Регулярні вирази (Regex) у Python: Теорія та синтаксис

1.1. Що таке Регулярні вирази (Regex)?

Регулярні вирази (Regular Expressions, Regex або Regexp) — це потужний інструмент для пошуку, заміни та маніпуляції текстовими рядками на основі шаблонів. Вони надають гнучкий і ефективний спосіб опису складних текстових патернів.

Для чого потрібні регулярні вирази?

- **Валідація даних:** Перевірка, чи відповідає введений текст певному формату (наприклад, електронна пошта, номер телефону, дата).
- **Пошук та вилучення:** Знаходження конкретних фрагментів тексту, що відповідають певному шаблону (наприклад, всі URL-адреси на веб-сторінці, всі числа).
- **Заміна тексту:** Заміна частин рядка, що відповідають шаблону, на інший текст.
- **Парсинг логів:** Аналіз лог-файлів для вилучення інформації про помилки, події тощо.
- **Очищення даних:** Видалення небажаних символів або форматування тексту.

1.2. Основний синтаксис Regular Expressions

Регулярні вирази складаються зі звичайних символів (літералів) та спеціальних символів (метасимволів), які мають особливе значення.

1. Літерали

Звичайні символи (літери, цифри, більшість знаків пунктуації) відповідають самі собі.

- abc — шукає точну послідовність "abc".
- 123 — шукає точну послідовність "123".

2. Метасимволи (Special Characters)

Це символи, які мають особливе значення в регулярних виразах. Щоб знайти їх як звичайні символи, їх потрібно "екранувати" за допомогою зворотного слеша (\).

- . (крапка): Відповідає будь-якому одному символу, крім нового рядка (за замовчуванням).
  - a.c — знайде "abc", "axc", "a1c" тощо.
- ^ (карет): Відповідає початку рядка.

    - ^Hello — знайде "Hello" тільки на початку рядка.
- $ (знак долара): Відповідає кінцю рядка.
    - World$ — знайде "World" тільки в кінці рядка.
- * (зірочка): Відповідає нулю або більше повторень попереднього символу/групи.
    - ab*c — знайде "ac", "abc", "abbc", "abbbc" тощо.
- + (плюс): Відповідає одному або більше повторень попереднього символу/групи.
    - ab+c — знайде "abc", "abbc", "abbbc" (але не "ac").
- ? (знак питання): Відповідає нулю або одному повторенню попереднього символу/групи (робить його необов'язковим).
    - colou?r — знайде "color" або "colour".
- {n}: Відповідає рівно n повторенням попереднього символу/групи.
    - a{3} — знайде "aaa".
- {n,}: Відповідає n або більше повторень.
    - a{2,} — знайде "aa", "aaa", "aaaa" тощо.
- {n,m}: Відповідає від n до m повторень (включно).
    - a{2,4} — знайде "aa", "aaa", "aaaa".
- [] (квадратні дужки): Визначає **набір символів** (character set). Відповідає будь-якому одному символу з цього набору.
    - [abc] — знайде "a", "b" або "c".
    - [a-z] — знайде будь-яку малу літеру від 'a' до 'z'.
    - [0-9] — знайде будь-яку цифру.
    - [^abc] — знайде будь-який символ, крім "a", "b" або "c" (^ всередині [] означає заперечення).
- \ (зворотний слеш): Використовується для екранування метасимволів (щоб вони відповідали самі собі) або для спеціальних послідовностей символів.
    - \. — знайде крапку.
    - \? — знайде знак питання.
- | (вертикальна риска): АБО (логічне "або"). Відповідає одному з кількох виразів.
    - cat|dog — знайде "cat" або "dog".
- () (круглі дужки): Визначають **групування** (capturing group). Дозволяють застосовувати квантифікатори до кількох символів або вилучати частини збігу.
    - (ab)+ — знайде "ab", "abab", "ababab" тощо.
    - (\d{3})-\d{2}-\d{2} — дозволяє вилучити перші три цифри як окрему групу.

3. Спеціальні послідовності символів (Character Classes)

Це скорочення для часто використовуваних наборів символів.

- \d: Відповідає будь-якій цифрі (еквівалентно [0-9]).
- \D: Відповідає будь-якому символу, що не є цифрою (еквівалентно [^0-9]).
- \w: Відповідає будь-якій букві, цифрі або символу підкреслення (word character) (еквівалентно [a-zA-Z0-9_]).
- \W: Відповідає будь-якому символу, що не є буквою, цифрою або підкресленням.
- \s: Відповідає будь-якому пробільному символу (пробіл, табуляція, новий рядок тощо).
- \S: Відповідає будь-якому символу, що не є пробільним.
- \b: Відповідає межі слова (word boundary).
    - \bcat\b — знайде "cat" як ціле слово, але не "catapult".
- \B: Відповідає не межі слова.

4. Квантифікатори (Quantifiers)

Визначають, скільки разів попередній елемент може повторюватися.

- * (0 або більше)
- + (1 або більше)
- ? (0 або 1)
- {n} (рівно n разів)
- {n,} (n або більше разів)
- {n,m} (від n до m разів)

За замовчуванням квантифікатори є "жадібними" (greedy) — вони намагаються збігтися з якомога більшою кількістю символів. Щоб зробити їх "лінивими" (non-greedy), додайте ? після квантифікатора:

- *? (0 або більше, лінивий)
- +? (1 або більше, лінивий)
- ?? (0 або 1, лінивий)
- {n,}? (n або більше, лінивий)

5. Групування та посилання (Groups and Backreferences)

- (...): **Захоплююча група (Capturing Group)**. Збіг з вмістом групи можна витягти окремо. Також дозволяє застосовувати квантифікатори до всього вмісту групи.
- (?:...): **Незахоплююча група (Non-capturing Group)**. Групує елементи, але не зберігає збіг для витягування. Корисно для застосування квантифікаторів без створення зайвих груп.
- (?P<name>...): **Іменована захоплююча група (Named Capturing Group)**.

Дозволяє звертатися до групи за іменем, а не за індексом.
- \1, \2, ...: **Зворотні посилання (Backreferences)**. Посилаються на текст, який був захоплений попередньою групою.
    - (\w+)\s+\1 — знайде повторюване слово, наприклад "word word".

1.3. Модуль re у Python

Python має вбудований модуль re для роботи з регулярними виразами.

Основні функції модуля re:

- re.search(pattern, string, flags=0):
    - Шукає **перший** збіг шаблону pattern у будь-якому місці рядка string.
    - Повертає об'єкт MatchObject, якщо знайдено збіг, і None, якщо не знайдено.
- re.match(pattern, string, flags=0):
    - Шукає збіг шаблону pattern **тільки на початку** рядка string.
    - Повертає об'єкт MatchObject, якщо знайдено збіг, і None, якщо не знайдено.
- re.findall(pattern, string, flags=0):
    - Знаходить **усі** непересічні збіги шаблону pattern у рядку string.
    - Повертає список рядків (якщо немає груп) або список кортежів (якщо є групи).
- re.finditer(pattern, string, flags=0):
    - Знаходить **усі** непересічні збіги шаблону pattern у рядку string.
    - Повертає ітератор об'єктів MatchObject. Корисно, коли потрібно отримати більше інформації про кожен збіг (позиції, групи).
- re.sub(pattern, repl, string, count=0, flags=0):
    - Замінює всі (або count перших) збіги шаблону pattern у рядку string на repl.
    - repl може бути рядком або функцією.
- re.split(pattern, string, maxsplit=0, flags=0):
    - Розбиває рядок string за збігами шаблону pattern.
    - Повертає список рядків.
- re.compile(pattern, flags=0):
    - Компілює регулярний вираз у об'єкт RegexObject.
    - Це корисно, якщо ви плануєте використовувати один і той самий шаблон багаторазово, оскільки компіляція шаблону прискорює подальші операції.

Об'єкт MatchObject

Якщо re.search() або re.match() знаходять збіг, вони повертають об'єкт

MatchObject. Основні методи цього об'єкта:

- match.group(0): Повертає весь збіг.
- match.group(1), match.group(2), ...: Повертає збіги для відповідних захоплюючих груп.
- match.group('name'): Повертає збіг для іменованої групи.
- match.groups(): Повертає кортеж збігів для всіх захоплюючих груп.
- match.start(): Повертає початковий індекс збігу.
- match.end(): Повертає кінцевий індекс збігу (ексклюзивно).
- match.span(): Повертає кортеж (start, end) індексів збігу.

Прапорці (Flags)

Прапорці змінюють поведінку регулярного виразу. Їх можна передавати як аргумент flags.

- re.IGNORECASE або re.I: Ігнорувати регістр (великі/малі літери).
- re.MULTILINE або re.M: ^ та $ відповідають початку/кінцю кожного рядка, а не всього рядка.
- re.DOTALL або re.S: Крапка (.) відповідає будь-якому символу, включаючи новий рядок.
- re.VERBOSE або re.X: Дозволяє додавати пробіли та коментарі до регулярного виразу для кращої читабельності.
- re.ASCII або re.A: Робить \w, \b, \s, \d відповідаючими лише ASCII символам.
- re.UNICODE або re.U: (За замовчуванням у Python 3) Робить \w, \b, \s, \d відповідаючими Unicode символам.

```
import re

print("--- Regex: Практичні Приклади ---")

# --- Приклад 1: re.search() ---
# Шукає перший збіг шаблону в рядку.

text = "Привіт, світ! Це тестовий рядок."
pattern = r"світ" # r"" означає "raw string" - сирий рядок, ігнорує екранування

print("\n--- re.search() ---")
match = re.search(pattern, text)
if match:
    print(f"Збіг знайдено: '{match.group(0)}'")
    print(f"Початок: {match.start()}, Кінець: {match.end()}")
    print(f"Діапазон: {match.span()}")
else:
    print("Збіг не знайдено.")

# Приклад з метасимволами
text2 = "Номер телефону: 123-456-7890 або 987-654-3210."
pattern2 = r"\d{3}-\d{3}-\d{4}" # Шукаємо шаблон XXX-XXX-XXXX
match2 = re.search(pattern2, text2)
if match2:
    print(f"Знайдено телефон: {match2.group(0)}")
else:
    print("Телефон не знайдено.")
```

```
# --- Приклад 2: re.match() ---
# Шукає збіг шаблону тільки на початку рядка.

print("\n--- re.match() ---")
text3 = "Python - чудова мова програмування."
pattern3 = r"Python"
match3 = re.match(pattern3, text3)
if match3:
    print(f"Збіг на початку: {match3.group(0)}")
else:
    print("Збіг на початку не знайдено.")

text4 = "Мова програмування: Python."
match4 = re.match(pattern3, text4) # pattern3 = r"Python"
if match4:
    print(f"Збіг на початку: {match4.group(0)}")
else:
    print("Збіг на початку не знайдено (очікувано, бо 'Python' не на початку).")

# --- Приклад 3: re.findall() ---
# Знаходить усі непересічні збіги шаблону.

print("\n--- re.findall() ---")
text5 = "Email: test@example.com, support@domain.org, info@mail.net."
pattern5 = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
emails = re.findall(pattern5, text5)
print(f"Знайдені email адреси: {emails}")
```

```
# Приклад з групами
text6 = "Дата: 2023-01-15, 2024-11-01, 2025-05-23."
pattern6 = r"(\d{4})-(\d{2})-(\d{2})" # Групи для року, місяця, дня
dates = re.findall(pattern6, text6)
print(f"Знайдені дати (групи): {dates}")
# Результат буде списком кортежів: [('2023', '01', '15'), ('2024', '11', '01'), ('2025', '05', '23')]

# --- Приклад 4: re.finditer() ---
# Повертає ітератор об'єктів MatchObject.

print("\n--- re.finditer() ---")
text7 = "Ціни: $10.50, $25.00, $5.99."
pattern7 = r"\$(\d+\.\d{2})"
for match_obj in re.finditer(pattern7, text7):
    print(f"Знайдена ціна: {match_obj.group(0)} (значення: {match_obj.group(1)})")
    print(f"  Початок: {match_obj.start()}, Кінець: {match_obj.end()}")

# --- Приклад 5: re.sub() ---
# Замінює збіги шаблону.

print("\n--- re.sub() ---")
text8 = "Я люблю програмувати на Python. Python - це круто!"
pattern8 = r"Python"
replacement8 = "Java"
new_text8 = re.sub(pattern8, replacement8, text8)
print(f"Оригінал: {text8}")
print(f"Замінено: {new_text8}")
```

```
# Заміна лише перших N збігів
text9 = "Ціна: $10.00. Знижка: $2.00. Фінал: $8.00."
pattern9 = r"\$"
replacement9 = "Євро"
new_text9 = re.sub(pattern9, replacement9, text9, count=2) # Замінити лише перші 2
print(f"Оригінал: {text9}")
print(f"Замінено (2 рази): {new_text9}")

# Заміна за допомогою функції
def censor_email(match):
    email = match.group(0)
    username, domain = email.split('@')
    return f"{username[0]}***@{domain}" # Залишаємо першу літеру юзернейму
text10 = "Мій email: user@example.com. Зв'яжіться зі мною."
pattern10 = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
censored_text = re.sub(pattern10, censor_email, text10)
print(f"Оригінал: {text10}")
print(f"Замінено (цензура email): {censored_text}")



# --- Приклад 6: re.split() ---
# Розбиває рядок за збігами шаблону.

print("\n--- re.split() ---")
text11 = "один, два; три. чотири"
pattern11 = r"[,;.]\s*" # Розділяємо за комою, крапкою з комою або крапкою, за якими йде пробіл
```

```
parts = re.split(pattern11, text11)
print(f"Оригінал: '{text11}'")
print(f"Розбито: {parts}")

# --- Приклад 7: re.compile() ---
# Компілює регулярний вираз для підвищення продуктивності.

print("\n--- re.compile() ---")
long_text = "Це довгий текст з багатьма словами. Слово, слово, слово. Шукаємо слово."
compiled_pattern = re.compile(r"\bслово\b", re.IGNORECASE) # Компілюємо шаблон

# Використовуємо скомпільований шаблон
found_words = compiled_pattern.findall(long_text)
print(f"Знайдені слова (скомпільований шаблон): {found_words}")

# --- Приклад 8: Прапорці (Flags) ---

print("\n--- Прапорці (Flags) ---")
text12 = "Hello World\nhello python"

# re.IGNORECASE (re.I) - ігнорувати регістр
match_i = re.search(r"hello", text12, re.IGNORECASE)
print(f"Збіг 'hello' (без регістру): {match_i.group(0) if match_i else 'Не знайдено'}")

# re.MULTILINE (re.M) - ^ і $ відповідають початку/кінцю кожного рядка
text13 = "Перший рядок\nДругий рядок\nТретій рядок"
```

```
match_m = re.findall(r"^Другий", text13, re.MULTILINE)
print(f"Збіг '^Другий' (багаторядковий): {match_m}")

# re.DOTALL (re.S) - . відповідає будь-якому символу, включаючи новий рядок
text14 = "Рядок 1\nРядок 2"
match_s = re.search(r"Рядок.Рядок", text14) # Без DOTALL не знайде
print(f"Збіг 'Рядок.Рядок' (без DOTALL): {match_s.group(0) if match_s else 'Не знайдено'}")
match_s_dotall = re.search(r"Рядок.Рядок", text14, re.DOTALL)
print(f"Збіг 'Рядок.Рядок' (з DOTALL): {match_s_dotall.group(0) if match_s_dotall else 'Не знайдено'}")

# --- Приклад 9: Іменовані групи ---

print("\n--- Іменовані групи ---")
text15 = "Ім'я: Іван, Вік: 30"
pattern15 = r"Ім'я: (?P<name>\w+), Вік: (?P<age>\d+)"
match15 = re.search(pattern15, text15)
if match15:
    print(f"Ім'я: {match15.group('name')}")
    print(f"Вік: {match15.group('age')}")
```