```
# Khối mã lệnh này cung cấp cách bạn có thể tải các hình ảnh có trong chương sách r

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
```

Vòng lặp và tìm kiếm

Năm 1939, Ernest Vincent Wright đã xuất bản một cuốn tiểu thuyết 50.000 từ có tên Gadsby mà không chứa chữ "e". Vì "e" là chữ cái phổ biến nhất trong tiếng Anh, nên việc viết thậm chí chỉ một vài từ mà không sử dụng nó đã là rất khó.

Để hiểu được mức độ khó khăn, trong chương này chúng ta sẽ tính tỷ lệ phần trăm các từ tiếng Anh có ít nhất một chữ "e".

Để làm điều đó, chúng ta sẽ sử dụng các câu lệnh `for` để lặp qua các chữ cái trong một chuỗi và các từ trong một tệp, và chúng ta sẽ cập nhật các biến trong vòng lặp để đếm số từ chứa chữ "e". Chúng ta sẽ sử dụng toán tử `in` để kiểm tra xem một chữ cái có xuất hiện trong một từ hay không, và bạn sẽ học một mẫu lập trình được gọi là "tìm kiếm tuyến tính".

Làm bài tập, bạn sẽ sử dụng các công cụ này để giải một câu đố từ có tên "Spelling Bee".

Vòng lặp và chuỗi

Trong Chương 3, chúng ta đã thấy một vòng lặp `for` sử dụng hàm `range` để hiển thị một chuỗi các số.

```
for i in range(3):
    print(i, end=' ')
```

```
0 1 2
```

Phiên bản này sử dụng tham số từ khóa `end` để hàm `print` đặt một dấu cách sau mỗi số thay vì xuống dòng.

Chúng ta cũng có thể sử dụng vòng lặp `for` để hiển thị các chữ cái trong một chuỗi.

```
for letter in 'Gadsby':
    print(letter, end=' ')
```

```
G a d s b y
```

Lưu ý rằng tôi đã thay đổi tên biến từ `i` thành `letter`, điều này cung cấp thêm thông tin về giá trị mà nó đại diện. Biến được định nghĩa trong một vòng lặp `for` được gọi là biến lặp.

Bây giờ khi chúng ta có thể lặp qua các chữ cái trong một từ, chúng ta có thể kiểm tra xem nó có chứa chữ cái "e" hay không.

```
for letter in "Gadsby":
    if letter == 'E' or letter == 'e':
        print('This word has an "e"')
```

Trước khi tiếp tục, hãy đóng gói vòng lặp đó trong một hàm.

```
def has_e():
    for letter in "Gadsby":
        if letter == 'E' or letter == 'e':
            print('This word has an "e"')
```

Và hãy biến nó thành một hàm thuần túy, trả về `True` nếu từ đó chứa chữ "e" và `False` nếu không.

```
def has_e():
    for letter in "Gadsby":
        if letter == 'E' or letter == 'e':
            return True
    return False
```

Chúng ta có thể tổng quát hóa nó để nhận từ đó làm tham số.

```
def has_e(word):
    for letter in word:
        if letter == 'E' or letter == 'e':
            return True
    return False
```

Bây giờ chúng ta có thể kiểm tra nó như sau:

```
has_e('Gadsby')
```

```
False
```

```
has_e('Emma')
```

```
True
```

Đọc danh sách từ

Để xem có bao nhiêu từ chứa chữ "e", chúng ta sẽ cần một danh sách từ. Danh sách mà chúng ta sẽ sử dụng là một danh sách khoảng 114.000 từ điển chính thức; tức là, những từ được coi là hợp lệ trong các câu đố ô chữ và các trò chơi từ khác.

Ô dưới đây tải xuống danh sách từ, là phiên bản chỉnh sửa của một danh sách được thu thập và đóng góp cộng đồng bởi Grady Ward như một phần của dự án từ điển Moby (xem http://wikipedia.org/wiki/Moby_Project).

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

```
Downloaded words.txt
```

Danh sách từ nằm trong một tệp có tên là `words.txt`, được tải xuống trong sổ tay cho chương này. Để đọc nó, chúng ta sẽ sử dụng hàm tích hợp sẵn `open`, hàm này nhận tên tệp làm tham số và trả về **một đối tượng tệp** mà chúng ta có thể sử dụng để đọc tệp.

```
file_object = open('words.txt')
```

Đối tượng tệp cung cấp một hàm có tên là `readline`, hàm này đọc các ký tự từ tệp cho đến khi gặp một ký tự xuống dòng và trả về kết quả dưới dạng chuỗi:

```
file_object.readline()
```

```
'aa\n'
```

Lưu ý rằng cú pháp để gọi `readline` khác với các hàm mà chúng ta đã thấy cho đến nay. Điều đó là vì nó là **một phương thức**, tức là một hàm liên kết với một đối tượng. Trong trường hợp này, `readline` được liên kết với đối tượng tệp, vì vậy chúng ta gọi nó bằng tên của đối tượng, toán tử chấm, và tên của phương thức.

Từ đầu tiên trong danh sách là "aa", một loại dung nham. Chuỗi `\n` đại diện cho ký tự xuống dòng tách biệt từ này với từ tiếp theo.

Đối tượng tệp theo dõi vị trí của nó trong tệp, vì vậy nếu bạn gọi `readline` một lần nữa, bạn sẽ nhận được từ tiếp theo:

```
line = file_object.readline()
line
```

```
'aah\n'
```

Để loại bỏ ký tự xuống dòng ở cuối từ, chúng ta có thể sử dụng `strip`, đó là một phương thức liên kết với các chuỗi, vì vậy chúng ta có thể gọi nó như sau.

```
word = line.strip()
word
```

```
'aah'
```

`strip` loại bỏ các ký tự khoảng trắng — bao gồm khoảng trắng, tab và ký tự xuống dòng — từ đầu và cuối chuỗi.

Bạn cũng có thể sử dụng một đối tượng tệp như một phần của vòng lặp `for`. Chương trình này đọc tệp `words.txt` và in ra mỗi từ, mỗi từ trên một dòng:

```
for line in open('words.txt'):
    word = line.strip()
    print(word)
```

```
Streaming output truncated to the last 5000 lines.
verditer
verditers
verdure
verdured
verdures
verecund
verge
verged
vergence
vergences
verger
vergers
verges
verging
verglas
verglases
veridic
verier
veriest
verifiable
verification
verifications
verified
verifier
verifiers
verifies
verify
verifying
verily
verism
verismo
verismos
verisms
```

```
verist
veristic
verists
veritable
veritably
veritas
veritates
verities
verity
verjuice
verjuices
vermeil
vermeils
vermes
vermian
vermicelli
vermicellis
vermin
vermis
vermoulu
vermouth
vermouths
vermuth
vermuths
vernacle
```

Bây giờ chúng ta có thể đọc danh sách từ, bước tiếp theo là đếm chúng. Để làm điều đó, chúng ta sẽ cần khả năng cập nhật các biến.

Cập nhật biến

Như bạn có thể đã phát hiện, việc thực hiện nhiều phép gán cho cùng một biến là hợp pháp. Một phép gán mới khiến một biến hiện có tham chiếu đến một giá trị mới (và ngừng tham chiếu đến giá trị cũ).

Ví dụ, đây là một phép gán ban đầu tạo ra một biến.

```
x = 5
x
```

```
5
```

Và đây là một phép gán thay đổi giá trị của một biến.

```
x = 7
x
```

```
7
```

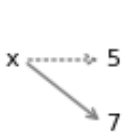
Hình dưới đây minh họa các phép gán này trong một sơ đồ trạng thái.

```
from diagram import make_rebind, draw_bindings

bindings = make_rebind('x', [5, 7])
```

```
from diagram import diagram, adjust

width, height, x, y = [0.54, 0.61, 0.07, 0.45]
ax = diagram(width, height)
bbox = draw_bindings(bindings, ax, x, y)
# adjust(x, y, bbox)
```



Mũi tên chấm chấm chỉ ra rằng `x` không còn tham chiếu đến giá trị `5`. Mũi tên liền chỉ ra rằng nó hiện tham chiếu đến giá trị `7`.

Một loại phép gán phổ biến là cập nhật, trong đó giá trị mới của biến phụ thuộc vào giá trị cũ.

```
x = 7
```

```
x = x + 1
x
```

```
8
```

Câu lệnh này có nghĩa là "lấy giá trị hiện tại của `x`, cộng thêm `1`, và gán kết quả trở lại cho `x`."

Nếu bạn cố gắng cập nhật một biến chưa tồn tại, bạn sẽ gặp lỗi, vì Python sẽ đánh giá biểu thức ở bên phải trước khi gán giá trị cho biến ở bên trái.

```
%%expect NameError

y = y + 1
```

Trước khi bạn có thể cập nhật một biến, bạn phải **khởi tạo** nó, thường là bằng một phép gán đơn giản:

```
y = 0
y = y + 1
y
```

```
1
```

Tăng giá trị của một biến được gọi là **tăng**; giảm giá trị được gọi là **giảm**. Vì những phép toán này rất phổ biến, Python cung cấp các **toán tử gán tăng cường** cho phép cập nhật một biến một cách ngắn gọn hơn.

Ví dụ, toán tử `+=` sẽ tăng giá trị của một biến theo số lượng đã cho.

```
y += 2
y
```

```
3
```

Có các toán tử gán tăng cường cho các toán tử số học khác, bao gồm `-=` và `*=`.

Vòng lặp và đếm

Chương trình dưới đây đếm số lượng từ trong danh sách từ.

```
total = 0

for line in open('words.txt'):
    word = line.strip()
    total += 1
```

Chương trình bắt đầu bằng cách khởi tạo `total` thành `0`. Mỗi lần lặp qua vòng lặp, nó sẽ tăng `total` lên `1`. Vì vậy, khi vòng lặp kết thúc, `total` sẽ tham chiếu đến tổng số từ.

```
total
```

```
113783
```

Một biến như thế này, được sử dụng để đếm số lần một sự việc xảy ra, được gọi là **bộ đếm**.

Chúng ta có thể thêm một bộ đếm thứ hai vào chương trình để theo dõi số lượng từ chứa chữ "e".

```
total = 0
count = 0

for line in open('words.txt'):
    word = line.strip()
    total = total + 1
    if has_e(word):
        count += 1
```

Hãy xem có bao nhiêu từ chứa chữ "e".

```
count
```

```
76162
```

Xét về phần trăm so với `total`, khoảng hai phần ba số từ sử dụng chữ "e".

```
count / total * 100
```

```
66.93618554617122
```

Vì vậy, bạn có thể hiểu tại sao việc viết một cuốn sách mà không sử dụng bất kỳ từ nào như vậy lại rất khó khăn.

Toán tử `in`

Phiên bản của `has_e` mà chúng ta đã viết trong chương này phức tạp hơn mức cần thiết. Python cung cấp một toán tử, `in`, để kiểm tra xem một ký tự có xuất hiện trong một chuỗi hay không.

```
word = 'Gadsby'
'e' in word
```

```
False
```

Vì vậy, chúng ta có thể viết lại `has_e` như sau.

```
def has_e(word):
    if 'E' in word or 'e' in word:
        return True
    else:
        return False
```

Và vì điều kiện của câu lệnh `if` có giá trị `boolean`, chúng ta có thể loại bỏ câu lệnh `if` và trả về giá trị `boolean` trực tiếp.

```
def has_e(word):
    return 'E' in word or 'e' in word
```

Chúng ta có thể đơn giản hóa hàm này hơn nữa bằng cách sử dụng phương thức `lower`, phương thức này chuyển đổi các chữ cái trong một chuỗi thành chữ thường. Đây là một ví dụ.

```
word.lower()
```

```
'gadsby'
```

`lower` tạo ra một chuỗi mới — nó không thay đổi chuỗi hiện có — vì vậy giá trị của `word` không bị thay đổi.

```
word
```

```
'Gadsby'
```

Dưới đây là cách chúng ta có thể sử dụng `lower` trong hàm `has_e`.

```
def has_e(word):
    return 'e' in word.lower()
```

```
has_e('Gadsby')
```

```
False
```

```
has_e('Emma')
```

```
True
```

Tìm kiếm

Dựa trên phiên bản đơn giản hơn của `has_e`, hãy viết một hàm tổng quát hơn có tên là `uses_any`, nhận một tham số thứ hai là một chuỗi các chữ cái. Hàm này sẽ trả về `True` nếu từ đó sử dụng bất kỳ chữ cái nào trong số đó và trả về `False` nếu không.

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

Đây là một ví dụ trong đó kết quả là `True`.

```
uses_any('banana', 'aeiou')
```

```
True
```

Và một ví dụ khác trong đó kết quả là `False`.

```
uses_any('apple', 'xyz')
```

```
False
```

`uses_only` chuyển đổi cả `word` và `letters` thành chữ thường, vì vậy nó hoạt động với bất kỳ kết hợp nào của chữ hoa và chữ thường.

```
uses_any('Banana', 'AEIOU')
```

```
True
```

Cấu trúc của `uses_any` tương tự như `has_e`. Nó lặp qua các chữ cái trong `word` và kiểm tra từng chữ cái một. Nếu nó tìm thấy một chữ cái xuất hiện trong `letters`, nó sẽ trả về `True` ngay lập tức. Nếu nó đi hết vòng lặp mà không tìm thấy chữ nào, nó sẽ trả về `False`.

Mẫu này được gọi là **tìm kiếm tuyến tính**. Trong các bài tập ở cuối chương này, bạn sẽ viết thêm các hàm sử dụng mẫu này.

Doctest

Trong Chương 4 chúng ta đã sử dụng docstring để tài liệu hóa một hàm — tức là, để giải thích hàm đó thực hiện chức năng gì.

Cũng có thể sử dụng docstring để *kiểm tra* một hàm. Dưới đây là một phiên bản của `uses_any` với một docstring bao gồm các bài kiểm tra.

```
def uses_any(word, letters):
    """Checks if a word uses any of a list of letters.

    >>> uses_any('banana', 'aeiou')
    True
    >>> uses_any('apple', 'xyz')
    False
    """
    for letter in word.lower():
        if letter in letters.lower():
            return True
    return False
```

Mỗi bài kiểm tra bắt đầu bằng `>>>`, được sử dụng như một dấu nhắc trong một số môi trường Python để chỉ ra nơi người dùng có thể nhập mã. Trong một doctest, dấu nhắc được theo sau bởi một biểu thức, thường là một cuộc gọi hàm. Dòng tiếp theo chỉ ra giá trị mà biểu thức nên có nếu hàm hoạt động chính xác.

Trong ví dụ đầu tiên, `banana` sử dụng chữ `a`, vì vậy kết quả nên là `True`. Trong ví dụ thứ hai, `apple` không sử dụng bất kỳ chữ nào trong `xyz`, vì vậy kết quả nên là `False`.

Để chạy các bài kiểm tra này, chúng ta phải nhập mô-đun `doctest` và chạy một hàm có tên là `run_docstring_examples`. Để làm cho hàm này dễ sử dụng hơn, tôi đã viết hàm sau, nhận một đối tượng hàm làm tham số.

```
from doctest import run_docstring_examples

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

Chúng ta vẫn chưa học về `globals` và `__name__` — bạn có thể bỏ qua chúng. Bây giờ, chúng ta có thể kiểm tra `uses_any` như sau.

```
run_doctests(uses_any)
```

```
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/lib/python3.10/doctest.py", line 1501, in run
    sys.settrace(save_trace)
```

`run_doctests` tìm các biểu thức trong docstring và đánh giá chúng. Nếu kết quả là giá trị mong đợi, bài kiểm tra **được thông qua**. Ngược lại, nó **thất bại**.

Nếu tất cả các bài kiểm tra đều đạt, `run_doctests` sẽ không hiển thị bất kỳ thông báo nào — trong trường hợp đó, không có tin tức là tin tốt. Để thấy điều gì xảy ra khi một bài kiểm tra thất bại, đây là một phiên bản sai của `uses_any`.

```
def uses_any_incorrect(word, letters):
    """Checks if a word uses any of a list of letters.

    >>> uses_any_incorrect('banana', 'aeiou')
    True
    >>> uses_any_incorrect('apple', 'xyz')
    False
    """
    for letter in word.lower():
        if letter in letters.lower():
            return True
        else:
            return False     # INCORRECT!
```

Và đây là những gì xảy ra khi chúng ta kiểm tra nó.

```
run_doctests(uses_any_incorrect)
```

```
**********************************************************************
File "__main__", line 4, in uses_any_incorrect
Failed example:
    uses_any_incorrect('banana', 'aeiou')
Expected:
    True
Got:
    False
```

Đầu ra bao gồm ví dụ mà bài kiểm tra đã thất bại, giá trị mà hàm dự kiến sẽ tạo ra, và giá trị mà hàm thực tế đã tạo ra.

Nếu bạn không chắc tại sao bài kiểm tra này thất bại, bạn sẽ có cơ hội để gỡ lỗi nó trong một bài tập.

Thuật ngữ

loop variable - biến lặp: Một biến được định nghĩa trong phần đầu của vòng lặp for.

file object - đối tượng tệp: Một đối tượng đại diện cho một tệp đã mở và theo dõi các phần nào của tệp đã được đọc hoặc ghi.

method - phương thức: Một hàm liên kết với một đối tượng và được gọi bằng toán tử chấm.

update - cập nhật: Một câu lệnh gán cung cấp một giá trị mới cho một biến đã tồn tại, thay vì tạo ra một biến mới.

initialize - khởi tạo: Tạo một biến mới và gán cho nó một giá trị.

increment - tăng: Tăng giá trị của một biến.

decrement - giảm: Giảm giá trị của một biến.

counter - bộ đếm: Một biến được sử dụng để đếm một cái gì đó, thường được khởi tạo bằng không và sau đó được tăng lên.

linear search - tìm kiếm tuyến tính: Một mẫu tính toán tìm kiếm qua một chuỗi các phần tử và dừng lại khi tìm thấy những gì nó đang tìm kiếm.

pass - đạt: Nếu một bài kiểm tra được chạy và kết quả như mong đợi, bài kiểm tra đạt.

fail - thất bại: Nếu một bài kiểm tra được chạy và kết quả không như mong đợi, bài kiểm tra thất bại.

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
```

Hỏi trợ lý ảo

Trong `uses_any`, bạn có thể đã nhận thấy rằng câu lệnh `return` đầu tiên nằm bên trong vòng lặp và câu lệnh thứ hai nằm bên ngoài.

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

Khi mọi người lần đầu tiên viết các hàm như thế này, một lỗi phổ biến là đặt cả hai câu lệnh `return` bên trong vòng lặp, như sau.

```
def uses_any_incorrect(word, letters):
    for letter in word.lower():
        if letter in letters.lower():
            return True
        else:
            return False     # INCORRECT!
```

Hãy hỏi một trợ lý ảo xem có gì sai với phiên bản này.

Bài tập 1

Viết một hàm có tên là `uses_none` nhận vào một từ và một chuỗi các chữ cái bị cấm, và trả về `True` nếu từ đó không sử dụng bất kỳ chữ cái nào trong số các chữ cái bị cấm.

Dưới đây là một phác thảo của hàm bao gồm hai doctest. Hãy hoàn thành hàm để nó vượt qua các bài kiểm tra này, và thêm ít nhất một doctest nữa.

```
def uses_none(word, forbidden):
    """Checks whether a word avoid forbidden letters.

    >>> uses_none('banana', 'xyz')
    True
    >>> uses_none('apple', 'efg')
    False
    """
    return None


run_doctests(uses_none)
```

Bài tập 2

Viết một hàm có tên là `uses_only` nhận vào một từ và một chuỗi các chữ cái, và trả về `True` nếu từ đó chỉ chứa các chữ cái trong chuỗi.

Dưới đây là một phác thảo của hàm bao gồm hai doctest. Hãy hoàn thành hàm để nó vượt qua các bài kiểm tra này, và thêm ít nhất một doctest nữa.

```
def uses_only(word, available):
    """Checks whether a word uses only the available letters.

    >>> uses_only('banana', 'ban')
    True
    >>> uses_only('apple', 'apl')
    False
    """
    return None


run_doctests(uses_only)
```

Bài tập 3

Viết một hàm có tên là `uses_all` nhận vào một từ và một chuỗi các chữ cái, và trả về `True` nếu từ đó chứa tất cả các chữ cái trong chuỗi ít nhất một lần.

Dưới đây là một phác thảo của hàm bao gồm hai doctest. Hãy hoàn thành hàm để nó vượt qua các bài kiểm tra này, và thêm ít nhất một doctest nữa.

```
def uses_all(word, required):
    """Checks whether a word uses all required letters.

    >>> uses_all('banana', 'ban')
    True
    >>> uses_all('apple', 'api')
    False
    """
    return None


run_doctests(uses_all)
```

Bài tập 4

Tờ *New York Times* xuất bản một câu đố hàng ngày có tên là "Spelling Bee" thách thức người đọc đánh vần càng nhiều từ càng tốt chỉ bằng bảy chữ cái, trong đó một trong số các chữ cái là bắt buộc. Các từ phải có ít nhất bốn chữ cái.

Ví dụ, vào ngày tôi viết điều này, các chữ cái là `ACDLORT`, với `R` là chữ cái bắt buộc. Vì vậy, "color" là một từ chấp nhận được, nhưng "told" thì không, vì nó không sử dụng `R`, và "rat" thì không vì nó chỉ có ba chữ cái. Các chữ cái có thể được lặp lại, vì vậy "ratatat" là một từ chấp nhận được.

Viết một hàm có tên là `check_word` kiểm tra xem một từ cho trước có chấp nhận được hay không. Hàm này nên nhận các tham số là từ cần kiểm tra, một chuỗi gồm bảy chữ cái có sẵn, và một chuỗi chứa một chữ cái bắt buộc duy nhất. Bạn có thể sử dụng các hàm mà bạn đã viết trong các bài tập trước.

Dưới đây là một phác thảo của hàm bao gồm doctests. Hãy hoàn thành hàm và sau đó kiểm tra xem tất cả các bài kiểm tra có đạt hay không.

```
def check_word(word, available, required):
    """Check whether a word is acceptable.

    >>> check_word('color', 'ACDLORT', 'R')
    True
    >>> check_word('ratatat', 'ACDLORT', 'R')
    True
    >>> check_word('rat', 'ACDLORT', 'R')
    False
    >>> check_word('told', 'ACDLORT', 'R')
    False
    >>> check_word('bee', 'ACDLORT', 'R')
    False
    """
    return False


run_doctests(check_word)
```

Theo quy tắc của trò chơi "Spelling Bee",

- Các từ bốn chữ cái được tính 1 điểm mỗi từ.
- Các từ dài hơn sẽ được tính 1 điểm cho mỗi chữ cái.
- Mỗi câu đố bao gồm ít nhất một "pangram", tức là từ sử dụng mọi chữ cái. Những từ này được tính thêm 7 điểm!

Viết một hàm có tên là `score_word` nhận vào một từ và một chuỗi các chữ cái có sẵn và trả về điểm số của nó. Bạn có thể giả định rằng từ đó là hợp lệ.

Dưới đây là một phác thảo của hàm với các doctest.

```
def word_score(word, available):
    """Compute the score for an acceptable word.

    >>> word_score('card', 'ACDLORT')
    1
    >>> word_score('color', 'ACDLORT')
    5
    >>> word_score('cartload', 'ACDLORT')
    15
    """
    return 0


run_doctests(word_score)
```

Khi tất cả các hàm của bạn vượt qua các bài kiểm tra, hãy sử dụng vòng lặp sau để tìm kiếm danh sách từ và cộng điểm của các từ hợp lệ.

```
available = 'ACDLORT'
required = 'R'
total = 0

file_object = open('words.txt')
for line in file_object:
    word = line.strip()
    if check_word(word, available, required):
        score = word_score(word, available)
        total = total + score
        print(word, score)

print("Total score", total)
```

Truy cập trang "Spelling Bee" tại https://www.nytimes.com/puzzles/spelling-bee và nhập các chữ cái có sẵn cho ngày hôm nay. Chữ cái ở giữa là bắt buộc.

Tôi đã tìm thấy một bộ chữ cái tạo ra các từ với tổng điểm là 5820. Bạn có thể vượt qua điểm số đó không? Tìm kiếm bộ chữ cái tốt nhất có thể là quá khó — bạn phải là người thực tế.

Bài tập 5

Bạn có thể đã nhận thấy rằng các hàm bạn viết trong các bài tập trước có nhiều điểm chung. Thực tế, chúng giống nhau đến mức bạn thường có thể sử dụng một hàm để viết hàm khác.

Ví dụ, nếu một từ không sử dụng bất kỳ chữ cái nào trong một tập hợp các chữ cái bị cấm, điều đó có nghĩa là nó không sử dụng bất kỳ chữ cái nào. Vì vậy, chúng ta có thể viết một phiên bản của `uses_none` như sau.

```
def uses_none(word, forbidden):
    """Checks whether a word avoids forbidden letters.

    >>> uses_none('banana', 'xyz')
    True
    >>> uses_none('apple', 'efg')
    False
    >>> uses_none('', 'abc')
    True
    """
    return not uses_any(word, forbidden)
```

```
run_doctests(uses_none)
```

Cũng có sự tương đồng giữa `uses_only` và `uses_all` mà bạn có thể tận dụng. Nếu bạn có một phiên bản hoạt động của `uses_only`, hãy xem liệu bạn có thể viết một phiên bản của `uses_all` gọi đến `uses_only` không.

Bài tập 6

Nếu bạn gặp khó khăn với câu hỏi trước, hãy thử hỏi một trợ lý ảo, "Cho một hàm, `uses_only`, nhận hai chuỗi và kiểm tra xem chuỗi đầu tiên chỉ sử dụng các chữ cái trong chuỗi thứ hai, hãy sử dụng nó để viết `uses_all`, nhận hai chuỗi và kiểm tra xem chuỗi đầu tiên có sử dụng tất cả các chữ cái trong chuỗi thứ hai hay không, cho phép lặp lại."

Sử dụng `run_doctests` để kiểm tra câu trả lời.

```
run_doctests(uses_all)
```

Bài tập 7

Bây giờ hãy xem liệu chúng ta có thể viết `uses_all` dựa trên `uses_any` không.

Hãy hỏi một trợ lý ảo, "Cho một hàm, `uses_any`, nhận hai chuỗi và kiểm tra xem chuỗi đầu tiên có sử dụng bất kỳ chữ cái nào trong chuỗi thứ hai hay không, bạn có thể sử dụng nó để viết `uses_all`, nhận hai chuỗi và kiểm tra xem chuỗi đầu tiên có sử dụng tất cả các chữ cái trong chuỗi thứ hai hay không, cho phép lặp lại không?"

Nếu nó nói rằng có thể, hãy chắc chắn kiểm tra kết quả!