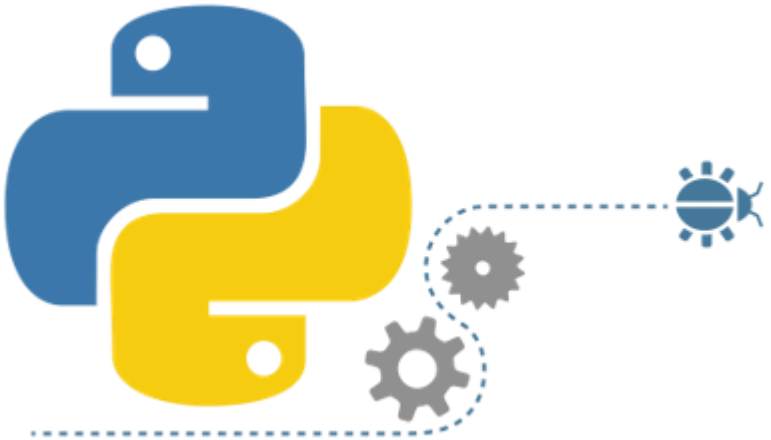
LẬP TRÌNH TRÍ TUỆ NHÂN TẠO

THS. VƯƠNG XUÂN CHÍ

VXCHI@NTT.EDU.VN

0903 270 567

CHƯƠNG 5



EXCEPTION & HANDLING

1. Ngoại lệ và xử lý ngoại lệ
2. Làm việc với tập tin

Ôn lại

- Tập hợp (set) và tập tĩnh (frozenset) là các kiểu dữ liệu liệt kê, các phần tử nằm trong nó không được phép trùng nhau, frozenset không thể bị thay đổi
  (Cả hai hỗ trợ những phép toán trên tập hợp như trong toán học)
- Từ điển (dictionary): nhóm các bộ đôi (key, value), từ điển là một dạng tập hợp theo các key
- Module và Package: là khái niệm của python tương ứng với file và thư mục vật lý, cho phép phân cấp và kiểm soát hiệu quả mã nguồn python

Ngoại lệ & xử lý ngoại lệ

Exception (Ngoại lệ) là gì?

- Ngoại lệ = lỗi, đúng, nhưng không hẳn
- Thường người ta chia lỗi thành 3 nhóm

  1. Lỗi khi viết chương trình: hệ quả là chương trình không chạy được nếu là thông dịch (hoặc không dịch được, nếu là biên dịch)

  2. Lỗi khi chương trình chạy: hệ quả là phải thực hiện lại
  - VD như nhập liệu không đúng, thì phải nhập lại

  3. Ngoại lệ: vẫn là lỗi, xảy ra khi có một bất thường và khiến một chức năng không thể thực hiện được
  - VD như đang ghi dữ liệu ra một file, nhưng file đó lại bị một tiến trình khác xóa mất

Ngoại lệ là gì?

- Ranh giới giữa ngoại lệ và lỗi khá mong manh, thậm chí khó phân biệt trong nhiều tình huống
- Cách chia lỗi thành 3 nhóm có khuynh hướng cho rằng môi trường thực thi của chương trình là thân thiện và hoàn hảo
- Python có xu hướng 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 sẽ khiến chương trình ổn định và hoạt động tốt trong mọi tình huống

Ngoại lệ là gì?

- Ví dụ về syntax error:

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

- Ví dụ về exception:

```
>>> 10 * (1/0)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- Có vẻ như syntax error cũng chỉ là một exception!

Exception Handling -*"Xử lý" ngoại lệ*

```
while True:
    try:
        x = int(input("Nhập số X: "))
        break
except ValueError:
        print("Lỗi, hãy nhập lại.")
print("X =", x)
```

Vòng lặp nhập X cho đến khi người dùng nhập vào đúng giá trị số

Khối nhập X (có thể nhập lỗi)

Xử lý khi lỗi xảy ra

Cú pháp try-except-else-finally

Có thể gồm tới 4 khối:

- 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

Cú pháp try-except-finally

Chú ý:

- 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>

Cú pháp try-except-finally

```
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 # trả lại lỗi này
except ValueError: # xử lý lỗi Value
   print("Value error")
except: # xử lý tất cả các lỗi còn lại
   print("An error occurred")
   raise NameError("Ko bit") # tạo ra một lỗi "Ko bit"
else: # thực hiện nếu không có lỗi nào
   print("OK")
```

Một số loại exception thường gặp

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
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

Một số loại exception thường gặp

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

Một số loại exception thường gặp

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

Một số loại exception thường gặp

Exception	Miêu tả
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

Ví dụ : try - except IOError

Sau mở một file để ghi trong khi không có quyền ghi, do đó nó sẽ tạo một **exception**:

```
try:
    f = open("testfile", "r")
    f.write("Kiểm tra về xử lý ngoại lệ")
except IOError:
    print "Lỗi: Không tìm thấy file"
else:
    print "Bạn đã ghi file thành công"
```

Ví dụ try -except -finally

Nếu có bất kỳ code nào 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.

```
try:
    f = open("testfile", "w")
    try:
        f.write("Kiểm tra về xử lý ngoại lệ")
    finally:
        print ("Chuẩn bị đóng file"
        f.close()
except IOError:
    print ("Lỗi: Không tim thấy file")
```

Làm việc với tập tin

Làm việc với tập tin

- Làm việc với tập tin trong python gồm 3 bước:

  1. Mở file

  2. Đọc/ghi file

  3. Đóng file

- Các bước này đều có thể phát sinh ngoại lệ IOError
- Thay vì đặt toàn bộ các bước này trong khối try, ta có thể mở file với **with** như dưới đây:

```
with open("myfile.txt") as f:
<khối xử lý file>
```

- **Ưu điểm**: file luôn được đóng, dù có lỗi hay không

Mở file và đóng file

- Mở file:

```
f = open(filename, mode)
```

- Đóng file:

```
f.close()
```

(File không sử dụng nữa thì nên đóng)

Mở file và đóng file

Mode	Mô tả
r	Mở file chỉ để đọc
r+	Mở file để đọc và ghi
rb	Mở file trong chế độ đọc cho định dạng nhị phân, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của file
rb+	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file
w	Tạo một file mới để ghi, nếu file đã tồn tại thì sẽ bị ghi mới
w+	Tạo một file mới để đọc và ghi, nếu file tồn tại thì sẽ bị ghi mới
wb	Mở file trong chế độ ghi trong định dạng nhị phân. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới
wb+	Mở file để đọc và ghi trong định dạng nhị phân. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi
a	Mở file để ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để ghi mới
a+	Mở file để đọc và ghi thêm vào cuối file, nếu không tìm thấy file sẽ tạo mới một file để đọc và ghi mới
ab	Mở file trong chế độ append trong chế độ nhị phân. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi
ab+	Mở file trong để đọc và append trong định dạng nhị phân. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi

Đọc file, ghi file

Thuộc tính của File

Thuộc tính	Mô tả
file.closed	Trả về True nếu file đã đóng, ngược lại là False
file.mode	Trả về chế độ truy cập của file đang được mở
file.name	Trả về tên của file

 # Ví dụ:

```
file = open("data.txt", "wb")
print ("Tên của file là: ", file.name)
print ("File có đóng hoặc không?: ",file.closed)
print ("Chế độ mở file :", file.mode)
```

Đọc file

Có 3 hàm đọc file cơ bản:

- **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 kí tự hết dòng hoặc hết file
- **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

Đọc file

```
fileObject.read([size])
```

Ví dụ:

```
f = open("data.txt", "r",encoding = "utf-8")
a = f.read(15) # đọc 15 kí tự đầu tiên
print("File có nội dung là:\n",(a))

b = f.tell() # Kiểm tra vị trí hiện tại
print ("Vị trí hiện tại:",(b))

f.seek(0) # Đặt lại vị trí con trỏ tại vị trí đầu file
c = f.read()
print("Nội dung mới là:\n",(c))
```

Đọc file

Đọc chuỗi trong file

Ví dụ:

```
# Mở file để đọc dữ liệu
   f = open("data.txt", "r+")
# Đọc một chuỗi trong file
   str = f.read(20)
# In ra chuỗi được đọc
   print("Chuỗi được đọc là: ", str)
# Đóng file lại
   f.close()
```

Đọc file

```
fileObject.readline()
```

Ví dụ:

```
f = open("data.txt", "r")

line1 = f.readline()
line2 = f.readline()

print ("Dòng 1: ", line1)
print ("Dòng 2: ", line2)
```

Đọc file

```
fileObject.readlines()
```

Ví dụ:

```
f = open("data.txt","r",encoding ="utf-8")

content = f.readlines()
print ("Nội dung của file: ", content)
```

Đọc file

- Nếu muốn duyệt hết file từ đầu đến cuối theo từng dòng có thể sử dụng đoạn mã sau:

```
with open("data.txt") as f:
for lines in f:
  print(lines, end ="")
```

Ghi file

- **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

```
fileObject.write(string)
```

Ví dụ:

```
# Mở file để ghi
f = open("data.txt","w",encoding="utf-8")
# Ghi dữ liệu lên file
f.write("Khoa học dữ liệu");
# Đóng file
f.close()
print("Bạn đã ghi dữ liệu thành công !")
```

Một số hàm khác của file

- `flush()`: ép 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)

Một số hàm khác của file

Ví dụ:

```
f = open("data.txt", "wb")
print "Tên file là: ", f.name
# Ở đây không thực hiện điều gì, chỉ có thể thực hiện thực hiện read.
f.flush()
# Đóng file đã mở
f.close()
```

THANK YOU

THS. VƯƠNG XUÂN CHÍ
VXCHI@NTT.EDU.VN
0903 270 567