# 1 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

这将提示你输入任意字符串，然后在屏幕上显示相同的字符串。当我输入"Hello Python！"，它的输出如下：

请输入：Hello Python！你输入的内容是: Hello Python！

### **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.nameprint "是否已关闭 : ", fo.closedprint "访问模式 : ", fo.modeprint "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();

例子：

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

# 打开一个文件

fo = open("foo.txt", "wb")print "文件名: ", fo.name

# 关闭打开的文件

fo.close()

以上实例输出结果：

文件名: foo.txt

读写文件：

file对象提供了一系列方法，能让我们的文件访问更轻松。来看看如何使用read()和write()方法来读取和写入文件。

### **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。

例子：

#!/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");

# 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。

### **例子：**

这里我们用到以上创建的 foo.txt 文件。

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

# 打开一个文件

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

str = fo.read(10);print "读取的字符串是 : ", 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

文件位置：

## 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。

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

# 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：

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)

例子：

下例将重命名一个已经存在的文件test1.txt。

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

import os

# 重命名文件test1.txt到test2.txt。

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

### **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)

例子：

下例将删除一个已经存在的文件test2.txt。

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

import os

# 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")

例子：

下例将在当前目录下创建一个新目录test。

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

import os

# 创建目录test

os.mkdir("test")

### **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")

例子：

下例将进入"/home/newdir"目录。

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

import os

# 将当前目录改为"/home/newdir"

os.chdir("/home/newdir")

getcwd()方法：

getcwd()方法显示当前的工作目录。

语法：

os.getcwd()

例子：

下例给出当前目录：

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

import os

# 给出当前的目录

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')

例子：

以下是删除" /tmp/test"目录的例子。目录的完全合规的名称必须被给出，否则会在当前目录下搜索该目录。

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

import os

# 删除”/tmp/test”目录

os.rmdir( "/tmp/test" )

## 文件、目录相关的方法

三个重要的方法来源能对Windows和Unix操作系统上的文件及目录进行一个广泛且实用的处理及操控，如下：

* [File 对象方法](http://www.runoob.com/python/file-methods.html): file对象提供了操作文件的一系列方法。
* [OS 对象方法](http://www.runoob.com/python/os-file-methods.html" \t "http://www.runoob.com/python/_blank): 提供了处理文件及目录的一系列方法。

# 2 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。 |

# 3 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。

* 异常处理: 本站Python教程会具体介绍。
* 断言(Assertions):本站Python教程会具体介绍。

## 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 |

## 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：

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

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!!

### **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，但文件没有写入权限，发生了异常：

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

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()

在执行代码前为了测试方便，我们可以先去掉 testfile 文件的写权限，命令如下：

chmod -w testfile

再执行以上代码：

$ python test.py Error: 没有找到文件或读取文件失败

## 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

### **实例**

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

try:

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

fh.write("这是一个测试文件，用于测试异常!!")finally:

print "Error: 没有找到文件或读取文件失败"

如果打开的文件没有可写权限，输出如下所示：

$ python test.py Error: 没有找到文件或读取文件失败

同样的例子也可以写成如下方式：

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

try:

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

try:

fh.write("这是一个测试文件，用于测试异常!!")

finally:

print "关闭文件"

fh.close()except IOError:

print "Error: 没有找到文件或读取文件失败"

当在try块中抛出一个异常，立即执行finally块代码。

finally块中的所有语句执行后，异常被再次触发，并执行except块代码。

参数的内容不同于异常。

## 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。

### **实例**

以下为单个异常的实例：

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

# 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");

以上程序执行结果如下：

$ python test.py 参数没有包含数字

invalid literal for int() with base 10: '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**

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

# 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 1else:

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