CHAPTER 2: STRINGS AND TEXT

Hầu hết một chương trình hữu ích đều liên quan đến xử lý văn bản, dù là phân tích dữ liệu đầu vào hay tạo đầu ra. Chương này tập trung vào các vấn đề phổ biến liên quan đến thao tác văn bản. Nhiều vấn đề trong số này có thể dễ dàng được giải quyết bằng các phương thức xây dựng sẵn của strings. Tuy nhiên, các thao tác phức tạp hơn có thể yêu cầu việc sử dụng regular expressions hoặc tạo trình phân tích cú pháp. Tất cả những topic này đều được đề cập. Thêm vào đó, một và khía cạnh phức tạp khi làm việc với Unicode được giải quyết.

2.1. Splitting Strings on Any of Multiple Delimiters

Problem:

Bạn cần tách một chuỗi thành các trường, nhưng các dấu phân tách không nhất quán trong cả chuỗi.

Solution:

Phương thức split() thực sự có ý nghĩa cho các trường hợp đơn giản, và không cho phép nhiều dấu phân cách hay khoảng trống xung quanh dấu phân cách.
Trong trường hợp khi chúng ta cần một chút linh hoạt hơn, hãy sử dụng phương thức re.split().
Ví dụ:

```
>>> line = 'asdf fjdk; afed, fjek,asdf,      foo'
>>> import re
>>> re.split(r'[;,\s]\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

Discussion:

Hàm re.split() thực sự có ích bởi vì bạn có thể chỉ định nhiều mẫu cho dấu phân cách. Ví dụ, như đã thể hiện trong giải pháp, dấu phân tách là dấu phẩy, dấu chấm phẩy, hay khoảng trắng đi theo sau bất kì một lượng khoảng trắng thừa nào. Mỗi khi mẫu được tìm thấy, toàn bộ match trở thành dấu phân tách giữa bất kỳ trường nào nằm hai bên của match. Kết quả là một danh sách các trường, như khi làm với str.split().
Khi sử dụng re.split(), bạn cần một chút cẩn thận nếu mẫu biểu thức chính quy liên quan đến một nhóm được đính kèm trong dấu ngoặc đơn. Nếu các nhóm được sử dụng, sau đó các văn bản được matched cũng được bao gồm trong kết quả.
Ví dụ:

```
>>> fields = re.split(r'(;|,|\s)\s*', line)
>>> fields
['asdf', ' ', 'fjdk', ';', 'afed', ',', 'fjek', ',', 'asdf', ',', 'foo']
>>>
```

Lấy các kí tự phân tách có thể có ích trong các hoàn cảnh nhất định. Ví dụ, có thể bạn cần tách các kí tự sau này để định dạng lại chuỗi đầu ra:

```
>>> values = fields[::2]
>>> delimiters = fields[1::2] + ['']
>>> values
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>> delimiters
[' ', ';', ',', ',', ',', '']

>>> # Reform the line using the same delimiters
>>> ''.join(v+d for v,d in zip(values, delimiters))
'asdf fjdk;afed,fjek,asdf,foo'
>>>
```

Nếu bạn không muốn các kí tự phân tách có trong kết qủa, nhưng vẫn cần sử dụng các dấu ngoặc đơn để nhóm các phần của mẫu biểu thức chính quy, đảm bảo rằng bạn sử dụng một nhóm không bắt chúng, được chỉ định như (?:...).
Ví dụ:

```
>>> re.split(r'(?:,|;|\s)\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>>
```

2.2. Matching Text at the Start or End of a String.

Problem:

Bạn cần kiểm tra đầu hoặc đuôi của một chuỗi cho các mẫu văn bản cụ thể, chẳng hạn như phần mở rộng của tên file, URL schemes và nhiều hơn nữa....

Solution:

Một cách đơn giản để kiểm tra bắt đầu và kết thúc của một chuỗi là sử dụng các phương thức str.startswith() và str.endswith().
Ví dụ:

```
>>> filename = 'spam.txt'
>>> filename.endswith('.txt')
True
>>> filename.startswith('file:')
False
>>> url = 'http://www.python.org'
>>> url.startswith('http:')
```

```
True
>>>
```

Nếu bạn muốn kiểm tra cho nhiều lựa chọn, đơn giản cung cấp một tuple các khả năng cho startswith() và endswith():

```
>>> import os
>>> filenames = os.listdir('.')
>>> filenames
[ 'Makefile', 'foo.c', 'bar.py', 'spam.c', 'spam.h' ]
>>> [name for name in filenames if name.endswith(('.c', '.h')) ]
['foo.c', 'spam.c', 'spam.h' ]
>>> any(name.endswith('.py') for name in filenames)
True
>>>
```

Đây là một cách khác:

```
from urllib.request import urlopen

def read_data(name):
    if name.startswith(('http:', 'https:', 'ftp:')):
        return urlopen(name).read()
    else:
        with open(name) as f:
            return f.read()
```

Ở trên là một phần của Python, nơi một tuple được yêu cầu như đầu vào. Nếu bạn muốn hoạt động với list hoặc set thì chỉ cần chuyển nó về tuple trước bằng cách sử dụng hàm tuple().
Ví dụ:

```
>>> choices = ['http:', 'ftp:']
>>> url = 'http://www.python.org'
>>> url.startswith(choices)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: startswith first arg must be str or a tuple of str, not list
>>> url.startswith(tuple(choices))
True
>>>
```

Discussion:
Các phương thức startswith() và endswith() cung cấp một cách rất thuận tiện để thực thi kiểm tra tiền tố và hậu tố cơ bản. Các thao tác tương tự có thể được thực thi với slices, nhưng ít thanh lịch hơn.
Ví dụ:

```
>>> filename = 'spam.txt'
>>> filename[-4:] == '.txt'
True
>>> url = 'http://www.python.org'
>>> url[:5] == 'http:' or url[:6] == 'https:' or url[:4] == 'ftp:'
True
>>>
```

Bạn cũng có thể sử dụng biểu thức chính quy như một giải pháp thay thế.
Ví dụ:

```
>>> import re
>>> url = 'http://www.python.org'
>>> re.match('http:|https:|ftp:', url)
<_sre.SRE_Match object at 0x101253098>
>>>
```

Điều này hoạt động, nhưng nó thường quá mức cần thiết cho các match đơn giản. Bằng cách sử dụng recipe này là đơn giản và nhanh hơn.
Cuối cùng, nhưng không kém phần quan trọng, các phương thức startswith() và endswith() trông đẹp khi được kết hợp với các thao tác khác, chẳng hạn như các data reductions phổ biến.
Ví dụ câu lệnh này kiểm tra một dict cho sự hiện diện một số loại file nhất định:

```
if any(name.endswith(('.c', '.h')) for name in listdir(dirname)):
  ...
```

2.3. Matching Text at the Start or End of a String.
Problem:
Bạn muốn match text bằng cách sử dụng cùng một mẫu các kí tự đại diện như thường sử dụng khi làm việc với Unix shells (ví dụ như *.py, Dat[0-9]*.csv, ...)

Solution:
Module fnmatch cung cấp 2 hàm là fnmatch() và fnmatchcase() mà có thể được sử dụng để thực thi khi matching. Cách sử dụng rất đơn giản:

```
>>> from fnmatch import fnmatch, fnmatchcase
>>> fnmatch('foo.txt', '*.txt')
True
>>> fnmatch('foo.txt', '?oo.txt')
True
>>> fnmatch('Dat45.csv', 'Dat[0-9]*')
True
>>> names = ['Dat1.csv', 'Dat2.csv', 'config.ini', 'foo.py']
>>> [name for name in names if fnmatch(name, 'Dat*.csv')]
['Dat1.csv', 'Dat2.csv']
>>>
```

Thông thường, hàm fnmatch() phù hợp với các mẫu sử dụng các quy tắc tương tự như chữ hoa chữ thường như hệ thống tệp cơ bản của hệ thống.
Ví dụ:

```
>>> # On OS X (Mac)
>>> fnmatch('foo.txt', '*.TXT')
False

>>> # On Windows
>>> fnmatch('foo.txt', '*.TXT')
True
>>>
```

Nếu sự khác biệt này quan trọng, sử dụng fnmatchcase(). Nó phù hợp đúng dựa trên các quy ước chữ hoa chữ thường mà bạn cung cấp:

```
>>> fnmatchcase('foo.txt', '*.TXT')
False
>>>
```

Một tính năng thường bị bỏ qua của những hàm này là tiềm năng sử dụng với việc xử lý dữ liệu các chuỗi không phải tên file.
Ví dụ, giả sử bạn có một danh sách các địa chỉ như:

```
addresses = [
    '5412 N CLARK ST',
    '1060 W ADDISON ST',
    '1039 W GRANVILLE AVE',
    '2122 N CLARK ST',
```

```
    '4802 N BROADWAY',
]
```

Bạn có thể làm như thế này:

```
>>> from fnmatch import fnmatchcase
>>> [addr for addr in addresses if fnmatchcase(addr, '* ST')]
['5412 N CLARK ST', '1060 W ADDISON ST', '2122 N CLARK ST']
>>> [addr for addr in addresses if fnmatchcase(addr, '54[0-9][0-9]
*CLARK*')]
['5412 N CLARK ST']
>>>
```

Discussion:
Việc thực thi bằng fnmatch() nằm đâu đó giữa chức năng của các phương thức chuỗi đơn giản và toàn bộ sức mạnh của biểu thức chính quy. Nếu bạn chỉ cố gắng cũng cấp một cơ chế đơn giản cho phép các kí tự đại diện trong các thao tác xử lý dữ liệu, nó thường là một giải pháp hợp lý.
Nếu bạn thực sự thử viết code match các tên file,thay vào đó sử dụng module glod. Xem recipe 5.13 sau này.