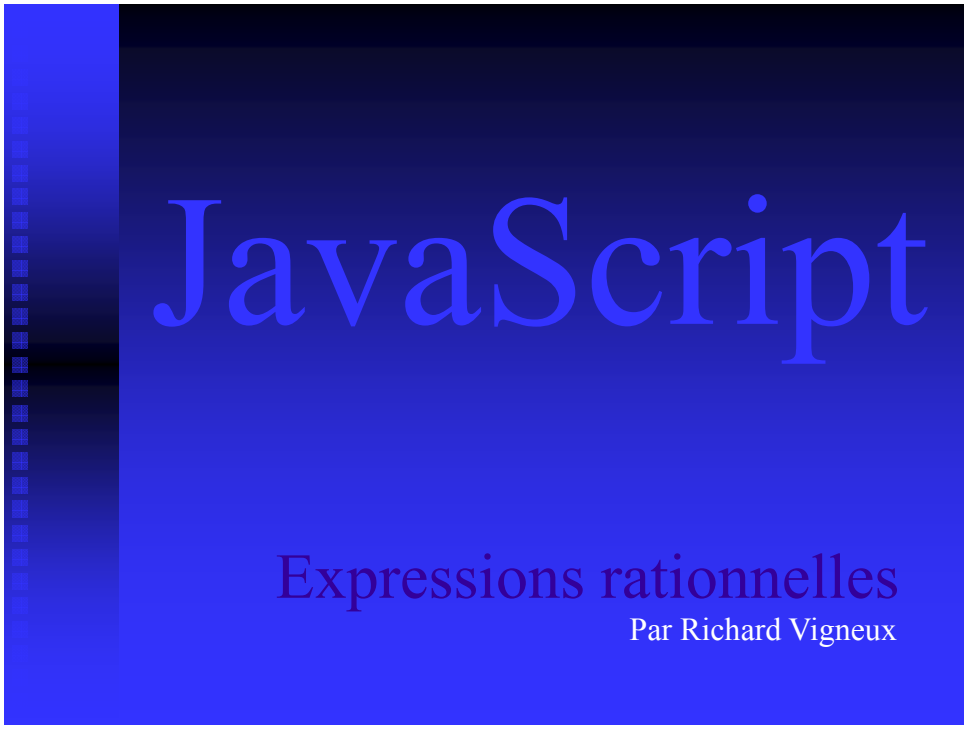

Table des matières

Introduction

- Une expression rationnelle est un outil puissant pour rechercher et manipuler une information spécifique dans un contenu textuel.
- Les expressions rationnelles peuvent être utilisées sur n'importe quelles sources d'information (documents, bases de données, contrôles de formulaires, etc.).
- La force principale des expressions rationnelles est de permettre le repérage d'éléments dont la structure est complexe et même variable.
- Vous avez déjà acquis une partie de la connaissance nécessaire pour créer des expressions rationnelles lorsque vous avez appris à créer des DTD XML.
- Référence:
  - https://developer.mozilla.org/en/JavaScript/Guide/Regular_Expressions

3

Objet String

L'objet « string » de JavaScript offre des méthodes pour appliquer des expressions rationnelles sur des objets « string ».

Méthode	Résultat
match( )	« Array » de toutes les correspondances. « null » s'il n'y a pas de correspondance.
replace( )	Nouvelle chaine de caractères avec remplacement effectué. Possible d'utiliser $ pour représenter les sous-expressions ( ).
search( )	Position de l'expression cherchée dans la chaine de caractères fournie (-1 si non trouvée).
split( )	« Array » de tous les bouts de texte qui étaient séparés par la valeur représentée par l'expression rationnelle.

- Référence:
  - https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/String

4

Objet RegExp (1/2)

L'objet « RegExp » de JavaScript offre des propriétés et méthodes pour récupérer les résultats et appliquer les expressions rationnelles.

Méthode	Résultat
exec( )	« Array » de toutes les correspondances. « null » s'il n'y a pas de correspondance (semblable à String.match()).
test( )	« true » si l'expression rationnelle est trouvée dans la chaine de caractères. « false » dans le cas contraire.
toString( )	Expression rationnelle sous forme de chaine de caractères.

5

Objet RegExp (2/2)

Propriété	Utilité
source	Représente l'expression rationnelle en format texte.
lastIndex	Donne la position, dans le texte, suivant la dernière correspondance avec l'expression rationnelle.
global	Est « true » si le déterminant « g » est utilisé avec l'expression.
ignoreCase	Est « true » si le déterminant « i » est utilisé avec l'expression.
multiline	Est « true » si le déterminant « m » est utilisé avec l'expression.

- Référence:
  - https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/RegExp

6

Les déterminants

Les déterminants affectent la portée de l'expression rationnelle. On peut en utiliser un seul ou une combinaison de plusieurs.

Déterminant	Utilité
g	Force à trouver toutes les occurrences contenues dans le texte.
i	Ne tiendra pas compte de la casse des caractères.
m	Étendra la recherche à toutes les lignes. Normalement, la rencontre d'un code de fin de ligne termine la recherche.

7

Déclaration d'une expression rationnelle

Une expression rationnelle peut être déclarée de deux façons.

- En la délimitant par un couple de barres obliques:
  - var var_ExprReg = / uneExpressionRationnelle / déterminants;
- En utilisant l'objet « RegExp »
  - var var_ExprReg = new RegExp("uneExpressionRationnelle" ,"déterminants");

Note: Dans les deux cas, les déterminants sont optionnels.

8

Codes de création d'une expression rationnelle (1/9)

Puisqu'une expression rationnelle permet de représenter plusieurs chaines de caractères différentes, il nous faut impérativement des codes nous permettant de représenter ces différentes possibilités.

- De façon générale, à part quelques exceptions, tous les caractères peuvent être utilisés directement dans l'expression pour servir de masque.
  - Par exemple: /étudiant_1/
    - Permet de retrouver l'occurrence de la chaine de caractères « étudiant_1 » dans le texte.

9

Codes de création d'une expression rationnelle (2/9)

- Par contre, les symboles « \ | ( ) [ ] { } ^ $ * + ? . » devront impérativement être « échappés » à l'aide d'une barre oblique inverse si vous voulez les retrouver dans le texte. La raison est que ces symboles ont une signification particulière dans une expression rationnelle.
  - Par exemple: /158$/
    - Permet de retrouver l'occurrence « 158 » si et seulement si, elle se trouve à la fin du texte.
    - Si je veux trouver « 158$ » n'importe ou dans le texte, j'utiliserai la syntaxe:
      - /158\$/

10

Codes de création d'une expression rationnelle (3/9)

- Un symbole particulièrement utile est le point « . » puisqu'il permet de remplacer n'importe quel symbole.
  Par exemple:
  - /p.s/ → permet de trouver « pas » , « pis » et « p-s ».
- Il existe aussi des caractères « échappés » normalisés qui fonctionnent de la même façon qu'en C et que vous pouvez utiliser dans vos expressions.

Code	Description
\n	Représente le code de changement de ligne.
\r	Représente le code de retour de chariot.
\t	Représente le code de tabulation horizontale (tabulation normale).
\v	Représente le code de tabulation verticale.
\f	Représente le code de saut de page

Codes de création d'une expression rationnelle (4/9)

- Il existe d'autres caractères « échappés », mais avant de les regarder, regardons la façon d'indiquer un choix de caractères.
- Pour représenter un choix, nous utiliserons les symboles crochets.
  - Par exemple, si je cherche une voyelle, je pourrais écrire:
    - [aeiouy] → un caractère parmi la liste.
      - Je pourrais alors retrouver le « a » dans « bas » ou le « i » dans le mot « lis ».
  - Il est aussi possible d'indiquer une étendue en utilisant le trait d'union. Par exemple, si je cherche un caractère alphabétique, je pourrais écrire:
    - [a-z] → n'importe quelle lettre de l'alphabet (minuscule).
    - [a-zA-Z] → n'importe quelle lettre de l'alphabet.

12

Codes de création d'une expression rationnelle (5/9)

- Il est aussi possible d'indiquer une exclusion à l'aide de l'accent circonflexe. Par exemple, si je cherche tous les symboles à l'exception des chiffres, je pourrais écrire:
  - [^0-9] → Tous les symboles à l'exception des chiffres.

- Maintenant, regardons les autres caractères « échappés ».
  - Chacun des codes de la page suivante représente une étendue de symboles appelés « classe de symboles ». C'est la raison pour laquelle nous avons regardé les étendues avant de regarder ces codes.
  - Prenez note que les codes en majuscule représentent l'inverse de l'étendue des codes en minuscule.

13

Codes de création d'une expression rationnelle (6/9)

Code	Description
\d	Représente un chiffre "0 à 9". Même que [0-9].
\D	Représente un caractère quelconque sauf un chiffre (contraire de \d). Même que [^0-9].
\w	Représente un caractère alphanumérique ou un trait de soulignement "0-9, a-Z, _". Même que [a-zA-Z_0-9].
\W	Représente tout ce qui n'est pas un caractère alphanumérique ou un trait de soulignement "contraire de \w". Même que [^a-zA-Z_0-9].
\s	Représente un espace sous une des formes suivantes: blanc, \t, \v, \n, \r, \f. Même que [\t\v\n\r\f].
\S	Représente tout ce qui n'est pas un espace "contraire de \s". Même que [^ \t\v\n\r\f].
\b	Représente un délimiteur de mots (début ou fin d'un mot)
\B	Représente un caractère quelconque sauf un délimiteur de mots (contraire de \b)

14

Codes de création d'une expression rationnelle (7/9)

- Nous avons aussi des codes de répétition qui se place juste après le symbole « ■ » ou regroupement sur lequel il s'applique.

Code	Description
■{n,m}	Le caractère précédent doit être répété au moins "n" fois, mais au plus "m" fois.
■{n, }	Le caractère précédent doit être répété au moins "n" fois.
■{n}	Le caractère précédent doit être répété le nombre de fois indiqué entre les accolades.
■*	Le caractère qui précède l'étoile peut être absent ou répété plusieurs fois "0 à N". Même que {0,}.
■+	Le caractère qui précède le plus doit être présent ou répété plusieurs fois "1 à N". Même que {1,}.
■?	Le caractère qui précède le point d'interrogation peut être absent ou présent une seule fois "0 à 1". Même que {0,1}.

15

Codes de création d'une expression rationnelle (8/9)

- Les parenthèses sont particulières dans les expressions rationnelles.
  - Premièrement, elles permettent de faire des regroupements. Ce qui est assez naturel.
  - Deuxièmement, elles permettent de créer des sous-expression qui seront réutilisable dans l'expression elle-même à l'aide de « \N », et par l'entremise de « $N » dans un « replace » (N représente un entier. N=1 pour la première sous-expression de l'expression).
  - Exemple: Si je voulais retrouver un contenu balisé dans un document, quel que soit la balise utilisée.
    /<(.*)>.*<\/\1>/
    - Explication: **.*** → Représente le nom de la balise → exemple h1. Puisque **.*** est entre parenthèses, alors **\1** représente h1 dans cet exemple (**.*** pourrait être « div » et ça fonctionnerait aussi).

16

Codes de création d'une expression rationnelle (9/9)

- Enfin, il nous reste deux types de code à voir.
  - Premièrement, le OU représenté par une barre verticale « | » qui permet de vérifier plusieurs expressions en même temps.
    - Par exemple, je cherche le prénom d'une personne. Ce prénom peut être Marc, Ève ou Louise.
      /Marc|Ève|Louise/
    - Cette technique est surtout intéressante lorsqu'il n'est pas possible dans une seule expression d'indiquer tous les cas valides. À ce moment on écrit plusieurs expressions séparées par des OU.
  - Deuxièmement, les codes de positions qui nous permettent d'indiquer si ce que l'on cherche est au début ou à la fin du texte.

Code	Description
^■	Trouve l'expression suivante « ■ » au début de la chaîne.
■$	Trouve l'expression précédente « ■ » à la fin de la chaîne.

17

Exemple

Il n'est pas possible de maîtriser les expressions rationnelles sans en faire beaucoup. Par contre cela vaut vraiment la peine, car il n'y a rien de plus performant.

- Donc voici un exemple pour vous aider à démarrer:

```
var  var_MonExprReg =
new RegExp ("(M\.|Mme|Mlle) ?(Tremblay|Sirois)", "g");
var  var_ResultatExprReg = var_MonExprReg.exec (var_UnTexte);
```

- Explication:
  - Cette expression me permet de trouver tous les Tremblay et les Sirois du texte ainsi que leur titre.
  - Remarquez que le titre et le nom peuvent être séparés ou non d'un espace (il y a un espace devant « ? »).
  - Ce qui aura été trouvé sera dans le Array var_ResultatExprReg.

18

Exercice du jour

- Avant de débuter l'exercice, je vous invite fortement à regarder et tester des exemples sur le site de Mozilla ou autres. C'est beaucoup plus facile de créer des expressions complexes lorsqu'on a une certaine expérience.
- Dans l'exercice d'aujourd'hui, vous devez essayer de faire une seule règle pour régler chaque demande. Ceci veut dire que vous devez éviter au maximum l'utilisation de OU « | » dans vos expressions. En aucun cas, vous ne pouvez utiliser des énoncés conditionnels pour suppléer une expression relationnelle.
- Truc: Vous pouvez utiliser les méthodes focus et select pour sélectionner un contrôle dont la valeur est erronée. De plus, l'ajout des codes ^ au début et $ à la fin de vos expressions vous assurera qu'aucun caractère n'a été entré dans le contrôle au-delà de la valeur de votre expression rationnelle.
- N'hésitez pas à vous créer des sous-fonctions pour rendre votre code plus structuré.
- Présentation du résultat:
  - Corrigé de l'exercice 7 sur les expressions rationnelles

19