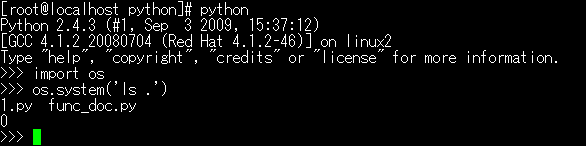
**第1章 基本概念**

**带提示符的Python解释器**

$ python  
Python 2.3.4 (#1, Oct 26 2004, 16:42:40)  
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print 'hello world'  
hello world  
>>>

**例子：**

****

**使用源文件**

#!/usr/bin/python  
# Filename : helloworld.py  
print 'Hello World'

**执行及输出：**

$ python helloworld.py  
Hello World

**注：这个程序也要先加上可执行权限才能运行。（例：chmod +x helloworld.py）**

**字面意义上的常量**

例：5、 1.23、 9.25e-3、'This is a string'、"It's a string!"

**Python字符串**

* **使用单引号（'）**

可以用单引号指示字符串。

例：'Quote me on this'

所有的空白，即空格和制表符都照原样保留。

* **使用双引号（"）**

在双引号中的字符串与单引号中的字符串的使用完全相同。

例："What's your name?"。

* **使用三引号（'''或"""）**

利用三引号，你可以指示一个多行的字符串。你可以在三引号中自由的使用单引号和双引号。例如：

'''This is a multi-line string. This is the first line.  
This is the second line.  
"What's your name?," I asked.  
He said "Bond, James Bond."  
'''

* **转义符**

假如有个字符串What's your name,那用字符串表示时就要进行转义。

单引号表示：‘What's your name?’×‘What\'s your name?’√

双引号表示：“What's your name?” √

另外值得注意的一件事是，在一个字符串中，行末的单独一个反斜杠表示字符串在下一行继续，而不是开始一个新的行。例如：

"This is the first sentence.\  
This is the second sentence."

等价于"This is the first sentence. This is the second sentence."

**>>>** 'spam eggs'

'spam eggs'

**>>>** 'doesn**\'**t'

"doesn't"

**>>>** "doesn't"

"doesn't"

**>>>** '"Yes," he said.'

'"Yes," he said.'

**>>>** "**\"**Yes,**\"** he said."

'"Yes," he said.'

**>>>** '"Isn**\'**t," she said.'

'"Isn\'t," she said.'

* **自然字符串**

如果你想要指示某些不需要如转义符那样的特别处理的字符串，那么你需要指定一个自然字符串。自然字符串通过给字符串加上前缀r或R来指定。例如r"Newlines are indicated by \n"。

给C/C++程序员的注释  
在Python中没有专门的char数据类型。确实没有需要有这个类型，我相信你不会为此而烦恼。

给正则表达式用户的注释  
一定要用自然字符串处理正则表达式。否则会需要使用很多的反斜杠。例如，后向引用符可以写成'\\1'或r'\1'。

**变量**

使用变量时只需要给它们赋一个值。不需要声明或定义数据类型。给变量赋值的时候决定这个变量是什么类型的。

例：

i = 8 表示i是个整型变量

i = 8.0 表示i是个浮点型变量

i = ‘I am a string‘ 表示i是字符串变量

i = class() 表示i是类的对象

**[root@localhost python]# python**

**Python 2.4.3 (#1, Sep 3 2009, 15:37:12)**

**[GCC 4.1.2 20080704 (Red Hat 4.1.2-46)] on linux2**

**Type "help", "copyright", "credits" or "license" for more information.**

**>>> 8.0/3**

**2.6666666666666665**

**>>> 8/3**

**2**

**>>>**

**对象**

Python把在程序中用到的任何东西都称为 对象 *。*

数、字符串、函数都被看作对象。

**缩进**

在逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。

同一层次的语句**必须**有相同的缩进。每一组这样的语句称为一个**块**。

错误的缩进会引发错误。例如：

你需要记住的一样东西是错误的缩进会引发错误。例如：

i = 5  
 print 'Value is', i # Error! Notice a single space at the start of the line  
print 'I repeat, the value is', i

当你运行这个程序的时候，你会得到下面的错误：

  File "whitespace.py", line 4  
    print 'Value is', i # Error! Notice a single space at the start of the line  
    ^  
SyntaxError: invalid syntax

**运算符**

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **名称** | **说明** | **例子** |
| + | 加 | 两个对象相加 | 3 + 5得到8。'a' + 'b'得到'ab'。 |
| - | 减 | 得到负数或是一个数减去另一个数 | -5.2得到一个负数。50 - 24得到26。 |
| \* | 乘 | 两个数相乘或是返回一个被重复若干次的字符串 | 2 \* 3得到6。'la' \* 3得到'lalala'。 |
| \*\* | 幂 | 返回x的y次幂 | 3 \*\* 4得到81（即3 \* 3 \* 3 \* 3） |
| / | 除 | x除以y | 4/3得到1（整数的除法得到整数结果）。4.0/3或4/3.0得到1.3333333333333333 |
| // | 取整除 | 返回商的整数部分 | 4 // 3.0得到1.0 |
| % | 取模 | 返回除法的余数 | 8%3得到2。-25.5%2.25得到1.5 |
| << | 左移 | 把一个数的比特向左移一定数目（每个数在内存中都表示为比特或二进制数字，即0和1） | 2 << 2得到8。——2按比特表示为10 |
| >> | 右移 | 把一个数的比特向右移一定数目 | 11 >> 1得到5。——11按比特表示为1011，向右移动1比特后得到101，即十进制的5。 |
| & | 按位与 | 数的按位与 | 5 & 3得到1。 |
| | | 按位或 | 数的按位或 | 5 | 3得到7。 |
| ^ | 按位异或 | 数的按位异或 | 5 ^ 3得到6 |
| ~ | 按位翻转 | x的按位翻转是-(x+1) | ~5得到-6。 |
| < | 小于 | 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。 | 5 < 3返回0（即False）而3 < 5返回1（即True）。比较可以被任意连接：3 < 5 < 7返回True。 |
| > | 大于 | 返回x是否大于y | 5 > 3返回True。如果两个操作数都是数字，它们首先被转换为一个共同的类型。否则，它总是返回False。 |
| <= | 小于等于 | 返回x是否小于等于y | x = 3; y = 6; x <= y返回True。 |
| >= | 大于等于 | 返回x是否大于等于y | x = 4; y = 3; x >= y返回True。 |
| == | 等于 | 比较对象是否相等 | x = 2; y = 2; x == y返回True。x = 'str'; y = 'stR'; x == y返回False。x = 'str'; y = 'str'; x == y返回True。 |
| != | 不等于 | 比较两个对象是否不相等 | x = 2; y = 3; x != y返回True。 |
| not | 布尔“非” | 如果x为True，返回False。如果x为False，它返回True。 | x = True; not x返回False。 |
| and | 布尔“与” | 如果x为False，x and y返回False，否则它返回y的计算值。 | x = False; y = True; x and y，由于x是False，返回False。在这里，Python不会计算y，因为它知道这个表达式的值肯定是False（因为x是False）。这个现象称为短路计算。 |
| or | 布尔“或” | 如果x是True，它返回True，否则它返回y的计算值。 | x = True; y = False; x or y返回True。短路计算在这里也适用。 |

**运算符优先级**

|  |  |
| --- | --- |
| **运算符** | **描述** |
| lambda | Lambda表达式 |
| or | 布尔“或” |
| and | 布尔“与” |
| not x | 布尔“非” |
| in，not in | 成员测试 |
| is，is not | 同一性测试 |
| <，<=，>，>=，!=，== | 比较 |
| | | 按位或 |
| ^ | 按位异或 |
| & | 按位与 |
| <<，>> | 移位 |
| +，- | 加法与减法 |
| \*，/，% | 乘法、除法与取余 |
| +x，-x | 正负号 |
| ~x | 按位翻转 |
| \*\* | 指数 |
| x.attribute | 属性参考 |
| x[index] | 下标 |
| x[index:index] | 寻址段 |
| f(arguments...) | 函数调用 |
| (experession,...) | 绑定或元组显示 |
| [expression,...] | 列表显示 |
| {key:datum,...} | 字典显示 |
| 'expression,...' | 字符串转换 |

**使用if语句**

**例6.1 使用if语句**

#!/usr/bin/python

# Filename: if.py   
  
number = 23  
guess = int(raw\_input('Enter an integer : '))  
  
if guess == number:  
    print 'Congratulations, you guessed it.' # New block starts here  
    print "(but you do not win any prizes!)" # New block ends here  
elif guess < number:  
    print 'No, it is a little higher than that' # Another block  
    # You can do whatever you want in a block ...  
else:  
    print 'No, it is a little lower than that'   
    # you must have guess > number to reach here  
  
print 'Done'  
# This last statement is always executed, after the if statement is executed

**输出**

$ python if.py  
Enter an integer : 50  
No, it is a little lower than that  
Done  
$ python if.py  
Enter an integer : 22  
No, it is a little higher than that  
Done  
$ python if.py  
Enter an integer : 23  
Congratulations, you guessed it.  
(but you do not win any prizes!)  
Done

注意if语句在结尾处包含一个冒号——我们通过它告诉Python下面跟着一个语句块。

给C/C++程序员的注释  
在Python中没有switch语句。你可以使用if..elif..else语句来完成同样的工作（在某些场合，使用[字典](http://sebug.net/paper/python/ch09s04.html)会更加快捷。）

**使用while语句**

#!/usr/bin/python  
# Filename: while.py  
  
number = 23  
running = True  
  
while running:  
    guess = int(raw\_input('Enter an integer : '))  
  
    if guess == number:  
        print 'Congratulations, you guessed it.'   
        running = False # this causes the while loop to stop  
    elif guess < number:  
        print 'No, it is a little higher than that'   
    else:  
        print 'No, it is a little lower than that'   
else:  
    print 'The while loop is over.'   
    # Do anything else you want to do here  
  
print 'Done'

## 输出

$ python while.py  
Enter an integer : 50  
No, it is a little lower than that.  
Enter an integer : 22  
No, it is a little higher than that.  
Enter an integer : 23  
Congratulations, you guessed it.  
The while loop is over.

给C/C++程序员的注释  
记住，你可以在while循环中使用一个else从句。

使用for语句

#!/usr/bin/python  
# Filename: for.py  
  
for i in range(1, 5):  
    print i  
else:  
    print 'The for loop is over'

## 输出

$ python for.py  
1  
2  
3  
4  
The for loop is over

给C/C++程序员的注释  
在C/C++中，如果你想要写for (int i = 0; i < 5; i++)，那么用Python，你写成for i in range(0,5)。你会注意到，Python的for循环更加简单、明白、不易出错。

使用break语句

#!/usr/bin/python  
# Filename: break.py  
  
while True:  
    s = raw\_input('Enter something : ')  
    if s == 'quit':  
        break  
    print 'Length of the string is', len(s)  
print 'Done'

## 输出

$ python break.py  
Enter something : Programming is fun  
Length of the string is 18  
Enter something : When the work is done  
Length of the string is 21  
Enter something : if you wanna make your work also fun:  
Length of the string is 37  
Enter something :       use Python!  
Length of the string is 12  
Enter something : quit  
Done

**第7章 函数**

定义函数

#!/usr/bin/python  
# Filename: function1.py  
  
def sayHello():  
    print 'Hello World!' # block belonging to the function  
  
sayHello() # call the function

## 输出

$ python function1.py  
Hello World!

使用函数形参

#!/usr/bin/python  
# Filename: func\_param.py  
  
def printMax(a, b):  
    if a > b:  
        print a, 'is maximum'  
    else:  
        print b, 'is maximum'  
  
printMax(3, 4) # directly give literal values  
  
x = 5  
y = 7  
  
printMax(x, y) # give variables as arguments

## 输出

$ python func\_param.py  
4 is maximum  
7 is maximum

例7.3 使用局部变量

#!/usr/bin/python  
# Filename: func\_local.py  
  
def func(x):  
    print 'x is', x  
    x = 2  
    print 'Changed local x to', x  
  
x = 50  
func(x)  
print 'x is still', x

## 输出

$ python func\_local.py  
x is 50  
Changed local x to 2  
x is still 50

使用global语句

#!/usr/bin/python  
# Filename: func\_global.py  
def func():  
    global x  
    print 'x is', x  
    x = 2  
    print 'Changed local x to', x  
  
x = 50  
func()  
print 'Value of x is', x

## 输出

$ python func\_global.py  
x is 50  
Changed global x to 2  
Value of x is 2

使用默认参数值

#!/usr/bin/python  
# Filename: func\_default.py  
  
def say(message, times = 1):  
    print message \* times  
  
say('Hello')  
say('World', 5)

输出

$ python func\_default.py  
Hello  
WorldWorldWorldWorldWorld

重要  
只有在形参表末尾的那些参数可以有默认参数值，即你不能在声明函数形参的时候，先声明有默认值的形参而后声明没有默认值的形参。  
这是因为赋给形参的值是根据位置而赋值的。例如，def func(a, b=5)是有效的，但是def func(a=5, b)是 无效 的。

使用关键参数

#!/usr/bin/python  
# Filename: func\_key.py  
  
def func(a, b=5, c=10):  
    print 'a is', a, 'and b is', b, 'and c is', c  
  
func(3, 7)  
func(25, c=24)  
func(c=50, a=100)

## 输出

$ python func\_key.py  
a is 3 and b is 7 and c is 10  
a is 25 and b is 5 and c is 24  
a is 100 and b is 5 and c is 50

return语句

#!/usr/bin/python  
# Filename: func\_return.py  
  
def maximum(x, y):  
    if x > y:  
        return x  
    else:  
        return y  
  
print maximum(2, 3)

## 输出

$ python func\_return.py  
3

**DocStrings**

Python有一个很奇妙的特性，称为 文档字符串 ，它通常被简称为 docstrings 。DocStrings是一个重要的工具，由于它帮助你的程序文档更加简单易懂，你应该尽量使用它。你甚至可以在程序运行的时候，从函数恢复文档字符串！

使用DocStrings

#!/usr/bin/python  
# Filename: func\_doc.py  
def printMax(x, y):  
    '''Prints the maximum of two numbers.  
  
    The two values must be integers.'''  
    x = int(x) # convert to integers, if possible  
    y = int(y)  
    if x > y:  
        print x, 'is maximum'  
    else:  
        print y, 'is maximum'

printMax(3, 5)  
print printMax.\_\_doc\_\_

## 输出

$ python func\_doc.py  
5 is maximum  
Prints the maximum of two numbers.  
  
        The two values must be integers.

文档字符串的惯例是一个多行字符串，它的首行以大写字母开始，句号结尾。第二行是空行，从第三行开始是详细的描述。 强烈建议 你在你的函数中使用文档字符串时遵循这个惯例。

**第8章 模块**

模块基本上就是一个包含了所有你定义的函数和变量的文件。为了在其他程序中重用模块，模块的文件名**必须**以.py为扩展名。

模块可以从其他程序 *输入* 以便利用它的功能。这也是我们使用Python标准库的方法。

使用sys模块

#!/usr/bin/python  
# Filename: using\_sys.py  
  
import sys  
  
print 'The command line arguments are:'  
for i in sys.argv:  
    print i  
  
print '\n\nThe PYTHONPATH is', sys.path, '\n'

## 输出

$ python using\_sys.py we are arguments  
The command line arguments are:  
using\_sys.py  
we  
are  
arguments  
  
  
The PYTHONPATH is ['/home/swaroop/byte/code', '/usr/lib/python23.zip',  
'/usr/lib/python2.3', '/usr/lib/python2.3/plat-linux2',  
'/usr/lib/python2.3/lib-tk', '/usr/lib/python2.3/lib-dynload',  
'/usr/lib/python2.3/site-packages', '/usr/lib/python2.3/site-packages/gtk-2.0']

**from..import语句**

如果你想要直接输入argv变量到你的程序中（避免在每次使用它时打sys.），那么你可以使用from sys import argv语句。如果你想要输入所有sys模块使用的名字，那么你可以使用from sys import \*语句

[root@localhost python]# python

Python 2.4.3 (#1, Sep 3 2009, 15:37:12)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-46)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> from sys import argv

>>> argv[0] 这种方法就不需要 sys.的前缀了。

''

>>>

**但是这种方法不太好，有可能和自己模块定义使用名称产生冲突。**

使用模块的\_\_name\_\_

每个模块都有一个名称，在模块中可以通过语句来找出模块的名称。这在一个场合特别有用——就如前面所提到的，当一个模块被第一次输入的时候，这个模块的主块将被运行。假如我们只想在程序本身被使用的时候运行主块，而在它被别的模块输入的时候不运行主块，我们该怎么做呢？这可以通过模块的\_\_name\_\_属性完成。

#!/usr/bin/python  
# Filename: using\_name.py  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    print 'This program is being run by itself'  
else:  
    print 'I am being imported from another module'

## 输出

$ python using\_name.py  
This program is being run by itself  
  
$ python  
>>> import using\_name  
I am being imported from another module  
>>>

[root@localhost python]# ls

1.py 2.py func\_doc.py using\_name.py

[root@localhost python]# cat using\_name.py

#!/usr/bin/python

# filename: using\_name.py

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

print \_\_name\_\_

print 'this program is being run by itself'

else:

print \_\_name\_\_

print 'i am being imported from another module'

[root@localhost python]# python using\_name.py

\_\_main\_\_

this program is being run by itself

[root@localhost python]# python

python 2.4.3 (#1, sep 3 2009, 15:37:12)

[gcc 4.1.2 20080704 (red hat 4.1.2-46)] on linux2

type "help", "copyright", "credits" or "license" for more information.

>>> import using\_name

using\_name

i am being imported from another module

>>>

[root@localhost python]# ls

1.py 2.py func\_doc.py using\_name.py using\_name.pyc

dir()函数

你可以使用内建的dir函数来列出模块定义的标识符。标识符有函数、类和变量。

当你为dir()提供一个模块名的时候，它返回模块定义的名称列表。如果不提供参数，它返回当前模块中定义的名称列表。

**使用dir函数**

$ python  
>>> import sys  
>>> dir(sys) # get list of attributes for sys module  
['\_\_displayhook\_\_', '\_\_doc\_\_', '\_\_excepthook\_\_', '\_\_name\_\_', '\_\_stderr\_\_',  
'\_\_stdin\_\_', '\_\_stdout\_\_', '\_getframe', 'api\_version', 'argv',  
'builtin\_module\_names', 'byteorder', 'call\_tracing', 'callstats',  
'copyright', 'displayhook', 'exc\_clear', 'exc\_info', 'exc\_type',  
'excepthook', 'exec\_prefix', 'executable', 'exit', 'getcheckinterval',  
'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding',  
'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint', 'maxunicode',  
'meta\_path','modules', 'path', 'path\_hooks', 'path\_importer\_cache',  
'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags',  
'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',  
'version', 'version\_info', 'warnoptions']  
>>> dir() # get list of attributes for current module  
['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'sys']  
>>>  
>>> a = 5 # create a new variable 'a'  
>>> dir()  
['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'a', 'sys']  
>>>  
>>> del a # delete/remove a name  
>>>  
>>> dir()  
['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'sys']  
>>>

**第9章 数据结构**

在Python中有三种内建的数据结构——列表、元组和字典

**列表**

list是处理一组有序项目的数据结构，即你可以在一个列表中存储一个 *序列* 的项目。在每个项目之间用逗号分割。

列表中的项目应该包括在方括号中，这样Python就知道你是在指明一个列表。一旦你创建了一个列表，你可以添加、删除或是搜索列表中的项目。由于你可以增加或删除项目，我们说列表是 *可变的* 数据类型，即这种类型是可以被改变的。

使用列表

#!/usr/bin/python  
# Filename: using\_list.py  
  
# This is my shopping list  
shoplist = ['apple', 'mango', 'carrot', 'banana']  
print 'I have', len(shoplist),'items to purchase.'  
print 'These items are:', # Notice the comma at end of the line  
for item in shoplist:  
    print item,  
print '\nI also have to buy rice.'  
shoplist.append('rice')  
print 'My shopping list is now', shoplist  
print 'I will sort my list now'  
shoplist.sort()  
print 'Sorted shopping list is', shoplist  
print 'The first item I will buy is', shoplist[0]  
olditem = shoplist[0]  
del shoplist[0]  
print 'I bought the', olditem  
print 'My shopping list is now', shoplist

## 输出

$ python using\_list.py  
I have 4 items to purchase.  
These items are: apple mango carrot banana  
I also have to buy rice.  
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']  
I will sort my list now  
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']  
The first item I will buy is apple  
I bought the apple  
My shopping list is now ['banana', 'carrot', 'mango', 'rice']

使用元组

#!/usr/bin/python  
# Filename: using\_tuple.py  
  
zoo = ('wolf', 'elephant', 'penguin')  
print 'Number of animals in the zoo is', len(zoo)  
  
new\_zoo = ('monkey', 'dolphin', zoo)  
print 'Number of animals in the new zoo is', len(new\_zoo)  
print 'All animals in new zoo are', new\_zoo  
print 'Animals brought from old zoo are', new\_zoo[2]  
print 'Last animal brought from old zoo is', new\_zoo[2][2]

## 输出

$ python using\_tuple.py  
Number of animals in the zoo is 3  
Number of animals in the new zoo is 3  
All animals in new zoo are ('monkey', 'dolphin', ('wolf', 'elephant', 'penguin'))  
Animals brought from old zoo are ('wolf', 'elephant', 'penguin')  
Last animal brought from old zoo is penguin

**含有0个或1个项目的元组。**一个空的元组由一对空的圆括号组成，如myempty = ()。然而，含有单个元素的元组就不那么简单了。你必须在第一个（唯一一个）项目后跟一个逗号，这样Python才能区分元组和表达式中一个带圆括号的对象。即如果你想要的是一个包含项目2的元组的时候，你应该指明singleton = (2 , )。

**字典**

字典类似于你通过联系人名字查找地址和联系人详细情况的地址簿，即，我们把**键**（名字）和**值**（详细情况）联系在一起。注意，键必须是唯一的，就像如果有两个人恰巧同名的话，你无法找到正确的信息。

注意，你只能使用不可变的对象（比如字符串）来作为字典的键，但是你可以不可变或可变的对象作为字典的值。基本说来就是，你应该只使用简单的对象作为键。

键值对在字典中以这样的方式标记：d = {key1 : value1, key2 : value2 }。注意它们的键/值对用冒号分割，而各个对用逗号分割，所有这些都包括在花括号中。

字典是dict类的实例/对象。

使用字典

#!/usr/bin/python  
# Filename: using\_dict.py  
  
# 'ab' is short for 'a'ddress'b'ook  
  
ab = {       'Swaroop'   : 'swaroopch@byteofpython.info',  
             'Larry'     : 'larry@wall.org',  
             'Matsumoto' : 'matz@ruby-lang.org',  
             'Spammer'   : 'spammer@hotmail.com'  
     }  
  
print "Swaroop's address is %s" % ab['Swaroop']  
  
# Adding a key/value pair  
ab['Guido'] = 'guido@python.org'  
  
# Deleting a key/value pair  
del ab['Spammer']  
  
print '\nThere are %d contacts in the address-book\n' % len(ab)  
for name, address in ab.items():  
    print 'Contact %s at %s' % (name, address)  
  
if 'Guido' in ab: # OR ab.has\_key('Guido')  
    print "\nGuido's address is %s" % ab['Guido']

## 输出

$ python using\_dict.py  
Swaroop's address is swaroopch@byteofpython.info  
  
There are 4 contacts in the address-book  
  
Contact Swaroop at swaroopch@byteofpython.info  
Contact Matsumoto at matz@ruby-lang.org  
Contact Larry at larry@wall.org  
Contact Guido at guido@python.org  
  
Guido's address is guido@python.org

**序列**

列表、元组和字符串都是序列，但是序列是什么，它们为什么如此特别呢？序列的两个主要特点是**索引**操作符和**切片**操作符。索引操作符让我们可以从序列中抓取一个特定项目。切片操作符让我们能够获取序列的一个切片，即一部分序列。

使用序列

#!/usr/bin/python  
# Filename: seq.py  
  
shoplist = ['apple', 'mango', 'carrot', 'banana']  
  
# Indexing or 'Subscription' operation  
print 'Item 0 is', shoplist[0]  
print 'Item 1 is', shoplist[1]  
print 'Item 2 is', shoplist[2]  
print 'Item 3 is', shoplist[3]  
print 'Item -1 is', shoplist[-1]  
print 'Item -2 is', shoplist[-2]  
  
# Slicing on a list  
print 'Item 1 to 3 is', shoplist[1:3]  
print 'Item 2 to end is', shoplist[2:]  
print 'Item 1 to -1 is', shoplist[1:-1]  
print 'Item start to end is', shoplist[:]  
  
# Slicing on a string  
name = 'swaroop'  
print 'characters 1 to 3 is', name[1:3]  
print 'characters 2 to end is', name[2:]  
print 'characters 1 to -1 is', name[1:-1]  
print 'characters start to end is', name[:]

## 输出

$ python seq.py  
Item 0 is apple  
Item 1 is mango  
Item 2 is carrot  
Item 3 is banana  
Item -1 is banana  
Item -2 is carrot  
Item 1 to 3 is ['mango', 'carrot']  
Item 2 to end is ['carrot', 'banana']  
Item 1 to -1 is ['mango', 'carrot']  
Item start to end is ['apple', 'mango', 'carrot', 'banana']  
characters 1 to 3 is wa  
characters 2 to end is aroop  
characters 1 to -1 is waroo  
characters start to end is swaroop

**参考**

当你创建一个对象并给它赋一个变量的时候，这个变量仅仅 参考 那个对象，而不是表示这个对象本身！也就是说，变量名指向你计算机中存储那个对象的内存。这被称作名称到对象的**绑定**。

对象与参考

#!/usr/bin/python  
# Filename: reference.py  
  
print 'Simple Assignment'  
shoplist = ['apple', 'mango', 'carrot', 'banana']  
mylist = shoplist # mylist is just another name pointing to the same object!  
  
del shoplist[0]  
  
print 'shoplist is', shoplist  
print 'mylist is', mylist  
# notice that both shoplist and mylist both print the same list without  
# the 'apple' confirming that they point to the same object  
  
print 'Copy by making a full slice'  
mylist = shoplist[:] # make a copy by doing a full slice  
del mylist[0] # remove first item  
  
print 'shoplist is', shoplist  
print 'mylist is', mylist  
# notice that now the two lists are different

## 输出

$ python reference.py  
Simple Assignment  
shoplist is ['mango', 'carrot', 'banana']  
mylist is ['mango', 'carrot', 'banana']  
Copy by making a full slice  
shoplist is ['mango', 'carrot', 'banana']  
mylist is ['carrot', 'banana']

**第11章 面向对象的编程**

**类**

创建一个类

#!/usr/bin/python  
# Filename: simplestclass.py  
  
class Person:  
    pass # An empty block  
  
p = Person()  
print p

## 输出

$ python simplestclass.py  
<\_\_main\_\_.Person instance at 0xf6fcb18c>

使用对象的方法

#!/usr/bin/python  
# Filename: method.py  
  
class Person:  
    def sayHi(self):  
        print 'Hello, how are you?'  
  
p = Person()  
p.sayHi()  
  
# This short example can also be written as Person().sayHi()

## 输出

$ python method.py  
Hello, how are you?

类的方法与普通的函数只有一个特别的区别——它们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你不为这个参数赋值，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。

假如你有一个类称为MyClass和这个类的一个实例MyObject。当你调用这个对象的方法MyObject.method(arg1, arg2)的时候，这会由Python自动转为MyClass.method(MyObject, arg1, arg2)——这就是self的原理了。

使用\_\_init\_\_方法

#!/usr/bin/python  
# Filename: class\_init.py  
  
class Person:  
    def \_\_init\_\_(self, name):  
        self.name = name  
    def sayHi(self):  
        print 'Hello, my name is', self.name  
  
p = Person('Swaroop')  
p.sayHi()  
  
# This short example can also be written as Person('Swaroop').sayHi()

## 输出

$ python class\_init.py  
Hello, my name is Swaroop

**类与对象的方法**

我们已经讨论了类与对象的功能部分，现在我们来看一下它的数据部分。事实上，它们只是与类和对象的**名称空间** 绑定 的普通变量，即这些名称只在这些类与对象的前提下有效。

有两种类型的 域 ——类的变量和对象的变量，它们根据是类还是对象 拥有 这个变量而区分。

类的变量 由一个类的所有对象（实例）共享使用。只有一个类变量的拷贝，所以当某个对象对类的变量做了改动的时候，这个改动会反映到所有其他的实例上。

对象的变量 由类的每个对象/实例拥有。因此每个对象有自己对这个域的一份拷贝，即它们不是共享的，在同一个类的不同实例中，虽然对象的变量有相同的名称，但是是互不相关的。通过一个例子会使这个易于理解。

使用类与对象的变量

#!/usr/bin/python  
# Filename: objvar.py  
  
class Person:  
    '''Represents a person.'''  
    population = 0  
  
    def \_\_init\_\_(self, name):  
        '''Initializes the person's data.'''  
        self.name = name  
        print '(Initializing %s)' % self.name  
  
        # When this person is created, he/she  
        # adds to the population  
        Person.population += 1  
  
    def \_\_del\_\_(self):  
        '''I am dying.'''  
        print '%s says bye.' % self.name  
  
        Person.population -= 1  
  
        if Person.population == 0:  
            print 'I am the last one.'  
        else:  
            print 'There are still %d people left.' % Person.population  
  
    def sayHi(self):  
        '''Greeting by the person.  
  
        Really, that's all it does.'''  
        print 'Hi, my name is %s.' % self.name  
  
    def howMany(self):  
        '''Prints the current population.'''  
        if Person.population == 1:  
            print 'I am the only person here.'  
        else:  
            print 'We have %d persons here.' % Person.population  
  
swaroop = Person('Swaroop')  
swaroop.sayHi()  
swaroop.howMany()  
  
kalam = Person('Abdul Kalam')  
kalam.sayHi()  
kalam.howMany()  
  
swaroop.sayHi()  
swaroop.howMany()

## 输出

$ python objvar.py  
(Initializing Swaroop)  
Hi, my name is Swaroop.  
I am the only person here.  
(Initializing Abdul Kalam)  
Hi, my name is Abdul Kalam.  
We have 2 persons here.  
Hi, my name is Swaroop.  
We have 2 persons here.  
Abdul Kalam says bye.  
There are still 1 people left.  
Swaroop says bye.  
I am the last one.

使用继承

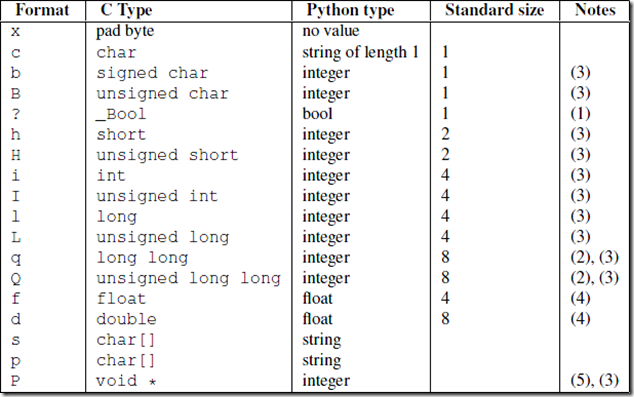
#!/usr/bin/python  
# Filename: inherit.py  
  
class SchoolMember:  
    '''Represents any school member.'''  
    def \_\_init\_\_(self, name, age):  
        self.name = name  
        self.age = age  
        print '(Initialized SchoolMember: %s)' % self.name  
  
    def tell(self):  
        '''Tell my details.'''  
        print 'Name:"%s" Age:"%s"' % (self.name, self.age),  
  
class Teacher(SchoolMember):  
    '''Represents a teacher.'''  
    def \_\_init\_\_(self, name, age, salary):  
        SchoolMember.\_\_init\_\_(self, name, age)  
        self.salary = salary  
        print '(Initialized Teacher: %s)' % self.name  
  
    def tell(self):  
        SchoolMember.tell(self)  
        print 'Salary: "%d"' % self.salary  
  
class Student(SchoolMember):  
    '''Represents a student.'''  
    def \_\_init\_\_(self, name, age, marks):  
        SchoolMember.\_\_init\_\_(self, name, age)  
        self.marks = marks  
        print '(Initialized Student: %s)' % self.name  
  
    def tell(self):  
        SchoolMember.tell(self)  
        print 'Marks: "%d"' % self.marks  
  
t = Teacher('Mrs. Shrividya', 40, 30000)  
s = Student('Swaroop', 22, 75)  
  
print # prints a blank line  
  
members = [t, s]  
for member in members:  
    member.tell() # works for both Teachers and Students

## 输出

$ python inherit.py  
(Initialized SchoolMember: Mrs. Shrividya)  
(Initialized Teacher: Mrs. Shrividya)  
(Initialized SchoolMember: Swaroop)  
(Initialized Student: Swaroop)  
  
Name:"Mrs. Shrividya" Age:"40" Salary: "30000"  
Name:"Swaroop" Age:"22" Marks: "75"

python的类方法和类的静态方法，其实就是一个用@classmethod和@staticmethod修饰的类中的函数。其中类方法要在定义的时候指出在调用它时隐式赋给他的第一个参数，这个参数一般情况下是类本身(cls)，这个有点像类的实例方法中代表类实例本身的self。类的静态方法在定义的时候就不用定义这个参数，因为他是静态方法不属于任何一个类实例。  
  
========================================================================  
类方法  
classmethod( function)   
  
Return a class method for function.   
A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:   
为一个函数返回一个类方法  
类方法隐式的接受类作为它的第一个参数,就像对象方法隐式的接受当前对象作为第一个参数一样.这样定义一个类方法:  
  
class C:  
    @classmethod  
    def f(cls, arg1, arg2, ...): ...  
  
The @classmethod form is a function decorator - see the description of function definitions in chapter 7 of the Python Reference Manual for details.   
@classmethod是修饰方法的写法-请查阅Python手册第七章里面关于函数定义的描述.  
  
It can be called either on the class (such as C.f()) or on an instance (such as C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.   
它可以通过类调用(比如C.f()),也可以通过对象实例调用(比如C().f()).真正发挥作用的是类而不是类的实例对象.如果一个类方法被一个子类调用,这个子类的对象实例会作为第一个参数传给类方法.  
  
Class methods are different than C++ or Java static methods. If you want those, see staticmethod() in this section. New in version 2.2. Changed in version 2.4: Function decorator syntax added.   
Python的类方法不同于C++或Java的静态方法.如果你想要的是那样的静态方法,请查看staticmethod()方法,  
  
=====================================================================  
类的静态方法  
  
staticmethod(function)  
Return a static method for function.  
为一个函数返回一个静态方法  
  
A static method does not receive an implicit first argument. To declare a static method, use this idiom:  
一个静态方法不接受一个隐式的第一个参数，要声明一个静态方法，用下面的方式：  
class C:  
    @staticmethod  
    def f(arg1, arg2, ...): ...  
      
The @staticmethod form is a function decorator – see the description of function definitions in Function definitions for details.  
@staticmethod是一个函数修饰器。  
  
It can be called either on the class (such as C.f()) or on an instance (such as C().f()). The instance is ignored except for its class.  
可以通过类(C.f())或者类实例(C().f())来调用。通过类实例来调用起作用的其实是类，而不是类实例。  
  
Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see classmethod() in this section.  
python的静态方法和java与c++中的很像。一些更高级的概念看classmethod()这一节。  
  
For more information on static methods, consult the documentation on the standard type hierarchy in The standard type hierarchy.  
关于静态函数更多的信息，看The standard type hierarchy中标准类型继承的文档。

定义format可以参照官方api提供的对照表：

[](http://images.cnblogs.com/cnblogs_com/coser/201112/201112171613441943.png)

在 Python 中，方法不止是类实例所独有：其它类型的对象也可有方法。例如，链表对象有 append，insert，remove，sort 等等方法