CHAPTER 2: STRINGS AND TEXT

2.10. Working with Unicode Characters in Regular Expressions

Problem:

Bạn đang sử dụng biểu thức chính quy để xử lý văn bản, nhưng đang gặp vấn đề liên quan đến xử lý các ký tự Unicode.

Solution:

Mặc định, module re đã được lập trình với kiến thức sơ bộ liên quan đến một số lớp ký tự Unicode.
Ví dụ, \d đã khớp với bất kỳ ký tự chữ số unicode nào:

```
>>> import re
>>> num = re.compile('\d+')
>>> # ASCII digits
>>> num.match('123')
<_sre.SRE_Match object at 0x1007d9ed0>

>>> # Arabic digits
>>> num.match('\u0661\u0662\u0663')
<_sre.SRE_Match object at 0x101234030>
>>>
```

Nếu bạn cần bao gồm các ký tự Unicode cụ thể trong các mẫu, bạn có thể sử dụng trình tự thoát thông thường cho các ký tự Unicode (ví dụ, \uFFF hoặc \UFFFFFFF).
Ví dụ, đây là một regex phù hợp với tất cả các ký tự trong một vài các trang mã nguồn Arabic khác:

```
>>> arabic =
re.compile('[\u0600-\u06ff\u0750-\u077f\u08a0-\u08ff]+')
>>>
```

Khi việc thực thi các thao tác matching và searching, nó là một ý tưởng tốt để bình thường hoá và có thể vệ sinh tất cả các văn bản thành một định dạng chuẩn trước tiên (xem recipe 2.9). Tuy nhiên, nó cũng quan trọng để lưu ý về các trường hợp đặc biệt.
Ví dụ, xem hành vi của việc matching không phân biệt chữ hoa chữ thường được kết hợp với trường hợp folding sau:

```
>>> pat = re.compile('stra\u00dfe', re.IGNORECASE)
>>> s = 'straße'
>>> pat.match(s)            # Matches
<_sre.SRE_Match object at 0x10069d370>
```

```
>>> pat.match(s.upper())      # Doesn't match
>>> s.upper()                 # Case folds
'STRASSE'
>>>
```

Discussion:
Việc trộn lẫn Unicode và biểu thức chính quy thường là cách hay để làm cho đầu của bạn phát nổ. Nếu bạn đang làm nó nghiêm trọng, bạn nên xem xét cài đặt thư viện regex, nó cung cấp đầy đủ hỗ trợ cho trường hợp Unicode folding, cũng như một loạt các tính năng thú vị khác, bao gồm cả việc matching gần đúng.

2.11. Stripping Unwanted Characters from Strings

Problem:
Bạn muốn tách các ký tự không mong muốn, chẳng hạn như khoảng trống, từ đầu đến cuối, hoặc giữa văn bản.

Solution:
Phương thức strip() có thể được sử dụng để tách các ký tự khỏi đầu hoặc cuối chuỗi. lstrip() và rstrip() thực thi việc tách từ bên trái và bên phải tương ứng. Mặc định, những phương thức này tách các khoảng trống nhưng các ký tự khác có thể được đưa ra.
Ví dụ:

```
>>> # Whitespace stripping
>>> s = '   hello world  \n'
>>> s.strip()
'hello world'
>>> s.lstrip()
'hello world  \n'
>>> s.rstrip()
'   hello world'
>>>

>>> # Character stripping
>>> t = '-----hello====='
>>> t.lstrip('-')
'hello====='
>>> t.strip('-=')
'hello'
>>>
```

Discussion:
Các phương thức strip() khác nhau thường được sử dụng khi đọc và dọn dẹp dữ liệu cho việc xử lý sau đó.
Ví dụ, bạn có thể sử dụng chúng để loại bỏ khoảng trắng, xoá các dấu ngoặc kép,...
Lưu ý rằng việc stripping không áp dụng cho bất kỳ đoạn văn bản nào nằm ở giữa một chuỗi.
Ví dụ:

```
>>> s = '  hello      world   \n'
>>> s = s.strip()
>>> s
'hello      world'
>>>
```

Nếu bạn cần làm gì đó đến không gian bên trong, bạn nên cần sử dụng các kỹ thuật khác, chẳng hạn như sử dụng phương thức replace() hoặc một biểu thức chính quy thay thế.
Ví dụ:

```
>>> s.replace(' ', '')
'helloworld'
>>> import re
>>> re.sub('\s+', ' ', s)
'hello world'
>>>
```

Đó thường là trường hợp bạn muốn kết hợp các thao tác stripping chuỗi với một số loại xử lý lặp khác, chẳng hạn như đọc nhiều dòng dữ liệu từ một file. Nếu vậy, đây là một khu vực nơi một hàm sinh có thể có ích.
Ví dụ:

```
with open(filename) as f:
    lines = (line.strip() for line in f)
    for line in lines:
        ...
```

Ở đây, biểu thức **lines = (line.strip() for line in f)** hành động giống như một loại dữ liệu chuyển đổi. Nó hiệu quả vì nó không thực sự đọc dữ liệu vào bất kỳ danh sách tạm thời nào trước tiên. Nó chỉ tạo một vòng lặp nơi tất cả các dòng được xuất ra có thao tác stripping được áp dụng cho chúng.
Đối với các stripping nâng cao hơn nữa, bạn có thể chuyển sang phương thức translate(). Xem recipe tiếp theo về các chuỗi sanitizing (tạm dịch là khử trùng,vệ sinh) để biết thêm chi tiết.

2.12. Sanitizing and Cleaning Up Text

Problem:
Một số kiddie đã nhập đoạn text “pýt̂ön” vào một biểu mẫu trên trang web của bạn và bạn muốn dọn dẹp nó bằng cách nào đó.

Solution:
Vấn đề dọn dẹp và vệ sinh văn bản được áp dụng cho nhiều vấn đề liên quan đến phân tích văn bản và xử lý dữ liệu. Ở level đơn giản, bạn có thể sử dụng các hàm chuỗi cơ bản (ví dụ như str.upper(), str.lower() để chuyển văn bản thành một trường hợp chuẩn. Việc thay thế đơn giản có thể sử dụng str.replace() hoặc re.sub() có thể tập trung vào việc xoá và thay đổi các chuỗi ký tự rất cụ thể. Bạn cũng có thể bình thường hoá văn bản bằng cách sử dụng unicodedata.normalize(), như đã nói trong recipe 2.9.
Tuy nhiên, bạn có thể muốn quá trình vệ sinh thêm một bước nữa. Có lẽ, ví dụ, bạn muốn loại bỏ toàn bộ dải ký tự hoặc dải dấu phụ. Để làm như vậy, bạn có thể chuyển qua phương thức str.translate() thường bị bỏ qua.
Để minh hoạ, giả sử bạn có một chuỗi lộn xộn như sau:

```
>>> s = 'pýt̂ön\fis\tawesome\r\n'
>>> s
'pýt̂ön\x0cis\tawesome\r\n'
>>>
```

Bước đầu tiên là dọn dẹp các khoảng trống. Để làm điều này, hãy tạo một bảng dịch nhỏ và sử dụng translate():

```
>>> remap = {
...     ord('\t') : ' ',
...     ord('\f') : ' ',
...     ord('\r') : None      # Deleted
... }
>>> a = s.translate(remap)
>>> a
'pýt̂ön is awesome\n'
>>>
```

Như bạn có thể thấy ở đây, các ký tự khoảng trắng như \t và \f đã được chuyển sang một không gian đơn, \r đã bị xoá hoàn toàn.
Bạn có thể thực hiện ý tưởng remapping này một bước xa hơn và xây dựng các bảng lớn hơn.
Ví dụ, hãy xoá tất cả các ký tự kết hợp:

```
>>> import unicodedata
>>> import sys
>>> cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode)
...                      if unicodedata.combining(chr(c)))
```

```
...
>>> b = unicodedata.normalize('NFD', a)
>>> b
'pýt̂ôñ is awesome\n'
>>> b.translate(cmb_chrs)
'python is awesome\n'
>>>
```

Trong ví dụ cuối cùng, một dict mapping mỗi ký tự kết hợp Unicode cho None được tạo bằng cách sử dụng dict.fromkeys().
Đầu vào ban đầu sau đó được bình thường hoá thành một dạng bị phân huỷ bằng cách sử dụng unicodedata.normalize(). Từ đó, chức năng translate được sử dụng để xoá tất cả các dấu trọng âm. Các kỹ thuật tương tự có thể được sử dụng để xoá các loại ký tự khác (ví dụ như điều khiển các ký tự…).
Một ví dụ khác, đây là một bảng translation maps tất cả các ký tự chữ số thập phân Unicode sang giá trị tương đương của chúng trong ASCII:

```
>>> digitmap = { c: ord('0') + unicodedata.digit(chr(c))
...           for c in range(sys.maxunicode)
...           if unicodedata.category(chr(c)) == 'Nd' }
...
>>> len(digitmap)
460
>>> # Arabic digits
>>> x = '\u0661\u0662\u0663'
>>> x.translate(digitmap)
'123'
>>>
```

Tuy nhiên một số kỹ thuật khác liên quan đến việc dọn dẹp văn bản liên quan đến các chức năng giải mã và mã hoá I/O. Ý tưởng ở đây là đầu tiên làm một số thao tác dọn dẹp sơ bộ của văn bản, và khi chạy nó qua các thao tác encode() và decode() để tách hoặc thay đổi nó.
Ví dụ:

```
>>> a
'pýt̂ôñ is awesome\n'
>>> b = unicodedata.normalize('NFD', a)
>>> b.encode('ascii', 'ignore').decode('ascii')
'python is awesome\n'
>>>
```

Ở đây, quá trình chuẩn hoá (là bình thường hoá như đã nói ở trên) đã phân huỷ văn bản ban đầu thành các ký tự đi cùng với các ký tự kết hợp riêng biệt. Việc mã hoá/ giải mã đơn giản tiếp theo đã bỏ đi tất cả những ký tự không mong muốn. Đương nhiên, điều này sẽ chị hoạt động nếu nhận được một biểu diễn ASCII là mục đích cuối cùng.

Discussion:
Một vấn đề chính với việc vệ sinh văn bản có thể là hiệu suất thời gian chạy. Như công thức chung, đơn giản hơn sẽ chạy nhanh hơn. Đối với các việc thay thế đơn giản, phương thức str.replace() thường là cách tiếp cận nhanh nhất - thậm chí nếu bạn có gọi nó nhiều lần.
Ví dụ, để dọn dẹp khoảng trắng, bạn có thể sử dụng mã nguồn như thế này:

```
def clean_spaces(s):
    s = s.replace('\r', '')
    s = s.replace('\t', ' ')
    s = s.replace('\f', ' ')
    return s
```

Nếu bạn thử làm điều này, bạn sẽ tìm nó nhanh hơn một chút hơn là sử dụng translate() hoặc một cách tiếp cận bằng cách sử dụng biểu thức chính quy.
Mặt khác, phương thức translate() là rất nhanh nếu bạn cần thực thi bất kỳ loại remapping hoặc xoá ký tự không tầm thường (nontrivial character-to-character).
Trong một bức tranh lớn, hiệu suất là một cái gì đó bạn phải nghiên cứu thêm trên ứng dụng cụ thể của bạn. Thật không may, nó không thể yêu cầu một kỹ thuật cụ thể mà làm việc tốt nhất cho tất cả các trường hợp, vì vậy hãy thử các cách tiếp cận khác nhau và đo lường nó.
Mặc dù trọng tập của recipe này là văn bản, tương tự các kỹ thuật có thể được áp dụng cho các byte, bao gồm các thay thế đơn giản, bản dịch và cụm từ thông dụng.