Computer Systems and -architecture

Reguliere Expressies

1 Ba INF 2021-2022

Stephen Pauwels
stephen.pauwels@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 commando's kunnen gebruiken.

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 een stap-voor-stap uitleg krijgen van de betekenis van de Reguliere Expressies.

1 Wat en Hoe?

Een Reguliere Expressie is een patroon dat een verzameling van strings beschrijft. We gebruiken deze patronen om heel eenvoudig textuele informatie op te zoeken (vb. alle e-mailadressen die in een gegeven file staan) of we kunnen deze ook gebruiken om ingegeven data te valideren (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 Character Sets

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

```
a
```

Deze expressie staat voor een enkel karakter '`a`'. Gebruik de standaard text die ingeladen wordt als je de web-tool opent en vul bovenstaande expressie in om te zien welke karakters gevonden worden. Strings (of deelstrings) die matchen met het opgegeven patroon worden aangeduid in de tekst. 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:

```
.
```

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 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. 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. Volgende expressie geeft enkel de punt-karakters terug.

```
\.
```

Merk het verschil op indien we wel of niet een backslash toevoegen. 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 semantische betekenis van karakters. Willen we, bijvoorbeeld, alle getallen van 0 tot en met 13 vinden dan zouden we dit op een naïve manier als volgt noteren:

```
[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 in de range tussen `0` en `1` ofwel het karakter `3`. We zullen later zien hoe we dit kunnen oplossen.

Opdracht 1

Geef een Reguliere Expressie die de volgende karakters kan vinden: `a b c d e f g`

3 Modifiers

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 nummers met 2 cijfers terug te vinden gebruiken we volgende expressie:

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

We kunnen ook het minimum en maximum aantal 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.

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, ....` 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.

Nu kan je onderstaande opdrachten maken via `regexr.com`, voeg telkens in het vak `Text` voldoende tests toe om na te kijken of je patroon voldoet aan de verwachtingen.

Opdracht 2

Geef een Reguliere Expressie die een getal op het begin van de lijn teruggeeft

Opdracht 3

Geef een Reguliere Expressie die filenames met de extensie ".py" teruggeeft

Opdracht 4

Geef een Reguliere Expressie die woorden van 4 karakters teruggeeft

Opdracht 5

Geef een Reguliere Expressie die elk nummer tussen 1 en 999 teruggeeft

Opdracht 6

Geef een Reguliere Expressie die url's met de extensie ".co.uk" teruggeeft

4 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 hebben tussen meerdere deelexpressies.

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

Bovenstaande code geeft exact alle nummers van 1 tot en met 13 terug. Het eerste deel `[1-9]` moet gevonden worden, ofwel het tweede `1[0-3]`.

Opdracht 7

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.

Opdracht 8

Geef een Extended Reguliere Expressie die geldige IPv4 adressen teruggeeft (0.0.0.0 tot 255.255.255.255)

5 Commando's: grep en sed

De echte kracht van de Reguliere Expressies hangt heel nauw samen met de 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 je `egrep` en `awk` gebruiken. Vanaf nu werken we telkens in een UNIX terminal voor de voorbeelden. Pas ook op als je de onderstaande commando's copy-paste in je terminal, de gebruikte quote's zijn niet degene die je terminal verwacht. Gebruik telkens enkele quote's voor je commando's. We kunnen `grep` op volgende manier gebruiken:

```
grep 'regex' file
```

Voor een concreet voorbeeld krijgen we dan:

```
grep '\<[a-z]\>' /etc/passwd
```

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.

Met het commando `sed` kunnen we een bepaalde Reguliere Expressie gebruiken om een bepaald patroon te zoeken en dit 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 substitureren, *`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. Beschouw nu onderstaand commando:

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

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`. 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'
```

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'
```

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'
```

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’
```

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.