CS50x Iran

Regular Expressions

Week note

پاییز و زمستان ۱۴۰۴

1 فهرست

- Regular Expressions یا به اختصار "regex" به ما این اجازه را می‌دهند تا الگوهای مدنظر خودمان را ایجاد کنیم و کار های زیادی با آن انجام دهیم. برای مثال ممکن است نیاز داشته باشیم تا ایمیل وارد شده توسط کاربر را از نظر ساختار آن بررسی و اعتبار سنجی کنیم، در این شرایط می‌توانیم از regex استفاده کنیم.
- برای شروع با دستور `code validate.py` در terminal، یک فایل جدید ایجاد کنید و کدهای زیر را داخل آن بنویسید.

validate.py

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

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

- در این برنامه تابع `strip` فضاهای خالی را در ابتدا یا انتهای ورودی حذف می‌کند. با اجرای این برنامه متوجه خواهید شد تا زمانی که ورودی کاربر شامل @ باشد، ورودی معتبر (Valid) شناخته می‌شود.

- در این صورت حتی رشته‌ای که شامل @@ باشد هم به عنوان یک ایمیل Valid یا معتبر شناخته می‌شود. برای بهتر کردن عملکرد این برنامه می‌توانیم اینطور درنظر بگیریم که رشته ورودی کاربر حداقل دارای یک @ و یک نقطه باشد. کد خود را به شکل زیر ویرایش کنید:

validate.py

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

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

- توجه داشته باشید که کاربر با وارد کردن "@." می‌تواند برنامه ما را فریب دهد و آن را به عنوان یک ایمیل Valid برای ما ارسال کند.

- برای توسعه و بهتر کردن عملکرد کد خود، آن را به شکل زیر تغییر دهید:

validate.py

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

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

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

در این برنامه ایمیل وارد شده توسط کاربر از قسمت @ به بعد به دو قسمت تقسیم می‌شود که شامل username ایمیل و domain خواهد بود. سپس بررسی می‌شود که آیا مقدار username خالی نبوده و کاراکتر نقطه (.) در متغیر domain وجود داشته باشد. حالا با وارد کردن یک ایمیل صحیح متوجه خواهیم شد که برنامه به درستی کار می‌کند و با وارد کردن ایمیلی مانند malan@harvard این برنامه اعلام می‌کند که ایمیل وارد شده Invalid و نامعتبر است.

- می‌توانیم باز هم عملکرد این برنامه را دقیق‌تر کنیم، کد خود را به صورت زیر تغییر دهید:

validate.py

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

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

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

توجه کنید که تابع `endswith` بررسی خواهد کرد که آیا دامنه domain حاوی " .edu" است یا نه. با این حال یک کاربر کنجکاو می‌تواند کد ما را بشکند و با وارد کردن ایمیلی مثل "cs50xiran@.edu" برنامه ما را فریب دهد و به عنوان یک ورودی Valid شناخته شود.

- ما می‌توانیم به این روش ادامه دهیم و مدام سعی کنیم که جلوی فریب خوردن برنامه خود را بگیریم. اما پایتون شامل یک کتابخانه به نام `re` می‌باشد که این امکان را به ما می‌دهد تا بتوانیم الگوهای مختلفی را از یک متن پیدا و استخراج کنیم.
- یکی از کاربردی ترین توابع در کتابخانه `re`، تابع `search` است.
- کتابخانه `search` به شکل روبه‌رو استفاده می‌شود `re.search(pattern, string, flags=0)` که به این شکل می‌توانیم یک ایمیل صحیح را تشخیص دهیم:

validate.py

```
import re

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

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

توجه کنید که این برنامه نه تنها پیشرفتی در برنامه ما ایجاد نکرده، بلکه یک قدم پسرفت هم کرده است.

- ما برای اینکه بتوانیم بررسی دقیق‌تری روی ایمیل ورودی داشته باشیم، نیاز داریم تا آن را از جنبه‌های مختلفی غیر از وجود @ بررسی کنیم. زیرا ممکن است این علامت در بسیاری از متن‌ها وجود داشته باشد. در دنیای regex کاراکتر و نمادهایی (symbols) مختلفی وجود دارد که هر کدام دارای معنا و کاربرد خاصی هستند. بعضی از آن‌ها را می‌توانید در لیست زیر ببینید.

```
.    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
```

repetition به معنای تکرار است.

- حالا کد خود را به صورت زیر تغییر دهید:

validate.py

```
import re

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

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

در این برنامه برای ما اهمیتی ندارد که مقدار username و domain چیست. تنها چیزی که الگوی ما یعنی `.+` بررسی می‌کند این است که سمت چپ و راست `@` نباید خالی باشد. حالا برای تست می‌توانید ایمیلی مانند cs50xiran@ را وارد کنید تا متوجه شوید که خروجی برنامه برای این ایمیل Invalid و نامعتبر است.

- توابع ماژول `re` مانند `search` در یک متن به دنبال الگویی خاص می‌گردند.
- برای بهتر کردن کد خود آن را به شکل زیر تغییر دهید.

validate.py

```
import re

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

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

در این برنامه کاربر می‌تواند "cs50@iran?edu" را وارد کرده و برنامه آن را Valid در نظر بگیرد. چرا این طور است؟ به این دلیل است که در زبان کاراکترهای اعتبارسنجی، نقطه (.) به معنای هر کارکتر می‌باشد.

- می‌توانیم کد خود را به صورت زیر تغییر دهیم:

validate.py

```
import re

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

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

توجه داشته باشید که چگونه از escape character یا "\" برای در نظر گرفتن "." به عنوان بخشی از رشته به جای اعتبارسنجی استفاده می‌کنیم. حالا کد خود را تست کنید. متوجه خواهید شد که "cs50x@iran.edu" معتبر (Valid) نظر گرفته می‌شود، در حالی که "cs50x@iran?edu" نامعتبر (Invalid) است.

- حالا که بحث escape characters شد، زمان خوبیست تا راجب raw strings صحبت کنیم. همانطور که در هفته‌های قبل بررسی کردیم، زمانی که بخواهیم یک خط جدید در رشته ایجاد کنیم باید از کاراکتر \n استفاده کنیم. این کاراکتر یک معنی خاص دارد که وقتی داخل رشته قرار داده باشد، آن معنی خاص درنظر گرفته شده و یک خط جدید ایجاد می‌شود. انواع مختلفی از این نوع کاراکتر ها وجود دارند. اما اگر بخواهیم تنها معنی خود کاراکتر درنظر گرفته شود و نه معنی خاص آن باید چه کرد؟ در این صورت می‌توانیم از raw strings استفاده کنیم. در پایتون برای ایجاد یک raw string، در ابتدای رشته و قبل از کوتیشن و یا دابل کوتیشن از r استفاده می‌کنیم. به این صورت پایتون متوجه می‌شود که هدف ما این است که یک رشته خام ایجاد کنیم. به نمونه کد زیر توجه کنید:

validate.py

```
import re

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

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

اکنون مطمئن شدیم که پایتون "\" را به عنوان یک کاراکتر خاص در نظر نمی‌گیرد. در عوض، به سادگی به عنوان یک "\" به دنبال یک "." در نظر گرفته می‌شود که، در regex، به معنی یک "." است.

- کاربر ما همچنان می‌تواند با وارد کردن عبارتی مانند My email address is cs50x@iran.edu خروجی Valid از برنامه بگیرد.

به این دلیل ما همچنان می‌توانیم کد دقیق‌تری بنویسیم.

- در بین کاراکترهای اعتبار سنجی ما طیف گسترده‌ای از عملکردها را می‌توانیم پوشش دهیم. مانند:

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

- با اضافه کردن کاراکترهای معرفی شده می‌توانیم کد خود را به شکل زیر تغییر دهیم حالا این برنامه تنها عباراتی را به عنوان ایمیل معتبر درنظر می‌گیرد که قبل و بعد از آدرس ایمیل متن دیگری وجود نداشته باشد.

بنابراین عبارت My email address is cs50x@iran.edu دیگر به عنوان یک ورودی معتبر تلقی نمی‌شود:

validate.py

```
import re

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

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

- با اینکه برنامه‌ی ما به درستی کار می‌کند، اما باز هم می‌توانیم آن را ارتقا دهیم. دقت کنید که کاربر با وارد کردن ایمیلی مانند cs50x@@iran.edu همچنان خروجی معتبر (Valid) را از برنامه می‌گیرد و ما می‌توانیم این مشکل را نیز رفع کنیم.

- دو کاراکتر پرکاربرد دیگر ار کاراکترهای اعتبارسنجی عبارت‌اند از:

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

- به کمک این دو کاراکتر می‌توانیم کد را به شکل زیر تغییر دهیم:

validate.py

```
import re

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

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

- در این الگو، کاراکتر ^ به معنای این است که عبارت مدنظر باید از ابتدای رشته شروع شده باشد و قبل از آن متن دیگری نباید قرار بگیرد. $ به این معنی است که عبارت مدنظر باید در انتهای رشته قرار داشته باشد و متن دیگری در ادامه آن قرار نگرفته باشد. [^@]+ به معنی این است که هر کاراکتری غیر از @ را در نظر بگیرید. سپس بصورت مشخص از @ استفاده می‌کنیم که وجود خود @ در ایمیل را بررسی کند و سپس [^@]+\.edu که به دنبال این الگو، هر کاراکتری غیر از @ در نظر گرفته خواهد شد. حالا ایمیلی مانند cs50x@@iran.edu دیگر Invalid شناخته خواهد شد.

- ما هنوز هم می‌توانیم این عبارت منظم را بهتر کنیم. برای نوشتن آدرس ایمیل الگوی خاصی وجود دارد. در حال حاضر، عبارت اعتبارسنجی ما گسترده است و موارد زیادی را شامل می‌شود. ما می‌خواهیم فقط کاراکترهایی را که در جملات استفاده می‌شوند، مجاز کنیم. می‌توانیم کد خود را به صورت زیر تغییر دهیم:

validate.py

```
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")
```

- توجه داشته باشید که `[a-zA-Z0-9_]` به الگویی گفته می‌شود که کاراکترها باید بین z و a، بین A و Z، بین 0 و 9 را در نظر بگیرند و یا شامل یک نماد _ باشند. با تست برنامه متوجه خواهید شد که بسیاری از اشتباهات کاربر در وارد کردن ایمیل تشخیص داده خواهد شد.

- خوشبختانه، الگوهای رایج در regex توسط برنامه نویسان ساخته شده است. می توانید کد خود را به صورت زیر تغییر دهید:

validate.py

```
import re

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

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

- توجه کنید که `\w` همان `[a-zA-Z0-9_]` است. دمتون گرم برنامه نویسای سخت کوش!

- در اینجا چند الگوی اضافی وجود دارد که می‌توانیم به واژگان خود اضافه کنیم:

```
\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
```

- از آنجایی که ایمیل‌هایی با دامنه edu. ایمیل‌های رایجی نیستند، بنابراین ما باید سایر ایمیل‌ها را نیز Valid درنظر بگیریم. توجه کنید که در الگوهای اعتبارسنجی کاراکتر "|" معادل همان or در پایتون می‌باشد.

validate.py

```
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")
```

- با اضافه کردن کاراکترهای بیشتر برای اعتبار سنجی می‌توانیم این فرایند را دقیق‌تر انجام دهیم، برای مثال:

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

- برای این که نشان دهیم چگونه می‌توانید حساسیت به بزرگ و کوچک بودن کاراکتر ها را مدیریت کنید، به کد قبل برمی‌گردیم و کد خود را دوباره به شکل زیر می‌نویسیم:

validate.py

```
import re

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

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

- توجه داشته باشید که چگونه "i" را حذف کرده ایم، طبق توضیحاتی که قبلا درباره عملکرد این کاراکتر ارائه شده است.
- به این نکته دقت کنید که در تابع `re.search`، یک پارامتر به نام flags وجود دارد.

- در نظر بگیرید که چگونه می‌توانید از متغیرهای flags داخلی در کد خود استفاده کنید. برخی از متغیرهای flags داخلی عبارتند از:

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

- با کمک این متغیرها ما می‌توانیم برنامه‌ی خود را به شکل زیر تغییر دهیم. توجه کنید که چگونه پارامتر سوم `re.IGNORECASE` را اضافه کردیم. با دادن CS50X@IRAN.EDU به برنامه، خروجی برنامه Valid خواهد بود.

validate.py

```
import re

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

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

- آدرس ایمیل cs50x@cs50x.iran.edu را درنظر بگیرید. با استفاده از کد بالا، برنامه Invalid تلقی می‌شود. چرا ممکن است اینطور باشد؟ دلیل این است که در قسمت دوم ایمیل یک نقطه اضافه تر از نمونه ایمیل قبل وجود داد.

validate.py

```
import re

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

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

- بیایید یکبار دیگر با چند نمونه از کاراکترهای اعتبارسنجی آشنا شویم.

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

- حالا توانیم کد خود را به صورت زیر تغییر دهیم:

validate.py

```
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")
```

- توجه کنید که چگونه `(\w+\.)?` به مفسر اطلاع می‌دهد که این عبارت جدید می‌تواند یک بار وجود داشته باشد یا اصلا وجود نداشته باشد. از این رو، هر دو cs50x@cs50x.iran.edu و cs50x@iran.edu معتبر یا Valid در نظر گرفته می‌شوند.

- با وجود تمام تغییراتی که ما اعمال کردیم و اصلاحاتی که در الگوی خود ایجاد کردیم، بازهم نمی‌توانیم از اون به عنوان یک الگوی کامل نام ببریم. برای این هدف میتوانیم از الگوی زیر استفاده کنیم:

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

- توابع دیگری در کتابخانه `re` وجود دارد که می‌توانند کارهای جذابی را انجام دهند. `re.match` و `re.fullmatch` مواردی هستند که ممکن است بسیار مفید باشند.

- شما می‌توانید در مستندات python از re اطلاعات بیشتری کسب کنید.

- شما هرگز نباید انتظار داشته باشید که کاربران همیشه بصورتی که انتظار دارید ورودی را وارد کنند. در واقع، کاربران اغلب اهداف شما را به عنوان یک برنامه نویس نقض می‌کنند.
- راه های مختلفی وجود دارد تا دیتاهای شما به آن صورت که مدنظر دارید تمیز و قابل بررسی شود.
- در ترمینال خود دستور `code format.py` را وارد کنید و کدهای زیر را درون آن قرار دهید.

format.py

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

- در اینجا فقط یک برنامه برای چاپ عبارت hello world ایجاد شده است. با اجرای این برنامه و تایپ David برنامه خوب کار می‌کند! با این حال، برنامه با تایپ David, Malan متوجه می‌شود که چگونه برنامه آنطور که در نظر گرفته شده عمل نمی‌کند. چگونه می‌توانیم برنامه خود را طوری توسعه دهیم که توانایی مدیریت کردن این شرایط را داشته باشد؟

- کد خود را به صورت زیر تغییر دهید:

format.py

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

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

- توجه داشته باشید که اگر یک ویرگول در name وجود داشته باشد، تابع `name = f"{first} {last}"` اجرا می‌شود. سپس، `name` به دو قسمت تقسیم می‌شود. با اجرای کد متوجه می‌شوید که برنامه ما این شرایط مخصوص را می‌تواند مدیریت کند.

- ممکن است متوجه شوید که تایپ کردن Malan,David بدون فاصله بعد از ویرگول باعث می‌شود که برنامه نتواند به درستی اجرا شود. از آنجایی که با regex آشنا شدیم، بیاید از آن برای رفع این مشکل استفاده کنیم. کد خود را به شکل زیر تغییر دهید:

format.py

```
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}")
```

- تابع `re.search` این امکان را دارد تا قسمت‌هایی از رشته که مطابق الگوی ما باشد را بصورت گروه بندی شده ارسال کند. برنامه را اجرا کنید و آن را با دو ورودی Malan, David و David Malan تست کنید. می‌بینند که در هر دو صورت برنامه به درستی خروجی می‌دهد.

- همچنین ما می‌توانستیم گروهی خاص از رشته‌ها که مطابق الگو بوده را استخراج کنیم. برای این کار مطابق کد زیر پیش برید:

format.py

```
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}")
```

- به این نکته دقت کنید که تابعی که اینجا استفاده شده `group` بوده و با تابع `groups` که در قسمت قبل بررسی کردیم تفاوت دارد.

- همچنین برای ساده تر کردن کد می‌توانیم آن را به شکل زیر تغییر دهیم:

format.py

```
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}")
```

- با وجود تغییراتی که دادیم، همچنان وارد کردن Malan,David بدون فاصله عملکرد کد ما را دچار مشکل می‌کند. برای رفع این مشکل کد را به صورت شکل زیر تغییر دهید:

format.py

```
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}")
```

- به اضافه شدن * در متن اعتبارسنجی ما توجه کنید. این کد اکنون Malan,David را می‌پذیرد و به درستی پردازش می‌کند. علاوه بر این، به درستی David,Malan با هر مقدار فاصله در ادامه آن را نیز مدیریت می‌شود.
- استفاده از `re.search` همانطور که در مثال‌های قبلی انجام دادیم بسیار رایج است، جایی که تطبیق‌ها در یک خط کد پس از آن قرار دارند. با این حال، می‌توانیم این عبارات را با هم ترکیب کنیم:

format.py

```
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}")
```

- توجه کنید که چگونه دو خط کد خود را ترکیب می‌کنیم. عملگر := مقداری را از راست به چپ اختصاص می‌دهد و به ما اجازه می‌دهد در همان زمان یک سوال منطقی بپرسیم. سر خود را به طرفین بچرخانید و خواهید دید که چرا به این حالت اپراتور walrus می‌گویند.

- تا اینجای کار ما ورودی کاربر را دریافت کردیم و سپس اعتبارسنجی و مرتب کردن آن را انجام دادیم.
- حالا میخوایم یکسری اطلاعات خاص و مشخص را از ورودی کاربر استخراج کنیم. در پنجره ی ترمینال خود `code twitter.py` را وارد کنید و در این فایل ایجاد شده تکه کد زیر را تایپ کنید:

twitter.py

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

- توجه کنید که اگر ما به عنوان ورودی https://twitter.com/davidjmalan را وارد کنیم، در آخر هم همین تکست را در خروجی تحویل می‌گیریم، اما اگر فقط username را بخواهیم و سایر قسمت های لینک را دور بریزیم چه کار باید کنیم؟
- برای انجام اینکار و در کل خلاص شدن از بخش ثابت لینک توییتر، می‌توانیم کد زیر را امتحان کنیم:

twitter.py

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

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

- دقت کنید که تابع `replace` به ما اجازه می‌دهد تا قسمتی از یک رشته را انتخاب و آن را با رشته دیگری جایگزین کنیم. در اینجا ما بخش استاندارد و ثابت لینک اکانت توییتر را پیدا کرده و آن را با یک رشته خالی جایگزین می‌کنیم. با این کار تنها بخشی که از لینک پروفایل توییتر باقی می‌ماند، username می‌باشد. در این حالت اگر کاربر ما در ورودی خود https://twitter.com/davidjmalan را وارد کند، برنامه به درستی کار می‌کند و username را به ما خروجی می‌دهد، اما این کد ایرادهایی دارد و بسیاری از خطاهای کاربر را مدیریت نمی‌کند. به عنوان مثال اگر کاربر بخش https:// را وارد نکند، برنامه کار نمی‌کند و در آخر twitter.com/davidjmalan را به ما در خروجی نمایش می‌دهد.
- حالا سعی کنید تمام حالاتی را که ممکن است باعث شود تا خروجی درست نمایش داده نشود را تصور کنید و در نظر بگیرید. برای ارتقای این کد می‌توان تغییرات زیر را اعمال کرد:

twitter.py

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

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

به متد `removeprefix` توجه کنید. این متد بخش اول یک رشته را پاک می‌کند.

- regex به ما این اجازه را می‌دهند تا به صورت خلاصه الگوی مدنظر خود را ارائه دهیم.
- داخل کتابخانه‌ی `re` متدی به نام `sub` وجود دارد که به ما امکان جایگزینی یک الگو را با رشته دیگری می‌دهد.
- نحوه‌ی استفاده از متد `sub` به این صورت است:

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

- دقت کنید که pattern الگوی مدنظر ماست. سپس به repl می‌رسیم که در واقع رشته‌ای است که می‌خواهیم الگو را با آن جایگزین کنیم و در آخر به string که همان رشته اصلی است که جایگزینی باید روی آن صورت گیرد.
- حال با پیاده سازی این متد در کد خود، می‌توانیم برنامه‌ی خود را اصلاح کنیم:

twitter.py

```
import re

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

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

- حال برنامه را با همان ورودی قبلی (https://twitter.com/davidjmalan) اجرا کنید. دقت کنید که اینبار برنامه‌ی ما درست کار کرد و خروجی مد نظر را به ما داده است. ولی همچنان مشکلاتی در برنامه ما وجود دارد.
- قسمت protocol، subdomain و هر قسمت دیگر از لینک که کاربر می‌تواند آن را به شکلی دیگر وارد کند قسمت هایی هستند که برنامه ما در آن ضعف خواهد داشت. اما بازهم این امکان را داریم تا ضعف های آن را کمتر و کمتر کنیم.

twitter.py

```
import re

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

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

دقت کنید که نشان ^ به ورودی اضافه شده و حتی یک نقطه هم می‌تواند توسط مفسر اشتباه تفسیر شده باشد. به همین دلیل با استفاده از \ و تبدیل آن به \. می‌توانیم آن را دور بزنیم.

- برای اینکه برنامه بتواند هر دوی http و https را تشخیص دهد به آخر http یک ؟ اضافه می‌کنیم تا اگر s هم آخر http وارد شده بود رفتارش را تغییر ندهد و یا به اشتباه نیفتد. به همین ترتیب برای اینکه www را هم با برنامه سازگار کنیم ?.www۱ را به کد اضافه می‌کنیم. و در آخر محض احتیاط، برای جلوگیری از اینکه کاربر تصمیم بگیرد کاملا بر خلاف پروتکل عمل کند، http:// یا https:// را هم با استفاده از (https?://) به صورت گزینه‌ی دلخواه به برنامه معرفی می‌کنیم تا درصورت برخورد با آن ها، برای هندل کردن هر دو آماده باشد.

- در عین حال ما انتظار داریم کاربر یک username وارد کند، یعنی در واقع لینکی را به عنوان ورودی به ما بدهد که دارای یک username هست. به همین دلیل با استفاده از دانشی که از `re.search` داریم می‌توانیم کد خود را باز هم ارتقا بدهیم.

- در نهایت کد ما به شکل زیر در می‌آید:

twitter.py

```
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))
```

- دقت کنید که ما داریم در رشته ای که کاربر برای ما فراهم کرده، دنبال الگویی که در کد بالا مشاهده می‌کنیم می‌گردیم. در واقع داریم با استفاده از الگوی $(.+)$، چیزی که در انتهای URL نمایان می‌شود را بدست می‌آوریم. درنتیجه اگر کاربر عبارتی که username ندارد را وارد کند، در انتها هم چیزی به عنوان خروجی نمایش داده نمی‌شود.

- در آخر می‌توانیم با استفاده از عملگر := کد خود را خلاصه تر کنیم:

twitter.py

```
import re

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

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

- دقت کنید که ?: به مفسر می‌گوید که در گرفتن چیزی که در جای مشخصی از عبارت ما هست، اجباری وجود ندارد.
- ما همچنان می‌توانیم برای بهینه تر بودن، از درستی username وارد شده توسط کاربر مطمئن شویم.

- با استفاده از مستندات توییتر، می‌توانیم عبارت منظم خود را تکمیل کنیم:

twitter.py

```
import re

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

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

- دقت کنید که `[a-z0-9_]+` به کامپایلر می‌گوید که فقط حروف a تا z و اعداد 1 تا 9 و ـ را به عنوان بخشی از عبارت منظم ما بپذیرد.
- می‌توانید در این لینک از اسناد پایتون، بیشتر راجع به کتابخانه‌ی `re` بخوانید.

در این هفته مباحث زیر را مورد بررسی قرار دادیم:

- Regular Expressions ✓
- Case Sensitivity ✓
- Cleaning Up User Input ✓
- Extracting User Input ✓

حالا یک توانایی به لیست رو به افزایش توانایی های شما در زبان پایتون اضافه شد. این رشد ادامه داره...

CS50x Iran

Harvard's Computer Science 50x Iran