Python 学习笔记

（学习网址： http://www.runoob.com/python/python-intro.html）

# 1 Python 简介

Python 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。

Python 的设计具有很强的可读性，相比其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其他语言更有特色语法结构。

* **Python 是一种解释型语言：** 这意味着开发过程中没有了编译这个环节。类似于PHP和Perl语言。
* **Python 是交互式语言：** 这意味着，您可以在一个Python提示符，直接互动执行写你的程序。
* **Python 是面向对象语言:** 这意味着Python支持面向对象的风格或代码封装在对象的编程技术。
* **Python 是初学者的语言：**Python 对初级程序员而言，是一种伟大的语言，它支持广泛的应用程序开发，从简单的文字处理到 WWW 浏览器再到游戏。

# 2 Python安装

Python已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。

您需要下载适用于您使用平台的二进制代码，然后安装Python。

如果您平台的二进制代码是不可用的，你需要使用C编译器手动编译源代码。

编译的源代码，功能上有更多的选择性， 为python安装提供了更多的灵活性。

以下为不同平台上安装Python的方法：

**Unix & Linux 平台安装 Python:**

以下为在Unix & Linux 平台上安装 Python 的简单步骤：

* 打开WEB浏览器访问<http://www.python.org/download/>
* 选择适用于Unix/Linux的源码压缩包。
* 下载及解压压缩包。
* 如果你需要自定义一些选项修改*Modules/Setup*
* **执行** ./configure 脚本
* make
* make install

执行以上操作后，Python会安装在 /usr/local/bin 目录中，Python库安装在/usr/local/lib/pythonXX，XX为你使用的Python的版本号。

**Window 平台安装 Python:**

以下为在 Window 平台上安装 Python 的简单步骤：

* 打开WEB浏览器访问<http://www.python.org/download/>
* 在下载列表中选择Window平台安装包，包格式为：*python-XYZ.msi* 文件 ， XYZ 为你要安装的版本号。
* 要使用安装程序 *python-XYZ.msi*, Windows系统必须支持Microsoft Installer 2.0搭配使用。只要保存安装文件到本地计算机，然后运行它，看看你的机器支持MSI。Windows XP和更高版本已经有MSI，很多老机器也可以安装MSI。
* 下载后，双击下载包，进入Python安装向导，安装非常简单，你只需要使用默认的设置一直点击"下一步"直到安装完成即可。

**环境变量配置**

程序和可执行文件可以在许多目录，而这些路径很可能不在操作系统提供可执行文件的搜索路径中。

path(路径)存储在环境变量中，这是由操作系统维护的一个命名的字符串。这些变量包含可用的命令行解释器和其他程序的信息。

Unix或Windows中路径变量为PATH（UNIX区分大小写，Windows不区分大小写）。

在Mac OS中，安装程序过程中改变了python的安装路径。如果你需要在其他目录引用Python，你必须在path中添加Python目录。

**在 Unix/Linux 设置环境变量**

* **在 csh shell:** 输入

setenv PATH "$PATH:/usr/local/bin/python"

, 按下"Enter"。

* **在 bash shell (Linux):** 输入

export PATH="$PATH:/usr/local/bin/python"

，按下"Enter"。

* **在 sh 或者 ksh shell:** 输入

PATH="$PATH:/usr/local/bin/python"

, 按下"Enter"。

**注意:** /usr/local/bin/python 是Python的安装目录。

**在 Windows 设置环境变量**

在环境变量中添加Python目录：

**在命令提示框中(cmd) :** 输入

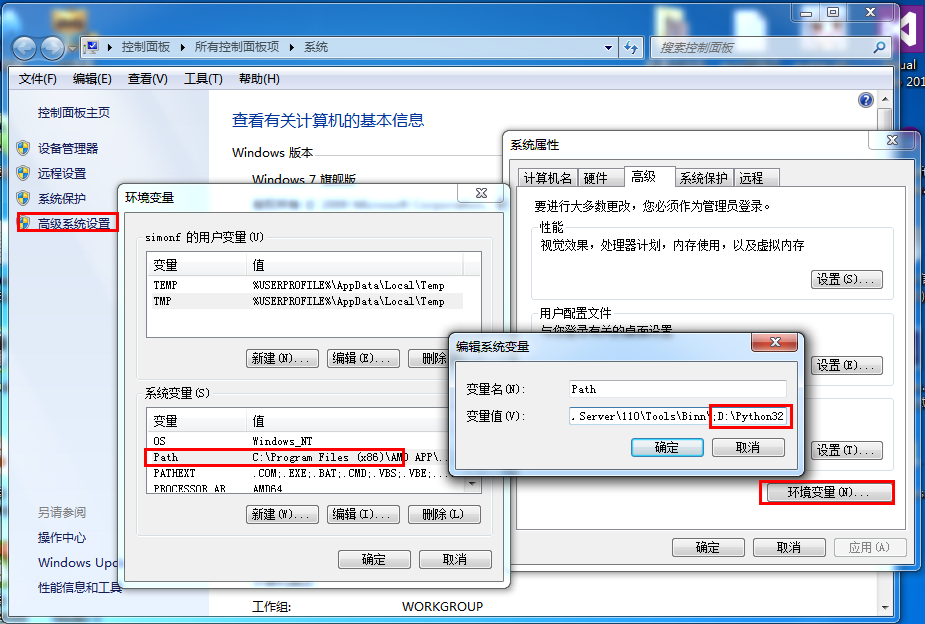
path=%path%;C:\Python

按下"Enter"。

**注意:** C:\Python 是Python的安装目录。

也可以通过以下方式设置：

* 右键点击"计算机"，然后点击"属性"
* 然后点击"高级系统设置"
* 选择"系统变量"窗口下面的"Path",双击即可！
* 然后在"Path"行，添加python安装路径即可(我的D:\Python32)，所以在后面，添加该路径即可。 **ps：记住，路径直接用分号"；"隔开！**
* 最后设置成功以后，在cmd命令行，输入命令"python"，就可以有相关显示。



**Python集成开发环境：PyCharm**

* PyCharm 是由 JetBrains 打造的一款 Python IDE，支持 macOS、 Windows、 Linux 系统。
* PyCharm 功能 : 调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制……
* PyCharm 下载地址 : https://www.jetbrains.com/pycharm/download/

# 3 Python语法

## 3.1 中文编码

Python 文件中如果未指定编码，在执行过程会出现报错，因此当代码中需要输出中文时要指定编码格式，Python中默认的编码格式是 ASCII 格式，在没修改编码格式时无法正确打印汉字，所以在读取中文时会报错。

解决方法为只要在文件开头加入 **# -\*- coding: UTF-8 -\*-** 或**#coding=utf-8** 就行了。

实例(Python 2.0+)

#!/usr/bin/python

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

print "你好，世界";

**注意：**Python3.X 源码文件默认使用utf-8编码，所以可以正常解析中文，无需指定 UTF-8 编码。

## 3.2 执行

（1）$ python test.py

（2）脚本里添加内容

#!/usr/bin/python

这里，假定Python解释器在/usr/bin目录中，使用以下命令执行脚本：

$ chmod +x test.py # 脚本文件添加可执行权限

$ ./test.py

## 3.3 标识符

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。

## 3.4 行和缩进

学习Python与其他语言最大的区别就是，Python的代码块不使用大括号（{}）来控制类，函数以及其他逻辑判断。python最具特色的就是用缩进来写模块。缩进的空白数量是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个必须严格执行。如下所示：

if True:

print "True"

else:

print "False"

if和else必须对齐，它的子句print也必须对齐，并且使用的缩进方式也得一致，要么都使用空格，要么都使用tab

## 3.5 多行语句

Python语句中一般以新行作为为语句的结束符。

但是我们可以使用斜杠（ \）将一行的语句分为多行显示，如下所示：

total = item\_one + \

item\_two + \

item\_three

语句中包含[], {} 或 () 括号就不需要使用多行连接符。如下实例：

days = ['Monday', 'Tuesday', 'Wednesday',

'Thursday', 'Friday']

## 3.6 引号和注释

Python 可以使用引号( **'** )、双引号( **"** )、三引号( **'''** 或 **"""** ) 来表示字符串，引号的开始与结束必须的相同类型的。其中三引号可以由多行组成，编写多行文本的快捷语法，常用语文档字符串，在文件的特定地点，被当做注释。

word = 'word'

sentence = "这是一个句子。"

paragraph = """这是一个段落。

包含了多个语句"""

python中单行注释采用 # 开头。

python 中多行注释使用三个单引号(''')或三个双引号(""")。

'''

这是多行注释，使用单引号。

这是多行注释，使用单引号。

这是多行注释，使用单引号。

'''

"""

这是多行注释，使用双引号。

这是多行注释，使用双引号。

这是多行注释，使用双引号。

"""

## 3.7 空行、用户输入和多行语句

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行并不是Python语法的一部分。书写时不插入空行，Python解释器运行也不会出错。但是空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

记住：空行也是程序代码的一部分。

**等待用户输入**

下面的程序在按回车键后就会等待用户输入：

#!/usr/bin/python

raw\_input("\n\nPress the enter key to exit.")

以上代码中 ，"\n\n"在结果输出前会输出两个新的空行。一旦用户按下 enter(回车) 键退出，其它键显示。

**同一行显示多条语句**

Python可以在同一行中使用多条语句，语句之间使用分号(;)分割，以下是一个简单的实例：

#!/usr/bin/python

import sys; x = 'runoob'; sys.stdout.write(x + '\n')

执行以上代码，输入结果为：

$ python test.py

runoob

**多个语句构成代码组**

缩进相同的一组语句构成一个代码块，我们称之代码组。

像if、while、def和class这样的复合语句，首行以关键字开始，以冒号( : )结束，该行之后的一行或多行代码构成代码组。

我们将首行及后面的代码组称为一个子句(clause)。

如下实例：

if expression :

suite

elif expression :

suite

else :

suite

## 3.8 变量

**变量赋值**

Python 中的变量赋值不需要类型声明。

每个变量在内存中创建，都包括变量的标识，名称和数据这些信息。

每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。

counter = 100 # 赋值整型变量

miles = 1000.0 # 浮点型

name = "John" # 字符串

print counter

print miles

print name

**多个变量赋值**

Python允许你同时为多个变量赋值。例如：

a = b = c = 1

以上实例，创建一个整型对象，值为1，三个变量被分配到相同的内存空间上。

您也可以为多个对象指定多个变量。例如：

a, b, c = 1, 2, "john"

以上实例，两个整型对象1和2的分配给变量 a 和 b，字符串对象 "john" 分配给变量 c。

**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）

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 #删除单个对象引用

del var\_a, var\_b #删除多个对象引用，逗号隔开

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]），Python使用"L"来显示长整型，例如 51924361**L** 。
* float（浮点型）
* complex（复数） ，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，例如3e+26**J**

## 3.9 字符串

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，可以使用变量 **[头下标:尾下标]，例如s[1:5]表示从下标1开始取，取到下标5-1=4，取的长度也是5-1=4。**加号（+）是字符串连接运算符，星号（\*）是重复操作。如下实例：

str = 'Hello World!'

print str # 输出完整字符串

print str[0] # 输出字符串中的第一个字符

print str[-1] # 输出字符串中从右开始的第一个字符

print str[2:5] # 输出字符串中第三个至第五个之间的字符串

print str[2:] # 输出从第三个字符开始的字符串

print str[-3**:**] #取最后三个字符

print str \* 2 # 输出字符串两次

print str + "TEST" # 输出连接的字符串

以上实例输出结果：

Hello World!

H

！

llo

llo World!

Hello World!Hello World!

Hello World!TEST

ld！

在python中，要想输出原始格符号，如\n，则使用 r/’\n’,这样输出的就是\n,而不是输出空行

**字符串格式化**

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!

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) |

python中三引号可以将复杂的字符串进行复制:

python三引号允许一个字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。

三引号的语法是一对连续的单引号或者双引号（通常都是成对的用）。

>>> hi = '''hi

there'''

>>> hi # repr()

'hi\nthere'

>>> print hi # str()

hi

there

三引号让程序员从引号和特殊字符串的泥潭里面解脱出来，自始至终保持一小块字符串的格式是所谓的WYSIWYG（所见即所得）格式的。

一个典型的用例是，当你需要一块HTML或者SQL时，这时用字符串组合，特殊字符串转义将会非常的繁琐。

errHTML = '''

<HTML><HEAD><TITLE>

Friends CGI Demo</TITLE></HEAD>

<BODY><H3>ERROR</H3>

<B>%s</B><P>

<FORM><INPUT TYPE=button VALUE=Back

ONCLICK="window.history.back()"></FORM>

</BODY></HTML>

'''

cursor.execute('''

CREATE TABLE users (

login VARCHAR(8),

uid INTEGER,

prid INTEGER)

''')

### 字符串内建函数

|  |  |
| --- | --- |
| **方法** | **描述** |
| [string.capitalize()](http://www.runoob.com/python/att-string-capitalize.html) | 把字符串的第一个字符大写 |
| [string.center(width)](http://www.runoob.com/python/att-string-center.html) | 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))](http://www.runoob.com/python/att-string-count.html) | 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')](http://www.runoob.com/python/att-string-decode.html) | 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')](http://www.runoob.com/python/att-string-encode.html) | 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))**](http://www.runoob.com/python/att-string-endswith.html) | **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)](http://www.runoob.com/python/att-string-expandtabs.html) | 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))**](http://www.runoob.com/python/att-string-find.html) | **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.index(str, beg=0, end=len(string))](http://www.runoob.com/python/att-string-index.html) | 跟find()方法一样，只不过如果str不在 string中会报一个异常. |
| [string.isalnum()](http://www.runoob.com/python/att-string-isalnum.html) | 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()](http://www.runoob.com/python/att-string-isalpha.html) | 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()](http://www.runoob.com/python/att-string-isdecimal.html) | 如果 string 只包含十进制数字则返回 True 否则返回 False. |
| [string.isdigit()](http://www.runoob.com/python/att-string-isdigit.html) | 如果 string 只包含数字则返回 True 否则返回 False. |
| [string.islower()](http://www.runoob.com/python/att-string-islower.html) | 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()](http://www.runoob.com/python/att-string-isnumeric.html) | 如果 string 中只包含数字字符，则返回 True，否则返回 False |
| [**string.isspace()**](http://www.runoob.com/python/att-string-isspace.html) | **如果 string 中只包含空格，则返回 True，否则返回 False.** |
| [string.istitle()](http://www.runoob.com/python/att-string-istitle.html) | 如果 string 是标题化的(见 title())则返回 True，否则返回 False |
| [string.isupper()](http://www.runoob.com/python/att-string-isupper.html) | 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)**](http://www.runoob.com/python/att-string-join.html) | **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)](http://www.runoob.com/python/att-string-ljust.html) | 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()](http://www.runoob.com/python/att-string-lower.html) | 转换 string 中所有大写字符为小写. |
| [string.lstrip()](http://www.runoob.com/python/att-string-lstrip.html) | 截掉 string 左边的空格 |
| [string.maketrans(intab, outtab])](http://www.runoob.com/python/att-string-maketrans.html) | 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)](http://www.runoob.com/python/att-string-max.html) | 返回字符串 str 中最大的字母。 |
| [min(str)](http://www.runoob.com/python/att-string-min.html) | 返回字符串 str 中最小的字母。 |
| [**string.partition(str)**](http://www.runoob.com/python/att-string-partition.html) | **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.**  **例如：**str = "http://www.w3cschool.cc/"  print str.partition("://")  ==》结果：('http', '://', 'www.w3cschool.cc/') |
| [**string.replace(str1, str2,  num=string.count(str1))**](http://www.runoob.com/python/att-string-replace.html) | **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) )**](http://www.runoob.com/python/att-string-rfind.html) | **类似于 find()函数，不过是从右边开始查找.** |
| [**string.rindex( str, beg=0,end=len(string))**](http://www.runoob.com/python/att-string-rindex.html) | **类似于 index()，不过是从右边开始.** |
| [string.rjust(width)](http://www.runoob.com/python/att-string-rjust.html) | 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()**](http://www.runoob.com/python/att-string-rstrip.html) | **删除 string 字符串末尾的空格.** |
| [**string.split(str="", num=string.count(str))**](http://www.runoob.com/python/att-string-split.html) | **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])](http://www.runoob.com/python/att-string-splitlines.html) | 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))**](http://www.runoob.com/python/att-string-startswith.html) | **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])**](http://www.runoob.com/python/att-string-strip.html) | **在 string 上执行 lstrip()和 rstrip()** |
| [string.swapcase()](http://www.runoob.com/python/att-string-swapcase.html) | 翻转 string 中的大小写 |
| [string.title()](http://www.runoob.com/python/att-string-title.html) | 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="")](http://www.runoob.com/python/att-string-translate.html) | 根据 str 给出的表(包含 256 个字符)转换 string 的字符,  要过滤掉的字符放到 del 参数中 |
| [string.upper()](http://www.runoob.com/python/att-string-upper.html) | 转换 string 中的小写字母为大写 |
| [string.zfill(width)](http://www.runoob.com/python/att-string-zfill.html) | 返回长度为 width 的字符串，原字符串 string 右对齐，前面填充0 |
| [string.isdecimal()](http://www.runoob.com/python/att-string-isdecimal.html) | isdecimal()方法检查字符串是否只包含十进制字符。这种方法只存在于unicode对象。 |

## 3.10 列表、元组和字典

**List（列表）** 是 Python 中使用最频繁的数据类型。列表可以重新赋值.

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（所谓嵌套）。

列表用[ ]标识。是python最通用的复合数据类型。

列表中的值得分割也可以用到变量[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开始的，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。如下实例：

list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print list # 输出完整列表

print list[0] # 输出列表的第一个元素

print list[1:3] # 输出第二个至第三个的元素

print list[2:] # 输出从第三个开始至列表末尾的所有元素

print tinylist \* 2 # 输出列表两次

print list + tinylist # 打印组合的列表

以上实例输出结果：

['runoob', 786, 2.23, 'john', 70.2]

runoob

[786, 2.23]

[2.23, 'john', 70.2]

[123, 'john', 123, 'john']

['runoob', 786, 2.23, 'john', 70.2, 123, 'john']

**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。其余的取值用法和列表一样。list( seq )可以将元组转化为列表**。

tinytuple = (123, 'john')

创建空元组

tup1 = ();

元组中只包含一个元素时，需要在元素后面添加逗号

tup1 = (50,);

**字典**(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象结合，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。

**实例(Python 2.0+)**

#!/usr/bin/python

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

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one'] # 输出键为'one' 的值

print dict[2] # 输出键为 2 的值

print tinydict # 输出完整的字典

print tinydict.keys() # 输出所有键

print tinydict.values() # 输出所有值

输出结果为：

This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']

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

在列表中，del s[2]可以直接删除列表中的第三个元素，剩下的元素还是按原先的序列存储。

### 列表元组字典的常用函数

**Python列表函数&方法**

Python包含以下函数:

|  |
| --- |
| **函数** |
| [cmp(list1, list2)](http://www.runoob.com/python/att-list-cmp.html) 比较两个列表的元素 |
| [len(list)](http://www.runoob.com/python/att-list-len.html) 列表元素个数 |
| [max(list)](http://www.runoob.com/python/att-list-max.html) 返回列表元素最大值 |
| [min(list)](http://www.runoob.com/python/att-list-min.html) 返回列表元素最小值 |
| [list(seq)](http://www.runoob.com/python/att-list-list.html) 将元组转换为列表 |

Python 3.X 的版本中已经没有 cmp 函数，如果你需要实现比较功能，需要引入 operator 模块，适合任何对象，包含的方法有：

operator.lt(a, b)

operator.le(a, b)

operator.eq(a, b)

operator.ne(a, b)

operator.ge(a, b)

operator.gt(a, b)

operator.\_\_lt\_\_(a, b)

operator.\_\_le\_\_(a, b)

operator.\_\_eq\_\_(a, b)

operator.\_\_ne\_\_(a, b)

operator.\_\_ge\_\_(a, b)

operator.\_\_gt\_\_(a, b)

Python包含以下方法:

|  |
| --- |
| **方法** |
| [list.append(obj)](http://www.runoob.com/python/att-list-append.html) 在列表末尾添加新的对象 |
| [list.count(obj)](http://www.runoob.com/python/att-list-count.html) 统计某个元素在列表中出现的次数 |
| [list.extend(seq)](http://www.runoob.com/python/att-list-extend.html) 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） |
| [list.index(obj)](http://www.runoob.com/python/att-list-index.html) 从列表中找出某个值第一个匹配项的索引位置 |
| [list.insert(index, obj)](http://www.runoob.com/python/att-list-insert.html) 将对象插入列表 |
| [list.pop(obj=list[-1])](http://www.runoob.com/python/att-list-pop.html) 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 |
| [list.remove(obj)](http://www.runoob.com/python/att-list-remove.html) 移除列表中某个值的第一个匹配项 |
| [list.reverse()](http://www.runoob.com/python/att-list-reverse.html) 反向列表中元素 |
| [list.sort([func])](http://www.runoob.com/python/att-list-sort.html) 对原列表进行排序 |

**Python元组包含了以下内置函数**

|  |
| --- |
| **方法及描述** |
| [cmp(tuple1, tuple2)](http://www.runoob.com/python/att-tuple-cmp.html) 比较两个元组元素。 |
| [len(tuple)](http://www.runoob.com/python/att-tuple-len.html) 计算元组元素个数。 |
| [max(tuple)](http://www.runoob.com/python/att-tuple-max.html) 返回元组中元素最大值。 |
| [min(tuple)](http://www.runoob.com/python/att-tuple-min.html) 返回元组中元素最小值。 |
| [tuple(seq)](http://www.runoob.com/python/att-tuple-tuple.html) 将列表转换为元组。 |

**字典内置函数&方法**

Python字典包含了以下内置函数：

|  |
| --- |
| **函数及描述** |
| [cmp(dict1, dict2)](http://www.runoob.com/python/att-dictionary-cmp.html) 比较两个字典元素。 |
| [len(dict)](http://www.runoob.com/python/att-dictionary-len.html) 计算字典元素个数，即键的总数。 |
| [str(dict)](http://www.runoob.com/python/att-dictionary-str.html) 输出字典可打印的字符串表示。 |
| [type(variable)](http://www.runoob.com/python/att-dictionary-type.html) 返回输入的变量类型，如果变量是字典就返回字典类型。 |

Python字典包含了以下内置方法：

|  |
| --- |
| **函数及描述** |
| [radiansdict.clear()](http://www.runoob.com/python/att-dictionary-clear.html) 删除字典内所有元素 |
| [radiansdict.copy()](http://www.runoob.com/python/att-dictionary-copy.html) 返回一个字典的浅复制 |
| [radiansdict.fromkeys()](http://www.runoob.com/python/att-dictionary-fromkeys.html) 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 |
| [radiansdict.get(key, default=None)](http://www.runoob.com/python/att-dictionary-get.html) 返回指定键的值，如果值不在字典中返回default值 |
| [radiansdict.has\_key(key)](http://www.runoob.com/python/att-dictionary-has_key.html) 如果键在字典dict里返回true，否则返回false |
| [radiansdict.items()](http://www.runoob.com/python/att-dictionary-items.html) 以列表返回可遍历的(键, 值) 元组数组 |
| [radiansdict.keys()](http://www.runoob.com/python/att-dictionary-keys.html) 以列表返回一个字典所有的键 |
| [radiansdict.setdefault(key, default=None)](http://www.runoob.com/python/att-dictionary-setdefault.html) 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 |
| [radiansdict.update(dict2)](http://www.runoob.com/python/att-dictionary-update.html) 把字典dict2的键/值对更新到dict里 |
| [radiansdict.values()](http://www.runoob.com/python/att-dictionary-values.html) 以列表返回字典中的所有值 |

## 3.11 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 |

## 3.12 运算符

**算术运算符**

|  |  |  |
| --- | --- | --- |
| \*\* | 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 | 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0 |

**比较运算符**

|  |  |  |
| --- | --- | --- |
| != | 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 != 。 |

**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 |

**成员运算符**

Python还支持成员运算符，测试实例中包含了一系列的成员，包括字符串，列表或元组。

|  |  |  |
| --- | --- | --- |
| **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。 |

**身份运算符**

身份运算符用于比较两个对象的存储单元即比较两个变量的 id。

|  |  |  |
| --- | --- | --- |
| **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, 如果 id(x) 等于 id(y) , **is** 返回结果 1 |
| 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, 如果 id(x) 不等于 id(y). **is not** 返回结果 1 |

**实例**

#!/usr/bin/python

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

a = 2

b = 20

list = [1, 2, 3, 4, 5 ];

if ( a in list ):

print "变量 a 在给定的列表中 list 中"

if ( b not in list ):

print "变量 b 不在给定的列表中 list 中"

a = 20

b = 20

if ( a is b ): 🡪true

if ( a is not b ): 🡪false

if ( id(a) is id(b) ): 🡪false，相当于比较id(id(a)) == id(id(b))

if ( id(a) == id(b) ): 🡪true

**注意：==比较的是值，is比较的是两个变量的id**

**其余运算符等同C语言**

## 3.13 条件、循环语句

**条件语句**

if 判断条件1:

执行语句1……

执行语句11……

elif 判断条件2:

执行语句2……

elif 判断条件3:

执行语句3……

else:

执行语句4……

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。当if有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先执行，此外 and 和 or 的优先级低于>（大于）、<（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断。

**while循环语句**

while 判断条件：

执行语句……

循环控制语句可以更改语句执行的顺序。Python支持以下循环控制语句：

|  |  |
| --- | --- |
| **控制语句** | **描述** |
| [break 语句](http://www.runoob.com/python/python-break-statement.html) | 在语句块执行过程中终止循环，并且跳出整个循环 |
| [continue 语句](http://www.runoob.com/python/python-continue-statement.html) | 在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。 |
| [pass 语句](http://www.runoob.com/python/python-pass-statement.html) | pass是空语句，是为了保持程序结构的完整性。不做任何事情，一般用做占位语句。 |

**while循环使用 else 语句**

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

#!/usr/bin/python

count = 0

while count < 5:

print count, " is less than 5"

count = count + 1

else:

print count, " is not less than 5"

**简单语句组**

类似 if 语句的语法，如果你的 while 循环体中只有一条语句，你可以将该语句与while写在同一行中， 如下所示：

while (flag): print 'Given flag is really true!'

**Python for循环**可以遍历任何序列的项目，如一个列表或者一个字符串。

**语法：**

for循环的语法格式如下：

for iterating\_var in sequence:

statements(s)

**实例**

for letter in 'Python': # 第一个实例

print '当前字母 :', letter

fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # 第二个实例

print '当前水果 :', fruit

for index in range(len(fruits)):

print '当前水果 :', fruits[index]

注意：内置函数 len() 和 range(),函数 len() 返回列表的长度，即元素的个数。 range返回一个序列的数。

在 python 中，for … else 表示这样的意思，for 中的语句和普通的没有区别，else 中的语句会在循环正常执行完（即 for 不是通过 break 跳出而中断的）的情况下执行，while … else 也是一样。

for num in range(10,20): # 迭代 10 到 20 之间的数字

for i in range(2,num): # 根据因子迭代

if num%i == 0: # 确定第一个因子

j=num/i # 计算第二个因子

print '%d 等于 %d \* %d' % (num,i,j)

break # 跳出当前循环

else: # 循环的 else 部分

print num, '是一个质数'

这个else是与for i这个循环对应的

Python 语言允许在一个循环体里面嵌入另一个循环。

**Python for 循环嵌套语法：**

for iterating\_var in sequence:

for iterating\_var in sequence:

statements(s)

statements(s)

**Python while 循环嵌套语法：**

while expression:

while expression:

statement(s)

statement(s)

**实例：**

以下实例使用了嵌套循环输出2~100之间的素数：

i = 2

while(i < 100):

j = 2

while(j <= (i/j)):

if not(i%j): break

j = j + 1

if (j > i/j) : print i, " 是素数"

i = i + 1

## 3.14 数学函数

数学函数：

|  |  |
| --- | --- |
| [cmp(x, y)](http://www.runoob.com/python/func-number-cmp.html) | 如果 x < y 返回 -1, 如果 x == y 返回 0, 如果 x > y 返回 1 |

Python包含以下常用随机数函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [choice(seq)](http://www.runoob.com/python/func-number-choice.html) | 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])](http://www.runoob.com/python/func-number-randrange.html) | 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()](http://www.runoob.com/python/func-number-random.html) | 随机生成下一个实数，它在[0,1)范围内。 |
| [seed([x])](http://www.runoob.com/python/func-number-seed.html) | 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)](http://www.runoob.com/python/func-number-shuffle.html) | 将序列的所有元素随机排序 |
| [uniform(x, y)](http://www.runoob.com/python/func-number-uniform.html) | 随机生成下一个实数，它在[x,y]范围内。 |

# 4 Python函数

## 定义一个函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

* 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号**()**。
* 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
* 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
* 函数内容以冒号起始，并且缩进。
* **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

**语法**

def functionname( parameters ):

"函数\_文档字符串"

function\_suite

return [expression]

默认情况下，参数值和参数名称是按函数声明中定义的的顺序匹配起来的。

## 参数传递

在 python 中，类型属于对象，变量是没有类型的：

a=[1,2,3]

a="Runoob"

以上代码中，**[1,2,3]** 是 List 类型，**"Runoob"** 是 String 类型，而变量 a 是没有类型，她仅仅是一个对象的引用（一个指针），可以是 List 类型对象，也可以指向 String 类型对象。

**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 的第二个元素值更改，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**

以下是调用函数时可使用的正式参数类型：

* 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，def printme( str )—》调用时也得printme( str )，不能这样调用printme()。
* 关键字参数：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，如def printinfo( name, age )，调用时printinfo( age=50, name="miki" )，可以不按照参数的声明顺序;
* 默认参数：调用函数时，缺省参数的值如果没有传入，则被认为是默认值。def printinfo( name, age = 35 )。
* 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 )

加了星号（\*）的变量名会存放所有未命名的变量参数。选择不多传参数也可。如下实例：

#!/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

如下实例：

# 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 )

## return 语句

return语句[表达式]退出函数，选择性地向调用方返回一个表达式。不带参数值的return语句返回None。

# 可写函数说明

def sum( arg1, arg2 ):

# 返回2个参数的和."

total = arg1 + arg2

print "函数内 : ", total

return total;

# 调用sum函数

total = sum( 10, 20 );

以上实例输出结果：

函数内 : 30

## 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：可以在整个程序范围内访问。
* 局部变量：只能在其被声明的函数内部访问，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。

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。如下实例：

total = 0; # 这是一个全局变量

# 可写函数说明

def sum( arg1, arg2 ):

#返回2个参数的和."

total = arg1 + arg2; # total在这里是局部变量.

print "函数内是局部变量 : ", total

return total;

#调用sum函数

sum( 10, 20 );

print "函数外是全局变量 : ", total

以上实例输出结果：

函数内是局部变量 : 30

函数外是全局变量 : 0

# 5 Python 模块

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。

模块也是Python对象，具有随机的名字属性用来绑定或引用。

简单地说，模块就是一个保存了Python代码的文件。模块能定义函数，类和变量。模块里也能包含可执行的代码。

**例子**

一个叫做aname的模块里的Python代码一般都能在一个叫aname.py的文件中找到。下例是个简单的模块support.py。

def print\_func( par ):

print "Hello : ", par

return

## import 语句

想使用Python源文件，只需在另一个源文件里执行import语句，语法如下：

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

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。如想要导入模块support.py，需要把命令放在脚本的顶端：

#!/usr/bin/python

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

# 导入模块

import support

# 现在可以调用模块里包含的函数了

support.print\_func("Zara")

以上实例输出结果：

Hello : Zara

一个模块只会被导入一次，不管你执行了多少次import。这样可以防止导入模块被一遍又一遍地执行。

## 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。**

## 定位模块

当你导入一个模块，Python解析器对模块位置的搜索顺序是：

* 当前目录
* 如果不在当前目录，Python 则搜索在 shell 变量 PYTHONPATH 下的每个目录。
* 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。

## PYTHONPATH变量

作为环境变量，PYTHONPATH由装在一个列表里的许多目录组成。PYTHONPATH的语法和shell变量PATH的一样。

在Windows系统，典型的PYTHONPATH如下：

set PYTHONPATH=c:\python20\lib;

在UNIX系统，典型的PYTHONPATH如下：

set PYTHONPATH=/usr/local/lib/python

## 命名空间和作用域

变量是拥有匹配对象的名字（标识符）。命名空间是一个包含了变量名称们（键）和它们各自相应的对象们（值）的字典。

一个Python表达式可以访问局部命名空间和全局命名空间里的变量。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。

每个函数都有自己的命名空间。类的方法的作用域规则和通常函数的一样。

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，如果要给全局变量在一个函数里赋值，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。

例如，我们在全局命名空间里定义一个变量money。我们再在函数内给变量money赋值，然后Python会假定money是一个局部变量。然而，我们并没有在访问前声明一个局部变量money，结果就是会出现一个UnboundLocalError的错误。 实例：

Money = 2000

def AddMoney():

# 函数内部修改全局变量

global Money

Money = Money + 1

print Money

AddMoney()

print Money

结果：

2000

2001

## 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。

考虑一个在Phone目录下的pots.py文件。这个文件有如下源代码：

#!/usr/bin/python

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

def Pots():

print "I'm Pots Phone"

同样地，我们有另外两个保存了不同函数的文件：

* Phone/Isdn.py 含有函数Isdn()
* Phone/G3.py 含有函数G3()

现在，在Phone目录下创建file \_\_init\_\_.py：

* Phone/\_\_init\_\_.py

当你导入Phone时，为了能够使用所有函数，你需要在\_\_init\_\_.py里使用显式的导入语句，如下：

from Pots import Pots

from Isdn import Isdn

from G3 import G3

当你把这些代码添加到\_\_init\_\_.py之后，导入Phone包的时候这些类就全都是可用的了。

#!/usr/bin/python

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

# 导入 Phone 包

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

以上实例输出结果：

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

如上，为了举例，我们只在每个文件里放置了一个函数，但其实你可以放置许多函数。你也可以在这些文件里定义Python的类，然后为这些类建一个包。

# 6 Python 文件I/O

## 读取键盘输入

Python提供了两个内置函数从标准输入读入一行文本，默认的标准输入是键盘。如下：

* raw\_input
* input

### 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）：

#!/usr/bin/python

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

str = raw\_input("请输入：");

print "你输入的内容是: ", str

### 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

现在，您已经可以向标准输入和输出进行读写。现在，来看看怎么读写实际的数据文件。

Python 提供了必要的函数和方法进行默认情况下的文件基本操作。你可以用 **file** 对象做大部分的文件操作。

### 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** |
| 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。 |
| 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。 |
| w | 打开一个文件只用于写入。如果该文件已存在则将其覆盖。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将其覆盖。如果该文件不存在，创建新文件。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则将其覆盖。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将其覆盖。如果该文件不存在，创建新文件。 |
| 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

一个文件被打开后，你有一个file对象，你可以得到有关该文件的各种信息。

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。 |

如下实例：

#!/usr/bin/python

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

# 打开一个文件

fo = open("foo.txt", "wb")

print "文件名: ", fo.name

print "是否已关闭 : ", fo.closed

print "访问模式 : ", fo.mode

print "末尾是否强制加空格 : ", fo.softspace

以上实例输出结果：

文件名: foo.txt

是否已关闭 : False

访问模式 : wb

末尾是否强制加空格 : 0

### 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);

在这里，被传递的参数是要写入到已打开文件的内容。

#!/usr/bin/python

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

# 打开一个文件

fo = open("foo.txt", "wb")

fo.write( "www.runoob.com!\nVery good site!\n");

#str = fo.read(10);

# 关闭打开的文件

fo.close()

上述方法会创建foo.txt文件，并将收到的内容写入该文件，并最终关闭文件。如果你打开这个文件，将看到以下内容:

$ cat foo.txt

www.runoob.com!

Very good site!

### 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。

例子：

就用我们上面创建的文件foo.txt。

# 打开一个文件

fo = open("foo.txt", "r+")

str = fo.read(10);

print "读取的字符串是 : ", str

# 查找当前位置

position = fo.tell();

print "当前文件位置 : ", position

# 把指针再次重新定位到文件开头

position = fo.seek(0, 0);

str = fo.read(10);

print "重新读取字符串 : ", str

# 关闭打开的文件

fo.close()

以上实例输出结果：

读取的字符串是 : www.runoob

当前文件位置 : 10

重新读取字符串 : www.runoob

## 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： 需要两个参数，当前的文件名和新文件名。

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。

os.remove(file\_name)

实例：

import os

# 重命名文件test1.txt到test2.txt。

os.rename( "test1.txt", "test2.txt" )

# 删除一个已经存在的文件test2.txt

os.remove("test2.txt")

## 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。

### 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')

**实例：**

import os

# 创建目录test

os.mkdir("test")

# 将当前目录改为"/home/newdir"

os.chdir("/home/newdir")

# 给出当前的目录

print os.getcwd()

# 删除”/tmp/test”目录

os.rmdir( "/tmp/test" )

# Python File(文件) 方法

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()](http://www.runoob.com/python/file-close.html)  关闭文件。关闭后文件不能再进行读写操作。 |
| 2 | [file.flush()](http://www.runoob.com/python/file-flush.html)  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()](http://www.runoob.com/python/file-fileno.html)  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()](http://www.runoob.com/python/file-isatty.html)  如果文件连接到一个终端设备返回 True，否则返回 False。 |
| 5 | [file.next()](http://www.runoob.com/python/file-next.html)  返回文件下一行。 |
| 6 | [file.read([size])](http://www.runoob.com/python/python-file-read.html)  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])](http://www.runoob.com/python/file-readline.html)  读取整行，包括 "\n" 字符。 |
| 8 | [file.readlines([sizehint])](http://www.runoob.com/python/file-readlines.html)  读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比sizhint较大, 因为需要填充缓冲区。 |
| 9 | [file.seek(offset[, whence])](http://www.runoob.com/python/file-seek.html)  设置文件当前位置 |
| 10 | [file.tell()](http://www.runoob.com/python/file-tell.html)  返回文件当前位置。 |
| 11 | [file.truncate([size])](http://www.runoob.com/python/file-truncate.html)  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)](http://www.runoob.com/python/python-file-write.html)  将字符串写入文件，没有返回值。 |
| 13 | [file.writelines(sequence)](http://www.runoob.com/python/file-writelines.html)  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 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)](http://www.runoob.com/python/os-access.html) 检验权限模式 |
| 2 | [os.chdir(path)](http://www.runoob.com/python/os-chdir.html) 改变当前工作目录 |
| 3 | [os.chflags(path, flags)](http://www.runoob.com/python/os-chflags.html) 设置路径的标记为数字标记。 |
| 4 | [os.chmod(path, mode)](http://www.runoob.com/python/os-chmod.html) 更改权限 |
| 5 | [os.chown(path, uid, gid)](http://www.runoob.com/python/os-chown.html) 更改文件所有者 |
| 6 | [os.chroot(path)](http://www.runoob.com/python/os-chroot.html) 改变当前进程的根目录 |
| 7 | [os.close(fd)](http://www.runoob.com/python/os-close.html) 关闭文件描述符 fd |
| 8 | [os.closerange(fd\_low, fd\_high)](http://www.runoob.com/python/os-closerange.html) 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)](http://www.runoob.com/python/os-dup.html) 复制文件描述符 fd |
| 10 | [os.dup2(fd, fd2)](http://www.runoob.com/python/os-dup2.html) 将一个文件描述符 fd 复制到另一个 fd2 |
| **11** | [**os.fchdir(fd)**](http://www.runoob.com/python/os-fchdir.html) **通过文件描述符改变当前工作目录** |
| 12 | [os.fchmod(fd, mode)](http://www.runoob.com/python/os-fchmod.html) 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)](http://www.runoob.com/python/os-fchown.html) 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)](http://www.runoob.com/python/os-fdatasync.html) 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]])](http://www.runoob.com/python/os-fdopen.html) 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)](http://www.runoob.com/python/os-fpathconf.html) 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)](http://www.runoob.com/python/os-fstat.html) 返回文件描述符fd的状态，像stat()。 |
| 18 | [os.fstatvfs(fd)](http://www.runoob.com/python/os-fstatvfs.html) 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)](http://www.runoob.com/python/os-fsync.html) 强制将文件描述符为fd的文件写入硬盘。 |
| 20 | [os.ftruncate(fd, length)](http://www.runoob.com/python/os-ftruncate.html) 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()](http://www.runoob.com/python/os-getcwd.html) 返回当前工作目录 |
| 22 | [os.getcwdu()](http://www.runoob.com/python/os-getcwdu.html) 返回一个当前工作目录的Unicode对象 |
| 23 | [os.isatty(fd)](http://www.runoob.com/python/os-isatty.html) 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)](http://www.runoob.com/python/os-lchflags.html) 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)](http://www.runoob.com/python/os-lchmod.html) 修改连接文件权限 |
| 26 | [os.lchown(path, uid, gid)](http://www.runoob.com/python/os-lchown.html) 更改文件所有者，类似 chown，但是不追踪链接。 |
| 27 | [os.link(src, dst)](http://www.runoob.com/python/os-link.html) 创建硬链接，名为参数 dst，指向参数 src |
| 28 | [os.listdir(path)](http://www.runoob.com/python/os-listdir.html) 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)](http://www.runoob.com/python/os-lseek.html) 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)](http://www.runoob.com/python/os-lstat.html) 像stat(),但是没有软链接 |
| 31 | [os.major(device)](http://www.runoob.com/python/os-major.html) 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 field)。 |
| 32 | [os.makedev(major, minor)](http://www.runoob.com/python/os-makedev.html) 以major和minor设备号组成一个原始设备号 |
| 33 | [os.makedirs(path[, mode])](http://www.runoob.com/python/os-makedirs.html) 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)](http://www.runoob.com/python/os-minor.html) 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])](http://www.runoob.com/python/os-mkdir.html) 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])](http://www.runoob.com/python/os-mkfifo.html) 创建命名管道，mode 为数字，默认为 0666 (八进制) |
| 37 | [os.mknod(filename[, mode=0600, device])](http://www.runoob.com/python/os-mknod.html) 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])](http://www.runoob.com/python/os-open.html) 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()](http://www.runoob.com/python/os-openpty.html) 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)](http://www.runoob.com/python/os-pathconf.html) 返回相关文件的系统配置信息。 |
| 41 | [os.pipe()](http://www.runoob.com/python/os-pipe.html) 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]])](http://www.runoob.com/python/os-popen.html) 从一个 command 打开一个管道 |
| 43 | [os.read(fd, n)](http://www.runoob.com/python/os-read.html) 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)](http://www.runoob.com/python/os-readlink.html) 返回软链接所指向的文件 |
| 45 | [os.remove(path)](http://www.runoob.com/python/os-remove.html) 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)](http://www.runoob.com/python/os-removedirs.html) 递归删除目录。 |
| 47 | [os.rename(src, dst)](http://www.runoob.com/python/os-rename.html) 重命名文件或目录，从 src 到 dst |
| 48 | [os.renames(old, new)](http://www.runoob.com/python/os-renames.html) 递归地对目录进行更名，也可以对文件进行更名。 |
| 49 | [os.rmdir(path)](http://www.runoob.com/python/os-rmdir.html) 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)](http://www.runoob.com/python/os-stat.html) 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])](http://www.runoob.com/python/os-stat_float_times.html) 决定stat\_result是否以float对象显示时间戳 |
| 52 | [os.statvfs(path)](http://www.runoob.com/python/os-statvfs.html) 获取指定路径的文件系统统计信息 |
| 53 | [os.symlink(src, dst)](http://www.runoob.com/python/os-symlink.html) 创建一个软链接 |
| 54 | [os.tcgetpgrp(fd)](http://www.runoob.com/python/os-tcgetpgrp.html) 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)](http://www.runoob.com/python/os-tcsetpgrp.html) 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]])](http://www.runoob.com/python/os-tempnam.html) 返回唯一的路径名用于创建临时文件。 |
| 57 | [os.tmpfile()](http://www.runoob.com/python/os-tmpfile.html) 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()](http://www.runoob.com/python/os-tmpnam.html) 为创建一个临时文件返回一个唯一的路径 |
| 59 | [os.ttyname(fd)](http://www.runoob.com/python/os-ttyname.html) 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)](http://www.runoob.com/python/os-unlink.html) 删除文件路径 |
| 61 | [os.utime(path, times)](http://www.runoob.com/python/os-utime.html) 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]]])](http://www.runoob.com/python/os-walk.html) 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)](http://www.runoob.com/python/os-write.html) 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 |

**参考地址：**

* http://kuanghy.github.io/python-os/
* http://python.usyiyi.cn/python\_278/library/os.html

# Python 异常处理

python提供了两个非常重要的功能来处理python程序在运行中出现的异常和错误。你可以使用该功能来调试python程序。

## 什么是异常？

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。

一般情况下，在Python无法正常处理程序时就会发生一个异常。

异常是Python对象，表示一个错误。

当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

## 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。

**实例**

下面是简单的例子，它打开一个文件，在该文件中的内容写入内容，且并未发生异常：

try:

fh = open("testfile", "w")

fh.write("这是一个测试文件，用于测试异常!!")

except IOError:

print "Error: 没有找到文件或读取文件失败"

else:

print "内容写入文件成功"

fh.close()

以上程序输出结果：

$ python test.py

内容写入文件成功

$ cat testfile # 查看写入的内容

这是一个测试文件，用于测试异常!!

**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。

# 定义函数

def temp\_convert(var):

try:

return int(var)

except ValueError, Argument:

print "参数没有包含数字\n", Argument

# 调用函数

temp\_convert("xyz");

## 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）参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"。

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。

**实例**

一个异常可以是一个字符串，类或对象。 Python的内核提供的异常，大多数都是实例化的类，这是一个类的实例的参数。

定义一个异常非常简单，如下所示：

def functionName( level ):

if level < 1:

raise Exception("Invalid level!", level)

# 触发异常后，后面的代码就不会再执行

**注意：**为了能够捕获异常，"except"语句必须有用相同的异常来抛出类对象或者字符串。

例如我们捕获以上异常，"except"语句如下所示：

try:

正常逻辑

except "Invalid level!":

触发自定义异常

else:

其余代码

**实例：**

# 定义函数

def mye( level ):

if level < 1:

raise Exception("Invalid level!", level)

# 触发异常后，后面的代码就不会再执行

try:

mye(0) // 触发异常

except "Invalid level!":

print 1

else:

print 2

执行以上代码，输出结果为：

$ python test.py

Traceback (most recent call last):

File "test.py", line 11, in <module>

mye(0)

File "test.py", line 7, in mye

raise Exception("Invalid level!", level)

Exception: ('Invalid level!', 0)

## 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