Standard Exception trong Python

ython cung cấp hai đặc điểm quan trọng là Xử lý ngoại lệ (Exception Handling) và Assertion để xử lý bất kỳ lỗi không mong đợi nào trong các chương trình Python của bạn và để thêm khả năng debug tới các chương trình đó.

Chương này chúng ta trước hết giới thiệu qua về các Exception chuẩn có trong Python. Bảng dưới đây liệt kê tất cả ngoại lệ chuẩn có sẵn trong Python:

|  |  |
| --- | --- |
| **Exception** | **Miêu tả** |
| Exception | Lớp cơ sở (base class) của tất cả các ngoại lệ |
| StopIteration | Được tạo khi phương thức next() của một iterator không trỏ tới bất kỳ đối tượng nào |
| SystemExit | Được tạo bởi hàm sys.exit() |
| StandardError | Lớp cơ sở của tất cả exception có sẵn ngoại trừ StopIteration và SystemExit. |
| ArithmeticError | Lớp cơ sở của tất cả các lỗi xảy ra cho phép tính số học |
| OverflowError | Được tạo khi một phép tính vượt quá giới hạn tối đa cho một kiểu số |
| FloatingPointError | Được tạo khi một phép tính số thực thất bại |
| ZeroDivisonError | Được tạo khi thực hiện phép chia cho số 0 với tất cả kiểu số |
| AssertionError | Được tạo trong trường hợp lệnh assert thất bại |
| AttributeError | Được tạo trong trường hợp tham chiếu hoặc gán thuộc tính thất bại |
| EOFError | Được tạo khi không có input nào từ hàm raw\_input() hoặc hàm input() và tới EOF (viết tắt của end of file) |
| ImportError | Được tạo khi một lệnh import thất bại |
| KeyboardInterrupt | Được tạo khi người dùng ngắt việc thực thi chương trình, thường là bởi nhấn Ctrl+c |
| LookupError | Lớp cơ sở cho tất cả các lỗi truy cứu |
| IndexError | Được tạo khi một chỉ mục không được tìm thấy trong một dãy (sequence) |
| KeyError | Được tạo khi key đã cho không được tìm thấy trong Dictionary |
| NameError | Được tạo khi một định danh không được tìm thấy trong local hoặc global namespace |
| UnboundLocalError | Được tạo khi cố gắng truy cập một biến cục bộ từ một hàm hoặc phương thức nhưng mà không có giá trị nào đã được gán cho nó |
| EnvironmentError | Lớp cơ sở cho tất cả ngoại lệ mà xuất hiện ở ngoài môi trường Python |
| IOError | Được tạo khi hoạt động i/o thất bại, chẳng hạn như lệnh print hoặc hàm open() khi cố gắng mở một file không tồn tại |
| OSError | Được do các lỗi liên quan tới hệ điều hành |
| SyntaxError | Được tạo khi có một lỗi liên quan tới cú pháp |
| IndentationError | Được tạo khi độ thụt dòng code không được xác định hợp lý |
| SystemError | Được tạo khi trình thông dịch tìm thấy một vấn đề nội tại, nhưng khi lỗi này được bắt gặp thì trình thông dịch không thoát ra |
| SystemExit | Được tạo khi trình thông dịch thoát ra bởi sử dụng hàm sys.exit(). Nếu không được xử lý trong code, sẽ làm cho trình thông dịch thoát |
| TypeError | Được tạo khi một hoạt động hoặc hàm sử dụng một kiểu dữ liệu không hợp lệ |
| ValueError | Được tạo khi hàm đã được xây dựng sẵn có các kiểu tham số hợp lệ nhưng các giá trị được xác định cho tham số đó là không hợp lệ |
| RuntimeError | Được tạo khi một lỗi đã được tạo ra là không trong loại nào |
| NotImplementedError | Được tạo khi một phương thức abstract, mà cần được triển khai trong một lớp được kế thừa, đã không được triển khai thực sự |

Cách đơn giản nhất để nghĩ về một Assertion là xem nó giống như một lệnh raise-if-no. Một biểu thức được kiểm nghiệm, và nếu kết quả là false, thì một exception được tạo ra.

Một Assertion là một sanity-test mà bạn có thể bật hoặc tắt khi bạn được thực hiện với sự kiểm nghiệm chương trình của bạn.

Assertion được mang bởi lệnh assert. Các lập trình viên thường đặt các Assertion tại phần đầu của một hàm để kiểm tra tính hợp lệ của input, và sau một lời gọi hàm để kiểm tra tính hợp của output.

## **Lệnh assert trong Python**

Khi bắt gặp lệnh assert, Python ước lượng biểu thức đi kèm, mà hy vọng rằng là true. Nếu biểu thức là false, thì Python tạo một ngoại lệ là **AssertionError**.

Dưới đây là cú pháp cho lệnh assert:

assert bieu\_thuc[, cac\_tham\_so]

Nếu Assertion thất bại, Python sử dụng cac\_tham\_so là tham số cho AssertionError. Các ngoại lệ AssertionError có thể được bắt và được xử lý giống như bất kỳ ngoại lệ khác bởi sử dụng lệnh try-except, nhưng nếu không được xử lý thì chúng sẽ kết thúc chương trình và sẽ tạo một traceback.

## **Ví dụ**

Dưới đây là một hàm để chuyển đổi từ độ K thành độ F.

def ChuyenKF(Nhietdo):

assert (Nhietdo >= 0),"Lanh hon do khong tuyet doi!"

return ((Nhietdo-273)\*1.8)+32

print ChuyenKF(273)

print int(ChuyenKF(505.78))

print ChuyenKF(-5)

Khi code trên được thực thi sẽ cho kết quả:

32.0

451

Traceback (most recent call last):

File "test.py", line 9, in <module>

print ChuyenKF(-5)

File "test.py", line 4, in ChuyenKF

assert (Nhietdo >= 0),"Lanh hon do khong tuyet doi!"

AssertionError: Lanh hon do khong tuyet doi!

## **Exception là gì?**

Ngoại lệ có thể là bất kỳ điều kiện bất thường nào trong chương trình mà phá vỡ luồng thực thi chương trình đó. Bất cứ khi nào một ngoại lệ xuất hiện, mà không được xử lý, thì chương trình ngừng thực thi và vì thế code không được thực thi.

Python đã định nghĩa sẵn rất nhiều ngoại lệ, mà đã được trình bày trong chương Standard Exception. Trong chương này chúng ta sẽ tìm hiểu cách xử lý ngoại lệ cũng như cách tạo các Custom Exception như thế nào.

## **Xử lý ngoại lệ trong Python**

Nếu bạn thấy bất cứ code nào là khả nghi (có thể gây ra ngoại lệ) thì bạn có thể phòng thủ chương trình của mình bằng cách đặt các khối code khả nghi này trong một khối try. Khối try này được theo sau bởi lệnh except. Sau đó, nó được theo sau bởi các lệnh mà xử lý vấn đề đó.

### **Cú pháp**

Dưới đây là cú pháp của khối *try....except...else* trong Python:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except *ExceptionI*:

Neu co ExceptionI, thi thuc thi khoi code nay

except *ExceptionII*:

Neu co ExceptionII, thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao thi thuc thi khoi code nay

Dưới đây là một số điểm bạn cần lưu ý:

* Phần code khả nghi mà có khả năng tạo exception cần được bao quanh trong khối try.
* Khối try được theo sau bởi lệnh except. Có thể có một hoặc nhiều lệnh except với một khối try đơn.
* Lệnh except xác định exception mà xảy ra. Trong trường hợp mà exception đó xảy ra, thì lệnh tương ứng được thực thi.
* Ở cuối khối try, bạn có thể cung cấp lệnh else. Nó được thực thi khi không có exception nào xảy ra. Khối else là địa điểm tốt cho code mà không cần sự bảo vệ của khối try.

### **Ví dụ**

Ví dụ sau mở một file, ghi nội dung vào file này và sau đó đóng file, tất cả hoạt động đều thành công:

try:

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

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

except IOError:

print "Error: Khong tim thay file"

else:

print "Thanh cong ghi noi dung vao file"

fh.close()

Kết quả là:

Thanh cong ghi noi dung vao file

### **Ví dụ**

Ví dụ sau mở một file để ghi trong khi bạn không có quyền ghi, do đó nó sẽ tạo một exception:

try:

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

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

except IOError:

print "Error: Khong tim thay file"

else:

print "Thanh cong ghi noi dung vao file"

Kết quả là:

Error: Khong tim thay file

## **Mệnh đề except mà không xác định Exception trong Python**

Lệnh except cũng có thể được sử dụng mà không xác định exception nào. Lệnh try-except này bắt tất cả exception mà xuất hiện. Sử dụng loại lệnh try-except này không phải là sự thực hành lập trình tốt, bởi vì nó bắt tất cả exception nhưng không làm cho lập trình viên biết được căn nguyên của vấn đề làm xuất hiện exception đó.

### **Cú pháp**

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except:

Neu co bat ky exception nao, thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao, thi thuc thi khoi code nay

## **Mệnh đề except với nhiều exception trong Python**

Sử dụng cùng lệnh except như trên, bạn có thể khai báo nhiều exception như sau:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except(Exception1[, Exception2[,...ExceptionN]]]):

Neu co bat ky exeption nao trong danh sach,

thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao, thi thuc thi khoi code nay

## **Khối try-finally trong Python**

Trong trường hợp nếu có bất kỳ code nào mà người dùng muốn được thực thi, dù cho có xuất hiện exception hay không thì khối code đó có thể được đặt trong khối finally. Khối finally sẽ luôn luôn được thực thi bất chấp có hay không exception. Cú pháp của khối try-finally là:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

Do co exeption nen khoi nay bi bo qua

finally:

Khoi nay nen duoc thuc thi

......................

**Ghi chú**: Bạn có thể cung cấp một hoặc nhiều mệnh đề except, hoặc một mệnh đề finally, nhưng không được cung cấp cả hai. Ngoài ra bạn cũng không thể sử dụng mệnh đề else với một mệnh đề finally.

### **Ví dụ**

try:

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

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

finally:

print "Error: Khong tim thay file"

Nếu bạn không có quyền mở file trong chế độ ghi, thì code trên sẽ cho kết quả:

Error: Khong tim thay file

Ví dụ trên có thể được viết rõ ràng hơn như sau:

try:

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

try:

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

finally:

print "Chuan bi dong file"

fh.close()

except IOError:

print "Error: Khong tim thay file"

Khi một exception được ném trong khối *try*, thì trình thực thi ngay lập tức truyền tới khối *finally*. Sau đó tất cả các lệnh trong khối *finally* được thực thi, exception được tạo lại lần nữa và được xử lý bởi lệnh *except* nếu có mặt trong lớp trên tiếp theo của lệnh *try-except*.

## **Tham số của một Exception trong Python**

Một Exception có thể có một tham số, mà là một giá trị mà cung cấp thông tin bổ sung về vấn đề. Nội dung của tham số là đa dạng tùy vào các exception. Dưới đây là cú pháp:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except *Kieu\_exception, Tham\_so*:

Ban co the in gia tri cua Tham\_so tai day...

Nếu bạn viết code trên để xử lý một exception đơn, bạn có thể có một biến theo sau tên của exception trong lệnh except. Nếu bạn đang khai báo nhiều exception thì bạn có một biến theo sau tuple của các exception đó.

### **Ví dụ**

Ví dụ của một exception đơn:

# Dinh nghia mot ham o day.

def temp\_convert(var):

try:

return int(var)

except ValueError, Argument:

print "Tham so khong chua cac so\n", Argument

# Goi ham tren.

temp\_convert("xyz");

## **Tạo một Exception trong Python**

Bạn có thể ném tường minh một Exception trong Python bởi sử dụng lệnh raise. Cú pháp của lệnh raise như sau:

raise [Exception [, args ]]

Ở đây, Lop\_Exception là kiểu của exception và tham số value (tùy ý) là một giá trị. Để truy cập giá trị này thì từ khóa as được sử dụng.

### **Ví dụ**

try:

a=10

print a

raise NameError("Hello")

except NameError as e:

print "Mot Exception xuat hien"

print e

Trong ví dụ trên, e được sử dụng như là một biến tham chiếu mà lưu trữ giá trị của exception.

Ghi chú: để bắt một exception, một mệnh đề except phải tham chiếu tới cùng exception đã được ném. Ví dụ, để bắt exception trên, chúng ta phải viết mệnh đề except như sau:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

...

except "Invalid level!":

Xu ly ngoai le o day...

else:

Phan con lai cua code...

## **Custom Exception trong Python**

Python cho phép bạn tạo riêng cho mình các exception bằng cách kế thừa các lớp từ các Standard Exception.

Ví dụ dưới đây liên quan tới RuntimeError. Ở đây, một lớp đã tạo là lớp con của của RuntimeError. Trong khối try, exception được định nghĩa bởi người dùng được tạo và được bắt trong khối except. Biến e được sử dụng để tạo một instance của lớp Networkerror.

class Networkerror(RuntimeError):

def \_\_init\_\_(self, arg):

self.args = arg

Sau khi định nghĩa lớp trên, bạn có thể tạo exception như sau:

try:

raise Networkerror("Bad hostname")

except Networkerror,e:

print e.args