BÀI 11

Sets và Xử lý ngoại lệ trong Python

Tóm Tắt Nội Dung

Trong bài học này chúng ta sẽ đi tìm hiểu lần lượt các nội dung

1. Tổng quan về Sets
2. Các phương thức xử lí với sets
3. Xử lý lỗi trong Python

11.1 Tổng quan về Sets

Khái niệm về Sets

- Trong Python, Set là một kiểu dữ liệu hỗn hợp dùng để lưu trữ tập hợp các phần tử duy nhất (distinct) không theo thứ tự. Set là một dạng của Collection trong Python và giống với tập hợp (set) trong toán học

- Một số điểm quan trọng về Set trong Python:

  1. Set không chứa các phần tử trùng lặp

  2. Set không duy trì thứ tự của các phần tử. Khi bạn in Set, thứ tự của các phần tử có thể thay đổi mỗi lần bạn thực hiện in

  3. Có thể chứa các phần tử không thay đổi như số nguyên, số thực, chuỗi, tuple trong Set. Nhưng Set không thể chứa các phần tử có thay đổi như danh sách (list) hoặc set khác

  4. Set không hỗ trợ truy cập phần tử bằng chỉ số như danh sách, vì vậy bạn không thể sử dụng Set[index]. Bạn phải sử dụng vòng lặp hoặc các phương thức có sẵn để truy cập và xử lý phần tử trong Set.

11.1 Tổng quan về Sets

Cách khai báo một Sets

```
thisset = {"apple", 3.14 , (1, 2), True, False, 30}
print(thisset)
```

- Set được lưu trữ trong một cặp ngoặc nhọn
- Các phần tử cách nhau bởi dấu phẩy ,
- Các phần tử có thể chứa nhiều loại dữ liệu int, float, bool, str, tuple, None

11.2 Các phương thức xử lý với Sets

Truy cập đến các phần tử của Sets

Bạn chỉ có thể duyệt qua các phần tử của sets bằng vòng lặp

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Sử dụng từ khóa in để kiểm tra một phần tử có tồn tại trong set không

```
print("banana" in thisset)
```

11.2 Các phương thức xử lý với Sets

Thêm mới phần tử vào Sets

Với set bạn không thể thay đổi các phần tử sau khi khởi tạo, tuy nhiên có thể thêm mới

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Hoặc bạn có thể sử dụng phương thức **update()** để thêm mới

```
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

Bạn có thể thêm một tuples, lists, dictionaries vào set với update()

11.2 Các phương thức xử lý với Sets

Xóa phần tử ra khỏi Sets

Xóa với phương thức **remove()**

```
thisset = {"apple", "banana", "cherry"}
thisset.remove(" banana ")
print(thisset)
```

Hoặc bạn có thể sử dụng phương thức **discard()**

```
thisset.discard(" banana ")
print(thisset)
```

11.2 Các phương thức xử lý với Sets

Xóa phần tử ra khỏi Sets

Bạn cũng có thể sử dụng phương thức pop() để xóa, tuy nhiên nó xóa một phần tử **ngẫu nhiên** trong set

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

Xóa tất cả phần tử trong set, trả lại một set rỗng

```
thisset.clear()
```

Xóa hoàn toàn một set

```
del thisset
```

11.2 Các phương thức xử lý với Sets

Gộp 2 Sets

Bạn có thể sử dụng union() hay update() để gộp 2 set với nhau

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
set1.update(set2)
print(set3)
```

union() trả lại một set3 mới từ 2 set đã gộp

update() là gộp thêm set 2 vào set 1

11.2 Các phương thức xử lý với Sets

Một số phương thức khác

Phương thức	Ý nghĩa
intersection()	Trả về một Set mới chứa các phần tử chung của hai Set
difference()	Trả về một Set mới chứa các phần tử chỉ có trong Set đầu tiên nhưng không có trong Set thứ ha
issubset()	Kiểm tra xem Set có là tập con của một Set khác hay không
issuperset()	Kiểm tra xem Set có là tập cha của một Set khác hay không

Set là một công cụ hữu ích trong Python để làm việc với các tập hợp duy nhất của dữ liệu mà không cần quan tâm đến thứ tự của chúng

11.2 Các phương thức xử lý với Sets

Nguồn tham chiếu

- Tổng quan về Sets

  https://www.w3schools.com/python/python_sets.asp

- Các phương thức thao tác với Sets

  https://www.w3schools.com/python/python_sets_methods.asp

11.1 Tổng quan bắt lỗi và kiểm soát lỗi

Mục tiêu

- Nhận dạng lỗi khi viết và chạy chương trình Python
- Nắm được cách bắt lỗi và kiểm soát lỗi khi viết chương trình

Tại sao lại cần biết ?

- Để biết cách xử lý, gở lỗi khi có lỗi xảy ra
- Dự đoán được lỗi và code làm sao để ít lỗi nhất

Thành công của một lập trình viên là tạo ra một chương trình hoạt động tốt và không có lỗi !

11.2 Bắt lỗi và kiểm soát lỗi

Lỗi Syntax và lỗi logic nội tại

- Lỗi **syntax** là lỗi khi viết sai cú pháp lệnh của ngôn ngữ lập trình. Khi gặp lỗi này Python lập tức dừng chương trình và đưa ra thông báo lỗi **SyntaxError**

```
print(4 => 4)
```

Thông báo lỗi ở terminal

```
SyntaxError: invalid syntax
```

- **Cách sửa:**

  Gặp lỗi này trong VS Code đã báo đỏ phần lỗi sai chỗ nào, và khi chạy lên bạn cũng thấy nó chỉ ra chỗ gây lỗi

11.2 Bắt lỗi và kiểm soát lỗi

Lỗi Exception (ngoại lệ)

Lỗi này khó phát hiện hơn vì do không sai cú pháp mà lỗi phát sinh trong quá trình chạy chương trình. Các lỗi này xảy ra cho nội tại logic của chương trình. ➔ Lỗi Logic

- Lỗi khi nhập liệu
- Lỗi chia cho 0
- Lỗi chỉ số vượt quá giới hạn của dãy
- Lỗi thời gian chạy quá lâu
- Lỗi lời gọi hàm có tham số không đúng kiểu

Với các lỗi này Python có một cơ chế cho phép bạn bắt lỗi và tìm cách sửa lỗi hoặc tránh lỗi

11.2 Bắt lỗi và kiểm soát lỗi

Try ..except…

Ví dụ bạn có một đoạn code sau:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Bắt lỗi dòng lệnh print

Nếu lỗi thì in ra lỗi

- Bạn đưa tất cả dòng lệnh cần check lỗi vào khối lệnh của **try**
- Trường hợp khối lệnh cần bắt lỗi có lỗi thì lỗi ra được in ra ở khối lệnh của **except**

Tất các các lỗi đều được Python đặt tên và gắn mã. Mỗi mà lỗi Exception đều có tên và mô tả lỗi tương ứng.

11.2 Bắt lỗi và kiểm soát lỗi

Cấu trúc bắt lỗi Exception với lệnh try đơn giản

```
try:
    <Lệnh hoặc nhóm lệnh cần bắt lỗi>
except <Mã lỗi Exception 1>
    <Các lệnh xử lý lỗi 1>
....
except <Mã lỗi Exception n>
    <Các lệnh xử lý lỗi n>
```

Cách bắt lỗi này vẫn chưa đầy đủ. Chương tình có thể dừng đột ngột

Trình tự xử lý

- Thực hiện nhóm lệnh cần bắt lỗi
- Nếu không lỗi thì chuyển sang lệnh tiếp theo say try..except
- Nếu có lỗi và lỗi này khớp với các mã lỗi có trong except thì sẽ thực hiện các lệnh xử lý tương ứng với except đó
- Nếu có lỗi và lỗi này không khớp với các mã lỗi except thì dừng chương trình và báo lỗi

11.2 Bắt lỗi và kiểm soát lỗi

Ví dụ

```
import random
try:
  r = random.randint(1,3) # lấy giá trị ngẫu nhiên cho r
  if r == 1:
     print(int("Fred")) # chuyển thành số nguyên sai
  elif r == 3:
     [][2] = 5 # giá giá trị cho index không tồn tại
     print(3/0)
except ValueError:
     print("Không thể chuyển thành số nguyên")
except IndexError:
     print("Chỉ số không tồn tại")
except ZeroDivisionError:
     print("Không thể chia cho 0")
```

11.2 Bắt lỗi và kiểm soát lỗi

Cấu trúc bắt lỗi Exception với lệnh try đầy đủ

```
try:
    <Lệnh hoặc nhóm lệnh cần bắt lỗi>
except <Mã lỗi Exception 1>
   <Các lệnh xử lý lỗi 1>
....
except <Mã lỗi Exception n>
   <Các lệnh xử lý lỗi n>
except Exception:
   <Các lệnh xử lý lỗi khác>
```

Trình tự xử lý

- Thực hiện nhóm lệnh cần bắt lỗi
- Nếu không lỗi thì chuyển sang lệnh tiếp theo say try..except
- Nếu có lỗi và lỗi này khớp với các mã lỗi có trong except thì sẽ thực hiện các lệnh xử lý tương ứng với except đó
- Nếu có lỗi và lỗi này không khớp với các mã lỗi except nằm trên thì rơi vào Xử lý các lỗi khác nằm cuối cùng

11.2 Bắt lỗi và kiểm soát lỗi

Ví dụ trên được sửa lại như sau

```
import random
try:
  r = random.randint(1,3) # lấy giá trị ngẫu nhiên cho r
  if r == 1:
     print(int("Fred")) # chuyển thành số nguyên sai
  elif r == 3:
     [][2] = 5 # giá giá trị cho index không tồn tại
     print(3/0)
except ValueError:
     print("Không thể chuyển thành số nguyên")
except IndexError:
     print("Chỉ số không tồn tại")
except ZeroDivisionError:
     print("Không thể chia cho 0")
except Exception:
     print("Lỗi khác")
```

16.2 Bắt lỗi và kiểm soát lỗi

Nhóm lệnh else và finally

```
try:
    <Lệnh hoặc nhóm lệnh cần bắt lỗi>
except <Mã lỗi Exception 1>
    <Các lệnh xử lý lỗi 1>
except <Mã lỗi Exception n>
    <Các lệnh xử lý lỗi n>
except Exception:
    <Các lệnh xử lý lỗi khác>
else:
    <Các lệnh xử lý khi không có lỗi>
finally:
<Các lệnh luôn thực hiện>
```

Trình tự xử lý

- Thực hiện nhóm lệnh cần bắt lỗi
- Nếu không lỗi thì chuyển sang lệnh tiếp theo say try..except
- Nếu có lỗi và lỗi này khớp với các mã lỗi có trong except thì sẽ thực hiện các lệnh xử lý tương ứng với except đó
- Nếu có lỗi và lỗi này không khớp với các mã lỗi except nằm trên thì rơi vào Xử lý các lỗi khác nằm cuối cùng
- Nếu không có lỗi sẽ rơi vào khối lệnh else
- Nhóm lệnh trong finally luôn được thực hiện dù lỗi hay không lỗi

11.2 Bắt lỗi và kiểm soát lỗi

Ví dụ

```
def divide(x, y)
try:
   result = x / y;
except ZeroDivisionError:
     print(“Không thể chia cho 0”)
else:
     print(“Kết quả là: ”, result)
finally:
     print(“Đã thực hiện xong lệnh”)
```

11.3 Bắt lỗi và kiểm soát lỗi

Nguồn tham chiếu

- Xử lý ngoại lệ trong Python

  https://www.w3schools.com/python/python_try_except.asp

Tổng kết lại bài

1. Nắm được tổng quan về Sets
2. Nắm được các phương thức xử lí với sets
3. Xử lý lỗi trong Python