

Chương 9. Tập và ngoại lệ

NGUYỄN HOÀNG ANH

CNTT1. PTIT



Nội dung trong chương

Đọc từ tệp

Ghi vào tệp

Ngoại lệ

Lưu trữ dữ liệu

9.1. Đọc từ file

Một lượng dữ liệu đáng kinh ngạc có sẵn trong các tệp văn bản. Tệp văn bản có thể chứa dữ liệu thời tiết, dữ liệu giao thông, dữ liệu kinh tế xã hội, tác phẩm văn học, v.v.

Đọc từ một tệp đặc biệt hữu ích trong các ứng dụng phân tích dữ liệu, nhưng nó cũng có thể áp dụng cho bất kỳ trường hợp nào ta muốn phân tích hoặc sửa đổi thông tin được lưu trữ trong tệp. Ví dụ: chúng ta có thể viết một chương trình đọc nội dung của một tệp văn bản và viết lại tệp với định dạng cho phép trình duyệt hiển thị nó.

Khi ta muốn làm việc với thông tin trong tệp văn bản, bước đầu tiên là đọc tệp đó vào bộ nhớ. Ta có thể đọc toàn bộ nội dung của tệp hoặc bạn có thể làm việc trên tệp từng dòng một.

Đọc toàn bộ tệp

Bắt đầu với một tệp chứa số pi đến 30 chữ số thập phân, với 10 chữ số thập phân trên mỗi dòng:

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
print(contents)
```

Hàm `open()` cần một đối số: tên của tệp bạn muốn mở. Python tìm kiếm tệp này trong thư mục lưu trữ chương trình hiện đang được thực thi.

`open('pi_digits.txt')` trả về một đối tượng đại diện cho `pi_digits.txt`. Python chỉ định đối tượng này cho `file_object`, đối tượng mà chúng ta sẽ làm việc sau này trong chương trình.

Sử dụng phương thức `read()` trong dòng thứ hai của chương trình để đọc toàn bộ nội dung của tệp và lưu trữ nó dưới dạng một chuỗi dài trong nội dung.

pi_digits.txt

3.1415926535
8979323846
2643383279

Output:

3.1415926535
8979323846
2643383279

Đọc toàn bộ tệp

Sự khác biệt duy nhất giữa đầu ra này và tệp gốc là thêm dòng trống ở cuối đầu ra. Dòng trống xuất hiện vì `read()` trả về một chuỗi trống khi nó đến cuối tệp; chuỗi trống này hiển thị dưới dạng một dòng trống.

Nếu muốn loại bỏ dòng trống thừa, ta có thể sử dụng `rstrip()` trong lệnh gọi `print()`:

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
print(contents.rstrip())
```

```
3.1415926535
8979323846
2643383279
```

Đường dẫn tệp

Để Python mở các tệp từ một thư mục khác với thư mục nơi tệp chương trình được lưu trữ, cần cung cấp một đường dẫn tệp, đường dẫn này cho Python biết để tìm kiếm ở một vị trí cụ thể trên hệ thống.

Ví dụ về đường dẫn tương đối

```
with open('text_files/filename.txt') as file_object:
```

Đường dẫn tuyệt đối thường dài hơn đường dẫn tương đối, vì vậy sẽ hữu ích nếu chỉ định chúng cho một biến và sau đó chuyển biến đó vào `open()`:

```
file_path = '/home/ehmatthes/other_files/text_files/filename.txt'  
with open(file_path) as file_object:
```

Đọc từng dòng

Có thể sử dụng vòng lặp for trên đối tượng tệp để kiểm tra từng dòng từ tệp tại một thời điểm:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    for line in file_object:
        print(line)
```

Để kiểm tra nội dung của tệp, chúng ta làm việc qua từng dòng trong tệp bằng cách lặp qua đối tượng tệp.

Những dòng trống này xuất hiện vì một ký tự dòng mới ẩn ở cuối mỗi dòng trong tệp văn bản. Hàm print sẽ thêm dòng mới của riêng nó mỗi khi chúng ta gọi ➔ thêm `rstrip()` như ví dụ trước

3.1415926535	3.1415926535
	8979323846
8979323846	2643383279
2643383279	

Tạo danh sách các dòng từ một tệp

Lưu trữ các dòng của `pi_digits.txt` trong danh sách bên trong khối `with` và sau đó in các dòng bên ngoài khối `with`:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
for line in lines:
    print(line.rstrip())
```

Phương thức `readlines()` lấy từng dòng từ tệp và lưu trữ nó trong một danh sách. Danh sách này sau đó được gán cho biến *lines*, chúng ta có thể tiếp tục làm việc với danh sách đó sau khi khối *with* kết thúc.

Làm việc với nội dung của tệp

Tạo một chuỗi đơn chứa tất cả các chữ số trong tệp mà không có khoảng trắng trong đó

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
```

```
pi_string = ''
for line in lines:
    pi_string += line.rstrip()
```

```
print(pi_string)
print(len(pi_string))
```

```
3.1415926535 8979323846 2643383279
```

```
36
```

pi_digits.txt

3.1415926535

8979323846

2643383279

Tạo một biến, `pi_string`, để chứa các chữ số của số pi.

Sau đó, tạo một vòng lặp thêm từng dòng chữ số vào chuỗi `pi_string` và xóa ký tự dòng mới khỏi mỗi dòng.

Làm việc với nội dung của tệp

Biến `pi_string` chứa khoảng trắng ở bên trái của các chữ số trong mỗi dòng, nhưng chúng ta có thể loại bỏ điều đó bằng cách sử dụng `strip()` thay vì `rstrip()`:

```
for line in lines:
```

```
    pi_string += line.strip()
```

```
print(pi_string)
```

```
print(len(pi_string))
```

```
3.141592653589793238462643383279
```

```
32
```

Tệp lớn: Một triệu chữ số

chúng ta bắt đầu với một tệp văn bản chứa pi đến 1.000.000 chữ số thập phân thay vì chỉ 30, chúng ta có thể tạo một chuỗi đơn chứa tất cả các chữ số này.

```
filename = 'pi_million_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
pi_string = ''
for line in lines:
    pi_string += line.strip()
print(f"{pi_string[:52]}... ")
print(len(pi_string))

3.14159265358979323846264338327950288419716939937510...
1000002
```

Tìm ngày sinh trong số Pi

Sử dụng chương trình vừa viết để tìm hiểu xem sinh nhật của ai đó có xuất hiện ở bất kỳ đâu trong một triệu chữ số đầu tiên của số pi hay không.

```
for line in lines:
    pi_string += line.strip()

birthday = input("Enter your birthday, in the form mmddyy: ")
if birthday in pi_string:
    print("Your birthday appears in the first million digits of pi!")
else:
    print("Your birthday does not appear in the first million digits of pi.")
```

Enter your birthdate, in the form mmddyy: 090284

Your birthday appears in the first million digits of pi!

9.2. Ghi vào file

Một trong những cách đơn giản nhất để tiết kiệm dữ liệu là ghi nó vào một tệp. Khi ta ghi văn bản vào một tệp, đầu ra sẽ vẫn có sẵn sau khi bạn đóng cửa sổ chứa đầu ra của chương trình (output).

Ta có thể kiểm tra đầu ra sau một chương trình đã hoàn thành và ta cũng có thể chia sẻ kết quả đầu ra với những người khác. Ta cũng có thể viết các chương trình đọc lại văn bản vào bộ nhớ và làm việc lại với nó sau này.

Ghi vào tệp rỗng

Để ghi văn bản vào tệp, cần gọi `open()` với đối số thứ hai cho Python biết rằng ta muốn ghi vào tệp.

```
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

Lời gọi `open()` trong ví dụ này có hai đối số. Đối số đầu tiên vẫn là tên của tệp chúng ta muốn mở. Đối số thứ hai, 'w', cho Python biết rằng chúng ta muốn mở tệp ở chế độ ghi.

Chế độ:

r: chế độ đọc

w: chế độ ghi

a: chế độ nối thêm

r+: chế độ đọc và ghi

Nếu bạn bỏ qua đối số thứ hai, Python sẽ mở tệp ở chế độ chỉ đọc (read only) theo mặc định.

Mở tệp programming.txt → I love programming.

Ghi nhiều dòng

Thêm tổ hợp dòng mới vào chuỗi, ta sẽ khiến các chuỗi được ghi vào tệp đúng theo dòng mong muốn:

```
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I love creating new games.\n")
```

I love programming.

I love creating new games.

Ta cũng có thể sử dụng dấu cách, ký tự tab và dòng trống để định dạng đầu ra của mình, giống như ta đã làm với đầu ra dựa trên cửa sổ đầu cuối.

Thêm vào một tệp

Nếu muốn thêm nội dung vào tệp thay vì ghi đè lên nội dung hiện có, ta có thể mở tệp ở chế độ nối thêm. Khi mở tệp ở chế độ nối thêm, Python không xóa nội dung của tệp trước khi trả lại đối tượng tệp.

Bất kỳ dòng nào được ghi vào tệp sẽ được thêm vào cuối tệp. Nếu tệp chưa tồn tại, Python sẽ tạo một tệp trống.

```
filename = 'programming.txt'

with open(filename, 'a') as file_object:
    file_object.write("I also love finding meaning in large datasets.\n")
    file_object.write("I love creating apps that can run in a browser.\n")
```

I love programming.

I love creating new games.

I also love finding meaning in large datasets.

I love creating apps that can run in a browser.

9.3. Ngoại lệ

Python sử dụng các đối tượng đặc biệt được gọi là ngoại lệ để quản lý các lỗi phát sinh trong quá trình thực thi chương trình. Bất cứ khi nào một lỗi xảy ra khiến Python không chắc chắn phải làm gì tiếp theo, nó sẽ tạo ra một đối tượng ngoại lệ.

Nếu ta viết code xử lý ngoại lệ, chương trình sẽ tiếp tục chạy. Nếu bạn không xử lý ngoại lệ, chương trình sẽ tạm dừng và hiển thị theo dõi, bao gồm báo cáo về ngoại lệ đã được nêu ra.

Các trường hợp ngoại lệ được xử lý bằng các khối ***try-except***. Một khối ***try-except*** yêu cầu Python làm điều gì đó, nhưng nó cũng cho Python biết phải làm gì nếu một ngoại lệ được đưa ra.

Xử lý ngoại lệ ZeroDivisionError

xem một lỗi đơn giản khiến Python đưa ra một ngoại lệ

```
print(5/0)
```

```
Traceback (most recent call last):
```

```
  File "division_calculator.py", line 1, in <module>
```

```
    print(5/0)
```

```
ZeroDivisionError: division by zero
```

Lỗi được báo cáo tại ZeroDivisionError: division by zero trong traceback, ZeroDivisionError, là một đối tượng ngoại lệ. Python tạo ra loại đối tượng này để đối phó với tình huống mà nó không thể làm những gì chương trình yêu cầu.

Khi điều này xảy ra, Python sẽ dừng chương trình và cho chúng ta biết loại ngoại lệ đã được đưa ra.

Sử dụng khối try-except

Có thể viết một khối try-except để xử lý trường hợp ngoại lệ có thể được đưa ra.

Yêu cầu Python thử chạy một số code và cho nó biết phải làm gì nếu code dẫn đến một loại ngoại lệ cụ thể.

```
try:  
    print(5/0)  
except ZeroDivisionError:  
    print("You can't divide by zero!")
```

Nếu code trong khối try hoạt động, Python sẽ bỏ qua khối except. Nếu code trong khối try gây ra lỗi, Python sẽ tìm một khối except có lỗi khớp với khối đã được nêu ra và chạy code trong khối đó.

```
You can't divide by zero!
```

Sử dụng ngoại lệ để ngăn chặn sự cố

Nếu chương trình phản ứng với đầu vào không hợp lệ một cách thích hợp, nó có thể nhắc nhập đầu vào hợp lệ hơn thay vì gặp sự cố:

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")
while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    if second_number == 'q':
        break
    answer = int(first_number) / int(second_number)
    print(answer)
```

Give me two numbers, and I'll divide them.

Enter 'q' to quit.

First number: 5

Second number: 0

Traceback (most recent call last):

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

answer = int(first_number) / int(second_number)

ZeroDivisionError: division by zero

Khối Else

Lỗi xảy ra trên dòng thực hiện phép chia, vì vậy đó là nơi chúng ta sẽ đặt khối try-except.

```
--snip--
while True:
    --snip--
    if second_number == 'q':
        break

    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
        print("You can't divide by 0!")
    else:
        print(answer)
```

Give me two numbers, and I'll divide them.

Enter 'q' to quit.

First number: 5

Second number: 0

You can't divide by 0!

First number: 5

Second number: 2

2.5

First number: q

Xử lý ngoại lệ FileNotFoundError

Một vấn đề phổ biến khi làm việc với tệp là xử lý tệp bị thiếu. Tệp đang được tìm kiếm có thể ở một vị trí khác, tên tệp có thể bị sai chính tả hoặc tệp có thể hoàn toàn không tồn tại.

Chúng ta có thể xử lý tất cả các tình huống này một cách dễ dàng với khối *try-except*.

```
filename = 'alice.txt'
with open(filename, encoding='utf-8') as f:
    contents = f.read()
```

Python không thể đọc từ một tệp bị thiếu, vì vậy nó tạo ra một ngoại lệ:

```
Traceback (most recent call last):
```

```
File "alice.py", line 3, in <module>
```

```
    with open(filename, encoding='utf-8') as f:
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'alice.txt'
```

Xử lý ngoại lệ FileNotFoundError

Dòng cuối cùng của traceback báo cáo lỗi FileNotFoundError: đây là trường hợp ngoại lệ mà Python tạo ra khi nó không thể tìm thấy tệp mà nó đang cố gắng mở. Trong ví dụ này, hàm `open()` tạo ra lỗi, vì vậy để xử lý nó, khối *try* sẽ bắt đầu bằng dòng chứa *open()*:

```
filename = 'alice.txt'

try:
    with open(filename, encoding='utf-8') as f:
        contents = f.read()
except FileNotFoundError:
    print(f"Sorry, the file {filename} does not exist.")
```

Trong ví dụ này, code trong khối *try* tạo ra một FileNotFoundError, vì vậy Python sẽ tìm một khối *except* phù hợp với lỗi đó.

```
Sorry, the file alice.txt does not exist.
```

Phân tích dữ liệu văn bản

Sử dụng phương thức chuỗi `split()`, có thể tạo danh sách các từ từ một chuỗi.

```
>>> title = "Alice in Wonderland"  
>>> title.split()  
['Alice', 'in', 'Wonderland']
```

Phương thức `split()` tách một chuỗi thành các phần ở bất kỳ nơi nào nó tìm thấy khoảng trống và lưu trữ tất cả các phần của chuỗi trong một danh sách.

Phân tích dữ liệu văn bản

Để đếm số từ trong Alice in Wonderland, chúng ta sẽ sử dụng `split()` trên toàn bộ văn bản

```
filename = 'alice.txt'

try:
    with open(filename, encoding='utf-8') as f:
        contents = f.read()
except FileNotFoundError:
    print(f"Sorry, the file {filename} does not exist.")
else:
    # Count the approximate number of words in the file.
    words = contents.split()
    num_words = len(words)
    print(f"The file {filename} has about {num_words} words.")
```

The file `alice.txt` has about 29465 words.

Làm việc với nhiều tệp

Xây dựng hàm `count_words()`

Xây dựng danh sách các tệp cần đếm từ

Tạo vòng for để duyệt qua tất cả tệp trong danh sách

```
filenames = ['alice.txt', 'siddhartha.txt',  
'moby_dick.txt', 'little_women.txt']
```

```
for filename in filenames:
```

```
    count_words(filename)
```

```
The file alice.txt has about 29465 words.
```

```
Sorry, the file siddhartha.txt does not exist.
```

```
The file moby_dick.txt has about 215830 words.
```

```
The file little_women.txt has about 189079  
words.
```

```
def count_words(filename):
```

```
    """Count the approximate number of words in a file."""
```

```
    try:
```

```
        with open(filename, encoding='utf-8') as f:
```

```
            contents = f.read()
```

```
        except FileNotFoundError:
```

```
            print(f"Sorry, the file {filename} does not exist.")
```

```
    else:
```

```
        words = contents.split()
```

```
        num_words = len(words)
```

```
        print(f"The file {filename} has about {num_words} words.")
```

Che giấu lỗi

```
def count_words(filename):  
    """Count the approximate number of words in a file."""  
    try:  
        --snip--  
    except FileNotFoundError:  
        pass  
    else:  
        --snip--  
  
filenames = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt', 'little_women.txt']  
for filename in filenames:  
    count_words(filename)
```

The file `alice.txt` has about 29465 words.

The file `moby_dick.txt` has about 215830 words.

The file `little_women.txt` has about 189079 words.

Quyết định lỗi nào cần báo cáo

Cung cấp cho người dùng thông tin mà họ không tìm kiếm có thể làm giảm khả năng sử dụng của chương trình của bạn.

Các cấu trúc xử lý lỗi của Python cung cấp cho ta khả năng kiểm soát tốt hơn đối với số lượng chia sẻ với người dùng khi có sự cố; tùy thuộc vào bạn để quyết định lượng thông tin cần chia sẻ.

Mỗi khi chương trình phụ thuộc vào thứ gì đó bên ngoài, chẳng hạn như đầu vào của người dùng, sự tồn tại của tệp hoặc tính khả dụng của kết nối mạng, thì sẽ có khả năng xảy ra ngoại lệ.

Một chút kinh nghiệm sẽ giúp ta biết nơi đưa các khối xử lý ngoại lệ vào chương trình và mức độ báo cáo cho người dùng về các lỗi phát sinh.

9.4. Lưu trữ dữ liệu

Mô-đun json cho phép kết xuất các cấu trúc dữ liệu Python đơn giản vào một tệp và tải dữ liệu từ tệp đó vào lần chạy chương trình tiếp theo. Ta cũng có thể sử dụng json để chia sẻ dữ liệu giữa các chương trình Python khác nhau.

Tốt hơn nữa, định dạng dữ liệu JSON không dành riêng cho Python, vì vậy ta có thể chia sẻ dữ liệu bạn lưu trữ ở định dạng JSON với những người làm việc bằng nhiều ngôn ngữ lập trình khác. Đó là một định dạng hữu ích và di động, đồng thời dễ học.

Sử dụng json.dump() và json.load()

Viết một chương trình ngắn lưu trữ một tập hợp các số và một chương trình khác đọc những số này trở lại bộ nhớ.

Chương trình đầu tiên sẽ sử dụng json.dump() để lưu trữ tập hợp các số và chương trình thứ hai sẽ sử dụng json.load().

```
import json
```

```
numbers = [2, 3, 5, 7, 11, 13]
```

```
filename = 'numbers.json'
```

```
with open(filename, 'w') as f:
```

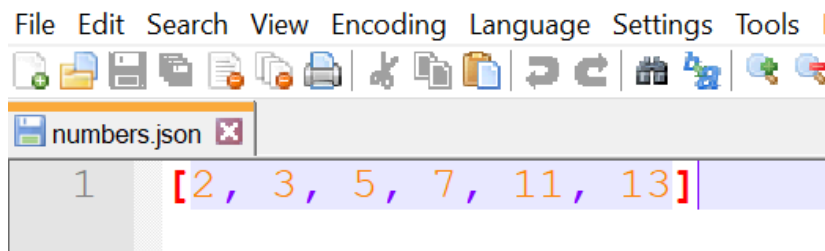
```
    json.dump(numbers, f)
```

1. nhập mô-đun json

2. tạo một danh sách các số

3. mở tệp ở chế độ ghi, cho phép json ghi dữ liệu

4. sử dụng hàm json.dump() để lưu trữ danh sách



Sử dụng json.dump() và json.load()

Viết một chương trình sử dụng json.load() để đọc lại danh sách trong bộ nhớ:

```
import json
filename = 'numbers.json'
with open(filename) as f:
    numbers = json.load(f)
print(numbers)
```

1. mở nó ở chế độ đọc

2. sử dụng hàm json.load() để tải thông tin được lưu trữ trong number.json và chúng ta gán nó vào biến numbers

3. in danh sách các số đã khôi phục

```
[2, 3, 5, 7, 11, 13]
```

Lưu và đọc dữ liệu do người dùng tạo

Ví dụ: nhắc người dùng tên của họ trong lần đầu tiên họ chạy một chương trình và sau đó ghi nhớ tên của họ khi họ chạy lại chương trình.

```
import json
username = input("What is your name? ")
filename = 'username.json'
with open(filename, 'w') as f:
    json.dump(username, f)
    print(f"We'll remember you when you come back, {username}!")
```

What is your name? **Eric**

We'll remember you when you come back, Eric!

Lưu và đọc dữ liệu do người dùng tạo

Viết một chương trình mới chào mừng người dùng có tên đã được lưu trữ:

```
import json
filename = 'username.json'
with open(filename) as f:
    username = json.load(f)
    print(f"Welcome back, {username}!")
```

Chúng ta sử dụng `json.load()` để đọc thông tin được lưu trữ trong `username.json` và gán nó cho biến `username`, sau đó in ra:

```
Welcome back, Eric!
```

Dùng try-except để gộp 2 phần

```
import json

# Load the username, if it has been stored previously.
# Otherwise, prompt for the username and store it.
filename = 'username.json'
try:
    with open(filename) as f:
        username = json.load(f)
except FileNotFoundError:
    username = input("What is your name? ")
    with open(filename, 'w') as f:
        json.dump(username, f)
        print(f"We'll remember you when you come back, {username}!")
else:
    print(f>Welcome back, {username}!")
```

Output:

What is your name? **Eric**

We'll remember you when you come back, Eric!

Welcome back, Eric!

Tái cấu trúc


Tái cấu trúc lại remember_me.py bằng cách chuyển phần lớn logic của nó vào một hoặc nhiều hàm.

```
def greet_user():
    """Greet the user by name."""
    filename = 'username.json'
    try:
        with open(filename) as f:
            username = json.load(f)
    except FileNotFoundError:
        username = input("What is your name? ")
        with open(filename, 'w') as f:
            json.dump(username, f)
        print(f"We'll remember you when you come back, {username}!")
    else:
        print(f>Welcome back, {username}!")
```

Tái cấu trúc (2)

Di chuyển code để truy xuất tên người dùng được lưu trữ sang một chức năng riêng biệt

```
def get_stored_username():  
    """Get stored username if available."""  
    filename = 'username.json'  
    try:  
        with open(filename) as f:  
            username = json.load(f)  
    except FileNotFoundError:  
        return None  
    else:  
        return username  
  
def greet_user():  
    """Greet the user by name."""  
    username = get_stored_username()  
    if username:  
        print(f>Welcome back, {username}!")  
    else:  
        username = input("What is your name? ")  
        filename = 'username.json'  
        with open(filename, 'w') as f:  
            json.dump(username, f)  
        print(f>We'll remember you when you come back,  
            {username}!")
```



Tái cấu trúc (3)

Thêm một khối code nữa ra khỏi `greet_user()`. Nếu tên người dùng không tồn tại, di chuyển code nhắc nhập tên người dùng mới sang một chức năng dành riêng cho mục đích đó:

```
def get_stored_username():
    """Get stored username if available."""
    --snip--

def get_new_username():
    """Prompt for a new username."""
    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f:
        json.dump(username, f)
    return username
```

```
def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        print(f>Welcome back, {username}!")
    else:
        username = get_new_username()
        print(f>We'll remember you when you come
back, {username}!")
greet_user()
```

Kết chương

Trong chương này, chúng ta đã học cách làm việc với tệp. Ta đã học cách đọc một toàn bộ tệp cùng một lúc và đọc qua nội dung của tệp từng dòng một. Ta đã học cách ghi vào tệp và nối văn bản vào cuối tệp. Ta cũng đọc về các ngoại lệ và cách xử lý các ngoại lệ mà ta có thể gặp trong các chương trình của mình. Cuối cùng, ta đã học được cách lưu trữ cấu trúc dữ liệu Python để có thể lưu thông tin mà người dùng cung cấp, ngăn họ phải bắt đầu lại mỗi lần chạy chương trình.

Bài tập