**Python的特点**

　　1. **简单**

　　　　Python是一种代表简单思想的语言。

　　2. **易学**

　　　　Python有极其简单的语法。

　　3. **免费、开源**

　　　　Python是FLOSS（自由/开放源码软件）之一。

　　4. **高层语言**

　　　　使用Python编写程序时无需考虑如何管理程序使用的内存一类的底层细节。

　　5. **可移植性**

　　　　Python已被移植到很多平台，这些平台包括Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、

　　BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、

　　Windows CE甚至还有PocketPC。

　　6. **解释性**

　　　　可以直接从源代码运行。在计算机内部，python解释器把源代码转换为字节码的中间形式，然后再把它翻译成计算机使用的机器语言。

　　7. **面向对象**

　　　　Python既支持面向过程编程也支持面向对象编程。

　　8. **可扩展性**

　　　　部分程序可以使用其他语言编写，如c/c++。

　　9. **可嵌入型**

　　　　可以把Python嵌入到c/c++程序中，从而提供脚本功能。

　　10. **丰富的库**

　　　　Python标准库确实很庞大。它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、

　　电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI（图形用户界面）、Tk和其他与系统有关的操作。

---------------分割线------------------------以下是Python的基本语法---------------------------------------------------------

**一、基本概念**

**1. python中数有四种类型：**整数、长整数、浮点数和复数。

* 整数， 如 1
* 长整数 是比较大的整数
* 浮点数 如 1.23、3E-2
* 复数 如 1 + 2j、 1.1 + 2.2j

**2. 字符串**（字符的序列）

* python中单引号和双引号使用完全相同。
* 使用三引号('''或""")可以指定一个多行字符串。
* 转义符 '\'
* 自然字符串， 通过在字符串前加r或R。 如 r"this is a line with \n" 则\n会显示，并不是换行。
* python允许处理unicode字符串，加前缀u或U， 如 u"this is an unicode string"。
* 字符串是不可变的。
* 按字面意义级联字符串，如"this " "is " "string"会被自动转换为this is string。

**3. 标识符的命名**

* 第一个字符必须是字母表中字母或下划线'\_'。
* 标识符的其他的部分有字母、数字和下划线组成。
* 标识符对大小写敏感。

**4. 对象**

　　　　python程序中用到的任何“东西”都成为“对象”。

**5. 逻辑行和物理行**

　　　　物理行是我们在编写程序时看到的，逻辑行则是python看到的。

　　　　python中分号;标识一个逻辑行的结束，但是实际中一般每个物理行只写一个逻辑行，可以避免使用分号。

　　　　多个物理行中可以写一个逻辑行，如下：

s = "peter is \

writing this article"

　　　　上面\的使用被称为‘明确的行连接’， 又如：

print \  
"peter"

**6. 缩进**

　　　　空白在python是非常重要的，行首的空白是最重要的，又称为缩进。行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而决定语句

　　分组。这意味着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。

　　注意：不要混合使用空格和制表符来缩进，因为在跨越不同的平台时无法正常工作。

**二、运算符与表达式**

**1. 运算符与其用法**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 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. 运算符优先级（从低到高）**

|  |  |
| --- | --- |
| **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 |

**3. python 控制台输出** 使用print

print "abc" 　　#打印abc并换行  
print "abc%s" % "d" 　　#打印abcd  
print "abc%sef%s" % ("d", "g") 　　#打印abcdefg

**三、控制流**

**1. if 语句**

[复制代码](javascript:void(0);)

i = 10  
n = int(raw\_input("enter a number:"))  
  
if n == i:  
 print "equal"  
elif n < i:  
 print "lower"  
else:  
 print "higher"

[复制代码](javascript:void(0);)

**2. while语句**

while True:  
 pass  
else:  
 pass  
#else语句可选，当while为False时，else语句被执行。 pass是空语句。

**3. for 循环 for..in**

for i in range(0, 5):  
 print i  
else:  
 pass  
# 打印0到4

　　注：当for循环结束后执行else语句；

　　　　range(a, b)返回一个序列，从a开始到b为止，但不包括b，range默认步长为1，可以指定步长，range(0,10,2)；

**4. break语句**

　　　　终止循环语句，如果从for或while中终止，任何对应循环的else将**不**执行。

**5. continue语句**

　　　　continue语句用来调过当前循环的剩余语句，然后继续下一轮循环。

**四、函数**

　　函数通过**def**定义。def关键字后跟函数的标识符名称，然后跟一对圆括号，括号之内可以包含一些变量名，该行以冒号结尾；接下来是一块语句，即函数体。

def sumOf(a, b):  
 return a + b

**1. 函数形参**

　　　　函数中的参数名称为‘形参’，调用函数时传递的值为‘实参’

**2. 局部变量**

　　　　在函数内定义的变量与函数外具有相同名称的其他变量没有任何关系，即变量名称对于函数来说是局部的。这称为变量的作用域。

　　　　global语句， 为定义在函数外的变量赋值时使用global语句。

[复制代码](javascript:void(0);)

def func():  
 global x  
 print "x is ", x  
 x = 1  
  
x = 3  
func()  
print x  
  
#3  
#1

[复制代码](javascript:void(0);)

**3. 默认参数**

　　　　通过使用默认参数可以使函数的一些参数是‘可选的’。

def say(msg, times = 1):  
 print msg \* times  
  
say("peter")  
say("peter", 3)

　　　　注意：只有在形参表末尾的那些参数可以有默认参数值，即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，只是因为赋给形参的值是根据位置而赋值的。

**4. 关键参数**

　　　　如果某个函数有很多参数，而现在只想指定其中的部分，那么可以通过命名为这些参数赋值（称为‘关键参数’）。

　　　　优点：不必担心参数的顺序，使函数变的更加简单；假设其他参数都有默认值，可以只给我们想要的那些参数赋值。

[复制代码](javascript:void(0);)

def func(a, b=2, c=3):  
 print "a is %s, b is %s, c is %s" % (a, b, c)  
  
func(1) #a is 1, b is 2, c is 3  
func(1, 5) #a is 1, b is 5, c is 3  
func(1, c = 10) #a is 1, b is 2, c is 10  
func(c = 20, a = 30) #a is 30, b is 2, c is 20

[复制代码](javascript:void(0);)

**5. return 语句**

　　　　return语句用来从一个函数返回，即跳出函数。可从函数返回一个值。

　　　　没有返回值的return语句等价于return None。None表示没有任何东西的特殊类型。

**6. DocStrings** (文档字符串)

[复制代码](javascript:void(0);)

def func():  
 '''This is self-defined function  
  
Do nothing'''  
 pass  
  
print func.\_\_doc\_\_  
  
#This is self-defined function  
#  
#Do nothing

[复制代码](javascript:void(0);)

**五、模块**

　　模块就是一个包含了所有你定义的函数和变量的文件，模块必须以.py为扩展名。模块可以从其他程序中‘输入’(import)以便利用它的功能。

　　在python程序中导入其他模块使用'import', 所导入的模块必须在sys.path所列的目录中，因为sys.path第一个字符串是空串''即当前目录，所以程序中可导入当前目录的模块。

**1. 字节编译的.pyc文件**

　　　　导入模块比较费时，python做了优化，以便导入模块更快些。一种方法是创建字节编译的文件，这些文件以.pyc为扩展名。

　　　　pyc是一种二进制文件，是py文件经编译后产生的一种byte code，而且是跨平台的（平台无关）字节码，是有python虚拟机执行的，类似于

　　java或.net虚拟机的概念。pyc的内容，是跟python的版本相关的，不同版本编译后的pyc文件是不同的。

**2. from .. import**

　　　　如果想直接使用其他模块的变量或其他，而不加'模块名+.'前缀，可以使用from .. import。

　　　　例如想直接使用sys的argv，from sys import argv 或 from sys import \*

**3. 模块的\_\_name\_\_**

　　　　每个模块都有一个名称，py文件对应模块名默认为py文件名，也可在py文件中为\_\_name\_\_赋值；如果是\_\_name\_\_，说明这个模块被用户

　　单独运行。

**4. dir()函数**

　　　　dir(sys)返回sys模块的名称列表；如果不提供参数，即dir()，则返回当前模块中定义名称列表。

　　　　del -> 删除一个变量/名称，del之后，该变量就不能再使用。

**六、数据结构**

　　python有三种内建的数据结构：列表、元组和字典。

**1. 列表**

　　　　list是处理一组有序项目的数据结构，列表是可变的数据结构。列表的项目包含在方括号[]中，eg: [1, 2, 3]， 空列表[]。判断列表中是否包含某项可以使用in， 比如 l = [1, 2, 3]; print 1 in l; #True；支持索引和切片操作；索引时若超出范围，则IndexError；使用函数len()查看长度；使用del可以删除列表中的项，eg: del l[0] # 如果超出范围，则IndexError

　　　　list函数如下：

***append（value）***　　---向列表尾添加项value

l = [1, 2, 2]  
l.append(3) #[1, 2, 2, 3]

***count(value)***　　---返回列表中值为value的项的个数

l = [1, 2, 2]  
print l.count(2) # 2

***extend(list2)***　　---向列表尾添加列表list2

l = [1, 2, 2]  
l1 = [10, 20]  
l.extend(l1)  
print l #[1, 2, 2, 10, 20]

***index(value, [start, [stop]])***　　---返回列表中第一个出现的值为value的索引，如果没有，则异常 ValueError

l = [1, 2, 2]  
a = 4  
try:  
 print l.index(a)  
except ValueError, ve:  
 print "there is no %d in list" % a

***insert(i, value)***　　---向列表i位置插入项vlaue，如果没有i，则添加到列表尾部

[复制代码](javascript:void(0);)

l = [1, 2, 2]  
  
l.insert(1, 100)  
print l #[1, 100, 2, 2]  
  
l.insert(100, 1000)  
print l #[1, 100, 2, 2, 1000]

[复制代码](javascript:void(0);)

***pop([i])***　　---返回i位置项，并从列表中删除；如果不提供参数，则删除最后一个项；如果提供，但是i超出索引范围，则异常IndexError

[复制代码](javascript:void(0);)

l = [0, 1, 2, 3, 4, 5]  
  
print l.pop() # 5  
print l #[0, 1, 2, 3, 4]  
  
print l.pop(1) #1  
print l #[0, 2, 3, 4]  
  
try:  
 l.pop(100)  
except IndexError, ie:  
 print "index out of range"

[复制代码](javascript:void(0);)

***remove(value)***　　---删除列表中第一次出现的value，如果列表中没有vlaue，则异常ValueError

[复制代码](javascript:void(0);)

l = [1, 2, 3, 1, 2, 3]  
  
l.remove(2)  
print l #[1, 3, 1, 2, 3]  
  
try:  
 l.remove(10)  
except ValueError, ve:  
 print "there is no 10 in list"

[复制代码](javascript:void(0);)

***reverse()***　　---列表反转

l = [1, 2, 3]  
l.reverse()  
print l #[3, 2, 1]

***sort(cmp=None, key=None, reverse=False)***　　---列表排序

【Python Library Reference】  
cmp:cmp specifies a custom comparison function of two arguments (iterable elements) which should return a negative, zero or positive number depending on whether the first argument is considered smaller than, equal to, or larger than the second argument:   
"cmp=lambda x,y: cmp(x.lower(), y.lower())"   
key:key specifies a function of one argument that is used to extract a comparison key from each list element: "key=str.lower"  
reverse:reverse is a boolean value. If set to True, then the list elements are sorted as if each comparison were reversed.In general, the key and reverse conversion processes are much faster than specifying an   
equivalent cmp function. This is because cmp is called multiple times for each list element while key and reverse touch each element only once.

[复制代码](javascript:void(0);)

l5 = [10, 5, 20, 1, 30]  
l5.sort()  
print l5 #[1, 5, 10, 20, 30]  
  
l6 = ["bcd", "abc", "cde", "bbb"]  
l6.sort(cmp = lambda s1, s2: cmp(s1[0],s2[1]))  
print l6 #['abc', 'bbb', 'bcd', 'cde']  
  
l7 = ["bcd", "abc", "cde", "bbb", "faf"]  
l7.sort(key = lambda s: s[1])  
print l7 #['faf', 'abc', 'bbb', 'bcd', 'cde']

**2. 元组**

　　　　tuple和list十分相似，但是tuple是不可变的，即不能修改tuple，元组通过圆括号中用逗号分割的项定义；支持索引和切片操作；可以使用 in

　　查看一个元素是否在tuple中。空元组()；只含有一个元素的元组("a",) #需要加个逗号

　　　　优点：tuple比list速度快；对不需要修改的数据进行‘写保护’，可以是代码更安全

　　　　tuple与list可以相互转换，使用内置的函数list()和tuple()。

[复制代码](javascript:void(0);)

l = [1, 2, 3]  
print l # [1, 2, 3]  
  
t = tuple(l)  
print t # (1, 2, 3)  
  
l1 = list(t)  
print l1 #[1, 2, 3]

[复制代码](javascript:void(0);)

　　　　元组最通常的用法是用在打印语句，如下例：

name = "Peter Zhang"  
age = 25  
print "Name: %s; Age: %d" % (name, age)  
# Name: Peter Zhang; Age: 25

　　　　函数如下：

***count(value)***　　---返回元组中值为value的元素的个数

t = (1, 2, 3, 1, 2, 3)  
  
print t.count(2) # 2

***index(value, [start, [stop]])***　　---返回列表中第一个出现的值为value的索引，如果没有，则异常 ValueError

[复制代码](javascript:void(0);)

t = (1, 2, 3, 1, 2, 3)  
  
print t.index(3) # 2  
try:  
 print t.index(4)  
except ValueError, ve:  
 print "there is no 4 in tuple" # there is no 4 in tuple

[复制代码](javascript:void(0);)

**3. 字典**

　　　　字典由键值对组成，键必须是唯一的；eg: d = {key1:value1, key2:value2}；空字典用{}表示；字典中的键值对是没有顺序的，如果想要

　　一个特定的顺序，那么使用前需要对它们排序；d[key] = value，如果字典中已有key，则为其赋值为value，否则添加新的键值对key/value；使

　　用del d[key] 可以删除键值对；判断字典中是否有某键，可以使用in 或 not in；

[复制代码](javascript:void(0);)

d = {}  
d["1"] = "one"  
d["2"] = "two"  
d["3"] = "three"  
  
del d["3"]  
  
for key, value in d.items():  
 print "%s --> %s" % (key, value)  
#1 --> one  
#2 --> two

[复制代码](javascript:void(0);)

　　　　dict函数如下:

***clear()***　　---删除字典中所有元素

d1 = {"1":"one", "2":"two"}  
d1.clear()  
  
print d1 # {}

***copy()***　　---返回字典的一个副本(浅复制)

d1 = {"1":"one", "2":"two"}  
d2 = d1.copy()  
  
print d2 #{'1': 'one', '2': 'two'}

***dict.fromkeys(seq,val=None)*** ---创建并返回一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值(默认为None)

[复制代码](javascript:void(0);)

l = [1, 2, 3]  
t = (1, 2, 3)  
  
d3 = {}.fromkeys(l)  
print d3 #{1: None, 2: None, 3: None}  
  
d4 = {}.fromkeys(t, "default")  
print d4 #{1: 'default', 2: 'default', 3: 'default'}

[复制代码](javascript:void(0);)

***get(key,[default])***　　---返回字典dict中键key对应值，如果字典中不存在此键，则返回default 的值(default默认值为None)

d5 = {1:"one", 2:"two", 3:"three"}  
  
print d5.get(1) #one  
print d5.get(5) #None  
print d5.get(5, "test") #test

***has\_key(key)***　　---判断字典中是否有键key

d6 = {1:"one", 2:"two", 3:"three"}  
  
print d6.has\_key(1) #True  
print d6.has\_key(5) #False

***items()***　　---返回一个包含字典中(键, 值)对元组的列表

[复制代码](javascript:void(0);)

d7 = {1:"one", 2:"two", 3:"three"}  
  
for item in d7.items():  
 print item  
#(1, 'one')  
#(2, 'two')  
#(3, 'three')  
  
for key, value in d7.items():  
 print "%s -- %s" % (key, value)  
#1 -- one  
#2 -- two  
#3 -- three

[复制代码](javascript:void(0);)

***keys()***　　---返回一个包含字典中所有键的列表

[复制代码](javascript:void(0);)

d8 = {1:"one", 2:"two", 3:"three"}  
  
for key in d8.keys():  
 print key  
#1  
#2  
#3

[复制代码](javascript:void(0);)

***values()***　　---返回一个包含字典中所有值的列表

[复制代码](javascript:void(0);)

d8 = {1:"one", 2:"two", 3:"three"}  
  
for value in d8.values():  
 print value  
#one  
#two  
#three

[复制代码](javascript:void(0);)

***pop(key, [default])***　　---若字典中key键存在，删除并返回dict[key]，若不存在，且未给出default值，引发KeyError异常

[复制代码](javascript:void(0);)

d9 = {1:"one", 2:"two", 3:"three"}  
print d9.pop(1) #one  
print d9 #{2: 'two', 3: 'three'}  
print d9.pop(5, None) #None  
try:  
 d9.pop(5) # raise KeyError  
except KeyError, ke:  
 print "KeyError:", ke #KeyError:5

[复制代码](javascript:void(0);)

***popitem()***　　---删除任意键值对，并返回该键值对，如果字典为空，则产生异常KeyError

d10 = {1:"one", 2:"two", 3:"three"}  
  
print d10.popitem() #(1, 'one')  
print d10 #{2: 'two', 3: 'three'}

***setdefault(key,[default])***　　---若字典中有key，则返回vlaue值，若没有key，则加上该key，值为default，默认None

[复制代码](javascript:void(0);)

d = {1:"one", 2:"two", 3:"three"}  
  
print d.setdefault(1) #one  
print d.setdefault(5) #None  
print d #{1: 'one', 2: 'two', 3: 'three', 5: None}  
print d.setdefault(6, "six") #six  
print d #{1: 'one', 2: 'two', 3: 'three', 5: None, 6: 'six'}

[复制代码](javascript:void(0);)

***update(dict2)***　　---把dict2的元素加入到dict中去，键字重复时会覆盖dict中的键值

d = {1:"one", 2:"two", 3:"three"}  
d2 = {1:"first", 4:"forth"}  
  
d.update(d2)  
print d #{1: 'first', 2: 'two', 3: 'three', 4: 'forth'}

***viewitems()***---返回一个view对象，（key, value）pair的列表，类似于视图。优点是，如果字典发生变化，view会同步发生变化。在

　　迭代过程中，字典不允许改变，否则会报异常

d = {1:"one", 2:"two", 3:"three"}  
for key, value in d.viewitems():  
 print "%s - %s" % (key, value)  
#1 - one  
#2 - two  
#3 - three

***viewkeys()***　　---返回一个view对象，key的列表

d = {1:"one", 2:"two", 3:"three"}  
for key in d.viewkeys():  
 print key  
#1  
#2  
#3

***viewvalues()***　　---返回一个view对象，value的列表

d = {1:"one", 2:"two", 3:"three"}  
for value in d.viewvalues():  
 print value  
#one  
#two  
#three

**4. 序列**

　　　　序列类型是指容器内的元素从0开始的索引顺序访问，一次可以访问一个或者多个元素；列表、元组和字符串都是序列；序列的两个主要特点是

　　索引操作符和切片操作符；索引可以得到特定元素；切片可以得到部分序列；

[复制代码](javascript:void(0);)

1 numbers = ["zero", "one", "two", "three", "four"]  
 2   
 3 print numbers[1] # one  
 4 print numbers[-1] # four  
 5 #print numbers[5] # raise IndexError  
 6   
 7 print numbers[:] # ['zero', 'one', 'two', 'three', 'four']  
 8 print numbers[3:] # ['three', 'four']  
 9 print numbers[:2] # ['zero', 'one']  
10 print numbers[2:4] # ['two', 'three']  
11 print numbers[1:-1] # ['one', 'two', 'three']

[复制代码](javascript:void(0);)

　　　　切片操作符中的第一个数（冒号之前）表示切片开始的位置，第二个数（冒号之后）表示切片到哪里结束。 如果不指定第一个数，Python就从

　　序列首开始。如果没有指定第二个数，则Python会停止在序列尾。 注意，返回的序列从开始位置 开始 ，刚好在结束位置之前 结束。即开始位置是

　　包含在序列切片中的，而结束位置被排斥在切片外。 可以用负数做切片。负数用在从序列尾开始计算的位置。  
　　**5. 绑定**

　　　　当创建一个对象并给它赋一个变量的时候，这个变量仅仅“指向”那个对象，而不是表示这个对象本身！ 也就是说，变量名指向计算机中存储那个

　　对象的内存，这被称作名称到对象的**绑定**。

　　　　如果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），那么必须使用切片操作符来取得拷贝。

**七、面向对象编程**

　　python支持面向对象编程；类和对象是面向对象编程的两个主要方面，类创建一个新的类型，对象是这个类的实例。

　　对象可以使用普通的属于对象的变量存储数据，属于对象或类的变量被称为**域**；对象也可以使用属于类的函数，这样的函数称为类的方法；域和方法可以合称为类的属性。

　　域有两种类型--属于实例的或属于类本身；它们分别被称为实例变量和类变量。

　　类使用关键字**class**创建，类的域和方法被列在一个缩进块中。

　　类的方法必须有一个额外的第一个参数，但是在调用时不为这个参数赋值，这个特殊变量指对象本身，按照惯例它的名称是**self**，类似C#中的this。

class Animal:  
 pass #empty block

　　\_\_init\_\_方法　　在类的一个对象被创建时调用该方法；相当于c++中的构造函数。

　　\_\_del\_\_方法　　在类的对象被销毁时调用该方法；相当于c++中的析构函数。在使用del删除一个对象时也就调用\_\_del\_\_方法。

　　Python中所有的类成员(包括数据成员)都是public的；只有一个例外，如果使用的数据成员以双下划线为前缀，则为私有变量。

[复制代码](javascript:void(0);)

class Person:  
 Count = 0  
 def \_\_init\_\_(self, name, age):  
 Person.Count += 1  
 self.name = name  
 self.\_\_age = age  
  
p = Person("peter", 25)  
p1 = Person("john", 20)  
  
print Person.Count #2  
print p.name #peter  
print p.\_\_age #AttributeError: Person instance has no attribute '\_\_age'

[复制代码](javascript:void(0);)

　　继承：为了使用继承，基类的名称作为一个元组跟在类名称的后面；python支持多重继承。下面是一个关于继承的例子：

[复制代码](javascript:void(0);)

1 class SchoolMember:  
 2 '''Represent any school member.'''  
 3 def \_\_init\_\_(self, name, age):  
 4 self.name = name  
 5 self.age = age  
 6 print "Initializing a school member."  
 7   
 8 def tell(self):  
 9 '''Tell my details'''  
10 print "Name: %s, Age: %s, " % (self.name, self.age),  
11   
12 class Teacher(SchoolMember):  
13 '''Represent a teacher.'''  
14 def \_\_init\_\_(self, name, age, salary):  
15 SchoolMember.\_\_init\_\_(self, name, age)  
16 self.salary = salary  
17 print "Initializing a teacher"  
18   
19 def tell(self):  
20 SchoolMember.tell(self)  
21 print "Salary: %d" % self.salary  
22   
23 class Student(SchoolMember):  
24 '''Represent a student.'''  
25 def \_\_init\_\_(self, name, age, marks):  
26 SchoolMember.\_\_init\_\_(self, name, age)  
27 self.marks = marks  
28 print "Initializing a student"  
29   
30 def tell(self):  
31 SchoolMember.tell(self)  
32 print "Marks: %d" % self.marks  
33   
34 print SchoolMember.\_\_doc\_\_  
35 print Teacher.\_\_doc\_\_  
36 print Student.\_\_doc\_\_  
37   
38 t = Teacher("Mr. Li", 30, 9000)  
39 s = Student("Peter", 25, 90)  
40   
41 members = [t, s]  
42   
43 for m in members:  
44 m.tell()

[复制代码](javascript:void(0);)

　　程序输出如下：

[复制代码](javascript:void(0);)

Represent any school member.  
Represent a teacher.  
Represent a student.  
Initializing a school member.  
Initializing a teacher  
Initializing a school member.  
Initializing a student  
Name: Mr. Li, Age: 30, Salary: 9000  
Name: Peter, Age: 25, Marks: 90

[复制代码](javascript:void(0);)

**八、输入/输出**

　　程序与用户的交互需要使用输入/输出，主要包括控制台和文件；对于控制台可以使用raw\_input和print，也可使用str类。raw\_input(xxx)输入xxx然后读取用户的输入并返回。

**1. 文件输入/输出**

　　　　可以使用file类打开一个文件，使用file的read、readline和write来恰当的读写文件。对文件读写能力取决于打开文件时使用的模式，常用模式

　　有读模式("r")、写模式("w")、追加模式("a")，文件操作之后需要调用close方法来关闭文件。

[复制代码](javascript:void(0);)

1 test = '''\  
 2 This is a program about file I/O.  
 3   
 4 Author: Peter Zhange  
 5 Date: 2011/12/25  
 6 '''  
 7   
 8 f = file("test.txt", "w") # open for writing, the file will be created if the file doesn't exist  
 9 f.write(test) # write text to file  
10 f.close() # close the file  
11   
12 f = file("test.txt") # if no mode is specified, the default mode is readonly.  
13   
14 while True:  
15 line = f.readline()  
16 if len(line) == 0: # zero length indicates the EOF of the file  
17 break  
18 print line,  
19   
20 f.close()

[复制代码](javascript:void(0);)

**2. 存储器**

　　　　python提供一个标准的模块，成为pickle，使用它可以在一个文件中存储任何python对象，之后可以完整的取出来，这被称为持久地存储对象；还有另外一个模块成为cPickle，它的功能和pickle完全一样，只不过它是用c写的，要比pickle速度快(大约快1000倍)。

[复制代码](javascript:void(0);)

import cPickle  
  
datafile = "data.data"  
  
namelist = ["peter", "john", "king"]  
  
f = file(datafile, "w")  
cPickle.dump(namelist, f)  
f.close()  
  
del namelist  
  
f = file(datafile)  
storednamelist = cPickle.load(f)  
  
print storednamelist  
#['peter', 'john', 'king']

[复制代码](javascript:void(0);)

**九、异常**

　　当程序中出现某些异常的状况时，异常就发生了。python中可以使用try ... except 处理。

[复制代码](javascript:void(0);)

try:  
 print 1/0  
except ZeroDivisionError, e:  
 print e  
except:  
 print "error or exception occurred."  
  
#integer division or modulo by zero

[复制代码](javascript:void(0);)

　　可以让try ... except 关联上一个else，当没有异常时则执行else。

　　我们可以定义自己的异常类，需要继承Error或Exception。

[复制代码](javascript:void(0);)

class ShortInputException(Exception):  
 '''A user-defined exception class'''  
 def \_\_init\_\_(self, length, atleast):  
 Exception.\_\_init\_\_(self)  
 self.length = length  
 self.atleast = atleast  
  
try:  
 s = raw\_input("enter someting-->")  
 if len(s) < 3:  
 raise ShortInputException(len(s), 3)  
except EOFError:  
 print "why you input an EOF?"  
except ShortInputException, ex:  
 print "The lenght of input is %d, was expecting at the least %d" % (ex.length, ex.atleast)  
else:  
 print "no exception"  
#The lenght of input is 1, was expecting at the least 3

[复制代码](javascript:void(0);)

　　try...finally

[复制代码](javascript:void(0);)

try:  
 f = file("test.txt")  
 while True:  
 line = f.readline()  
 if len(line) == 0:  
 break  
 time.sleep(2)  
 print line,  
finally:  
 f.close()  
 print "Cleaning up..."

[复制代码](javascript:void(0);)

**十、Python标准库**

　　Python标准库是随Pthon附带安装的，包含了大量极其有用的模块。

**1. sys模块**sys模块包含系统对应的功能

* sys.argv　　---包含命令行参数，第一个参数是py的文件名
* sys.platform　　---返回平台类型
* sys.exit([status])　　---退出程序，可选的status(范围：0-127)：0表示正常退出，其他表示不正常，可抛异常事件供捕获
* sys.path　　　　---程序中导入模块对应的文件必须放在sys.path包含的目录中，使用sys.path.append添加自己的模块路径
* sys.modules　　---This is a dictionary that maps module names to modules which have already been loaded
* sys.stdin,sys.stdout,sys.stderr　　---包含与标准I/O 流对应的流对象

s = sys.stdin.readline()  
  
sys.stdout.write(s)

**2. os模块**　　该模块包含普遍的操作系统功能

* 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.sep 操作系统特定的路径分割符
* os.path.split()函数返回一个路径的目录名和文件名
* 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
* os.path.existe()函数用来检验给出的路径是否真地存在

**十一、其他**

**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。 |

　　下面的类中定义了上表中的方法：

[复制代码](javascript:void(0);)

class Array:  
 \_\_list = []  
  
 def \_\_init\_\_(self):  
 print "constructor"  
  
 def \_\_del\_\_(self):  
 print "destructor"  
  
 def \_\_str\_\_(self):  
 return "this self-defined array class"  
  
 def \_\_getitem\_\_(self, key):  
 return self.\_\_list[key]  
  
 def \_\_len\_\_(self):  
 return len(self.\_\_list)  
  
 def Add(self, value):  
 self.\_\_list.append(value)  
  
 def Remove(self, index):  
 del self.\_\_list[index]  
  
 def DisplayItems(self):  
 print "show all items----"  
 for item in self.\_\_list:  
 print item  
  
arr = Array() #constructor  
print arr #this self-defined array class  
print len(arr) #0  
arr.Add(1)  
arr.Add(2)  
arr.Add(3)  
print len(arr) #3  
print arr[0] #1  
arr.DisplayItems()  
#show all items----  
#1  
#2  
#3  
arr.Remove(1)  
arr.DisplayItems()  
#show all items----  
#1  
#3  
#destructor

[复制代码](javascript:void(0);)

**2. 综合列表**

　　　　通过列表综合，可以从一个已有的列表导出一个新的列表。

list1 = [1, 2, 3, 4, 5]  
list2 = [i\*2 for i in list1 if i > 3]  
  
print list1 #[1, 2, 3, 4, 5]  
print list2 #[8, 10]

**3. 函数接收元组/列表/字典**

　　　　当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，使用\*和\*\*前缀。该方法在函数需要获取可变数量的参数的时候特别有用。

　　　　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。

[复制代码](javascript:void(0);)

def powersum(power, \*args):  
 total = 0  
 for i in args:  
 total += pow(i, power)  
 return total  
  
print powersum(2, 1, 2, 3) #14

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

def displaydic(\*\*args):  
 for key,value in args.items():  
 print "key:%s;value:%s" % (key, value)  
  
  
displaydic(a="one", b="two", c="three")  
#key:a;value:one  
#key:c;value:three  
#key:b;value:two

[复制代码](javascript:void(0);)

**4. lambda**

　　　　lambda语句被用来创建新的函数对象，并在运行时返回它们。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。

func = lambda s: s \* 3  
print func("peter ") #peter peter peter  
  
func2 = lambda a, b: a \* b  
print func2(2, 3) #6

**5. exec/eval**

　　　　exec语句用来执行储存在字符串或文件中的Python语句；eval语句用来计算存储在字符串中的有效Python表达式。

cmd = "print 'hello world'"  
exec cmd #hello world  
  
expression = "10 \* 2 + 5"  
print eval(expression) #25

**6. assert**

　　　　assert语句用来断言某个条件是真的，并且在它非真的时候引发一个错误--AssertionError。

[复制代码](javascript:void(0);)

flag = True  
  
assert flag == True  
  
try:  
 assert flag == False  
except AssertionError, err:  
 print "failed"  
else:  
 print "pass"

[复制代码](javascript:void(0);)

**7. 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。

　　　　可以通过定义类的\_\_repr\_\_方法来控制对象在被repr函数调用的时候返回的内容。

arr = [1, 2, 3]  
print `arr` #[1, 2, 3]  
print repr(arr) #[1, 2, 3]

**十二、练习**

　　　　实现一个通讯录，主要功能：添加、删除、更新、查询、显示全部联系人。

[复制代码](javascript:void(0);)

1 import cPickle  
 2 import os  
 3 import sys  
 4   
 5 class Contact:  
 6 def \_\_init\_\_(self, name, phone, mail):  
 7 self.name = name  
 8 self.phone = phone  
 9 self.mail = mail  
10   
11 def Update(self, name, phone, mail):  
12 self.name = name  
13 self.phone = phone  
14 self.mail = mail  
15   
16 def display(self):  
17 print "name:%s, phone:%s, mail:%s" % (self.name, self.phone, self.mail)  
18   
19   
20 # begin   
21   
22 # file to store contact data  
23 data = os.getcwd() + os.sep + "contacts.data"  
24   
25 while True:  
26 print "-----------------------------------------------------------------------"  
27 operation = raw\_input("input your operation(add/delete/modify/search/all/exit):")  
28   
29 if operation == "exit":  
30 sys.exit()  
31   
32 if os.path.exists(data):  
33 if os.path.getsize(data) == 0:  
34 contacts = {}  
35 else:  
36 f = file(data)  
37 contacts = cPickle.load(f)  
38 f.close()  
39 else:  
40 contacts = {}  
41   
42 if operation == "add":  
43 flag = False  
44 while True:  
45 name = raw\_input("input name(exit to back choose operation):")  
46 if name == "exit":  
47 flag = True  
48 break  
49 if name in contacts:  
50 print "the name already exists, please input another or input 'exit' to back choose operation"  
51 continue  
52 else:  
53 phone = raw\_input("input phone:")  
54 mail = raw\_input("input mail:")  
55 c = Contact(name, phone, mail)  
56 contacts[name] = c  
57 f = file(data, "w")  
58 cPickle.dump(contacts, f)  
59 f.close()  
60 print "add successfully."  
61 break  
62 elif operation == "delete":  
63 name = raw\_input("input the name that you want to delete:")  
64 if name in contacts:  
65 del contacts[name]  
66 f = file(data, "w")  
67 cPickle.dump(contacts, f)  
68 f.close()  
69 print "delete successfully."  
70 else:  
71 print "there is no person named %s" % name  
72 elif operation == "modify":  
73 while True:  
74 name = raw\_input("input the name which to update or exit to back choose operation:")  
75 if name == "exit":  
76 break  
77 if not name in contacts:  
78 print "there is no person named %s" % name  
79 continue  
80 else:  
81 phone = raw\_input("input phone:")  
82 mail = raw\_input("input mail:")  
83 contacts[name].Update(name, phone, mail)  
84 f = file(data, "w")  
85 cPickle.dump(contacts, f)  
86 f.close()  
87 print "modify successfully."  
88 break  
89 elif operation == "search":  
90 name = raw\_input("input the name which you want to search:")  
91 if name in contacts:  
92 contacts[name].display()  
93 else:  
94 print "there is no person named %s" % name  
95 elif operation == "all":  
96 for name, contact in contacts.items():  
97 contact.display()  
98 else:  
99 print "unknown operation"

[复制代码](javascript:void(0);)