

Xử lý ngoại lệ và File

1

Nội dung

1. Ngoại lệ và xử lý ngoại lệ
2. File

2

2

Ngoại lệ

- Ngoại lệ = không hẳn là lỗi
- Chia lỗi thành 3 nhóm:
 1. **Lỗi khi viết chương trình**: chương trình không chạy được nếu là thông dịch (hoặc không dịch được, nếu là biên dịch)
 2. **Lỗi khi chương trình chạy**: phải thực hiện lại
 3. **Ngoại lệ**: vẫn là lỗi, xảy ra khi có một bất thường và khiến một chức năng không thể thực hiện được.

3

3

- Python chia lỗi thành 2 loại:
 - **Syntax error**: viết sai cú pháp, khiến chương trình thông dịch không dịch được
 - **Exception**: xảy ra bất thường không như thiết kế
=> xử lý exception => chương trình ổn định và hoạt động tốt trong mọi tình huống.

4

4

Xử lý ngoại lệ

```
except (NameError, TypeError):    # xử lý 2 loại lỗi
    print("Name or Type error")
except IOError as e:              # lấy đối tượng lỗi, đặt tên e
    print(e)
    raise
except ValueError:                # trả lại lỗi này
    print("Value error")          # xử lý lỗi Value
                                  # xử lý tất cả các lỗi còn lại
except:
    print("An error occurred")
    raise NameError("Không xử lý") # tạo ra một lỗi "Không xử lý"
else:
    print("OK")                   # thực hiện nếu không có lỗi nào
```

5

5

Xử lý ngoại lệ

- **Khối “try”**: đoạn mã có khả năng gây lỗi, khi lỗi xảy ra, khối này sẽ bị dừng ở dòng gây lỗi
- **Khối “except”**: đoạn mã xử lý lỗi, chỉ thực hiện nếu có lỗi xảy ra, nếu không sẽ bị bỏ qua.
- **Khối “else”**: có thể xuất hiện ngay sau khối except cuối cùng, đoạn mã sẽ được thực hiện nếu không có except nào được thực hiện (đoạn try không có lỗi)
- **Khối “finally”**: còn được gọi là khối clean-up, luôn được thực hiện dù có xảy ra lỗi hay không

6

6

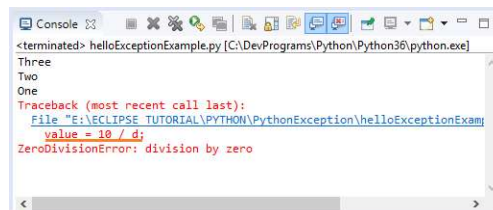
Xử lý ngoại lệ

- Khối **try** chỉ có 1 khối duy nhất, phải viết đầu tiên
- Khối **finally** có thể có hay không, nếu có thì khối này phải viết cuối cùng
- Khối **except** có thể không viết, có một khối, hoặc nhiều khối **except** (để xử lý nhiều tình huống lỗi khác nhau)
- Một khối **except** có thể xử lý một loại lỗi, nhiều loại lỗi hoặc tất cả các loại lỗi
- Nếu không xử lý triệt để lỗi có thể “ném” trả lại lỗi này bằng lệnh **“raise”** => Có thể phát sinh một ngoại lệ bằng lệnh **“raise<lỗi>”**

7

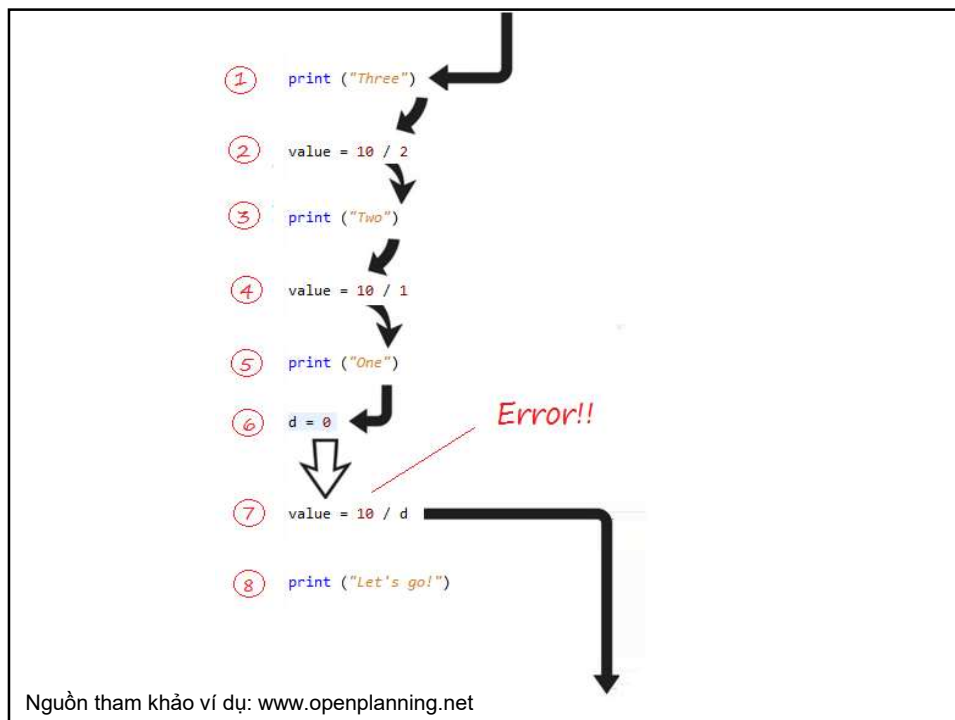
Ví dụ

```
1. print("Three")
2.
3. # Phép chia này không vấn đề.
4. value = 10 / 2
5.
6. print("Two")
7.
8. # Phép chia này không vấn đề.
9. value = 10 / 1
10.
11. print("One")
12.
13. d = 0
14.
15. # Phép chia này có vấn đề, cho cho 0
16. # Một lỗi được phát ra tại đây.
17. value = 10 / d
18.
19. # Dòng mã dưới đây sẽ không được thực thi.
20. print("Let's go!")
```



Nguồn tham khảo ví dụ: www.openplanning.net

8



9

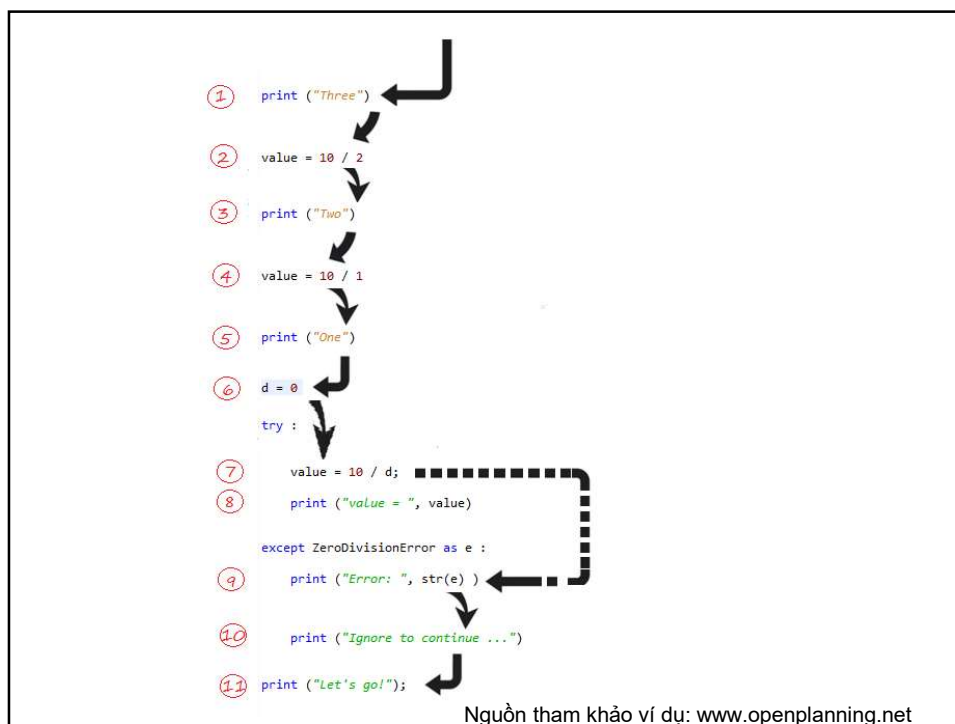
```

1. print ("Three")
2. value = 10 / 2
3. print ("Two")
4. value = 10 / 1
5. print ("One")
6. d = 0
7. try :
8.     # Phép chia này có vấn đề, chia cho 0.
9.     # Một lỗi được phát ra tại đây (ZeroDivisionError).
10.    value = 10 / d
11.
12.    print ("value = ", value)
13.
14. except ZeroDivisionError as e :
15.
16.    print ("Error: ", str(e) )
17.    print ("Ignore to continue ...")
18. print ("Let's go!")

```

Source: www.openplanning.net

10



11

Ví dụ

Nhập hai số a, b từ bàn phím. Tính thương a/b.

Hướng dẫn:

1. def tinhThuong():
2. a = int(input("Vui long nhap a: "))
3. b = int(input("Vui long nhap b: "))
4. thuong = a / b
5. print("==> Ket qua:", a, "/", b, "=", thuong)
6. **try:**
7. tinhThuong();
8. **except ZeroDivisionError:**
9. print("-----\nLoi! Vui long nhap lai! (Do khong the chia cho 0)")
10. tinhThuong();

12

Xử lý ngoại lệ	
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 trở tới bất kỳ đối tượng nào
StandardError	Lớp cơ sở của tất cả exception có sẵn ngoại trừ Stop Iteration 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

13

13

Xử lý ngoại lệ	
Exception	Miêu tả
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

14

14

Xử lý ngoại lệ

Exception	Miêu tả
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

15

15

Xử lý ngoại lệ

Exception	Miêu tả
SystemExit	Được tạo khi trình thông dịch thoát ra bởi sử dụng hàm <code>sys.exit()</code> . 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

16

16

Ví dụ

Nhập 3 số a,b,c từ bàn phím. Giải phương trình bậc 2: $ax^2+bx+c=0$

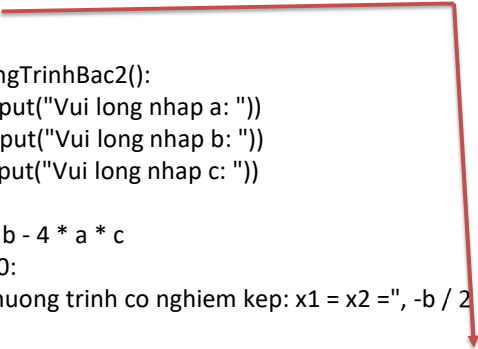
Hướng dẫn:

```
import math

def giaiPhuongTrinhBac2():
    a = float(input("Vui long nhap a: "))
    b = float(input("Vui long nhap b: "))
    c = float(input("Vui long nhap c: "))

    delta = b * b - 4 * a * c
    if delta == 0:
        print("Phuong trinh co nghiem kep: x1 = x2 =", -b / 2 * a)
    else:
        print("Phuong trinh co 2 nghiem: x1 =", (-b + math.sqrt(delta)) / (2 * a), ", ",
              x2 =", (-b + math.sqrt(delta)) / (2 * a))

try:
    giaiPhuongTrinhBac2();
except:
    print("Phuong trinh da cho vo nghiem")
```



17

File

- Các thao tác với tập tin:
 1. Mở file
 2. Đọc/ghi file
 3. Đóng file
- Có thể phát sinh ngoại lệ IOError
- Thay vì đặt toàn bộ các bước này trong khối try, ta có thể mở file :

```
with open("myfile.txt") as f :  
    <khối xử lý file>
```
- Ưu điểm: file luôn được đóng, dù có lỗi hay không

18

18

File

- Mở file:

f = open (filename, mode)

- Các chế độ **mode** mở file haysử dụng:

‘r’: chỉ đọc

‘w’: chỉ ghi

‘a’: ghi vào cuối file

‘r+’: cả đọc và ghi

‘t’: mở file văn bản (mặc định)

‘b’: mở file nhị phân

- Đóng file:

f.close()

19

19

File

- **read(x)**: đọc x byte tiếp theo, nếu không viết x thì sẽ đọc đến cuối file
- **readline(x)**: đọc 1 dòng từ file, tối đa là x byte, nếu không viết x thì đọc tới khi nào gặp ký tự hết dòng hoặc hết file là dừng.
- **readlines(x)**: sử dụng readline đọc các dòng cho đến hết file và trả về một danh sách các string, nếu viết x thì sẽ đọc tối đa là x byte.

20

20

File

- Nếu muốn duyệt hết file từ đầu đến cuối theo từng dòng thì sử dụng đoạn mã sau là hiệu quả nhất

```
with open('work file ') as f:  
    for line in f:  
        print(line end='')
```

- Ghi dữ liệu ra file:
 - **write(x)**: ghi x ra file, trả về số byte ghi được
 - **writelines(x)**: ghi toàn bộ nội dung x theo từng dòng, ở đây x là list of string

21

21

File

- **flush()**: đẩy các dữ liệu trên bộ nhớ tạm ra file
- **tell()**: trả về vị trí hiện tại của con trỏ file
- **seek(n)**: dịch con trỏ file đến vị trí byte thứ n
 - ❖ Hàm có thêm tham số thứ 2, cho phép diễn giải cách hiểu của tham số n
 - ❖ Nếu không viết, hoặc =0: vị trí n tính từ đầu file
 - ❖ ▪ =1: vị trí n tính từ vị trí hiện tại
 - ❖ ▪ =2: vị trí n tính từ cuối file
- **truncate(n)**: cắt file ở vị trí byte thứ n, hoặc vị trí hiện tại (nếu không viết giá trị n)

22

22

Ví dụ

- Bạn tạo file a.txt trong notepad và lưu vào thư mục hiện hành của bạn. Viết code để ghi “I Love You” vào file a.txt

- **Hướng dẫn:**

```
with open("D:/Documents/a.txt", mode = 'r+',  
          encoding = 'utf-8') as f:  
  
    f.write("I Love You");  
  
f.close();
```

23

23

Ví dụ

- Bạn tạo file a.txt trong notepad và lưu vào thư mục hiện hành của bạn. Viết code để ghi bài thơ sau vào file a.txt

1. Công cha như núi Thái Sơn,
2. Nghĩa Mẹ như nước.....
3. Một lòng thờ Mẹ....
4. Cho tròn chữ hiếu...

- a. Viết code để xuất bài thơ trên ra màn hình,
- b. Viết code để in dòng thứ 2 ra màn hình.
- c. Viết code để in dòng thứ 1 và thứ 4 ra màn hình

24

24

Hướng dẫn:

```
f = open(r"D:\nhap\a.txt", "w", encoding='utf-8')
```

```
str = 'Công cha như núi Thái Sơn\nNghĩa mẹ như nước trong nguồn chảy ra\nMột lòng  
thờ mẹ, kính cha\ncho tròn chữ hiếu mới là đạo con.'
```

```
f.write(str)
```

```
f.close()
```

```
f = open(r"D:\nhap\a.txt", "r", encoding='utf-8')
```

```
noidung = f.readlines()
```

```
print(noidung[1])
```

```
print(noidung[3])
```