Expressions régulières

zone de test

Pourquoi ?

pour chercher/vérifier/remplacer du texte

- vérifier qu'un texte correspond bien à un nombre entier, à une adresse web...
- remplacer toutes les occurrences d'un texte par un autre dans une string
- ...

Comment ?

un langage qui *décrit* des chaînes de caractères. Par exemple :

```
/[A-Z][a-z]*/
```

est une expression régulière qui *reconnaît* tous les mots qui commencent par une majuscule et sont composés d'un nombre quelconque de lettres minuscules. /.../ délimite l'expression.

Utilisation

- Méthodes des strings :

```
var s= "une chaîne";
if (s.match(/[A-Z][a-z]*/)) {
  alert("mot avec majuscule reconnu");
}
```

- Méthodes des regexp

```
var s= "une chaîne";
if (/[A-Z][a-z]*/.test(s)) {
  alert("mot avec majuscule reconnu");
}
```

Le langage des expressions régulières

- texte "normal" + opérateurs spéciaux ;
- les caractères non spéciaux se "reconnaissent" eux-mêmes : par exemple /a/ reconnaît la chaîne "a"
- reconnaissance des séquences : /abc/ reconnaît la chaîne "abc"
- exemple de caractère spécial : "*" = "répétition zéro ou plus" : /ab*c/ reconnaît "ac", "abc", "abbc"...
- La plupart du temps, on cherche *une partie* du texte qui sera reconnue par l'expression régulière

Séquences et caractères simples

symbole	sens	exemple
.	n'importe quel caractère	/.../ = 3 caractères quelconques
lettre, caractère "normal"	le caractère en question	/abc/ = un "a", puis un "b", puis un "c"
\n	Passage à la ligne	
\t	Tabulation	
\u0000	caractère unicode	/\u03B2/ pour β
\w	lettre latine non accentuée	
\d	chiffre	
\s	caractère d'espacement (espace, tabulation, passage à la ligne)	
\W, \D, \S	négation des précédents	
\\, *, \+, \.	le "\" permet de "protéger" les caractères spéciaux pour pouvoir les reconnaître	/\.\.\./ reconnaît trois points (comparer avec ci-dessus)

Intervalles

[...] : permettent de délimiter un ou plusieurs intervalles ou ensembles de caractères.

Ensemble de caractères

il suffit de mettre les caractères en question entre les crochets :

```
[aeiouy]
```

reconnaît n'importe quelle voyelle

Intervalle

Suite de caractères dans unicode. Exemple [0-9] n'importe quel chiffre

Négation

"^" : caractères qui ne *sont pas* dans l'ensemble.

Exemple `[^0-9]` : caractère qui n'est pas un chiffre.

"-"

Pour mettre le "-" dans un ensemble, on le place en premier :

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

reconnaît toute lettre minuscule ou "-".

Combinaison

On peut mettre entre crochets plusieurs intervalles et des caractères isolés : `[0-9A-E.]` reconnaît toute chiffre hexadécimal ou ".".

Répétition/optionalité

- *, après une expression : reconnaît l'expression 0 ou n fois
- +, après une expression : reconnaît l'expression 1 ou n fois
- ?, après une expression : reconnaît l'expression 0 ou 1 fois

*, +, ? porte sur le dernier élément. Sinon, il faut des (...) :

/a[a-z]+/ : reconnaît les mots qui commencent par a

/(a[a-z])+/ : reconnaît les mots dont les lettres impaires sont des "a" ("ananas")

Disjonction

```
[0-9]+\s*(francs|euros)
```

- "ou"
- marquée par "|" (barre verticale
- priorité moins forte que la séquence

Autres opérateurs

- ^ : reconnaît le début de la chaîne
- $: reconnaît la fin de la chaîne

Exemples :

- /^a/ : toute chaîne commençant par "a"
- /^[a-z]$/ : une chaîne contenant exactement un mot en minuscules (et sans accents).

Options des regexp

- Se placent après le "/" final
- g : "général". reconnaît *toutes* les occurrences de l'expression dans la chaîne (surtout pour replace)
- i : ignore la casse (majuscules/minuscules)

Exemples

- /^a/i : toute chaîne commençant par "a" ou "A".

Parenthèses et groupes

Les parenthèses délimitent des *groupes*. La première parenthèse *ouvrante* délimite le groupe 1, la seconde le groupe 2, etc...

On peut extraire les groupes, mais aussi les utiliser dans l'expression elle-même. \1 désigne le texte reconnu par le premier groupe, \2 par le second, etc...

Exemple `/([0-9])\1+/` reconnaît un même chiffre répété au moins deux fois ("111" ou "33333" mais pas "123")

Pour qu'une parenthèse ne crée pas de groupe, on utilise `(?:....)` :

```
/(?:\s|,)/
```

string.match

- s'applique à une string et prend comme argument une regexp
- Permet de savoir si une expression reconnaît une partie d'une chaîne de caractères
- Permet d'extraire la partie reconnue
- Renvoie null si non reconnue, et un tableau sinon
- La première case du tableau contient le texte reconnu par l'expression en entier, la seconde le texte reconnu par le groupe 1, la troisième par le groupe 2, etc...

Exemples

```
if (s.match(/[0-9]+/)) {
  ...
}
```

"si s contient un entier"

```
var t= s.match(/[0-9]+/);
if (t != null) {
  val= t[0];
}
```

Vérifie que s contient un entier et extrait le texte en question dans val.

string.split

Deux versions

- argument string : découpe selon le texte exact :

```
s="192.168.0.1";
t= s.split("."); // t = {"192", "168", "0", "1"}
```

- argument regexp : découpe selon l'expression régulière

```
s="du texte  et des\t\t espacements     étranges";
t= s.split(/\s+/); // t= du,texte,et,des,espacements,étranges
alert(t);
```

  Note : si la regexp contient des groupes, le résultat contiendra pour chaque élément séparé :
  - l'élément
  - les cases du tableau associé au groupe (voir match).

string.replace

Prend deux arguments : une regexp et une string

Ne modifie pas la chaîne à laquelle elle s'applique

Exemple :

```
s1= s.replace(/\s+/g, "-");
```

Remplace toutes (/g) les suites d'espaces par un "-".

g : sans le "g", on remplace uniquement la première occurrence

String replace, et les groupes

On peut utiliser $0, $1,... dans la chaîne de remplacement pour désigner une partie de la chaîne d'origine

Exemple : remplacer toutes les occurrences de NOMBRE eur./euros par NOMBRE €

```
s= "le prix est de 300 eur., mais on peut transiger à 199 euros."
s= s.replace(/([0-9]+)\s*(eur\.|euros)/g, "$1 \u20AC");
alert(s);
```

Construction d'une regexp

- On peut utiliser /..../
- ou le constructeur Regexp(...) qui prend comme argument une string :

```
var s= ".*"; // string !!!
var r= new Regexp(s); // regexp
```

Regexp et unicode

- bibliothèque orientée ASCII (\w = [a-zA-Z] !!!)
- bonne couverture difficile
- utiliser des bibliothèques comme http://xregexp.com/ plus unicode plugin.

Exercices

Exercice1

Reprendre l'exercice sur l'âge, le nom et l'adresse mail en utilisant les expressions régulières

Exercice2

Écrire un code qui permette de vérifier qu'un texte est bien une adresse IP, et qui en extraie les quatre parties

Exercice3

On tape une suite de nombre (y compris des réels) dans un champ, séparés par des espaces ou des retour-chariot, et on veut calculer leur somme. Adaptez avec des expressions régulières