**Engineer**

# Регулярний вираз

шаблон, що описує, які рядки тексту вважати збігом

## Що це таке

Регулярний вираз — це короткий шаблон, що описує множину рядків, щоб програма могла вибрати ті, які до неї належать. Стандарт POSIX, який видають The Open Group та IEEE, відкриває свій розділ на цю тему описом регулярних виразів як способу вибирати певні рядки з множини рядків символів. У щоденному мовленні та в назвах багатьох функцій і бібліотек термін скорочують до regex або regexp — саме так пишуться об'єкт RegExp у JavaScript і пакет regexp у Go.

Більшість символів у шаблоні означають самі себе: `a` у шаблоні збігається з `a` в тексті. Кілька символів особливі. Крапка збігається з будь-яким символом, `*` повторює попередній елемент нуль або більше разів, вираз у квадратних дужках, як-от `[0-9]`, збігається з одним символом із набору, `^` і `$` прив'язують шаблон до початку та кінця, а інтервал на кшталт `{4}` вимагає точної кількості повторень. Отже, `^[0-9]{4}$` описує будь-який рядок рівно з чотирьох цифр, без інших символів до чи після. Власний приклад інтервалу в стандарті — `c{3}`, застосований до `abababccccccd`: він збігається із символами з сьомого по дев'ятий, тобто з першими трьома із шести с.

Діалект не один. POSIX сам визначає два: базові регулярні вирази (BRE), якими користується більшість його утиліт, і розширені регулярні вирази (ERE), якими натомість користуються деякі. Функції C `regcomp()` і `regexec()` підтримують обидва. Різниця не косметична. В ERE `a+` означає одне або більше `a`. У BRE `+` — звичайний символ, а чи означає `\+` «одне або більше», чи буквальний знак плюс, стандарт залишає на розсуд кожної реалізації.

Друга велика родина походить від Perl. Специфікація ECMAScript зазначає, що форма й функціональність регулярних виразів JavaScript змодельовані за засобами регулярних виразів Perl 5. Пакет `regexp` у Go описує свій синтаксис як загальний синтаксис, яким користуються Perl, Python та інші мови, а модуль `re` у Python — третій представник цієї родини. Ці рушії мають спільну більшу частину нотації, але не всі деталі. Скорочені класи — показовий приклад: `\d` (цифра) і `\w` (символ слова) скрізь виглядають однаково, але що саме вони охоплюють, залежить від рушія та його прапорців, як показує розділ про практику нижче.

Застосуйте поняття до рішення

Застосуйте «Що таке регулярний вираз?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Одна команда веде ідею від ескізу до продукту, у який ваші клієнти вже можуть увійти, а не перекидає роботу між трьома підрядниками.

**Переглянути наші послуги** Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Збільшила дохід GIS-застосунку в 10 разів — 10x зростання виторгу завдяки картографічному продукту

**Збільшила дохід GIS-застосунку в 10 разів** Зв'язатися з нами →

## Чому це важливо

Регулярні вирази сидять у великій частині програмного забезпечення, яке зовсім не схоже на обробку тексту: перевірка полів форм, фільтри, що витягують значення з рядків журналу, правила маршрутизації у вебсерверах, пошук і заміна в текстових редакторах. Кожен шаблон короткий, тож легко не помітити, що він насправді приймає. Результат визначають два питання, і відповідь на обидва залежить від рушія: який збіг обрано, коли можливих кілька, і скільки часу може знадобитися, щоб знайти один.

POSIX обирає найлівіший найдовший збіг: з усіх збігів, що починаються найраніше в тексті, перемагає найдовший (англійською leftmost-longest). Приклади стандарту — BRE `bb*`, що збігається із символами з другого по четвертий у `abbbc`, і ERE `(wee|week)(knights|night)`, що збігається з усіма десятьма символами `weeknights`. Родина Perl працює інакше. Вона пробує альтернативи в тому порядку, в якому їх записано, і бере першу, що веде до збігу; документація Go називає це leftmost-first. Тож той самий шаблон на тому самому тексті може дати дві відповіді:

```
$ echo ab | grep -oE 'a|ab'
ab
>>> re.search("a|ab", "ab").group()
'a'
> "ab".match(/a|ab/)[0]
'a'
```

Go пропонує обидві поведінки. Функція `Compile` використовує семантику leftmost-first, а `CompilePOSIX` обмежує синтаксис до POSIX ERE і перемикається на leftmost-longest. Навіть тоді пакет зберігає власне правило вибору підзбігів, і документація пояснює чому: правило POSIX, за її словами, обчислювально непомірно дороге і навіть не визначене чітко.

Друге питання — час. Модуль `re` у Python, як і більшість рушіїв родини Perl, працює з поверненням (backtracking). Коли частина шаблону не може продовжитися, рушій відступає назад і пробує інший спосіб зіставити те, що було раніше. Документація Python показує цю ідею на `a*a` проти `aaaa`: `a*` спершу забирає всі чотири літери, а потім віддає одну назад, щоб фінальне `a` могло збігтися. Для більшості шаблонів ці витрати непомітні. Для деяких кількість варіантів, які треба перебрати, зростає експоненційно з довжиною вхідних даних, і рядок із кількох десятків символів може тримати процесор зайнятим хвилинами. MITRE каталогізує це як CWE-1333, Inefficient Regular Expression Complexity, і наводить ReDoS (regular expression denial of service) та catastrophic backtracking як інші назви того самого. Пакет regexp у Go спроєктовано навколо протилежної обіцянки. Його документація стверджує, що зіставлення гарантовано виконується за час, лінійний щодо розміру вхідних даних, і додає, що більшість реалізацій з відкритим кодом такої гарантії не дають.

## Як це виглядає на практиці

Візьмімо форму реєстрації для клієнтів у Данії та Україні і поле імені, яке має приймати лише літери. Очевидний шаблон — `[A-Za-z]+`, і в Python 3.15 він поводиться так:

```
>>> import re
>>> re.fullmatch(r"[A-Za-z]+", "Mia")
<re.Match object; span=(0, 3), match='Mia'>
>>> re.fullmatch(r"[A-Za-z]+", "Søren")
>>> re.fullmatch(r"[A-Za-z]+", "Олена")
>>> re.fullmatch(r"\w+", "Søren_42")
```

```
<re.Match object; span=(0, 8), match='Søren_42'>
>>> re.fullmatch(r"[^\W\d_]+", "Олена")
<re.Match object; span=(0, 5), match='Олена'>
>>> re.fullmatch(r"[^\W\d_]+", "Søren_42")
>>>
```

Виклик, що повертає `None`, в інтерактивній оболонці взагалі не друкує рядка, тож ASCII-клас відкидає Søren та Олену, два цілком звичайні імена в Копенгагені та Києві. Перехід на `\w` їх пропускає, але в Python `\w` охоплює всі символи, для яких `str.isalnum()` повертає істину, плюс знак підкреслення, тож цей клас пропускає й `Søren_42`. Клас `[^\W\d_]` виключає символи, що не є символами слова, десяткові цифри та знак підкреслення. Він також допускає деякі числові символи, які не є десятковими цифрами, тож це не сувора перевірка того, чи всі символи є літерами. Для імен із кількох слів `[^\W\d_]+( [^\W\d_]+)*` приймає і `Anne Sofie Østergaard`, і `Олена Коваль`. Цей шаблон усе ще відкидає `Anne-Marie` з дефісом, і це привід тримати поле імені настільки ліберальним, наскільки дозволяє бізнес. Інструменти POSIX мають ту саму проблему в іншій формі: стандарт залишає значення діапазону на кшталт `A-Z` невизначеним поза локаллю POSIX і пропонує клас `[[:alpha:]]`, склад якого визначає поточна локаль.

Правильне рішення тут важить значно більше, ніж одна форма. Одна з послуг цього сайту, Інтернаціоналізація та локалізація, описує свою мету як програму, що природно читається на кожному ринку, де продає клієнт, аж до сортування імен. Поле імені, яке відкидає Søren, розходиться з цією метою задовго до будь-якого сортування.

Цифри ставлять протилежне питання. Данський поштовий індекс складається з чотирьох цифр, тож `\d{4}` виглядає надійною перевіркою для `2200`. Але в Python `\d` у текстовому шаблоні збігається з будь-якою десятковою цифрою Unicode, а не лише з цифрами від 0 до 9, і приймає той самий індекс, записаний арабсько-індійськими цифрами:

```
>>> s = "\u0662\u0662\u0660\u0660"
>>> print(s, int(s))
٢٢٠٠ 2200
>>> re.fullmatch(r"\d{4}", s)
<re.Match object; span=(0, 4), match='٢٢٠٠'>
>>> re.fullmatch(r"\d{4}", s, re.ASCII)
>>> re.fullmatch(r"[0-9]{4}", s)
>>>
```

JavaScript стоїть на протилежному боці. Специфікація ECMAScript визначає `\d` як рівно десять символів від 0 до 9, тож `/^\d{4}$/.test(s)` для того самого рядка повертає `false` у Node.js, тоді як `/^\p{Nd}{4}$/u` повертає `true`. Форма, яку браузер перевіряє за одним правилом, а сервер — за іншим, дасть дві різні відповіді на ті самі дані. `int()` у Python читає рядок як 2200, але стовпець бази даних, таблиця-довідник чи інший сервіс можуть прочитати його інакше. Вирішіть, які цифри маються на увазі: `[0-9]` або прапорець `re.ASCII`, коли годяться лише цифри ASCII, і `\p{Nd}` з прапорцем `u` у JavaScript, коли підходить будь-яка десяткова цифра.

## Де виникають проблеми

**Шаблон, що подвоює час роботи з кожною літерою.** Шаблон для імені, який має дозволяти слова, розділені одиничними пробілами, легко написати як `([^\W\d_]+ ?)+`. На правильному імені він швидкий. На послідовності літер, за якою йде символ, що його шаблон прийняти не може, рушій перебирає всі способи поділити літери на слова, перш ніж здатися, і кількість способів подвоюється з кожною літерою. На одній машині з Python 3.15 найшвидший із п’яти запусків для кожної довжини дав такі часи для `fullmatch` на рядку з літер, за якими йде `!`:

```
літери    секунди
    18      0.017
    20      0.067
    22      0.27
    24      1.0
    26      4.1
```

Кожна літера подвоює час, тож ще дві літери збільшують його вчетверо. Якщо екстраполювати цей множник далі, тридцять літер забрали б близько хвилини, а тридцять дві — понад чотири хвилини. Це саме той випадок, який описує CWE-1333: кількість спроб повернення зростає експоненційно з вхідними даними, вхідні дані не мусять збігатися з шаблоном, і вони можуть бути достатньо довгими. Переписаний шаблон `[^\W\d_]+( [^\W\d_]+)*` описує ті самі імена без вкладеного квантифікатора, відхиляє рядок із 24 літер приблизно за мікросекунду, а мільйон літер, за якими йде `!`, — приблизно за тридцять мілісекунд. Приберіть вкладені квантифікатори, обмежте довжину вхідних даних перед зіставленням і запускайте шаблони, що бачать неперевірені вхідні дані, на рушії з гарантією лінійного часу, як-от у Go.

**Пошук там, де має збігатися все значення.** У Python `re.search` приймає збіг будь-де в рядку, а `re.match` прив'язується лише до початку; Python 3.15 дає цій функції зрозуміліше ім'я `prefixmatch` і м'яко визнає старе застарілим (soft deprecation), зауважуючи, що в інших мовах match означає те, що Python завжди називав search. Тож `re.prefixmatch(r"\d{4}", "22000")` спрацьовує на перших чотирьох цифрах, а `re.search(r"\d{4}", "DK-2200")` знаходить індекс усередині рядка. Навіть `re.search(r"^\d{4}$", "2200\n")` спрацьовує, бо `$` збігається ще й безпосередньо перед символом нового рядка в кінці. Функція `re.fullmatch`, доступна з Python 3.4, відкидає всі три. Використовуйте `fullmatch` щоразу, коли шаблон описує все значення, або прив'язуйте його через `\A` і `\Z`, який Python 3.14 пише також як `\z`.

**Список дозволених адрес із незахищеною крапкою.** Перевірку, яка пропускає перенаправлення лише на власний сайт компанії, можна написати як `^https://www.example.com`. Вона приймає `https://www.example.com.evil.example/`, бо шаблон закінчується раніше, ніж ім'я хоста, а кожна неекранована крапка збігається з будь-яким символом, тож хост, який відрізняється від справжнього одним символом на місці крапки, теж проходить. Екранування крапок, як це робить `re.escape("www.example.com")`, і `/` в кінці шаблону закривають обидві діри. А ще краще — розберіть URL і порівняйте його ім'я хоста з очікуваним як цілий рядок.

**Шаблон, перенесений між рушіями.** Шаблон, перевірений через `grep -E` і вставлений у Python, може обрати інший збіг, як показує приклад з `a|ab`, а шаблон, що працює в одному інструменті BRE з `\+`, в іншому може означати буквальний знак плюс. Той самий `\d`, що в Python приймає арабсько-індійські цифри, у JavaScript їх відкидає. Тестуйте кожен шаблон у тому рушії, який його виконуватиме, і з тими прапорцями, з якими він працюватиме.

**Глобальний прапорець, який пам'ятає.** У JavaScript регулярний вираз із прапорцем `g` або `y` зберігає в `lastIndex` позицію, де закінчився його останній збіг, і починає звідти наступний пошук; після невдалої спроби він скидає `lastIndex` до 0. Шаблон, збережений у константі й повторно використаний для перевірки, тоді відповідає по-різному через раз:

```
> const re = /^\d{4}$/g;
undefined
> re.test("2200")
true
> re.test("2200")
```

```
false
> re.test("2200")
true
```

Не ставте прапорець `g` на шаблони, які використовуєте з `test()`, або встановлюйте `lastIndex` у 0 перед кожним викликом.

**Текст, який виглядає однаково, але відрізняється.** Ім'я André можна зберегти з é як одним готовим символом або зі звичайним e, за яким іде комбінувальний діакритичний знак. На екрані обидва варіанти однакові, але POSIX стверджує, що зіставлення спирається на бітовий шаблон, яким закодовано символ, а не на його графічне подання, і `re.search("André", "Andre\u0301")` у Python повертає `None`. Після `unicodedata.normalize("NFC", ...)` на тексті виклик знаходить ім'я. Нормалізуйте шаблон і текст до однієї форми перед зіставленням: через `unicodedata.normalize` у Python або `String.prototype.normalize` у JavaScript.

## Що почитати далі

- The Open Group Base Specifications Issue 8: Regular Expressions — визначення POSIX: BRE і ERE, вирази у квадратних дужках, класи символів і локалі та правило leftmost-longest.
- regexp package (Go) — гарантія лінійного часу, leftmost-first проти leftmost-longest і чому CompilePOSIX відходить від POSIX щодо підзбігів.
- re — Regular expression operations (Python) — синтаксис, класи Unicode і ASCII, пошук із поверненням і присвійні квантифікатори, а також search, prefixmatch і fullmatch.
- ECMAScript: Text Processing (living draft, ECMAScript 2027) — регулярні вирази JavaScript: спадок Perl 5, визначення `\d`, екрановані послідовності властивостей Unicode і використання lastIndex.
- CWE-1333: Inefficient Regular Expression Complexity — catastrophic backtracking і ReDoS: умови, за яких шаблон можна використати для атаки, зафіксовані приклади та заходи захисту.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

**Дивитися, скільки це коштує** Зв'язатися з нами →

# Зробіть термінологію практичною

Запитайте, як «Що таке регулярний вираз?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

**Переглянути наші послуги**

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

**Дбайливо зроблено з любов'ю — D.**

**CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N**

**Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.**

**Надруковано з https://engineer.company/uk/glossary/regular-expression/**