Python: Регулярні вирази

Прикладна аналітика при розробці IT

Ігор Мірошниченко
КНУ імені Тараса Шевченка, ФІТ

ЗМІСТ

- Регулярні вирази

РЕГУЛЯРНІ ВИРАЗИ

РЕГУЛЯРНІ ВИРАЗИ

Регулярні вирази (англ. *regular expressions*, *regexes*) - це патерни, які використовуються для знаходження певних комбінацій символів у тексті.

Наприклад, якщо користувач вводить адресу електронної пошти у вашій програмі, на веб-сайті або в додатку на телефоні, в ідеалі ви хочете мати можливість перевірити, що він дійсно ввів адресу електронної пошти, а не щось інше.

Використовуючи регулярні вирази, ми можемо перевірити, чи = введений текст певному формату. Це дуже корисно, оскільки ви можете використовувати регулярні вирази для перевірки введення користувача, або для пошуку певних комбінацій символів у тексті.

Вони дуже потужні, але в той же час достатньо складні. Якщо ви вперше з ними зіткнулися, то вони здаються дуже незрозумілими. Але якщо ви вже з ними працювали, то ви не зможете без них.

ЗВИЧАЙНИЙ ПОШУК

Для початку розглянемо декілька задач і спробуємо їх вирішити використовуючи більш простий синтаксис, і подивитися, з якими обмеженнями ми зіткнемося.

Створимо файл `validate.py` метою якого є перевірка адреси електронної пошти користувача.

Terminal

```
code validate.py
```

Напишемо просту програму, яка буде приймати від користувача адресу електронної пошти і перевіряти, чи вона = певному формату.

Використаємо метод `strip()` для видалення зайвих пробілів з початку і кінця рядка. Але як перевірити, що введене значення дійсно є валідним для електронної пошти?

Перша ідея, яка з'являється - це перевірити, чи введений текст містить символ `@`. Якщо так, то ми можемо припустити, що це адреса електронної пошти:

```
email = input("Введіть email: ").strip()

if "@" in email:
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: potter@hogwarts.edu
Валідна адреса електронної пошти
```

Зрозуміло, що така програма не є ідеальною, оскільки ми можемо ввести тільки один або декілька символів `@` і програма буде пропускати такі випадки.

ЗВИЧАЙНИЙ ПОШУК

Наступна ідея - це додати додаткову перевірку, що введений текст містить символ “крапки” (.).

Якщо так, то ми можемо припустити, що це адреса електронної пошти. Давайте спробуємо це зробити:

```
1  email = input("Введіть email: ").strip()
2
3  if "@" in email and "." in email:
4      print("Валідна адреса електронної пошти")
5  else:
6      print("Невалідна адреса електронної пошти")
```

Terminal

```
1  python validate.py
2  Введіть email: potter@hogwarts.edu
3  Валідна адреса електронної пошти
```

ЗВИЧАЙНИЙ ПОШУК

Програма все ще не ідеальна.

Давайте припустимо, що крапка може бути тільки після символа @.

Для цього ми можемо поділити введений рядок на дві частини, використовуючи метод `split()` і перевірити, що друга частина містить крапку:

```
1  email = input("Введіть email: ").strip()
2
3  username, domain = email.split("@")
4
5  if username and "." in domain:
6      print("Валідна адреса електронної пошти")
7  else:
8      print("Невалідна адреса електронної пошти")
```

Terminal

```
1  python validate.py
2  Введіть email: potter@hogwarts.edu
3  Валідна адреса електронної пошти
```

МЕТОД `endswith()`

Тепер давайте звузимо сферу застосування цієї програми і будемо шукати валідні адреси навчальних закладів, які мають домен `.edu`.

Для цього ми можемо використати метод `endswith()`:

```
1  email = input("Введіть email: ").strip()
2
3  username, domain = email.split("@")
4
5  if username and domain.endswith(".edu"):
6      print("Валідна адреса електронної пошти")
7  else:
8      print("Невалідна адреса електронної пошти")
```

Terminal

```
1  python validate.py
2  Введіть email: potter@hogwarts.edu
3  Валідна адреса електронної пошти
4
5  python validate.py
6  Введіть email: potter@.hogwarts.edu
7  Валідна адреса електронної пошти
```

Ми можемо продовжувати роботу над цією програмою. В кінцевому підсумку нам доведеться писати багато коду, просто щоб валідувати адресу електронної пошти.

БІБЛІОТЕКА `re`

В Python є бібліотека для регулярних виразів, яка лаконічно називається `re`. В ній є багато можливостей для визначення, перевірки і заміни шаблонів.

> **Примітка**
>
> Документація до бібліотеки `re` доступна за посиланням https://docs.python.org/3/library/re.html.

Почнемо з однієї з найбільш універсальних функцій пошуку:

`re.search(pattern, string, flags=0)`:

- `pattern` - регулярний вираз, який ми шукаємо
- `string` - рядок, в якому ми шукаємо
- `flags` - додаткові флаги, які визначають поведінку функції

Ця функція повертає об’єкт, який містить інформацію про знайдений шаблон. Якщо шаблон не знайдено, то функція повертає None.

БІБЛІОТЕКА `re`

Давайте спробуємо переписати нашу програму використовуючи функцію `re.search()`:

```python
import re

email = input("Введіть email: ").strip()

if re.search("@", email):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: potter@hogwarts.edu
Валідна адреса електронної пошти

python validate.py
Введіть email: @
Валідна адреса електронної пошти
```

Цей варіант програми працює так само, як і наш перший варіант, але вже з використанням пакету `re`.

РЕГУЛЯРНІ ВИРАЗИ

Нам треба уточнити потер пошуку: ліворуч від символа `@` може бути певний запис, праворуч від символа `@` має бути також якийсь запис, який закінчується на `.edu`. Для цього існує ряд спеціальних символів, які дозволяють визначити певні шаблони:

Регулярний вираз	Опис	Приклади
`.`	Будь-який символ	`a.b` = “acb”, “a1b”, “a#b”
`*`	0 або більше повторень попереднього символу	`ab*c` = “ac”, “abc”, “abbc”
`+`	1 або більше повторень попереднього символу	`ab+c` = “abc”, “abbc”, але не “ac”
`?`	0 або 1 повторення попереднього символу	`colou?r` = “color” і “colour”
`{n}`	Рівно `n` повторень попереднього символу	`a{3}b` = “aaab”
`{n, m}`	Від `n` до `m` повторень попереднього символу	`a{2,4}b` = “aab”, “aaab” і “aaaab”
`{n,}`	Від `n` повторень попереднього символу	`a{2,}b` = “aab”, “aaab”, “aaaab” і так далі
`^`	Початок рядка	`^start` =, якщо рядок починається з “start”
`$`	Кінець рядка	`end$` =, якщо рядок закінчується на “end”
`[]`	Набір символів	`[aeiou]` = будь-якому голосному символу
`[^]`	Набір символів, які не повинні зустрічатися	`[^0-9]` = будь-якому символу, крім цифр
`A|B`	Або	`cat|dog` = “cat” або “dog”
`(...)`	Група символів	`(ab)+` = “ab”, “abab”, “ababab” і так далі
`?:...`	Не захоплювати групу	`(?:ab)+` = “ab”, “abab”, “ababab” і так далі

РЕГУЛЯРНІ ВИРАЗИ

Давайте спробуємо переписати нашу програму використовуючи функцію `re.search()` і регулярний вираз:

```
1  import re
2
3  email = input("Введіть email: ").strip()
4
5  if re.search(".*@.*", email):
6      print("Валідна адреса електронної пошти")
7  else:
8      print("Невалідна адреса електронної пошти")
```

Комп'ютер використовує свого роду машину, реалізовану в програмному забезпеченні, відомому як **скінченний автомат** (англ. *finite state machine*) або **недетермінований скінченний автомат**.

Візуально це можна зобразити так:

Вираз: `.*@.*`

РЕГУЛЯРНІ ВИРАЗИ

Ми можемо переписати нашу програму з використанням регулярного виразу `.+`, який означає “один або більше будь-яких символів”:

```
1  import re
2
3  email = input("Введіть email: ").strip()
4
5  if re.search(".+@.+", email):
6      print("Валідна адреса електронної пошти")
7  else:
8      print("Невалідна адреса електронної пошти")
```

В такому випадку візуалізація недетермінованого скінченного автомату буде наступна:

Вираз: `.+@.+`

РЕГУЛЯРНІ ВИРАЗИ

Підемо далі і додамо перевірку, що домен закінчується на `.edu`:

```python
import re

email = input("Введіть email: ").strip()

if re.search(r".+@.+\.edu", email):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: potter@hogwarts.edu
Валідна адреса електронної пошти

python validate.py
Введіть email: potter?hogwarts.edu
Невалідна адреса електронної пошти
```

Попередження

Зверніть увагу, що ми використовуємо символ `\` для екранування крапки, оскільки в іншому випадку крапка буде сприйматися як будь-який символ.

Крім того, слід враховувати, що комбінація символів `\n` вважається спеціальним символом, який позначає перехід на новий рядок. Тому ми вказуємо Python читати рядок як **“сирий”** (англ. *raw*), використовуючи префікс `r` перед рядком.

Візуалізація:

Вираз: `.+@.+\.edu`

ПОЧАТОК ТА КІНЕЦЬ РЯДКА

Наша програма все ще має ряд недоліків. Наприклад, вона не враховує пробіли, які можуть зустрічатися у введеному тексті:

Terminal

```
python validate.py
Введіть email: Моя пошта potter@hogwarts.edu
Валідна адреса електронної пошти
```

Для таких випадків у світі регулярних виразів існують спеціальні символи початку ^ та кінця рядка $:

```
import re

email = input("Введіть email: ").strip()

if re.search(r"^.+@.+\.edu$", email):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: Моя пошта potter@hogwarts.edu
Невалідна адреса електронної пошти
```

Візуалізація:

Вираз: `^.+@.+\.edu$`

ПОШУК ТА ВИЛУЧЕННЯ

Є ще один значний недолік нашої програми. Вона не враховує умову одного символу @ у адресі електронної пошти:

Terminal

```
python validate.py
Введіть email: potter@@@hogwarts.edu
Валідна адреса електронної пошти
```

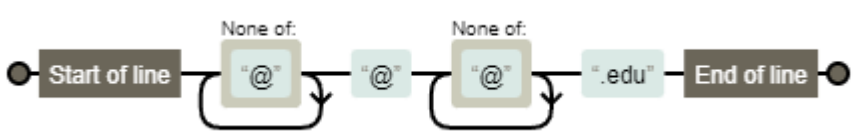
Для цього ми можемо використати символи [ ] для визначення набору символів, які можуть зустрічатися у тексті та [^] для символів, які не повинні зустрічатися у тексті. Оскільки ми не хочемо щоб символ @ зустрічався більше одного разу, то ми можемо використати наступний регулярний вираз [^@]:

```python
import re

email = input("Введіть email: ").strip()

if re.search(r"^[^@]+@[^@]+\.edu$", email):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: potter@@@hogwarts.edu
Невалідна адреса електронної пошти
```

Візуалізація:



Вираз: ^[^@]+@[^@]+\.edu$

СЛОВА ТА ЦИФРИ

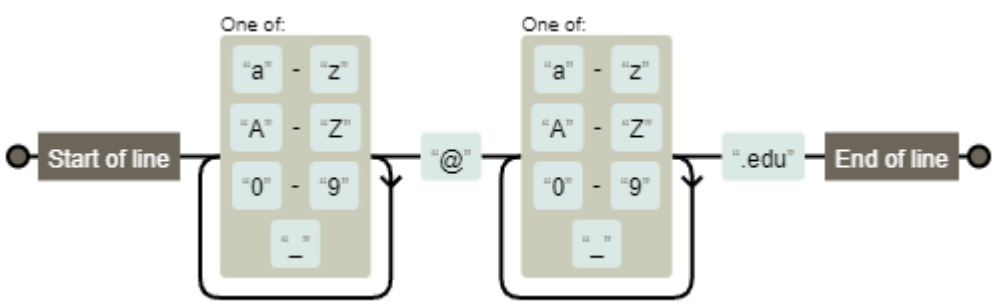
У світі стандартів існує багато різних специфікацій для адрес електронної пошти. Наприклад, вони повинні починатися з літери. Нам не потрібно перераховувати всі можливі літери алфавіту у нижньому та верхньому регістрі. Замість цього ми можемо використати спеціальний символ `[a-zA-Z]`, який означає будь-яку літеру англійського алфавіту незалежно від регістру. Якщо ж я хочу також врахувати цифри та символ _, то я можу використати спеціальний символ `[a-zA-Z0-9_]`. Давайте спробуємо використати цей регулярний вираз:

```python
import re

email = input("Введіть email: ").strip()

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

Terminal

```
python validate.py
Введіть email: potter@@@hogwarts.edu
Невалідна адреса електронної пошти
```

Візуалізація:



Вираз: `^[a-zA-Z0-9_]+@[a-zA-Z0-9_]+\.edu$`

СЛОВА ТА ЦИФРИ

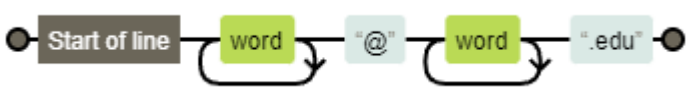
Такі патерни досить часто зустрічаються, тому у регулярних виразах є готові спеціальні символи, які дозволяють замінити їх.

Наприклад, спеціальний метасимвол `\w` означає **будь-яку літеру англійського алфавіт**у незалежно від регістру, цифру або символ `_`.

Також є спеціальний метасимвол `\d`, який означає **будь-яку цифру**.

```python
import re

email = input("Введіть email: ").strip()

if re.search(r"^\w+@\w+\.edu$", email):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Візуалізація:



Вираз: `^\w+@\w+\.edu$`

СЛОВА ТА ЦИФРИ

Повний перелік спеціальних метасимволів:

Метасимвол	Опис
\w	Будь-яка літера англійського алфавіту незалежно від регістру, цифра або символ _
\d	Будь-яка цифра
\s	Будь-який пробіл
\W	Будь-який символ, крім літер англійського алфавіту незалежно від регістру, цифр та символу _
\D	Будь-який символ, крім цифри
\S	Будь-який символ, крім пробілу

ФЛАГИ

Давайте припустимо, що при введені адреси електронної пошти користувач випадково натиснув клавішу Caps Lock і ввів адресу з використанням великих літер.

Якщо ми використаємо наш регулярний вираз, то програма не буде вважати таку адресу валідною:

Terminal

```
python validate.py
Введіть email: POTTER@HOGWARTS.EDU
Невалідна адреса електронної пошти
```

Вирішити таку задачу можна декількома шляхами.

Наприклад, ми можемо використати функцію `lower()` для перетворення всіх символів у нижній регістр. Це можна зробити на початку програми або використовуючи метод `lower()` для об'єкта `email`:

```
import re

email = input("Введіть email: ").strip().lower()

if re.search(r"^\w+@\w+\.edu$", email): # або email.lower()
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

ФЛАГИ

Але є інший спосіб. Ми можемо використати аргумент `flag` для функції `re.search()`, який може приймати наступні значення:

- `re.IGNORECASE` або `re.I` - ігнорувати регістр символів
- `re.MULTILINE` або `re.M` - використовувати багаторядковий режим
- `re.DOTALL` або `re.S` - використовувати режим, в якому крапка `.` відповідає будь-якому символу, включаючи символ нового рядка `\n`

Тож для нашої задачі код буде виглядати наступним чином:

```python
import re

email = input("Введіть email: ").strip()

if re.search(r"^\w+@\w+\.edu$", email, re.IGNORECASE):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: POTTER@HOGWARTS.EDU
Валідна адреса електронної пошти
```

ОПЦІОНАЛЬНІСТЬ

Припустимо, що адреса користувача містить піддомен `gryff.hogwarts.edu`. В такому випадку нам слід врахувати варіативність піддоменів. Для цього ми можемо використати спеціальний символ `?`, який означає 0 або 1 повторення попереднього символу. Давайте спробуємо використати цей символ:

```python
import re

email = input("Введіть email: ").strip()

if re.search(r"^\w+@(\w+\.)?edu$", email, re.IGNORECASE):
    print("Валідна адреса електронної пошти")
else:
    print("Невалідна адреса електронної пошти")
```

Terminal

```
python validate.py
Введіть email: potter@hogwarts.edu
Валідна адреса електронної пошти

python validate.py
Введіть email: potter@gryff.hogwarts.edu
Валідна адреса електронної пошти
```

Візуалізація:

Вираз: `^\w+@(\w+\.)?edu$`

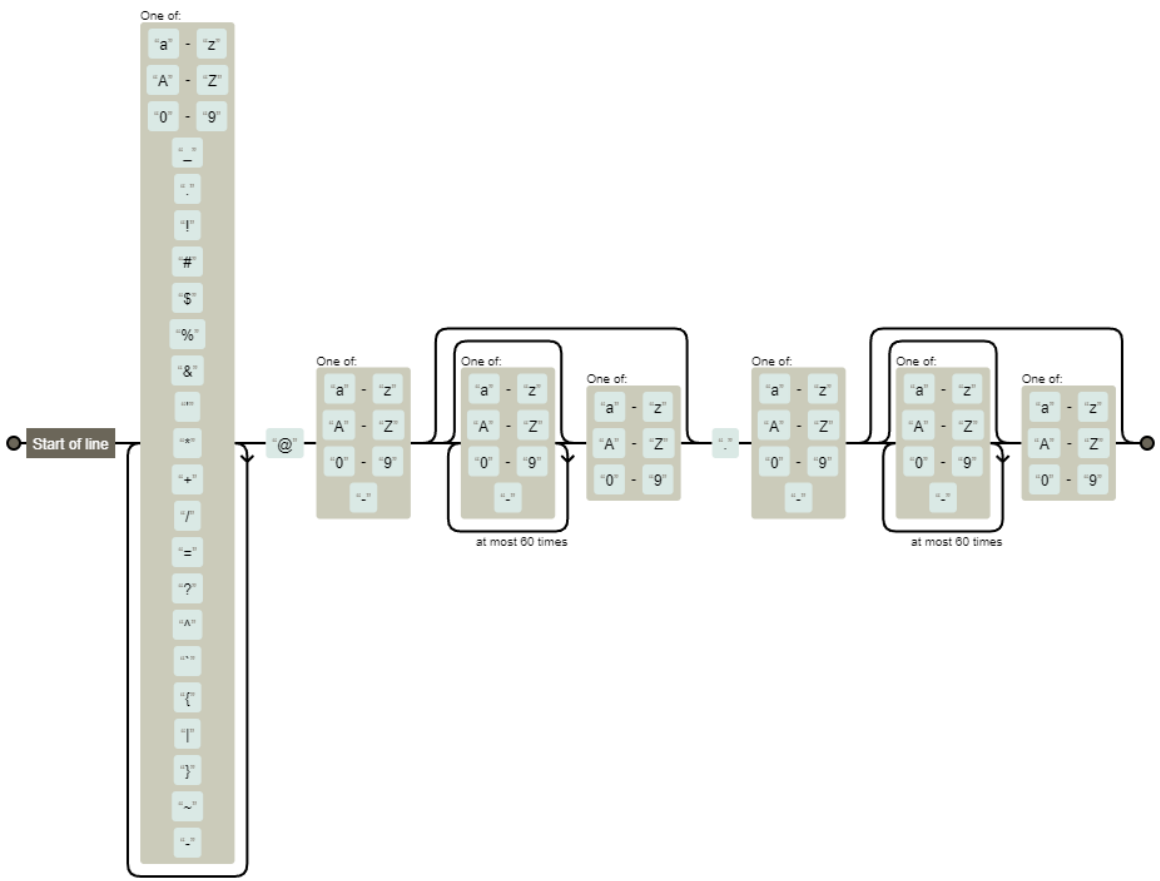
Запис `(\w+\.)?` означає, що група записів `\w+\.` може зустрічатися 0 або 1 раз. Таким чином, ми можемо врахувати варіативність піддоменів.

ПРИКЛАДИ З РЕАЛЬНОГО СВІТУ

Все що ми зробили до цього часу для відстеження адрес електронної пошти, все ще має ряд недоліків. Я наведу приклад регулярного виразу, який використовується в реальному світі для відстеження адрес електронної пошти:

```
^[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])?)
```

Візуально такий регулярний вираз виглядає так:

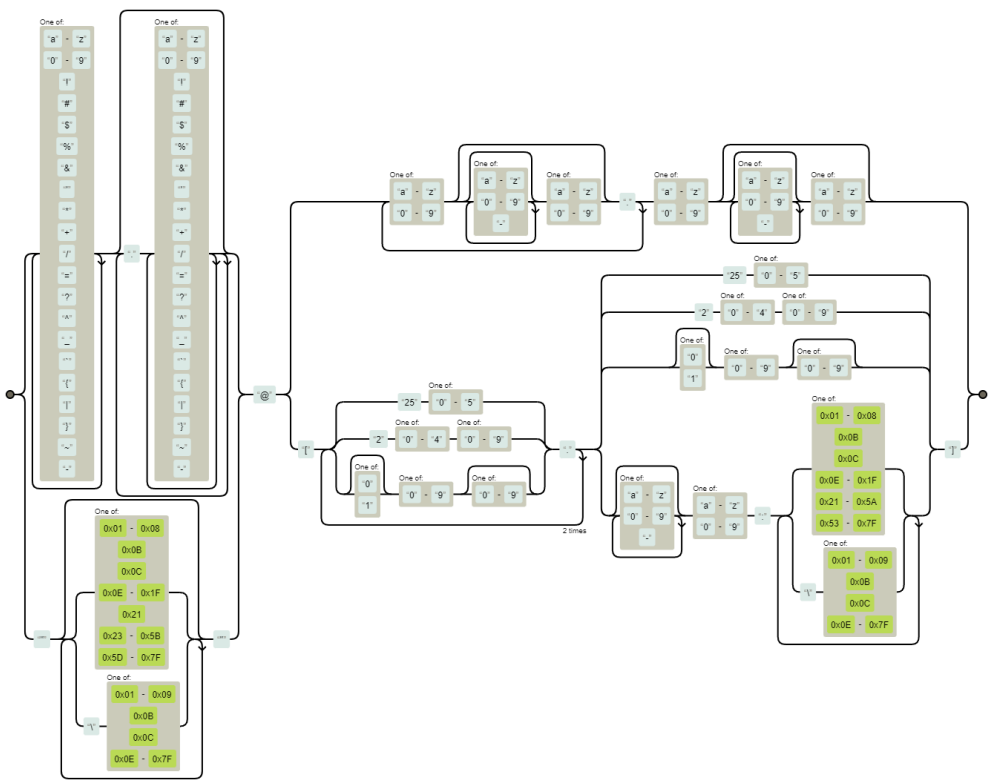


ПРИКЛАДИ З РЕАЛЬНОГО СВІТУ

В інтернеті є ряд стандартів, які визначають, якими мають бути адреси електронної пошти. Один з них - це RFC 5322. Згідно цього стандарту 99.99% адрес електронної пошти можна відстежити за допомогою наступного регулярного виразу:

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-
\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:
(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-
\x7f])+)\])
```

Візуально такий запис виглядає наступним чином:



Валідація email за RFC 5322

ОЧИЩЕННЯ ТЕКСТУ

Замість того, щоб просто перевіряти вхідні дані користувача і переконуватися, що вони виглядають так, як ми хочемо, давайте просто припустимо, що користувачі не збираються вводити дані саме так, як ми хочемо. Тому нам доведеться очистити їхні дані.

Створимо програму `format.py`, яка очищуватиме введене ім'я користувача:

```
Terminal
code format.py
```

Деякі користувачі можуть мати звичку вводити спочатку своє ім'я, а потім прізвище через кому: `Гаррі, Поттер` замість `Поттер, Гаррі`. Це нормально, тому що обидва варіанти однаково добре читаються людиною. Але для комп'ютера це різні рядки. Тому нам потрібно буде очистити введені дані користувача. Створимо змінну `name`, яка буде містити введені дані користувача:

```
name = input("Введіть ім'я: ").strip()
print(f'Привіт, {name}!')
```

Подивимось, як працює програма з різними введеними даними:

```
Terminal
python format.py
Введіть ім'я: Гаррі Поттер
Привіт, Гаррі Поттер!
```

```
Terminal
python format.py
Введіть ім'я: Поттер, Гаррі
Привіт, Поттер, Гаррі!
```

ОЧИЩЕННЯ ТЕКСТУ

Перший варіант нас може задовільнити, а от другий виглядає зовсім не так, як ми очікуємо. Давайте спробуємо виправити цю ситуацію через умовний оператор `if`:

```
1  name = input("Введіть ім'я: ").strip()
2
3  if ',' in name:
4      last, first = name.split(', ')
5      name = f'{first} {last}'
6
7  print(f'Привіт, {name}!')
```

Terminal

```
1  python format.py
2  Введіть ім'я: Поттер, Гаррі
3  Привіт, Гаррі Поттер!
```

Ми виправили ситуацію, але що якщо користувач введе ім’я без використання пробілів? Наприклад, `Поттер,Гаррі`?

Це призведе до помилки, оскільки ми використовуємо метод `split()` для розділення рядка на дві частини.

Для цього нам знадобиться функція `re.search()`, яка повертає об’єкт `Match`. Якщо відповідність знайдена, то ми можемо використати метод `groups()` для отримання відповідного значення, які були записані у дужки.

ОЧИЩЕННЯ ТЕКСТУ

Давайте спробуємо використати регулярний вираз `^(.+), (.+)$`, який означає *“початок рядка, один або більше будь-яких символів, кома, пробіл, один або більше будь-яких символів, кінець рядка”*:

```
1  import re
2
3  name = input("Введіть ім'я: ").strip()
4
5  matches = re.search(r'^(.+), (.+)$', name)
6
7  if matches:
8      last, first = matches.groups()
9      name = f'{first} {last}'
10
11 print(f'Привіт, {name}!')
```

Якщо ми введемо ім’я користувача без коми, то з умовним оператором `if` нічого не буде відбуватися, оскільки `matches` буде `None` і програма одразу перейде до виведення привітання:

Terminal

```
1  python format.py
2  Введіть ім'я: Гаррі Поттер
3  Привіт, Гаррі Поттер!
```

У випадку, коли користувач введе ім’я з комою, то відбудеться відповідність регулярному виразу і ми зможемо використати метод `groups()` для отримання відповідних значень:

Terminal

```
1  python format.py
2  Введіть ім'я: Поттер, Гаррі
3  Привіт, Гаррі Поттер!
```

ОЧИЩЕННЯ ТЕКСТУ

Якщо ж потрібно повернути конкретні групи замість всіх, то можна використати метод `group()`, який приймає номер групи:

```
1  import re
2
3  name = input("Введіть ім'я: ").strip()
4
5  matches = re.search(r'^(.+), (.+)$', name)
6
7  if matches:
8      name = f'{matches.group(2)} {matches.group(1)}'
9
10 print(f'Привіт, {name}!')
```

Ця програма має значний недолік - ми очікуємо, що ім'я та прізвище користувача будуть розділені одним пробілом. Якщо користувач введе ім'я без пробілу або навпаки використає декілька, то програма виведе не те, що ми очікуємо:

Terminal

```
1  python format.py
2  Введіть ім'я: Поттер,Гаррі
3  Привіт, Поттер,Гаррі!
```

ОЧИЩЕННЯ ТЕКСТУ

Перший варіант, як врахувати таку ситуацію - це додати `*` до регулярного виразу:`^(.+), *(.+)?$`.

Це буде означати *“початок рядка, один або більше будь-яких символів, кома, нуль або більше пробілів, один або більше будь-яких символів, кінець рядка”*:

```python
import re

name = input("Введіть ім'я: ").strip()

matches = re.search(r'^(.+), *(.+)?$', name)

if matches:
    name = f'{matches.group(2)} {matches.group(1)}'

print(f'Привіт, {name}!')
```

Terminal

```
python format.py
Введіть ім'я: Поттер,Гаррі
Привіт, Гаррі Поттер!

python format.py
Введіть ім'я: Поттер,      Гаррі
Привіт, Гаррі Поттер!
```

Попередження

Залежно від того, наскільки безладними є дані, ваші регулярні вирази можуть ставати все складнішими і складнішими. Кількість умов напряму впливає на їх складність.

МОРЖЕВИЙ ОПЕРАТОР

Ми можемо скоротити наш код, використовуючи “моржевий оператор” `:=` (англ. *walrus operator*), який дозволяє присвоювати значення змінній та одночасно використовувати її у виразі:

```
1  import re
2
3  name = input("Введіть ім'я: ").strip()
4
5  if matches := re.search(r'^(.+), *(.+)?$', name):
6      name = f'{matches.group(2)} {matches.group(1)}'
7
8  print(f'Привіт, {name}!')
```

ОТРИМАННЯ ДАНИХ З РЯДКА

Розглянемо варіант програми, де нам потрібно отримати дані з рядка, щоб відповісти на якесь питання. Наприклад, давайте створимо програму `twitter.py`, яка буде запитувати у користувачів URL-адресу їхнього профілю в Twitter і витягти з неї ім'я користувача:

Terminal

```
code twitter.py
```

Розглянемо рядок `https://twitter.com/harrypotter`. Перше на що звертаємо увагу - це те, адреса сервісу завжди починається з `https://twitter.com/`. Тому ми можемо замінити цю частину рядка на порожній рядок. Для цього ми можемо використати метод `replace()`:

```
url = 'https://twitter.com/harrypotter'

username = url.replace('https://twitter.com/', '')
print(f"Ім'я користувача: {username}")
```

```
Ім'я користувача: harrypotter
```

ОТРИМАННЯ ДАНИХ З РЯДКА

Але така проста програма не враховує низку ситуацій:

- адреса може починатися без `https://`
- адреса може починатися з `http://`
- адреса може містити `www.`
- тощо

Використаємо бібліотеку функцію:

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

яка замінює всі входження патерну `pattern` на `repl` у рядку `string`.

Використаємо регулярний вираз `^(https?://)?(www\.)?twitter\.com/`, який читається як: *“початок рядка, група символів `https://` або `http://` 0 або 1 раз, група символів `www.` 0 або 1 раз, `twitter.com/`”*:

```
1  import re
2
3  url = 'https://twitter.com/harrypotter'
4
5  username = re.sub(r'^(https?://)?(www\.)?twitter\.com/', '', url)
6
7  print(f"Ім'я користувача: {username}")
```

```
Ім'я користувача: harrypotter
```

ОТРИМАННЯ ДАНИХ З РЯДКА

Проте, якщо користувач введе замість URL-адреси Twitter адресу іншого сервісу, регулярний вираз працювати не буде:

```
1  import re
2
3  url = 'https://facebook.com/harrypotter'
4
5  username = re.sub(r'^(https?://)?(www\.)?twitter\.com/', '', url)
6
7  print(f"Ім'я користувача: {username}")
```

```
Ім'я користувача: https://facebook.com/harrypotter
```

ОТРИМАННЯ ДАНИХ З РЯДКА

В такому випадку ми можемо повернутися до `re.search()` та використати групи для отримання імені користувача, але для виключення певної групи з переліку можна використати `?:`:

```
1  import re
2
3  url = 'https://facebook.com/harrypotter'
4
5  if matches := re.search(r'^https?://(?:www\.)?twitter\.com/(.+)$', url, re.IGNORECASE):
6      username = matches.group(1)
7      print(f"Ім'я користувача: {username}")
8  else:
9      print('Невідомий сервіс')
```

```
Невідомий сервіс
```

Тепер програма буде працювати з будь-якими URL-адресами:

```
1  import re
2
3  url = 'https://www.twitter.com/harrypotter'
4
5  if matches := re.search(r'^https?://(?:www\.)?twitter\.com/(.+)$', url, re.IGNORECASE):
6      username = matches.group(1)
7      print(f"Ім'я користувача: {username}")
8  else:
9      print('Невідомий сервіс')
```

```
Ім'я користувача: harrypotter
```

ОТРИМАННЯ ДАНИХ З РЯДКА

Якщо ж ознайомитися з документацією Twitter, то можна знайти, що ім'я користувача може містити тільки літери, цифри, символ _, тобто не просто `.+`, що може бути чим завгодно. Тому ми можемо використати регулярний вираз `[a-z0-9_]+` для більш точного пошуку імені користувача, який читається як: *"будь-яка літера англійського алфавіту, цифра або символ _ 1 або більше разів"*:

```
1  import re
2
3  url = 'https://www.twitter.com/harrypotter'
4
5  if matches := re.search(r'^https?://(?:www\.)?twitter\.com/([a-z0-9_]+)$', url, re.IGNORECASE):
6      username = matches.group(1)
7      print(f"Ім'я користувача: {username}")
8  else:
9      print('Невідомий сервіс')
```

```
Ім'я користувача: harrypotter
```

В пакеті `re` є ряд цікавих функцій:

- `re.split(pattern, string, maxsplit=0, flags=0)` - розбиває рядок на частини за допомогою регулярного виразу
- `re.findall(pattern, string, flags=0)` - повертає список всіх знайдених входжень регулярного виразу

Порада

Для побудови, тестування та налагодження регулярних виразів можна використовувати regex101.com.

ДЯКУЮ ЗА УВАГУ!

- Матеріали курсу
- ihor.miroshnychenko@kneu.ua
- Data Mirosh
- @ihormiroshnychenko
- @aranaur
- aranaur.rbind.io