# Regulérne Výrazy*

Tomáš Tytykalo
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xtytykalo@stuba.sk

8. október 2023

**Abstrakt**

Článok sa zameriava na aplikáciu regulérnych výrazov v oblasti informatiky a textového spracovania. Regulérne výrazy poskytujú používateľsky jednoduchý a efektívny spôsob vyhľadávania a manipuláciu textových dát. Táto práca začína popisom základov regulérnych výrazov, vrátane ich syntaxe a operácií. Následne sa zameriavame na ich použitie v rôznych kontextoch, vrátane bežného využitia v praxi a teda aj v množstve bežne dostupných programov, alebo programovacích jazykoch. Predstavíme si v ktorých situáciách sa dajú efektívne a jednoducho aplikovať. Regulérne výrazy sú teda mocným nástrojom pre každého dobrého informatika, či programátora.

*Semestrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2023/24, vedenie: Ing. Ivan Kapustík

# Obsah

# 1 Úvod

Regulérne výrazy, taktiež známe ako RegEx, či RegExp (Regular Expression) sú populárne a husto podporované v programoch, kde majú zmysel. Ako napríklad textové editory.

Regulérne výrazy predstavujú jednoduchý a efektívny spôsob vyhladávania v textových reťazcoch. Ich aplikácia zahŕňa extrakcia informácií, overovanie užívateľského vstupu, alebo aj nahrádzanie textu.

Hlavné využitie má v programovacích a skriptovacích ako napríklad JavaScript, PHP, Python ale taktiež aj v kope iných typu C++, C#. Ich podporu nachádzame bežne v textových editoroch ako nástroj na vyhladávanie a nahrádzanie.

Prejdeme si ich zápis, funkcionalitu, využitie v programoch, a aj v programovacích jazykoch.

Výborný papier na túto tému je [2]. Obsahuje podobný obsah, ale s inými príkladmi a iným vysvetlením. V prípade zložitosti tohoto papiera alebo vzniknutých nejasností ho doporučujem skontrolovať.

# 2 Zápis regulérnych výrazov

Pre túto sekciu sme čerpali informácie z publikácie [1]. Je to podobný papier tejto sekcií s rozdielom že sa aj zaoberá linuxovým prostredím, ktorý sme mi nespomenuli. Taktiež papier [4] obsahuje podrobnejšie informácie ohľadom implementácie samotných regulérnych výrazov, ktorá tu tiež nie je spomenutá aby sme nekomplikovali jednoduchosť tohoto papiera.

## 2.1 Základné regulérne výrazy

Jednoduchý regulérny výraz môže vyzerať následovne:

```
/Auto/g
```

Tento výraz nájde všetky prípady 'Auto' v celom textovom reťazci. Poďme si ich ale rozobrať trošku hlbšie. Všetko medzi dvoma // je formát vyhľadávania, teda to čo nás teraz zaujíma. Všetko vonku // je modifikátor, k týmto sa dostaneme neskôr.

### 2.1.1 Rozsah

Výrazy typu hladania jedného reťazca sú pomerne jednoduché. Napíšeme iba vyhľadávaný reťazec, tak ako v predošlom príklade.

```
Výraz: /Auto/g
Vyberie: Auto
```

Výraz nám teda nájde všetky 'Auto' v texte, ale čo keď chceme aj äuto"s malým písmenkom na začiatku? V tomto prípade prichádza prvý koncept menom *rozsah*. Zapisuje sa do hranatých zátvoriek. Rozsahy môžeme aj negovať pomocou operátora ^, aby sme vybrali všetky znaky okrem tých v rozsahu.

```
Výraz: /[aA]uto/g
Vyberie: auto Auto
```

| Operátor | Popis |
|---|---|
| + | musí byť aspoň raz, ale môže viac krát |
| ? | môže a nemusí tam byť |
| * | môže byť 0-krát alebo viac krát |
| {min, max} | vyskytuje sa aspoň min-krát a najviac max-krát |

Tabuľka 1: RegEx Kvantifikátory

Teraz náš výraz vyberie 'auto' aj 'Auto' z textu, lebo sme v rozsahu pre prvé písmenko špecifikovaly *malé* a aj *veľké* a. Ak by sme chceli vybrať všelijaké veľké a malé písmenko, tak rozsah zapíšeme s '-' ako interval.

```
Výraz:  /[a-zA-z]uto/g
Vyberie:  auto Auto xuto puto
```

Tento výraz vyberie všetky prípady, kde je ľubovolné písmenko a za ním 'uto'.

```
Výraz:  /[^a-zA-z]uto/g
Vyberie:  0uto 1uto $uto !uto
```

Výraz vyberie reťazce s lubovoľným znakon okrem písmen a s 'uto'.

### 2.1.2 Kvantifikátory

Začneme opäť s jednoduchím výrazom pre 'Auto'.

```
Výraz: /Auto/g
Vyberie: Auto
```

Ako by sme vybrali opakujúce sa znaky, alebo písmena, ktoré tam môže a nemusia byť? Máme na to následujúci špeciálne znaky určené pre kvantifikáciu. Tieto operátory sa aplikujú na jednotlivé znaky, ale môžu aj na rozsahy a skupiny.

- ```
  Výraz: /A+uto/g
  Vyberie: AAuto Auto AAAAAAAAAAuto
  ```
- ```
  Výraz: /A?uto/g
  Vyberie: Auto uto
  ```
- ```
  Výraz: /A*uto/g
  Vyberie: AAuto Auto AAAAAAAAAAuto uto
  ```
- ```
  Výraz: /[Aa]{3,3}uto/g
  Vyberie: AAAuto AAauto Aaauto aaauto
  ```

## 2.2 Špeciálne znaky a skupiny znakov

```
Výraz: /Auto/g
Vyberie: Auto
```

Dokážeme už vybrať všetky vopred známe znaky a prípadne ich opakovanie. A čo keď nevieme aký znak bude prvý? Alebo ak sa nám nechce vypisovať každé písmenko abecedy a všetky známe znaky. Našťastie nemusíme, tvorcovia regulérnych výrazov na to mysleli. Medzi takéto špeciálne znaky a skupiny znakov patrí.

| Znak / Znaková skupina | Slovný popis | Rozsah |
|---|---|---|
| . | lubovolný znak okrem nového riadku '\n' | |
| ^ | začiatok reťazca | |
| $ | koniec reťazca | |
| \s | všetky 'whitespace' znaky | [ \f\n\r \v] |
| \S | všetko okrem 'whitespace' znakov | [^ \f\n\r\t\v] |
| \w | všetky znaky slov | [a-zA-Z0-9_] |
| \W | všetko okrem znakov slov | [^a-zA-Z0-9_] |
| \d | všetky číslice | [0-9] |
| \D | všetko okrem číslic | [Ô-9] |

Tabuľka 2: Špeciálne znaky a znakové skupiny

Tabuľka 3: RegEx Skupiny

| Zápis | Popis |
|---|---|
| (x) | Zachytávacia skupina |
| (?Názov_Skupinyx) | Pomenovaná skupina |
| (?:x) | Skupina ktorú nezachytávame |
| (?=x) | Skupina, ktorá ak nie je za výrazom, tak záznam vynechávame |
| (?!x) | Skupina, ktorá ak je za výrazyom, tak záznam vynechávame |
| (?=x) | Skupina, ktorá ak nie je pred výrazom, tak záznam vynechávame |
| (?!x) | Skupina, ktorá ak je za výrazyom, tak záznam vynechávame |

### 2.2.1 Skupiny

Skupiny slúžia na segmentáciu nášho vyhladávania alebo pridávania podmienok. Tieto skupiny sú nápomocné pri programovaní, kde môžeme k jednotlivým skupinám pristupovať podľa pomenovania. Taktiež ich môžeme použiť na nájdenie reťazca, ktorý nechceme vo finále vybrať ako napríklad pri práci s CSV (Comma Seperated Values) súbormi, kde jednotlivé záznamy sú zakončené čiarkou, ale nechceme vybrať čiarku.

```
Vstup:   Name,Email,Phone Number,Address
Výraz: /[^,\n]+(?=[,\n]?)/g
Vyberie: 'Name', 'Email', 'Phone Number', 'Address'
```

Tento výraz vyberá záznamy z CSV zoznamu hodnôt a teda vyberá všetko pokiaľ nenájde čiarku, alebo nový riadok. Je možné tento výraz vylepšiť aby ignoroval medzeri na začiatku zápisu, alebo aby mohla byť čiarka v zázname, ale to necháme ako príklad čitateľovi.

Skupinu dokážeme nastaviť tak, aby sme ju nezahŕňali do výsledkov. Taktiež ich dokážeme použiť ako podmienku a vyhodnotiť podľa toho či náš reťazec uložiť, alebo preskočil.

```
Výraz: /[Aa]{3,3}uto/g
Vyberie: AAAuto AAauto Aaauto aaauto
```

### 2.3 Nahrádzanie

Prebieha s pomocou nástrojov alebo programovacieho jazyku. Doteraz sme iba vyhľadávali skupiny reťazcov, teraz môžeme do programu zadať náš regulérny výraz a dosadzovací text. Program každú nájdenú skupinu nahradí za náš text a namiesto toho aby nám vrátil všetky nájdené skupiny, tak nám dá nový text po dosadení.

```
Vstup: 'I love cats, do you love cats as well?'
Výraz: /cats/g
Dosadzovací text:  dogs
Výsledok: 'I love dogs, do you love dogs as well?'
```

### 2.4 Modifikátory

Doposial sme menili výraz medzi //, ale konečne sa dostávame mimo lomítka. Tieto modifikátory dávajú informáciu programu vyhodnocujúcemu nášho regulérneho výrazu extra informácie. Taktiež sa nazývajú vlajky a môžu ich byť vybraných viacero.

```
Výraz: /auto/gi
Vyberie: Auto auto AUto aUto
```

| **Možnosť** | **Popis** | **Znak** |
|---|---|---|
| global | vyhľadáva všetky zhody namiesto iba prvej | g |
| case insensitive | nerozlišuje medzi veľkými a malými písmenkami | i |
| multiline | znak ^ a $ ukazujú na začiatok a koniec riadka | m |
| single line | znak . bude vyberať aj nový riadok '\n' | s |
| unicode | pridáva možnosť používať | u |
| sticky | vyraz začne vyhľadávať na konci minulého vyhľadávania | y |

Tabuľka 4: RegEx modifikátory

## 3 Aplikácia

Posnažíme sa vylistovať niekoľko populárnych programov, ktoré pravdepodobne už poznáte a kde môžete uplatniť svoje nadobudnuté vedomosti.

Taktiež aj zopár bežne používaných programovacích jazykoch, ktoré majú podporu pre regulérne výrazy a ukážeme si príklad využiťia v programovaciom jazyku python.

### 3.1 Textové editory

Použitie regulérnych výrazov je veľmi situačné pri bežnej práci s textovými editormi, ale občas sa stane že potrebujeme niečo premenovať, alebo vyhladať konkrétny reťazec a naše bežné vyhladávanie s konštatným textom nestačí.

V týchto prípadoch môžeme využiť regulérne výrazy, vďaka ich užitočnosti sú pomerne často podporované.

Vylistujeme si zopár textových editorov čo podporujú ich funkcionalitu a aj úroveň podpory.

1. Microsoft Visual Studio Code
    - Populárny textový editor s kompatibilitou pre väčšinu súborových formáto
    - Podporuje používanie regulérnych výrazov pri vyhľadávaní textu a nahrádzaní textu
2. MikText - TeXworks
    - Integrované prostredie pre prácu s LaTeX dokumentami
    - Podporuje používanie regulérnych výrazov pri vyhľadávaní textu a nahrádzaní textu
    - Vrati celé riadky namiesto skupín
3. Microsoft Visual Studio
    - Integrované vývojové prostriedie pre programovanie v množstve programovacích jazykoch
    - Podporuje používanie regulérnych výrazov pri vyhľadávaní textu a nahrádzaní textu

## 3.2 Programovacie jazyky

V prípade skriptovacieho jazyka Perl si môžete prečítať papier [5] v ktorom autor predstavuje skriptovací jazyk Perl a taktiež obsahuje sekciu pre regulérne výrazy.

Okram skriptovacieho jazyka Perl, regulérne výrazy podpurujú aj následovné jazyky a pravdepodobne aj všetky jazyky, ktoré sa dnes bežne používajú v priemysle.

1. Javascript
    - Skriptovací jazyk s hlavným účelom pre webové stránky
    - Obsahuje podporu pre takmer všetky funkcie regulérnych výrazov
    - Funkcionalita môže závisieť od prehliadača v ktorom daný skript beží
2. Python
    - Populárny skriptovací jazyk vhodný pre programy ktoré sú časovo nekritické
    - Jednoduchý pre začiatočníkov
    - Knižnica 're' poskytuje funkcionalitu regulérnych výrazov
3. C++
    - Jazyk používaný v priemysle pre aplikácie, kde výkon je nutnosť
    - Štandarná knižnica poskytuje implementáciu regulérnych výraz
    - Definície funkcií sa nachádzajú v hlavičkovom súbore 'regex'

## 3.3 Príklady použitia

### 3.3.1 Validácia vstupu používateľa

Nápad je pomerne jednoduchý. Napíšeme výraz, ktorý následne aplikujeme na používateľov vstup. Ak nám vráti zhodu a tá zhoda je celý vstup, tak potom je vstup platný, inak je vstup nevhodný.

Platný email

```
Vstup: 'vsprintf@comcast.net' 'crobles@gmail.com' 'skola@is.stuba.sk' 'test@.com' 'st
Výraz: /^\w+@\w+[\w.]+$/gm
Vyberie: 'vsprintf@comcast.net' 'crobles@gmail.com' 'skola@is.stuba.sk'
```

Telefónne číslo Kontroluje iba bežné čísla, čiže 112 a iné špeciálne telefónne čísla vyradí.

```
Vstup: '+421940264961' '+421908127668' '0950279858' '0911538050' '0911 538 050' '+420
Výraz: /(?:\+\d{3,3}|0)(?: )?\d{3,3}(?: )?\d{3,3}(?: )?\d{3,3}/gm
Vyberie: '+421940264961' '+421908127668' '0950279858' '0911538050' '0911 538 050' '+4
```

### 3.3.2 Načítavanie údajov z dátového súboru

Máme JSON súbor, v ktorom sú údaje z vozidiel MHD v Bratislave. Drží informácie ako identifikátor vozidla, liknu obsluhy, GPS pozíciu a čas aktualizácie záznamu deného vozidla.

Chceme tieto údaje spracovať programovo a teda ich najprv musíme načítať. Keďže máme zoznam vozidiel, inými slovami pole a údaje sú pomenované, tak nám regulérne výrazy podstatne uľahčia život.

Nadovšetko musím dať na vedomie. Tento spôsob je pomalý a existujú lepšie spôsoby ako pracovať s JSON formátom a teda toto je iba príklad využitia.

```
Vstup:
[
  {
    "vehicleNumber": 1021,
    "lineNumber": 53,
    "gpsLatitude": 48.177199,
    "gpsLongitude": 17.166676,
    "lastModified": "2023-09-22T16:40:23.1371261+02:00"
  },
  {
    "vehicleNumber": 2339,
    "lineNumber": 95,
    "gpsLatitude": 48.190326,
    "gpsLongitude": 17.134353,
    "lastModified": "2022-12-22T10:34:01.96552"
  },
  {
    "vehicleNumber": 7406,
    "lineNumber": 4,
    "gpsLatitude": 48.142004,
    "gpsLongitude": 17.089938,
```

```
    "lastModified": "2023-09-22T16:40:22.8595525+02:00"
  }
]
Výraz: /(?<="vehicleNumber": )\d+/g
Výstup: ’1021’ ’2339’ ’7406’
```

V programovacom jazyku python by to vyzeralo následovne.

```
import re

vstup = """
[
  {
    "vehicleNumber": 1021,
    "lineNumber": 53,
    "gpsLatitude": 48.177199,
    "gpsLongitude": 17.166676,
    "lastModified": "2023-09-22T16:40:23.1371261+02:00"
  },
  {
    "vehicleNumber": 2339,
    "lineNumber": 95,
    "gpsLatitude": 48.190326,
    "gpsLongitude": 17.134353,
    "lastModified": "2022-12-22T10:34:01.96552"
  },
  {
    "vehicleNumber": 7406,
    "lineNumber": 4,
    "gpsLatitude": 48.142004,
    "gpsLongitude": 17.089938,
    "lastModified": "2023-09-22T16:40:22.8595525+02:00"
  }
]"""

regulerny_vyraz = re.compile(’(?<="vehicleNumber": )\d+’)
vystup = re.findall(regulerny_vyraz, vstup)

print(vystup) # [’1021’, ’2339’, ’7406’]
```

# 4 Záver

V tomto článku sme si prešli prakticou časťou regulérnych výrazov a nie teóriou ako sú implementované. Prešli sme ich zápis, teda syntax a význam špeciálných znakov. Ukázali si pár ukážkových príkladov na pochopenie a vylistovali aj niekoľko prípadov, kde by sa dali využiť. Pevne verím že sme sa spoločne oboznámili s touto témou a dobre nám poslúži do budúcna pri práci ako informatici.

V prípade nenaplnenej túžby pochopenia implementácie, či záujmu ako to môže fungovať, doporučujem papier [3] v ktorom autor má jasný diagram a popis fungovania regulérnych výrazov.

# Literatúra

[1] Regular expression 1) regular expression basics.

[2] Sanjiv K. Bhatia. Regular expressions.

[3] Regular Expression. Regular expressions. 1995.

[4] Bjørn Bugge Grathwohl, Ulrik Terp Rasmussen, Fritz Henglein, T Jk, T Jk, T Jak a, T Jeek, T Jek, T Jek, T Je, Ek T Jek, and T Jek. Regular expression usage.

[5] Bioperl I, Jason Stajich, Perl Intro, Other General Perl Modules, Perl Bioinformatics, Perl Intro Slide, and Perl Intro. Regular expressions. 2011.