**0、命令行常用命令**

命令行（Windows下叫“命令提示符”，Mac下叫“终端”）里的常用命令。打开命令行，我们会看到每行前面都有诸如

C:\Documents and Settings\Crossin>

或者

MyMacBook:~ crossin$

之类的。

这个提示符表示了当前命令行所在目录。

第一个常用的命令是：

**dir （windows环境下）**

**ls （mac环境下）**

dir和ls的作用差不多，都是显示出当前目录下的文件和文件夹。

第二个常用命令是：

**cd 目录名**

通过dir或ls了解当前目录的结构之后，可以通过“cd 目录名”的方式，进入到当前目录下的子目录里。

如果要跳回到上级目录，可以用命令：

**cd ..**

另外，Windows下如果要写换盘符，需要输入

**盘符:**

比如从c盘切换到d盘 C:\Documents and Settings\Crossin>d:

有了以上两个命令，就可以在文件目录的迷宫里游荡了。虽然没可视化的目录下的操作那么直观，但是会显得你更像个程序员。。。

于是乎，再说个高阶玩法：现在你可以不用idle那套东西了，随便找个顺手的文本软件，把你的代码写好，保存好，最好是保存成py文件。然后在命令行下进入到py文件保存的目录，使用命令：**python 程序保存的文件名**

就可以运行你写的程序了。

**一、关于分号“；”**

在C、Java等语言的语法中规定，必须以分号作为语句结束的标识。Python也支持分号，同样用于一条语句的结束标识。但在Python中分号的作用已经不像C、Java中那么重要了，Python中的分号可以省略，主要通过换行来识别语句的结束。

例如，以下两行代码是等价的：

1. print "hello world!"
2. print "hello world!";

第1行代码的输出结果：hello world!

第2行代码的输出结果：hello world!

如果要在一行中书写多条句，就必须使用分号分隔每个语句，否则Python无法识别语句之间的间隔：

1. # 使用分号分隔语句
2. x=1; y=1 ; z=1

第2行代码有3条赋值语句，语句之间需要用分号隔开。如果不隔开语句，Python解释器将不能正确解释，提示语法错误：SyntaxError: invalid syntax

注意分号不是Python推荐使用的符号，Python倾向于使用换行符作为每条语句的分隔，简单直白是Python语法的特点。通常一行只写一条语句，这样便于阅读和理解程序。一行写多条语句的方式是不好的习惯。

**二、关于连行符 “ \” (双反斜杠好像不可以，已测试)**

Python同样支持多行写一条语句，Python使用“\”作为连行符。在实践中，一条语句写在多行也是非常常见的。

【例】把SQL语句作为参数传递给函数，由于SQL的语句一般非常长，为了阅读方便，因此需要换行书写。

1. # 字符串的换行
2. # 写法一
3. sql = "select id,name \
4. from dept \
5. where name = 'A'"
6. print sql
7. # 写法二
8. sql = "select id,name " \
9. "from dept " \
10. "where name = 'A'"
11. print sql

写法一只使用了一对双引号，把SQL语句分为select、from、where等3部分分别书写。

第6行代码输出结果：select id,name from dept where name = 'A'

写法二使用了3对双引号，select、from、where分别对应一对双引号。

第11行代码输出结果：select id,name from dept where name = 'A'

第二种写法比第一种写法的可读性更强，可以使用空格和制表符对齐语句，使代码显得更工整。对于简短的语句不推荐换行的写法，这种写法只会造成阅读的复杂性。下面这段程序是不合理的换行写法：

1. # 一条语句写在多行
2. print \
3. "hello world!"

第2行～第3行代码是一个整体，调用print输出“hello world!”，这种情况不适合分行书写。

**三、关于单引号和双引号**

双引号和单引号都是可以用的，只不过用单引号的时候可以在字符串里面使用双引号。同理，使用双引号时也可以在字符串里面使用单引号

先说1个双引号与3个双引号的区别，双引号所表示的字符串通常要写成一行
如：
s1 = "hello,world"
如果要写成多行，那么就要使用\(“连行符”)，如
s2 = "hello,\
world"
s2与s1是一样的。如果你用3个双引号的话，就可以直接写了，如下：
s3 = """hello,
world,
hahaha."""，那么s3实际上就是"hello,\n world,\n hahaha.", 注意“\n”，所以，
如果你的字符串里\n很多，你又不想在字符串中用\n的话，那么就可以使用3个双
引号。而且使用3个双引号还可以在字符串中增加注释，如下：
s3 = """hello,  #hoho, this is hello, 在3个双引号的字符串内可以有注释哦
world,          #hoho, this is world
hahaha."""

这里我试过一下，确实可以加注释，不过在print s3的时候连注释内容会一起给打印出来
这就是3个双引号和1个双引号表示字符串的区别了，3个双引号与1个单引号的区别也
是和这个一样的，实际上python支持单引号是有原因的，下面我来比较1个单引号和
1个双引号的区别。

当我用单引号来表示一个字符串时，如果要表示 Let's go 这个字符串，必须这样：
s4 = 'Let\'s go'，注意没有，字符串中有一个'，而字符串又是用'来表示，所以 这个时候就要使用转义符 \, 如果你的字符串中有一大堆的转义符，看起来肯定不舒服，python也很好的解决了这个问题，如下：
s5 = "Let's go"
对于双引号，也是一样的，下面举个例子
s6 = 'I realy like "python"!'
这就是单引号和双引号都可以表示字符串的原因了。

例子：

1. print ("Stay hungry,\n" #这里换行符\n必须写在双引号里面

 "stay foolish.\n"

 "-- Steve Jobs")

1. print ('''Stay hungry, #这里用了三个单引号，没用换行符

stay foolish.

-- Steve Jobs''')

1. print ("""Stay hungry, #这里用了三个双引号，没用换行符

stay foolish.

-- Steve Jobs""")

这三个输出的结果都是三行：

Stay hungry,

stay foolish.

-- Steve Jobs

1. print ("Stay hungry, \

stay foolish. \

-- Steve Jobs")

这个输出的结果是一行：Stay hungry, stay foolish. -- Steve Jobs

**四、关于注释符**

用井号作为注释一行
#此处为注释

一般用"""（三个英文引号）也可以注释多行，不过一般用来表示函数文档
"""
此处是注释
此函数功能为。。。。
"""

**五、逻辑量True和 False 首字母必须大写**

[**Python中的None**](http://blog.csdn.net/liuxincumt/article/details/7988351)**：**

None是一个特殊的常量。

None和False不同。

None不是0。

None不是空字符串。

None和任何其他的数据类型比较永远返回False。

None有自己的数据类型NoneType。

你可以将None复制给任何变量，但是你不能创建其他NoneType对象。

**六、if语句，if…elif…else语句，while循环，for循环, break和continue**

if 条件： #注意冒号必须有

选择执行的语句 #注意这里有四个空格的缩进

if, elif, else可组成一个整体的条件语句：

if是必须有的；elif可以没有，也可以有很多个，每个elif条件不满足时会进入下一个elif判断；else可以没有，如果有的话只能有一个，必须在条件语句的最后。

if a == 1:

   print 'one'

elif a == 2:

   print 'two'

else:

   print 'too many'

while 条件： #注意冒号必须有

 选择执行的语句 #注意这里有四个空格的缩进

for i in range(1, 101): #i从1取到100
　 print i

注：range(x, y, c)表示一个list：[x, x+c, x+2c, … , y-c]。如果没有第三个参数c, 默认c=1.

**break**

# while 循环在条件不满足时 结束，for 循环遍历完序列后 结束。如果在循环条件仍然满足或序列没有遍历完的时候，想要强行跳出循环，就需要用到break语句。

# continue

# break是彻底地跳出循环，而continue只是略过本次循环的余下内容，直接进入下一次循环。

**七、 变量命名规则**

python在定义一个变量时**不需要给它限定类型**。变量会根据赋给它的值，自动决定它的类型。你也可以在程序中，改变它的值，于是也就改变了它的类型。
**变量命名时**第一个字符必须是字母或者下划线“\_”， 剩下的部分可以是字母、下划线“\_”或数字（0-9）
变量名称是对大小写敏感的，myname和myName不是同一个变量。

**八、字符串的输出和运算**

字符串之间可以通过加法连接起来：

str1 = 'good'

str2 = 'bye'

print (str1 + str2)

print ('very ' + str1)

print (str1 + ' and ' + str2)

print ('My age is ' + str(18))

num = 18

print ('My age is %d' %num) #注意，%num可以直接用%18代替，但是%num和字符串之间不能加逗号，只能是空格（没有空格也可以）

print ('Today is %s.' % 'Friday')

print (‘Price is %.2f’ % 4.99)

注意：%d表示整数，%s表示字符串，%f表示浮点数，%.4f表示带4位小数的浮点数

print （"%s's score is %d" % ('Mike', 87)）

无论你有多少个值需要代入字符串中进行格式化，只需要在字符串中的合适位置用对应格式的%表示，然后在后面的括号中按顺序提供代入的值就可以了。占位的%和括号中的值在数量上必须相等，类型也要匹配。
('Mike', 87)这种用()表示的一组数据在python中被称为元组（tuple），是python的一种基本数据结构，以后我们还会用到。

**九、变量强制类型转换**

int(x) #把x转换成整数
float(x) #把x转换成浮点数
str(x) #把x转换成字符串
bool(x) #把x转换成bool值

例如，以下等式的结果均为真：int('123') == 123
float('3.3') == 3.3
str(111) == '111'
bool(0) == False

 bool('abc') == True
bool('False') == True
bool('') == False

并不是所有的值都能做类型转换，比如int('abc')同样会报错，python没办法把它转成一个数。

关于bool类型的转换，在python中，以下数值会被认为是False：

为0的数字，包括0，0.0
空字符串，包括''，""
表示空值的None （None是python中的一个特殊值，表示什么都没有，它和0、空字符、False、空集合都不一样）
空集合，包括()，[]，{}
其他的值都认为是True。

**十、函数**

如果我们要写一个函数，就需要去定义它。Python里定义函数的关键字是def，格式如下：

def sayHello():

 print 'hello world!'

我们用sayHello()来调用这个函数。（注意括号必须带着）

return是函数的结束语句，return后面的值被作为这个函数的返回值。函数中任何地方的return被执行到的时候，这个函数就会结束。

**十一、list列表**

**格式**：用中括号包围、逗号隔开的一组元素；列表中的元素可以是不同的类型。

例如，range(1,10)或者[1, 2, 3, 4, 5, 6, 7, 8, 9]；I = [365, 'everyday', 0.618, True]。

**操作list：**

1. 访问list中的元素

python计数是从0开始，例如I[0]=365, I[1]= 'everyday'

2. 修改list中的元素

修改list中的某一个元素，只需要直接给那个元素赋值就可以了：l[0] = 123

注意如下例子：c=[1,2,3]

b=c

a=c

b[0]=2

print a 得到 a=[2,2,3]

特别注意：如果用一个list赋值给另一个list，但是想要改变一个list的时候另一个不变，这时需用如下赋值方法： a=[1,2]

b=[]

b[0 : len(a)]=a[0 : len(a)]

b.append(3)

print a 得到[1,2]

print b 得到[1,2,3]

3. 向list中添加元素

list有一个**append**方法，可以增加元素。以I这个列表为例，调用的方法是：

I.append(1024) #注意这里是小括号

输出I，你会看到[123, 'everyday', 0.618, True, 1024]，1024被添加到了I，成为最后一个元素。（第一个元素在上一步被改成了123。）然后同样可以用I[4]得到1024。

4. 删除list中的元素

删除list中的某一个元素，要用到**del**：

del I[0]，然后输出I，得到['everyday', 0.618, True, 1024]。这时候再调用I[0]，会得到'everyday'，其他元素的序号也相应提前。

例子，#==== 点球小游戏 ====#

**from random import choice**

print 'Choose one side to shoot:'

print 'left, center, right'

you = raw\_input()

print 'You kicked ' + you

direction = ['left', 'center', 'right']

com = choice(direction)

print 'Computer saved ' + com

if you != com:

   print 'Goal!'

else:

   print 'Oops...'

1. 求和：例如 sum([1,2,3])=6, sum(range(101))=5050

list有两类常用操作：**索引(index)和切片(slice)**。

用[]加序号访问的方法就是**索引操作**。除了指定位置进行索引外，list还可以处理负数的索引，例如I[-1]表示l中的最后一个元素，l[-3]表示倒数第3个元素。

**切片操作**符是在[]内提供一对可选数字，用冒号分割。冒号前的数表示切片的开始位置，冒号后的数字表示切片到哪里结束。同样，计数从0开始。注意，开始位置包含在切片中，而结束位置不包括。例如，I = [365, 'everyday', 0.618, True]，I[1:3]得到的结果是['everyday', 0.618]。

如果不指定第一个数，切片就从列表第一个元素开始。

如果不指定第二个数，就一直到最后一个元素结束。

都不指定，则返回整个列表的一个拷贝。

I[:3] I[1:] I[:]

同索引一样，切片中的数字也可以使用负数。比如：I[1:-1] 得到['everyday', 0.618]

**十二、字符串的索引和切片（类似list的操作）**

1. 遍历
通过for...in可以遍历字符串中的每一个字符。
word = 'helloworld'
for c in word:
　print c
2. 索引访问
通过[]加索引的方式，访问字符串中的某个字符。
word[0]= 'h'
word[-2]= 'l'
与list不同的是，字符串不能通过索引访问去更改其中的字符。
word[1] = 'a' 这样的赋值是错误的。
3. 切片
通过两个参数，截取一段子串，具体规则和list相同。
print word[5:7]
print word[:-5]
print word[:]
4. 连接字符
join方法也可以对字符串使用，与list的join用法类似。

','.join(word)= 'h,e,l,l,o,w,o,r,l,d'

**十三、字符串的分割 --🡪 list**

假设需要把一个句子中的每一个单词拿出来单独处理。

sentence = 'I am an English sentence'

这时就需要对字符串进行分割: **sentence.split()**

**字符串.split()**会把字符串按照其中的空格进行分割，分割后的每一段都是一个新的字符串，最终返回这些字符串组成一个list。于是得到['I', 'am', 'an', 'English', 'sentence']

除了空格外，split()同时也会按照换行符\n，制表符\t进行分割。所以应该说，**split默认是按照空白字符进行分割。**

之所以说默认，是因为split还可以指定分割的符号。比如你有一个很长的字符串 section ='Hi. I am the one. Bye.' 通过指定分割符号为'.'，可以把每句话分开：**section.split('.')**

得到['Hi', ' I am the one', ' Bye', '']。这时候，'.'作为分割符被去掉了，而空格仍然保留在它的位置上。注意**最后那个空字符串**。每个'.'都会被作为分割符，即使它的后面没有其他字符，也会有一个空串被分割出来。例如 'aaa'.split('a') 将会得到['', '', '', '']，由四个空串组成的list。

**十四、连接list--🡪字符串**

**join：**和split正好相反,join把一个list中的所有字符串连接成一个字符串,而split是把一个字符串分割成很多字符串组成的list。

join的格式有些奇怪，它不是list的方法，而是字符串的方法。首先需要有一个字符串作为list中所有元素的连接符，然后再调用这个连接符的join方法，join的参数是被连接的list：
例如：s = ';'
 li = ['apple', 'pear', 'orange']
 fruit = s.join(li)
 print fruit
 得到结果'apple;pear;orange'。

又例如：''.join(['hello', 'world'])得到'helloworld'，字符串被无缝连接在一起。

# 十一\*\*、元组 （tuple）元组（tuple）也是一种序列，和我们用了很多次的list类似，只是元组中的元素在创建之后就不能被修改。例如position = (1, 2)

# geeks = ('Sheldon', 'Leonard', 'Rajesh', 'Howard')

都是元组的实例。它有和list同样的索引、切片、遍历等操作：

1. print postion[0]
2. for g in geeks:
3. print g
4. print geeks[1:3]

其实我们之前一直在用元组，就是在print语句中：

print '%s is %d years old' % ('Mike', 23)

('Mike', 23)就是一个元组。这是元组最常见的用处。

再来看一下元组作为函数返回值的例子：

1. def get\_pos(n):
2. return (n/2, n\*2)

得到这个函数的返回值有两种形式，一种是根据返回值元组中元素的个数提供变量：

1. x, y = get\_pos(50)
2. print x
3. print y

这就是我们在开头那句代码中使用的方式。
还有一种方法是用一个变量记录返回的元组：

1. pos = get\_pos(50)
2. print pos[0]
3. print pos[1]

**十五、读文件**

新建个文件，就叫它data.txt。在里面随便写上一些话，保存。把这个文件放在接下来你打算保存代码的文件夹下，这么做是为了方便我们的程序找到它。
 打开一个文件的命令很简单：**file('文件名')**
这里的文件名可以用文件的完整路径，也可以是相对路径。因为我们把要读取的文件和代码放在了同一个文件夹下，所以只需要写它的文件名就够了。
f = file('data.txt') #注意这里f不是字符串,是文件对象
 但这一步只是打开了一个文件，并没有得到其中的内容。变量f保存了这个文件，还需要去读取它的内容。你可以通过read()函数把文件内所有内容读进一个字符串中。
data = f.read() #这里data是字符串
 做完对文件的操作之后，记得用close()关闭文件，释放资源。

完整程序示例：
f = file('data.txt')
data = f.read()
print data
f.close()

读取文件内容的方法还有：
readline() #读取一行内容
readlines() #把内容按行读取至一个list中
去替换程序的第二行，看看它们的区别。

**十六、写文件**

**打开文件的几种模式：**

python默认是以只读模式打开文件。如果想要写入内容，在打开文件的时候需要指定打开模式为写入： **f = file('output.txt', 'w')**
'w'就是writing，以这种模式打开文件，原来文件中的内容会被你新写入的内容覆盖掉，如果文件不存在，会自动创建文件。不加参数时，file为你默认为'r'，reading，只读模式，文件必须存在，否则引发异常。
 另外还有一种模式是**f = file('output.txt', 'a')**，其中'a'是appending。它也是一种写入模式，但你写入的内容不会覆盖之前的内容，而是添加到文件中。
 打开文件还有一种方法，就是open()，用法和file()是一致的。

**写文件：**
写入内容的方法同样简单：

f.write('a string you want to write')

write的参数可以是一个字符串，或者一个字符串变量。

示例程序：
data = 'I will be in a file.\nSo cool!'
out = open('output.txt', 'w')
out.write(data)
out.close()

在你的程序保存目录下，打开output.txt就会看到结果。

类似的，writelines(list) #把list的每一个元素写成一行

没有writeline(list)。

**十七、处理文件中的数据**

比如我现在拿到一份文档，里面有某个班级里所有学生的平时作业成绩。因为每个人交作业的次数不一样，所以成绩的数目也不同，没交作业的时候就没有分。我现在需要统计每个学生的平时作业总得分。

看一下我们的文档里的数据：
#-- scores.txt
刘备 23 35 44 47 51
关羽 60 77 68
张飞 97 99 89 91
诸葛亮 100

1. 先把文件读进来：
f = file('scores.txt')
2. 取得文件中的数据。因为每一行都是一条学生成绩的记录，所以用readlines，把每一行分开，便于之后的数据处理：
lines = f.readlines()
f.close()
提示：在程序中，经常使用print来查看数据的中间状态，可以便于你理解程序的运行。比如这里你可以print lines，看一下内容被存成了什么格式。
3. 对每一条数据进行处理。按照空格，把姓名、每次的成绩分割开：
for line in lines:
　data = line.split() #这里data是list
接下来的4、5两个步骤都是针对一条数据的处理，所以都是在for循环的内部。
4. 整个程序最核心的部分到了。如何把一个学生的几次成绩合并，并保存起来呢？我的做法是：对于每一条数据，都新建一个字符串，把学生的名字和算好的总成绩保存进去。最后再把这些字符串一起保存到文件中：
sum = 0
for score in data[1:]:
　sum += int(score)
result = '%s\t: %d\n' % (data[0], sum)
这里几个要注意的点：
 对于每一行分割的数据，data[0]是姓名，data[1:]是所有成绩组成的列表。
每次循环中，sum都要先清零。
 score是一个字符串，为了做计算，需要转成整数值int。
 result中，我加了一个制表符\t和换行符\n，让输出的结果更好看些。
5. 得到一个学生的总成绩后，把它添加到一个list中。
results.append(result) #这里results是list
results需要在循环之前初始化results = []
6. 最后，全部成绩处理完毕后，把results中的内容保存至文件。因为results是一个字符串组成的list，这里我们直接用writelines方法：
output = file('result.txt', 'w')
output.writelines(results)
outpus.close()

**十八、异常处理**

在python中，可以使用try...except语句来处理异常。做法是，把可能引发异常的语句放在try-块中，把处理异常的语句放在except-块中。例如：

try:
　f = file('non-exist.txt')
　print 'File opened!'
　f.close()
except:
　print 'File not exists.'
print 'Done'

# 十九、字典dictionary

# python中的基本类型--字典（dictionary）：

# 字典这种数据结构有点像我们平常用的通讯录，有一个名字和这个名字对应的信息。在字典中，名字叫做“键(key)”，对应的内容信息叫做“值(value)”。字典就是一个键/值对的集合。键/值对用冒号分割，每个对之间用逗号分割，整个字典包括在花括号中。

# d = {key1 : value1, key2 : value2 }

# 举个简单的字典例子：score = {'萧峰': 95, '段誉': 97,'虚竹': 89} 关于字典的键要注意的是：1.键必须是唯一的；2.键只能是简单对象，比如字符串、整数、浮点数、bool值。list就不能作为键，但是 可以作为值。3.python字典中的键/值对没有顺序，我们无法用索引访问字典中的某一项，而是要用键来访问。例如print score['段誉'] 或者print score.get('段誉')。如果你的键是字符串，通过键访问的时候就需要加引号，如果是数字作为键则不用。注意，字典类的get方法是按照给定key寻找对应项，如果不存在这样的key，就返回空值None

4.字典也可以通过for...in遍历：
for name in score:
　print score[name] #这里要用中括号

或者 print score.get(name) #这里要用小括号
 注意，遍历的变量中存储的是字典的键。
5.如果要改变某一项的值，就直接给这一项赋值： score['虚竹'] = 91
6.增加一项字典项的方法是，给一个新键赋值： score['慕容复'] = 88
7.删除一项字典项的方法是del： del score['萧峰']
 注意，这个键必须已存在于字典中。
8.如果你想新建一个空的字典，只需要: d = {}

# 二十、模块

# python自带了功能丰富的标准库，另外还有数量庞大的各种第三方库。使用这些功能的基本方法就是使用模块。通过函数，可以在程序里重用代码；通过模块，则可以重用别的程序中的代码。 模块可以理解为是一个包含了函数和变量的py文件。在你的程序中引入了某个模块，就可以使用其中的函数和变量。

来看一个我们之前使用过的模块： import random
**import语句**告诉python，我们要用**random模块**中的内容。然后便可以使用random中的方法，比如：random.randint(1, 10)
 random.randchoic([1, 3, 5])
注意，函数前面需要加上“random.”，这样python才知道你是要调用random中的方法。
想知道random有哪些函数和变量，可以用**dir()**方法：**dir(random)**

如果你只是用到random中的某一个函数或变量，也可以通过from...import...指明：
from random import randint；
from math import pi
print pi

等同于

import math

print math.pi
为了便于理解和避免冲突，你还可以给引入的方法换个名字：
from math import pi as math\_pi
print math\_pi

# 二十一、函数的默认参数

之前我们用过函数，比如：
def hello(name):
　print 'hello ' + name
然后我们去调用这个函数：hello('world')
程序就会输出： hello world
 如果很多时候，我们都是用world来调用这个函数，少数情况才会去改参数。那么，我们就可以给这个函数一个默认参数：
def hello(name = 'world'):
　print 'hello ' + name
当你没有提供参数值时，这个参数就会使用默认值；如果你提供了，就用你给的。这样，在默认情况下，你只要调用hello()就可以输出 hello world
同样你也可以指定参数：hello('python')，输出hello python

注意，当函数有多个参数时，如果你想给部分参数提供默认参数，那么这些参数必须在参数的末尾。比如：
def func(a, b=5)
是正确的
def func(a=5, b)
就会出错

# 二十二、数学运算python的数学运算模块叫做math，再用之前，需要 import math1.math包里有两个常量： math.pi 圆周率π：3.141592... math.e 自然常数：2.718281...2.数值运算：math.ceil(x) 对x向上取整，比如x=1.2，返回2math.floor(x) 对x向下取整，比如x=1.2，返回1math.pow(x,y) 指数运算，得到x的y次方math.log(x)对数，默认基底为e。可以使用第二个参数，来改变对数的基底。比如math.log(100, 10)math.sqrt(x) 平方根math.fabs(x) 绝对值三角函数: math.sin(x)math.cos(x)math.tan(x)math.asin(x)math.acos(x)math.atan(x)注意：这里的x是以弧度为单位，所以计算角度的话，需要先换算角度和弧度互换: math.degrees(x) 弧度转角度math.radians(x) 角度转弧度

以上是你平常可能会用到的函数。除此之外，还有一些，这里就不罗列，可以去

http://docs.python.org/2/library/math.html 查看官方的完整文档。

**二十三、类class和对象object**

class Person:

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

self.name = name

def sayHi(self):

print 'Hello, my name is', self.name

p = Person('Swaroop')

p.sayHi()

\_\_init\_\_函数会在类被创建的时候自动调用，用来初始化类。**它的参数，要在创建类的时候提供（所以定义p时Person后面的括号里有东西）**

**注意：**\_\_init\_\_前后各有**两道**下划线