements2 = **range(0, 20, 5)**

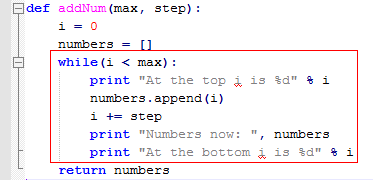
for i in elements2:

print "Element2 was: %d" % i

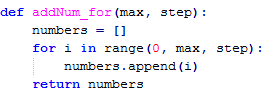


# 习题 33: While 循环

* While循环没有什么特别好说的，和java的差不多。也是注意冒号、缩进。



上面同样功能函数的for-loop版本：

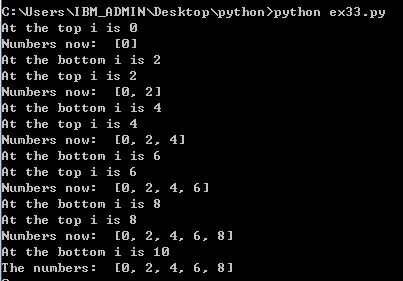


作者比较推崇用for，少用while。

* 可以直接打印list:



结果：



# 习题 34: 访问列表的元素

* 可以像访问java数组那样访问python list里的元素。如：

animals = ['bear', 'tiger', 'penguin', 'zebra']

bear = animals[0]

* 讲了一堆关于序数(ordinal number)和基数(cardinal number)的东西，感觉没啥用。

# 习题 35: 分支和函数

* **判断字符串中包含某字符**的方式：



* **exit(0)**有什么功能？

在很多类型的操作系统里，``exit(0)`` 可以中断某个程序，而其中的数字参数则用来表示程序是否是碰到错误而中断。 exit(1) 表示发生了错误，而 exit(0) 则表示程序是正常退出的。这和我们学的布尔逻辑 0==False 正好相反，不过你可以用不一样的数字表示不同的错误结果。比如你可以用 exit(100) 来表示另一种和 *exit(2)`* 或 exit(1) 不同的错误。

# 习题 37: 复习各种符号

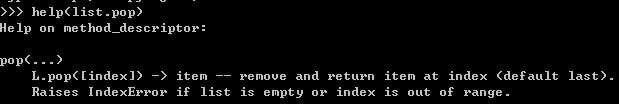
关键字

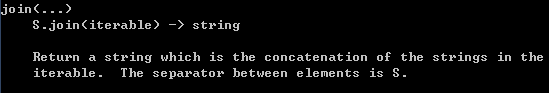
|  |  |
| --- | --- |
| 关键字 | 说明 |
| **del** | del对**list列表操作，用来删除一个或连续几个元素**。    [0:2]表示删除0-1，含首不含尾。 |
| **with…as** | 比较复杂，<http://python.42qu.com/11155501>  Python’s with statement provides a very convenient way of dealing with the situation where you have to do a setup and teardown to make something happen. A very good example for this is the situation where you want to gain a handler to a file, read data from the file and the close the file handler.  file = open("/tmp/foo.txt")  try:      data = file.read()  finally:      file.close()  等价于  with open("/tmp/foo.txt") as file:      data = file.read() |
| **global** | 定义全局变量 |
| **pass** | 1、空语句 do nothing  2、保证格式完整  3、保证语义完整  java中写法：  if(true)  ; // do nothing  else  {} // do nothing  python中写法：  if true:  **pass** # do nothing  else:  print "do something." |
| **yield** | 比较复杂，没看 |
| **try except finally** | 类似Java try-catch-finally |
| **raise** | 用来抛出异常，类似java throw，具体用法细节没看。 |
| **Break / continue** | Break: 用来终止循环语句  Continue: 应该和Java类似 |
| **exec** | exec语句用来执行储存在字符串或者文件中的python语句。可以生成一个包含python代码的字符串，然后使用exec语句执行这些语句。  >>>**exec** 'print "hello word"'  hello world |
| **lambda** | lambda函数也叫匿名函数，即，函数没有具体的名称。先来看一个最简单例子：  def f(x): return x\*\*2  print f(4)  Python中使用lambda的话，写成这样  g = lambda x : x\*\*2  print g(4) |

操作符

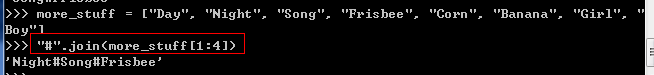
|  |  |
| --- | --- |
| **\* 和 \*\* 传参** | 当函数的参数不确定时，可以使用\*args 和\*\*kwargs，\*args 没有key值（感觉类似Java的…传参），\*\*kwargs有key值，涉及到dictionary 的概念。  <http://blog.csdn.net/chenjinyu_tang/article/details/8136841> |
| **\*\*=** | 乘方。 如2\*\*3=8 |

# 习题38列表的操作

* list的pop()方法，可以不带参数，默认返回list最后一个index的元素。  
  
* str.**join**(iterable)方法：



例子：



注意[1:4]的用法，和range(1,4)类似

# 习题 39: 字典, 可爱的字典

* dictionary/dict/字典，感觉等同于Java的HashMap。
* cities = {

'jl': 'Bai Shan',

'bj': 'Hai Dian'

}

# add some more cities

cities['sx'] = 'Yan An'

print cities.get('sss') #result: **None**

print cities.get('sx') #result: Yan An

#print cities['sss'] #result: KeyError: 'sss' (异常)

cities = {

'jl': 'Bai Shan',

'bj': 'Hai Dian'

}

for **abbrev, city** in **cities.items():**

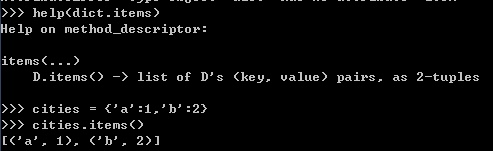
print "%s has city %s" % (abbrev, city)

结果:

jl has city Bai Shan

bj has city Hai Dian

dict.items()的用法：



* 和Java的HashMap一样，dict是无序的。

**What if I need a dictionary, but I need it to be in order?**

Take a look at the collections.OrderedDict data structure in Python. Search for it online to find the documentation.

* 其**get()**方法，可以这么用：

D.get(‘key’, obj): 如果D里有key，返回key对应的value。否则返回obj。obj默认是None。

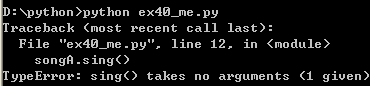
# 习题 40: Modules, Classes, and Objects

* 主要介绍了Module、Class、Object。Moduel可以理解成一个.py文件。
* 一个例子：



1. *\_\_init\_\_*方法感觉类似Java的构造函数。并且是可以由多个参数的！
2. *Self*类似Java的this。当class实例化后，Self代表这个object。
3. 函数声明里的self参数不能去掉。否则会出现参数个数不对的error。具体原因就是a.func(arg)，实际上python看做是A.sfunc(a, arg)。

因此，songA.sing()实际上是调用Song.sing(songA) -> 其中的参数songA即是self。如果sing方法定义的时候没有self，那么就会报如下错误：



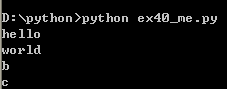
1. 注意self.a，挺有意思的。这个’a’是个class实例的属性，不是本地变量。

貌似在class里面可以这样不声明变量就是用一个属性。

相比之下，第1行的lyrics就是个global变量。

第6行的’b’是个local变量。在第9行如果写成for line in **b**，会出错，错误信息表示找不到全局变量’b’。

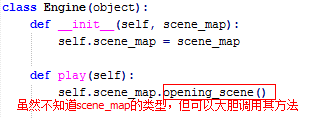
1. 如果把第3行括号里的object去掉，不影响结果。

执行结果：  


# 习题 43: Basic Object Oriented Analysis And Design

* 自己动手做了一下，见ex43\_me.py。一些值得注意的地方：
  + 当一个class包含另一个class的引用时(has-A)，因为Python不需要声明一个对象的类型是什么，怎么调用这个被引用类型的方法让我疑惑。

其实，初始化后，是可以直接大胆调用其方法就好了。

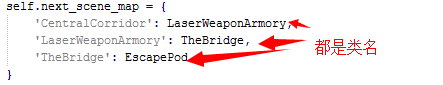


* + 根据一个类名，直接加括号()就可以实例化其对象，很方便。

nextScene = self.next\_scene\_map.get(scene\_name)**()**

是不是有点Java反射的赶脚？

其中，()前面的语句是从一个Map里获取一个Type。

****

* + 获取一个object的Class的名字：

objClassName = <the obj>**.\_\_class\_\_.\_\_name\_\_**

# 习题 44: Inheritance Vs. Composition

* 作者的一个忠告：
  + Most of the uses of inheritance can be simplified or replaced with composition
  + multiple-inheritance should be avoided at all costs.
* Python从module 1的Class A调用Module 2的Class B，必须在module1里import module2的class B，如：

|  |  |
| --- | --- |
| a.py  --------------- | b.py  --------------- |
| 结果：    注意，如果去掉a.py里的,结果会变成： | |

* 本章的练习是阅读<https://www.python.org/dev/peps/pep-0008/> pep0008，我还没看。

# 习题 46: A Project Skeleton

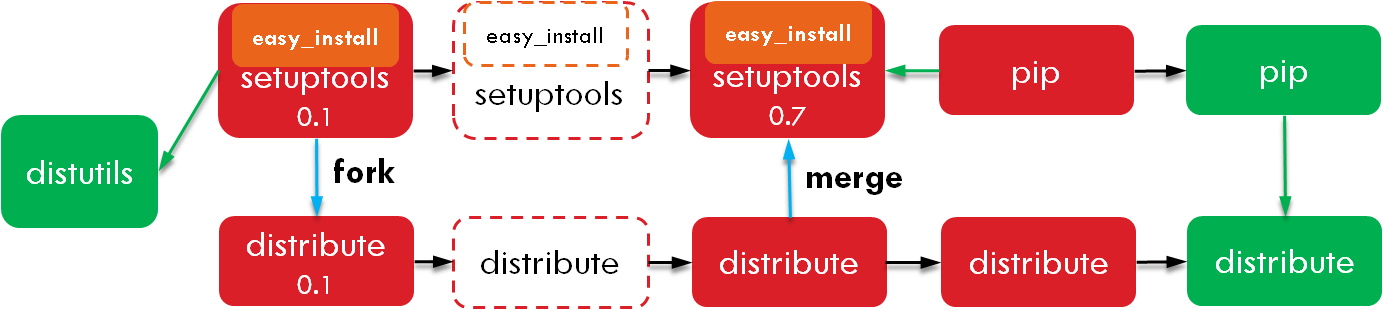
* 首先是**Python打包**的问题，需要靠接一些术语和历史。

参考文献：

[Python打包的艺术（一）- 综述](http://blog.chinaunix.net/uid-15174104-id-3863249.html)

[Python打包的艺术（二）- Packaging Tools的昨天，今天，明天](http://blog.chinaunix.net/uid-15174104-id-4036539.html)

**历史简介**：

* + 随着python刚发展，需要有个集中的地方来共享和下载第三方模块和包的地方，就有了，[**[Python Package Index]**](https://pypi.python.org/pypi/)(也叫**PyPI**)，全球开发者都可以在此共享自己的软件包。
  + 最早的发布Python模块和包的库是distutils, 它现在已是Python的标准库。如果你想详细了解**distutils**，请参看Python官方文档里由Greg Ward撰写的[[Distributing Python Modules]](http://docs.python.org/2/distutils/)。
  + **setuptool**是基于distutils的，可以说是distutils库的加强和扩展，是其替代品。它可以让开发者更容易的构建和发布Python软件包，并添加了包的版本化和依赖性管理。它可以将你开发的python模组打包成一个后缀为.egg的文件，用于发布和安装。同时它兼容distutils构建的包。提到setuptools，我们不得不提**easy\_install**，[**Distribute**](https://bitbucket.org/tarek/distribute/wiki/Home)和**pip。**
  + **easy\_install**是绑定在setuptool中的一个模块，让开发者可以自动下载、构建、安装和管理python软件包。完全可以把它看做调用setuptool的一个命令，有了这个命令我们可以自如使用setuptool的功能。
  + **distribute**则是从setuptools分支（fork）出来的一个项目，目的是为了替换setuptools作为Python模块发布的标准方法。不过要注意的是Distribute是setuptools的drop-in（嵌入式）替代品。也就是说Distribute到某个时候会merge回到setuptools里。在setuptools 0.7, Distribute就被merge进入到了setuptools，现在setuptools最新的版本已是1.0。\
  + **pip**可以被认为是easy\_install的超级加强版。它不再绑定到setuptools，是个独立的项目，但是它也依然依赖于setuptools。
  + 下面的图诠释了它们过去，现在和未来的关系。最终，distribute和pip都应该会成为Python的标准库，从而成为Python打包标准。下面的图是来自<http://python-distribute.org/> 的一个announcement。**最终，distribute将会取代setuptools，而pip会取代easy\_install。我相信distribute和pip最终应该能成为标准的Python库。**
* **如何安装distribute:**

参考<https://pypi.python.org/pypi/distribute/0.6.49>

有三种方法。其中第一种是推荐方法：

1. 直接下载一个python文件*distribute\_setup.py*，然后执行这个文件即可：

*$ curl -O* <http://python-distribute.org/distribute_setup.py>

*$ python distribute\_setup.py*

1. 使用pip或者easy install:

*$ pip install distribute*

1. 下载source tarball，解压，安装：

*$ curl -O* [*http://pypi.python.org/packages/source/d/distribute/distribute-0.6.49.tar.gz*](http://pypi.python.org/packages/source/d/distribute/distribute-0.6.49.tar.gz)

*$ tar -xzvf distribute-0.6.49.tar.gz*

*$ cd distribute-0.6.49*

*$ python setup.py install*

* 这一章用到的包，我都是按照第三种方法安装的。在用pip安装distribute的时候，报了一个和好像和编码相关的错误，没细看。此外，装distribute的时候，报这么个错误：（方法1和方法3都是这样）

*error: C:\Python27\Scripts\easy\_install-2.7.exe: Permission denied*

不过暂时看不出来有什么影响。

晚上搜了下原因，大致是因为“On windows, the easy\_install binary is in-use during install so it can't overwrite itself.” (Easy\_install itself is part of the setuptools you're trying to install with itself ;)

* 自己尝试写了个工程，安装了。见：D:\python\packages\xwtest

里面只有一个module：D:\python\packages\xwtest\XwTest\my1stTest.py，只做一件事打印hello world。



卸载的时候有问题：

*ValueError: ('Expected version spec in', 'XwTest\*', 'at', '\*')*

这章讲的很不清楚，但已经浪费不少时间了，可能需要找别的正常点的教材再补补。

# 习题 47: AutomatedTesting

* 测试的代码是放到package的tests目录下面。

貌似nosetests会把这个目录下所有的\*\_tests.py module运行一遍。

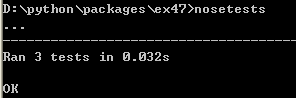
运行时执行这些py文件里的每个test\_\*()方法。

* 本例中，用到了nosetests的assert\_equal()方法来测试验证点，如：

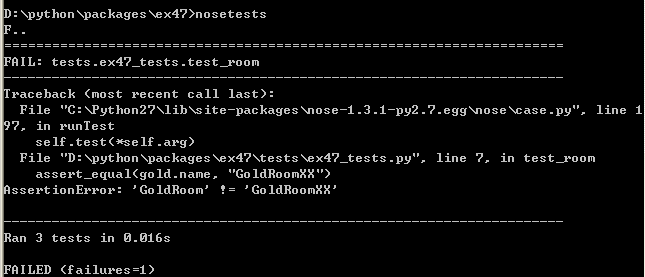
***assert\_equal(gold.name,*** *"GoldRoom")*

注意，assert\_equal()方法也适用于对象的比较。

如果测试通过的话，

**

否则：



# 习题 48: Advanced User Input

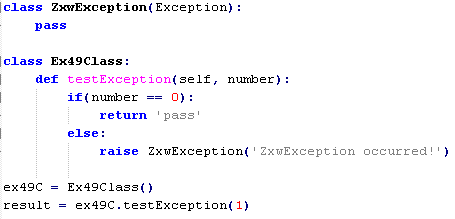
过了一遍，习题没做。里面用到了一个**tuple**（元组）的概念：

Tuple 是不可变 list。 Tuple用小括号表示元素集合而不是List的中括号。

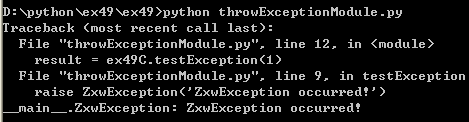
# 习题 49: Making Sentences

* 首先讲了**Raise Exception**的用法:

#定义一个Exception类，让它继承Exception即可：

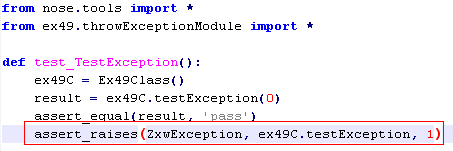


结果：

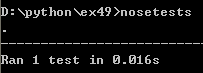


* 然后，本章侧重在写test类上：用**assert\_raises**来写一个期望结果是fail的test，这也是测试的很重要的一部分。我做的例子：

用法：***assert\_raises(exception, callable, parameters)***



结果：



上面是符合预期的结果。但是如果把最后一行的参数1改成参数0（这样ex49c.testException方法就不抛出异常），执行结果为：

