### 置换两个变量的值。

a,b = b,a

### 链式比较

a = 3

b = 1

1 <= b <= a < 10 #True

### 字符串反转

s[::-1]

### for…else…语句

for x in xrange(1,5):

if x == 5:

print 'find 5'

break

else:

print 'can not find 5!'

#can not find 5!

for...else...的else部分用来处理没有从for循环中断的情况。有了它，我们不用设置状态变量来检查是否for循环有break出来，简单方便。

### 使用zip创建键值对

keys = ['Name', 'Sex', 'Age']

values = ['Tim', 'Male', 23]

dic = dict(zip(keys, values))

#{'Age': 23, 'Name': 'Tim', 'Sex': 'Male'}

序列乘法¶

Zeroes = [0] \* 100

“Print to”语法¶

最近,”>>” 操作符被重载了, 这样你就可以像下面那样在print语句中使用它了:

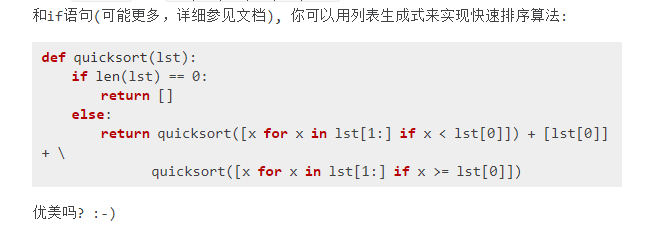
**print** >> sys.stderr, "this is an error message"

>>右边应该是一个文件对象。

译者注例子(Python2.7)

**print** >> file("myfile", "w"), "hello world"







# Python 内置函数

|  |  | **内置函数** |  |  |
| --- | --- | --- | --- | --- |
| [abs()](http://www.runoob.com/python/func-number-abs.html) | [divmod()](http://www.runoob.com/python/python-func-divmod.html) | [input()](http://www.runoob.com/python/python-func-input.html) | [open()](http://www.runoob.com/python/python-func-open.html) | [staticmethod()](http://www.runoob.com/python/python-func-staticmethod.html) |
| [all()](http://www.runoob.com/python/python-func-all.html) | [enumerate()](http://www.runoob.com/python/python-func-enumerate.html) | [int()](http://www.runoob.com/python/python-func-int.html) | [ord()](http://www.runoob.com/python/python-func-ord.html) | [str()](http://www.runoob.com/python/python-func-str.html) |
| [any()](http://www.runoob.com/python/python-func-any.html) | [eval()](http://www.runoob.com/python/python-func-eval.html) | [isinstance()](http://www.runoob.com/python/python-func-isinstance.html) | [pow()](http://www.runoob.com/python/func-number-pow.html) | [sum()](http://www.runoob.com/python/python-func-sum.html) |
| [basestring()](http://www.runoob.com/python/python-func-basestring.html) | [execfile()](http://www.runoob.com/python/python-func-execfile.html) | [issubclass()](http://www.runoob.com/python/python-func-issubclass.html) | [print()](http://www.runoob.com/python/python-func-print.html) | [super()](http://www.runoob.com/python/python-func-super.html) |
| [bin()](http://www.runoob.com/python/python-func-bin.html) | [file()](http://www.runoob.com/python/python-func-file.html) | [iter()](http://www.runoob.com/python/python-func-iter.html) | [property()](http://www.runoob.com/python/python-func-property.html) | [tuple()](http://www.runoob.com/python/att-tuple-tuple.html) |
| [bool()](http://www.runoob.com/python/python-func-bool.html) | [filter()](http://www.runoob.com/python/python-func-filter.html) | [len()](http://www.runoob.com/python/att-string-len.html) | [range()](http://www.runoob.com/python/python-func-range.html) | [type()](http://www.runoob.com/python/python-func-type.html) |
| [bytearray()](http://www.runoob.com/python/python-func-bytearray.html) | [float()](http://www.runoob.com/python/python-func-float.html) | [list()](http://www.runoob.com/python/att-list-list.html) | [raw\_input()](http://www.runoob.com/python/python-func-raw_input.html) | [unichr()](http://www.runoob.com/python/python-func-unichr.html) |
| [callable()](http://www.runoob.com/python/python-func-callable.html) | [format()](http://www.runoob.com/python/att-string-format.html) | [locals()](http://www.runoob.com/python/python-func-locals.html) | [reduce()](http://www.runoob.com/python/python-func-reduce.html) | unicode() |
| [chr()](http://www.runoob.com/python/python-func-chr.html) | [frozenset()](http://www.runoob.com/python/python-func-frozenset.html) | [long()](http://www.runoob.com/python/python-func-long.html) | [reload()](http://www.runoob.com/python/python-func-reload.html) | [vars()](http://www.runoob.com/python/python-func-vars.html) |
| [classmethod()](http://www.runoob.com/python/python-func-classmethod.html) | [getattr()](http://www.runoob.com/python/python-func-getattr.html) | [map()](http://www.runoob.com/python/python-func-map.html) | [repr()](http://www.runoob.com/python/python-func-repr.html) | [xrange()](http://www.runoob.com/python/python-func-xrange.html) |
| [cmp()](http://www.runoob.com/python/func-number-cmp.html) | [globals()](http://www.runoob.com/python/python-func-globals.html) | [max()](http://www.runoob.com/python/func-number-max.html) | [reverse()](http://www.runoob.com/python/att-list-reverse.html) | [zip()](http://www.runoob.com/python/python-func-zip.html) |
| [compile()](http://www.runoob.com/python/python-func-compile.html) | [hasattr()](http://www.runoob.com/python/python-func-hasattr.html) | [memoryview()](http://www.runoob.com/python/python-func-memoryview.html) | [round()](http://www.runoob.com/python/func-number-round.html) | [\_\_import\_\_()](http://www.runoob.com/python/python-func-__import__.html) |
| [complex()](http://www.runoob.com/python/python-func-complex.html) | [hash()](http://www.runoob.com/python/python-func-hash.html) | [min()](http://www.runoob.com/python/func-number-min.html) | [set()](http://www.runoob.com/python/python-func-set.html) |  |
| [delattr()](http://www.runoob.com/python/python-func-delattr.html) | [help()](http://www.runoob.com/python/python-func-help.html) | [next()](http://www.runoob.com/python/python-func-next.html) | [setattr()](http://www.runoob.com/python/python-func-setattr.html) |  |
| [dict()](http://www.runoob.com/python/python-func-dict.html) | [hex()](http://www.runoob.com/python/python-func-hex.html) | object() | [slice()](http://www.runoob.com/python/python-func-slice.html) |  |
| [dir()](http://www.runoob.com/python/python-func-dir.html) | [id()](http://www.runoob.com/python/python-func-id.html) | [oct()](http://www.runoob.com/python/python-func-oct.html) | [sorted()](http://www.runoob.com/python/python-func-sorted.html) | [exec 内置表达式](http://www.runoob.com/python/python-func-exec.html) |

### Getattr 函数（）

描述

**getattr()** 函数用于返回一个对象属性值。

语法

getattr 语法：

getattr(object, name[, default])

参数

* 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。

### Hasattr（）函数

描述

**getattr()** 函数用于返回一个对象属性值。

语法

getattr 语法：

getattr(object, name[, default])

参数

* 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。

### python中方法也是属性

我们在 **class** 中定义的实例方法其实也是属性，它实际上是一个函数对象：

class Person(object):

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

self.name = name

self.score = score

def get\_grade(self):

return 'A'

p1 = Person('Bob', 90)

print p1.get\_grade

# => <bound method Person.get\_grade of <\_\_main\_\_.Person object at 0x109e58510>>

print p1.get\_grade()

# => A

也就是说，**p1.get\_grade**返回的是一个函数对象，但这个函数是一个绑定到实例的函数，**p1.get\_grade()**才是方法调用。

因为方法也是一个属性，所以，它也可以动态地添加到实例上，只是需要用 types.MethodType() 把一个函数变为一个方法：

import types

def fn\_get\_grade(self):

if self.score >= 80:

return 'A'

if self.score >= 60:

return 'B'

return 'C'

class Person(object):

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

self.name = name

self.score = score

p1 = Person('Bob', 90)

p1.get\_grade = types.MethodType(fn\_get\_grade, p1, Person)

print p1.get\_grade()

# => A

p2 = Person('Alice', 65)

print p2.get\_grade()

# ERROR: AttributeError: 'Person' object has no attribute 'get\_grade'

# 因为p2实例并没有绑定get\_grade

给一个实例动态添加方法并不常见，直接在class中定义要更直观。

## python中 @property

考察 **Student**类：

class Student(object):

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

self.name = name

self.score = score

当我们想要修改一个**Student** 的 **scroe** 属性时，可以这么写：

s = Student('Bob', 59)

s.score = 60

但是也可以这么写：

s.score = 1000

显然，直接给属性赋值无法检查分数的有效性。

如果利用两个方法：

class Student(object):

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

self.name = name

self.\_\_score = score

def get\_score(self):

return self.\_\_score

def set\_score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

这样一来，**s.set\_score(1000)** 就会报错。

这种使用 **get/set**方法来封装对一个属性的访问在许多面向对象编程的语言中都很常见。

但是写**s.get\_score()** 和 **s.set\_score()**没有直接写 **s.score** 来得直接。

有没有两全其美的方法？----有。

因为Python支持高阶函数，在函数式编程中我们介绍了装饰器函数，可以用装饰器函数把 **get/set**方法“装饰”成属性调用：

class Student(object):

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

self.name = name

self.\_\_score = score

@property

def score(self):

return self.\_\_score

@score.setter

def score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

**注意:** 第一个score(self)是get方法，用@property装饰，第二个score(self, score)是set方法，用@score.setter装饰，@score.setter是前一个@property装饰后的副产品。

现在，就可以像使用属性一样设置score了：

>>> s = Student('Bob', 59)

>>> s.score = 60

>>> print s.score

60

>>> s.score = 1000

Traceback (most recent call last):

...

ValueError: invalid score

说明对**score**赋值实际调用的是 **set方法**。

### 任务

如果没有定义**set方法**，就不能对“属性”赋值，这时，就可以创建一个只读“属性”。

请给**Student**类加一个**grade**属性，根据 **score**计算 A（>=80）、B、C（<60）。

**?**不会了怎么办

用 @property 修饰 grade 的 get 方法即可实现只读属性。

**参考代码:**

class Student(object):

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

self.name = name

self.\_\_score = score

@property

def score(self):

return self.\_\_score

@score.setter

def score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

@property

def grade(self):

if self.score < 60:

return 'C'

if self.score < 80:

return 'B'

return 'A'

s = Student('Bob', 59)

print s.grade

s.score = 60

print s.grade

s.score = 99

print s.grade

## python中 \_\_slots\_\_

由于Python是动态语言，任何实例在运行期都可以动态地添加属性。

如果要限制添加的属性，例如，**Student**类只允许添加 **name、gender**和**score**这3个属性，就可以利用Python的一个特殊的**\_\_slots\_\_**来实现。

顾名思义，**\_\_slots\_\_**是指一个类允许的属性列表：

class Student(object):

\_\_slots\_\_ = ('name', 'gender', 'score')

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

self.name = name

self.gender = gender

self.score = score

现在，对实例进行操作：

>>> s = Student('Bob', 'male', 59)

>>> s.name = 'Tim' # OK

>>> s.score = 99 # OK

>>> s.grade = 'A'

Traceback (most recent call last):

...

AttributeError: 'Student' object has no attribute 'grade'

**\_\_slots\_\_**的目的是限制当前类所能拥有的属性，如果不需要添加任意动态的属性，使用**\_\_slots\_\_**也能节省内存。

### 任务

假设**Person**类通过**\_\_slots\_\_**定义了**name**和**gender**，请在派生类**Student**中通过**\_\_slots\_\_**继续添加**score**的定义，使**Student**类可以实现**name、gender**和**score** 3个属性。

**?**不会了怎么办

**Student**类的**\_\_slots\_\_**只需要包含**Person**类不包含的**score**属性即可。

**参考代码:**

class Person(object):

\_\_slots\_\_ = ('name', 'gender')

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

self.name = name

self.gender = gender

class Student(Person):

\_\_slots\_\_ = ('score',)

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

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

self.score = score

s = Student('Bob', 'male', 59)

s.name = 'Tim'

s.score = 99

print s.score

## python中 \_\_call\_\_

在Python中，函数其实是一个对象：

>>> f = abs

>>> f.\_\_name\_\_

'abs'

>>> f(-123)

123

由于 **f** 可以被调用，所以，**f** 被称为可调用对象。

所有的函数都是可调用对象。

一个类实例也可以变成一个可调用对象，只需要实现一个特殊方法**\_\_call\_\_()**。

我们把**Person**类变成一个可调用对象：

class Person(object):

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

self.name = name

self.gender = gender

def \_\_call\_\_(self, friend):

print 'My name is %s...' % self.name

print 'My friend is %s...' % friend

现在可以对**Person**实例直接调用：

>>> p = Person('Bob', 'male')

>>> p('Tim')

My name is Bob...

My friend is Tim...

单看**p('Tim')**你无法确定 **p** 是一个函数还是一个类实例，所以，在Python中，函数也是对象，对象和函数的区别并不显著。

### 任务

改进一下前面定义的斐波那契数列：

class Fib(object):

???

请加一个\_\_call\_\_方法，让调用更简单：

>>> f = Fib()

>>> print f(10)

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

**?**不会了怎么办

要正确定义参数：\_\_call\_\_(self, num)

**参考代码:**

class Fib(object):

def \_\_call\_\_(self, num):

a, b, L = 0, 1, []

for n in range(num):

L.append(a)

a, b = b, a + b

return L

f = Fib()

print f(10)