Table of Contents

Chapter 1. 常用函数

- abs(x)

  abs()返回一个数字的绝对值。如果给出复数，返回值就是该复数的模。

  >>>print abs(-100) 100 >>>print abs(1+2j) 2.2360679775
- callable(object)

  callable()函数用于测试对象是否可调用，如果可以则返回 1(真)；否则返回 0(假)。可调用对象包括函数、方法、代码对象、类和已经定义了“调用”方法的类实例。

  >>> a="123" >>> print callable(a) 0 >>> print callable(chr) 1
- cmp(x,y)

  cmp()函数比较 x 和 y 两个对象，并根据比较结果返回一个整数，如果 x<y，则返回-1；如果 x>y，则返回 1,如果 x==y 则返回 0。

  >>>a=1 >>>b=2 >>>c=2 >>> print cmp(a,b) -1 >>> print cmp(b,a) 1 >>> print cmp(b,c) 0
- divmod(x,y)

  divmod(x,y)函数完成除法运算，返回商和余数。

  >>> divmod(10,3) (3, 1) >>> divmod(9,3) (3, 0)
- isinstance(object,class-or-type-or-tuple) -> bool

  测试对象类型

  >>> a='isinstance test' >>> b=1234 >>> isinstance(a,str) True >>> isinstance(a,int) False >>> isinstance(b,str) False >>> isinstance(b,int) True
- len(object) -> integer

  len()函数返回字符串和序列的长度。

  >>> len("aa") 2 >>> len([1,2]) 2
- pow(x,y[,z])

  pow()函数返回以 x 为底，y 为指数的幂。如果给出 z 值，该函数就计算 x 的 y 次幂值被 z 取模的值。

  >>> print pow(2,4) 16 >>> print pow(2,4,2) 0 >>> print pow(2.4,3) 13.824
- range([lower,]stop[,step])

  range()函数可按参数生成连续的有序整数列表。

  >>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] >>> range(1,10) [1, 2, 3, 4, 5, 6, 7, 8, 9] >>> range(1,10,2) [1, 3, 5, 7, 9]
- round(x[,n])

  round()函数返回浮点数 x 的四舍五入值，如给出 n 值，则代表舍入到小数点后的位数。

  >>> round(3.333) 3.0 >>> round(3) 3.0 >>> round(5.9) 6.0
- type(obj)

  type()函数可返回对象的数据类型。

>>> type(a) <type 'list'> >>> type(copy) <type 'module'> >>> type(1) <type 'int'>

- xrange([lower,]stop[,step])

  xrange()函数与 range()类似，但 xrnage()并不创建列表，而是返回一个 xrange 对象，它的行为与列表相似，但是只在需要时才计算列表值，当列表很大时，这个特性能为我们节省内存。

  >>> a=xrange(10) >>> print a[0] 0 >>> print a[1] 1 >>> print a[2] 2

Chapter 2. 内置类型转换函数

- chr(i)

  chr()函数返回 **ASCII** 码对应的字符串。

  >>> print chr(65) A >>> print chr(66) B >>> print chr(65)+chr(66) AB

- complex(real[,imaginary])

  complex()函数可把字符串或数字转换为复数。

  >>> complex("2+1j") (2+1j) >>> complex("2") (2+0j) >>> complex(2,1) (2+1j) >>> complex(2L,1) (2+1j)

- float(x)

  float()函数把一个数字或字符串转换成浮点数。

  >>> float("12") 12.0 >>> float(12L) 12.0 >>> float(12.2) 12.199999999999999

- hex(x)

  hex()函数可把整数转换成十六进制数。

  >>> hex(16) '0x10' >>> hex(123) '0x7b'

- long(x[,base])

  long()函数把数字和字符串转换成长整数，base 为可选的基数。

  >>> long("123") 123L >>> long(11) 11L

- list(x)

  list()函数可将序列对象转换成列表。如：

  >>> list("hello world") ['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'] >>> list((1,2,3,4)) [1, 2, 3, 4]

- int(x[,base])

  int()函数把数字和字符串转换成一个整数，base 为可选的基数。

  >>> int(3.3) 3 >>> int(3L) 3 >>> int("13") 13 >>> int("14",15) 19

- min(x[,y,z...])

  min()函数返回给定参数的最小值，参数可以为序列。

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

- max(x[,y,z...])

  max()函数返回给定参数的最大值，参数可以为序列。

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

- oct(x)

  oct()函数可把给出的整数转换成八进制数。

  >>> oct(8) '010' >>> oct(123) '0173'

- ord(x)

  ord()函数返回一个字符串参数的 **ASCII** 码或 Unicode 值。

>>> ord("a") 97 >>> ord(u"a") 97

- str(obj)

  str()函数把对象转换成可打印字符串。

  >>> str("4") '4' >>> str(4) '4' >>> str(3+2j) '(3+2j)'

- tuple(x)

  tuple()函数把序列对象转换成 tuple。

  >>> tuple("hello world") ('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd') >>> tuple([1,2,3,4]) (1, 2, 3, 4)

Chapter 3. 序列处理函数

- 常用函数中的 len()、max()和 min()同样可用于序列。
- filter(function,list)

  调用 filter()时，它会把一个函数应用于序列中的每个项，并返回该函数返回真值时的所有项，从而过滤掉返回假值的所有项。

  >>> def nobad(s): ... return s.find("bad") == -1 ... >>> s = ["bad","good","bade","we"] >>> filter(nobad,s) ['good', 'we']
  这个例子通过把 nobad()函数应用于 s 序列中所有项，过滤掉所有包含“bad”的项。

- map(function,list[,list])

  map()函数把一个函数应用于序列中所有项，并返回一个列表。

  >>> import string >>> s=["**python**","zope","linux"] >>> map(string.capitalize,s) ['**Python**', 'Zope', 'Linux']
  map()还可同时应用于多个列表。如：

  >>> import operator >>> s=[1,2,3]; t=[3,2,1] >>> map(operator.mul,s,t) # s[i]*t[j] [3, 4, 3]
  如果传递一个 None 值，而不是一个函数，则 map()会把每个序列中的相应元素合并起来，并返回该元组。如：

  >>> a=[1,2];b=[3,4];c=[5,6] >>> map(None,a,b,c) [(1, 3, 5), (2, 4, 6)]

- reduce(function,seq[,init])

  reduce()函数获得序列中前两个项，并把它传递给提供的函数，获得结果后再取序列中的下一项，连同结果再传递给函数，以此类推，直到处理完所有项为止。

  >>> import operator >>> reduce(operator.mul,[2,3,4,5]) # ((2*3)*4)*5 120 >>> reduce(operator.mul,[2,3,4,5],1) # (((1*2)*3)*4)*5 120 >>> reduce(operator.mul, [2,3,4,5],2) # (((2*2)*3)*4)*5 240

- zip(seq[,seq,...])

  zip()函数可把两个或多个序列中的相应项合并在一起，并以元组的格式返回它们，在处理完最短序列中的所有项后就停止。

  >>> zip([1,2,3],[4,5],[7,8,9]) [(1, 4, 7), (2, 5, 8)]
  如果参数是一个序列，则 zip()会以一元组的格式返回每个项，如：

  >>> zip((1,2,3,4,5)) [(1,), (2,), (3,), (4,), (5,)] >>> zip([1,2,3,4,5]) [(1,), (2,), (3,), (4,), (5,)]

Chapter 4. String 模块

- replace(string,old,new[,maxsplit])

字符串的替换函数，把字符串中的 old 替换成 new。默认是把 string 中所有的 old 值替换成 new 值，如果给出 maxsplit 值，还可控制替换的个数，如果 maxsplit 为 1，则只替换第一个 old 值。

>>>a="11223344" >>>print string.replace(a,"1","one") oneone2223344 >>>print string.replace(a,"1","one",1) one12223344

- capitalize(string)

  该函数可把字符串的首个字符替换成大字。

  >>> import string >>> print string.capitalize("**python**") **Python**

- split(string,sep=None,maxsplit=-1)

  从 string 字符串中返回一个列表，以 sep 的值为分界符。

  >>> import string >>> ip="192.168.3.3" >>> ip_list=string.split(ip,'.') >>> print ip_list ['192', '168', '3', '3']

import__(name[, globals[, locals[, fromlist[, level]]]])
被 import 语句调用的函数。 它的存在主要是为了你可以用另外一个有兼容接口的函数 来改变 import 语句的语义. 为什么和怎么做的例子, 标准库模块 ihooks 和 rexec. 也可以查看 imp, 它定义了有用的操作，你可以创建你自己的 __import__()函数.
例 如, 语句"import spam" 结果对应下面的调用: __import__('spam', globals(), locals(), [], -1); 语句 "from spam.ham import eggs" 结果对应调用 "__import__('spam.ham', globals(), locals(), ['eggs'], -1)". 注意即使 locals()和['eggs']作为参数传递, __import__() 函数不会设置局部变量 eggs; import 语句后面的代码完成这项功能的. (实事上, 标准的执行根本没有使用局部参数 , 仅仅使用 globals 决定 import 语句声明 package 的上下文.)

当 name 变量是 package.module 的形 式, 正常讲, 将返回顶层包 (第一个点左边的部分), 而不是名为 name 的模块. 然而, 当指定一个非空的 formlist 参数， 将返回名为 name 的模块. 这样做是为了兼容为不同种类的 import 语句产生的字节码; 当使用 "import spam.ham.eggs", 顶层包 spam 必须在导入的空间中, 但是当使用"from spam.ham import eggs", 必须使用 spam.ham 子包来查找 eggs 变量. 作为这种行为的工作区间, 使用 getattr()提取需要的组件. 例如, 你可以定义下面:

```
def my_import(name):
    mod = __import__(name)
    components = name.split('.')
    for comp in components[1:]:
        mod = getattr(mod, comp)
    return mod
```

level 指定了是否使用相对或绝对导入. 默认是 -1 将使用将尝试使用相对或绝对导入. 0 仅使用绝对导入.正数意味着相对查找模块文件夹的 level 层父文件夹中调用__import__。

abs(x)
返回一个数的绝对值。参数也许是一个普通或长整型，或者一个浮点数。如果参数是一个复数，返回它的积。

all(iterable)
如果迭代的所有元素都是真就返回真。

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

版本 2.5 中新增.

any(iterable)
如果迭代中有一个元素为真就返回真。

```
def any(iterable):
    for element in iterable:
        if element:
            return True
    return False
```

版本 2.5 中新增.

basestring( )
这个抽象类型是 str 和 unicode 的父类。它不能被调用或初始化，但是它可以使用来测试一个对象是否是 str 或 unicode 的实例。isinstance(obj, basestring)等价于 isinstance(obj, (str, unicode))
版本 2.3 中新增.

bool([x])
将一个值转换为 Boolean,使用标准的真测试程序。如果 x 是假或忽略了，将返回 False;否则将返回 True.bool 也是一个 class，它是 int 的一个子类，bool 类不能进一步子类化。它仅有 False 和 True 两个实例。

callable(object)
如果 object 参数可以调用就返回 True,否则返回 False。如果返回 True,它仍然可能调用失败，但是如果返回 False，就永远不可能调用成功。注类是可调用的（调用一个类返回一个实例）;类的实例如果有一个__call__()方法就是可调用的。

chr(i)
返回一个一个 ascii 码是整数 i 的字符的字符串。例如，chr(97)返回 string 'a'.这和 ord()刚好相反。这参数在[0..255]之间,全部包含。如果超出这个范围，就抛出 ValueError

classmethod(function)
返回函数的一个类方法。一个类方法
Return a class method for function. 一个类方法接收类作为它的第一个潜在参数，就像一个实例方法接收一个实例。声明一个类方法，如下：

```
class C:
    @classmethod
    def f(cls, arg1, arg2, ...): ...
```

@classmethod 形式是一个函数 decorator。在函数参考手册的第七章有关函数定义的描述寻找更多的细节.
它即可以在一个类（例如 C.f）上调用，也可以在一个实例上调用。这个实例将忽略它的类。如果一个继承类调用一个类方法，这继承类对象将作为第一个潜在的参数传递。

类方法不同于 C++或 Java 的静态方法。如果你想这样做，使用 staticmethod()。

cmp(x, y)

根据比较两介对象 x 和 y 的结果，返回一个整数。如果 x<y,返回值是负的,如果 x==y,返回 0，如果 x > y,返回一个正数.

compile(string, filename, kind[, flags[, dont_inherit]])

编 译 string 为一个代码对象。代码对象能够通过 exec 语句执行或者通过调用 eval()计算。这 filename 参数指定代码从哪个文件读取。如果不 从文件中读取，就须传递一些可识别的值（通常使用'<string>'）。kind 参数指定哪种代码被编译;如果是包含一系列语句组成的子符 串可以‘exec’,如果是由一个表达式组成，就'eval',如果由一个交互语句组成就‘singlw’(在后面的例子，表达式语句计算的结果不

是 None 将打印出来)。
当编译一个多行语句时，应用两个警告：必须以'\n'作为行结束符，同时输入必须至少以一个'\n'作为结束。如果是以'\r\n'作为行结束，使用 string 的 repalce()方法将其改为'\n'.

可先的参数 flags 和 dont_inherit 控制影响 string 编译的 future 语句。更详细的请参考英文文档。

complex([real[, imag]])

创 建一个复数 real + imag*j 或者将一个 string 或者 number 转化为一个复数. 如果第一个参数是一个字符串,它将作为复数解释，函数将被调用，而忽略第二个参数。第二个参数不可能是一个字符串。每一个参数都可能是一个数字类型包括复 数.如果 imag 省略了, 它默认为 0，函数将当作一个数字转换函数像 int(), long() and float().如果参数都省略了，将返回 0j.

delattr(object, name)
与 setattr()相对的，参数是一个对象和一个 string.string 必须是对象的一个属性。函数删除 object 这个名为 string 的属性。例如，delattr(x, 'foobar')等价于 del x.foobar

dict([arg])
以 一个可选的位置参数或一组关键词参数初始化一个字典。如果没有给出参数，返回一个空的字典。如果位置参数 arg 是一个映射对象，返回一个字典映射与映射对 象相同的 keys 对应同样的值。那么位置参数应是一个序列，支持迭代的容器，或者是一个迭代对象。参数的每一个元素也必须是其中的一种，同时每一个包含两 个对象。第一个作为新字典的 key,同时第二个作为这个 key 的值。如果给定的 key 不只出现一次，这新字典将包含最后一个值。
如果给出的是关键词参数，关键词和与其相关的值都将作为字典的项添加到字典中。如果一个 ke 既作为位置参数和作为一个关键词参数，关键词关联的值将保留在字典中。例如，下面所有返回都等价于 {"one": 2, "two": 3}:

```
dict({'one': 2, 'two': 3})
dict({'one': 2, 'two': 3}.items())
dict({'one': 2, 'two': 3}.iteritems())
dict(zip(('one', 'two'), (2, 3)))
dict([['two', 3], ['one', 2]])
dict(one=2, two=3)
dict([(['one', 'two'][i-2], i) for i in (2, 3)])
```

版本 2.2 中新增.

dir([object])
如 果没有参数，返回当前 local 符号表中名字的列表。如果有一个参数将尝试返回这个参数对象的有效属性。从对象的__dict__（如果定义了）属性，类 或 type 对象收集信息。这个列表不一定是完整的。如果对象是一个模块对象，list 包含模块属性的列表。如果对象是一个 type 或 class 对象，列表 将包含属性的名字，同时递归其基地的属性。否则，列表将包含对象属性的名字，类属性的名字，递归若基类的属性名字。结果列表按字母顺序排序。例如：

```
>>> import struct
>>> dir()
['__builtins__', '__doc__', '__name__', 'struct']
>>> dir(struct)
['__doc__', '__name__', 'calcsize', 'error', 'pack', 'unpack']
```

注:因为 dir()主要在交互提示下方便使用，它尝试提供一给有意思的名字而不是尝试提供严格的或与定义一样的名字,在 relrase 中它的细节行为也许会改变。

divmod(a, b)

用 两个数字（不是复数）作为参数，返回 long 除法得到的商和余数组成的一对数字。使用混合的操作类型，混合的类型，将应用应用二进制算术操作。对于一般和 长整数，结果与(a // b, a % b)相同。对于浮

点数结果和(q, a % b)相同，q 通常是 math.floor(a / b)，但如果是比 1 小，就为 1.在任何情况下 q * b + a % b 与 a 很接近，如果 a % b 是非 0 的，它与 b 有相同的符号，同时 0 <= abs(a % b) < abs(b).

enumerate(iterable)

返回 enumerate 对象. iterable 必须是一个序列, 一个迭代, 或者其它对象它支持迭代.enumerate()返回的 iterator 的 next()方法 返回一个元组包含一定的数目（从 0 开始）和从迭代中获取的对应的值。
enumerate() 对于获取一个索引系列很有用: (0, seq[0]), (1, seq[1]), (2, seq[2]), ... 版本 2.3 中新增.

eval(expression[, globals[, locals]])

该参数是一个字符串和可选的 globals 和 locals。如果提供 globals，globals 必须是一个字典。如果提供 locals，locals 可以是任何映射对象。2.4 版本修改：以前 locals 被要求是一个字典。

expression 参数是作为一个 Python 表 达式被分析和评价（技术上来说，一个条件列表）使用 globals 以及 locals 字典作为 global 和 local 名字空间。如果提供了 globals 字典但没有'__builtins__'，当前 globals 在表达式被分析前被复制到 globals 中。这意味着表达式可以完全访问标准 __builtin__模块和受限的环境。如果 locals 字典省略则默认为 globals 字典。如果两个字典都被省略，表达式在调用 eval 的环境中执 行。返回值是计算表达式的结果。语法错误报告为 exceptions。 例如:

```
>>> x = 1
>>> print eval('x+1')
2
```

此函数也可以用来执行任意代码的对象（如 compile()创建的）。在这种情况下，传入一个代码对象，而不是一个字符串。该代码对象必须已编译传给'eval'作为这种参数。

提示：EXEC 语句支持是动态执行语句。execfile（）函数支持从一个文件中执行语句。globals()和 locals()函数分别返回当前的 global 和 local 字典，这对使用 eval（）或 execfile（）很有帮助。

execfile(filename[, globals[, locals]])

这个功能类似于 EXEC 语句，但分析一个文件，而不是一个字符串。这是不同之处在于它的 import 语句不使用模块管理 - 它无条件读取文件，并不会创建一个新的 module。
该参数是一个文件名和两个可选字典。该文件被作为 Python 语 句序列（类似于一个模块）使用 globals 和 locals 作为 global and local 命名空间来分析和计算。如果提供 locals，locals 可以是任何映射对象。2.4 版本修改：以前 locals 被要求是一个字典。如果 locals 字典省略则默认为全局字典。如果两个字典都被省略，表达式 execfile（）被调用的环境中执行。返回值为 None。

警告：默认的 locals 为下面的 locals()：不要尝试修改默认的本地词典。如果你需要看到在函数 execfile() 返回后 locals 代码的的影响，传递一个明确的 locals 字典。 execfile（）不能用于依赖修改函数的 locals。

file(filename[, mode[, bufsize]])

文件类型的构造函数，3.9 节进一步描述，`文件对象'。构造函数的参数与下面的内建的 open()函数是一样的。
当打开一个文件，它是最好使用的 open（），而不是直接调用此构造函数。文件更适合检验类型（例如，isinstance(f, file)）。

版本 2.2 中新增.

filter(function, iterable)

function 返回 true 时从 iterable 的元素中构造一个列表。迭代可以是一个序列，一个支持迭代的容器，或一个迭代器，如果 Iterable 的是一个字符串或 一个元组，其结果也有这种类型的，否则它始终是一个列表。如果 function 是 None，假定它是恒等函数，即，迭代是 false 其所有元素都被删除。
请注意，filter(function,iterable)，如果函数不为 None 等价于[item for item in iterable if function(item)]，如果函数为 None 等价于[item for item in iterable if item]。

float([x])

将字符串或数字转换或一个浮点数。如果参数是一个字符串，它必须包含一个可能带符号的十进制或浮点数，可能嵌入空格。否则，参数可以是一个普通或长整数或浮点数，返回一个与之相同值的浮点数（在 Python 的浮点精度内）。如果没有给出参数，返回 0.0。
注意：当传递一个字符串，可能会返回 NaN 和 Infinity，这取决于底层 C 库。

frozenset([iterable])

返 回一个 frozenset 对象，其元素来自于 Iterable。 Frozensets 组没有更新的方法，但可以哈希和其他组成员或作为字典键使用。一个 frozenset 的元素必须是不可改变。内部 sets 也应是 frozenset 对象。如果迭代没有指定，返回一个新的空集，frozenset ([])。 版本 2.4 中新增

getattr(object, name[, default])

返 回 object 名为 name 属性的值。名称必须是一个字符串。如果该字符串是对象的其中属性名字，结果是该属性的值。例如，getattr(x, 'foobar')相当于 x.foobar。如果指定的属性不存在，则返回默认提供的，否则抛出 AttributeError。

globals( )

返回代表当前 global 符号表字典的字典。这始终是当前模块字典（在一个函数或方法内，是在它被定义的模块，而不是被调用的模块）。
hasattr(object, name)

该参数是一个对象和一个字符串。如果字符串是对象的其中一个属性，结果为 True，如果没有返回 False。（这是通过调用的 getattr（对象名称），看是否引发异常与否。）

hash(object)

返回对象（如果有的话）的哈希值。哈希值是整数。它们被用来在词典查找时，作为一个快速比较字典 keys 键。具有相同的哈希值，数值相等（即使它们属于不同的类型，因为是 1 和 1.0 的情况）。
help([object])

调 用内置的帮助系统。（此功能是为交互使用。）如果没有给出参数，交互式帮助系统启动解释控制台。如果参数是一个字符串，然后是字符串被作为一个 module，function，class，method，keyword 或文档主题名称和帮助页面名字进行查找后在控制台上打印出来。如果参数是任何 其他类型的对象，将产生该对象的一个帮助页面。 版本 2.2 中新增.

hex(x)

Convert an integer number (of any size) to a hexadecimal string. The result is a valid Python expression. Changed in version 2.4: Formerly only returned an unsigned literal.
转换一个（任意大小）整数为十六进制字符串。其结果是一个有效的 Python 表达式。在 2.4 版本变更：原只产生一个无符号的文字。
id(object)

返回对象的``identity''。这是一个整数（或长整型），这是保证是唯一的，与对象的生命周期一样长。两个非重叠的生命周期的对象可能有相同的 ID（）值。（实现注意：这是对象的地址。）

input([prompt])

警告：此函数是不安全的，因为用户错误的输入！它期待一个有效的 Python 表达式作为输入，如果输入

语法上是无效的，将抛出 SyntaxError。如果地计算过程中有一个的错误，将抛出其他 exceptions。（另一方面，有时这是你为特殊使用需要写一个快速脚本。）

如果 readline 模块被加载，input()将使用它来提供详细行编辑和历史特性。
考虑使用 raw_input（）函数作为从用户进行一般输入。

int([x[, radix]])

转换为字符串或数字为纯整数。如果参数是一个字符串，它必须包含一个可能有符号的十进制数作为一个 Python 整 数，可能嵌入空格。以 radix 参数给出的基数为基础进行转换（这是默认 10），可以是任何在[2，36]范围内的整数，或零。如果基数为零，根据字符串 的内容猜测正确的基数。如果指定的基数 x 是不是一个字符串，引发 TypeError 异常。否则，参数可以是一个普通或长整数或浮点数。转换浮点数截断为整 数（直到零）。如果参数是整数范围之外的，将返回一个 long object。如果没有给出参数，返回 0

isinstance(object, classinfo)

返 回 true 如果该 object 参数是 classinfo 的一个实例，或其（直接或间接）子类的实例。也返回 true 如果 classinfo 是一种 type 对象（new-style class）和是该类型或其（直接或间接）子类的对象。如果 object 不是 class 一个的实例或者给定类型的对象，函数返回 false。如果 classinfo 既不是一个类的对象也不是一个 type 的对象，它可能是一个包含类或类型的对象的 tuple，也可能包含其他的递归元组（序列类型不接 受）。如果 classinfo 不是一个类，类型或元组类，类型，或者这种元组，将抛出一个 TypeError 异常。

issubclass(class, classinfo)

返回 true 如果 class 是 classinfo（直接或间接）的子类。一个类被认为是自己的子类。 classinfo 可能是类对象元组，在这种情况下元组中的每个 classinfo 项将被进行测试。在其他任何情况下，抛出一个 TypeError 异常。

iter(o[, sentinel])

返 回一个迭代器对象。第一个参数有不同的解释，视第二个参数的存在与否而定。如果没有第二个参数，o 必须是一个对象的集合，支持迭代协议（__iter__（）方法），或者它必须支持序列协议（以整数 0 开始的参数__getitem__()方法）。如果它不支持这些协议，将抛出 TypeError 异常。如果第二个参数，sentinel，给出，然后 o 必须是可调用的对象。在这种情况下创造的每一个迭代器无参调用 o 它的 next（）方法，如果返回值等于 sentinel，将抛出 StopIteration，否则将返回其它的值。

len(s)

返回一个对象的长度。参数可以是一个序列（字符串，元组或列表）或映射（词典）。

list([iterable])

返 回一个列表的 items 与可迭代的项目相同的顺序且相同的 items。可迭代可以是一个序列，一个容器，支持迭代，或一个迭代器对象。如果可迭代已经是一 个列表，将返回一个副本，类似的于 iterable[:]。例如，list('abc')返回['a', 'b', 'c']和 list((1, 2, 3)) 返回[1，2，3]。如果没有给出参数，返回一个新的空列表，[]。

locals( )

更新并返回一个代表当前 local 符号表的字典。警告：本词典的内容不应该被修改，更改可能不会影响由 interpreter 用作局部变量的值。

long([x[, radix]])

转 换字符串或数字为一个长整数。如果参数是一个字符串，它必须包含一个任意大小的可能有符号的数字，并有可能嵌入空格。radix 参数解释和 int()一 样，而且只能当 x 是一个字符串时才需要它。否则，参数可以是一个普通或长整数或浮点数，返回与其相同值的长整数。转换浮点数到截断的整数（直到零）。

如果 没有给出参数，返回 0L。

map(function, iterable, ...)

应用 function 在 iterable 的每一个项上并返回一个列表。如果有其他可迭代的参数，函数必须采取许多参数应用于来自所有 iterables 项。如果一个迭代比另一个 短，将以 None 进行扩展。如果 function 是 None，将假设为 identity function，如果有多个参数，map()返回一个列表包含所有 iterables 相应的项目的元组组成。可迭代的参数可能是一个序列或任何可迭代的 对象，结果总是一个列表。

max(iterable[, args...][key])

一个 Iterable 参数，返回其中一个最大的非空可迭代项，（如一个字符串，元组或列表）。如有多个参数，返回最大的参数。
可选的 key 参数指定带一个参数的排序函数，用于 list.sort（）。key 参数，如果有，必须在以 keyword 的形式（例如，"max(a,b,c,key=func)"）。

min(iterable[, args...][key])

一个 Iterable 参数，返回其中一个最小的非空可迭代项，（如一个字符串，元组或列表）。如有多个参数，返回最小的参数。
可选的 key 参数指定带一个参数的排序函数，用于 list.sort（）。key 参数，如果有，必须在以 keyword 的形式（例如，"max(a,b,c,key=func)"）。

object( )

返回一个新特征的对象。object 是所有 new style class 的基类。它的方法是新样式类的所有实例共有的。

oct(x)

转换一（任意大小）整数到一个八进制字符串。其结果是一个有效的 Python 表达式。

open(filename[, mode[, bufsize]])

打开一个文件，返回一个在 3.9 节中描述的文件类型的对象，`File Objects'。如果文件无法打开，IOError 异常引发。当打开一个文件，最好调用 open（），而不是直接用 file 构造。
前两个参数与 stdio 的 fopen（）函数一样：filename 是要打开的文件名，mode 是一个字符串，表示该文件是如何被打开。

mode， 最常用的值是'r'读，'w'写（文件如果已存在就截断），和'a'追加（在一些 Unix 系统意味着所有写入追加到文件尾部，无论其现在的 seek 位 置）。如果模式被省略，默认为'r'等。当打开一个二进制文件，你应该模式值加上'b'，打开二进制模式，从而提高可行性。（在某些不区分二进制文件和文本文件的系统追加‘b’，，它将作为文档）。下面是 mode 的可能值：

可选 bufsize 参数指定文件的所需缓冲区大小：0 表示无缓冲，1 表示行缓冲，任何其他的正数使用其大小（在约）的一个缓冲区。负数 bufsize，使用系统默认，这 tty 设备通常使用行缓冲和其他文件的完全缓冲。如果省略，使用系统默认。

模式'r+', 'w+'和'a+'打开文件进行更新（请注意，'w+'截断该文件）。附加'b'的模式在区分二进制和文本文件的系统上以二进制方式打开文件，系统上没有这个区别，加入了'b'没有效果。

ord(c)

给 定一个长度为 1 的字符串，返回一个整数，当参数是一个 Unicode 对象，代表字符的 Unicode 代码，或参数是一个 8 位字符串，代表其字节值，。例 如，ord('a')返回整数 97，ord(u'\u2020')返回 8224。这

是 8 位串 chr()和用于 Unicode 对象的 unichr（ ）的逆函 数。如果给出 Unicode 参数和 Python 是 UCS2 Unicode 的，字符的代码点必须在范围[0 .. 65535]内，否则字符串的长度是 2，抛出一个 TypeErro。

pow(x, y[, z])

返回 x 的 Y 次方，如果给出 z，返回 x 的 y 次方，模 Z（比 pow(x, y) % z 更有效）的。这两个参数的形式 pow(x, y)，相当于：x ** y

property([fget[, fset[, fdel[, doc]]]])

返回一个 new-style 类（从 object 派生的类）的属性。
fget 是一个获取属性值的 function，同样 fset 是设置属性值的 function，fdel 为删除属性的函数。典型的用途是定义一个托管属性 x：

```
class C(object):
    def __init__(self): self._x = None
    def getx(self): return self._x
    def setx(self, value): self._x = value
    def delx(self): del self._x
    x = property(getx, setx, delx, "I'm the 'x' property.")
```

如果给出，doc 将是该属性的文档字符串。否则，属性将复制 fget 的文档字符串（如果存在）。这使得有可能轻松地使用 property() 作为 decorator 创造只读属性：

```
class Parrot(object):
    def __init__(self):
        self._voltage = 100000

    @property
    def voltage(self):
        """Get the current voltage."""
        return self._voltage
```

原来的 voltage（ ）方法成为一个`getter'同一名称的只读属性。版本 2.2 中新增.

range([start,] stop[, step])

这 是一个通用函数来创建包含算术级数的列表，这是经常使用于循环。该参数必须是普通整数。如果 step 参数被省略，默认为 1。如果省略 start 参数，默认 为 0。完整形式是返回一个普通整数列表[start, start + step, start + 2 * step, ...]。step 不能为零（否则引发 ValueError）。例如

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
[]
```

raw_input([prompt])

如果 prompt 参数存在，它被写入到标准输出，结尾没有换行。然后函数从输入行读取，将其转换为一个字符串（去掉换行），后返回。当为 EOF，抛出 EOFError。例如：

```
>>> s = raw_input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

如果的 ReadLine 模块被加载，然后 raw_input（ ）将使用它来提供详细行编辑和历史特性。

reduce(function, iterable[, initializer])

使 用带两参数的函数从左到右计算 iterable 的项，reduce 这 iterable 得到一个数字。例如，reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) 就是计算 ((((1+2)+3)+4)+5)。左参数 x，是累加值和右边的参数，y，是 iterable 中更新的值。如果可选的 initializer 存在，在计 算中摆在可迭代的项的前面，当 iterable 为空时，作为默认。如果没有给出 initializer，则只包含一项，返回第一项。

reload(module)

重新导入先前导入的模块。该参数必须是一个模块对象，因此它之前必须已成功导入。如果您使用外部编辑器编辑源文件的模块，并想不离开 Python 解释器尝试新版本,这是有用的。返回值是模块对象（与 module 参数相同的值）。
当执行 reload(module)时：

Python 模块代码重新编译和模块级代码重新执行，定义绑定在模块词典中名称的新的对象。该扩展模块初始化函数不会第二调用。
如同 Python 中的其它所有的对象，只有引用计数下降到零，对象才会被回收。

在模块命名空间的名称更新为指向任何新的或更改过的对象。

旧对象的其他引用（如引用外部模块的名字）不会重新绑定到新的对象，必须在每个命名空间更新它们如果有需要。

还有其他一些警告：
如 果一个模块在语法上是正确的，但它的初始化失败，因为它首次导入声明并未绑定它的 locally 名字，但在 sys.modules 模块中存储了一个模块对 象（部分初始化）。要重新加载的模块，您必须先 import 一遍（这将绑定名称到部分初始化模块对象），然后才能重新加载（ ）它。

当重 新加载一个模块，它的字典（含模块的 global 变量）的保留。重新定义的名称将取代旧的定义，所以这通常不是一个问题。如果一个模块的新版本没有定义一 个由旧版本定义的名称，旧的定义仍然存在。此功能可用于模块的优势，如果它保持 global 表或对象的缓存----使用 try 语句可以测试 table 的存 在和跳过它的初始化 - 如果需要的话：

```
try:
    cache
except NameError:
    cache = {}
```

这是合法的，尽管一般对于重新加载内建或动态加载的模块不会太有用，除开 SYS，__main__和__builtin__。在许多情况下，然而，扩展模块的并不被设计成初始化一次以上，以任意方式重新加载可能失败。

如果一个模块使用 from ... import ...从另一个模块导入对象的，调用 reload()，没有从其中导入的对象 - 一个办法是重新执行 from 语句，另一个是使用 import 和限定名（module.name）。

如果一个模块实例化类的实例，重新加载的模块定义了类不会影响实例中定义方法 - 他们继续使用旧的类的定义。此对于派生类同样是如。

repr(object)

返回一个字符串，其中包含一个对象的可打印形式。有时是对能够访问一个普通的函数的操作很有用。对于许多类型，该函数使得试图返回一个字符串，会产生一个对象与传递给 eval()相同的值产生的对象一样。

reversed(seq)

返回一个反向迭代器。seq 必须是一个支持序列协议的对象（__len__（）方法和__getitem__（）以 0 开始的整数参数的方法） 版本 2.4 中新增

round(x[, n])

返回浮点值 x 四舍五入到小数点后 n 位后数字。如果 n 被省略，默认为零。结果是一个浮点数。

set([iterable])

返回其元素都是从 iterable 得到的 set。元素必须是不可改变的。如果 iterable 没有指定，返回一个新的空集，设置([]). 版本 2.4 中新增

setattr(object, name, value)

与 getattr（）相对应。该参数是一个对象，一个字符串和一个任意值。该字符串可以是现有属性名称或一个新的属性。函数分配给该属性值，只要该对象允许的话。例如，setattr(x, 'foobar', 123)，相当于 x.foobar = 123。

slice([start,] stop[, step])

返回一个切片对象，它表示的是 range(start, stop, step)指定的范围。start 和 step 参数默认为 None。切片对象有只读数据属性 start，stop 和 step，它只是返回参数值（或默认）。没有其他明确的功能，但它们的作为数值 Python 和其他第三方扩展使用。当使用扩展索引语法时也产生切片对象。例如：“a[start:stop:step]”或“a[start:stop, i]”。

sorted(iterable[, cmp[, key[, reverse]]])

返回一个新的排序的列表，包含 Iterable 的项。
可选参数 cmp，key，reverse 与 list.sort()具相同涵义（详见第 3.6.4）。

cmp 指定带两个参数（Iterable 的元素），返回自一个负数，零或正数的函数“cmp=lambda x,y: cmp(x.lower(), y.lower())“。

key 指定带一个参数的函数，用来从列表每个元素中提取一个比较 key：“key=str.lower”

reverse 是一个布尔值。如果设置为 True，则对列表中的元素进行排序，同时每一次比较都是逆向的。

一般而言，key 和 reverse 转换过程是远远快于指定一个相当于 cmp 的功能。这是因为 cmp 是为每个列表元素调用很多次，而 key 和 reverse 接触每个元素只有一次。
版本 2.4 中新增

staticmethod(function)

函数返回一个静态方法。静态方法没有接收一个隐含的第一个参数。要声明一个静态方法，如下：

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

@staticmethod 形式 是一个 function decorator

它即可以在类上如 C.f()进行调用，也可以在实例上，如：C().f()。实例被忽略，除了类。

在 Python 静态方法类似于 Java 或 C++的。对于更先进的概念，见 classmethod（ ）。

str([object])

返回一个字符串，其中包含对象可打印的描述。对于字符串，这将返回字符串本身。与 repr(object)不同的是，str(object) ）并不总是试图返回一个 eval（ ）可以接受的字符串，其目标是返回一个可打印字符串。如果没有给出参数，返回空字符串，''。
sum(iterable[, start])

求 start 和可迭代的从左至右的项和并返回总和。start 默认为 0。在可迭代的项，通常是数字，不能是字符串。快速，正确的连接的字符串序列的方法是通 过调用''.join(sequence)。注意 sum(range(n), m)相当于 reduce(operator.add, range(n), m)。 版本 2.3 中新增.

super(type[, object-or-type])

返回类型的超 类。如果第二个参数被省略，返回的超级对象是未绑定。如果第二个参数是一个对象，isinstance(obj, type)必须是 true。如果第二个参数是一个类型，issubclass(type2, type)必须是 true。super() 只能用于新型类。
一个典型用途是调用一个合适的超类的方法：

```
class C(B):
    def meth(self, arg):
        super(C, self).meth(arg)
```

请 注意，super 是作为显式的点属性绑定过程查找的一部分，例如“super(C, self).__getitem__(name)”。因此，super 是未定义对于使用语句或操作进行隐式的查找，如“super(C, self)[name]”。 版本 2.2 中新增.

tuple([iterable])

返回一个元组的 items 与可迭代的 iterable 是相同的且有相同的顺序。iterable 可能是一个序列，容器支持迭代，或迭代器对象。如果 iterable 已是 元组，直接返回。例如，tuple('abc')返回('a', 'b', 'c') 和 tuple([1, 2, 3])返回(1, 2, 3)。如果没有给出参数，返回一个新的空元组，()。

type(object)

返回对象的类型。返回值是一个类型对象。

type(name, bases, dict)

返 回一个新的类型的对象。这基本上是类声明的动态形式。该 name 字符串是类名，成为__name__的属性;bases 元组详细列明了基类，并成为 __bases__的属性，以及 dict 字典是命名空间定义为类体，成为__dict__属性。例如，下面两个语句创建相同类型的对象：

```
>>> class X(object):
...     a = 1
...
>>> X = type('X', (object,), dict(a=1))
```

版本 2.2 中新增.

unichr(i)

返回一个 Unicode 码为整数 i 的字符的 Unicode 字符串。例如，unichr（97）返回字符串 u'a'。这是 Unicode 字符串的 ord()的逆函数。参数的有效范围取决于 Python 如何被配置 - 它可以是 UCS2 [0 .. 0xFFFF 的]或 UCS4 [0 .. 0x10FFFF]。否则引发 ValueError。 版本 2.0 中新增

unicode([object[, encoding [, errors]]])

返回 object 的 Unicode 版本字符串，使用下列方式之一：
如 果给出 encoding 和/或 errors，Unicode（）将解码可以是一个 8 位字符串或使用 encoding 解码器字符缓冲区的对象。编 encoding 参数是一个编码名称的字符串;如果 encoding 不知道，抛出 LookupError。错误处理是根据 errors，errors 指定 字符是在输入编码无效时的处理方案。如果错误是'strict'（默认），引发 ValueError，而'ignore'将忽略错误，以及 'replace'值的导致官方 Unicode 替换字符，U+FFFD，用来取代输入的不能解码的字符。另见编解码器模块。

如果没有可选参数，Unicode（）将模仿 str()，但它返回 Unicode 字符串，而不是 8 位字符串。更确切地说，如果对象是一个 Unicode 字符串或其子类将返回不带任何附加解码的 Unicode 字符串。

对于对象提供 __unicode__（）方法，它将不带参数调用这种方法来创建一个 Unicode 字符串。对于其他所有对象，8 位字符串版本或请求 representation，使用编码'strict'模式的默认编解码器转换为 Unicode 字符串。

vars([object])

如果没有参数，根据现在的 local 符号表返回一个字典。如果是一个模块，类或类的实例对象作为参数（或其它任何有__dict__属性），根据对象的符号表返回一个字典。返回的字典不应被被修改：在相应符号表上的影响是未定义的。

xrange([start,] stop[, step])

这 个功能非常类似于 range（），但返回一个'xrange object'而不是一个列表。这是一个不透明的序列类型，包含相应的列表相同的值而实际上没有储存这些值。xrange 的优势（）比 range（）是很 小的（xrange（）还是要创造请求的值），除非使用一个非常大的范围内存的机器或所有元素从来没有使用过（例如循环通常是被打破终止的）。

注：xrange（）是为了简单和快速而设计的。施加了一定的限制，以实现这一目标。 Python 的 C 语言实现限制所有参数的为 native C longs（“short”Python 整数），并要求在这些 elements 都与 native C long 兼容。

zip([iterable, ...])

这 个函数返回一个元组列表，其中的第 i 元组包含每一个序列或 iterables 的第 i 个元素。返回的列表被截断长度为参数最短的序列的长度。当有多个参数是 相同长度，zip()类似于用无参数初始化的 map()。一个单独序列参数，它返回 1 个元组的列表。不带参数，它返回一个空列表。

abs

第一个是 abs

```
print 'abs(-10):'
print abs(-10)
print 'abs(-2.3):'
```

```
print abs(-2.3)
print 'abs(1+1j):'
print abs(1+1j)
```

结果是：

```
abs(-10)
10
abs(-2.3)
2.3
abs(1+1j)
1.41421356237
```

这个体现了函数的多态啊。整数，浮点数，复数进行了不同的处理。功能强吧。

ord 和 chr

第二个是 ord 和 chr

```
print 'the ASCII code of Z'
print ord('Z')
print 'the ASCII code of 90'
print chr(90)
```

结果是

```
the ASCII code of Z
90
the ASCII code of 90
Z
```

如果给 ord 的是多于一个字符的字符串，那么会有异常给你看。如果给 chr 的参数不在 0 到 255 的范围内，也会有异常的。

str, dict, tuple 和 list

其实这些都不是内置函数了，如果从传统观点来说，应该是构造函数才对。str 用来构造一个新的字符串，dict 用来构造字典，list 用来构在序列。

```
org_str = 'hello'
new_str = str(org_str)
print org_str, new_str
org_dict = {'phone':68942443, 'qq':81711392}
new_dict = dict(org_dict)
new_dict['qq']=0
print org_dict, new_dict
another_dict = org_dict
another_dict['qq']=0
print org_dict, another_dict
org_list = [1,2,3]
new_list = list(org_list)
new_list[0] = 4
print org_list, new_list
```

输出

```
hello hello
{'qq': 81711392, 'phone': 68942443} {'qq': 0, 'phone': 68942443}
```

{'qq': 0, 'phone': 68942443} {'qq': 0, 'phone': 68942443}
[1, 2, 3] [4, 2, 3]

可见，用构造函数产生新的对象然后在赋值是真正的拷贝，如果直接赋值就是同一个名称绑定到了同一个对象之上了。对于字符串，由于字符串是不可改变的，所以其实两个东西还是指向了一个对象，并没有发生想象中的拷贝过程。

cmp

```
print 'cmp(1,2)'
print cmp(1,2)
print 'cmp(1,1)'
print cmp(1,1)
print 'cmp(2,1)'
print cmp(2,1)
print 'cmp(1.2,2)'
print cmp(1.2,2)
print "cmp('a','b')"
print cmp('a','b')
print "cmp('ab','ac')"
print cmp('ab', 'ac')
```

结果是：

```
cmp(1,2)
-1
cmp(1,1)
0
cmp(2,1)
1
cmp(1.2,2)
-1
cmp('a','b')
-1
cmp('ab','ac')
-1
```

divmod

```
a = 10
b = 3
print a/b
print a%b
print divmod(a,b)
```

结果大家都猜得到：

```
3
1
(3, 1)
```

需要一个 divmod 我猜的原因是汇编中能够同时算出商和模，所以用一个这样的函数可以提高效率，免得同时需要商和模的时候算两次。充分体现了 python 能够返回两个返回值的好处啊，其实返回的是一个 tuple

hex 和 oct

```
val = int(raw_input('input a value in decimal:'))
print 'converted to hex'
print hex(val)
print 'converted to oct'
print oct(val)
```

运行结果是：

```
input a value in decimal:123
converted to hex
0x7b
converted to oct
0173
```

需要注意 hex 和 oct 返回的是字符串，而且 hex 返回的前缀是 0x，而 oct 的前缀是 0。在 python 中，如果你在数字前面加了 0 或者 0x 就表示了它们分别是 8 进制和十六进制表示的

int, long 和 float

```
a = '1'
a_val = int(a)
b = '1.1'
b_val = float(b)
print a_val+b_val
```

结果是

```
2.1
```

long 和 int 其实差不多，其实应该说 int 当数字太大的时候会调用 long。用 int 就好了，无所谓。

int 还能把字符串解释为不同的进制来进行转换，比如
int('12', 16)会得到 18，但是你不能：
int('0x12')，这个是不行的。

如果不用 int 和 float 把字符串编程数字，那么是无法作加法的。如果字符串是无法转换的那种，则会给出异常：

```
ValueError: invalid literal for int()
```

有可能会把 eval 也给扯进来，但是 eval 是把字符串当作 python 的源代码来看，尝试进行解释执行。和 int 和 float 单纯的类型转换的本意想去太远了。

input 和 raw_input

```
a = input('a string to eval:')
print a
a = raw_input('a string:')
print a
a = eval(raw_input('a string then eval:'))
print a
```

我给了几个输入，结果是：

```
a string to eval:1+2
3
a string:1+2
1+2
a string then eval:1+2
3
```

特别注意，input 由于是调用了 eval 的 raw_input，如果用于用户输入是不安全的。因为输入一句就会被当作源代码来执行，那就是很大的安全漏洞了。还是能用 raw_input 就用 raw_input。

len, max 和 min

```
a_list = [1,3,4,6,7]
print a_list
print 'length:',len(a_list)
print 'max:', max(a_list)
print 'min:', min(a_list)
a_tuple = (1,2,3)
print a_tuple
print 'length:', len(a_tuple)
print 'max:', max(a_tuple)
print 'min:', min(a_tuple)
a_string = 'hello'
print 'length:', len(a_string)
print 'max:', max(a_string)
print 'min:', min(a_string)
a_dict = {'hello':'hello!', 'how are you':'fine'}
print 'length:', len(a_dict)
print 'max:', max(a_dict)
print 'min:', min(a_dict)
```

运行结果是：

```
[1, 3, 4, 6, 7]
length: 5
max: 7
min: 1
(1, 2, 3)
length: 3
max: 3
min: 1
length: 5
max: o
min: e
length: 2
max: how are you
min: hello
```

很明白的。只是对于 dict 来说，比较大小的只是 key，value 不参加的。

filter, map, sum, zip 和 reduce

这几个函数是很有名的：

```
def my_filter(val_in_list):
    if val_in_list < 5:
```

```
        return False
    else:
        return True

def my_map(val_in_list):
    return val_in_list**2

print filter(my_filter, [1,2,3,6,7])
print map(my_map, [1,2,3])
```

结果是：

```
[6, 7]
[1, 4, 9]
```

就是定义一个自己的函数，接收从 list 中取出的每个值。然后 filter 是这样的：
如果希望这个值继续存在在过滤之后的 list 中，返回"True，否则返回 false
而 map 是返回一个新值来代替这个值，这里是用原值的平方来进行代替的。

```
sum([1,2,3])
```
结果是
6
很见的求和

```
zip([1,3,5],[2,4,6])
```
结果是
[(1, 2), (3, 4), (5, 6)]
解释配对成 tuple，参数可以是多个的 list。

```
def func(val1, val2):
    return val1*val2

print reduce(func, [1,2,3,4])
```

结果是：

```
24
```

这个比较费解。其实是 func((func((func(1,2)),3)),4)。

pow

```
print '2**3'
print 2**3
print 'pow(2,3)'
print pow(2,3)
print 'pow(2,3)%3'
print pow(2,3)%3
print 'pow(2,3,3)'
print pow(2,3,3)
```

结果是：

```
2**3
8
pow(2,3)
8
```

```
pow(2,3)%3
2
pow(2,3,3)
2
```

其实**运算就是 pow，只不过 pow 还带一个可选的第三个参数来把乘方结果进行取模运算。
值得注意的是乘方的底数和幂都能够是整数，小数，甚至是复数。但是当有第三个参数的时候，由于取模运算的限制，只能把底数和幂都限制为整数。

range

```
print range(10)
print range(1,10)
print range(1,10,3)
```

结果是

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 4, 7]
```

可见 range 是左开右闭的。而且我们使用 for i in range(xxx)的时候其实是遍历一个产生的 list。

round

```
print round(0.5)
print round(-0.5)
print round(0.55, 1)
```

结果是:

```
1.0
-1.0
0.6
```

第二个参数是用来把浮点数变成指定小数位数的。一个规律是 round 的结果总是远离 0 的。

```
>>> a=[1,2]
>>> a.append(3)#追加到链尾
>>> a
[1, 2, 3]
>>> b=[4,5]
>>> a.extend(b)#追加一个列表
>>> a
[1, 2, 3, 4, 5]
>>> a.insert(0,9)#在位置 0 插入 9
>>> a
[9, 1, 2, 3, 4, 5]
>>> a.remove(9)#删除值为"9"的元素(如果存在多个 9,则只删除第一个),如果不存在会抛出异常
>>> a
[1, 2, 3, 4, 5]
>>> a.reverse()#反转序列
>>> a
[5, 4, 3, 2, 1]
>>> a.pop(1)#返回并删除位置为 i 的元素,i 默认为最后一个元素
4
>>> a
```

```
[5, 3, 2, 1]
>>> a.index(2)#返回第一个值为'2'的元素,不存在则抛出异常
2
>>> a
[5, 3, 2, 1]
>>> a.count(5)#统计一个字符出现的次数
1
>>> a.sort()#排序数字和字母各排各的，按升序排列
>>> a
[1, 2, 3, 5]
>>> a[0:3]#包括 a[0],不包括 a[3]
[1, 2, 3]
>>> a[:-1]#包括开始,不包括最后一个
[1, 2, 3]
>>> a[-3:]#抽取最后 3 个
[2, 3, 5]
>>> a[::-1]#倒序排列
[5, 3, 2, 1]
>>> a[::]#正序排列
[1, 2, 3, 5]

>>> fruit=['  apple ',' orange  ']
>>> fruit
['  apple ', ' orange  ']
>>> [fruit.strip() for fruit in fruit]
['apple', 'orange']
>>> a
[1, 2, 3, 5]
>>> [3*x for x in a if x>3]#3*x 代表 3 乘以 x
[15]
>>> [(x,x**2) for x in a]#x**2 代表 x 的平方
[(1, 1), (2, 4), (3, 9), (5, 25)]
>>> a
[1, 2, 3, 5]
>>> b
[4, 5]
>>> [x*y for x in a for y in b]
[4, 5, 8, 10, 12, 15, 20, 25]
>>> [a[i]+b[i] for i in range(len(b))]#len(b)为 2,所以为 a[0]+b[0],a[1]+b[1]
[5, 7]
>>> [str(round(200/110.0,i)) for i in range(1,10)]#str()是转换类型为可以打印的字符,round(x,n)表示对
x 保留 n 位小数(四舍五入)
['1.8', '1.82', '1.818', '1.8182', '1.81818', '1.818182', '1.8181818', '1.81818182', '1.818181818']

>>> test='abc',123,'school'#test=('abc',123,'school')的简写
>>> x,y,z=test#拆分操作可以应用于所有 sequence
>>> x
'abc'
>>> y
123
>>> z
'school'
>>> union = test,(1,2,3)
>>> union
(('abc', 123, 'school'), (1, 2, 3))
>>> empty=()#空元组
>>> singleton='hi',#单个元素的元组,注意逗号
>>> singleton
('hi',)
>>> url=set('http://sucre.javaeye.com')
```

```
>>> url
set(['a', 'o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y', ':'])
>>> url1=set('http://hi.baidu.com/beijingalana')
>>> url1
set(['a', 'o', 'c', 'b', 'e', 'd', 'g', 'i', 'h', 'j', 'm', 'l', '/', '.', 'p', 'n', 'u', 't', ':'])
>>> url-url1#差
set(['y', 's', 'r', 'v'])
>>> url|url1#并
set(['/', 'n', 'a', 'c', 'b', 'e', 'd', 'g', 'i', 'h', 'j', 'm', 'l', 'o', '.', 'p', 's', 'r', 'u', 't', 'v', 'y', ':'])
>>> url&url1#交
set(['a', 'c', 'e', 'h', 'j', 'm', 'o', '.', 'p', 'u', '/', ':', 't'])
>>> a^b#a,b 都为元组，^对元组无效
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: unsupported operand type(s) for ^: 'list' and 'list'
>>> url
set(['a', 'o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y', ':'])
>>> url.remove(':')
>>> url
set(['a', 'o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y'])
>>> url1
set(['a', 'o', 'c', 'b', 'e', 'd', 'g', 'i', 'h', 'j', 'm', 'l', '/', '.', 'p', 'n', 'u', 't', ':'])
>>> url1.remove(':')
>>> url
set(['a', 'o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y'])
>>> url1
set(['a', 'o', 'c', 'b', 'e', 'd', 'g', 'i', 'h', 'j', 'm', 'l', '/', '.', 'p', 'n', 'u', 't'])
>>> url^url1#(并-交)
set(['n', 'b', 'd', 'g', 'i', 'l', 's', 'r', 'v', 'y'])
>>> url.add(':')
>>> url
set(['a', 'o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y', ':'])
>>> url1.add(':')
>>> url^url1
set(['n', 'b', 'd', 'g', 'i', 'l', 's', 'r', 'v', 'y'])
>>> url.pop(-1)
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: pop() takes no arguments (1 given)
>>> url.pop()#对于列表用 pop 默认去掉的竟然是第一个
'a'
>>> url
set(['o', 'c', 'e', 'h', 'j', 'm', '/', '.', 'p', 's', 'r', 'u', 't', 'v', 'y', ':'])
>>> a
[1, 2, 3, 5]
>>> a.pop()
5
```

python 中函数参数的传递是通过赋值来传递的。函数参数的使用又有俩个方面值得注意：1.函数参数是如何定义的 2.在调用函数的过程中参数是如何被解析

先看第一个问题，在 python 中函数参数的定义主要有四种方式：

1.F(arg1,arg2,...)

这 是最常见的定义方式，一个函数可以定义任意个参数，每个参数间用逗号分割，用这种方式定义的函数在调用的的时候也必须在函数名后的小括号里提供个数相等的 值（实际参数），而且顺序必须相同，也就是说在这种调用方式中，形参和实参的个数必须一致，而且必须一一对应，也就是说第一个形参对应这第一个实参。例 如：

```
def a(x,y):
 print x,y
```

调用该函数，a(1,2)则 x 取 1，y 取 2，形参与实参相对应，如果 a(1)或者 a(1,2,3)则会报错。

2.F(arg1,arg2=value2,...)
这种方式就是第一种的改进版，提供了默认值
```
def a(x,y=3):
 print x,y
```
调用该函数，a(1,2)同样还是 x 取 1，y 取 2，但是如果 a(1)，则不会报错了，这个时候 x 还是 1，y 则为默认的 3。上面这俩种方式，还可以更换参数位置，比如 a(y=8,x=3)用这种形式也是可以的。

3.F(*arg1)
上 面俩个方式是有多少个形参，就传进去多少个实参，但有时候会不确定有多少个参数，则此时第三种方式就比较有用，它以一个*加上形参名的方式来表示这个函数 的实参个数不定，可能为 0 个也可能为 n 个。注意一点是，不管有多少个，在函数内部都被存放在以形参名为标识符的 tuple 中。
```
>>> def a(*x):
if len(x)==0:
 print 'None'
else:
 print x
>>> a(1)
(1,) #存放在元组中
>>> a()
None
>>> a(1,2,3)
(1, 2, 3)
>>> a(m=1,y=2,z=3)
Traceback (most recent call last):
File "<pyshell#16>", line 1, in -toplevel-
 a(m=1,y=2,z=3)
TypeError: a() got an unexpected keyword argument 'm'
```

4.F(**arg1)
形参名前加俩个*表示，参数在函数内部将被存放在以形式名为标识符的 dictionary 中，这时调用函数的方法则需要采用 arg1=value1,arg2=value2 这样的形式。
```
>>> def a(**x):
if len(x)==0:
 print 'None'
else:
 print x
>>> a()
None
>>> a(x=1,y=2)
{'y': 2, 'x': 1} #存放在字典中
>>> a(1,2) #这种调用则报错
Traceback (most recent call last):
File "<pyshell#25>", line 1, in -toplevel-
 a(1,2)
TypeError: a() takes exactly 0 arguments (2 given)
```

上面介绍了四种定义方式，接下来看函数参数在调用过程中是怎么被解析的,其实只要记住上面这四种方法优先级依次降低，先 1，后 2，再 3，最后 4，也就是先把方式 1 中的 arg 解析，然后解析方式 2 中的 arg=value，再解析方式 3，即是把多出来的 arg 这种形式的实参组成个 tuple 传进去，最后把剩下的 key=value 这种形式的实参组成一个 dictionary 传给带俩个星号的形参，也就方式 4。

```
>>> def test(x,y=1,*a,**b):
print x,y,a,b

>>> test(1)
1 1 () {}
>>> test(1,2)
1 2 () {}
>>> test(1,2,3)
1 2 (3,) {}
>>> test(1,2,3,4)
1 2 (3, 4) {}
>>> test(x=1,y=2)
1 2 () {}
>>> test(1,a=2)
1 1 () {'a': 2}
>>> test(1,2,3,a=4)
1 2 (3,) {'a': 4}
>>> test(1,2,3,y=4)
Traceback (most recent call last):
File "<pyshell#52>", line 1, in -toplevel-
 test(1,2,3,y=4)
TypeError: test() got multiple values for keyword argument 'y'
```

特殊方法 描述
基本定制型
C.__init__(self[, arg1, ...]) 构造器（带一些可选的参数）
C.__new__(self[, arg1, ...]) 构造器（带一些可选的参数）；通常用在设置不变数据类型的子类。
C.__del__(self) 解构器
C.__str__(self) 可打印的字符输出；内建 str()及 print 语句
C.__repr__(self) 运行时的字符串输出；内建 repr() 和‘‘ 操作符
C.__unicode__(self)b Unicode 字符串输出；内建 unicode()
C.__call__(self, *args) 表示可调用的实例
C.__nonzero__(self) 为 object 定义 False 值；内建 bool()（从 2.2 版开始）
C.__len__(self) “长度”（可用于类）；内建 len()

特殊方法 描述
对象（值）比较 c
C.__cmp__(self, obj) 对象比较；内建 cmp()
C.__lt__(self, obj) and 小于/小于或等于；对应<及<=操作符
C.__gt__(self, obj) and 大于/大于或等于；对应>及>=操作符
C.__eq__(self, obj) and 等于/不等于；对应==,!=及<>操作符
属性
C.__getattr__(self, attr) 获取属性；内建 getattr()；仅当属性没有找到时调用
C.__setattr__(self, attr, val) 设置属性
C.__delattr__(self, attr) 删除属性
C.__getattribute__(self, attr) 获取属性；内建 getattr()；总是被调用
C.__get__(self, attr)（描述符）获取属性
C.__set__(self, attr, val) （描述符）设置属性
C.__delete__(self, attr) （描述符）删除属性
定制类/模拟类型

数值类型：二进制操作符
C.__*add__(self, obj) 加；＋操作符
C.__*sub__(self, obj) 减；-操作符
C.__*mul__(self, obj) 乘；*操作符
C.__*div__(self, obj) 除；/操作符
C.__*truediv__(self, obj) True 除；/操作符
C.__*floordiv__(self, obj) Floor 除；//操作符
C.__*mod__(self, obj) 取模/取余；%操作符
C.__*divmod__(self, obj) 除和取模；内建 divmod()
C.__*pow__(self, obj[, mod]) 乘幂；内建 pow();**操作符
C.__*lshift__(self, obj) 左移位；<<操作符
特殊方法 描述
定制类/模拟类型
数值类型：二进制操作符
C.__*rshift__(self, obj) 右移；>>操作符
C.__*and__(self, obj) 按位与；&操作符
C.__*or__(self, obj) 按位或；|操作符
C.__*xor__(self, obj) 按位与或；^操作符
数值类型：一元操作符
C.__neg__(self) 一元负
C.__pos__(self) 一元正
C.__abs__(self) 绝对值；内建 abs()
C.__invert__(self) 按位求反；~操作符
数值类型：数值转换
C.__complex__(self, com) 转为 complex(复数);内建 complex()
C.__int__(self) 转为 int;内建 int()
C.__long__(self) 转为 long；内建 long()
C.__float__(self) 转为 float；内建 float()
数值类型：基本表示法（String）
C.__oct__(self) 八进制表示；内建 oct()
C.__hex__(self) 十六进制表示；内建 hex()
数值类型：数值压缩
C.__coerce__(self, num) 压缩成同样的数值类型；内建 coerce()
C.__index__(self)g 在有必要时,压缩可选的数值类型为整型（比如：用于切片索引等等

序列类型
C.__len__(self) 序列中项的数目
C.__getitem__(self, ind) 得到单个序列元素
C.__setitem__(self, ind,val) 设置单个序列元素
C.__delitem__(self, ind) 删除单个序列元素
特殊方法 描述
序列类型
C.__getslice__(self, ind1,ind2) 得到序列片断
C.__setslice__(self, i1, i2,val) 设置序列片断
C.__delslice__(self, ind1,ind2) 删除序列片断
C.__contains__(self, val) f 测试序列成员；内建 in 关键字
C.__*add__(self,obj) 串连；＋操作符
C.__*mul__(self,obj) 重复；*操作符
C.__iter__(self) 创建迭代类；内建 iter()
映射类型
C.__len__(self) mapping 中的项的数目
C.__hash__(self) 散列(hash)函数值

C.__getitem__(self,key) 得到给定键(key)的值
C.__setitem__(self,key,val) 设置给定键(key)的值
C.__delitem__(self,key) 删除给定键(key)的值
C.__missing__(self,key) 给定键如果不存在字典中，则提供一个默认值