

Khối mã này cung cấp cách bạn có thể tải các hình ảnh có trong chương sách này.

```
from os.path import basename, exists
```

```
def download(url):  
    filename = basename(url)  
    if not exists(filename):  
        from urllib.request import urlretrieve  
  
        local, _ = urlretrieve(url, filename)  
        print("Downloaded " + str(local))  
    return filename
```

```
download('https://github.com/AllenDowney/ThinkPython/raw/v3/thinkpython.py');  
download('https://github.com/AllenDowney/ThinkPython/raw/v3/diagram.py');
```

```
import thinkpython
```

↔ Downloaded thinkpython.py
Downloaded diagram.py

✓ Chuỗi và biểu thức chính quy

Chuỗi không giống như số nguyên, số thực và boolean. Một chuỗi là một chuỗi giá trị, có nghĩa là nó chứa nhiều giá trị theo một thứ tự cụ thể. Trong chương này, chúng ta sẽ xem cách truy cập các giá trị tạo nên một chuỗi và sẽ sử dụng các hàm xử lý chuỗi.

Chúng ta cũng sẽ sử dụng các **biểu thức chính quy**, là một công cụ mạnh mẽ để tìm các mẫu trong một chuỗi và thực hiện các thao tác như tìm kiếm và thay thế.

Trong một bài tập, bạn sẽ có cơ hội áp dụng những công cụ này vào một trò chơi chữ gọi là Wordle.

✓ Một chuỗi là một chuỗi giá trị

Một chuỗi là một chuỗi các **ký tự**. Một ký tự có thể là một chữ cái (trong hầu hết các bảng chữ cái), một chữ số, một dấu câu, hoặc khoảng trắng.

Bạn có thể chọn một ký tự từ một chuỗi bằng cách sử dụng toán tử ngoặc vuông. Câu lệnh ví dụ này chọn ký tự số 1 từ `fruit` và gán nó cho `letter`:

```
fruit = 'banana'
letter = fruit[1]
```

Biểu thức trong dấu ngoặc vuông là một **chỉ số**, được gọi như vậy vì nó *chỉ ra* ký tự nào trong chuỗi cần chọn. Nhưng kết quả có thể không như bạn mong đợi.

```
letter
```

```
↔ 'a'
```

Ký tự có chỉ số 1 thực ra là ký tự thứ hai của chuỗi. Một chỉ số là một độ lệch tính từ đầu chuỗi, vì vậy độ lệch của ký tự đầu tiên là 0.

```
fruit[0]
```

```
↔ 'b'
```

Bạn có thể nghĩ rằng b là chữ cái thứ 0 của banana — phát âm là "zero-eth".

Chỉ số trong dấu ngoặc vuông có thể là một biến.

```
i = 1
fruit[i]
```

```
↔ 'a'
```

Hoặc một biểu thức chứa các biến và toán tử.

```
fruit[i+1]
```

```
↔ 'n'
```

Nhưng giá trị của chỉ số phải là một số nguyên — nếu không, bạn sẽ gặp lỗi `TypeError`.

```
%%expect TypeError
```

```
fruit[1.5]
```

```
↳ TypeError: string indices must be integers
```

Như chúng ta đã thấy trong Chương 1, chúng ta có thể sử dụng hàm tích hợp `len` để kiểm tra độ dài của một chuỗi.

```
n = len(fruit)
```

```
n
```

```
↳ 6
```

Để lấy chữ cái cuối cùng của một chuỗi, bạn có thể bị cám dỗ viết như sau:

```
%%expect IndexError
```

```
fruit[n]
```

```
↳ IndexError: string index out of range
```

Nhưng điều này gây ra lỗi `IndexError` vì không có chữ cái nào trong `banana` có chỉ số 6. Bởi vì chúng ta bắt đầu đếm từ 0, sáu chữ cái được đánh số từ 0 đến 5. Để lấy ký tự cuối cùng, bạn phải trừ 1 từ `n`:

```
fruit[n-1]
```

```
↳ 'a'
```

Nhưng có một cách dễ hơn. Để lấy chữ cái cuối cùng trong một chuỗi, bạn có thể sử dụng chỉ số âm, chỉ số này đếm lùi từ cuối chuỗi.

```
fruit[-1]
```

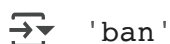
```
↳ 'a'
```

Chỉ số `-1` chọn chữ cái cuối cùng, `-2` chọn chữ cái thứ hai từ cuối, và cứ thế tiếp tục.

✓ Cắt chuỗi

Một đoạn của một chuỗi được gọi là **một mảnh**. Việc chọn một mảnh tương tự như việc chọn một ký tự.

```
fruit = 'banana'
fruit[0:3]
```

 `'ban'`

Toán tử `[n:m]` trả về phần của chuỗi từ ký tự thứ `n` đến ký tự thứ `m`, bao gồm ký tự đầu tiên nhưng không bao gồm ký tự thứ hai. Hành vi này có thể không trực quan, nhưng có thể giúp tưởng tượng rằng các chỉ số chỉ vào giữa các ký tự, như trong hình sau:

```
from diagram import make_binding, Element, Value

binding = make_binding("fruit", ' b a n a n a ')
elements = [Element(Value(i), None) for i in range(7)]
```

```

import matplotlib.pyplot as plt
from diagram import diagram, adjust
from matplotlib.transforms import Bbox

width, height, x, y = [1.35, 0.54, 0.23, 0.39]

ax = diagram(width, height)
bbox = binding.draw(ax, x, y)
bboxes = [bbox]

def draw_elts(x, y, elements):
    for elt in elements:
        bbox = elt.draw(ax, x, y, draw_value=False)
        bboxes.append(bbox)

        x1 = (bbox.xmin + bbox.xmax) / 2
        y1 = bbox.ymax + 0.02
        y2 = y1 + 0.14
        handle = plt.plot([x1, x1], [y1, y2], ':', lw=0.5, color='gray')
        x += 0.105

draw_elts(x + 0.48, y - 0.25, elements)
bbox = Bbox.union(bboxes)
# adjust(x, y, bbox)

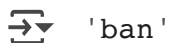
```



Ví dụ, mảnh `[3:6]` chọn các chữ cái `ana`, điều này có nghĩa là 6 hợp lệ như một phần của mảnh, nhưng không hợp lệ như một chỉ số.

Nếu bạn bỏ qua chỉ số đầu tiên, mảnh sẽ bắt đầu từ đầu chuỗi.

```
fruit[:3]
```



Nếu bạn bỏ qua chỉ số thứ hai, mảnh sẽ kéo dài đến cuối chuỗi:

```
fruit[3:]
```

```
➡ 'ana'
```

Nếu chỉ số đầu tiên lớn hơn hoặc bằng chỉ số thứ hai, kết quả sẽ là một **chuỗi rỗng**, được biểu diễn bằng hai dấu ngoặc kép:

```
fruit[3:3]
```

```
➡ ''
```

Một chuỗi rỗng không chứa ký tự nào và có độ dài là 0.

Tiếp tục với ví dụ này, bạn nghĩ rằng `fruit[:]` có nghĩa là gì? Hãy thử nó và xem.

```
fruit[:]
```

```
➡ 'banana'
```

✓ Chuỗi là bất biến

Bạn có thể bị cám dỗ sử dụng toán tử `[]` ở bên trái của một phép gán, với ý định thay đổi một ký tự trong chuỗi, như sau:

```
%%expect TypeError
```

```
greeting = 'Hello, world!'
greeting[0] = 'J'
```

```
➡ TypeError: 'str' object does not support item assignment
```

Kết quả là một lỗi `TypeError`. Trong thông báo lỗi, "đối tượng" là chuỗi và "mục" là ký tự mà chúng ta đã cố gắng gán.

Hiện tại, một **đối tượng** giống như một giá trị, nhưng chúng ta sẽ tinh chỉnh định nghĩa đó sau.

Lý do cho lỗi này là chuỗi là **bất biến**, nghĩa là bạn không thể thay đổi một chuỗi đã tồn tại. Điều tốt nhất bạn có thể làm là tạo ra một chuỗi mới là một biến thể của chuỗi gốc.

```
new_greeting = 'J' + greeting[1:]
new_greeting
```

```
↔ 'Jello, world!'
```

Ví dụ này nối một chữ cái đầu tiên mới vào một mảnh của `greeting`. Nó không ảnh hưởng gì đến chuỗi gốc.

```
greeting
```

```
↔ 'Hello, world!'
```

✓ So sánh chuỗi

Các toán tử quan hệ hoạt động trên các chuỗi. Để kiểm tra xem hai chuỗi có giống nhau hay không, chúng ta có thể sử dụng toán tử `==`.

```
word = 'banana'
```

```
if word == 'banana':
    print('All right, banana.')
```

Các phép toán quan hệ khác cũng hữu ích để sắp xếp các ký tự theo thứ tự bảng chữ cái:

```
def compare_word(word):
    if word < 'banana':
        print(word, 'comes before banana.')
    elif word > 'banana':
        print(word, 'comes after banana.')
    else:
        print('All right, banana.')
```

```
compare_word('apple')
```

Python không xử lý chữ hoa và chữ thường giống như con người. Tất cả các chữ cái viết hoa đều đứng trước tất cả các chữ cái viết thường, vì vậy:

```
compare_word('Pineapple')
```

Để giải quyết vấn đề này, chúng ta có thể chuyển đổi các chuỗi về định dạng chuẩn, chẳng hạn như tất cả đều là chữ thường, trước khi thực hiện phép so sánh. Hãy ghi nhớ điều đó nếu bạn phải tự vệ trước một người đàn ông cầm một quả dứa.

✓ Phương thức chuỗi

Chuỗi cung cấp các phương thức thực hiện nhiều thao tác hữu ích. Một phương thức giống như một hàm — nó nhận các đối số và trả về một giá trị — nhưng cú pháp lại khác nhau.

Chẳng hạn, phương thức `upper` nhận một chuỗi và trả về một chuỗi mới với tất cả các ký tự viết hoa.

Thay vì cú pháp hàm `upper(word)`, nó sử dụng cú pháp phương thức `word.upper()`.

```
word = 'banana'
new_word = word.upper()
new_word
```

Việc sử dụng toán tử chấm này chỉ định tên của phương thức, `upper`, và tên của chuỗi mà phương thức sẽ được áp dụng, `word`.

Cặp dấu ngoặc đơn rỗng cho biết rằng phương thức này không nhận đối số nào.

Một lần gọi phương thức được gọi là gọi (invocation); trong trường hợp này, chúng ta sẽ nói rằng chúng ta đang gọi `upper` trên `word`.

✓ Ghi tệp

Các toán tử và phương thức chuỗi rất hữu ích trong việc đọc và ghi các tệp văn bản. Một ví dụ, chúng ta sẽ làm việc với văn bản của *Dracula*, một tiểu thuyết của Bram Stoker có sẵn từ Project Gutenberg (<https://www.gutenberg.org/ebooks/345>).

```
import os

if not os.path.exists('pg345.txt'):
    !wget https://www.gutenberg.org/cache/epub/345/pg345.txt
```


Tôi đã tải xuống cuốn sách dưới dạng tệp văn bản thuần túy có tên là `pg345.txt` , mà chúng ta có thể mở để đọc như sau:

```
reader = open('pg345.txt')
```

Ngoài văn bản của cuốn sách, tệp này còn chứa một phần ở đầu với thông tin về cuốn sách và một phần ở cuối với thông tin về giấy phép. Trước khi xử lý văn bản, chúng ta có thể loại bỏ các tài liệu bổ sung này bằng cách tìm các dòng đặc biệt ở đầu và cuối bắt đầu bằng `***` .

Hàm sau đây nhận một dòng và kiểm tra xem nó có phải là một trong những dòng đặc biệt không. Nó sử dụng phương thức `startswith` , phương thức này kiểm tra xem một chuỗi có bắt đầu bằng một chuỗi ký tự nhất định hay không.

```
def is_special_line(line):  
    return line.startswith('*** ')
```

Chúng ta có thể sử dụng hàm này để lặp qua các dòng trong tệp và chỉ in ra các dòng đặc biệt.

```
for line in reader:  
    if is_special_line(line):  
        print(line.strip())
```

Bây giờ, hãy tạo một tệp mới, có tên là `pg345_cleaned.txt` , chỉ chứa nội dung của cuốn sách. Để lặp qua cuốn sách một lần nữa, chúng ta phải mở nó để đọc lại. Để ghi một tệp mới, chúng ta có thể mở nó để ghi.

```
reader = open('pg345.txt')  
writer = open('pg345_cleaned.txt', 'w')
```

`open` nhận một tham số tùy chọn để xác định "chế độ" -- trong ví dụ này, `w` cho biết rằng chúng ta đang mở tệp để ghi. Nếu tệp không tồn tại, nó sẽ được tạo; nếu nó đã tồn tại, nội dung sẽ bị thay thế.

Bước đầu tiên, chúng ta sẽ lặp qua tệp cho đến khi tìm thấy dòng đặc biệt đầu tiên.

```
for line in reader:
    if is_special_line(line):
        break
```

Câu lệnh `break` giúp chúng ta "thoát" khỏi vòng lặp -- tức là, nó khiến vòng lặp kết thúc ngay lập tức, trước khi chúng ta đến cuối tệp.

Khi vòng lặp thoát, `line` chứa dòng đặc biệt đã làm cho điều kiện trở thành đúng.

```
line
```

Bởi vì `reader` theo dõi vị trí hiện tại trong tệp, chúng ta có thể sử dụng một vòng lặp thứ hai để tiếp tục từ chỗ chúng ta đã dừng lại.

Vòng lặp sau sẽ đọc phần còn lại của tệp, từng dòng một. Khi nó tìm thấy dòng đặc biệt chỉ ra sự kết thúc của văn bản, nó sẽ thoát khỏi vòng lặp. Nếu không, nó sẽ ghi dòng đó vào tệp đầu ra.

```
for line in reader:
    if is_special_line(line):
        break
    writer.write(line)
```

Khi vòng lặp này kết thúc, `line` chứa dòng đặc biệt thứ hai.

```
line
```

Tại thời điểm này, `reader` và `writer` vẫn đang mở, điều này có nghĩa là chúng ta có thể tiếp tục đọc các dòng từ `reader` hoặc ghi các dòng vào `writer`. Để chỉ ra rằng chúng ta đã hoàn tất, chúng ta có thể đóng cả hai tệp bằng cách gọi phương thức `close`.

```
reader.close()
writer.close()
```

Để kiểm tra xem quá trình này có thành công hay không, chúng ta có thể đọc một vài dòng đầu tiên từ tệp mới mà chúng ta vừa tạo.

```
for line in open('pg345_cleaned.txt'):
    line = line.strip()
    if len(line) > 0:
        print(line)
    if line.endswith('Stoker'):
        break
```

Phương thức `endswith` kiểm tra xem một chuỗi có kết thúc bằng một chuỗi ký tự nhất định hay không.

✓ Tìm và thay thế

Trong bản dịch tiếng Iceland của *Dracula* từ năm 1901, tên của một trong các nhân vật đã được đổi từ "Jonathan" thành "Thomas". Để thực hiện thay đổi này trong phiên bản tiếng Anh, chúng ta có thể lặp qua cuốn sách, sử dụng phương thức `replace` để thay một tên bằng tên khác, và ghi kết quả vào một tệp mới.

Chúng ta sẽ bắt đầu bằng cách đếm số dòng trong phiên bản tệp đã được làm sạch.

```
total = 0
for line in open('pg345_cleaned.txt'):
    total += 1
```

`total`

Để xem liệu một dòng có chứa "Jonathan" hay không, chúng ta có thể sử dụng toán tử `in`, toán tử này kiểm tra xem chuỗi ký tự này có xuất hiện ở bất kỳ đâu trong dòng hay không.

```
total = 0
for line in open('pg345_cleaned.txt'):
    if 'Jonathan' in line:
        total += 1
```

`total`

Có 199 dòng chứa tên nhân vật này, nhưng đó không phải là tổng số lần xuất hiện, vì tên có thể xuất hiện nhiều lần trong một dòng.

Để có được tổng số, chúng ta có thể sử dụng phương thức `count`, phương thức này trả về số lần một chuỗi xuất hiện trong một đoạn văn bản.

```
total = 0
for line in open('pg345_cleaned.txt'):
    total += line.count('Jonathan')

total
```

Bây giờ chúng ta có thể thay thế Jonathan bằng Thomas như sau:

```
writer = open('pg345_replaced.txt', 'w')

for line in open('pg345_cleaned.txt'):
    line = line.replace('Jonathan', 'Thomas')
    writer.write(line)
```

Kết quả là một tệp mới có tên là `pg345_replaced.txt`, trong đó chứa phiên bản của *Dracula* mà Jonathan Harker được gọi là Thomas.

```
total = 0
for line in open('pg345_replaced.txt'):
    total += line.count('Thomas')

total
```

✓ Biểu thức chính quy

Nếu chúng ta biết chính xác chuỗi ký tự mà mình đang tìm kiếm, chúng ta có thể dùng toán tử `in` để tìm và phương thức `replace` để thay thế. Tuy nhiên, có một công cụ khác, gọi là **biểu thức chính quy**, có thể thực hiện những thao tác này – và còn nhiều hơn thế.

Để minh họa, tôi sẽ bắt đầu với một ví dụ đơn giản và chúng ta sẽ tiến dần lên. Giả sử, một lần nữa, rằng chúng ta muốn tìm tất cả các dòng có chứa một từ cụ thể. Đổi mới một chút, hãy tìm các tham chiếu đến nhân vật chính của cuốn sách, *Bá tước Dracula*.

Dưới đây là một dòng nhắc đến ông ấy.

```
text = "I am Dracula; and I bid you welcome, Mr. Harker, to my house."
```

Và đây là **mẫu** mà chúng ta sẽ sử dụng để tìm kiếm.

```
pattern = 'Dracula'
```

Một mô-đun có tên là `re` cung cấp các hàm liên quan đến biểu thức chính quy. Chúng ta có thể nhập nó như sau và sử dụng hàm `search` để kiểm tra xem mẫu có xuất hiện trong văn bản hay không.

```
import re

result = re.search(pattern, text)
result
```

Nếu mẫu xuất hiện trong văn bản, `search` sẽ trả về một đối tượng `Match` chứa kết quả của quá trình tìm kiếm. Bên cạnh các thông tin khác, nó có một biến tên là `string`, chứa văn bản đã được tìm kiếm.

```
result.string
```

Nó cũng cung cấp một phương thức gọi là `group` trả về phần văn bản khớp với mẫu.

```
result.group()
```

Nó cũng cung cấp một phương thức gọi là `span`, trả về chỉ số trong văn bản nơi mà mẫu bắt đầu và kết thúc.

```
result.span()
```

Nếu mẫu không xuất hiện trong văn bản, giá trị trả về từ `search` sẽ là `None`.

```
result = re.search('Count', text)
print(result)
```

Vì vậy, chúng ta có thể kiểm tra xem việc tìm kiếm có thành công hay không bằng cách kiểm tra xem kết quả có phải là `None` hay không.

```
result == None
```

Gộp tất cả lại với nhau, đây là một hàm lặp qua các dòng trong cuốn sách cho đến khi tìm thấy một dòng khớp với mẫu đã cho và trả về đối tượng `Match`.

```
def find_first(pattern):
    for line in open('pg345_cleaned.txt'):
        result = re.search(pattern, line)
        if result != None:
            return result
```

Chúng ta có thể sử dụng nó để tìm sự đề cập đầu tiên đến một nhân vật.

```
result = find_first('Harker')
result.string
```

Trong ví dụ này, chúng ta không cần phải sử dụng biểu thức chính quy — chúng ta có thể thực hiện điều tương tự một cách dễ dàng hơn với toán tử `in`. Nhưng biểu thức chính quy có thể làm những điều mà toán tử `in` không thể làm được.

Ví dụ, nếu mẫu bao gồm ký tự dấu gạch đứng, `'|'`, nó có thể khớp với chuỗi bên trái hoặc bên phải. Giả sử chúng ta muốn tìm sự đề cập đầu tiên đến Mina Murray trong cuốn sách, nhưng chúng ta không chắc liệu cô ấy được nhắc đến bằng tên hoặc họ. Chúng ta có thể sử dụng mẫu sau, khớp với một trong hai tên.

```
pattern = r'Mina|Murray'
result = find_first(pattern)
result.string
```

Chúng ta có thể sử dụng một mẫu như thế này để xem một nhân vật được đề cập bao nhiêu lần bằng một trong hai tên. Dưới đây là một hàm lặp qua cuốn sách và đếm số dòng khớp với mẫu đã cho.

```
def count_matches(pattern):
    count = 0
    for line in open('pg345_cleaned.txt'):
        result = re.search(pattern, line)
        if result != None:
            count += 1
    return count
```

Bây giờ hãy xem Mina được nhắc đến bao nhiêu lần.

```
count_matches('Mina|Murray')
```

Ký tự đặc biệt `'^'` khớp với đầu của một chuỗi, vì vậy chúng ta có thể tìm một dòng bắt đầu bằng một mẫu đã cho.

```
result = find_first('^Dracula')
result.string
```

Và ký tự đặc biệt `'$'` khớp với cuối một chuỗi, vì vậy chúng ta có thể tìm một dòng kết thúc bằng một mẫu đã cho (bỏ qua ký tự xuống dòng ở cuối).

```
result = find_first('Harker$')
result.string
```

✓ Thay thế chuỗi

Bram Stoker sinh ra ở Ireland, và khi *Dracula* được xuất bản vào năm 1897, ông đang sống ở Anh. Vì vậy, chúng ta sẽ mong đợi ông sử dụng cách viết tiếng Anh của những từ như "centre" và "colour". Để kiểm tra, chúng ta có thể sử dụng mẫu sau, khớp với cả "centre" hoặc cách viết tiếng Mỹ "center".

```
pattern = 'cent(er|re)'
```

Trong mẫu này, dấu ngoặc đơn bao quanh phần của mẫu mà dấu gạch đứng áp dụng. Vì vậy, mẫu này khớp với một chuỗi bắt đầu bằng 'cent' và kết thúc bằng hoặc 'er' hoặc 're'.

```
result = find_first(pattern)
result.string
```

Như mong đợi, ông đã sử dụng cách viết tiếng Anh.

Chúng ta cũng có thể kiểm tra xem ông có sử dụng cách viết tiếng Anh của từ "colour" hay không. Mẫu sau sử dụng ký tự đặc biệt '?', có nghĩa là ký tự trước đó là tùy chọn.

```
pattern = 'colou?r'
```

Mẫu này khớp với cả "colour" có chứa 'u' hoặc "color" không có nó.

```
result = find_first(pattern)
line = result.string
line
```

Một lần nữa, như mong đợi, ông đã sử dụng cách viết tiếng Anh.

Bây giờ giả sử chúng ta muốn sản xuất một phiên bản của cuốn sách với cách viết tiếng Mỹ. Chúng ta có thể sử dụng hàm `sub` trong mô-đun `re`, thực hiện **thay thế chuỗi**.


```
re.sub(pattern, 'color', line)
```

Tham số đầu tiên là mẫu mà chúng ta muốn tìm và thay thế, tham số thứ hai là nội dung mà chúng ta muốn thay thế, và tham số thứ ba là chuỗi mà chúng ta muốn tìm kiếm. Trong kết quả, bạn có thể thấy rằng "colour" đã được thay thế bằng "color".

Tôi đã sử dụng hàm này để tìm các dòng làm ví dụ.

```
def all_matches(pattern):  
    for line in open('pg345_cleaned.txt'):  
        result = re.search(pattern, line)  
        if result:  
            print(line.strip())
```

```
names = r'(?!\.\s)[A-Z][a-zA-Z]+'
```

```
all_matches(names)
```

✓ Gỡ lỗi

Khi bạn đọc và ghi tệp, việc gỡ lỗi có thể trở nên khó khăn. Nếu bạn đang làm việc trong một Jupyter Notebook, bạn có thể sử dụng lệnh **shell** để trợ giúp. Ví dụ, để hiển thị vài dòng đầu tiên của một tệp, bạn có thể sử dụng lệnh `!head`, như sau:

```
!head pg345_cleaned.txt
```

Dấu chấm than ở đầu, `!`, cho biết rằng đây là một lệnh **shell**, không phải là một phần của Python. Để hiển thị vài dòng cuối cùng, bạn có thể sử dụng `!tail`.

```
!tail pg345_cleaned.txt
```

Khi bạn làm việc với các tệp lớn, việc gỡ lỗi có thể trở nên khó khăn vì có thể có quá nhiều đầu ra để kiểm tra bằng tay. Một chiến lược gỡ lỗi tốt là bắt đầu với chỉ một phần của tệp, làm cho chương trình hoạt động, và sau đó chạy nó với toàn bộ tệp.

Để tạo một tệp nhỏ chứa phần của một tệp lớn hơn, chúng ta có thể sử dụng lại `!head` với toán tử chuyển hướng, `>`, chỉ ra rằng kết quả nên được ghi vào một tệp thay vì hiển thị.

```
!head pg345_cleaned.txt > pg345_cleaned_10_lines.txt
```

Mặc định, `!head` đọc 10 dòng đầu tiên, nhưng nó nhận một tham số tùy chọn cho biết số dòng cần đọc.

```
!head -100 pg345_cleaned.txt > pg345_cleaned_100_lines.txt
```

Lệnh shell này đọc 100 dòng đầu tiên từ `pg345_cleaned.txt` và ghi chúng vào một tệp có tên là `pg345_cleaned_100_lines.txt`.

Lưu ý: Các lệnh **shell** `!head` và `!tail` không có sẵn trên tất cả các hệ điều hành. Nếu chúng không hoạt động với bạn, chúng ta có thể viết các hàm tương tự trong Python. Xem bài tập đầu tiên ở cuối chương này để biết gợi ý.

Thuật ngữ

sequence - dãy số: Một bộ sưu tập các giá trị theo thứ tự, trong đó mỗi giá trị được xác định bởi chỉ mục số nguyên.

character - ký tự: Một phần tử của chuỗi, bao gồm chữ cái, số và ký hiệu.

index - chỉ mục: Một giá trị số nguyên được sử dụng để chọn một mục trong một dãy số, chẳng hạn như một ký tự trong chuỗi. Trong Python, chỉ mục bắt đầu từ 0.

slice - cắt chuỗi: Một phần của chuỗi được xác định bởi một phạm vi chỉ mục.

empty string - chuỗi rỗng: Một chuỗi không chứa ký tự nào và có độ dài bằng 0.

object - đối tượng: Một thứ mà một biến có thể tham chiếu đến. Một đối tượng có kiểu và giá trị.

immutable - bất biến: Nếu các phần tử của một đối tượng không thể thay đổi, đối tượng đó là bất biến.

invocation - gọi phương thức: Một biểu thức – hoặc một phần của biểu thức – gọi một phương thức.

regular expression - biểu thức chính quy: Một dãy ký tự định nghĩa một mẫu tìm kiếm.

pattern - mẫu: Một quy tắc xác định yêu cầu mà một chuỗi phải đáp ứng để phù hợp với một mẫu.

string substitution - thay thế chuỗi: Thay thế một chuỗi, hoặc một phần của chuỗi, bằng một chuỗi khác.

shell command - lệnh shell: Một câu lệnh trong ngôn ngữ shell, là ngôn ngữ dùng để tương tác với hệ điều hành.

✓ Bài tập

```
# Ô này yêu cầu Jupyter cung cấp thông tin gỡ lỗi chi tiết
# khi xảy ra lỗi thời gian chạy. Chạy nó trước khi làm các bài tập.
```

```
%xmode Verbose
```

```
download('https://raw.githubusercontent.com/AllenDowney/ThinkPython/v3/words.txt')
```

✓ Hỏi một trợ lý ảo

Trong chương này, chúng ta chỉ mới lướt qua bề mặt của những gì biểu thức chính quy có thể làm. Để có một ý tưởng về những gì có thể, hãy hỏi một trợ lý ảo: "Các ký tự đặc biệt phổ biến nhất được sử dụng trong biểu thức chính quy của Python là gì?"

Bạn cũng có thể hỏi về một mẫu khớp với các loại chuỗi cụ thể. Ví dụ, hãy thử hỏi:

- Viết một biểu thức chính quy của Python khớp với số điện thoại 10 chữ số có dấu gạch nối.
- Viết một biểu thức chính quy của Python khớp với địa chỉ đường phố có số và tên đường, theo sau là ST hoặc AVE .
- Viết một biểu thức chính quy của Python khớp với tên đầy đủ có bất kỳ tiêu đề thông dụng nào như Mr hoặc Mrs , theo sau là bất kỳ số lượng tên nào bắt đầu bằng chữ cái in hoa, có thể có dấu gạch ngang giữa một số tên.

Và nếu bạn muốn thấy điều gì đó phức tạp hơn, hãy thử hỏi về một biểu thức chính quy khớp với bất kỳ URL hợp lệ nào.

Một biểu thức chính quy thường có chữ cái `r` trước dấu nháy, điều này chỉ ra rằng nó là **một chuỗi thô**. Để biết thêm thông tin, hãy hỏi một trợ lý ảo: "Chuỗi thô trong Python là gì?"

```
from doctest import run_docstring_examples

def run_doctests(func):
    run_docstring_examples(func, globals(), name=func.__name__)
```

✓ Bài tập 1

Xem bạn có thể viết một hàm thực hiện cùng một việc như lệnh **shell** `!head` hay không. Hàm này nên nhận các tham số là tên của tệp để đọc, số dòng cần đọc và tên của tệp để ghi các dòng vào. Nếu tham số thứ ba là `None` , nó nên hiển thị các dòng thay vì ghi chúng vào một tệp.

Hãy xem xét việc hỏi một trợ lý ảo để được trợ giúp, nhưng nếu bạn làm vậy, hãy nói với nó không sử dụng câu lệnh `with` hoặc câu lệnh `try` .

Bạn có thể sử dụng các ví dụ sau để kiểm tra hàm của mình.

```
head('pg345_cleaned.txt', 10)
```

```
head('pg345_cleaned.txt', 100, 'pg345_cleaned_100_lines.txt')
```

```
!tail pg345_cleaned_100_lines.txt
```

✓ Bài tập 2

"Wordle" là một trò chơi từ trực tuyến, trong đó mục tiêu là đoán một từ năm chữ cái trong sáu lần thử trở xuống. Mỗi lần thử phải được công nhận là một từ, không bao gồm danh từ riêng. Sau mỗi lần thử, bạn sẽ nhận được thông tin về các chữ cái mà bạn đã đoán có xuất hiện trong từ mục tiêu hay không, và chữ nào thì ở vị trí đúng.

Ví dụ, giả sử từ mục tiêu là MOWER và bạn đoán là TRIED. Bạn sẽ biết rằng E có trong từ và ở vị trí đúng, R có trong từ nhưng không ở vị trí đúng, và T, I, và D không có trong từ.

Một ví dụ khác, giả sử bạn đã đoán các từ SPADE và CLERK, và bạn đã biết rằng E có trong từ, nhưng không ở vị trí nào trong hai từ đó, và không có chữ cái nào khác xuất hiện trong từ.

Trong số các từ trong danh sách từ, có bao nhiêu từ có thể là từ mục tiêu? Viết một hàm có tên là `check_word` nhận vào một từ năm chữ cái và kiểm tra xem nó có thể là từ mục tiêu hay không.

Bạn có thể sử dụng bất kỳ hàm nào từ chương trước, như `uses_any`.

```
def uses_any(word, letters):  
    for letter in word.lower():  
        if letter in letters.lower():  
            return True  
    return False
```

Bạn có thể sử dụng vòng lặp sau để kiểm tra hàm của mình.

```
for line in open('words.txt'):  
    word = line.strip()  
    if len(word) == 5 and check_word(word):  
        print(word)
```

Bài tập 3

Tiếp tục bài tập trước, giả sử bạn đoán từ TOTEM và biết rằng E vẫn không ở đúng vị trí, nhưng M thì có. Vậy còn lại bao nhiêu từ?

✓ Bài tập 4

The Count of Monte Cristo là một tiểu thuyết của Alexandre Dumas được coi là một tác phẩm kinh điển. Tuy nhiên, trong phần giới thiệu của một bản dịch tiếng Anh của cuốn sách, nhà văn Umberto Eco thừa nhận rằng ông thấy cuốn sách này là "một trong những tiểu thuyết được viết tệ nhất mọi thời đại".

Cụ thể, ông nói rằng nó "không biết xấu hổ trong việc lặp lại cùng một tính từ," và đề cập đặc biệt đến số lần "các nhân vật của nó hoặc là rùng mình hoặc là tái nhợt".

Để xem liệu ý kiến của ông có hợp lý hay không, hãy đếm số lần từ pale xuất hiện ở bất kỳ dạng nào, bao gồm pale, pales, paled, và paleness, cũng như từ liên quan pallor. Sử dụng một biểu thức chính quy duy nhất khớp với tất cả các từ này và không từ nào khác.

Ô dưới đây sẽ tải cuốn sách từ Project Gutenberg <https://www.gutenberg.org/ebooks/1184>.

```
import os
```

```
if not os.path.exists('pg1184.txt'):
    !wget https://www.gutenberg.org/cache/epub/1184/pg1184.txt
```

Ô dưới đây chạy một hàm đọc tệp từ Project Gutenberg và ghi một tệp chỉ chứa văn bản của cuốn sách, không bao gồm thông tin bổ sung về cuốn sách.

```
def clean_file(input_file, output_file):
    reader = open(input_file)
    writer = open(output_file, 'w')

    for line in reader:
        if is_special_line(line):
            break

    for line in reader:
        if is_special_line(line):
            break
        writer.write(line)

    reader.close()
    writer.close()

clean_file('pg1184.txt', 'pg1184_cleaned.txt')
```

Theo tôi đếm, những từ này xuất hiện 223 lần trong một cuốn sách chứa khoảng 461,000 từ. Ông Eco có thể có lý.