* 数据结构格式：
* （）#字符串
* Dict={…}
* List创建方法

student=[{'name':'Allen','age':80},{'name':'BOB','age':90},{'name':'CARD','age':44}]

* 逗号是分隔符，加号是字符串连接符

print 'abc','def' 输出的是abc def这个两个字符串，中间用空格隔开，

return 'abc','def' 是返回两个值，用元组tuple的形式来传递，

所以记住，字符串连接是用加号

Return 实例的参数

return’（%s，%s）’ % （self.name,self.gender）

所以记住，字符串连接是用加号

* 转义

Print 内容含有双引号的单引号，这个时候，就需要对字符串的某些特殊字符进行“转义”，Python字符串用\进行转义。

要表示字符串 Bob said "I'm OK".

由于 ' 和 " 会引起歧义，因此，我们在它前面插入一个\表示这是一个普通字符，不代表字符串的起始，因此，这个字符串又可以表示为

'Bob said \"I\'m OK\".'

注意：转义字符 \ 不计入字符串的内容中。

常用的转义字符还有：

\n 表示换行

\t 表示一个制表符

\\ 表示 \ 字符本身

r 为前缀表示：（非转义）原始字符串，忽略\等特殊字符的效果

以u或U开头的字符串表示unicode字符串

* 多次转义

如果一个字符串包含很多需要转义的字符，对每一个字符都进行转义会很麻烦。为了避免这种情况，我们可以在字符串前面加个前缀r ，表示这是一个 raw 字符串，里面的字符就不需要转义了。例如：

r'\(~\_~)/ \(~\_~)/'

但是r'...'表示法不能表示多行字符串，也不能表示包含'和 "的字符串（为什么？）

如果要表示多行字符串，可以用'''...'''表示：

'''Line 1

Line 2

Line 3'''

上面这个字符串的表示方法和下面的是完全一样的：

'Line 1\nLine 2\nLine 3'

* Unicode

Unicode字符串除了多了一个 u 之外，与普通字符串没啥区别，转义字符和多行表示法仍然有效：

* 求余数

11 / 4 # ==> 2

令很多初学者惊讶的是，Python的整数除法，即使除不尽，结果仍然是整数，余数直接被扔掉。不过，Python提供了一个求余的运算 % 可以计算余数：

11 % 4 # ==> 3

如果我们要计算 11 / 4 的精确结果，按照“整数和浮点数混合运算的结果是浮点数”的法则，把两个数中的一个变成浮点数再运算就没问题了

Python 3.x已经改进了整数的除法运算，“**/**”除将得到浮点数，“**//**”除才仍是整数：

* 布尔运算

在Python中，布尔类型还可以与其他数据类型做 and、or和not运算，请看下面的代码：

a = True

print a and 'a=T' or 'a=F'

计算结果不是布尔类型，而是字符串 'a=T'，这是为什么呢？

因为Python把0、空字符串''和None看成 False，其他数值和非空字符串都看成 True，所以：

True and 'a=T' 计算结果是 'a=T'

继续计算 'a=T' or 'a=F' 计算结果还是 'a=T'

* 短路计算

1. 在计算 a and b 时，如果 a 是 False，则根据与运算法则，整个结果必定为 False，因此返回 a；如果 a 是 True，则整个计算结果必定取决与 b，因此返回 b。

2. 在计算 a or b 时，如果 a 是 True，则根据或运算法则，整个计算结果必定为 True，因此返回 a；如果 a 是 False，则整个计算结果必定取决于 b，因此返回 b。

所以Python解释器在做布尔运算时，只要能提前确定计算结果，它就不会往后算了，直接返回结果。

* **List检索**

**需要特别注意的是**，索引从 0 开始，也就是说，第一个元素的索引是0，第二个元素的索引是1，以此类推。

顺序索引：L[0]#顺数第一个

倒序索引：L[-1]#倒数第一个

* **List添加**

**append()**总是把新的元素添加到 list 的尾部。

 insert()方法，它接受两个参数，第一个参数是索引号，第二个参数是待添加的新元素：

eg：L.insert（0，’paul’）

* **List删除**

L.Pop（）自动删除最后一个元素

L．pop（x）删除L列表第X个元素

* **List修改**

L[x]=‘hi’#更改Llist中的第x个元素为hi

* **Tuple**

Tuple列表和LIST一样是有序列表，但是Tuple不能修改。

#tuple性能比list好，创建速度快

T=（…）#创建tuple，注意是（），而list是[]

* **创立单元素tuple**

因为用()定义单元素的tuple有歧义，所以 Python 规定，单元素 tuple 要多加一个逗号“,”，这样就避免了歧义：

>>> t = (1,)

>>> print t

(1,)

* **内容可变的tuple**

Tuple的不变性是指指向不变，如果tuple的元素指向一个list，那么可以通过更改list的元素来“更改”tuple

* **If语句**

Python代码的缩进规则。具有相同缩进的代码被视为代码块。

缩进请严格按照Python的习惯写法：4个空格，不要使用Tab，更不要混合Tab和空格，否则很容易造成因为缩进引起的语法错误。

**注意**: if 语句后接表达式，然后用:表示代码块开始。

如果你在Python交互环境下敲代码，还要特别留意缩进，并且退出缩进需要多敲一行回车：

IF …：

Else：

* **If-elif-else**

IF…：

Elif：

Elif：

Else：

这一系列条件判断会从上到下依次判断，如果某个判断为 True，执行完对应的代码块，后面的条件判断就直接忽略，不再执行了。

* Break 语句

以使用 break 语句。

比如计算1至100的整数和，我们用while来实现：

sum = 0

x = 1

while True:

sum = sum + x

x = x + 1

if x > 100:

break

print sum

* dict{…，…，…}#按照**key: value**, 写出来即可。最后一个 key: value 的逗号可以省略。

dict的三个特点

第一个特点：其中的key不能重复，key不可变化，如：不能为list;value可变化

第二特点：查询速度块，占用内存大。与list相反

第三特点：其内部排序无序，不能存储有序集合。在不同机器打印时结果可能不同

* len（）#计算任意集合的大小，一个key-value算一个

可以简单地使用 d[key] 的形式来查找对应的 value，这和 list 很像，不同之处是，list 必须使用索引返回对应的元素，而dict使用key：

**一是先判断一下 key 是否存在，用 in 操作符：**

if 'Paul' in d:

print d['Paul']

如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。

**二是使用dict本身提供的一个 get 方法，在Key不存在的时候，返回None：**

>>> print d.get('Bart')

59

>>> print d.get('Paul')

None

* Dict赋值语句

d['Paul'] = 72 #如果key已存在，就会替换VALUE

* **set 持有一系列元素，这一点和 list 很像，但是set的元素没有重复，而且是无序的，这点和 dict 的 key很像。**创建 set 的方式是调用 set() 并传入一个 list，list的元素将作为set的元素：上述打印的形式类似 list， 但它不是 list，仔细看还可以发现，打印的顺序和原始 list 的顺序有可能是不同的，因为set内部存储的元素是**无序**的。set不能包含重复的元素，如果有重复元素，会自动去除
* SET是无序的，所以不能指向查询，只能查询元素是否存在。

'Bart' in s

True

* >>> s = set([1, 2, 3])

>>> s.add(4)

>>> print s

set([1, 2, 3, 4]) #SET 添加

* >>> s = set([1, 2, 3, 4])

>>> s.remove(4)

>>> print s

set([1, 2, 3]) #SET删除

* range(a,b,c)表示的从a到b-1的整数，步长为c

for x in range(1,100): #x=1,2，……99

* 编写函数

在Python中，定义一个函数要使用 def 语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用 return 语句返回。

我们以自定义一个求绝对值的 my\_abs 函数为例：

def my\_abs(x):

if x >= 0:

return x

else:

return -x

请注意，函数体内部的语句在执行时，一旦执行到return时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有return语句，函数执行完毕后也会返回结果，只是结果为 None。

return None可以简写为return。

* 递归函数

递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。可以试试计算 fact(10000)。

定义函数的时候，还可以有默认参数。

* 函数默认参数

例如Python自带的 int() 函数，其实就有两个参数，我们既可以传一个参数，又可以传两个参数：

>>> int('123')

123

>>> int('123', 8)

83

int()函数的第二个参数是转换进制，如果不传，默认是十进制 (base=10)，如果传了，就用传入的参数。

可见，函数的默认参数的作用是简化调用，你只需要把必须的参数传进去。但是在需要的时候，又可以传入额外的参数来覆盖默认参数值。

我们来定义一个计算 x 的N次方的函数:

def power(x, n):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

假设计算平方的次数最多，我们就可以把 n 的默认值设定为 2：

def power(x, n=2):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

这样一来，计算平方就不需要传入两个参数了：

>>> power(5)

25

由于函数的参数按从左到右的顺序匹配，所以默认参数只能定义在必需参数的后面：

# OK:

def fn1(a, b=1, c=2):

pass

# Error:

def fn2(a=1, b):

pass

* 函数可变参数

如果想让一个函数能接受任意个参数，我们就可以定义一个可变参数：

def fn(\*args):

print args

可变参数的名字前面有个 \* 号，我们可以传入0个、1个或多个参数给可变参数：

>>> fn()

()

>>> fn('a')

('a',)

>>> fn('a', 'b')

('a', 'b')

>>> fn('a', 'b', 'c')

('a', 'b', 'c')

可变参数也不是很神秘，Python解释器会把传入的一组参数组装成一个tuple传递给可变参数，因此，在函数内部，直接把变量 args 看成一个 tuple 就好了。

定义可变参数的目的也是为了简化调用。假设我们要计算任意个数的平均值，就可以定义一个可变参数：

def average(\*args):

...

这样，在调用的时候，可以这样写：

>>> average()

0

>>> average(1, 2)

1.5

>>> average(1, 2, 2, 3, 4)

2.4

* List切片

笨办法：

>>> [L[0], L[1], L[2]]

['Adam', 'Lisa', 'Bart']

之所以是笨办法是因为扩展一下，取前N个元素就没辙了。

取前N个元素，也就是索引为0-(N-1)的元素，可以用循环：

>>> r = []

>>> n = 3

>>> for i in range(n):

... r.append(L[i])

...

>>> r

['Adam', 'Lisa', 'Bart']

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

>>> L[0:3]

['Adam', 'Lisa', 'Bart']

**L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3**。即索引0，1，2，正好是3个元素。如果第一个索引是0，还可以省略：

>>> L[:3]['Adam', 'Lisa', 'Bart']

也可以从索引1开始，取出2个元素出来：

>>> L[1:3]

['Lisa', 'Bart']

只用一个 : ，表示从头到尾：

>>> L[:]

['Adam', 'Lisa', 'Bart', 'Paul']

因此，L[:]实际上复制出了一个新list。

切片操作还可以指定第三个参数：

>>> L[::2]

['Adam', 'Bart']

**第三个参数表示每N个取一个，上面的 L[::2] 会每两个元素取出一个来，也就是隔一个取一个。**

把list换成tuple，切片操作完全相同，只是切片的结果也变成了tuple。

* **因为 Python 的 for循环不仅可以用在list或tuple上，还可以作用在其他任何可迭代对象上。**

因此，迭代操作就是对于一个集合，无论该集合是有序还是无序，我们用 for 循环总是可以依次取出集合的每一个元素。

**注意**: 集合是指包含一组元素的数据结构，我们已经介绍的包括：

1. **有序集合**：list，tuple，str和unicode；

2. **无序集合**：set

3. **无序集合并且具有 key-value 对**：dict

* Value/intervalue

寻找List和tuple里面的值 #value生成list，占用的内存多，但是intervalue直接生成tuple，占用内存少

* Items/iteritems

可以看到，items() 方法把dict对象转换成了包含tuple的list，我们对这个list进行迭代，可以同时获得key和value

iteritem() 方法替换 items() ，最终实现的效果一样，但是消耗的内存降低50%，为什么差距那么大呢？因为 items() 返回的是一个 list，list 在迭代的时候会预先把所有的元素加载到内存，而 iteritem() 返回的一个迭代器(iterators)，迭代器在迭代的时候，迭代元素逐个的生成。

任意关键字参数for k, v in kw.iteritems():

* 生成list

要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]，我们可以用range(1, 11)：

>>> range(1, 11)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

但如果要生成[1x1, 2x2, 3x3, ..., 10x10]怎么做？方法一是循环：

>>> L = []

>>> for x in range(1, 11):

... L.append(x \* x)

...

>>> L

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

但是循环太繁琐，而列表生成式则可以用一行语句代替循环生成上面的list：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

这种写法就是Python特有的列表生成式。利用列表生成式，可以以非常简洁的代码生成 list。

写列表生成式时，把要生成的元素 x \* x 放到前面，后面跟 for 循环，就可以把list创建出来，十分有用，多写几次，很快就可以熟悉这种语法。

* isinstance(x, str) 可以判断变量 x 是否是字符串；
* 字符串的 upper() 方法可以返回大写的字母。
* **filter()**函数是 Python 内置的另一个有用的高阶函数，filter()函数接收一个**函数 f**和一个**list**，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，**filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。**
* 百分号（%）的使用：1，求余数；2，字符串格式化

字符串格式化使用字符串格式化操作符百分号（ % ）实现，在操作符的左侧是格式化字符串，右侧是希望被格式化的值；

注意：

只有元组和字典可以被格式化为一个以上的值，列表和其他序列会被格式化为一个值；

转换说明符，用于标记需要插入转换值的位置；

如果在格式化字符串中要输出百分号，则需要使用 %%

%s 字符串 (采用str()的显示)

%r 字符串 (采用repr()的显示)

%c 单个字符

%b 二进制整数

%d 十进制整数

%i 十进制整数

%o 八进制整数

%x 十六进制整数

%e 指数 (基底写为e)

%E 指数 (基底写为E)

%f 浮点数

%F 浮点数，与上相同%g 指数(e)或浮点数 (根据显示长度)

%G 指数(E)或浮点数 (根据显示长度)

%% 字符"%"

* 排序函数sort和sorted

Sorted(函数名，key)

sorted(iterable, /, \*, key=None, reverse=False)

Return a new list containing all items from the iterable in ascending order.

函数名.sort（）

sort(...)

L.sort(key=None, reverse=False) -> None -- stable sort \*IN PLACE\*



>>>sorted([36, 5, 12, 9, 21])

[5, 9, 12, 21, 36]

但 sorted()也是一个高阶函数，它可以接收一个比较函数来实现自定义排序，比较函数的定义是，传入两个待比较的元素 x, y，如果 x 应该排在 y 的前面，返回 -1，如果 x 应该排在 y 的后面，返回 1。如果 x 和 y 相等，返回 0。

* 闭包

def calc\_sum(lst):

def lazy\_sum():

return sum(lst)

return lazy\_sum

注意: 发现没法把 lazy\_sum 移到 calc\_sum 的外部，因为它引用了 calc\_sum 的参数 lst。

像这种内层函数引用了外层函数的变量（参数也算变量），然后返回内层函数的情况，称为闭包（Closure）。

闭包的特点是返回的函数还引用了外层函数的局部变量，所以，要正确使用闭包，就要确保引用的局部变量在函数返回后不能变。因此，返回函数不要引用任何循环变量，或者后续会发生变化的变量。需要增加一个函数来实现闭包。

* Lambda

Lambda（a,b:a+b）

传入参数a，b，返回a+b

在Python中，对匿名函数提供了有限支持。还是以map()函数为例，计算 f(x)=x2 时，除了定义一个f(x)的函数外，还可以直接传入匿名函数：

>>> map(lambda x: x \* x, [1, 2, 3, 4, 5, 6, 7, 8, 9])

[1, 4, 9, 16, 25, 36, 49, 64, 81]

通过对比可以看出，匿名函数 lambda x: x \* x 实际上就是：

def f(x):

return x \* x

关键字lambda 表示匿名函数，冒号前面的 x 表示函数参数。

匿名函数有个限制，就是只能有一个表达式，不写return，返回值就是该表达式的结果。

使用匿名函数，可以不必定义函数名，直接创建一个函数对象，很多时候可以简化代码：

* 时间戳

Python 的 time 模块下有很多函数可以转换常见日期格式。如函数time.time()用于获取当前时间戳, 如下实例:

实例(Python 2.0+)

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

import time; # 引入time模块

ticks = time.time()

print "当前时间戳为:", ticks

以上实例输出结果：

* 装饰器执行的顺序就是从最后一个装饰器开始，执行到第一个装饰器，再执行函数本身



* int()函数可以把字符串转换为整数，当仅传入字符串时，int()函数默认按十进制转换：
* 利用ImportError错误，我们经常在Python中动态导入模块：

try:

from cStringIO import StringIO

except ImportError:

from StringIO import StringIO

上述代码先尝试从cStringIO导入，如果失败了（比如cStringIO没有被安装），再尝试从StringIO导入。这样，如果cStringIO模块存在，则我们将获得更快的运行速度，如果cStringIO不存在，则顶多代码运行速度会变慢，但不会影响代码的正常执行。

try 的作用是捕获错误，并在捕获到指定错误时执行 except 语句。

* Python访问限制

Python对属性权限的控制是通过属性名来实现的，如果一个属性由双下划线开头(\_\_)，该属性就无法被外部访问。 但是，如果一个属性以"\_\_xxx\_\_"的形式定义，那它又可以被外部访问了，以"\_\_xxx\_\_"定义的属性在Python的类中被称为特殊属性，有很多预定义的特殊属性可以使用，通常我们不要把普通属性用"\_\_xxx\_\_"定义。

* Python的继承

Class person（object）必须要继承一个父类，没有的话就继承object

初始化父类

super(Student,self).\_\_init\_\_()

如果已经定义了Person类，需要定义新的Student和Teacher类时，可以直接从Person类继承：

class Person(object):

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

self.name = name

self.gender = gender

定义Student类时，只需要把额外的属性加上，例如score：

class Student(Person):

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

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

一定要用 super(Student, self).\_\_init\_\_(name, gender) 去初始化父类，否则，继承自 Person 的 Student 将没有 name 和 gender。

函数super(Student, self)将返回当前类继承的父类，即 Person ，然后调用\_\_init\_\_()方法，注意self参数已在super()中传入，在\_\_init\_\_()中将隐式传递，不需要写出（也不能写）。

* 获取对象属性

isinstance(object, classinfo)

#object实例对象，classinfo类名，判断实例对象与classinfo是不是同一类型

type() 函数获取变量的类型，它返回一个 Type 对象：

dir() 函数获取变量的所有属性：

isinstance() 函数来判断一个对象是否是一个已知的类型，类似 type()。

isinstance() 与 type() 区别：

type() 不会认为子类是一种父类类型，不考虑继承关系。

isinstance() 会认为子类是一种父类类型，考虑继承关系。

如果要判断两个类型是否相同推荐使用 isinstance()。

* 修改对象属性

setattr() 函数：设置对象的属性值，且该属性必须存在

setattr(object, name, value)

getattr() 函数：返回对象的属性值

getattr(object, name[, default])

#object -- 对象。

name -- 字符串，对象属性。

default -- 默认返回值，如果不提供该参数，在没有对应属性时，将触发 AttributeError。

* \*args和\*\*kw

\*args和\*\*kw都代表不确定的函数变量，即在函数调用时可以选择传入或者不传入参数，都不影响函数运行。

其中\*args将传入对象当成是元组对待；\*\*kw将传入对象当成字典对待。\*\*kw的作用是在函数调用时，可变参数的key,value值打包成dic，传给kw。

* a,b=b,a+b就是a=b, b=a+b
* self.p \* r.q # self对象的p值乘以r对象的q值