1 Python 的变量

- 字面意义的常量。 如同 5、 1.23、 9.25e-3 这样的数，或者如同'This is a string'、 "It's a string!"这样的字符串。
- 数。整数、长整数、浮点数和复数。
- 字符串。
  - 单引号。
  - 双引号。
  - 三引号。
  - 转义符。
  - 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"。
  - Unicode 字符串。
  - 字符串连接。例如，'What\'s' 'your name?'会被自动转为"What's your name?"。
- 变量。
- 标识符命名。只能是下划线，字母，数字。
- 数据类型。
  - 数字
  - 字符串
  - 类
- 对象。

- 变量只需要直接赋值，不需要再声明类型。变量前面也不需要加$符号

- 语句后面可以不用加分号，也可以加。

2 运算符与表达式

2.1 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。
		返回x是否小于y。所有比较运算符返回1	5 < 3返回0（即False）而3 < 5返回

<	小于	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。	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 y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。

2.2 运算符优先级

略。建议使用括号来处理想要的优先级顺序。

3 控制流

3.1 If 语句

- if...:
- if...: else...:
- if...: elif...: else...:

```
#!/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
```

3.2 while 语句

两种形式：

- while...:
- while...: else... （可选的 else 从句）

```
#!/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'
```

3.3 for 语句

- for..in...:
- for...in...:... else...

```
#!/usr/bin/python
# Filename: for.py

for i in range(1, 5):
    print i
else:
    print 'The for loop is over'
```

3.4 break 语句

用来退出循环。同 C 语言。

```
#!/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'
```

3.5 continue 语句

用来跳出本次循环，直接进入下次循环。同 C 语言，

```
#!/usr/bin/python
# Filename: continue.py

while True:
    s = raw_input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        continue
    print 'Input is of sufficient length'
    # Do other kinds of processing here...
```

4 函数

4.1 函数定义

def func_name(parameter…) :

xxx

函数通过def关键字定义。def关键字后跟一个函数的 标识符 名称，然后跟一对圆括号。圆括号之中可以包括一些变量名，该行以冒号结尾。接下来是一块语句，它们是函数体。下面这个例子将说明这事实上是十分简单的：

定义函数

例7.1 定义函数

```
#!/usr/bin/python
# Filename: function1.py

def sayHello():
    print 'Hello World!' # block belonging to the
function

sayHello() # call the function
```

4.2 函数形式参数

def func_name(a,b,c…) :

xxx

例7.2 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
```

4.3 局部变量

即函数内定义的变量的作用域只在函数内部。

4.4 全局变量

global 关键字声明

```
#!/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
```

4.5 默认参数值

定义函数时可以指定参数的默认值

例7.5 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)
```

（源文件：code/func_default.py）

输出

```
$ python func_default.py
Hello
WorldWorldWorldWorldWorld
```

4.6 关键参数

可以通过在调用函数时，指定参数名字的方式来给某几个参数赋值。这些参数被称为关键参数。

例7.6 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)
```

4.7 Return 语句

可以用 return 语句来退出函数，以及返回值

```
#!/usr/bin/python
# Filename: func_return.py

def maximum(x, y):
    if x > y:
        return x
    else:
        return y

print maximum(2, 3)
```

4.8 Pass 语句

Pass 语句在 python 中就是一个空语句

```
def someFunction():
    pass
```

4.9 DocStrings

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。

其实就是打印一些说明信息。

例7.8 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.
```

5 模块

5.1 文本模块

- 模块就是定义了很多函数和变量的一个文件的集合
- 模块的文件名必须以 py 结尾
- 导入模块：import module. 当导入模块时，模块的主块语句将被执行（一次）
- 访问模块变量名：module.variable

5.2 字节编译模块

- 以.pyc 结尾
- 目的是导入模块更快
- 字节编译的文件与 Python 变换程序的中间状态有关

5.3 From...impart 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。这对于所有模块都适用。一般说来，应该避免使用 from..import 而使用 import 语 句，因为这样可以使你的程序更加易读，也可以避免名称的冲突

5.4 模块__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。

例8.2 使用模块的__name__

```
#!/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'
```

5.5 制造自己的模块

创建你自己的模块是十分简单的，你一直在这样做！每个Python程序也是一个模块。你已经确保它具有.py扩展名了。下面这个例子将会使它更加清晰。

创建你自己的模块

例8.3 如何创建你自己的模块

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
    print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

5.6 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。

6 数据结构

6.1 列表 list

List 属于一个类。而 python 中类的概念与 C++其实是很类似的。

```
#!/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
```

6.2 元组

元组就是不可改变元素值的列表，用圆括号来定义。

```
#!/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]
```

给**Perl**程序员的注释

列表之中的列表不会失去它的身份，即列表不会像Perl中那样被打散。同样元组中的元组，或列表中的元组，或元组中的列表等等都是如此。只要是Python，它们就只是使用另一个对象存储的对象。

元组与打印语句

例9.3 使用元组输出

```
#!/usr/bin/python
# Filename: print_tuple.py

age = 22
name = 'Swaroop'

print '%s is %d years old' % (name, age)
print 'Why is %s playing with that python?' % name
```

（源文件：code/print_tuple.py）

输出

```
$ python print_tuple.py
Swaroop is 22 years old
Why is Swaroop playing with that python?
```

6.3 字典

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。

记住字典中的键/值对是没有顺序的。如果你想要一个特定的顺序，那么你应该在使用前自己对它们排序。 字典是 dict 类的实例/对象

```
#!/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']
```

6.4 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

[:] -- 切片操作符

```
#!/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[:]
```

6.5 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。

一般说来，你不需要担心这个，只是在参考上有些细微的效果需要你注意。这会通过下面这个例子加以说明

```
#!/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
```

如果你想要复制一个列表或者类似的序列或者其他 复杂的对象(不是如整数那样的简单 对象)，那么你必须使用切片操作符来取得拷贝

如果你只是想要使 用另一个变量名，两个名称都 参考 同一个对象，那么如果你不小心的话，可能会引来各种麻烦

6.6 更多字符串的内容

字符串也是 str 类。有很多可用的方法。

7 面向对象的编程

- Python 中，任何类型都是类。比如 int 是一个类
- 类的属性为域和方法
- 域分为实例变量和类变量。

7.1 Self

类方法与普通函数的区别就是他的第一个参数一定是一个 self。Self 指的是对象本身。

7.2 类

```
#!/usr/bin/python
# Filename: simplestclass.py

class Person:
    pass # An empty block

p = Person()
print p
```

7.3 对象的方法

例11.2 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()
```

这里我们看到了 self 的用法。注意 sayHi 方法没有任何参数，但仍然在函数定义时有 self。

7.4 __init__方法

在 Python 的类中有很多方法的名字有特殊的重要意义。现在我们将学习__init__方法的意义。__init__方法在类的一个对象被建立时，马上运行。这个方法可以用来对你的对象做一些你希望的 初始 化 。注意，这个名称的开始和结尾都是双下划线。

```
#!/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()
```

注意。self.name 这一句即定义了一个新的实例变量。

7.5 类与对象的变量

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。

```
#!/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
```

7.6 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,
```

```
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
```

- 如果在继承元组中列了一个以上的类，那么它就被称作 多重继承
- Python 不会自动调用基本类的 constructor，你得亲自专门调用它
- Python 总是首先查找对应类型的方法。如果它不能在导出类中找到对应的方法，它才开始到基本类中逐个查找

8 输入输出

raw_input 和 print 语句可以实现从标准输入输出获取打印字符串。

8.1 文件

File 类的 read，readline，write 等方法可以恰当的读写文件。

8.2 存储器

Python 的标准模块：pickle，可以在文件中存储任何的 python 对象。并且可以把它完整的取出来。

- Import A as B: 可以给模块取别名。后面可以直接使用 B 代替 A

9 异常

9.1 Try..exept 可以用来处理异常

```
try:
    xxx
except xxx:
    xxx
except:
    xxx
```

9.2 引发异常

你可以使用 raise 语句 引发 异常。你还得指明错误/异常的名称和伴随异常 触发的 异常对象。你可以引发的错误或异常应该分别是一个 Error 或 Exception 类的直接或间接导出类。

9.3 try...finally

假如你在读一个文件的时候，希望在无论异常发生与否的情况下都关闭文件，该怎么做呢？这可以使用 finally 块来完成。注意，在一个 try 块下，你可以同时使用 except 从句和 finally 块。如果你要同时使用它们的话，需要把一个嵌入另外一个

try:
    xxx
finally:
    xxx

10 python 标准库

10.1 sys 模块

- sys.argv 命令行参数
- sys.version python 版本安装信息
- sys.version_info
- sys.stdin, sys.stdout, sys.stderr 标准输入，输出，错误

10.2 OS 模块

这个模块包含普遍的操作系统功能。如果你希望你的程序能够与平台无关的话，这个模块是尤为重要的。即它允许一个程序在编写后不需要任何改动，也不会发生任何问题，就可以在Linux和Windows下运行。一个例子就是使用os.sep可以取代操作系统特定的路径分割符。

下面列出了一些在os模块中比较有用的部分。它们中的大多数都简单明了。

- os.name字符串指示你正在使用的平台。比如对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'。
- os.getcwd()函数得到当前工作目录，即当前Python脚本工作的目录路径。
- os.getenv()和os.putenv()函数分别用来读取和设置环境变量。
- os.listdir()返回指定目录下的所有文件和目录名。
- os.remove()函数用来删除一个文件。
- os.system()函数用来运行shell命令。
- os.linesep字符串给出当前平台使用的行终止符。例如，Windows使用'\r\n'，Linux使用'\n'而Mac使用'\r'。
- os.path.split()函数返回一个路径的目录名和文件名。

```
>>> os.path.split('/home/swaroop/byte/code/poem.txt')
('/home/swaroop/byte/code', 'poem.txt')
```

- os.path.isfile()和os.path.isdir()函数分别检验给出的路径是一个文件还是目录。类似地，os.path.existe()函数用来检验给出的路径是否真地存在。

你可以利用Python标准文档去探索更多有关这些函数和变量的详细知识。你也可以使用help(sys)等等。

11 更多 python 类容

11.1 特殊方法

有一些特殊的方法用来模仿某个对象，你可以自己实现它

表15.1 一些特殊的方法

名称	说明
__init__(self,...)	这个方法在新建对象恰好要被返回使用之前被调用。
__del__(self)	恰好在对象要被删除之前调用。
__str__(self)	在我们对对象使用print语句或是使用str()的时候调用。
__lt__(self,other)	当使用 小于 运算符（<）的时候调用。类似地，对于所有的运算符（+，>等等）都有特殊的方法。
__getitem__(self,key)	使用x[key]索引操作符的时候调用。
__len__(self)	对序列对象使用内建的len()函数的时候调用。

11.2 单语句块

在 if 语句后面，最好是用缩进的方式来预留一个语句块，这样方便以后添加语句

11.3 列表综合

通过列表综合，可以从一个已有的列表导出一个新的列表。例如，你有一个数的列表，而你想要得到一个对应的列表，使其中所有大于2的数都是原来的2倍。对于这种应用，列表综合是最理想的方法。

使用列表综合

例15.1 使用列表综合

```
#!/usr/bin/python
# Filename: list_comprehension.py

listone = [2, 3, 4]
listtwo = [2*i for i in listone if i > 2]
print listtwo
```

（源文件：code/list_comprehension.py）

输出

```
$ python list_comprehension.py
[6, 8]
```

它如何工作

这里我们为满足条件（if i > 2）的数指定了一个操作（2*i），从而导出一个新的列表。注意原来的列表并没有发生变化。在很多时候，我们都是使用循环来处理列表中的每一个元素，而使用列表综合可以用一种更加精确、简洁、清楚的方法完成相同的工作。

11.4 在函数中接收元组和列表

其实是可变参数。

当要使函数接收元组或字典形式的参数的时候，有一种特殊的方法，它分别使用*和**前缀。这种方法在函数需要获取可变数量的参数的时候特别有用。

```
>>> def powersum(power, *args):
...         '''Return the sum of each argument raised to
specified power.'''
...     total = 0
...     for i in args:
...         total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25

>>> powersum(2, 10)
100
```

由于在args变量前有*前缀，所有多余的函数参数都会作为一个元组存储在args中。如果使用的是**前缀，多余的参数则会被认为是一个字典的键/值对。

11.5 Lambda 语句

lambda语句被用来创建新的函数对象，并且在运行时返回它们。

例15.2 使用lambda形式

```
#!/usr/bin/python
# Filename: lambda.py

def make_repeater(n):
    return lambda s: s*n

twice = make_repeater(2)

print twice('word')
print twice(5)
```

（源文件：code/lambda.py）

输出

```
$ python lambda.py
wordword
10
```

它如何工作

这里，我们使用了make_repeater函数在运行时创建新的函数对象，并且返回它。lambda语句用来创建函数对象。本质上，lambda需要一个参数，后面仅跟单个表达式作为函数体，而表达式的值被这个新建的函数返回。注意，即便是print语句也不能用在lambda形式中，只能使用表达式。

11.6 exec 和 eval 语句

exec语句用来执行储存在字符串或文件中的Python语句。例如，我们可以在运行时生成一个包含Python代码的字符串，然后使用exec语句执行这些语句。下面是一个简单的例子。

```
>>> exec 'print "Hello World"'
Hello World
```

eval语句用来计算存储在字符串中的有效Python表达式。下面是一个简单的例子。

```
>>> eval('2*3')
6
```

11.7 assert 语句

assert语句

assert语句用来声明某个条件是真的。例如，如果你非常确信某个你使用的列表中至少有一个元素，而你想要检验这一点，并且在它非真的时候引发一个错误，那么assert语句是应用在这种情形下的理想语句。当assert语句失败的时候，会引发一个AssertionError。

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AssertionError
```

11.8 repr 函数

repr函数

repr函数用来取得对象的规范字符串表示。反引号（也称转换符）可以完成相同的功能。注意，在大多数时候有eval(repr(object)) == object。

```
>>> i = []
>>> i.append('item')
>>> `i`
"['item']"
>>> repr(i)
"['item']"
```

基本上，repr函数和反引号用来获取对象的可打印的表示形式。你可以通过定义类的__repr__方法来控制你的对象在被repr函数调用的时候返回的内容。
