EXPRESSIONS RÉGULIÈRES

Petit aide-mémoire

Introduction

Les **expressions régulières** (*regular expression* ou ***regex*** en anglais) ou parfois aussi *expressions rationnelles* permettent de décrire un ensemble de caractères au moyen d'un **motif** (*pattern* en anglais). Ces expressions permettent de faire des **recherches** ou des **remplacements** avec des motifs complexes.

Métacaractères

.	^	$	*
+	?	{	}
[	]	\	|
(	)		

Le métacaractère `\` permet d'échapper les métacaractères.

Ancrages

- `^` : début d'une ligne (multilignes)
- `$` : fin d'une ligne (multilignes)
- `\A` : début d'une chaine de caractères
- `\Z` : fin d'une chaine de caractères
- `\b` : limite d'un mot (début ou fin)
- `\B` : opposé de `\b`

Classes de caractères

- `.` : n'importe quel symbole
- `[0-9]` ou `\d` : chiffres de *0* à *9*
- `[^0-9]` ou `\D` : tout sauf les chiffres de *0* à *9*
- `[aeiou]` : les lettres *a, e, i, o* et *u*
- `[^,.!?]` : tout sauf les symboles `,.!?`
- `[a-z]` : lettre minuscule *a* à *z*
- `\s` : tous les espaces
- `\S` : tout sauf les espaces
- `\w` : lettres, chiffres et `_`
- `\W` : tout sauf aux lettres, chiffres et `_`

Standard *POSIX*

- `[:upper:]` : Lettres majuscules
- `[:lower:]` : Lettres minuscules
- `[:alpha:]` : Lettres majuscules et minuscules
- `[:alnum:]` : Lettres et chiffres
- `[:digit:]` : Chiffres (correspond à `\d` ou `[0-9]`)
- `[:punct:]` : Ponctuation

Quantifieurs

- `?` : 0 ou 1 fois
- `*` : 0 fois ou plus
- `+` : 1 fois ou plus
- `{6}` : 6 fois (exactement)
- `{3,5}` : 3, 4 ou 5 fois
- `{4,}` : 4 fois ou plus

Groupes et opérateur *OU*

Les métacaractères `(` et `)` permettent de définir un groupe dans un motif.

- `(bug)` : groupe *capturant* avec le motif `bug`
- `(?:bug)` : groupe *non capturant* avec le motif `bug`

Un groupe *capturant* peut être référencé dans un remplacement avec `\1` pour faire référence au premier groupe, `\2` le deuxième groupe et ainsi de suite.

Le métacaractère `|` fait office d'opérateur logique *OU*. Par exemple :

- `m(a|i)c` : motif qui correspond à *mac* ou à *mic*
- `(jour|nuit)` : motif qui correspond à *jour* ou à *nuit*

Ressources utiles

Références

- Guides et références (en anglais) :
  https://www.regular-expressions.info
- Guides et références (axé sur le langage PHP, en français) :
  http://www.expreg.com
- Exemples d'expressions fréquentes :
  https://projects.lukehaas.me/regexhub/
- Documentation *regex* de *Python* 🐍 (en français) :
  https://docs.python.org/fr/3/howto/regex.html

Moteurs d'expressions régulières

- https://regexr.com
- https://regex101.com
- https://debuggex.com

Python 🐍 et le module *re*

Le module *re* permet d'utiliser les expressions régulières dans *Python* 🐍. Pour l'importer :

```
import re
```

Méthodes

- re.**findall**(*pattern, string, flags=0*)
  retourne toutes les occurences de `pattern`, retourne une liste
- re.**split**(*pattern, string, maxsplit=0, flags=0*)
  divise `string` à partir de `pattern`, retourne une liste
- re.**sub**(*pattern, repl, string, count=0, flags=0*)
  remplace `pattern` par `repl` dans `string`, retourne une chaine de caractères

`pattern` : expression régulière
`repl` : chaine de caractères de remplacement
`string` : chaine de caractères dans laquelle chercher le motif
`maxsplit` : nombre maximal de morceaux
`count` : nombre maximal de remplacements à effectuer
`flags` : options spécifiques aux expressions régulières

Exemples

```
re.findall(r"\d+", "Un ordinateur fait autant d'erreur en
        2 secondes que 20 humains en 20 ans.")
```

```
## ['2', '20', '20']
```

```
re.split(r"/", "un/deux/trois/quatre/cinq/six/sept")
```

```
## ['un', 'deux', 'trois', 'quatre', 'cinq', 'six', 'sept']
```

```
re.sub(r"à[^,]*", "aux expressions régulières", "Quand on
        se met à l'informatique, il vaut mieux avoir
        BEAUCOUP d'amis.")
```

```
## 'Quand on se met aux expressions régulières, il vaut
mieux avoir BEAUCOUP d'amis.'
```