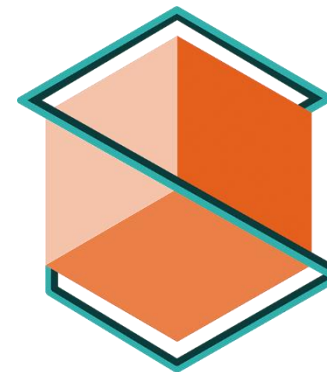


Expressions régulières



Semifir

contact@semifir.com
13 Avenue du Président John F. Kennedy,
59000 Lille.

Les RegEx



A top-down view of a wooden desk. In the upper left, a silver laptop is open, showing its keyboard and trackpad. To the right of the laptop is a white computer mouse. Further right is a white ceramic cup filled with dark coffee. In the bottom right corner, there is a chocolate muffin. In the bottom left corner, a portion of a black smartphone is visible. A yellow rectangular box is positioned on the left side of the image, partially overlapping the laptop and the text.

Introduction aux Expressions Régulières

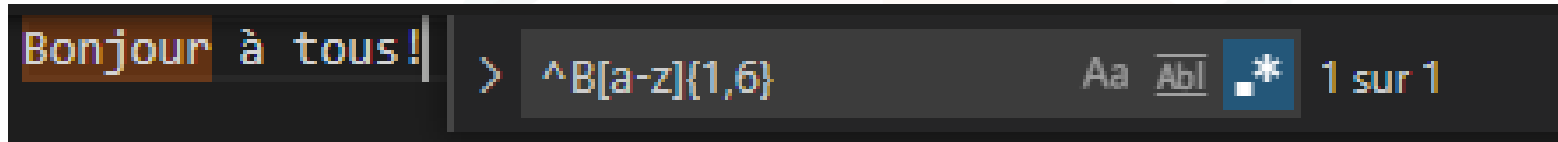
Introduction

- Une expression régulière (ou regex, issu de Regular Expression) est une chaîne de caractères qui décrit un pattern (modèle) de chaînes de caractères possibles.
- Ces regex permettent de cibler des chaînes de caractères très précises afin de les extraire, contrôler, remplacer...

Semifir

Introduction

- La plupart des éditeurs de texte permettent la recherche grâce aux RegEx.



- Les langages ont implémenté les RegEx afin de travailler sur les chaînes de caractères

```
Pattern regex = Pattern.compile("^B.{1,6}");  
Matcher phrase = regex.matcher(input: "Bonjour");
```

```
Regex regex = new Regex(@"^B\w{1,6}");  
string phrase = "Bonjour à tous!";  
MatchCollection matches = regex.Matches(phrase);
```

```
let str = "Bonjour à tous!"  
let match = str.match(/^B.{1,6}/);
```

Utilisation

- Une Regex est construite en fonction de la chaîne de caractères que nous souhaitons rechercher.
- Ainsi, la représentation finale concatène les différentes recherches et renvoie les éléments correspondants.

Semifir

Utilisation

- Les caractères de début et de fin de chaîne.
 - Ils sont respectivement représentés par ^ et \$
 - Exemple:

```
String phrase = "Bonjour à tous et bonjour à toi";  
Pattern regex = Pattern.compile("regex: ^b[a-z]{1,6}", Pattern.CASE_INSENSITIVE);
```

Nous retournera le premier « Bonjour » uniquement tandis que

```
String phrase = "Bonjour à tous et bonjour à toi";  
Pattern regex = Pattern.compile("regex: b[a-z]{1,6}", Pattern.CASE_INSENSITIVE);
```

Nous retournera les deux

Semifir

Utilisation

- Le caractère OU
 - Représenté par le caractère |
 - Permet de retourner les éléments qui correspondent à un pattern OU un autre
 - Exemple:

```
String phrase = "Bonjour à tous et bonjour à toi";  
Pattern regex = Pattern.compile("regex: bonjour|toi", Pattern.CASE_INSENSITIVE);
```

Nous retournera: Bonjour, bonjour, toi

Semifir

Utilisation

- Les ensembles de caractères

[abc]

Soit a, soit b, soit c

[^abc]

Exclusion de a,b et c

[a-z]

Lettres minuscules entre a et z

[A-Z]

Lettres majuscules entre A et Z

[0-9]

Chiffres entre 0 et 9

- Exemple:

" t[a-z]t[a-z]

TEST STRING

toto	↵
tata	↵
titi	↵
tato	↵
tuta	↵
tztp	↵

Nous retourne tous les éléments

REGULAR EXPRESSION

" t[a-u]t[a-z]

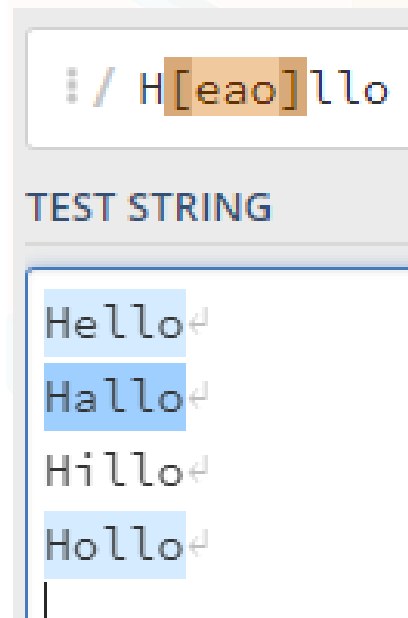
TEST STRING

toto	↵
tata	↵
titi	↵
tato	↵
tuta	↵
tztp	↵

Exclut le dernier

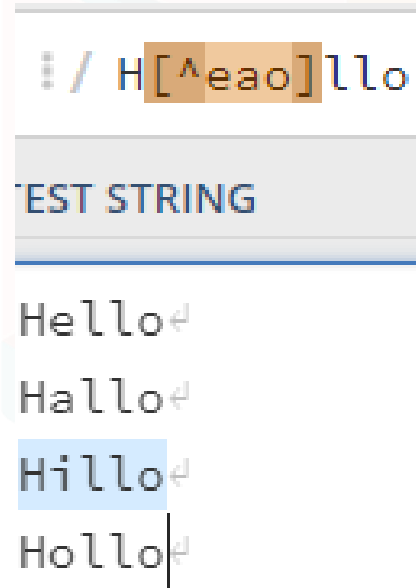
Utilisation

- Les possibilités par ensembles
 - Il est possible de définir une liste de caractères possibles, dans ce cas nous spécifions simplement lesquels entre []
 - Exemple:



Utilisation

- Les possibilités par ensembles
 - Il est également possible d'indiquer que tous les caractères sont acceptés, SAUF ceux spécifiés:



The screenshot shows a regex testing interface. At the top, the regex pattern `H[^eao]llo` is entered. Below it, a button labeled "TEST STRING" is visible. Underneath the button, four test strings are listed: "Hello", "Hallo", "Hillo", and "Hollo". The string "Hillo" is highlighted in blue, indicating it is the only one that matches the pattern. The other strings ("Hello", "Hallo", "Hollo") are not highlighted, indicating they do not match.

Le caractère ^ nous sert ici à indiquer la condition inverse

Utilisation

- Les ensembles préconçus
 - Du fait de leur utilisation fréquente, certains ensembles sont préconçus afin de faciliter l'utilisation et la lecture

.

\w

\d

N'importe quel caractère

Équivaut à [a-zA-Z0-9_]

Équivaut à [0-9]

t[a-u]t\w

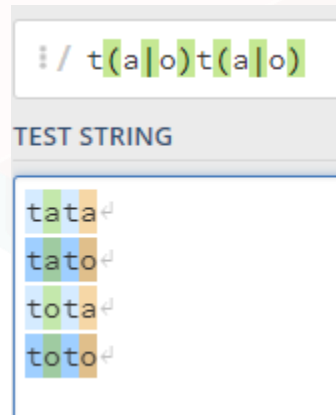
t[a-u]t.

t[a-u]t[a-z]

Nous retourneront tous les 3 les mêmes résultats dans le contexte de la démonstration précédente.

Utilisation

- Grouper les conditions
 - Il est possible de regrouper plusieurs conditions afin qu'elles soient alternatives et non cumulatives par exemple.
 - Ainsi, si nous voulons nous assurer que le caractère est soit a soit o, il est possible d'écrire:



The screenshot shows a code editor with a regular expression `/t(a|o)t(a|o)/` in the top bar. Below it, a section labeled "TEST STRING" contains a list of four strings: "tata", "tato", "tota", and "toto". Each string is preceded by a small colored square (green for "tata" and "toto", blue for "tato" and "tota") indicating the match result.

- Ou encore [ao]

Utilisation

- Les flags
 - Il est possible de préciser certaines options:
 - g : continue la recherche après la première occurrence trouvée
 - m: la recherche se fait sur plusieurs lignes(un saut de ligne est une fin de chaîne)
 - i: insensible à la casse
 - D'autres sont disponibles, il s'agit ici des plus courantes.

Semifir

Utilisation

- Les quantificateurs
 - Comment s'assurer de ne pas avoir à déterminer chaque caractère indépendamment, notamment si ils présentent les mêmes caractéristiques?
 - Nous utiliserons les quantificateurs, qui permettent de définir une répétition.
 - Les quantificateurs se placent entre {} et permettent de définir un nombre précis ou une fourchette de répétitions:
 - {nombre} : nombre précis de répétitions
 - {min, max} : valeur minimale et maximale
 - {min,} : entre la valeur minimale et l'infini

Utilisation

- Les quantificateurs

- Il est également possible d'utiliser le « + », qui permet de répéter l'élément précédent un nombre illimité de fois: (mais au moins une fois)

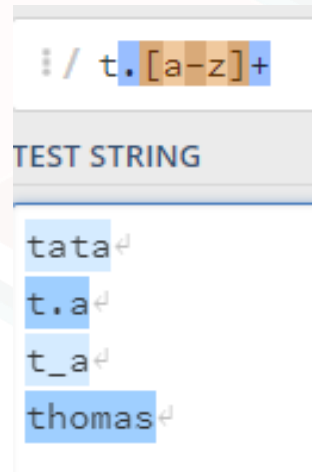
```
t[a-z]+
```

- Ou encore le « * », qui lui permet de quantifier la répétition mais l'objet peut ne pas être présent.

Semifir

Utilisation

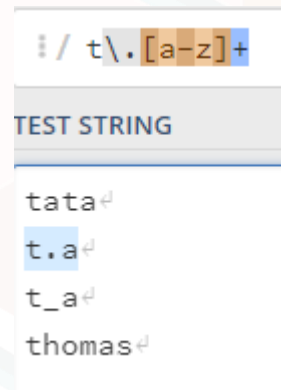
- L'échappement de caractères
 - Imaginons que nous souhaitons récupérer un nom d'utilisateur.
 - Celui-ci DOIT comporter un point.
 - Le problème est que dans le contexte, le point représente tout caractère:



Semifir

Utilisation

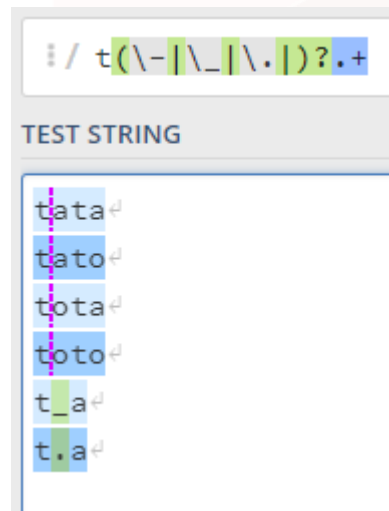
- L'échappement de caractères
 - Dans ce cas, il est possible d'échapper le point, afin qu'il soit bel et bien considéré comme le caractère correspondant:



Semifir

Utilisation

- La présence conditionnelle de caractère
 - Imaginons à présent que nous voulions récupérer un nom d'utilisateur qui commence par un t puis comporte SOIT un _ , SOIT un . , SOIT un – OU aucun des 3.
 - Il nous suffira d'écrire:



Exercices

- Le numéro de téléphone:
 - Ecrivez une RegEx qui permet de déterminer si un numéro de téléphone respecte le format attendu.
 - Ainsi, les numéros suivants doivent fonctionner:
 - 0607080910
 - +33607080910
 - 06.07.08.09.10
 - 06 07 08 09 10
 - Mais pas ceux-ci:
 - +330607080910
 - 060708091011
 - 5568711100
 - +21487811011
 - +33607080910' OR 1=1

Exercices

- L'e-mail:
 - Ecrivez une RegEx qui permet de déterminer si une adresse mail respecte le schéma d'une réelle adresse
 - Les adresses suivantes doivent fonctionner:
 - `thomas@semifir.com`
 - `thomas.timio@social4.microsoft-fr.com`
 - jean-jean.jean@jean.it
 - jean_jean@jean-jean.fr
 - toto@kc.cc
 - t1@gm.com
 - `c_ou_cou@gmail.com`
 - Mais pas celles-ci:
 - `jean@jean`
 - `jean@jean.com'` OR `1=1 #`
 - 1@kc.cc
 - tho#mas@gmail.com
 - `test@nom.n`