```
# 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
```

Ảnh được tải xuống từ Lorem Picsum, một dịch vụ cung cấp giữ chỗ hình ảnh. Tên gọi này là một tham chiếu đến “lorem ipsum”, thuật ngữ dùng để chỉ giữ chỗ văn bản.

```
# Ô này tải xuống một tệp lưu trữ chứa các tệp mà chúng ta sẽ
# sử dụng cho các ví dụ trong chương này.

download('https://github.com/AllenDowney/ThinkPython/raw/v3/photos.zip');
```

```
Downloaded photos.zip
```

```
# CẢNH BÁO: Ô này xóa thư mục photos/ nếu nó đã tồn tại.
# Bất kỳ tệp nào đã có trong thư mục photos/ sẽ bị xóa.

# !rm -rf photos/
```

```
!unzip -o photos.zip
```

```
Archive:  photos.zip
   creating: photos/
  inflating: photos/notes.txt
   creating: photos/mar-2023/
  inflating: photos/mar-2023/photo2.jpg
  inflating: photos/mar-2023/photo1.jpg
   creating: photos/jan-2023/
  inflating: photos/jan-2023/photo3.jpg
  inflating: photos/jan-2023/photo2.jpg
  inflating: photos/jan-2023/photo1.jpg
   creating: photos/feb-2023/
  inflating: photos/feb-2023/photo2.jpg
  inflating: photos/feb-2023/photo1.jpg
```

Tệp và Cơ sở dữ liệu

Hầu hết các chương trình chúng ta đã thấy cho đến nay đều **tạm thời**, nghĩa là chúng chạy trong một thời gian ngắn, tạo ra đầu ra, nhưng khi kết thúc, dữ liệu của chúng biến mất. Mỗi lần bạn chạy một chương trình tạm thời, nó bắt đầu với một trạng thái trống.

Các chương trình khác là **kết nối lâu dài**: chúng chạy trong một thời gian dài (hoặc mọi lúc); chúng giữ lại ít nhất một phần dữ liệu của chúng trong bộ nhớ dài hạn; và nếu chúng tắt và khởi động lại, chúng sẽ tiếp tục từ nơi đã dừng lại.

Một cách đơn giản để các chương trình duy trì dữ liệu của chúng là bằng cách đọc và ghi tệp văn bản. Một giải pháp thay thế linh hoạt hơn là lưu trữ dữ liệu trong một cơ sở dữ liệu. Cơ sở dữ liệu là các tệp chuyên dụng có thể được đọc và ghi hiệu quả hơn so với tệp văn bản, và chúng cung cấp các khả năng bổ sung.

Trong chương này, chúng ta sẽ viết các chương trình đọc và ghi tệp văn bản, cơ sở dữ liệu, và như một bài tập, bạn sẽ viết một chương trình tìm kiếm các bản sao trùng lặp trong một bộ sưu tập ảnh. Nhưng trước khi bạn có thể làm việc với một tệp, bạn phải tìm thấy nó, vì vậy chúng ta sẽ bắt đầu với tên tệp, đường dẫn, và thư mục.

Tên tệp và Đường dẫn

Các tệp được tổ chức thành các thư mục, còn được gọi là "folder". Mỗi chương trình đang chạy có một thư mục làm việc hiện tại, là thư mục mặc định cho hầu hết các hoạt động. Ví dụ: khi bạn mở một tệp, Python sẽ tìm kiếm nó trong thư mục làm việc hiện tại.

Mô-đun **os** cung cấp các hàm để làm việc với tệp và thư mục ("os" viết tắt của "hệ điều hành"). Nó cung cấp một hàm gọi là `getcwd` để lấy tên của thư mục làm việc hiện tại.

```
# Ô này thay thế `os.cwd` bằng một hàm trả về đường dẫn giả.

import os

def getcwd():
    return "/home/dinsdale"

os.getcwd = getcwd
```

```
import os

os.getcwd()
```

```
'/home/dinsdale'
```

Kết quả trong ví dụ này là thư mục `home` của người dùng có tên `dinsdale`.

Một chuỗi như `/home/dinsdale` xác định một tệp hoặc thư mục được gọi là **đường dẫn**.

Một tên tệp đơn giản như `memo.txt` cũng được coi là một đường dẫn, nhưng nó là **một đường dẫn tương đối** vì nó chỉ định tên tệp tương đối với thư mục hiện tại. Trong ví dụ này, thư mục hiện tại là `/home/dinsdale`, vì vậy `memo.txt` tương đương với đường dẫn đầy đủ `/home/dinsdale/memo.txt`.

Một đường dẫn bắt đầu bằng `/` không phụ thuộc vào thư mục hiện tại - nó được gọi là **đường dẫn tuyệt đối**. Để tìm đường dẫn tuyệt đối đến một tệp, bạn có thể sử dụng `abspath`.

```
os.path.abspath('memo.txt')
```

```
'/home/dinsdale/memo.txt'
```

Mô-đun `os` cung cấp các hàm khác để làm việc với tên tệp và đường dẫn. `listdir` trả về một danh sách các nội dung của thư mục đã cho, bao gồm các tệp và các thư mục khác.

Dưới đây là một ví dụ liệt kê nội dung của một thư mục có tên `photos`.

```
os.listdir('photos')
```

```
['jan-2023', 'mar-2023', 'feb-2023', 'notes.txt']
```

Thư mục này chứa một tệp văn bản có tên `notes.txt` và ba thư mục. Các thư mục chứa các tệp hình ảnh ở định dạng JPEG.

```
os.listdir('photos/jan-2023')
```

```
['photo2.jpg', 'photo1.jpg', 'photo3.jpg']
```

Để kiểm tra xem một tệp hoặc thư mục có tồn tại hay không, chúng ta có thể sử dụng `os.path.exists`.

```
os.path.exists('photos')
```

```
True
```

```
os.path.exists('photos/apr-2023')
```

```
False
```

Để kiểm tra xem một đường dẫn có tham chiếu đến một tệp hay thư mục, chúng ta có thể sử dụng `isdir`, trả về `True` nếu đường dẫn tham chiếu đến một thư mục.

```
os.path.isdir('photos')
```

```
True
```

Và `isfile` trả về `True` nếu đường dẫn tham chiếu đến một tệp.

```
os.path.isfile('photos/notes.txt')
```

```
True
```

Một thách thức khi làm việc với đường dẫn là chúng trông khác nhau trên các hệ điều hành khác nhau. Trên macOS và các hệ thống UNIX như Linux, các tên thư mục và tệp trong một đường dẫn được phân tách bằng dấu gạch chéo, `/`.

Windows sử dụng dấu gạch chéo ngược, `\`. Vì vậy, nếu bạn chạy các ví dụ này trên Windows, bạn sẽ thấy dấu gạch chéo ngược trong các đường dẫn và bạn sẽ phải thay thế dấu gạch chéo trong các ví dụ.

Hoặc, để viết mã hoạt động trên cả hai hệ thống, bạn có thể sử dụng `os.path.join`, nối các thư mục và tên tệp thành một đường dẫn bằng dấu gạch chéo hoặc dấu gạch chéo ngược, tùy thuộc vào hệ điều hành bạn đang sử dụng.

```
os.path.join('photos', 'jan-2023', 'photo1.jpg')
```

```
'photos/jan-2023/photo1.jpg'
```

Sâu trong chương này, chúng ta sẽ sử dụng các hàm này để tìm kiếm một tập hợp các thư mục và tìm tất cả các tệp hình ảnh.

Chuỗi định dạng f

Một cách để chương trình lưu trữ dữ liệu là viết nó vào một tệp văn bản.

Ví dụ: giả sử bạn là một người quan sát lạc đà và bạn muốn ghi lại số lượng lạc đà bạn đã nhìn thấy trong một khoảng thời gian quan sát.

Và giả sử rằng trong một năm rưỡi, bạn đã phát hiện ra 23 con lạc đà.

Dữ liệu trong sổ ghi chép quan sát lạc đà của bạn có thể trông như thế này.

`Giải thích thêm của dịch giả`

Chuỗi f là cách viết tắt của formatted string literals, được giới thiệu từ Python 3.6 để giúp việc chèn giá trị vào chuỗi trở nên dễ dàng và trực quan hơn.

```
num_years = 1.5
num_camels = 23
```

Để viết dữ liệu này vào một tệp, bạn có thể sử dụng phương thức `write`, mà chúng ta đã thấy trong Chương 8. Đối số của `write` phải là một chuỗi, vì vậy nếu chúng ta muốn đặt các giá trị khác vào một tệp, chúng ta phải chuyển đổi chúng thành chuỗi. Cách dễ nhất để làm điều đó là với hàm tích hợp sẵn `str`.

Đây là những gì nó trông như thế này:

```
writer = open('camel-spotting-book.txt', 'w')
writer.write(str(num_years))
writer.write(str(num_camels))
writer.close()
```

Điều đó hiệu quả, nhưng `write` không thêm khoảng trắng hoặc xuống dòng trừ khi bạn bao gồm nó một cách rõ ràng. Nếu chúng ta đọc lại tệp, chúng ta sẽ thấy rằng hai số được chạy cùng nhau.

```
open('camel-spotting-book.txt').read()
```

```
'1.523'
```

Ít nhất, chúng ta nên thêm khoảng trắng giữa các số. Và trong khi chúng ta đang ở đó, hãy thêm một số văn bản giải thích.

Để viết kết hợp chuỗi và các giá trị khác, chúng ta có thể sử dụng **định dạng chuỗi f (f-string)**, là một chuỗi có chữ cái `f` trước dấu ngoặc kép mở và chứa một hoặc nhiều biểu thức Python trong dấu ngoặc nhọn.

f-string sau đây chứa một biểu thức, là tên biến.

```
f'I have spotted {num_camels} camels'
```

```
'I have spotted 23 camels'
```

Kết quả là một chuỗi trong đó biểu thức đã được đánh giá và thay thế bằng kết quả. Có thể có nhiều hơn một biểu thức.

```
f'In {num_years} years I have spotted {num_camels} camels'
```

```
'In 1.5 years I have spotted 23 camels'
```

Và các biểu thức có thể chứa toán tử và các cuộc gọi hàm.

```
line = f'In {round(num_years * 12)} months I have spotted {num_camels} camels'
line
```

```
'In 18 months I have spotted 23 camels'
```

Vì vậy, chúng ta có thể viết dữ liệu vào một tệp văn bản như thế này.

```
writer = open('camel-spotting-book.txt', 'w')
writer.write(f'Years of observation: {num_years}\n')
writer.write(f'Camels spotted: {num_camels}\n')
writer.close()
```

Cả hai **f-string** đều kết thúc bằng chuỗi `\n`, tức thêm một ký tự xuống dòng.

Chúng ta có thể đọc lại tệp như thế này:

```
data = open('camel-spotting-book.txt').read()
print(data)
```

```
Years of observation: 1.5
Camels spotted: 23
```

Trong một f-string, một biểu thức trong dấu ngoặc nhọn được chuyển đổi thành một chuỗi, vì vậy bạn có thể bao gồm danh sách, từ điển và các loại khác.

```
t = [1, 2, 3]
d = {'one': 1}
f'Here is a list {t} and a dictionary {d}'
```

```
'Here is a list [1, 2, 3] and a dictionary {'one': 1}'
```

Nếu một f-string chứa một biểu thức không hợp lệ, kết quả là một lỗi.

```
%%expect TypeError

f'This is not a valid expression {t + 2}'
```

```
TypeError: can only concatenate list (not "int") to list
```

YAML

Một trong những lý do chương trình đọc và ghi tệp là để lưu trữ **cấu hình dữ liệu**, là thông tin chỉ định những gì chương trình nên làm và như thế nào.

Ví dụ: trong một chương trình tìm kiếm ảnh trùng lặp, chúng ta có thể có một từ điển có tên `config` chứa tên của thư mục cần tìm kiếm, tên của một thư mục khác nơi nó nên lưu trữ kết quả và một danh sách các phần mở rộng tệp mà nó nên sử dụng để xác định các tệp hình ảnh.

Đây là những gì nó có thể trông như thế này:

```
config = {
    'photo_dir': 'photos',
    'data_dir': 'photo_info',
    'extensions': ['jpg', 'jpeg'],
}
```

Để viết dữ liệu này vào một tệp văn bản, chúng ta có thể sử dụng f-string, như trong phần trước. Nhưng dễ dàng hơn là sử dụng một mô-đun có tên là `yaml` được thiết kế dành riêng cho loại việc này.

Mô-đun `yaml` cung cấp các hàm để làm việc với các tệp YAML, là các tệp văn bản được định dạng để dễ dàng cho con người *và* chương trình đọc và viết.

Dưới đây là một ví dụ sử dụng hàm `dump` để ghi từ điển `config` vào một tệp YAML.

```
# Ô này cài đặt gói thư viện PyYAML, cung cấp mô-đun yaml.

try:
    import yaml
except ImportError:
    !pip install pyyaml
```

```
import yaml

config_filename = 'config.yaml'
writer = open(config_filename, 'w')
yaml.dump(config, writer)
writer.close()
```

Nếu chúng ta đọc lại nội dung của tệp, chúng ta có thể thấy định dạng YAML trông như thế nào.

```
readback = open(config_filename).read()
print(readback)
```

```
data_dir: photo_info
extensions:
- jpg
- jpeg
photo_dir: photos
```

Bây giờ, chúng ta có thể sử dụng `safe_load` để đọc lại tệp YAML.

```
reader = open(config_filename)
config_readback = yaml.safe_load(reader)
config_readback
```

```
{'data_dir': 'photo_info',
 'extensions': ['jpg', 'jpeg'],
 'photo_dir': 'photos'}
```

Kết quả là một từ điển mới chứa cùng thông tin với bản gốc, nhưng nó không phải là cùng một từ điển.

```
config is config_readback
```

```
False
```

Chuyển đổi một đối tượng như một từ điển thành một chuỗi được gọi là **tuần tự hoá**. Chuyển đổi chuỗi trở lại thành một đối tượng được gọi là **phi tuần tự hoá**. Nếu bạn tuần tự hoá và sau đó phi tuần tự hoá một đối tượng, kết quả sẽ tương đương với bản gốc.

Shelve

Cho đến nay, chúng ta đã đọc và ghi các tệp văn bản - bây giờ hãy xem xét các cơ sở dữ liệu. **Cơ sở dữ liệu** là một tệp được tổ chức để lưu trữ dữ liệu. Một số cơ sở dữ liệu được tổ chức giống như một bảng với các hàng và cột thông tin. Những cơ sở dữ liệu khác được tổ chức giống như một từ điển ánh xạ từ khóa đến giá trị; chúng đôi khi được gọi là các **cửa hàng khóa - giá trị**.

Mô-đun `shelve` cung cấp các hàm để tạo và cập nhật một cửa hàng khóa - giá trị được gọi là `"shelf"`. Để minh họa, chúng ta sẽ tạo một `shelf` để chứa chú thích cho các hình trong thư mục `photos`. Chúng ta sẽ sử dụng từ điển `config` để lấy tên của thư mục nơi chúng ta nên đặt `shelf`.

```
config['data_dir']
```

```
'photo_info'
```

Chúng ta có thể sử dụng `os.makedirs` để tạo thư mục này, nếu nó chưa tồn tại.

```
os.makedirs(config['data_dir'], exist_ok=True)
```

Và `os.path.join` để tạo một đường dẫn bao gồm tên của thư mục và tên của tệp shelf, `captions`.

```
db_file = os.path.join(config['data_dir'], 'captions')
db_file
```

```
'photo_info/captions'
```

Bây giờ, chúng ta có thể sử dụng `shelve.open` để mở tệp shelf. Tham số `c` cho biết tệp sẽ được tạo nếu cần thiết.

```
import shelve

db = shelve.open(db_file, 'c')
db
```

```
<shelve.DbfilenameShelf at 0x7cbec7197710>
```

Giá trị trả về chính thức là một đối tượng `DbfilenameShelf`, thường được gọi là đối tượng shelf.

Đối tượng shelf hoạt động theo nhiều cách giống như một từ điển. Ví dụ: chúng ta có thể sử dụng toán tử ngoặc vuông để thêm một mục, là một ánh xạ từ một khóa đến một giá trị.

```
key = 'jan-2023/photo1.jpg'
db[key] = 'Cat nose'
```

Trong ví dụ này, khóa là đường dẫn đến một tệp hình ảnh và giá trị là một chuỗi mô tả hình ảnh.

Chúng ta cũng sử dụng toán tử ngoặc vuông để tra cứu một khóa và lấy giá trị tương ứng.

```
value = db[key]
value
```

```
'Cat nose'
```

Nếu bạn thực hiện một phép gán khác cho một khóa hiện có, `shelve` sẽ thay thế giá trị cũ.

```
db[key] = 'Close up view of a cat nose'
db[key]
```

```
'Close up view of a cat nose'
```

Một số phương thức từ điển, như `keys`, `values` và `items`, cũng hoạt động với các đối tượng shelf.

```
list(db.keys())
```

```
['jan-2023/photo1.jpg']
```

```
list(db.values())
```

```
['Close up view of a cat nose']
```

Chúng ta có thể sử dụng toán tử `in` để kiểm tra xem một khóa có xuất hiện trong shelf hay không.

```
key in db
```

```
True
```

Và chúng ta có thể sử dụng một câu lệnh `for` để lặp qua các khóa.

```
for key in db:
    print(key, ':', db[key])
```

```
jan-2023/photo1.jpg : Close up view of a cat nose
```

Cũng như các tệp khác, bạn nên đóng cơ sở dữ liệu khi bạn hoàn tất.

```
db.close()
```

Bây giờ, nếu chúng ta liệt kê nội dung của thư mục dữ liệu, chúng ta sẽ thấy hai tệp.

```
# Khi bạn mở một tệp shelf, một tệp sao lưu được tạo ra có hậu tố là .bak.
# Nếu bạn chạy sổ ghi chép này nhiều hơn một lần, bạn có thể thấy tệp đó lại bị b
# Ô này xóa nó để đầu ra hiển thị trong sách là chính xác.

!rm -f photo_info/captions.bak
```

```
os.listdir(config['data_dir'])
```

```
['captions.db']
```

`captions.dat` chứa dữ liệu chúng ta vừa lưu trữ. `captions.dir` chứa thông tin về tổ chức của cơ sở dữ liệu giúp truy cập hiệu quả hơn. Hậu tố `dir` viết tắt của "directory", nhưng nó không liên quan gì đến các thư mục mà chúng ta đã làm việc chứa các tệp.

Lưu trữ cấu trúc dữ liệu

Trong ví dụ trước, các khóa và giá trị trong shelf là các chuỗi. Nhưng chúng ta cũng có thể sử dụng một shelf để chứa các cấu trúc dữ liệu như danh sách và từ điển.

Để minh họa, hãy xem lại ví dụ về từ hoá vị từ một bài tập trong Chương 11. Nhắc lại rằng chúng ta đã tạo một từ điển ánh xạ từ một chuỗi các chữ cái được sắp xếp đến danh sách các từ có thể được đánh vần bằng các chữ cái đó. Ví dụ: khóa `'opst'` ánh xạ đến danh sách `['opts', 'post', 'pots', 'spot', 'stop', 'tops']`.

Chúng ta sẽ sử dụng hàm sau để sắp xếp các chữ cái trong một từ.

```
def sort_word(word):
    return ''.join(sorted(word))
```

Và đây là một ví dụ.

```
word = 'pots'
key = sort_word(word)
key
```

```
'opst'
```

Bây giờ, hãy mở một shelf có tên `anagram_map`. Tham số `'n'` có nghĩa là chúng ta nên luôn tạo một shelf mới, trống, ngay cả khi nó đã tồn tại.

```
db = shelve.open('anagram_map', 'n')
```

Bây giờ, chúng ta có thể thêm một mục vào shelf như thế này.

```
db[key] = [word]
db[key]
```

```
['pots']
```

Trong mục này, khóa là một chuỗi và giá trị là một danh sách các chuỗi.

Giờ giả sử chúng ta tìm thấy một từ khác chứa cùng các chữ cái, như `tops`

```
word = 'tops'
key = sort_word(word)
key
```

```
'opst'
```

Khóa giống như trong ví dụ trước, vì vậy chúng ta muốn nối thêm một từ thứ hai vào cùng một danh sách các chuỗi.

Đây là cách chúng ta sẽ làm nếu `db` là một từ điển.

```
db[key].append(word)          # KHÔNG ĐÚNG
```

Nhưng nếu chúng ta chạy điều đó và sau đó tra cứu khóa trong shelf, có vẻ như nó chưa được cập nhật.

```
db[key]
```

```
['pots']
```

Đây là vấn đề: khi chúng ta tra cứu khóa, chúng ta nhận được một danh sách các chuỗi, nhưng nếu chúng ta sửa đổi danh sách các chuỗi, nó sẽ không ảnh hưởng đến shelf. Nếu chúng ta muốn cập nhật shelf, chúng ta phải đọc giá trị cũ, cập nhật nó, và sau đó ghi giá trị mới trở lại shelf.

```
anagram_list = db[key]
anagram_list.append(word)
db[key] = anagram_list
```

Bây giờ, giá trị trong shelf đã được cập nhật.

```
db[key]
```

```
['pots', 'tops']
```

Hãy hoàn thành ví dụ này bằng cách đọc danh sách từ và lưu trữ tất cả các từ đồng nghĩa vào một shelf.

```
db.close()
```

Kiểm tra các tệp tương đương

Bây giờ, hãy quay lại mục tiêu của chương này: tìm kiếm các tệp khác nhau chứa cùng dữ liệu. Một cách để kiểm tra là đọc nội dung của cả hai tệp và so sánh.

Nếu các tệp chứa hình ảnh, chúng ta phải mở chúng với chế độ `'rb'`, trong đó `'r'` có nghĩa là chúng ta muốn đọc nội dung và `'b'` chỉ ra **chế độ nhị phân**. Ở chế độ nhị phân, nội dung không được giải thích là văn bản - chúng được xử lý như một chuỗi các byte.

Đây là một ví dụ mở và đọc một tệp hình ảnh.

```
path1 = 'photos/jan-2023/photo1.jpg'
data1 = open(path1, 'rb').read()
type(data1)
```

```
bytes
```

Kết quả từ `read` là một đối tượng `bytes` - như tên gọi, nó chứa một chuỗi các byte.

Nói chung, nội dung của một tệp hình ảnh không thể đọc được bởi con người. Nhưng nếu chúng ta đọc nội dung từ một tệp thứ hai, chúng ta có thể sử dụng toán tử `==` để so sánh.

```
path2 = 'photos/jan-2023/photo2.jpg'
data2 = open(path2, 'rb').read()
data1 == data2
```

```
False
```

Hai tệp này không tương đương.

Hãy đóng gói những gì chúng ta đã có cho đến nay trong một hàm.

```
def same_contents(path1, path2):
    data1 = open(path1, 'rb').read()
    data2 = open(path2, 'rb').read()
    return data1 == data2
```

Nếu chúng ta chỉ có hai tệp, hàm này là một lựa chọn tốt. Nhưng giả sử chúng ta có một số lượng lớn các tệp và chúng ta muốn biết liệu có hai tệp nào chứa cùng một dữ liệu hay không. Sẽ không hiệu quả nếu so sánh từng cặp tệp.

Một giải pháp thay thế là sử dụng **hàm băm**, lấy nội dung của một tệp và tính toán một **mã băm (digest)**, thường là một số nguyên lớn. Nếu hai tệp chứa cùng một dữ liệu, chúng sẽ có cùng một mã băm. Nếu hai tệp khác nhau, chúng sẽ *hầu như luôn* có các mã băm khác nhau.

Mô-đun `hashlib` cung cấp một số hàm băm - hàm chúng ta sẽ sử dụng được gọi là `md5`. Chúng ta sẽ bắt đầu bằng cách sử dụng `hashlib.md5` để tạo một đối tượng `HASH`.

```
import hashlib

md5_hash = hashlib.md5()
type(md5_hash)
```

```
_hashlib.HASH
```

Đối tượng `HASH` cung cấp một phương thức `update` lấy nội dung của tệp làm đối số.

```
md5_hash.update(data1)
```

Bây giờ, chúng ta có thể sử dụng `hexdigest` để lấy mã băm dưới dạng một chuỗi các chữ số thập lục phân đại diện cho một số nguyên ở cơ số 16.

```
digest = md5_hash.hexdigest()
digest
```

```
'aa1d2fc25b7ae247b2931f5a0882fa37'
```

Hàm sau đóng gói các bước này.

```
def md5_digest(filename):
    data = open(filename, 'rb').read()
    md5_hash = hashlib.md5()
    md5_hash.update(data)
    digest = md5_hash.hexdigest()
    return digest
```

Nếu chúng ta băm nội dung của một tệp khác, chúng ta có thể xác nhận rằng chúng ta nhận được một mã băm khác nhau.

```
filename2 = 'photos/feb-2023/photo2.jpg'
md5_digest(filename2)
```

```
'6a501b11b01f89af9c3f6591d7f02c49'
```

Bây giờ chúng ta đã có gần như mọi thứ cần thiết để tìm các tệp tương đương. Bước cuối cùng là tìm kiếm một thư mục và tìm tất cả các tệp hình ảnh.

Thư mục làm việc

Hàm sau nhận làm đối số thư mục mà chúng ta muốn tìm kiếm. Nó sử dụng `listdir` để lặp qua nội dung của thư mục. Khi tìm thấy một tệp, nó in đường dẫn đầy đủ của nó. Khi tìm thấy một thư mục, nó gọi chính nó đệ quy để tìm kiếm thư mục con.

```
def walk(dirname):
    for name in os.listdir(dirname):
        path = os.path.join(dirname, name)

        if os.path.isfile(path):
            print(path)
        elif os.path.isdir(path):
            walk(path)
```

Chúng ta có thể sử dụng nó như thế này:

```
walk('photos')
```

```
photos/jan-2023/photo2.jpg
photos/jan-2023/photo1.jpg
photos/jan-2023/photo3.jpg
photos/mar-2023/photo2.jpg
photos/mar-2023/photo1.jpg
photos/feb-2023/photo2.jpg
photos/feb-2023/photo1.jpg
photos/notes.txt
```

Thứ tự của kết quả phụ thuộc vào chi tiết của hệ điều hành.

Gỡ lỗi

Khi bạn đang đọc và ghi tệp, bạn có thể gặp phải các vấn đề với khoảng trắng. Những lỗi này có thể khó gỡ lỗi vì các ký tự khoảng trắng thường không nhìn thấy được. Ví dụ: đây là một chuỗi chứa khoảng trắng, một ký tự tab được biểu thị bằng chuỗi `\t` và một dòng mới được biểu thị bằng chuỗi `\n`. Khi chúng ta in nó ra, chúng ta không thấy các ký tự khoảng trắng.

```
s = '1 2\t 3\n 4'
print(s)
```

Hàm tích hợp `repr` có thể giúp ích. Nó nhận bất kỳ đối tượng nào làm đối số và trả về một chuỗi biểu diễn của đối tượng. Đối với các chuỗi, nó biểu diễn các ký tự khoảng trắng bằng các chuỗi thoát ngược.

```
print(repr(s))
```

Điều này có thể hữu ích cho việc gỡ lỗi.

Một vấn đề khác mà bạn có thể gặp phải là các hệ thống khác nhau sử dụng các ký tự khác nhau để chỉ ra kết thúc của một dòng. Một số hệ thống sử dụng một dòng mới, được biểu thị là `\n`. Những hệ thống khác sử dụng một ký tự trả về, được biểu thị là `\r`. Một số sử dụng cả hai. Nếu bạn di chuyển các tệp giữa các hệ thống khác nhau, những mâu thuẫn này có thể gây ra sự cố.

Viết hoa tên tệp là một vấn đề khác mà bạn có thể gặp phải nếu bạn làm việc với các hệ điều hành khác nhau. Trong macOS và UNIX, tên tệp có thể chứa chữ thường, chữ hoa, chữ số, và hầu hết các ký hiệu. Nhưng nhiều ứng dụng Windows bỏ qua sự khác biệt giữa chữ thường và chữ hoa, và một số ký hiệu được phép trong macOS và UNIX không được phép trong Windows.

Thuật ngữ

ephemeral - tạm thời: Một chương trình tạm thời thường chạy trong một thời gian ngắn và khi kết thúc, dữ liệu của nó bị mất.

persistent - liên tục: Một chương trình liên tục chạy vô thời hạn và giữ lại ít nhất một số dữ liệu của nó trong bộ nhớ vĩnh viễn.

directory - thư mục: Một tập hợp các tệp và các thư mục khác.

current working directory - thư mục làm việc hiện tại: Thư mục mặc định được sử dụng bởi một chương trình trừ khi một thư mục khác được chỉ định.

path - đường dẫn: Một chuỗi xác định một chuỗi các thư mục, thường dẫn đến một tệp.

relative path - đường dẫn tương đối: Một đường dẫn bắt đầu từ thư mục làm việc hiện tại hoặc một số thư mục được chỉ định khác.

absolute path - đường dẫn tuyệt đối: Một đường dẫn không phụ thuộc vào thư mục hiện tại.

f-string - chuỗi định dạng f: Một chuỗi có chữ cái f trước dấu ngoặc kép mở và chứa một hoặc nhiều biểu thức trong dấu ngoặc nhọn.

configuration data - cấu hình dữ liệu: Dữ liệu, thường được lưu trữ trong một tệp, chỉ định những gì một chương trình nên làm và như thế nào.

serialization - tuần tự hoá: Chuyển đổi một đối tượng thành một chuỗi.

deserialization - giải tuần tự hoá: Chuyển đổi một chuỗi thành một đối tượng.

database - cơ sở dữ liệu: Một tệp có nội dung được tổ chức để thực hiện các hoạt động nhất định hiệu quả.

key-value stores - cửa hàng khóa-giá trị: Một cơ sở dữ liệu có nội dung được tổ chức giống như một từ điển với các khóa tương ứng với các giá trị.

binary mode - chế độ nhị phân: Một cách mở một tệp để nội dung được giải thích là một chuỗi các byte chứ không phải là một chuỗi các ký tự.

hash function - hàm băm: Một hàm nhận một đối tượng và tính toán một số nguyên, đôi khi được gọi là digest.

digest - mã băm: Kết quả của một hàm băm, đặc biệt khi nó được sử dụng để kiểm tra xem hai đối tượng có giống nhau hay không.

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 một trợ lý ảo

Có một số chủ đề được nêu ra trong chương này mà tôi chưa giải thích chi tiết. Dưới đây là một số câu hỏi bạn có thể hỏi trợ lý ảo để có thêm thông tin.

- "Sự khác biệt giữa các chương trình tạm thời và liên tục là gì?"
- "Một số ví dụ về các chương trình liên tục là gì?"
- "Sự khác biệt giữa đường dẫn tương đối và đường dẫn tuyệt đối là gì?"
- "Tại sao mô-đun `yaml` có các hàm gọi là `load` và `safe_load`?"
- "Khi tôi viết một shelf trong Python, các tệp có hậu tố `dat` và `dir` là gì?"
- "Ngoài các cửa hàng khóa - giá trị, còn có những loại cơ sở dữ liệu nào khác?"
- "Khi tôi đọc một tệp, sự khác biệt giữa chế độ nhị phân và chế độ văn bản là gì?"
- "Sự khác biệt giữa đối tượng byte và chuỗi là gì?"
- "Hàm băm là gì?"
- "Mã băm MD5 là gì?"

Như thường lệ, nếu bạn gặp khó khăn với bất kỳ bài tập nào sau đây, hãy cân nhắc hỏi trợ lý ảo để được trợ giúp. Cùng với câu hỏi của bạn, bạn có thể muốn dán các hàm liên quan từ chương này.

Bài tập 1

Viết một hàm có tên `replace_all` nhận làm đối số một chuỗi mẫu, một chuỗi thay thế và hai tên tệp. Nó nên đọc tệp đầu tiên và ghi nội dung vào tệp thứ hai (tạo nó nếu cần). Nếu chuỗi mẫu xuất hiện ở bất kỳ đâu trong nội dung, nó sẽ được thay thế bằng chuỗi thay thế.

Đây là một phác thảo của hàm để giúp bạn bắt đầu.

Để kiểm tra hàm của bạn, hãy đọc tệp `photos/notes.txt`, thay thế `'photos'` bằng `'images'` và ghi kết quả vào tệp `photos/new_notes.txt`.

```
source_path = 'photos/notes.txt'
open(source_path).read()
```

```
dest_path = 'photos/new_notes.txt'
old = 'photos'
new = 'images'
replace_all(old, new, source_path, dest_path)
```

```
open(dest_path).read()
```

Bài tập 2

Trong một phần trước, chúng ta đã sử dụng mô-đun `shelve` để tạo một cửa hàng khóa - giá trị ánh xạ từ một chuỗi các chữ cái được sắp xếp đến một danh sách các từ đồng nghĩa. Để hoàn thành ví dụ, hãy viết một hàm có tên `add_word` nhận làm đối số một chuỗi và một đối tượng shelf.

Nó nên sắp xếp các chữ cái của từ để tạo thành một khóa, sau đó kiểm tra xem khóa đã có trong shelf chưa. Nếu không, nó nên tạo một danh sách chứa từ mới và thêm nó vào shelf. Nếu có, nó nên nối thêm từ mới vào giá trị hiện có.

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

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

```
word_list = open('words.txt').read().split()

db = shelve.open('anagram_map', 'n')
for word in word_list:
    add_word(word, db)
```

Nếu mọi thứ hoạt động, bạn sẽ có thể tra cứu một khóa như `'opst'` và nhận được một danh sách các từ có thể được đánh vần bằng các chữ cái đó.

```
db['opst']
```

```
for key, value in db.items():
    if len(value) > 8:
        print(value)
```

```
db.close()
```

Bài tập 3

Trong một bộ sưu tập tệp lớn, có thể có nhiều hơn một bản sao của cùng một tệp, được lưu trữ trong các thư mục khác nhau hoặc với tên tệp khác nhau. Mục tiêu của bài tập này là tìm kiếm các bản sao trùng lặp. Để minh họa, chúng ta sẽ làm việc với các tệp hình ảnh trong thư mục `photos`.

Đây là cách nó sẽ hoạt động:

- Chúng ta sẽ sử dụng hàm `walk` từ để tìm kiếm thư mục này các tệp kết thúc bằng một trong các phần mở rộng trong `config['extensions']`.
- Đối với mỗi tệp, chúng ta sẽ sử dụng `md5_digest` từ để tính toán một mã băm của nội dung.
- Sử dụng một shelf, chúng ta sẽ tạo một ánh xạ từ mỗi mã băm đến một danh sách các đường dẫn có mã băm đó.
- Cuối cùng, chúng ta sẽ tìm kiếm trong shelf mã băm bất kỳ nào được ánh xạ đến nhiều tệp.
- Nếu chúng ta tìm thấy bất kỳ, chúng ta sẽ sử dụng `same_contents` để xác nhận rằng các tệp chứa cùng một dữ liệu.

Tôi sẽ đề xuất một số hàm để viết trước, sau đó chúng ta sẽ kết hợp tất cả lại với nhau.

1. Để xác định các tệp hình ảnh, hãy viết một hàm có tên `is_image` nhận một đường dẫn và một danh sách các phần mở rộng tệp, và trả về `True` nếu đường dẫn kết thúc bằng một trong các phần mở rộng trong danh sách. Gợi ý: Sử dụng `os.path.splitext` - hoặc yêu cầu trợ lý ảo viết hàm này cho bạn.

Bạn có thể sử dụng `doctest` để kiểm tra hàm của mình.

```
from doctest import run_docstring_examples

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

run_doctests(is_image)
```

2. Viết một hàm có tên `add_path` nhận làm đối số một đường dẫn và một shelf. Nó nên sử dụng `md5_digest` để tính toán một mã băm của nội dung tệp. Sau đó, nó nên cập nhật shelf, hoặc tạo một mục mới ánh xạ từ mã băm đến một danh sách chứa đường dẫn, hoặc nối thêm đường dẫn vào danh sách nếu nó tồn tại.

3. Viết một phiên bản của `walk` có tên `walk_images` nhận một thư mục và đi qua các tệp trong thư mục và các thư mục con của nó. Đối với mỗi tệp, nó nên sử dụng `is_image` để kiểm tra xem đó có phải là tệp hình ảnh hay không và `add_path` để thêm nó vào shelf.

Khi mọi thứ hoạt động, bạn có thể sử dụng chương trình sau để tạo shelf, tìm kiếm thư mục `photos` và thêm đường dẫn vào shelf, và sau đó kiểm tra xem có nhiều tệp có cùng mã băm hay không.

```
db = shelve.open('photos/digests', 'n')
walk_images('photos')

for digest, paths in db.items():
    if len(paths) > 1:
        print(paths)
```