Computersystemen en -architectuur

Reguliere Expressies

1 Ba INF
2022–2023

Kasper Engelen
kasper.engelen@uantwerpen.be

Deze les is bedoeld om zelfstandig Reguliere Expressies te leren kennen en te leren hoe en waar je ze kan gebruiken. Eerst kijken we naar wat een Reguliere Expressie is en hoe we deze kunnen opbouwen. Op het einde zullen we zien hoe we deze expressies in combinatie met enkele UNIX commando's kunnen gebruiken.

De reguliere expressies die je zelf kan uitproberen staan in een grijs kader:

```
reguliere expressie
```

Voor sommige oefeningen staat het resultaat dat je moet krijgen erbij in een blauw kader:

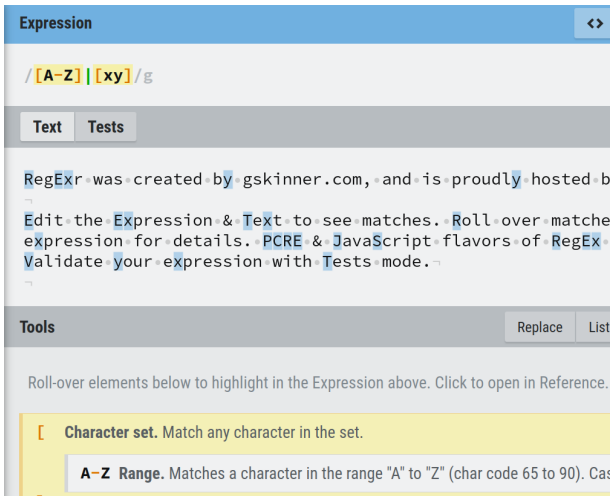
```
resultaat
```

Af en toe zal je ook opdrachten tegenkomen in de groene kaders. Hier is het de bedoeling dat je zelf nadenkt en een commando, of reeks van commando's bedenkt die tot het gevraagde resultaat komen.

Voorbeeld Opdracht

Hier staan de instructies.

Tijdens het eerste deel van deze les maken we gebruik van de online omgeving `https://regexr.com` om de verschillende componenten van Reguliere Expresssies te leren kennen. Het voordeel van deze tool is dat deze uitleg geeft over de betekenis van de expressie die je ingegeven hebt. Bovenaan de tool onder `Expression` kunnen we onze Reguliere Expressie neerschrijven. Vervolgens kunnen we `Text` gebruiken om te testen wat de ingegeven expressie als resultaat heeft. Onderaan in `Tools` kunnen we stap-voor-stap uitleg krijgen van de betekenis van de Reguliere Expressies.



Figuur 1: Screenshot van de RegExr website.

1 Wat en Hoe?

Een Reguliere Expressie is een patroon dat beschrijft hoe een stuk tekst eruitziet. Het beschrijft dus een verzameling van strings. We gebruiken deze patronen om heel eenvoudig textuele informatie op te zoeken (vb. alle e-mailadressen die in een bestand staan) of we kunnen deze ook gebruiken om ingegeven data te controleren op fouten (vb. voldoet het opgegeven adres aan de vereisten voor een e-mailadres?). Een Reguliere Expressie laat toe om de volgende beschrijving formeel uit te drukken: “Een e-mailadres bestaat uit een aantal karakters gevolgd door het symbool ‘@’ gevolgd door een aantal karakters die uiteindelijk gevolgd worden door een extensie van de vorm ‘.’ en dan een landcode”.

2 Basic Reguliere Expressies

Opmerking: Soms kan het zijn dat er op de website in het rood **warning** staat. Indien de oefening werkt zoals het moet, kan je dit gewoon negeren.

Ga naar `https://regexr.com/6vdn5`, je ziet nu de RegExr website met wat voorbeeldtekst. Vul telkens de reguliere expressies (tekst met grijze achtergrond) in om te zien welke karakters gevonden worden. Karakters en woorden die matchen met het opgegeven patroon worden aangeduid in de tekst.

Om aan te geven welke karakters zijn toegelaten, maken we gebruik van verzamelingen van karakters, *Character Sets*. De eenvoudigste verzameling is de volgende:

```
a
```

Deze expressie staat voor een enkel karakter ‘`a`’. We kunnen ook een sequentie van karakters opgeven:

```
at
```

Enkel deelstrings die exact hieraan voldoen worden gevonden. Reguliere Expressies kunnen we gebruiken om te zoeken op exacte woorden. Soms willen we echter ook kunnen zeggen dat eender welk ander karakter toegestaan is. Hiervoor gebruiken we een enkel punt, wat overeenkomt met eender welk karakter:

```
.
```

Merk op dat als we dit invullen in de tool we als resultaat krijgen dat elk karakter in de tekst voldoet aan het patroon. Belangrijk om te onthouden is dat een punt ook met een spatie en andere symbolen overeen komt. Enkel line breaks worden niet herkend door het punt. Buiten exacte matches kunnen we ook zeggen dat we een karakter zoeken uit een verzameling van karakters. Deze verzamelingen van toegestane karakters worden omgeven door rechte haken `[` en `]`. Zoeken we het character `a` of `b` dan kunnen we dat met volgende expressie doen:

```
[ab]
```

We zien nu in onze tekst dat zowel het karakter `a` als `b` gevonden wordt. Stel dat we nu elk karakter van `a` tot en met `z` willen vinden dan kunnen we dit eenvoudiger schrijven door middel van ranges, die worden aangeduid met `-`. Willen we alle kleine letters vinden in een tekst, dan gebruiken we volgende expressie:

```
[a-z]
```

In de tekst zijn nu alle kleine letters aangeduid. Willen we alle hoofdletters vinden dan kunnen we dit op de volgende manier doen:

```
[A-Z]
```

We kunnen ook aangeven dat we alle karakters in een verzameling *niet* willen vinden, dan gebruiken we `[^` en `]`. Om alle hoofdletters te vinden kunnen we dan volgende expressie gebruiken:

```
[^a-z]
```

Zoals we tot nu toe gezien hebben, hebben bepaalde karakters in Reguliere Expressies een speciale betekenis, soms gaan we echter op zoek willen gaan naar deze karakters zelf. We kunnen dit aangeven door een backslash voor dit karakter te zetten. De backslash noemen we dan een *escape character*. We kunnen deze backslash gebruiken als we willen zoeken naar het karakter 'punt' in de tekst. Herinner dat indien we `.` als expressie opgaven, we alle karakters terugkregen. In de onderstaande expressie is het punt *escaped*. Merk het verschil op indien we wel of niet een backslash toevoegen.

```
\.
```

Het punt karakter kunnen we ook matchen door het in een character set te plaatsen:

```
[.]
```

Buiten letters kunnen we uiteraard ook op zoek gaan naar cijfers (en andere karakters). Indien we elk cijfer tussen 0 en 5 willen vinden kunnen we volgende expressie gebruiken:

```
[0-5]
```

Let op: bij Reguliere Expressies hebben we enkel te maken met strings en karakters. Deze expressies hebben geen weet van de eigenlijke betekenis van karakters. Willen we, bijvoorbeeld,

alle getallen van 0 tot en met 13 vinden dan gaat dit bijvoorbeeld niet met de onderstaande expressie:

```
[0-13]
```

Indien we enkele voorbeelden aan het `Text` vak toevoegen kunnen we het resultaat hiervan nakijken. Voeg `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15` toe aan het tekstvak. Dan krijgen we een, op het eerste zicht, vreemd resultaat. Om dit resultaat te begrijpen, moeten we goed doorhebben dat een karakter verzameling slechts slaat op één karakter. De betekenis van bovenstaande expressie is de volgende: we zoeken ofwel een karakter gelijk aan `0` of `1` ofwel het karakter `3`. We zullen later zien hoe we dit kunnen oplossen.

Opdracht 1

Open `https://regexr.com/6vdn8`.
Geef een Reguliere Expressie die de volgende karakters kan vinden: `a b c d e f g`

We kunnen deze character sets ook gaan combineren om patronen van meerdere karakters te vinden. We kunnen zo bijvoorbeeld op zoek gaan naar een karakter `T` (hoofdletter), gevolgd door eender welke kleine letter en vervolgens door een klinker met volgende expressie:

```
T[a-z][aeiou]
```

Zoals met de exacte matches kunnen we ook character sets gaan combineren om langere strings te gaan vinden. Stel we willen elk getal vinden, maar in plaats van elk karakter apart te vinden willen we het hele nummer in zijn geheel vinden. Aangezien we flexibiliteit willen hebben voor het aantal karakters waaruit een nummer bestaat kunnen we de character sets niet simpelweg achtereen plaatsen. Om aan te geven dat een bepaalde character set 0 of meerdere keren mag voorkomen kunnen we gebruik maken van `*`.

```
[0-9]*
```

Als we nu kijken naar welke deelstrings worden gevonden zien we dat elk nummer dat we daarjuist hebben toegevoegd aan de tekst als geheel wordt herkend. We kunnen echter ook een exact aantal karakters opgeven zonder dat we elke character set een aantal keren achtereen moeten kopiëren. Om enkel getallen met 2 cijfers terug te vinden gebruiken we volgende expressie:

```
regexr.com: [0-9]{2}       UNIX: [0-9]\{2\}
```

Bemerk dat er voor deze regex een verschil is tussen de versie voor de RegExr website (links) en de UNIX terminal (rechts). Let dus goed op welke versie je gebruikt!

We kunnen ook het minimum en maximum aantal keren opgeven dat een character set mag voorkomen. Willen we zowel de nummers met 1 als 2 cijfers terugvinden, dan kunnen we volgende expressie gebruiken:

```
regexr.com: [0-9]{1,2}       UNIX: [0-9]\{1,2\}
```

Willen we enkel het minimale of maximale aantal opgeven dan kunnen we de andere waarde weglaten.

```
regexr.com: niet mogelijk       UNIX: [0-9]\{,2\}
```

```
regexr.com: [0-9]{2,}       UNIX: [0-9]\{2,\}
```

We kunnen verder ook expliciet aangeven dat een bepaalde sequentie van karakters omgeven moet zijn door spaties door het gebruik van \< en \> op UNIX of \b op regexr.com. Volgende expressie matcht ofwel het woord The of het woord the:

```
regexr.com: \b[Tt]he\b       UNIX: \<[Tt]he\>
```

Verder kunnen we ook aangeven op welke positie op de lijn een bepaald patroon gevonden moet worden. Zo kunnen we ^ gebruiken om aan te geven dat het patroon op het begin van een lijn gevonden moet worden en $ voor het einde van de lijn. Merk op dat we op regexr.com de *multiline* optie moeten aanzetten om gebruik te maken van deze functionaliteit (zie menu "Flags" rechtsboven).

Dit laat ons bijvoorbeeld toe om alle hoofdletters aan het begin van een lijn te matchen.

```
^[A-Z]
```

Onderstaande expressie zal een punt aan het einde van een lijn vinden.

```
\.$
```

Indien je een karakter wil vinden dat geen whitespace is (spaties, tabs, linebreak) kun je van \S gebruikmaken. Zo zal onderstaande expressie 3 niet-whitespace karakters matchen:

```
\S{3}
```

Soms willen we ook een gevonden patroon opnieuw gebruiken. We kunnen een bepaald patroon groeperen en onthouden op de volgende manier:

```
regexr.com: ([a-z])       UNIX: \([a-z]\)
```

In bovenstaande expressie onthouden we telkens welk karakter (van a tot z) we reeds zijn tegengekomen. Willen we het gevonden patroon opnieuw gebruiken dan kunnen we gebruik maken van \1, \2, \3, etc. Een reguliere expressie om twee identieke karakters te vinden ziet er dan als volgt uit:

```
regexr.com: ([a-z])\1       UNIX: \([a-z]\)\1
```

Een palindroom van lengte 5 kunnen we als volgt noteren:

```
regexr.com: ([a-z])([a-z])[a-z]\2\1      UNIX: \([a-z]\)\([a-z]\)[a-z]\2\1
```

waarbij \2 verwijst naar het tweede opgeslagen patroon en \1 naar het eerste. Deze *backreferences* bevatten het effectief gevonden patroon en verwijzen niet naar het algemene patroon. Zorg ervoor dat je alle voorbeelden hierboven perfect begrijpt voordat je verder gaat.

Hieronder vindt je enkele opdrachten om bovenstaande technieken uit te proberen.

Opdracht 2

Open `https://regexr.com/6vdnb`.
Geef een Reguliere Expressie die een getal op het begin van de lijn teruggeeft.

Opdracht 3

Open `https://regexr.com/6vdne`.
Geef een Reguliere Expressie die filenames met de extensie ".py" teruggeeft

Opdracht 4

Open `https://regexr.com/6vdnh`.
Geef een Reguliere Expressie die woorden van 4 karakters teruggeeft

Opdracht 5

Open `https://regexr.com/6vdnn`.
Geef een Reguliere Expressie die elk nummer tussen 1 en 999 teruggeeft

Opdracht 6

Open `https://regexr.com/6vdnq`.
Geef een Reguliere Expressie die url's met de extensie ".co.uk" teruggeeft

3 Extended Reguliere Expressies

In sommige gevallen zouden we echter meer mogelijkheden willen hebben om keuzes te maken of bepaalde zaken korter op te schrijven. Stel we willen een woord van minstens 1 karakter zoeken dan kunnen we dat doen met volgende Reguliere Expressie:

```
regexr.com: \b[a-z][a-z]*\b       UNIX: \<[a-z][a-z]*\>
```

Met de Extended Reguliere Expressies kunnen aangeven dat we een bepaalde character set 1 of meerdere keren mogen tegenkomen door gebruik te maken van `+`. Bovenstaand commando wordt dan:

```
regexr.com: \b[a-z]+\b       UNIX: \<[a-z]+\>
```

Verder kunnen we ook aangeven dat een bepaalde character set optioneel is (dus 0 of 1 keer voorkomt), dit doen we met `?`.

Een laatste toevoeging is dat we een keuze kunnen maken tussen meerdere deelexpressies door de expressies te scheiden met behulp van het `|` teken.

```
(1[0-3]|[1-9])
```

Bovenstaande code geeft exact alle nummers van 1 tot en met 13 terug. Het eerste deel `1[0-3]` moet gevonden worden, ofwel het tweede `[1-9]`. Merk op dat de volgorde van de deelexpressies van belang is, zo zal de onderstaande expressie bijvoorbeeld niet werken.

```
([1-9]|1[0-3])
```

Opdracht 7

Open `https://regexr.com/6vdnt`.
Geef één Extended Reguliere Expressie die datums van alle onderstaande vormen kan herkennen:

- 31/08/1933
- 2-03-2002
- 09 4 1966
- 15.12.1999

Maak de Reguliere Expressie zo eenvoudig mogelijk door gebruik te maken van al de bovenstaande constructies. Om het eenvoudig te houden, mag je er vanuit gaan dat elke maand 31 dagen telt.

Opdracht 8

Open `https://regexr.com/6vdo0`.
Geef een Extended Reguliere Expressie die geldige IPv4 adressen teruggeeft (0.0.0.0 tot 255.255.255.255)

4 Commando's: grep en sed

De echte kracht van de Reguliere Expressies hangt heel nauw samen met de terminal commando's die van deze expressies gebruik maken. Twee veelgebruikte commando's in UNIX die gebruik maken van Reguliere Expressies zijn `grep` (om te zoeken in een tekstfile) en `sed` (om voorkomens van patronen te vervangen in een tekst). Let op dat beide commando's gebruik maken van de standaard Reguliere Expressies en niet van de uitgebreide. Om de uitgebreide te gebruiken kan men `grep -E` of `sed -E` doen. Vanaf nu werken we telkens in een UNIX terminal voor de voorbeelden. Pas op dat je telkens enkele quotes gebruikt voor je commando's. We kunnen `grep` op volgende manier gebruiken:

```
grep 'regex' file
```

Afhankelijk van welk systeem je gebruikt, zal je output er licht anders uitzien. Standaard zal `grep` de lijnen uitprinten waar het bepaalde patroon is teruggevonden. Afhankelijk van je terminal kan het zijn dat `grep` in een andere kleur aangeeft waar het patroon exact is teruggevonden. Tussen de quotes zetten we de Reguliere Expressie waarnaar we op zoek zijn. Het laatste argument is de file waarin we willen zoeken.

Download de file `tekst_voor_grep.txt` van de MSDL-website en open een terminal in dezelfde map als de file. Om alle lijnen in de file weer te geven doe je

```
grep '.*' ./tekst_voor_grep.txt
```

```
hello world
hello world 1
hello world 2
...
hello earth
goodbye
```

Indien we enkel de lijnen willen die `hello` bevatten doen we

```
grep 'hello' ./tekst_voor_grep.txt
```

```
hello world
hello world 1
hello world 2
...
hello earth
```

We kunnen echter nog specifieker gaan door te specifieren dat we enkel de `hello world` lijnen willen die gevolgd worden door een cijfer:

```
grep 'hello world [0-9]' ./tekst_voor_grep.txt
```

```
hello world 1
hello world 2
hello world 3
```

Indien men een hele map met bestanden wilt doorzoeken naar een stuk tekst kan met de `-r` optie gebruiken. Grep zal dan *recursief* door de hele file-structuur zoeken. We nemen het volgende voorbeeld:

```
grep -E '(struct|class)' -r /usr/include
```

Met het bovenstaande commando openen we de map `/usr/include` en vinden we alle files die de woorden `class` en `struct` bevatten. Andere nuttige opties kun je vinden met het commando:

```
grep --help
```

Met het commando `sed` kunnen we een bepaalde Reguliere Expressie gebruiken om een bepaald patroon te zoeken en dit patroon te vervangen door een string. Het commando ziet er als volgt uit:

```
sed 's/from/to/g'
```

waarbij `s` staat voor het feit dat we willen substitueren, *`from`* is het te zoeken patroon, en *`to`* de string waarmee we de gevonden patronen vervangen. Het laatste deel bevat de mogelijke opties voor het commando. In dit geval is `g` (*global*) opgegeven, wat als resultaat heeft dat we alle voorkomens van het patroon gaan vervangen. Hieronder zie je drie voorbeelden waarbij letters, cijfers en speciale tekens vervangen werden met behulp van `sed`:

```
echo 'Ik woon in Gent' | sed 's/Gent/Antwerpen/g'
```

```
Ik woon in Antwerpen
```

```
echo 'Antwerpen heeft postcode 3000' | sed 's/3/2/g'
```

```
Antwerpen heeft postcode 2000
```

```
echo 'Ik ben geboren op 01-01-2001' | sed 's/-/./g'
```

```
Ik ben geboren op 01.01.2001
```

Het is ook mogelijk om gebruik te maken van ingewikkeldere patronen. Beschouw onderstaand voorbeeld:

```
echo '<dit is een >test>' | sed 's/[<>]//g'
```

```
dit is een test
```

Dit commando geeft de string `<dit is een >test>` door aan het commando `sed` en gaat vervolgens de karakters < en > verwijderen (vervangen door de lege string). Als resultaat krijgen we dan: `dit is een test`. Indien we alle cijfers willen verwijderen uit een string, kunnen we dit doen als volgt:

```
echo 'abc123def' | sed 's/[0-9]\+//g'
```

```
abcdef
```

Merk op dat de + operator hier voorafgegaan moet worden door een backslash. We kunnen `sed` echter ook toepassen op een file, net zoals we dat bij `grep` kunnen doen. We krijgen dan het volgende commando:

```
sed 's/from/to/g' file
```

Stel we willen nu uit een bepaalde string alle karakters '/' verwijderen. Dit zouden we doen door volgend commando:

```
echo '/dit is een test/' | sed 's////g'
```

```
sed: -e expression #1, char 5: unknown option to `s'
```

Zoals je kan zien is het nu echter onmogelijk om een onderscheid te maken tussen de verschillende onderdelen van het `sed` commando. We kunnen daarom de scheidingstekens (/) vervangen

door een ander karakter, belangrijk is dat we voor elke scheiding hetzelfde karakter gebruiken. We kunnen dan het commando veranderen in:

```
echo '/dit is een test/' | sed 's:/::g'
```

```
dit is een test
```

Waarbij de vier onderdelen wel duidelijk te onderscheiden zijn van elkaar. Net als in Reguliere Expressies kunnen we ook delen van gevonden patronen onthouden om dan opnieuw te gebruiken, we kunnen deze ook in de substitutie-string gebruiken. Willen we bijvoorbeeld alle tags met < en > omzetten naar | dan kunnen we volgend commando gebruiken:

```
echo '<hello>, <test>' | sed 's/<\([a-z]*\)>/|\1|/g'
```

```
|hello|, |test|
```

We kunnen ook verwijzen naar het volledige gevonden patroon door `&` te gebruiken. Stel we willen haakjes zetten rond een bepaalde string dan kunnen we dat op volgende manier doen:

```
echo 'hello' | sed 's/[a-z]*/(&)/g'
```

```
(hello)
```

Opdracht 9

Schrijf een `sed` commando dat in een tekst alle correcte HTML-tags eruit filtert, zodat de gewone tekst overblijft. Voor correcte HTML tags volstaat het dat de begin- en eindtag gelijk is. Om te testen kan je de file `test.html` gebruiken. Het resultaat zou dan "`Dit is vetgedrukte tekst en dit is belangrijke tekst.`" moeten zijn.