Cours 21-22 : Expressions régulières

Introduction à la programmation Web

Expressions régulières

- Lorsqu'il faut valider le contenu venant d'un usager, notamment d'un formulaire, il est souvent nécessaire de trouver un algorithme qui vérifier qu'une chaîne de caractères suit un format particulier.
    - Par exemple, un code postal canadien doit absolument contenir une alternance de lettres et de chiffres, et certaines lettres ne sont pas permises…
- Ces algorithmes peuvent devenir passablement complexes pour des tâches qui pourraient sembler simples à première vue.
- Heureusement, il existe une syntaxe permettant de spécifier un **modèle** de chaîne de caractères : les **expressions régulières**.

Expressions régulières (suite)

- Une expression régulière est un modèle (ou *masque*) rédigé dans un langage formel qui, lorsque appliqué à une chaîne *sujet*, de gauche à droite, permet de retrouver ce masque à l'intérieur de la chaîne sujet. On peut donc utiliser les expressions régulières pour trouver une chaîne dans un texte, ou pour valider le format d'une chaîne.
- Celles-ci ont l'avantage de fonctionner de la même manière dans plusieurs langages (PHP, JS, HTML, etc.) et utilisent la même syntaxe.
- Elles peuvent être utilisées pour plusieurs raisons :
  - Trouvez des courriels dans un texte
  - Valider un code postal, un numéro de téléphone, un courriel
  - Extraire de l'information d'une page Web qui suit un template
  - …
- Il y a deux syntaxes utilisées. Nous utiliserons la syntaxe PERL.

Apprendre et tester les “regex”

Avant d’utiliser les expressions régulières dans notre code PHP, il est bon de pouvoir les tester.

Voici deux sites vous permettant de tester des expressions régulières dans votre fureteur, ce qui est aussi parfait pour apprendre!

https://www.regexpal.com/

https://regexr.com/

Regex : les littéraux

- La plus simple expression régulière possible serait constituée d'un seul caractère, qui serait analysé de façon "littérale".
  - Exemple de masque : a
  - Sur la chaîne "Jack a un chat", le masque 'a' trouverait le caractère a situé dans Jack.
  - Il sera possible dans la plupart des langages de programmation de continuer à chercher après une première recherche pour trouver le prochain résultat.
- Il est aussi possible d'utiliser un masque qui possède plusieurs "littéraux".
  - Le masque "chat" trouverait, évidemment, le mot chat. Mais pas le mot Chat!
- La plupart des caractères sont analysés de façon "littérale" : c'est le cas des lettres minuscules et majuscules ainsi que des nombres et de plusieurs autres caractères spéciaux.

Regex : les caractères spéciaux

- Plusieurs caractères spéciaux sont utilisés pour créer des expressions régulières, et doivent être “échappés” si l’on désire les retrouver dans une chaîne. C’est le cas de [,], \, ^, $, le point . , |, ?, *, + et ( ).
- Ce sont des métacaractères.
- Pour les trouver, il faut les échapper avec le backslash.
- Le backslash ne doit pas être utilisé pour les autres caractères, à moins que l’on veuille utiliser les fonctionnalités qui s’y rattachent (\t, \n, etc., et les autres que nous verrons plus tard)

Regex : les ancres

- Dans le cas de la validation, on ne cherche pas seulement à trouver un *pattern* dans l'entrée de l'usager, mais à s'assurer qu'il n'y *absolument rien d'autre* dans cette entrée que le *pattern* recherché.
- Pour s'en assurer, on utilise les ancres. Ils vous permettent de spécifier dans votre masque le début de la chaîne à tester avec le caractère ^ et la fin de la chaîne à tester avec le caractère $.
- Par exemple, le masque a$, testé avec la phrase "ca va", trouvera le dernier a, et pas le premier. Testé dans la phrase "ca ne va pas", il ne trouvera rien, puisque le a n'est pas à côté de la fin de la chaîne.
- Si on utilise les deux ancres, on cherche donc un masque et seulement ce masque.
    - Ex : ^chat$ fonctionnerait donc pour "chat", mais pas pour "mon chat"

Regex : classes de caractères

- Lorsqu'on est flexible sur les caractères que l'on accepte de retrouver dans la chaîne, on utilise les classes de caractères. Elles sont sont dénotées par les caractères [et].
- Par exemple, le masque "m[aio]che" va trouver les mots miche, mache et moche.
- Il est possible de mettre un tiret à l'intérieur d'une classe pour dénoter un intervalle de chiffre ou de lettre.
  - Le masque 1[0-9] trouvera 10, 11, 12, 13, 14...19.
    - (N'oubliez pas, c'est des caractères, pas des nombres! On ne peut pas écrire 10-20 et penser que ce sera les nombres entre 10 et 20...)
  - Le masque [A-Fa-f] trouvera a, A, b, B, c, C jusqu'à f, F.
- Il est possible dans une classe de faire la négation d'un caractère en utilisant ^.
  - ▫ Le masque [^a] trouvera n'importe quel caractère sauf a.

Regex : classes de caractères (suite)

Tous les métacaractères mentionnés précédemment peuvent être utilisés à l'intérieur d'une classe de caractères sans les échapper, à l'exception de ceux qui ont une signification dans les [ ], donc [,], - et ^, et du backslash qui sert à échapper.

Quelques classes de caractères souvent utilisées ont des raccourcis. Les voici :

- \s, qui représente tous les caractères d'espace, soit généralement la classe [\t\r\n].
- \w, qui représente tous les caractères qui peuvent être dans un mot, soit [A-Za-z0-9_].
- \d, qui représente tous les chiffres, soit [0-9]
- \b, qui représente les limites d'un mot (soit le début d'un texte, la fin d'un texte, la position entre un caractère qui ne fait pas partie d'un mot et un caractère qui fait partie d'un mot.
- Chaque raccourci possède sa négation, soit \W, \S et \D.

Regex : le point

Le point permet de trouver n’importe quel caractère.

Exemple : Le masque ab.c trouvera n’importe quelle chaîne de la forme abac, abbc, ab3c, ab:c, ab/c, etc.

Regex : répétitions

- Trois éléments peuvent être utilisés pour tester la *présence* d'un élément et ses répétitions.
  - ? teste pour la présence d'un élément ou d'une classe 0 ou 1 fois
  - * teste pour la présence d'un élément ou d'une classe 0 ou plusieurs fois
  - + teste pour la présence d'un élément ou d'une classe 1 ou plusieurs fois
- Il faut par contre faire attention avec ces opérateurs : ils sont gourmands. Pourquoi? Ils vont toujours chercher le nombre maximum de répétition possible dans votre masque, puisqu'ils commencent par tenter de faire le plus de répétitions possible.
  - Exemple : Vous voulez trouver les tags HTML dans un texte. Vous décidez d'utiliser <.+> pour trouver toutes les chaînes du type , , , etc, puisque le point peut être n'importe quelle lettre. Le problème est que . peut aussi être > et <, et que les opérateurs de répétition sont gourmands.
  - Avec la phrase "Ceci est un premier test.", votre masque devrait trouver **** et **** , mais le résultat trouvé sera **premier**.
- Pour empêcher ces opérateurs d'être gourmand, on peut utiliser le point d'interrogation après l'opérateur.
  - <.+?> trouverait
  - On pourrait aussi procéder de façon plus précise…
  - <[^>]+> trouverait

Regex : répétitions

Il est aussi possible d'utiliser les accolades { } pour spécifier un nombre de répétitions précis que l'on désire avoir, ou un éventail du nombre de répétitions.

Par exemple **a{4}** cherchera 4 a consécutifs, alors que **[a-z]{1,3}** va chercher entre un et trois caractères minuscules entre a et z.

Regex : alternatives

- Vous pouvez utiliser | dans votre expression régulière pour créer un masque avec des alternatives.
- Le masque chat|chien trouver le mot chat ou le mot chien dans votre chaîne.
- Pour délimiter les caractères qui constituent l'alternative, on utilise les parenthèses.
- Ex : Le masque J'aime mon (chat|chien) ! trouvera J'aime mon chat ! et J'aime mon chien !. Une des deux alternatives devra absolument être présente.

Utilisation des regex en PHP

On utilise les expressions régulières en PHP avec les fonctions de la suite PREG, comme par exemple :

preg_match : https://www.php.net/manual/fr/function.preg-match.php

preg_match_all : https://www.php.net/manual/fr/function.preg-match-all

preg_replace : https://www.php.net/manual/fr/function.preg-replace.php

Pour faire de la validation, c'est généralement avec preg_match que l'on va procéder!