# 错误和异常

## 语法错误

Python 的语法错误或者称之为解析错，是初学者经常碰到的，如下实例

>>>while True

print('Hello world')

File "<stdin>", line 1, in ? while True print('Hello world') ^ SyntaxError: invalid syntax

这个例子中，函数 print() 被检查到有错误，是它前面缺少了一个冒号（:）。

语法分析器指出了出错的一行，并且在最先找到的错误的位置标记了一个小小的箭头。

## 异常

即便Python程序的语法是正确的，在运行它的时候，也有可能发生错误。运行期检测到的错误被称为异常。

大多数的异常都不会被程序处理，都以错误信息的形式展现在这里:

>>>10 \* (1/0)

Traceback (most recent call last):

File "<stdin>", line 1, in ? ZeroDivisionError: division by zero

>>> 4 + spam\*3

Traceback (most recent call last):

File "<stdin>", line 1, in ? NameError: name 'spam' is not defined

>>> '2' + 2

Traceback (most recent call last):

File "<stdin>", line 1, in ? TypeError: Can't convert 'int' object to str implicitly

异常以不同的类型出现，这些类型都作为信息的一部分打印出来: 例子中的类型有 ZeroDivisionError，NameError 和 TypeError。

错误信息的前面部分显示了异常发生的上下文，并以调用栈的形式显示具体信息。

## 异常处理

以下例子中，让用户输入一个合法的整数，但是允许用户中断这个程序（使用 Control-C 或者操作系统提供的方法）。用户中断的信息会引发一个 KeyboardInterrupt 异常。

KeyboardInterrupt 异常。

>>>while True:

try:

x = int(input("Please enter a number: "))

break

except ValueError: print("Oops! That was no valid number. Try again ")

try语句按照如下方式工作；

* 首先，执行try子句（在关键字try和关键字except之间的语句）
* 如果没有异常发生，忽略except子句，try子句执行后结束。
* 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。最后执行 try 语句之后的代码。
* 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的try子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如:

except (RuntimeError, TypeError, NameError):

pass

最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再次把异常抛出。

import sys

try:

f = open('myfile.txt') s = f.readline() i = int(s.strip())

except OSError as err:

print("OS error: {0}".format(err))

except ValueError:

print("Could not convert data to an integer.")

except:

print("Unexpected error:", sys.exc\_info()[0])

raise

try except 语句还有一个可选的else子句，如果使用这个子句，那么必须放在所有的except子句之后。这个子句将在try子句没有发生任何异常的时候执行。例如:

for arg in sys.argv[1:]:

try:

f = open(arg, 'r')

except IOError:

print('cannot open', arg)

else:

print(arg, 'has', len(f.readlines()), 'lines')

f.close()

使用 else 子句比把所有的语句都放在 try 子句里面要好，这样可以避免一些意想不到的、而except又没有捕获的异常。

异常处理并不仅仅处理那些直接发生在try子句中的异常，而且还能处理子句中调用的函数（甚至间接调用的函数）里抛出的异常。例如:

>>>def this\_fails():

x = 1/0

>>> try: this\_fails()

except ZeroDivisionError as err:

print('Handling run-time error:', err)

Handling run-time error: int division or modulo by zero

## 抛出异常

Python 使用 raise 语句抛出一个指定的异常。例如:

>>>raise NameError('HiThere')

Traceback (most recent call last): File "<stdin>", line 1, in ? NameError: HiThere

raise 唯一的一个参数指定了要被抛出的异常。它必须是一个异常的实例或者是异常的类（也就是 Exception 的子类）。

如果你只想知道这是否抛出了一个异常，并不想去处理它，那么一个简单的 raise 语句就可以再次把它抛出。

>>> try:

raise NameError('HiThere')

except NameError:

print('An exception flew by!')

raise

An exception flew by! Traceback (most recent call last): File "<stdin>", line 2, in ? NameError: HiThere

**try**:

print('try...')

r = 10 / 0

print('result:', r)

**except** ZeroDivisionError **as** e:

print('except:', e)

**finally**:

print('finally...')

print('END')

当我们认为某些代码可能会出错时，就可以用try来运行这段代码，如果执行出错，则后续代码不会继续执行，而是直接跳转至错误处理代码，即except语句块，执行完except后，如果有finally语句块，则执行finally语句块，至此，执行完毕。

**try**:

print('try...')

r = 10 / int('2')

print('result:', r)

**except** ValueError **as** e:

print('ValueError:', e)

**except** ZeroDivisionError **as** e:

print('ZeroDivisionError:', e)

**else**:

print('no error!')

**finally**:

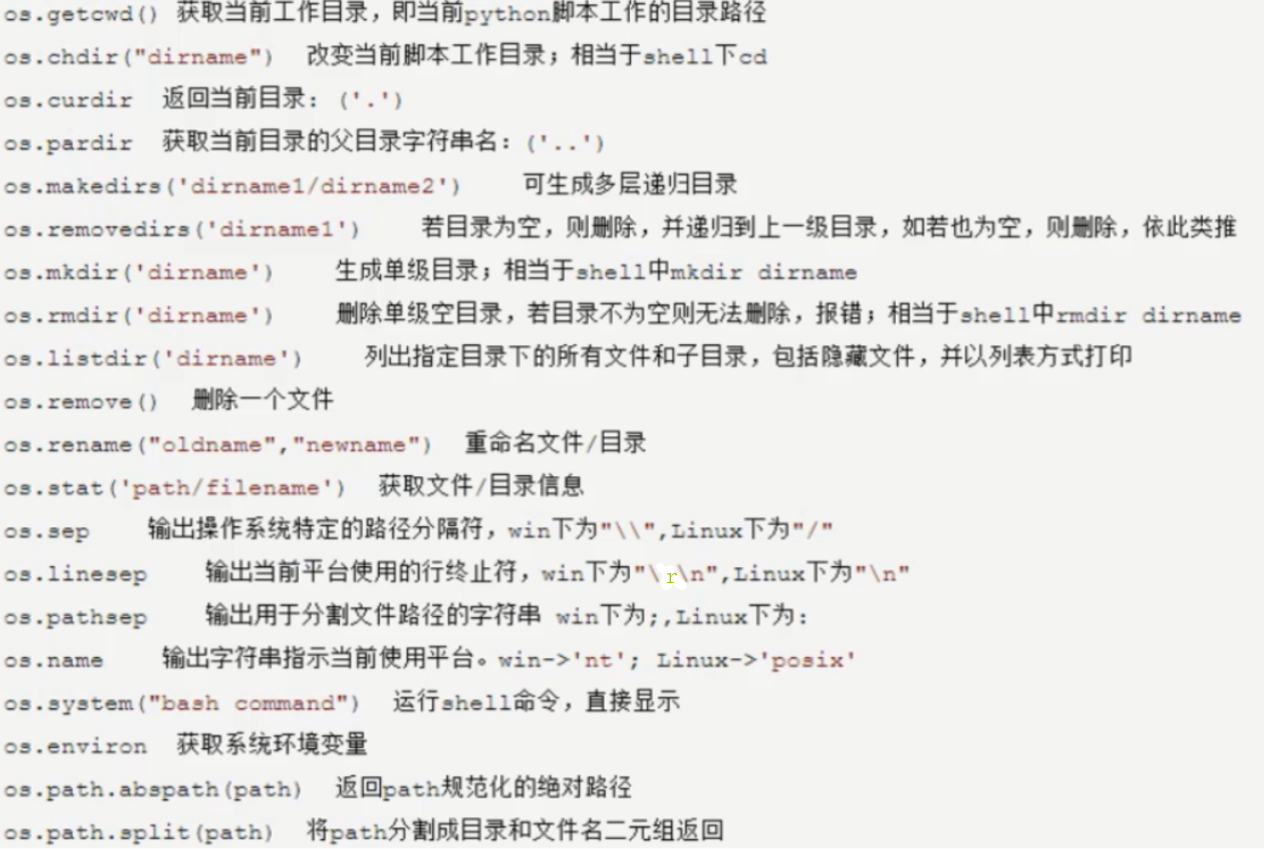
print('finally...')

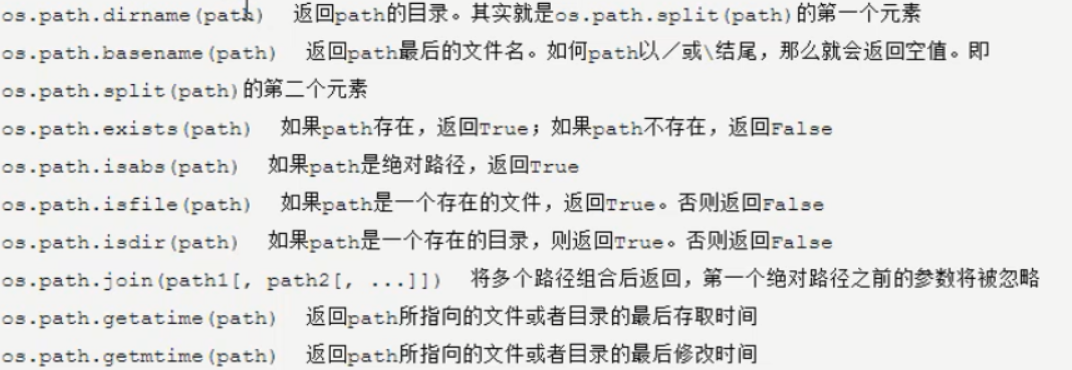
print('END')

int()函数可能会抛出ValueError，所以我们用一个except捕获ValueError，用另一个except捕获ZeroDivisionError。

此外，如果没有错误发生，可以在except语句块后面加一个else，当没有错误发生时，会自动执行else语句：

## Python ：Os模块





# **python之sys模块详解**



<https://www.cnblogs.com/Archie-s/p/6860301.html>