#EurelisAcademy

CHEAT SHEET REGEX

La synthèse parfaite pour s'en sortir avec les expressions régulières...

eurelis
Digital Delivery Experts

eurelis
Digital Delivery Experts

CHEAT SHEET REGEX

#EurelisAcademy

Syntaxes

BASES

Caractères simples

x satisfait x
\t satisfait "tabulation"
\n satisfait "nouvelle ligne"
\r satisfait "retour chariot"
\f satisfait "saut de page
\e satisfait "échappement"

Caractères de limite

^ début de la ligne
$ fin de la ligne
\b coupure de mot (début ou fin)
- \beur mot commençant par eur ex : eurelis
- eur\b mot finissant par eur ex : agitateur
\B intérieur d'un mot
\Beur mot commençant pas par eur ex : agitateur
- eur\B mot ne finissant par eur ex : eurelis

Classes abrégées

. tous les caractères
\d équivalent de [0-9] digit
\D équivalent de [^0-9]
\s équivalent de [\t\n\f\r] space
\S équivalent de [^\s]
\w équivalent de [a-zA-Z_0-9] word
\W équivalent de [^\w]

GROUPES ET OPÉRATEURS

Groupes

groupe: sous-motif d'une regex

capturer x : (x)

ne pas capturer x :(? :x)
► *Utiliser des groupes non-capturant est plus performant*

référence arrière : \n
► <(b|i)>x</\1> satisfait x
► x et ne ne satisfait pas x

Opérateurs logiques

xy : x suivi de y
x|y : x ou y
x(?!y) : x non suivi de y (sans capturer y)
(.*)an(?!o)(.*) : mot qui contient an non suivi de o
► panier capture p comme groupe 1 et ier comme groupe 2
► piano ne satisfait pas l'expression

Captures

Suivre l'ordre des parenthèses
(anti)((con)sti(tu)tion)n(elle(ment))
- groupe 0: anticonstitutionnellement
- groupe 1: anti
- groupe 2: constitution
- groupe 3: con
- groupe 4: tu
- groupe 5: ellement
- groupe 6: ment
récupérer le nième groupe capturé
- en référence arrière (dans la regex) : \n
- en substitution (hors de la regex) : $n

QUANTIFICATEURS

Gourmands

*cherchent le nombre **maximal** de répétitions qui **autorisent** le succès de la recherche*
x? x 0 ou 1 fois
x* x 0 ou plusieurs fois
x+ x 1 ou plusieurs fois =xx*
x{n} x exactement n fois
x{n,} x au moins n fois
x{n,p} x au moins n fois mais pas plus de p fois

Réticents

*cherchent le nombre **minimal** de répétitions qui **autorisent** le succès de la recherche*
Même syntaxe que "gourmand" à laquelle on ajoute ?
si la chaîne est: Eurelis
gourmand : <a href=(.*)>
$1 : "http://www.eurelis.com">Eurelis</a
réticent : <a href=(.*?)>
$1 : "http://www.eurelis.com"

Possessifs

*cherchent le nombre **minimal** de répétitions mais **sans autoriser** le succès de la recherche*
Même syntaxe que "gourmand" à laquelle on ajoute +
si la chaîne est: Eurelis
possessif : <a href=(.*+)>
ne satisfait pas la regex
(.*+) commence par capturer "http://www.eurelis.com">Eurelis</a
et ne trouve pas de > après

Comparaisons

Si la chaine est abcdaefag
Gourmand : ^(.*)a([^a]+).*$
=> la chaîne satisfait la regex
le groupe 1 est abcdaef et le groupe 2 est g
Réticent : ^(.*?)a([^a]+?).*$
=> la chaîne satisfait la regex
le groupe 1 est *vide* et le groupe 2 est b
Possessif : ^(.*?)a([^a]++).*$
=> la chaîne satisfait la regex
le groupe 1 est vide et le groupe 2 est bcd

DRAPEAUX

Drapeaux courants

sont les options de l'expression régulière
g rechercher globalement, c'est-à-dire trouver toutes les occurrences
i ignorer la casse
m multiple lines, la chaîne de caractères peut comporter plusieurs lignes

Autres drapeaux

sont les options de l'expression régulière
c au cas où une erreur survient, ne pas réinitialiser la position de la recherche
o once, n'appliquer le modèle de recherche qu'une seule fois
s single line, considérer la chaîne de caractères comme une ligne distincte
x extended, utiliser la syntaxe étendue

En pratique - les langages

CODE JAVASCRIPT

Match

Vérifions en Javascript que notre chaîne de caractère s est bien un login contenant entre 5 et 12 caractères alphanumériques (y compris -)

```
if (/^[\w-]{5,12}$/gi.test(s)) { // traitement }
```

Extract

Récupérons en Javascript la valeur du port dans l'URL contenue dans notre chaîne de caractère s et choisissons 80 si pas trouvé

```
var regexExpression = "^https?://[^:]*:(\\d+)/.*$";
var regex= new RegExp(regexExpression, "gi");
var groups = regex.exec(s);
var port = groups ? groups[1] : 80;
// traitement
```

Substitute

Changeons en Javascript le protocole en HTTPS de la source d'une image dont le code HTML est contenu dans s

```
var regexExpression = "^[^:]+(://.*)$";
var regex= new RegExp(regexExpression, "gi");
s = s.replace(regex, "https$1");
```

CODE PHP

Match

Vérifions en PHP que notre chaîne de caractère s est bien un numéro de téléphone français (10 chiffres séparés d'un ., d'un - ou d'un espace)

```
if (preg_match("^0\\d([-. ]?\\d{2}){4}$", $s)) {
   // traitement
}
```

Extract

Récupérons en PHP la valeur du paramètre param1 de l'URL contenue dans notre chaîne de caractère s

```
String regex = "/.*?\\[?&]param1=([^&]+).*/";
preg_match(regex, $s, $groups, PREG_OFFSET_CAPTURE);
$param1Value = $groups[1];
```

Substitute

Changeons en PHP un nom de domaine dans l'URL contenue dans s

```
String regex = "^(https?://[^/])(/.*)$";
$s, = preg_replace(regex, "https://eurelis.com$2", $s);
```

CODE PYTHON

Pré-requis

Pour utiliser des expressions régulières en Python, il est nécessaire d'importer le package re:

```
import re
```

Match

Vérifions en Python que notre chaîne de caractère s contient bien 3 mots

```
import re
if re.search("^(\w+ ){3}$", s):
// traitement OK
else:
// traitement KO
```

Extract

Récupérons en Python un tableau de tous les mots contenus dans notre chaîne de caractère s

```
import re
groups = re.split("\s+", s);
// traitement
```

Substitute

Remplaçons en Python les 2 premières tabulations par des ; dans la chaîne de caractères s

```
import re
s = re.sub("\t", ";", s, 2);
```

CODE PERL

Match

Vérifions en Perl qu'une balise XML encadrant un texte est bien fermée dans notre chaîne de caractère s et contient au moins 2 attributs

```
if ( $s =~ m(<(\w+)(?:\s+[^=]+=\S*){2,}>[^<]*</\1>) ) {
   // traitement si OK
}
```

Extract

Trouvons en Perl un mot répété dans notre chaîne de caractère s

```
if ( $s =~ m((\w+)\W+\1, $s) ) {  my $word = "$1";
   // traitement si OK on récupère le mot dans $word
}
```

Substitute

Remplaçons en Perl le mot cat par le mot dog dans la chaîne de caractères s

```
$s =~ s/cat/dog/;        OU        $s =~ s#cat#dog#;
```

En fait, n'importe quel caractère peut être utilisé pour séparer, cela peut être utilse pour éviter de déspécialiser la regex.

Translation

C'est une substitution un peu particulière qui permet par exemple de remplacer les minuscules par des majuscules...

```
$s =~ tr/a-z/A-Z/;
OU
$s = 'food'; $s =~ tr/a-z/a-z/s; // => fod
```

... ou de supprimer des lettres en trop

CODE JAVA

Match

Vérifions en Java que notre chaîne de caractère s est bien un code postal *(5 chiffres)*

```
if (s.matches("\\d{5}$")) { // traitement }
```

Substitute

Changeons en Java un format de date simplement avec des expressions régulières

```
String newDateAsString = "2021-01-06".replaceAll("(\\d{4})-(\\d{2})-(\\d{2})", "$3/$2/$1"); // => 06/01/2021
```

Extract

Récupérons en Java l'attribut *href* d'un lien HTML dans notre chaîne de caractère s

```
String regex = "^<(a).*?href="([^"]+)"[^<]>[^<]+<\\1>$";
Matcher m = Pattern.compile(regex,
Pattern.DOTALL).matcher(s);
if (m.find()) {
   String href = m.group(2);
   // traitement
}
```

En pratique - les éditeurs

ÉDITEUR VI

Avant-propos

vi est souvent le seul éditeur de texte disponible sur un serveur linux. Savoir utiliser les REGEX est un réel atout d'autant que la syntaxe diffère quelque peu.
Les caractères spéciaux suivants doivent être échappés (avec \):
+ { ( ) |
Le ? est remplacé par \=
Les groupes sont constitués de \ et de leur numéro:
\1 \2 \3 ...

Match

On cherche un texte qui matche une regex avec /<regex>
Exemple de recherche d'un log du 11 septembre 2020 à 15h31:

```
11 Sep 2020 15:30:36,232  INFO [CmsLogReport: 113] ( 1 ) Indexing file /sh
11 Sep 2020 15:30:36,234  INFO [eIndexWriter: 118] Committing changes to s
11 Sep 2020 15:30:36,262  INFO [CmsLogReport: 113] [ ... finished updating
11 Sep 2020 15:31:06,263  INFO [CmsLogReport: 113] [ Updating search index
11 Sep 2020 15:31:06,329  INFO [CmsLogReport: 113] ( 1 ) Indexing file /sh
11 Sep 2020 15:31:06,374  INFO [CmsLogReport: 113] ( 2 ) Indexing file /sh
11 Sep 2020 15:31:06,421  INFO [CmsLogReport: 113] ( 3 ) Indexing file /sh
11 Sep 2020 15:31:06,482  INFO [CmsLogReport: 113] [ ... finished updating
11 Sep 2020 15:40:06,879  INFO [CmsLogReport: 113] [ Updating search index
11 Sep 2020 15:40:06,917  INFO [CmsLogReport: 113] ( 1 ) Indexing file /sh
11 Sep 2020 15:40:06,979  INFO [CmsLogReport: 113] [ ... finished updating
11 Sep 2020 15:40:06,979  INFO [CmsLogReport: 113] [ Updating search index
```

Substitute

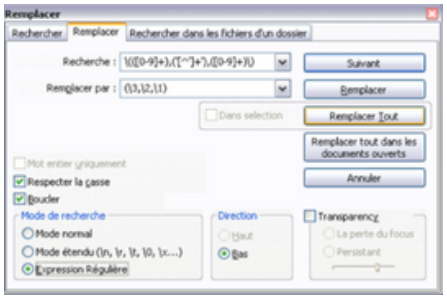
On remplace une chaîne par : %s:<regex>:<replacement>:<flags>
Exemple de remplacement pour autoriser l'IP 82.62.125.78 dans un VHost Apache:

```
<Directory /home/mysite/www>
Require all granted
Options FollowSymLinks MultiViews
AllowOverride All

Order deny allow
deny from all
allow from 78.243.43.69
</Directory>

:%s:78.243.43.69:78.243.43.69 82.62.125.78:g
```

ÉDITEURS DE TEXTE NOTEPAD++, VISUAL STUDIO CODE, ECLIPSE

Notepad ++

Ctrl + F Pour lancer la recherche et sélectionner l'onglet Remplacer

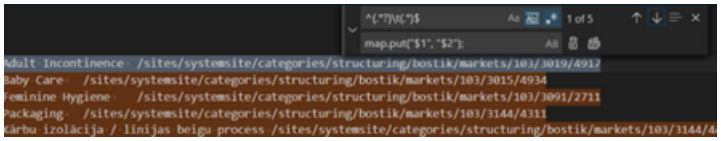


MATCH : cliquez sur suivant

SUBSTITUTE : cliquez sur Remplacer ou Remplacer tout

Visual Studio Code

Ctrl + F Pour lancer la recherche



MATCH : cliquez sur suivant ↓ ou précédent ↑

SUBSTITUTE : cliquez sur Remplacer ou Remplacer tout

Eclipse

Ctrl + F Pour lancer la recherche

MATCH : cliquez sur Find

SUBSTITUTE : cliquez sur Replace all

Conclusion

RÉSUMÉ ET BIBLIOGRAPHIE

Résumé

Une bonne expression régulière est une expression régulière (Règle des **3C**):
Concise : écrire le strict minimum pour que cette expression régulière soit nécessaire et suffisante
Claire : être capable de comprendre facilement cette expression régulière, même en la regardant longtemps après
Correspondant précisément au besoin : bien définir le besoin (les règles) et pas d'erreur possible en fonction de différentes chaînes de caractères en entrée

Bibliographie

Test en ligne d'expressions régulières:
https://ap2cu.com/tools/regex/
http://www.regexplanet.com/advanced/java/index.html

Cours et exemples :
Perl: https://www.tutorialspoint.com/perl/perl_regular_expressions.htm
Python: https://docs.python.org/fr/2.7/howto/regex.html
Java:
https://docs.oracle.com/javase/9/docs/api/java/util/regex/Pattern.html
PHP: https://www.php.net/manual/fr/ref.pcre.php
Javascript: https://javascript.info/regexp-methods

Générateur d'Expressions Régulières :
http://www.txt2re.com/index-java3.html

Bonus

Tester la primalité (=si un nombre est premier) juste avec une REGEX et en Javascript:

```
var n = 5;
var regex = /^1?$|^(11+?)\1+$/;
alert( n +
(!regex.test((function(n){
s = '';
while(n--) s += '1';
return s;
})(n)) ?
" is prime" : " is NOT prime"
));
```