## Python 标识符

在 Python 里，标识符由字母、数字、下划线组成。

在 Python 中，所有标识符可以包括英文、数字以及下划线(_)，但不能以数字开头。

Python 中的标识符是区分大小写的。

以下划线开头的标识符是有特殊意义的。以单下划线开头 **_foo** 的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用 **from xxx import *** 而导入。

以双下划线开头的 **__foo** 代表类的私有成员，以双下划线开头和结尾的 **__foo__** 代表 Python 里特殊方法专用的标识，如 **__init__()** 代表类的构造函数。

Python 可以同一行显示多条语句，方法是用分号 **;** 分开

## Python 保留字符

下面的列表显示了在 Python 中的保留字。这些保留字不能用作常数或变数，或任何其他标识符名称。

所有 Python 的关键字只包含小写字母。

| | | |
|---|---|---|
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |
| continue | global | raise |
| def | if | return |
| del | import | try |
| elif | in | while |
| else | is | with |
| except | lambda | yield |

## 标准数据类型

Python 有五个标准的数据类型：

- Numbers（数字）
- String（字符串）
- List（列表）
- Tuple（元组）

- Dictionary（字典）

数字数据类型用于存储数值。

他们是不可改变的数据类型，这意味着改变数字数据类型会分配一个新的对象。

您可以通过使用 del 语句删除单个或多个对象的引用。例如：

del var

del var_a, var_b

Python 支持四种不同的数字类型：

- int（有符号整型）
- long（长整型[也可以代表八进制和十六进制]）
- float（浮点型）
- complex（复数）

注意：

长整型也可以使用小写 l，但是还是建议您使用大写 L，避免与数字 1 混淆。Python 使用 L 来显示长整型。

Python 还支持复数，复数由实数部分和虚数部分构成，可以用 a + bj,或者 complex(a,b) 表示， 复数的实部 a 和虚部 b 都是浮点型。

## Python 字符串

字符串或串(String)是由数字、字母、下划线组成的一串字符。

一般记为 ：

```
s="a1a2···an"(n>=0)
```

它是编程语言中表示文本的数据类型。

python 的字串列表有 2 种取值顺序:

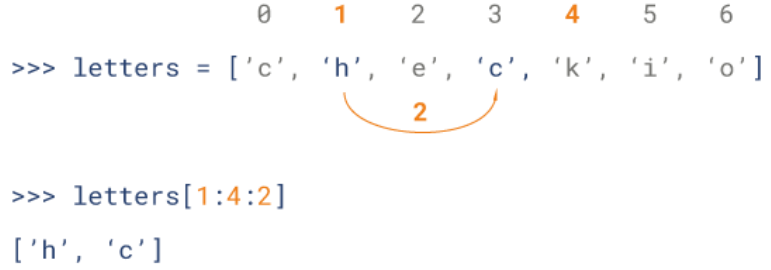
- 从左到右索引默认 0 开始的，最大范围是字符串长度少 1
- 从右到左索引默认-1 开始的，最大范围是字符串开头

如果你要实现从字符串中获取一段子字符串的话，可以使用 [头下标:尾下标] 来截取相应的字符串，其中下标是从 0 开始算起，可以是正数或负数，下标可以为空表示取到头或尾。

[头下标:尾下标] 获取的子字符串包含头下标的字符，但不包含尾下标的字符。

加号（+）是字符串连接运算符，星号（*）是重复操作

加号（+）是字符串连接运算符，星号（*）是重复操作



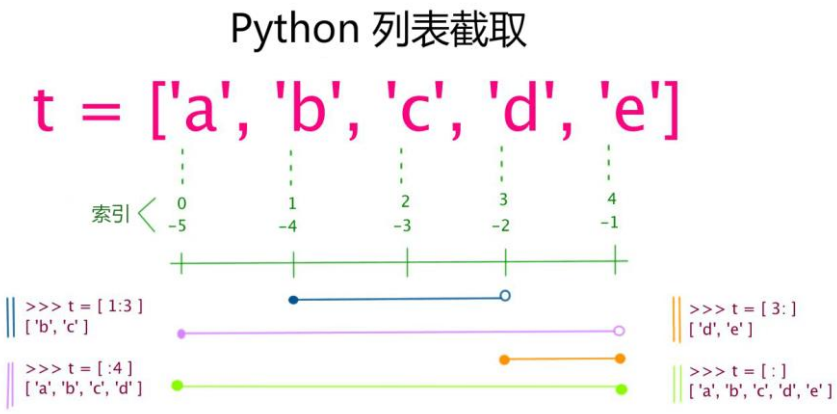
## Python 列表

List（列表） 是 Python 中使用最频繁的数据类型。

列表可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。

列表用 [ ] 标识，是 python 最通用的复合数据类型。

列表中值的切割也可以用到变量 [头下标:尾下标] ，就可以截取相应的列表，从左到右索引默认 0 开始，从右到左索引默认 -1 开始，下标可以为空表示取到头或尾。




加号 + 是列表连接运算符，星号 * 是重复操作

## Python 元组

元组是另一个数据类型，类似于 List（列表）。

元组用 () 标识。内部元素用逗号隔开。但是**元组不能二次赋值，相当于只读列表**。

## Python 字典

字典(dictionary)是除列表以外 python 之中最灵活的内置数据结构类型。列表是有序的对象集合，字典是无序的对象集合。

两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典用"{ }"标识。字典由索引(key)和它对应的值 value 组成。

```
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print tinydict.keys() # 输出所有键
print tinydict.values() # 输出所有值
```

## Python 数据类型转换

| 函数 | 描述 |
| --- | --- |
| int(x [,base]) | 将 x 转换为一个整数 |

| | |
|---|---|
| long(x [,base] ) | 将 x 转换为一个长整数 |
| float(x) | 将 x 转换到一个浮点数 |
| complex(real [,imag]) | 创建一个复数 |
| str(x) | 将对象 x 转换为字符串 |
| repr(x) | 将对象 x 转换为表达式字符串 |
| eval(str) | 用来计算在字符串中的有效 Python 表达式,并返回一个对象 |
| tuple(s) | 将序列 s 转换为一个元组 |
| list(s) | 将序列 s 转换为一个列表 |
| set(s) | 转换为可变集合 |
| dict(d) | 创建一个字典。d 必须是一个序列 (key,value)元组。 |
| frozenset(s) | 转换为不可变集合 |
| chr(x) | 将一个整数转换为一个字符 |
| unichr(x) | 将一个整数转换为 Unicode 字符 |
| ord(x) | 将一个字符转换为它的整数值 |
| hex(x) | 将一个整数转换为一个十六进制字符串 |
| oct(x) | 将一个整数转换为一个八进制字符串 |

## Python 算术运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| + | 加 - 两个对象相加 | a + b 输出结果 30 |
| - | 减 - 得到负数或是一个数减去另一个数 | a - b 输出结果 -10 |
| * | 乘 - 两个数相乘或是返回一个被重复若干次的字符串 | a * b 输出结果 200 |

| | | |
|---|---|---|
| / | 除 - x 除以 y | b / a 输出结果 2 |
| % | 取模 - 返回除法的余数 | b % a 输出结果 0 |
| ** | 幂 - 返回 x 的 y 次幂 | a**b 为 10 的 20 次方， 输出结果 100000000000000000000 |
| // | 取整除 - 返回商的整数部分（**向下取整**） | `>>> 9//2`<br>`4`<br>`>>> -9//2`<br>`-5` |

## Python 比较运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| == | 等于 - 比较对象是否相等 | (a == b) 返回 False。 |
| != | 不等于 - 比较两个对象是否不相等 | (a != b) 返回 true. |
| <> | 不等于 - 比较两个对象是否不相等 | (a <> b) 返回 true。这个运算符类似 != 。 |
| > | 大于 - 返回 x 是否大于 y | (a > b) 返回 False。 |
| < | 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。 | (a < b) 返回 true。 |
| >= | 大于等于 - 返回 x 是否大于等于 y。 | (a >= b) 返回 False。 |
| <= | 小于等于 - 返回 x 是否小于等于 y。 | (a <= b) 返回 true。 |

## Python 赋值运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| = | 简单的赋值运算符 | c = a + b 将 a + b 的运算结果赋值为 c |
| += | 加法赋值运算符 | c += a 等效于 c = c + a |
| -= | 减法赋值运算符 | c -= a 等效于 c = c - a |
| *= | 乘法赋值运算符 | c *= a 等效于 c = c * a |

| | | |
|---|---|---|
| /= | 除法赋值运算符 | c /= a 等效于 c = c / a |
| %= | 取模赋值运算符 | c %= a 等效于 c = c % a |
| **= | 幂赋值运算符 | c **= a 等效于 c = c ** a |
| //= | 取整除赋值运算符 | c //= a 等效于 c = c // a |

## Python 位运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| & | 按位与运算符：参与运算的两个值,如果两个相应位都为 1,则该位的结果为 1,否则为 0 | (a & b) 输出结果 12 ，二进制解释：0000 1100 |
| \| | 按位或运算符：只要对应的二个二进位有一个为 1 时，结果位就为 1。 | (a \| b) 输出结果 61 ，二进制解释：0011 1101 |
| ^ | 按位异或运算符：当两对应的二进位相异时，结果为 1 | (a ^ b) 输出结果 49 ，二进制解释：0011 0001 |
| ~ | 按位取反运算符：对数据的每个二进制位取反,即把 1 变为 0,把 0 变为 1 。**~x** 类似于 **-x-1** | (~a ) 输出结果 -61 ，二进制解释：1100 0011，在一个有符号二进制数的补码形式。 |
| << | 左移动运算符：运算数的各二进位全部左移若干位，由 **<<** 右边的数字指定了移动的位数，高位丢弃，低位补 0。 | a << 2 输出结果 240 ，二进制解释：1111 0000 |
| >> | 右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，**>>** 右边的数字指定了移动的位数 | a >> 2 输出结果 15 ，二进制解释：0000 1111 |

## Python 逻辑运算符

| 运算符 | 逻辑表达式 | 描述 | 实例 |
|---|---|---|---|
| and | x and y | 布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。 | (a and b) 返回 20。 |

| | | | |
|---|---|---|---|
| or | x or y | 布尔"或" - 如果 x 是非 0，它返回 x 的值，否则它返回 y 的计算值。 | (a or b) 返回 10。 |
| not | not x | 布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。 | not(a and b) 返回 False |

## Python 成员运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| in | 如果在指定的序列中找到值返回 True，否则返回 False。 | x 在 y 序列中 ，如果 x 在 y 序列中返回 True。 |
| not in | 如果在指定的序列中没有找到值返回 True，否则返回 False。 | x 不在 y 序列中 ，如果 x 不在 y 序列中返回 True。 |

## Python 身份运算符

| 运算符 | 描述 | 实例 |
|---|---|---|
| is | is 是判断两个标识符是不是引用自一个对象 | **x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True, 否则返回 False |
| is not | is not 是判断两个标识符是不是引用自不同对象 | **x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True, 否则返回 False。 |

**注：** id() 函数用于获取对象内存地址。

*is 与 == 区别：*

*is 用于判断两个变量引用对象是否为同一个， == 用于判断引用变量的值是否相等。*

## Python 运算符优先级

以下表格列出了从最高到最低优先级的所有运算符：

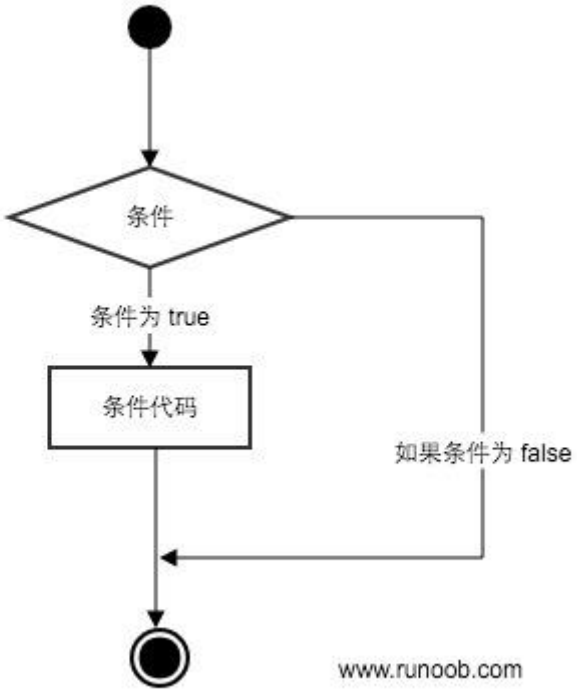
| 运算符 | 描述 |
|---|---|
| ** | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| * / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |

| | |
|---|---|
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ \| | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += *= **= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| not and or | 逻辑运算符 |

## Python 条件语句

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。

可以通过下图来简单了解条件语句的执行过程:



Python 程序语言指定任何非 0 和非空（null）值为 true，0 或者 null 为 false。

## Python 循环语句

在 python 中，while … else 在循环条件为 false 时执行 else 语句块

你可以在循环体内嵌入其他的循环体，如在 while 循环中可以嵌入 for 循环， 反之，你可以在 for 循环中嵌入 while 循环。

## Python Number(数字)

Python Number 数据类型用于存储数值。

数据类型是不允许改变,这就意味着如果改变 Number 数据类型的值，将重新分配内存空间。

Python 支持四种不同的数值类型：

- 整型(Int) - 通常被称为是整型或整数，是正或负整数，不带小数点。
- 长整型(long integers) - 无限大小的整数，整数最后是一个大写或小写的 L。
- 浮点型(floating point real values) - 浮点型由整数部分与小数部分组成，浮点型也可以使用科学计数法表示（2.5e2 = 2.5 x 102 = 250）
- 复数(complex numbers) - 复数由实数部分和虚数部分构成，可以用 a + bj,或者 complex(a,b)表示， 复数的实部 a 和虚部 b 都是浮点型。

长整型也可以使用小写"L"，但是还是建议您使用大写"L"，避免与数字"1"混淆。Python 使用"L"来显示长整型。

Python 还支持复数，复数由实数部分和虚数部分构成，可以用 a + bj,或者 complex(a,b)表示， 复数的实部 a 和虚部 b 都是浮点型

## Python Number 类型转换

int(x [,base ]) 将 x 转换为一个整数

long(x [,base ]) 将 x 转换为一个长整数

float(x ) 将 x 转换到一个浮点数

complex(real [,imag ]) 创建一个复数

str(x ) 将对象 x 转换为字符串

repr(x ) 将对象 x 转换为表达式字符串

eval(str ) 用来计算在字符串中的有效 Python 表达式,并返回一个对象

tuple(s ) 将序列 s 转换为一个元组

list(s ) 将序列 s 转换为一个列表

chr(x ) 将一个整数转换为一个字符

unichr(x ) 将一个整数转换为 Unicode 字符

ord(x ) 将一个字符转换为它的整数值

hex(x ) 将一个整数转换为一个十六进制字符串

oct(x ) 将一个整数转换为一个八进制字符串

## Python math 模块、cmath 模块

Python 中数学运算常用的函数基本都在 math 模块、cmath 模块中。

Python math 模块提供了许多对浮点数的数学运算函数。

Python cmath 模块包含了一些用于复数运算的函数。

cmath 模块的函数跟 math 模块函数基本一致，区别是 cmath 模块运算的是复数，math 模块运算的是数学运算。

要使用 math 或 cmath 函数必须先导入

## Python 数学函数

| 函数 | 返回值 ( 描述 ) |
|---|---|
| abs(x) | 返回数字的绝对值，如 abs(-10) 返回 10 |
| ceil(x) | 返回数字的上入整数，如 math.ceil(4.1) 返回 5 |
| cmp(x, y) | 如果 x < y 返回 -1，如果 x == y 返回 0，如果 x > y 返回 1 |
| exp(x) | 返回 e 的 x 次幂($e^x$),如 math.exp(1) 返回 2.718281828459045 |
| fabs(x) | 返回数字的绝对值，如 math.fabs(-10) 返回 10.0 |
| floor(x) | 返回数字的下舍整数，如 math.floor(4.9)返回 4 |
| log(x) | 如 math.log(math.e)返回 1.0,math.log(100,10)返回 2.0 |
| log10(x) | 返回以 10 为基数的 x 的对数，如 math.log10(100)返回 2.0 |
| max(x1, x2,...) | 返回给定参数的最大值，参数可以为序列。 |
| min(x1, x2,...) | 返回给定参数的最小值，参数可以为序列。 |
| modf(x) | 返回 x 的整数部分与小数部分，两部分的数值符号与 x 相同，整数部分以浮点型表示。 |
| pow(x, y) | x**y 运算后的值。 |
| round(x [,n]) | 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。 |
| sqrt(x) | 返回数字 x 的平方根 |

## Python 随机数函数

Python 包含以下常用随机数函数：

| 函数 | 描述 |
| --- | --- |
| choice(seq) | 从序列的元素中随机挑选一个元素，比如 random.choice(range(10))，从 0 到 9 中随机挑选一个整数。 |
| randrange ([start,] stop [,step]) | 从指定范围内，按指定基数递增的集合中获取一个随机数，基数缺省值为 1 |
| random() | 随机生成下一个实数，它在[0,1)范围内。 |
| seed([x]) | 改变随机数生成器的种子 seed。如果你不了解其原理，你不必特别去设定 seed，Python 会帮你选择 seed。 |
| shuffle(lst) | 将序列的所有元素随机排序 |
| uniform(x, y) | 随机生成下一个实数，它在[x,y]范围内。 |

## Python 三角函数

Python 包括以下三角函数：

| 函数 | 描述 |
| --- | --- |
| acos(x) | 返回 x 的反余弦弧度值。 |
| asin(x) | 返回 x 的反正弦弧度值。 |
| atan(x) | 返回 x 的反正切弧度值。 |
| atan2(y, x) | 返回给定的 X 及 Y 坐标值的反正切值。 |
| cos(x) | 返回 x 的弧度的余弦值。 |
| hypot(x, y) | 返回欧几里德范数 sqrt(x*x + y*y)。 |
| sin(x) | 返回的 x 弧度的正弦值。 |
| tan(x) | 返回 x 弧度的正切值。 |
| degrees(x) | 将弧度转换为角度,如 degrees(math.pi/2) ， 返回 90.0 |

| | |
|---|---|
| radians(x) | 将角度转换为弧度 |

## Python 数学常量

| 常量 | 描述 |
|---|---|
| pi | 数学常量 pi（圆周率，一般以 π 来表示） |
| e | 数学常量 e，e 即自然常数（自然常数）。 |

## Python 字符串

字符串是 Python 中最常用的数据类型。我们可以使用引号('或")来创建字符串。

创建字符串很简单，只要为变量分配一个值即可。例如：

var1 = 'Hello World!'

var2 = "Python Runoob"

## Python 访问字符串中的值

Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

Python 访问子字符串，可以使用方括号来截取字符串。

## Python 字符串运算符

下表实例变量 a 值为字符串 "Hello"，b 变量值为 "Python"：

| 操作符 | 描述 | 实例 |
|---|---|---|
| + | 字符串连接 | `>>>a + b 'HelloPython'` |
| * | 重复输出字符串 | `>>>a * 2 'HelloHello'` |
| [] | 通过索引获取字符串中字符 | `>>>a[1] 'e'` |
| [ : ] | 截取字符串中的一部分 | `>>>a[1:4] 'ell'` |
| in | 成员运算符 - 如果字符串中包含给定的字符返回 True | `>>>"H" in a True` |
| not in | 成员运算符 - 如果字符串中不包含给定的字符返回 True | `>>>"M" not in a True` |

Python 支持格式化字符串的输出 。尽管这样可能会用到非常复杂的表达式，但最基本的用法是将一个值插入到一个有字符串格式符 %s 的字符串中。

在 Python 中，字符串格式化使用与 C 中 sprintf 函数一样的语法。

如下实例：

#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)

以上实例输出结果：

My name is Zara and weight is 21 kg!

python 字符串格式化符号:

| 符　号 | 描述 |
| --- | --- |
| %c | 格式化字符及其 ASCII 码 |
| %s | 格式化字符串 |
| %d | 格式化整数 |
| %u | 格式化无符号整型 |
| %o | 格式化无符号八进制数 |
| %x | 格式化无符号十六进制数 |
| %X | 格式化无符号十六进制数（大写） |
| %f | 格式化浮点数字，可指定小数点后的精度 |
| %e | 用科学计数法格式化浮点数 |
| %E | 作用同%e，用科学计数法格式化浮点数 |
| %g | %f 和%e 的简写 |
| %G | %f 和 %E 的简写 |
| %p | 用十六进制数格式化变量的地址 |

格式化操作符辅助指令:

| 符号 | 功能 |
| --- | --- |
| * | 定义宽度或者小数点精度 |
| - | 用做左对齐 |
| + | 在正数前面显示加号( + ) |
| <sp> | 在正数前面显示空格 |
| # | 在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X') |

| | |
|---|---|
| 0 | 显示的数字前面填充'0'而不是默认的空格 |
| % | '%%'输出一个单一的'%' |
| (var) | 映射变量(字典参数) |
| m.n. | m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话) |

## Unicode 字符串

Python 中定义一个 Unicode 字符串和定义一个普通字符串一样简单：

>>> u'Hello World !'

u'Hello World !'

引号前小写的"u"表示这里创建的是一个 Unicode 字符串。如果你想加入一个特殊字符，可以使用 Python 的 Unicode-Escape 编码。如下例所示：

>>> u'Hello\u0020World !'

u'Hello World !'

被替换的 \u0020 标识表示在给定位置插入编码值为 0x0020 的 Unicode 字符（空格符）。

## python 的字符串内建函数

| 方法 | 描述 |
|---|---|
| string.capitalize() | 把字符串的第一个字符大写 |
| string.center(width) | 返回一个原字符串居中,并使用空格填充至长度 width 的新字符串 |
| **string.count(str, beg=0, end=len(string))** | 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数 |
| string.decode(encoding='UTF-8', errors='strict') | 以 encoding 指定的编码格式解码 string，如果出错默认报一个 ValueError 的异常，除非 errors 指定的是 'ignore' 或者'replace' |

| | |
|---|---|
| string.encode(encoding='UTF-8', errors='strict') | 以 encoding 指定的编码格式编码 string，如果出错默认报一个 ValueError 的异常，除非 errors 指定的是'ignore'或者'replace' |
| **string.endswith(obj, beg=0, end=len(string))** | 检查字符串是否以 obj 结束，如果 beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False. |
| string.expandtabs(tabsize=8) | 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8。 |
| **string.find(str, beg=0, end=len(string))** | 检测 str 是否包含在 string 中，如果 beg 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1 |
| **string.format()** | 格式化字符串 |
| **string.index(str, beg=0, end=len(string))** | 跟 find()方法一样，只不过如果 str 不在 string 中会报一个异常. |
| string.isalnum() | 如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回 False |
| string.isalpha() | 如果 string 至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False |
| string.isdecimal() | 如果 string 只包含十进制数字则返回 True 否则返回 False. |
| string.isdigit() | 如果 string 只包含数字则返回 True 否则返回 False. |

| | |
|---|---|
| string.islower() | 如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False |
| string.isnumeric() | 如果 string 中只包含数字字符，则返回 True，否则返回 False |
| string.isspace() | 如果 string 中只包含空格，则返回 True，否则返回 False. |
| string.istitle() | 如果 string 是标题化的(见 title())则返回 True，否则返回 False |
| string.isupper() | 如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False |
| **string.join(seq)** | 以 string 作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串 |
| string.ljust(width) | 返回一个原字符串左对齐,并使用空格填充至长度 width 的新字符串 |
| string.lower() | 转换 string 中所有大写字符为小写. |
| string.lstrip() | 截掉 string 左边的空格 |
| string.maketrans(intab, outtab]) | maketrans() 方法用于创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。 |
| max(str) | 返回字符串 *str* 中最大的字母。 |
| min(str) | 返回字符串 *str* 中最小的字母。 |
| **string.partition(str)** | 有点像 find()和 split()的结合体,从 str 出现的第一个位置起,把 字 符 串 string 分 成 一 个 3 元 素 的 元 组 (string_pre_str,str,string_post_str), 如果 string 中不包含 str 则 string_pre_str == string. |

| | |
|---|---|
| **string.replace(str1, str2, num=string.count(str1))** | 把 string 中的 str1 替换成 str2,如果 num 指定，则替换不超过 num 次. |
| string.rfind(str, beg=0,end=len(string) ) | 类似于 find()函数，不过是从右边开始查找. |
| string.rindex( str, beg=0,end=len(string)) | 类似于 index()，不过是从右边开始. |
| string.rjust(width) | 返回一个原字符串右对齐,并使用空格填充至长度 width 的新字符串 |
| string.rpartition(str) | 类似于 partition()函数,不过是从右边开始查找 |
| string.rstrip() | 删除 string 字符串末尾的空格. |
| **string.split(str="", num=string.count(str))** | 以 str 为分隔符切片 string，如果 num 有指定值，则仅分隔 num+ 个子字符串 |
| string.splitlines([keepends]) | 按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。 |
| string.startswith(obj, beg=0,end=len(string)) | 检查字符串是否是以 obj 开头，是则返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查. |
| **string.strip([obj])** | 在 string 上执行 lstrip()和 rstrip() |
| string.swapcase() | 翻转 string 中的大小写 |

| | |
|---|---|
| string.title() | 返回"标题化"的 string,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle()) |
| **string.translate(str, del="")** | 根据 str 给出的表(包含 256 个字符)转换 string 的字符,<br>要过滤掉的字符放到 del 参数中 |
| string.upper() | 转换 string 中的小写字母为大写 |
| string.zfill(width) | 返回长度为 width 的字符串，原字符串 string 右对齐，前面填充 0 |

## Python 列表(List)

序列是 Python 中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是 0，第二个索引是 1，依此类推。

Python 有 6 个序列的内置类型，但最常见的是列表和元组。

序列都可以进行的操作包括索引，切片，加，乘，检查成员。

此外，Python 已经内置确定序列的长度以及确定最大和最小的元素的方法。

列表是最常用的 Python 数据类型，它可以作为一个方括号内的逗号分隔值出现。

**列表的数据项不需要具有相同的类型**

创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。如下所示：

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
```

与字符串的索引一样，**列表索引从 0 开始。列表可以进行截取、组合等**。

## 访问列表中的值

你可以对列表的数据项进行修改或更新，你也可以使用 append()方法来添加列表项

## 删除列表元素

可以使用 del 语句来删除列表的元素

## Python 列表函数&方法

| 序号 | 函数 |
|---|---|
| 1 | cmp(list1, list2)<br>比较两个列表的元素 |

| | |
|---|---|
| 2 | len(list)<br>列表元素个数 |
| 3 | max(list)<br>返回列表元素最大值 |
| 4 | min(list)<br>返回列表元素最小值 |
| 5 | list(seq)<br>将元组转换为列表 |

| 序号 | 方法 |
|---|---|
| 1 | list.append(obj)<br>在列表末尾添加新的对象 |
| 2 | list.count(obj)<br>统计某个元素在列表中出现的次数 |
| 3 | list.extend(seq)<br>在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表） |
| 4 | list.index(obj)<br>从列表中找出某个值第一个匹配项的索引位置 |
| 5 | list.insert(index, obj)<br>将对象插入列表 |
| 6 | list.pop([index=-1])<br>移除列表中的一个元素（默认最后一个元素），并且返回该元素的值 |
| 7 | list.remove(obj)<br>移除列表中某个值的第一个匹配项 |
| 8 | list.reverse()<br>反向列表中元素 |
| 9 | list.sort(cmp=None, key=None, reverse=False)<br>对原列表进行排序 |

# Python 元组

Python 的元组与列表类似，不同之处在于**元组的元素不能修改**。

元组使用小括号，列表使用方括号。

元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

元组中只包含一个元素时，需要在元素后面添加逗号

```
tup1 = (50,)
```

元组与字符串类似，下标索引从 0 开始，可以进行截取，组合等。

## 访问元组

元组可以使用下标索引来访问元组中的值，如下实例:

```
#!/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

以上实例输出结果：

```
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
```

## 修改元组

元组中的元素值是不允许修改的，但我们可以对元组进行连接组合.

## 删除元组

元组中的元素值是不允许删除的，但我们可以使用 del 语句来删除整个元组.

## 元组运算符

与字符串一样，元组之间可以使用 + 号和 * 号进行运算。这就意味着他们可以组合和复制，运算后会生成一个新的元组。

| Python 表达式 | 结果 | 描述 |
| --- | --- | --- |
| len((1, 2, 3)) | 3 | 计算元素个数 |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | 连接 |
| ('Hi!',) * 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | 复制 |
| 3 in (1, 2, 3) | True | 元素是否存在 |

| | | |
|---|---|---|
| for x in (1, 2, 3): print x, | 1 2 3 | 迭代 |

## 元组内置函数

| 序号 | 方法及描述 |
|---|---|
| 1 | cmp(tuple1, tuple2)<br>比较两个元组元素。 |
| 2 | len(tuple)<br>计算元组元素个数。 |
| 3 | max(tuple)<br>返回元组中元素最大值。 |
| 4 | min(tuple)<br>返回元组中元素最小值。 |
| 5 | tuple(seq)<br>将列表转换为元组。 |

## Python 字典(Dictionary)

字典是另一种可变容器模型，且可**存储任意类型对象**。

字典的每个键值 key=>value 对用冒号 : 分割，每个键值对之间用逗号 , 分割，整个字典包括在花括号 {} 中 ,格式如下所示：

d = {key1 : value1, key2 : value2 }

键一般是唯一的，如果重复，最后的一个键值对会替换前面的，值不需要唯一。

>>>dict = {'a': 1, 'b': 2, 'b': '3'}

>>> dict['b'] '3'

>>> dict {'a': 1, 'b': '3'}

值可以取任何数据类型，但**键必须是不可变的**，如字符串，数字或元组。

一个简单的字典实例：

dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}

也可如此创建字典：

dict1 = { 'abc': 456 } dict2 = { 'abc': 123, 98.6: 37 }

## 修改字典

向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对。

## 删除字典元素

能删单一的元素也能清空字典，清空只需一项操作。

显示删除一个字典用 del 命令

del dict['Name'] # 删除键是'Name'的条目

dict.clear() # 清空词典所有条目

del dict # 删除词典

## 字典键的特性

字典值可以没有限制地取任何 python 对象，既可以是标准的对象，也可以是用户定义的，但键不行。

两个重要的点需要记住：

1）不允许同一个键出现两次。创建时如果同一个键被赋值两次，后一个值会被记住

2）键必须不可变，所以可以用数字，字符串或元组充当，所以**用列表就不行**

## 字典内置函数&方法

| 序号 | 函数及描述 |
| --- | --- |
| 1 | cmp(dict1, dict2)<br>比较两个字典元素。 |
| 2 | len(dict)<br>计算字典元素个数，即键的总数。 |
| 3 | str(dict)<br>输出字典可打印的字符串表示。 |
| 4 | type(variable)<br>返回输入的变量类型，如果变量是字典就返回字典类型。 |

| 序号 | 函数及描述 |
| --- | --- |
| 1 | dict.clear()<br>删除字典内所有元素 |
| 2 | dict.copy()<br>返回一个字典的浅复制 |
| 3 | dict.fromkeys(seq[, val])<br>创建一个新字典，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 |

| | |
|---|---|
| 4 | dict.get(key, default=None)<br>返回指定键的值，如果值不在字典中返回 default 值 |
| 5 | dict.has_key(key)<br>如果键在字典 dict 里返回 true，否则返回 false |
| 6 | dict.items()<br>以列表返回可遍历的(键，值) 元组数组 |
| 7 | dict.keys()<br>以列表返回一个字典所有的键 |
| 8 | dict.setdefault(key, default=None)<br>和 get()类似，但如果键不存在于字典中，将会添加键并将值设为 default |
| 9 | dict.update(dict2)<br>把字典 dict2 的键/值对更新到 dict 里 |
| 10 | dict.values()<br>以列表返回字典中的所有值 |
| 11 | pop(key[,default])<br>删除字典给定键 key 所对应的值，返回值为被删除的值。key 值必须给出。 否则，返回 default 值。 |
| 12 | popitem()<br>随机返回并删除字典中的一对键和值。 |

## Python 日期和时间

Python 程序能用很多方式处理日期和时间，转换日期格式是一个常见的功能。

Python 提供了一个 time 和 calendar 模块可以用于格式化日期和时间。

时间间隔是以秒为单位的浮点小数。

每个时间戳都以自从 1970 年 1 月 1 日午夜（历元）经过了多长时间来表示。

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:

```
#!/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：

当前时间戳为: 1459994552.51

## 什么是时间元组？

| 序号 | 字段 | 值 |
| --- | --- | --- |
| 0 | 4 位数年 | 2008 |
| 1 | 月 | 1 到 12 |
| 2 | 日 | 1 到 31 |
| 3 | 小时 | 0 到 23 |
| 4 | 分钟 | 0 到 59 |
| 5 | 秒 | 0 到 61 (60 或 61 是闰秒) |
| 6 | 一周的第几日 | 0 到 6 (0 是周一) |
| 7 | 一年的第几日 | 1 到 366 (儒略历) |
| 8 | 夏令时 | -1, 0, 1, -1 是决定是否为夏令时的旗帜 |

| 序号 | 属性 | 值 |
| --- | --- | --- |
| 0 | tm_year | 2008 |
| 1 | tm_mon | 1 到 12 |
| 2 | tm_mday | 1 到 31 |
| 3 | tm_hour | 0 到 23 |
| 4 | tm_min | 0 到 59 |
| 5 | tm_sec | 0 到 61 (60 或 61 是闰秒) |
| 6 | tm_wday | 0 到 6 (0 是周一) |
| 7 | tm_yday | 1 到 366(儒略历) |

| | | |
|---|---|---|
| 8 | tm_isdst | -1, 0, 1, -1 是决定是否为夏令时的旗帜 |

## 获取当前时间

从返回浮点数的时间戳方式向时间元组转换，只要将浮点数传递给如 localtime 之类的函数。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import time localtime = time.localtime(time.time())
print "本地时间为 :", localtime
```

以上实例输出结果：

```
本地时间为 : time.struct_time(tm_year=2016, tm_mon=4, tm_mday=7, tm_hour=10, tm_min=3, tm_sec=27, tm_wday=3, tm_yday=98, tm_isdst=0)
```

## 获取格式化的时间

你可以根据需求选取各种格式，但是最简单的获取可读的时间模式的函数是 asctime():

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import time localtime = time.asctime( time.localtime(time.time()) )
print "本地时间为 :", localtime
```

以上实例输出结果：

```
本地时间为 : Thu Apr  7 10:05:21 2016
```

## 格式化日期

我们可以使用 time 模块的 strftime 方法来格式化日期，：

```
time.strftime(format[, t])
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import time # 格式化成 2016-03-20 11:45:39 形式
print time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
# 格式化成 Sat Mar 28 22:24:24 2016 形式
print time.strftime("%a %b %d %H:%M:%S %Y", time.localtime())
# 将格式字符串转换为时间戳
a = "Sat Mar 28 22:24:24 2016" print time.mktime(time.strptime(a,"%a %b %d %H:%M:%S %Y"))
```

## 获取某月日历

Calendar 模块有很广泛的方法用来处理年历和月历，例如打印某月的月历：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import calendar cal = calendar.month(2016, 1)
print "以下输出 2016 年 1 月份的日历:" print cal
```

以上实例输出结果：

```
以下输出 2016 年 1 月份的日历:
      January 2016
Mo Tu We Th Fr Sa Su
                 1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

## Time 模块

Time 模块包含了以下内置函数，既有时间处理的，也有转换时间格式的：

| 序号 | 函数及描述 |
|---|---|
| 1 | time.altzone<br>返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。 |
| 2 | time.asctime([tupletime])<br>接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008 年 12 月 11 日 周二 18 时 07 分 14 秒）的 24 个字符的字符串。 |
| 3 | time.clock( )<br>用以浮点数计算的秒数返回当前的 CPU 时间。用来衡量不同程序的耗时，比 time.time()更有用。 |
| 4 | time.ctime([secs])<br>作用相当于 asctime(localtime(secs))，未给参数相当于 asctime() |
| 5 | time.gmtime([secs])<br>接收时间戳（1970 纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组 t。注：t.tm_isdst 始终为 0 |

| | |
|---|---|
| 6 | time.localtime([secs])<br>接收时间戳（1970 纪元后经过的浮点秒数）并返回当地时间下的时间元组 t（t.tm_isdst 可取 0 或 1，取决于当地当时是不是夏令时）。 |
| 7 | time.mktime(tupletime)<br>接受时间元组并返回时间戳（1970 纪元后经过的浮点秒数）。 |
| 8 | time.sleep(secs)<br>推迟调用线程的运行，secs 指秒数。 |
| 9 | time.strftime(fmt[,tupletime])<br>接收以时间元组，并返回以可读字符串表示的当地时间，格式由 fmt 决定。 |
| 10 | time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')<br>根据 fmt 的格式把一个时间字符串解析为时间元组。 |
| 11 | time.time( )<br>返回当前时间的时间戳（1970 纪元后经过的浮点秒数）。 |
| 12 | time.tzset()<br>根据环境变量 TZ 重新初始化时间相关设置。 |

Time 模块包含了以下 2 个非常重要的属性：

| 序号 | 属性及描述 |
|---|---|
| 1 | **time.timezone**<br>属性 time.timezone 是当地时区（未启动夏令时）距离格林威治的偏移秒数（>0，美洲;<=0 大部分欧洲，亚洲，非洲）。 |
| 2 | **time.tzname**<br>属性 time.tzname 包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。 |

## 日历（Calendar）模块

此模块的函数都是日历相关的，例如打印某月的字符月历。

星期一是默认的每周第一天，星期天是默认的最后一天。更改设置需调用 calendar.setfirstweekday()函数。模块包含了以下内置函数：

| 序号 | 函数及描述 |
| --- | --- |
| 1 | **calendar.calendar(year,w=2,l=1,c=6)**<br>返回一个多行字符串格式的 year 年年历，3 个月一行，间隔距离为 c。 每日宽度间隔为 w 字符。每行长度为 21* W+18+2* C。l 是每星期行数。 |
| 2 | **calendar.firstweekday( )**<br>返回当前每周起始日期的设置。默认情况下，首次载入 caendar 模块时返回 0，即星期一。 |
| 3 | **calendar.isleap(year)**<br><br>是闰年返回 True，否则为 False。<br>`>>> import calendar`<br>`>>> print(calendar.isleap(2000))`<br>`True`<br>`>>> print(calendar.isleap(1900))`<br>`False` |
| 4 | **calendar.leapdays(y1,y2)**<br>返回在 Y1，Y2 两年之间的闰年总数。 |
| 5 | **calendar.month(year,month,w=2,l=1)**<br>返回一个多行字符串格式的 year 年 month 月日历，两行标题，一周一行。每日宽度间隔为 w 字符。每行的长度为 7* w+6。l 是每星期的行数。 |
| 6 | **calendar.monthcalendar(year,month)**<br>返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year 年 month 月外的日期都设为 0;范围内的日子都由该月第几日表示，从 1 开始。 |
| 7 | **calendar.monthrange(year,month)**<br>返回两个整数。第一个是该月的星期几的日期码，第二个是该月的日期码。日从 0（星期一）到 6（星期日）;月从 1 到 12。 |
| 8 | **calendar.prcal(year,w=2,l=1,c=6)**<br>相当于 print calendar.calendar(year,w,l,c). |
| 9 | **calendar.prmonth(year,month,w=2,l=1)**<br>相当于 print calendar.calendar（year，w，l，c）。 |

| | |
|---|---|
| 10 | **calendar.setfirstweekday(weekday)**<br>设置每周的起始日期码。0（星期一）到 6（星期日）。 |
| 11 | **calendar.timegm(tupletime)**<br>和 time.gmtime 相反：接受一个时间元组形式，返回该时刻的时间戳（1970 纪元后经过的浮点秒数）。 |
| 12 | **calendar.weekday(year,month,day)**<br>返回给定日期的日期码。0（星期一）到 6（星期日）。月份为 1（一月） 到 12（12 月）。 |

# Python 函数

**可更改(mutable)与不可更改(immutable)对象**

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- 不可变类型：变量赋值 a=5 后再赋值 a=10，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变 a 的值，相当于新生成了 a。
- 可变类型：变量赋值 la=[1,2,3,4] 后再赋值 la[2]=5 则是将 list la 的第三个元素值更改，本身 la 没有动，只是其内部的一部分值被修改了。

python 函数的参数传递：

- **不可变类型：**类似 c++ 的值传递，如 整数、字符串、元组。如 fun（a），传递的只是 a 的值，没有影响 a 对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。
- **可变类型：**类似 c++ 的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后 fun 外部的 la 也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说**传不可变对象和传可变对象**。

# 参数

## 必备参数

必备参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

## 关键字参数

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。

使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

## 默认参数

调用函数时，默认参数的值如果没有传入，则被认为是默认值。下例会打印默认的 age，如果 age 没有被传入：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
#可写函数说明
def printinfo( name, age = 35 ):
  "打印任何传入的字符串"
```

```
  print "Name: ", name;
  print "Age ", age;
  return;

#调用 printinfo 函数
printinfo( age=50, name="miki" );
printinfo( name="miki" );
```

### 不定长参数

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述 2 种参数不同，声明时不会命名。基本语法如下：

```
def functionname([formal_args,] *var_args_tuple ):
"函数_文档字符串"
    function_suite
    return [expression]
```

加了星号（*）的变量名会存放所有未命名的变量参数。不定长参数实例如下：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
# 可写函数说明
def printinfo( arg1, *vartuple ):
    "打印任何传入的参数"
    print "输出: "
    print arg1
    for var in vartuple:
        print var
    return;
# 调用 printinfo 函数
printinfo( 10 );
printinfo( 70, 60, 50 );
```

## 匿名函数

python 使用 lambda 来创建匿名函数。

- lambda 只是一个表达式，函数体比 def 简单很多。
- lambda 的主体是一个表达式，而不是一个代码块。仅仅能在 lambda 表达式中封装有限的逻辑进去。
- lambda 函数拥有自己的命名空间，且不能访问自有参数列表之外或全局命名空间里的参数。
- 虽然 lambda 函数看起来只能写一行，却不等同于 C 或 C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

### 语法

lambda 函数的语法只包含一个语句，如下：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
```

```
# 可写函数说明
sum = lambda arg1, arg2: arg1 + arg2;
# 调用 sum 函数
print "相加后的值为 : ", sum( 10, 20 )
print "相加后的值为 : ", sum( 20, 20 )
```

## 变量作用域

一个程序的所有的变量并不是在哪个位置都可以访问的。访问权限决定于这个变量是在哪里赋值的。
变量的作用域决定了在哪一部分程序你可以访问哪个特定的变量名称。两种最基本的变量作用域如下：

- 全局变量
- 局部变量

### 全局变量和局部变量

定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。
局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入到作用域中。

## Python 模块

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和 Python 语句。
模块让你能够有逻辑地组织你的 Python 代码段。
把相关的代码分配到一个模块里能让你的代码更好用，更易懂。
模块能定义函数，类和变量，模块里也能包含可执行的代码。

## import 语句

### 模块的引入

模块定义好后，我们可以使用 import 语句来引入模块，语法如下：

```
import module1[, module2[,... moduleN]]
```

比如要引用模块 math，就可以在文件最开始的地方用 **import math** 来引入。在调用 math 模块中的函数时，必须这样引用：

```
模块名.函数名
```

当解释器遇到 import 语句，如果模块在当前的搜索路径就会被导入。
搜索路径是一个解释器会先进行搜索的所有目录的列表。

## from…import 语句

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中。语法如下：

```
    from modname import name1[, name2[, ... nameN]]
```

例如，要导入模块 fib 的 fibonacci 函数，使用如下语句：

```
    from fib import fibonacci
```

这个声明不会把整个 fib 模块导入到当前的命名空间中，它只会将 fib 里的 fibonacci 单个引入到执行这个声明的模块的全局符号表。

## from…import* 语句

把一个模块的所有内容全都导入到当前的命名空间也是可行的，只需使用如下声明：

```
from modname import *
```

这提供了一个简单的方法来导入一个模块中的所有项目。然而这种声明不该被过多地使用。
例如我们想一次性引入 math 模块中所有的东西，语句如下：

```
from math import *
```

## 搜索路径

当你导入一个模块，Python 解析器对模块位置的搜索顺序是：
1、当前目录
2、如果不在当前目录，Python 则搜索在 shell 变量 PYTHONPATH 下的每个目录。
3、如果都找不到，Python 会察看默认路径。UNIX 下，默认路径一般为/usr/local/lib/python/。
模块搜索路径存储在 system 模块的 sys.path 变量中。变量里包含当前目录，PYTHONPATH 和由安装过程决定的默认目录。
**如果一个局部变量和一个全局变量重名，则局部变量会覆盖全局变量。**
Python 会智能地猜测一个变量是局部的还是全局的，它假设任何在函数内赋值的变量都是局部的。
因此，**如果要给函数内的全局变量赋值，必须使用 global 语句。**
global VarName 的表达式会告诉 Python， VarName 是一个全局变量，这样 Python 就不会在局部命名空间里寻找这个变量了。

## dir()函数

dir() 函数一个排好序的字符串列表，内容是一个模块里定义过的名字。
返回的列表容纳了在一个模块里定义的所有模块，变量和函数。如下一个简单的实例：

```
#!/usr/bin/python

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

# 导入内置 math 模块

import math

content = dir(math)

print content;
```

以上实例输出结果：

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
```

```
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',

'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',

'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',

'sqrt', 'tan', 'tanh']
```

在这里，特殊字符串变量__name__指向模块的名字，__file__指向该模块的导入文件名。

## globals() 和 locals() 函数

根据调用地方的不同，globals() 和 locals() 函数可被用来返回全局和局部命名空间里的名字。
如果在函数内部调用 locals()，返回的是所有能在该函数里访问的命名。
如果在函数内部调用 globals()，返回的是所有在该函数里能访问的全局名字。
两个函数的返回类型都是字典。所以名字们能用 keys() 函数摘取。

## reload() 函数

当一个模块被导入到一个脚本，模块顶层部分的代码只会被执行一次。
因此，如果你想重新执行模块里顶层部分的代码，可以用 reload() 函数。该函数会重新导入之前导入过的模块。语法如下：

```
    reload(module_name)
```

在这里，module_name 要直接放模块的名字，而不是一个字符串形式。比如想重载 hello 模块，如下：

```
    reload(hello)
```

## Python 中的包

包是一个分层次的文件目录结构，它定义了一个由模块及子包，和子包下的子包等组成的 Python 的应用环境。
简单来说，包就是文件夹，但该文件夹下必须存在 __init__.py 文件，该文件的内容可以为空。**__init__.py** 用于标识当前文件夹是一个包。

## Python 文件 I/O

## 读取键盘输入

## raw_input 函数

raw_input([prompt]) 函数从标准输入读取一个行，并返回一个字符串（去掉结尾的换行符）

## input 函数

input([prompt]) 函数和 raw_input([prompt]) 函数基本类似，但是 input 可以接收一个 Python 表达式作为输入，并将运算结果返回。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

str = input("请输入：")
print "你输入的内容是: ", str
```

这会产生如下的对应着输入的结果：

```
请输入：[x*5 for x in range(2,10,2)]
你输入的内容是:  [10, 20, 30, 40]
```

# 打开和关闭文件

## open 函数

你必须先用 Python 内置的 open()函数打开一个文件，创建一个 file 对象，相关的方法才可以调用它进行读写。
语法：

```
file object = open(file_name [, access_mode][, buffering])
```

各个参数的细节如下：

- file_name：file_name 变量是一个包含了你要访问的文件名称的字符串值。
- access_mode：access_mode 决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。
- buffering:如果 buffering 的值被设为 0，就不会有寄存。如果 buffering 的值取 1，访问文件时会寄存行。如果将 buffering 的值设为大于 1 的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

不同模式打开文件的完全列表：

| 模式 | 描述 |
| --- | --- |
| t | 文本模式 (默认)。 |
| x | 写模式，新建一个文件，如果该文件已存在则会报错。 |
| b | 二进制模式。 |
| + | 打开一个文件进行更新(可读可写)。 |
| U | 通用换行模式（不推荐）。 |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。一般用于非文本文件如图片等。 |

| | |
|---|---|
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。一般用于非文本文件如图片等。 |
| w | 打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb+ | 以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。 |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| a+ | 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。 |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 |

## File 对象的属性

以下是和 file 对象相关的所有属性的列表：

| 属性 | 描述 |
|---|---|
| file.closed | 返回 true 如果文件已被关闭，否则返回 false。 |
| file.mode | 返回被打开文件的访问模式。 |
| file.name | 返回文件的名称。 |

| | |
|---|---|
| file.softspace | 如果用 print 输出后，必须跟一个空格符，则返回 false。否则返回 true。 |

## close()方法

File 对象的 close（）方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。
当一个文件对象的引用被重新指定给另一个文件时，Python 会关闭之前的文件。用 close（）方法关闭文件是一个很好的习惯。
语法：

```
fileObject.close()
```

## write()方法

write()方法可将任何字符串写入一个打开的文件。需要重点注意的是，Python 字符串可以是二进制数据，而不是仅仅是文字。
write()方法不会在字符串的结尾添加换行符('\n')
语法：

```
fileObject.write(string)
```

## read()方法

read（）方法从一个打开的文件中读取一个字符串。需要重点注意的是，Python 字符串可以是二进制数据，而不是仅仅是文字。
语法：

```
    fileObject.read([count])
```

在这里，被传递的参数是要从已打开文件中读取的字节计数。该方法从文件的开头开始读入，如果没有传入 count，它会尝试尽可能多地读取更多的内容，很可能是直到文件的末尾。

## 文件定位

tell()方法告诉你文件内的当前位置，换句话说，下一次的读写会发生在文件开头这么多字节之后。
seek（offset [,from]）方法改变当前文件的位置。Offset 变量表示要移动的字节数。From 变量指定开始移动字节的参考位置。
如果 from 被设为 0，这意味着将文件的开头作为移动字节的参考位置。如果设为 1，则使用当前的位置作为参考位置。如果它被设为 2，那么该文件的末尾将作为参考位置。

## 重命名和删除文件

Python 的 os 模块提供了帮你执行文件处理操作的方法，比如重命名和删除文件。
要使用这个模块，你必须先导入它，然后才可以调用相关的各种功能。
rename()方法：
rename()方法需要两个参数，当前的文件名和新文件名。
语法：

```
os.rename(current_file_name, new_file_name)
```

## remove()方法

你可以用 remove()方法删除文件，需要提供要删除的文件名作为参数。
语法：

```
    os.remove(file_name)
```

## Python 里的目录：

所有文件都包含在各个不同的目录下，不过 Python 也能轻松处理。os 模块有许多方法能帮你创建，删除和更改目录。

## mkdir()方法

可以使用 os 模块的 mkdir()方法在当前目录下创建新的目录们。你需要提供一个包含了要创建的目录名称的参数。
语法：

```
os.mkdir("newdir")
```

## chdir()方法

可以用 chdir()方法来改变当前的目录。chdir()方法需要的一个参数是你想设成当前目录的目录名称。
语法：

```
    os.chdir("newdir")
```

getcwd()方法：
getcwd()方法显示当前的工作目录。
语法：

```
os.getcwd()
```

## rmdir()方法

rmdir()方法删除目录，目录名称以参数传递。
在**删除这个目录之前，它的所有内容应该先被清除**。
语法：

```
    os.rmdir('dirname')
```

## Python File(文件) 方法

## open() 方法

Python open() 方法用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。

**注意：**使用 open() 方法一定要保证关闭文件对象，即调用 close() 方法。
open() 函数常用形式是接收两个参数：文件名(file)和模式(mode)。

```
open(file, mode='r')
```

完整的语法格式为：

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd
=True, opener=None)
```

参数说明:

- file: 必需，文件路径（相对或者绝对路径）。
- mode: 可选，文件打开模式
- buffering: 设置缓冲
- encoding: 一般使用 utf8
- errors: 报错级别
- newline: 区分换行符
- closefd: 传入的 file 参数类型
- opener:

## file 对象

file 对象使用 open 函数来创建，下表列出了 file 对象常用的函数：

| 序号 | 方法及描述 |
| --- | --- |
| 1 | file.close()<br>关闭文件。关闭后文件不能再进行读写操作。 |
| 2 | file.flush()<br>刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件，而不是被动的等待输出缓冲区写入。 |
| 3 | file.fileno()<br>返回一个整型的文件描述符(file descriptor FD 整型)，可以用在如 os 模块的 read 方法等一些底层操作上。 |
| 4 | file.isatty()<br>如果文件连接到一个终端设备返回 True，否则返回 False。 |
| 5 | file.next()<br>返回文件下一行。 |
| 6 | file.read([size])<br>从文件读取指定的字节数，如果未给定或为负则读取所有。 |

| | |
|---|---|
| 7 | file.readline([size])<br>读取整行，包括 "\n" 字符。 |
| 8 | file.readlines([sizeint])<br>读取所有行并返回列表，若给定 sizeint>0，则是设置一次读多少字节，这是为了减轻读取压力。 |
| 9 | file.seek(offset[, whence])<br>设置文件当前位置 |
| 10 | file.tell()<br>返回文件当前位置。 |
| 11 | file.truncate([size])<br>截取文件，截取的字节通过 size 指定，默认为当前文件位置。 |
| 12 | file.write(str)<br>将字符串写入文件，返回的是写入的字符长度。 |
| 13 | file.writelines(sequence)<br>向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。 |

# Python 异常处理

## python 标准异常

| 异常名称 | 描述 |
|---|---|
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |

| | |
|---|---|
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达 EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于 Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |

| | |
|---|---|
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

## 异常处理

捕捉异常可以使用 try/except 语句。
try/except 语句用来检测 try 语句块中的错误，从而让 except 语句捕获异常信息并处理。
如果你不想在异常发生时结束你的程序，只需在 try 里捕获它。
语法：
以下为简单的 *try....except...else* 的语法：

```
try:
<语句>        #运行别的代码
except <名字>:
```

```
<语句>        #如果在 try 部份引发了'name'异常
except <名字>，<数据>:
<语句>        #如果引发了'name'异常，获得附加的数据
else:
<语句>        #如果没有异常发生
```

try 的工作原理是，当开始一个 try 语句后，python 就在当前程序的上下文中作标记，这样当异常出现时就可以回到这里，try 子句先执行，接下来会发生什么依赖于执行时是否出现异常。

- 如果当 try 后的语句执行时发生异常，python 就跳回到 try 并执行第一个匹配该异常的 except 子句，异常处理完毕，控制流就通过整个 try 语句（除非在处理异常时又引发新的异常）。
- 如果在 try 后的语句里发生了异常，却没有匹配的 except 子句，异常将被递交到上层的 try，或者到程序的最上层（这样将结束程序，并打印缺省的出错信息）。
- 如果在 try 子句执行时没有发生异常，python 将执行 else 语句后的语句（如果有 else 的话），然后控制流通过整个 try 语句。

## 使用 except 而不带任何异常类型

你可以不带任何异常类型使用 except，如下实例：

```
try:
    正常的操作
   ......................
except:
    发生异常，执行这块代码
   ......................
else:
    如果没有异常执行这块代码
```

以上方式 try-except 语句捕获所有发生的异常。但这不是一个很好的方式，我们不能通过该程序识别出具体的异常信息。因为它捕获所有的异常。

## 使用 except 而带多种异常类型

你也可以使用相同的 except 语句来处理多个异常信息，如下所示：

```
try:
    正常的操作
   ......................
except(Exception1[, Exception2[,...ExceptionN]]):
   发生以上多个异常中的一个，执行这块代码
   ......................
else:
    如果没有异常执行这块代码
```

## try-finally 语句

try-finally 语句无论是否发生异常都将执行最后的代码。

```
try:
<语句>
finally:
<语句>    #退出 try 时总会执行
raise
```

## 异常的参数

一个异常可以带上参数，可作为输出的异常信息参数。
你可以通过 except 语句来捕获异常的参数，如下所示：

```
try:
    正常的操作
   ......................
except ExceptionType, Argument:
    你可以在这输出 Argument 的值...
```

变量接收的异常值通常包含在异常的语句中。在元组的表单中变量可以接收一个或者多个值。
元组通常包含错误字符串，错误数字，错误位置。

## 触发异常

我们可以使用 raise 语句自己触发异常
raise 语法格式如下：

```
raise [Exception [, args [, traceback]]]
```

语句中 Exception 是异常的类型（例如，NameError）参数标准异常中任一种，args 是自己提供的异常参数。
最后一个参数是可选的（在实践中很少使用），如果存在，是跟踪异常对象。

## 用户自定义异常

通过创建一个新的异常类，程序可以命名它们自己的异常。异常应该是典型的继承自 **Exception 类**，通过直接或间接的方式。
以下为与 RuntimeError 相关的实例,实例中创建了一个类，基类为 RuntimeError，用于在异常触发时输出更多的信息。
在 try 语句块中，用户自定义的异常后执行 except 块语句，变量 e 是用于创建 Networkerror 类的实例。

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

在你定义以上类后，你可以触发该异常，如下所示：

```
try:
```

```
        raise Networkerror("Bad hostname")
    except Networkerror,e:
        print e.args
```

## Python OS 文件/目录方法

os 模块提供了非常丰富的方法用来处理文件和目录。常用的方法如下表所示：

| 序号 | 方法及描述 |
|---|---|
| 1 | os.access(path, mode)<br>检验权限模式 |
| 2 | os.chdir(path)<br>改变当前工作目录 |
| 3 | os.chflags(path, flags)<br>设置路径的标记为数字标记。 |
| 4 | os.chmod(path, mode)<br>更改权限 |
| 5 | os.chown(path, uid, gid)<br>更改文件所有者 |
| 6 | os.chroot(path)<br>改变当前进程的根目录 |
| 7 | os.close(fd)<br>关闭文件描述符 fd |
| 8 | os.closerange(fd_low, fd_high)<br>关闭所有文件描述符，从 fd_low（包含）到 fd_high（不包含），错误会忽略 |
| 9 | os.dup(fd)<br>复制文件描述符 fd |

| | |
|---|---|
| 10 | os.dup2(fd, fd2)<br>将一个文件描述符 fd 复制到另一个 fd2 |
| 11 | os.fchdir(fd)<br>通过文件描述符改变当前工作目录 |
| 12 | os.fchmod(fd, mode)<br>改变一个文件的访问权限，该文件由参数 fd 指定，参数 mode 是 Unix 下的文件访问权限。 |
| 13 | os.fchown(fd, uid, gid)<br>修改一个文件的所有权，这个函数修改一个文件的用户 ID 和用户组 ID，该文件由文件描述符 fd 指定。 |
| 14 | os.fdatasync(fd)<br>强制将文件写入磁盘，该文件由文件描述符 fd 指定，但是不强制更新文件的状态信息。 |
| 15 | os.fdopen(fd[, mode[, bufsize]])<br>通过文件描述符 fd 创建一个文件对象，并返回这个文件对象 |
| 16 | os.fpathconf(fd, name)<br>返回一个打开的文件的系统配置信息。name 为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多标准中指定（POSIX.1, Unix 95, Unix 98, 和其它）。 |
| 17 | os.fstat(fd)<br>返回文件描述符 fd 的状态，像 stat()。 |
| 18 | os.fstatvfs(fd)<br>返回包含文件描述符 fd 的文件的文件系统的信息，像 statvfs() |

| | |
|---|---|
| 19 | os.fsync(fd)<br>强制将文件描述符为 fd 的文件写入硬盘。 |
| 20 | os.ftruncate(fd, length)<br>裁剪文件描述符 fd 对应的文件, 所以它最大不能超过文件大小。 |
| 21 | os.getcwd()<br>返回当前工作目录 |
| 22 | os.getcwdu()<br>返回一个当前工作目录的 Unicode 对象 |
| 23 | os.isatty(fd)<br>如果文件描述符 fd 是打开的，同时与 tty(-like)设备相连，则返回 true, 否则 False。 |
| 24 | os.lchflags(path, flags)<br>设置路径的标记为数字标记，类似 chflags()，但是没有软链接 |
| 25 | os.lchmod(path, mode)<br>修改连接文件权限 |
| 26 | os.lchown(path, uid, gid)<br>更改文件所有者，类似 chown，但是不追踪链接。 |
| 27 | os.link(src, dst)<br>创建硬链接，名为参数 dst，指向参数 src |
| 28 | os.listdir(path)<br>返回 path 指定的文件夹包含的文件或文件夹的名字的列表。 |
| 29 | os.lseek(fd, pos, how)<br>设置文件描述符 fd 当前位置为 pos, how 方式修改: SEEK_SET 或者 0 设置从文件 |

| | |
|---|---|
| | 开始的计算的 pos; SEEK_CUR 或者 1 则从当前位置计算; os.SEEK_END 或者 2 则从文件尾部开始. 在 unix，Windows 中有效 |
| 30 | os.lstat(path)<br>像 stat(),但是没有软链接 |
| 31 | os.major(device)<br>从原始的设备号中提取设备 major 号码 (使用 stat 中的 st_dev 或者 st_rdev fiel d)。 |
| 32 | os.makedev(major, minor)<br>以 major 和 minor 设备号组成一个原始设备号 |
| 33 | os.makedirs(path[, mode])<br>递归文件夹创建函数。像 mkdir(), 但创建的所有 intermediate-level 文件夹需要包含子文件夹。 |
| 34 | os.minor(device)<br>从原始的设备号中提取设备 minor 号码 (使用 stat 中的 st_dev 或者 st_rdev field )。 |
| 35 | os.mkdir(path[, mode])<br>以数字 mode 的 mode 创建一个名为 path 的文件夹.默认的 mode 是 0777 (八进制)。 |
| 36 | os.mkfifo(path[, mode])<br>创建命名管道，mode 为数字，默认为 0666 (八进制) |
| 37 | os.mknod(filename[, mode=0600, device])<br>创建一个名为 filename 文件系统节点（文件，设备特别文件或者命名 pipe）。 |
| 38 | os.open(file, flags[, mode])<br>打开一个文件，并且设置需要的打开选项，mode 参数是可选的 |

| | |
|---|---|
| 39 | os.openpty()<br>打开一个新的伪终端对。返回 pty 和 tty 的文件描述符。 |
| 40 | os.pathconf(path, name)<br>返回相关文件的系统配置信息。 |
| 41 | os.pipe()<br>创建一个管道. 返回一对文件描述符(r, w) 分别为读和写 |
| 42 | os.popen(command[, mode[, bufsize]])<br>从一个 command 打开一个管道 |
| 43 | os.read(fd, n)<br>从文件描述符 fd 中读取最多 n 个字节，返回包含读取字节的字符串，文件描述符 fd 对应文件已达到结尾，返回一个空字符串。 |
| 44 | os.readlink(path)<br>返回软链接所指向的文件 |
| 45 | os.remove(path)<br>删除路径为 path 的文件。如果 path 是一个文件夹，将抛出 OSError; 查看下面的 rmdir()删除一个 directory。 |
| 46 | os.removedirs(path)<br>递归删除目录。 |
| 47 | os.rename(src, dst)<br>重命名文件或目录，从 src 到 dst |
| 48 | os.renames(old, new)<br>递归地对目录进行更名，也可以对文件进行更名。 |

| | |
|---|---|
| 49 | os.rmdir(path)<br>删除 path 指定的空目录，如果目录非空，则抛出一个 OSError 异常。 |
| 50 | os.stat(path)<br>获取 path 指定的路径的信息，功能等同于 C API 中的 stat()系统调用。 |
| 51 | os.stat_float_times([newvalue])<br>决定 stat_result 是否以 float 对象显示时间戳 |
| 52 | os.statvfs(path)<br>获取指定路径的文件系统统计信息 |
| 53 | os.symlink(src, dst)<br>创建一个软链接 |
| 54 | os.tcgetpgrp(fd)<br>返回与终端 fd（一个由 os.open()返回的打开的文件描述符）关联的进程组 |
| 55 | os.tcsetpgrp(fd, pg)<br>设置与终端 fd（一个由 os.open()返回的打开的文件描述符）关联的进程组为 pg。 |
| 56 | os.tempnam([dir[, prefix]])<br>返回唯一的路径名用于创建临时文件。 |
| 57 | os.tmpfile()<br>返回一个打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。 |
| 58 | os.tmpnam()<br>为创建一个临时文件返回一个唯一的路径 |

| | |
|---|---|
| 59 | os.ttyname(fd)<br>返回一个字符串，它表示与文件描述符 fd 关联的终端设备。如果 fd 没有与终端设备关联，则引发一个异常。 |
| 60 | os.unlink(path)<br>删除文件路径 |
| 61 | os.utime(path, times)<br>返回指定的 path 文件的访问和修改的时间。 |
| 62 | os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])<br>输出在文件夹中的文件名通过在树中游走，向上或者向下。 |
| 63 | os.write(fd, str)<br>写入字符串到文件描述符 fd 中. 返回实际写入的字符串长度 |
| 64 | os.path 模块<br>获取文件的属性信息。 |

## Python 内置函数

| 内置函数 | | | | |
|---|---|---|---|---|
| abs() | divmod() | input() | open() | staticmethod() |
| all() | enumerate() | int() | ord() | str() |
| any() | eval() | isinstance() | pow() | sum() |
| basestring() | execfile() | issubclass() | print() | super() |
| bin() | file() | iter() | property() | tuple() |
| bool() | filter() | len() | range() | type() |
| bytearray() | float() | list() | raw_input() | unichr() |
| callable() | format() | locals() | reduce() | unicode() |

| 内置函数 | | | | |
|---|---|---|---|---|
| chr() | frozenset() | long() | reload() | vars() |
| classmethod() | getattr() | map() | repr() | xrange() |
| cmp() | globals() | max() | reverse() | zip() |
| compile() | hasattr() | memoryview() | round() | __import__() |
| complex() | hash() | min() | set() | |
| delattr() | help() | next() | setattr() | |
| dict() | hex() | object() | slice() | |
| dir() | id() | oct() | sorted() | exec 内置表达式 |