Giới thiệu về lập trình với Python của CS50

OpenCourseWare

Quyên tặng (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan)
(https://twitter.com/davidjmalan)

Bài giảng 7

Biểu thức chính quy

- Biểu thức chính quy hoặc "biểu thức chính quy" sẽ cho phép chúng tôi kiểm tra các mẫu trong mã của mình. Ví dụ: chúng tôi có thể muốn xác thực rằng địa chỉ email được định dạng chính xác. Biểu thức chính quy sẽ cho phép chúng ta kiểm tra các biểu thức theo cách này.
- Để bắt đầu, gõ `code validate.py` vào cửa sổ terminal. Sau đó, mã như sau trong trình soạn thảo văn bản:

```
email = input("What's your email? ").strip()

if "@" in email:
    print("Valid")
else:
    print("Invalid")
```

Lưu ý rằng `strip` sẽ xóa khoảng trắng ở đầu hoặc cuối dữ liệu nhập. Chạy chương trình này, bạn sẽ thấy rằng miễn là một `@` ký hiệu được nhập vào, chương trình sẽ coi dữ liệu đầu vào đó

là hợp lệ.

- Tuy nhiên, bạn có thể tưởng tượng rằng người ta có thể nhập liệu `@@` một mình và dữ liệu nhập đó có thể được coi là hợp lệ. Chúng ta có thể coi một địa chỉ email có ít nhất một `@` và một `.` vị trí nào đó bên trong nó. Sửa đổi mã của bạn như sau:

```
email = input("What's your email? ").strip()

if "@" in email and "." in email:
    print("Valid")
else:
    print("Invalid")
```

Lưu ý rằng mặc dù điều này hoạt động như mong đợi, nhưng người dùng của chúng tôi có thể gây bất lợi, việc gõ đơn giản `@.` sẽ dẫn đến chương trình trả về `valid`.

- Chúng ta có thể cải thiện tính logic của chương trình như sau:

```
email = input("What's your email? ").strip()

username, domain = email.split("@")

if username and "." in domain:
    print("Valid")
else:
    print("Invalid")
```

Lưu ý cách `strip` sử dụng phương thức này để xác định xem `username` có tồn tại không và `.` có nằm trong `domain` biến hay không. Khi chạy chương trình này, địa chỉ email tiêu chuẩn mà bạn nhập vào có thể được coi là `valid`. Chỉ gõ vào `malan@harvard`, bạn sẽ thấy rằng chương trình coi đầu vào này là `invalid`.

- Chúng tôi thậm chí có thể chính xác hơn bằng cách sửa đổi mã của mình như sau:

```
email = input("What's your email? ").strip()

username, domain = email.split("@")

if username and domain.endswith(".edu"):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý cách `endswith` phương thức sẽ kiểm tra xem có `domain` chứa `.edu`. Tuy nhiên, một người dùng bất chính vẫn có thể phá mã của chúng tôi. Ví dụ: người dùng có thể nhập `malan@.edu` và nó sẽ được coi là hợp lệ.

- Thật vậy, chúng ta có thể tiếp tục lặp lại mã này. Tuy nhiên, hóa ra Python có một thư viện hiện có tên `re` là có một số hàm dựng sẵn có thể xác thực dữ liệu đầu vào của người dùng dựa trên các mẫu.
- Một trong những chức năng linh hoạt nhất trong thư viện `re` là `search`.

- Thư `search` viện tuân theo chữ ký `re.search(pattern, string, flags=0)`. Theo chữ ký này, chúng tôi có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search("@", email):
    print("Valid")
else:
    print("Invalid")
```

  Lưu ý rằng điều này không làm tăng chức năng của chương trình của chúng tôi chút nào. Trên thực tế, đó là một bước lùi.

- Chúng tôi có thể nâng cao hơn nữa chức năng của chương trình của mình. Tuy nhiên, chúng ta cần nâng cao vốn từ vựng của mình xung quanh `validation`. Hóa ra trong thế giới của các biểu thức chính quy có một số ký hiệu nhất định cho phép chúng ta xác định các mẫu. Tại thời điểm này, chúng tôi chỉ kiểm tra các đoạn văn bản cụ thể như `@`. Điều đó xảy ra là nhiều ký hiệu đặc biệt có thể được chuyển đến trình biên dịch nhằm mục đích xác thực. Danh sách không đầy đủ các mẫu đó như sau:

```
.   any character except a new line
*   0 or more repetitions
+   1 or more repetitions
?   0 or 1 repetition
{m} m repetitions
{m,n} m-n repetitions
```

- Triển khai điều này bên trong mã của chúng tôi, hãy sửa đổi mã của bạn như sau:

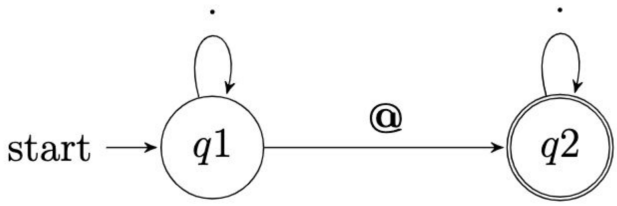
```
import re

email = input("What's your email? ").strip()

if re.search(".+@.+", email):
    print("Valid")
else:
    print("Invalid")
```

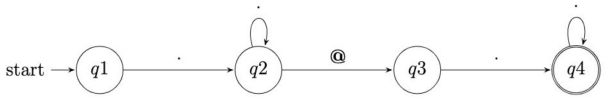
  Lưu ý rằng chúng tôi không quan tâm tên người dùng hoặc tên miền là gì. Điều chúng tôi quan tâm là mẫu mã. `.+` được sử dụng để xác định xem có nội dung nào ở bên trái địa chỉ email hay không và có nội dung nào ở bên phải địa chỉ email hay không. Chạy mã của bạn, nhập vào `malan@`, bạn sẽ nhận thấy rằng dữ liệu đầu vào được coi như `invalid` chúng tôi mong đợi.

- Nếu chúng tôi sử dụng biểu thức chính quy `.*@.*` trong mã ở trên, bạn có thể hình dung điều này như sau:



  Hãy chú ý mô tả `state machine` biểu thức chính quy của chúng tôi. Ở bên trái, trình biên dịch bắt đầu đánh giá câu lệnh từ trái sang phải. Khi chúng ta đến `q1` câu hỏi 1, trình biên dịch sẽ đọc đi đọc lại dựa trên biểu thức được truyền cho nó. Sau đó, trạng thái được thay đổi khi xem xét `q2` hoặc câu hỏi thứ hai đang được xác thực. Một lần nữa, mũi tên chỉ ra cách biểu thức sẽ được đánh giá nhiều lần dựa trên chương trình của chúng tôi. Sau đó, như được mô tả bằng vòng tròn kép, trạng thái cuối cùng của máy trạng thái sẽ đạt đến.

- Xem xét biểu thức chính quy mà chúng tôi đã sử dụng trong mã của mình, `.+@.+`, bạn có thể hình dung nó như sau:



  Lưu ý cách `q1` bất kỳ ký tự nào do người dùng cung cấp, bao gồm 'q2' là 1 hoặc nhiều ký tự lặp lại. Tiếp theo là biểu tượng '@'. Sau đó, `q3` tìm kiếm bất kỳ ký tự nào do người dùng cung cấp, bao gồm `q4` 1 hoặc nhiều ký tự lặp lại.
- Các `re` chức `re.search` năng và các chức năng tương tự tìm kiếm các mẫu.
- Tiếp tục cải tiến mã này, chúng tôi có thể cải thiện mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(".+@.+.edu", email):
    print("Valid")
else:
    print("Invalid")
```

  Tuy nhiên, hãy lưu ý rằng người ta có thể gõ vào `malan@harvard?edu` và nó có thể được coi là hợp lệ. Tại sao điều này là trường hợp? Bạn có thể nhận ra rằng trong ngôn ngữ xác thực, a `.` có nghĩa là bất kỳ ký tự nào!
- Chúng ta có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(".+@.+\.edu", email):
    print("Valid")
```

```
else:
    print("Invalid")
```

Lưu ý cách chúng tôi sử dụng "ký tự thoát" hoặc `\` như một cách liên quan đến `.` phần as của chuỗi thay vì biểu thức xác thực của chúng tôi. Kiểm tra mã của bạn, bạn sẽ nhận thấy mã nào `malan@harvard.edu` được coi là hợp lệ, mã nào `malan@harvard?edu` không hợp lệ.

- Bây giờ chúng ta đang sử dụng các ký tự thoát, đây là thời điểm thích hợp để giới thiệu "chuỗi thô". Trong Python, chuỗi thô là các chuỗi *không* định dạng các ký tự đặc biệt—thay vào đó, mỗi ký tự được lấy theo mệnh giá. Hãy tưởng tượng `\n`, ví dụ. Chúng ta đã thấy trong bài giảng trước đó, trong một chuỗi thông thường, hai ký tự này trở thành một như thế nào: một ký tự dòng mới đặc biệt. Tuy nhiên, trong một chuỗi thô, `\n` nó không được coi là `\n` ký tự đặc biệt mà là một ký tự đơn `\` và một ký tự đơn `n`. Việc đặt một `r` trước một chuỗi sẽ yêu cầu trình thông dịch Python coi chuỗi đó là một chuỗi thô, tương tự như cách đặt một `f` trước một chuỗi sẽ yêu cầu trình thông dịch Python coi chuỗi đó là một chuỗi định dạng:

```
import re

email = input("What's your email? ").strip()

if re.search(r".+@.+\.edu", email):
    print("Valid")
else:
    print("Invalid")
```

Bây giờ chúng tôi đã đảm bảo trình thông dịch Python sẽ không coi `\.` đó là một ký tự đặc biệt. Thay vào đó, chỉ đơn giản là `\` theo sau bởi một `.`—mà trong thuật ngữ biểu thức chính quy có nghĩa là khớp với một chữ ".".

- Bạn vẫn có thể tưởng tượng người dùng có thể gây ra vấn đề cho chúng tôi như thế nào! Ví dụ: bạn có thể nhập một câu chẳng hạn như `My email address is malan@harvard.edu` và toàn bộ câu này sẽ được coi là hợp lệ. Chúng tôi thậm chí có thể chính xác hơn trong mã hóa của mình.
- Tình cờ là chúng tôi có nhiều ký hiệu đặc biệt hơn để xác thực:

```
^   matches the start of the string
$   matches the end of the string or just before the newline at the end of the st
```

- Chúng tôi có thể sửa đổi mã của mình bằng cách sử dụng từ vựng đã thêm như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^.+@.+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý rằng điều này có tác dụng tìm kiếm mẫu khớp chính xác này với phần đầu và phần cuối của biểu thức đang được xác thực. Việc gõ một câu như `My email address is malan@harvard.edu` bây giờ được coi là không hợp lệ.

- Chúng tôi đề xuất rằng chúng tôi có thể làm tốt hơn nữa! Mặc dù hiện tại chúng tôi đang tìm kiếm tên người dùng ở đầu chuỗi, `@` ký hiệu và tên miền ở cuối chuỗi, chúng tôi có thể nhập bao nhiêu `@` ký hiệu tùy thích! `malan@@@harvard.edu` được coi là hợp lệ!
- Chúng ta có thể thêm vào vốn từ vựng của mình như sau:

```
[]    set of characters
[^]   complementing the set
```

- Bằng cách sử dụng những khả năng mới phát hiện này, chúng ta có thể sửa đổi biểu thức của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^[^@]+@[^@]+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý rằng `^` có nghĩa là khớp ở đầu chuỗi. Ở cuối biểu thức của chúng ta, `$` có nghĩa là khớp ở cuối chuỗi. `[^@]+` có nghĩa là bất kỳ ký tự nào ngoại trừ một `@`. Sau đó, chúng ta có một nghĩa đen `@`. `[^@]+\.edu` có nghĩa là bất kỳ ký tự nào ngoại trừ một `@` ký tự theo sau là một biểu thức kết thúc bằng `.edu`. Việc nhập vào `malan@@@harvard.edu` bây giờ được coi là không hợp lệ.

- Chúng tôi vẫn có thể cải thiện biểu thức chính quy này hơn nữa. Hóa ra có một số yêu cầu nhất định đối với địa chỉ email! Hiện tại, biểu thức xác thực của chúng tôi quá phù hợp. Chúng tôi có thể chỉ muốn cho phép các ký tự thường được sử dụng trong câu. Chúng ta có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^[a-zA-Z0-9_]+@[a-zA-Z0-9_].+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý rằng `[a-zA-Z0-9_]` cho quá trình xác thực biết rằng các ký tự phải ở giữa `a` và `z`, giữa `A` và `Z`, giữa `0` và `9` và có khả năng bao gồm một `_` ký hiệu. Kiểm tra đầu vào, bạn sẽ thấy rằng nhiều lỗi tiềm ẩn của người dùng có thể được chỉ ra.

- Rất may, các mẫu phổ biến đã được các lập trình viên chăm chỉ tích hợp vào biểu thức chính quy. Trong trường hợp này, bạn có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w.+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

  Lưu ý rằng nó `\w` giống như `[a-zA-Z0-9_]`. Cảm ơn các lập trình viên chăm chỉ!

- Dưới đây là một số mẫu bổ sung mà chúng ta có thể thêm vào vốn từ vựng của mình:

```
\d    decimal digit
\D    not a decimal digit
\s    whitespace characters
\S    not a whitespace character
\w    word character, as well as numbers and the underscore
\W    not a word character
```

- Bây giờ, chúng tôi biết rằng không chỉ có `.edu` địa chỉ email. Chúng tôi có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w.+\.(com|edu|gov|net|org)$", email):
    print("Valid")
else:
    print("Invalid")
```

  Lưu ý rằng the `|` có tác động của an `or` trong biểu thức của chúng ta.

- Thêm nhiều ký hiệu hơn nữa vào vốn từ vựng của chúng ta, dưới đây là một số biểu tượng khác cần xem xét:

```
A|B     either A or B
(...)   a group
(?:...) non-capturing version
```

Phân biệt chữ hoa chữ thường

- Để minh họa cách bạn có thể giải quyết các vấn đề xung quanh phân biệt chữ hoa chữ thường, trong đó có sự khác biệt giữa `EDU` và `edu` và tương tự, hãy tua lại mã của chúng ta như sau:

```
import re
```

```
email = input("What's your email? ").strip()

if re.search(r"^\w+@\w.+\.edu$", email):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý cách chúng tôi loại bỏ các `|` tuyên bố được cung cấp trước đó.

- Hãy nhớ lại rằng trong `re.search` hàm có một tham số cho `flags`.
- Một số biến cờ tích hợp là:

```
re.IGNORECASE
re.MULTILINE
re.DOTALL
```

Hãy xem xét cách bạn có thể sử dụng những thông tin này trong mã của mình.

- Do đó, chúng ta có thể thay đổi mã của mình như sau.

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@\w.+\.edu$", email, re.IGNORECASE):
    print("Valid")
else:
    print("Invalid")
```

Hãy chú ý cách chúng tôi thêm tham số thứ ba `re.IGNORECASE`. Chạy chương trình này với `MALAN@HARVARD.EDU`, đầu vào hiện được coi là hợp lệ.

- Hãy xem xét địa chỉ email sau đây `malan@cs50.harvard.edu`. Sử dụng mã của chúng tôi ở trên, điều này sẽ được coi là không hợp lệ. Tại sao có thể như vậy?
- Vì có thêm `.`, chương trình coi điều này là không hợp lệ.
- Hóa ra là chúng ta có thể, nhìn vào vốn từ vựng trước đó, chúng ta có thể nhóm các ý tưởng lại với nhau.

```
A|B     either A or B
(...)   a group
(?:...) non-caputuring version
```

- Chúng ta có thể sửa đổi mã của mình như sau:

```
import re

email = input("What's your email? ").strip()

if re.search(r"^\w+@(\w+\.)?\w+\.edu$", email, re.IGNORECASE):
    print("Valid")
else:
    print("Invalid")
```

Lưu ý cách `(\w+\.)?` giao tiếp với trình biên dịch rằng biểu thức mới này có thể ở đó một lần hoặc không hề có. Do đó, cả hai `malan@cs50.harvard.edu` và `malan@harvard.edu` đều được coi là hợp lệ.

- Thật thú vị, những chỉnh sửa mà chúng tôi đã thực hiện cho đến nay đối với mã của chúng tôi không bao gồm đầy đủ tất cả các hoạt động kiểm tra có thể được thực hiện để đảm bảo địa chỉ email hợp lệ. Thật vậy, đây là biểu thức đầy đủ mà người ta phải gõ để đảm bảo rằng một email hợp lệ được nhập vào:

```
^[a-zA-Z0-9.!#$%&'*+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(
```

- Có các chức năng khác trong `re` thư viện mà bạn có thể thấy hữu ích. `re.match` và `re.fullmatch` là những thứ bạn có thể thấy cực kỳ hữu ích.
- Bạn có thể tìm hiểu thêm trong tài liệu về re (https://docs.python.org/3/library/re.html) .

Dọn dẹp đầu vào của người dùng

- Bạn không bao giờ nên mong đợi người dùng luôn làm theo hy vọng của bạn về đầu vào rõ ràng. Quả thực, người dùng sẽ thường xuyên vi phạm ý định của bạn với tư cách là một lập trình viên.
- Có nhiều cách để làm sạch dữ liệu của bạn.
- Trong cửa sổ terminal, gõ `code format.py` . Sau đó, trong trình soạn thảo văn bản, mã như sau:

```
name = input("What's your name? ").strip()
print(f"hello, {name}")
```

Lưu ý rằng về cơ bản, chúng tôi đã tạo một chương trình "hello world". Chạy chương trình này và gõ vào `David` , nó hoạt động tốt! Tuy nhiên, gõ vào `Malan, David` thông báo chương trình không hoạt động như dự định. Làm cách nào chúng tôi có thể sửa đổi chương trình của mình để dọn sạch đầu vào này?

- Sửa đổi mã của bạn như sau.

```
name = input("What's your name? ").strip()
if "," in name:
    last, first = name.split(", ")
    name = f"{first} {last}"

print(f"hello, {name}")
```

Lưu ý cách `last, first = name.split(", ")` chạy nếu có `,` tên. Sau đó, tên được chuẩn hóa thành đầu và cuối. Chạy mã của chúng tôi, nhập `Malan, David` , bạn có thể thấy chương trình

này xử lý ít nhất một tình huống trong đó người dùng nhập nội dung nào đó không mong muốn như thế nào.

- Bạn có thể nhận thấy rằng việc nhập `Malan,David` không có dấu cách sẽ khiến trình biên dịch báo lỗi. Vì bây giờ chúng ta đã biết một số cú pháp biểu thức chính quy, hãy áp dụng cú pháp đó vào mã của chúng ta:

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    last, first = matches.groups()
    name = f"{first} {last}"
print(f"hello, {name}")
```

  Lưu ý rằng `re.search` có thể trả về một tập hợp kết quả khớp được trích xuất từ đầu vào của người dùng. Nếu kết quả khớp được trả về bởi `re.search`. Chạy chương trình này, gõ `David Malan` thông báo điều `if` kiện không chạy và tên được trả về. Nếu bạn chạy chương trình bằng cách gõ `Malan, David`, tên cũng được trả về đúng.

- Tình cờ là chúng tôi có thể yêu cầu quay lại các nhóm cụ thể bằng cách sử dụng `matches.group`. Chúng ta có thể sửa đổi mã của mình như sau:

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    last = matches.group(1)
    first = matches.group(2)
    name = f"{first} {last}"
print(f"hello, {name}")
```

  Lưu ý rằng, trong cách triển khai này, `group` không phải là số nhiều (không có `s`).

- Mã của chúng tôi có thể được thắt chặt hơn nữa như sau:

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

  Lưu ý cách `group(2)` và `group(1)` được nối với nhau bằng một khoảng trắng. Nhóm đầu tiên là nhóm còn lại của dấu phẩy. Nhóm thứ hai là nhóm bên phải dấu phẩy.

- Vẫn nhận ra rằng việc gõ `Malan,David` không có dấu cách vẫn sẽ làm hỏng mã của chúng ta. Vì vậy, chúng ta có thể thực hiện sửa đổi sau:

```
import re

name = input("What's your name? ").strip()
matches = re.search(r"^(.+), *(.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

Lưu ý việc bổ sung `*` trong tuyên bố xác nhận của chúng tôi. Mã này bây giờ sẽ chấp nhận và xử lý đúng cách các tệp `Malan,David`. Hơn nữa, nó sẽ xử lý đúng cách `David,Malan with many spaces in front of David`.

- Nó rất phổ biến để sử dụng `re.search` như chúng ta có trong các ví dụ trước, ở đâu `matches` trên dòng mã sau. Tuy nhiên, chúng ta có thể kết hợp những tuyên bố này:

```
import re

name = input("What's your name? ").strip()
if matches := re.search(r"^(.+), *(.+)$", name):
    name = matches.group(2) + " " + matches.group(1)
print(f"hello, {name}")
```

Lưu ý cách chúng tôi kết hợp hai dòng mã của mình. Toán tử hải mã `:=` gán một giá trị từ phải sang trái và cho phép chúng ta đặt một câu hỏi boolean cùng một lúc. Hãy quay đầu sang một bên và bạn sẽ hiểu tại sao điều này được gọi là toán tử hải mã.

- Bạn có thể tìm hiểu thêm trong tài liệu về re (https://docs.python.org/3/library/re.html) .

Trích xuất đầu vào của người dùng

- Cho đến nay, chúng tôi đã xác thực dữ liệu đầu vào của người dùng và làm sạch dữ liệu đầu vào của người dùng.
- Bây giờ, hãy trích xuất một số thông tin cụ thể từ đầu vào của người dùng. Trong cửa sổ terminal, nhập `code twitter.py` và mã như sau trong cửa sổ soạn thảo văn bản:

```
url = input("URL: ").strip()
print(url)
```

Lưu ý rằng nếu chúng ta nhập vào `https://twitter.com/davidjmalan`, nó sẽ hiển thị chính xác nội dung người dùng đã nhập. Tuy nhiên, làm cách nào chúng tôi có thể chỉ trích xuất tên người dùng và bỏ qua phần còn lại của URL?

- Bạn có thể tưởng tượng chúng ta có thể loại bỏ phần đầu của URL Twitter tiêu chuẩn như thế nào. Chúng ta có thể thử điều này như sau:

```
url = input("URL: ").strip()
```

```
username = url.replace("https://twitter.com/", "")
print(f"Username: {username}")
```

Lưu ý cách `replace` phương thức này cho phép chúng ta tìm một mục và thay thế nó bằng một mục khác. Trong trường hợp này, chúng tôi đang tìm một phần của URL và không thay thế nó bằng gì cả. Nhập URL đầy đủ `https://twitter.com/davidjmalan`, chương trình sẽ xuất ra tên người dùng một cách hiệu quả. Tuy nhiên, chương trình hiện nay có những hạn chế gì?

- Điều gì sẽ xảy ra nếu người dùng chỉ cần gõ `twitter.com` thay vì bao gồm `https://` và những thứ tương tự? Bạn có thể tưởng tượng nhiều tình huống trong đó người dùng có thể nhập hoặc bỏ qua các phần nhập của URL sẽ tạo ra kết quả lạ bởi chương trình này. Để cải thiện chương trình này, chúng ta có thể viết mã như sau:

```
url = input("URL: ").strip()

username = url.removeprefix("https://twitter.com/")
print(f"Username: {username}")
```

Lưu ý cách chúng tôi sử dụng `removeprefix` phương pháp này. Phương pháp này sẽ loại bỏ phần đầu của một chuỗi.

- Cụm từ thông dụng chỉ đơn giản cho phép chúng ta diễn đạt ngắn gọn các mẫu và mục tiêu.
- Trong `re` thư viện, có một phương thức gọi là `sub`. Phương pháp này cho phép chúng ta thay thế một mẫu bằng một mẫu khác.
- Chữ ký của `sub` phương thức như sau

```
re.sub(pattern, repl, string, count=0, flags=0)
```

Lưu ý cách `pattern` đề cập đến biểu thức chính quy mà chúng tôi đang tìm kiếm. Sau đó, có một `repl` chuỗi mà chúng ta có thể thay thế mẫu bằng. Cuối cùng, có cái `string` mà chúng tôi muốn thực hiện thay thế.

- Triển khai phương pháp này trong mã của chúng tôi, chúng tôi có thể sửa đổi chương trình của mình như sau:

```
import re

url = input("URL: ").strip()

username = re.sub(r"https://twitter.com/", "", url)
print(f"Username: {username}")
```

Lưu ý cách thực hiện chương trình này và nhập dữ liệu `https://twitter.com/davidjmalan` để tạo ra kết quả chính xác. Tuy nhiên, có một số vấn đề vẫn còn tồn tại trong mã của chúng tôi.

- Giao thức, tên miền phụ và khả năng người dùng đã nhập bất kỳ phần nào của URL sau tên người dùng đều là những lý do khiến mã này vẫn không lý tưởng. Chúng ta có thể khắc phục thêm những nhược điểm này như sau:

```
import re

url = input("URL: ").strip()

username re.sub(r"^(https?://)?(www\.)?twitter\.com/", "", url)
print(f"Username: {username}")
```

Lưu ý cách `^` dấu mũ được thêm vào url. Cũng lưu ý cách `.` trình biên dịch có thể giải thích không chính xác. Do đó, chúng ta thoát khỏi nó bằng cách sử dụng a `\` để tạo thành nó . `\.` Với mục đích dung thứ cho cả hai `http` và `https`, chúng ta thêm a `?` vào cuối `https?`, tạo thành `s` tùy chọn. Hơn nữa, để phù hợp, `www` chúng tôi thêm `(www\.)?` vào mã của mình. Cuối cùng, chỉ trong trường hợp người dùng quyết định loại bỏ hoàn toàn giao thức, `http://` or `https://` được tạo tùy chọn bằng cách sử dụng `(https?://)`.

- Tuy nhiên, chúng tôi vẫn mù quáng mong đợi rằng những gì người dùng đã nhập vào url thực sự có tên người dùng.
- Bằng cách sử dụng kiến thức về `re.search`, chúng tôi có thể cải thiện mã của mình hơn nữa.

```
import re

url = input("URL: ").strip()

matches = re.search(r"^https?://(www\.)?twitter\.com/(.+)$", url, re.IGNORECASE)
if matches:
    print(f"Username:", matches.group(2))
```

Lưu ý cách chúng tôi đang tìm kiếm biểu thức chính quy ở trên trong chuỗi do người dùng cung cấp. Cụ thể, chúng tôi đang ghi lại nội dung xuất hiện ở cuối URL bằng `(.+)$` biểu thức chính quy. Do đó, nếu người dùng không nhập URL mà không có tên người dùng thì sẽ không có thông tin đầu vào nào được hiển thị.

- Thậm chí thắt chặt hơn nữa chương trình của chúng tôi, chúng tôi có thể sử dụng `:=` toán tử của mình như sau:

```
import re

url = input("URL: ").strip()

if matches := re.search(r"^https?://(?:www\.)?twitter\.com/(.+)$", url, re.IGNORE
    print(f"Username:", matches.group(1))
```

Lưu ý rằng trình `?:` biên dịch nói với trình biên dịch rằng nó không cần phải nắm bắt những gì ở vị trí đó trong biểu thức chính quy của chúng ta.

- Tuy nhiên, chúng tôi có thể rõ ràng hơn để đảm bảo rằng tên người dùng đã nhập là chính xác. Bằng cách sử dụng tài liệu của Twitter, chúng ta có thể thêm dòng sau vào biểu thức chính quy của mình:

```
import re

url = input("URL: ").strip()

if matches := re.search(r"^https?://(?:www\.)?twitter\.com/([a-z0-9_]+)", url, re
    print(f"Username:", matches.group(1))
```

Lưu ý rằng `[a-z0-9_]+` trình biên dịch chỉ mong đợi `a-z`, `0-9` và `_` như một phần của biểu thức chính quy. Dấu hiệu này `+` cho biết chúng ta đang mong đợi một hoặc nhiều ký tự.

- Bạn có thể tìm hiểu thêm trong tài liệu về re (https://docs.python.org/3/library/re.html) .

Tổng hợp

Bây giờ, bạn đã học được một ngôn ngữ biểu thức chính quy hoàn toàn mới có thể được sử dụng để xác thực, dọn dẹp và trích xuất thông tin đầu vào của người dùng.

- Biểu thức chính quy
- Phân biệt chữ hoa chữ thường
- Dọn dẹp đầu vào của người dùng
- Trích xuất đầu vào của người dùng