Introduction aux expressions régulières (regexp).

Les expressions régulières, ou plus communément regex (contraction de *regular expression*) permettent de représenter des modèles de chaînes de caractère. Ce sont des outils très puissants et très utilisés : on peut les retrouver dans de nombreux langages comme le PHP, MySQL, Javascript... ou encore dans des logiciels d'édition de code ! Cependant, si cet outil est très puissant, il est relativement difficile à appréhender au début car les expressions régulières peuvent prendre des formes de ce genre :

```
#^[a-zA-Z-]+@[a-zA-Z-]+\.[a-zA-Z]{2,6}$#
```

Cet expression régulière permettant, par exemple, de représenter toutes les adresses emails contenues dans un texte. Vous pouvez donc voir que la syntaxe est peu commune et qu'un petit tutoriel s'impose pour pouvoir comprendre et utiliser le concept des expressions régulières. Mais avant de se plonger dans la théorie, voici quelques exemples :

`hello`	contains {hello}
`gray|grey`	contains {gray, grey}
`gr(a|e)y`	contains {gray, grey}
`gr[ae]y`	contains {gray, grey}
`b[aeiou]bble`	contains {babble, bebble, bibble, bobble, bubble}
`[b-chm-pP]at|ot`	contains {bat, cat, hat, mat, nat, oat, pat, Pat, ot}
`colou?r`	contains {color, colour}
`rege(x(es)?|xps?)`	contains {regex, regexes, regexp, regexps}
`go*gle`	contains {ggle, gogle, google, gooogle, goooogle, ...}
`go+gle`	contains {gogle, google, gooogle, goooogle, ...}
`g(oog)+le`	contains {google, googoogle, googoogoogle, googoogoogoogle, ...}
`z{3}`	contains {zzz}
`z{3,6}`	contains {zzz, zzzz, zzzzz, zzzzzz}
`z{3,}`	contains {zzz, zzzz, zzzzz, ...}
`[Bb]rainf\*\*k`	contains {Brainf**k, brainf**k}
`\d`	contains {0,1,2,3,4,5,6,7,8,9}
`\d{5}(-\d{4})?`	contains a United States zip code
`1\d{10}`	contains an 11-digit string starting with a 1
`[2-9]|[12]\d|3[0-6]`	contains an integer in the range 2..36 inclusive
`Hello\nworld`	contains Hello followed by a newline followed by world

Les caractères de début et fin de chaîne

Les caractères de début et de fin de chaîne, respectivement ^ et $, représentent, comme leur nom l'indique, le début et la fin de la chaîne.

Le caractères OU

Imaginons maintenant que nous voulons rechercher, dans le texte, les mots *Bonjour* et *revoir*, c'est à dire le mot *Bonjour* OU le mot *revoir* : nous devrons alors utiliser la barre verticale |. Ainsi la regex suivante sélectionnera toutes les occurrences de *Bonjour* et *revoir* :

```
Bonjour|revoir
```

Il est aussi possible de sélectionner les occurrences du mot *Bonjour* se trouvant au début du texte et du mot *revoir* se trouvant à la fin, ce qui revient à fusionner, en les séparant par une barre verticale, les 2 regex vues dans la sous partie précédente. Voici donc la regex correspondant :

```
^Bonjour|revoir$
```

Les ensembles de caractères

Passons maintenant à des regex un peu plus complexes, et beaucoup plus puissantes, en nous attardons sur les ensembles de caractères.

Les différents ensembles

Maintenant, nous voulons chercher dans notre texte, les mots *mots*, *mats* et *mits*. Nous pourrions très bien utiliser cette regex :

```
mots|mats|mits
```

Mais il y a plus simple et ce grâce aux ensembles de caractères qui font office, en quelques sortes, de OU en plus courts et plus puissants. Un ensemble de caractère est délimité par des crochets dans lesquels se trouvent les caractères faisant parti du OU. Ainsi, la regex suivante :

```
m[oai]ts
```

est beaucoup plus succincte que la précédente et sélectionne les mêmes mots. Cette regex peut être expliciter par la phrase suivante : "Sélectionne les parties du texte où il y a un m, suivi d'un o ou d'un a ou d'un i, suivi d'un t, suivi d'un s." Les ensembles de caractères permettent aussi d'exclure des caractères grâce à l'accent circonflexe ^. La regex suivante :

```
m[^oai]ts
```

sélectionnera, cette fois-ci, seulement le mots *mets* et peut être explicitée par la phrase suivante : "Sélectionne les parties du texte où il y a un m, suivi d'une lettre qui n'est ni o, ni a, ni i, suivi d'un t, suivi d'un s." Enfin, imaginons que nous voulons sélectionner tous les mots commençant par un m, suivi de n'importe quelle lettre, suivi d'un t, suivi d'un s. La regex qui nous viendrait à l'esprit serait une regex de ce type :

```
m[abcdefghijklmnopqrstuvwxyz]ts
```

La regex serait donc longue et fastidieuse à écrire, surtout que pour celle-ci, seules les minuscules ont été sélectionnées ! Heureusement, un moyen plus simple existe pour écrire de telles regex : cela s'appelle les intervalles et se note [*debut intervalle-fin intervalle*]. Voici quelques petites exemples :

Intervalle	**Equivalent**	**Traduction**
[a-z]	[abcdefghijklmonpqrstuvwxyz]	Lettres minuscules de a à z
[A-Z]	[ABCDEFGHIJKLMNOPQRSTUVWXYZ]	Lettres majuscules de A à Z
[0-9]	[0123456789]	Chiffres de 0 à 9
[a-z0-9]	[abcdefghijklmonpqrstuvwxyz0123456789]	Lettres minuscules de a à z ou chiffres de 0 à 9

Les ensembles préconçus

Un ensemble préconçu est une façon très simple de représenter plusieurs intervalles. Voici quelques exemples :

Ensemble	Equivalent
.	Absolument n'importe quel charactère
\w	[a-zA-Z0-9_]
\d	[0-9]
\n	Un retour à la ligne
\t	Une tabulation

Ainsi, cette regex :

```
m\wts
```

sélectionnera les mêmes éléments que celle-ci :

```
m[a-zA-Z0-9_]ts
```

Les quantificateurs

Nous venons de voir qu'un ensemble de caractères permet de définir de manière très simple les valeurs possible d'un caractère. Mais qu'en est-il si l'on définir les mêmes valeurs possibles pour plusieurs caractères ? Par exemple, si l'on veut sélectionner les parties du texte où il y a un m, suivi d'un a, suivi de 3 fois n'importe quelle lettre minuscule, est-on obligé d'utiliser une regex de ce type :

```
ma[a-z][a-z][a-z]
```

Non. Il existe une méthode plus simple qui consiste à utiliser les quantificateurs : ce sont des caractères qui indiquent le nombre de répétition du caractère ou de la suite de caractère qui les précèdent. Le quantificateur, dans sa forme explicite, peut s'écrire de 4 façons :

- {min,max} : le nombre de répétition varie entre la valeur minimale et la valeur maximale incluses
- {min,} : le nombre de répétition varie entre la valeur minimale incluse et l'infini
- {,max} : le nombre de répétition varie entre 0 et la valeur maximale incluse
- {nombre} : le nombre de répétition correspond au nombre marqué entre les accolades

Par exemple, la regex suivante :

```
[a-zA-Z]{6}
```

permet de sélectionner les parties du texte où il y a 6 lettres consécutives. Celle-ci :

```
[0-9]{2,4}
```

permet de sélectionner les parties du texte où il y a entre 2 et 4 chiffres consécutifs. Comme pour les ensembles de caractères, il existe aussi des quantificateurs préconçus. En voici la liste :

Quantificateur	**Traduction**	**Équivalent**
*	0 ou plusieurs répétitions	{0,}
+	1 ou plusieurs répétitions	{1,}
?	0 ou 1 répétition	{,1}

L'échappement

Reprenons notre texte et imaginons que l'on veuille y sélectionner les noms de domaine des adresse email *contact@johndoe.fr* et *contact@johndoe.com*. La regex qui nous viendrait donc à la tête serait la suivante :

```
johndoe.[a-z]{2,3}
```

Et malheureusement, ce n'est pas exactement celle-là, car même si les 2 noms de domaine sont bien sélectionnés, la chaîne de caractère *johndoe-blo* l'est aussi.

Ce problème vient alors du point présent dans la regex car rappelez-vous que le point est un ensemble de caractères préconçus qui représente n'importe quel caractère : il peut représenter un a, un 2 mais aussi un tiret. Pour faire comprendre que le point présent dans la regex est bien un point et non pas un ensemble de caractères, il nous faut échapper le point avec le caractère d'échappement qui est le backslash \. Ainsi la regex correcte est la suivante :

```
johndoe\.[a-z]{2,3}
```

Cet échappement n'est pas seulement valable pour le point, mais pour tous les caractères qui ont, de base, une valeur différente que celle habituelle. En voici la liste : ^ $ \ | { } [ ] ( ) ? # ! + * .