

Wyrażenia regularne - regex

Ewa Namysł

Uniwersytet Śląski

10 marca 2022

Spis treści

Czym są wyrażenia regularne?

Podstawy składni i przykłady

Proste przykłady w praktyce

Podsumowanie

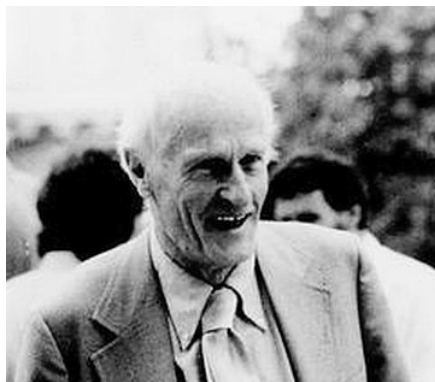
Czym są wyrażenia regularne?

Wyrażenia regularne, w skrócie regex,
to wzorce opisujące łańcuchy symboli.

Upraszczając, jest to zbiór zasad opisujących
jakiś tekst, zestaw znaków i cyfr etc.

Historia

W 1951 roku matematyk Stephen Cole Kleene stworzył podwaliny teorii opisującej języki regularne używając notacji matematycznej.

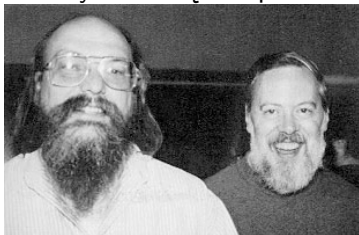


Regex zyskał na popularności pod koniec lat sześćdziesiątych wraz z rozwojem Unixa.

Ken Thompson, współtwórca Unixa, jako jeden z pierwszych wykorzystał notację Kleene'a w edytorze tekstu *QED*.

Thompson stworzył również na własny użytek narzędzie wyszukiujące pliki według wzorców regexa.

Obecnie znamy to narzędzie pod nazwą *grep*.



Co zyskujemy?

Dzięki wyrażeniom regularnym jesteśmy w stanie odnaleźć w tekście wszystkie wystąpienia danego wzorca.

Jeśli poszukujemy konkretnego słowa, odnajdziemy je jako osobny string, ale również jeśli występuje jako składowa część wyrazu.

Przykład

REGULAR EXPRESSION

```
:/ [Kk] arol
```

TEST STRING

Król • Karol • kupił • królowej • Karolinie • korale • koloru • karolinowego.

REGULAR EXPRESSION

```
:/ [Kk] arol [A-Za-z]*
```

TEST STRING

Król • Karol • kupił • królowej • Karolinie • korale • koloru • karolinowego.

Popularne zastosowania

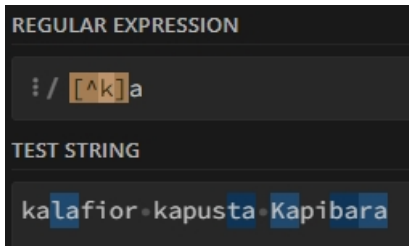
- Walidacja danych - sprawdzanie adresów mailowych, kodów pocztowych, siły haseł etc.
- Webscraping - zautomatyzowane pozyskiwanie danych ze stron internetowych.
- Modyfikacja i podmienianie łańcuchów znaków, obróbka danych.
- Edytory tekstu, kompilatory.

Wiele programów i komend Linuxa wykorzystuje regexy, m.in.
grep, sed, find.

Większość współczesnych języków programowania ma własne
biblioteki obsługujące regexy, np. *re* w Pythonie.

Podstawy składni - kwadratowe nawiasy

- `[]` → wybiera jeden ze znaków wewnątrz nawiasów, np.
`[abc]` wyszuka literę a, b lub c.
- `[A – Za – z]` → dowolna wielka lub mała litera.
- `[0 – 9]` → jedna cyfra od 0 do 9
- `[^]` → zaprzeczenie znaków wewnątrz nawiasu, np.



Krótszy zapis

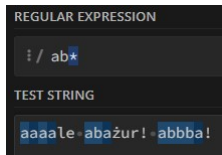
- $\backslash d \rightarrow$ jeden znak, który jest cyfrą
- $\backslash D \rightarrow$ jeden znak, który NIE jest cyfrą
- $\backslash w \rightarrow$ jeden znak, który jest literą
- $\backslash W \rightarrow$ jeden znak, który NIE jest literą
- $\backslash s \rightarrow$ jeden znak, który jest białym znakiem
- $\backslash S \rightarrow$ jeden znak, który NIE jest białym znakiem

Podstawy składni - znaki specjalne i kwantyfikatory

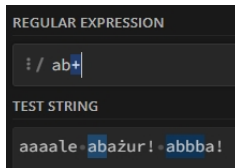
- $\cdot \rightarrow$ jeden dowolny znak (oprócz znaków nowej linii), np.

REGULAR EXPRESSION	REGULAR EXPRESSION
<code>:/ Paj.k</code>	<code>:/ Paj..k</code>
TEST STRING	TEST STRING
<code>Paja.k o imieniu Pajonk.</code>	<code>Paja.k o imieniu Pajonk.</code>

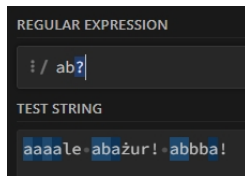
- * → zero lub więcej powtórzeń znaku, np.



- $+$ \rightarrow jedno lub więcej powtórzeń znaku, np.



- ? \rightarrow zero lub jedno powtórzenie znaku, np.



- $\{ \}$ → określona liczba powtórzeń znaku, np.

REGULAR EXPRESSION

```
:/kra{3}
```

TEST STRING

```
kraaaa kraaa kraa kra kr!
```

- $\{min, max\} \rightarrow$ zakres powtórzeń znaku

REGULAR EXPRESSION

`/kra{1,2}/`

TEST STRING

`kraaaa kraaa kraa kra kr!`

Kotwice

- \wedge → szuka na początku tekstu podanego ciągu znaków, np.

```
REGULAR EXPRESSION
: / ^tka

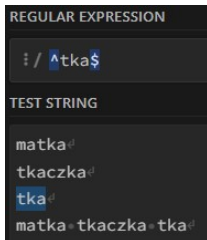
TEST STRING
matkad
tkaczkad
tkad
matka = tkaczka = tkad
```

- $\$$ → szuka na końcu tekstu podanego ciągu znaków, np.

```
REGULAR EXPRESSION
: / tka$

TEST STRING
matkad
tkaczkad
tkad
matka = tkaczka = tkad
```

- `^` oraz `$` można też łączyć. Regex wyszuka wówczas tekst, który od początku do końca pasuje do wzorca:

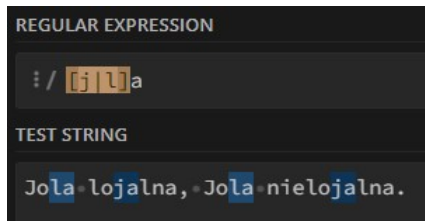


- $\backslash b \rightarrow$ znajduje granicę słów.

REGULAR EXPRESSION	REGULAR EXPRESSION
<code>:: / \babc\b</code>	<code>:: / abc</code>
TEST STRING	TEST STRING
<code>ab•abc•abcabc</code>	<code>ab•abc•abcabc</code>

Alternatywa

- \rightarrow umożliwia użycie alternatywy:




Python

 regex.py - C:/Users/eknam/Desktop/regex.py (3.9.7)

File Edit Format Run Options Window Help

```
1 import re
2
3 wzorzec1 = "[0-9][0-9]-[0-9][0-9][0-9]"
4 wzorzec2 = "\d\d-\d\d\d"
5
6 kod_pocztowy = "40-750"
7
8 wynik = re.match(wzorzec1, kod_pocztowy)
9
10 if wynik:
11     print("Poprawny kod pocztowy")
12 else:
13     print("Niepoprawny kod pocztowy")
```

 IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021,
64) on win32
Type "help", "copyright", "credits" or "license"
>>>
===== RESTART: C:/Users/eknam/Des
Poprawny kod pocztowy
>>>
```

Linux

The image shows a terminal window with a dark background. The prompt is `enraha@enraha-VirtualBox:/$`. The command entered is `echo "Zamiana * i _ na #" | sed 's:[*_]:#:g'`. The output is `Zamiana # i # na #`. There are four white arrows pointing to specific parts of the command and output with labels in Polish:

- An arrow points to the `*` in the string `"Zamiana * i _ na #"` with the label "dzięki backslashowi gwiazdka traktowana jest jak zwykły znak".
- An arrow points to the `#` in the replacement string `#:g` with the label "zamiana na znak #".
- An arrow points to the `s:` in the sed command with the label "znajdź wzorec i zamień na wybrany znak".
- An arrow points to the `g` in the sed command with the label "operację wykonaj na każdej instancji pasującej do wzorca".

Below the output `Zamiana # i # na #`, there is a white double-headed arrow pointing to the spaces between the words.

Wady

'Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.' – Jamie Zawinski

- Bardzo złożona składnia, podatność na pomyłki i fałszywie pozytywne wyniki.
- Ekstremalnie trudny do modyfikacji dla osoby, która go nie pisała.
- Zasobożerny.

Zalety

- Często krótsze, mniej rozbudowane rozwiązanie dla bieżących problemów (choć niekoniecznie szybciej na nie wpadniemy).
- Współpraca z komendami Linuxa.
- W niektórych sytuacjach (np. loginy userów, wprowadzanie tekstu przez usera) może okazać się niezastąpiony przez swoją elastyczność.

Bibliografia:

J. Goyvaerts, *Regular Expressions*,
<https://www.regular-expressions.info/> (dostęp kwiecień 2022).

R. Winslow, *Regex basics*, <https://ubuntu.com/blog/regex-basics/>
(dostęp kwiecień 2022).

M. Mamczur, *Wyrażenia regularne – czym są i jak pisać własne regex'y?*, <https://mirosławmamczur.pl/wyrażenia-regularne-czym-sa-i-jak-pisac-wlasne-regexy/> (dostęp kwiecień 2022).

Regular Expressions 101, <https://regex101.com/>, (dostęp kwiecień 2022).