RegExp en JavaScript

Table des matières

1. Expressions Régulières.

1.1. principe.

Les expressions régulières constituent **une méthode de description** de chaînes de caractère. **Une expression régulière est un objet** qui constitue une telle description. La chaîne de caractère qui sert à exprimer la description s'appelle **un motif (*pattern*)**. Les principales utilisations des expressions régulières sont la recherche de motifs dans une chaîne donnée, et la recherche/remplacement de sous chaînes d'une chaîne qui correspondent au motif.

exemple :

- le motif à décrire : suite d'espaces en début de ligne ;
- action : supprimer dans un texte les suites d'espace en début de ligne.

note : par abus, on utilise le mot **motif** pour désigner le texte de la description, la chaîne descriptive, ou l'objet JavaScript créé à partir de la chaîne descriptive (ou chaîne de motif, techniquement appelée **source** de l'objet **RegExp**). Il y a un abus également dans l'utilisation du terme «expression régulière», qui peut désigner aussi bien la chaîne de motif que l'objet **RegExp**. Le vocabulaire est issu des usages dans d'autres langages et dans la théorie des automates, en particulier du langage **Pearl** dont le module a été adapté pour **JavaScript**.

1.2. Les objets RegExp.

RegExp() est un constructeur ; il produit des **objets RegExp**.

syntaxe : **new RegExp (*chaîne de motif*)**

L'argument est une chaîne de caractères avec une syntaxe propre, pour pouvoir être transformée en objet **RegExp**.

littéral : il existe un littéral **RegExp**, qui évite d'appeler explicitement le constructeur :

***/chaîne de motif/*options**

Pour des raisons pratiques, nous utiliseront presque toujours la forme littérale dans les exemples.

1.3. retour sur les objets String.

Les objets **RegExp** sont inséparables des objets **String**. En effet, les méthodes les plus immédiates sont celles qui consistent à identifier (et éventuellement à remplacer) dans un objet **String** les sous-chaînes qui sont décrites par le motif. On reviendra sur les détails concernant chacune des méthodes.

Pour l'instant, voici quelques indications pour comprendre les exemples :

*** match(*expression régulière*)**

Recherche la ou les sous-chaînes décrites par l'expression régulière (si celle-ci est remplacée par une chaîne, le transtypage est automatique). Retourne un tableau

Retourne **null** si aucune sous-chaîne n'a pu être identifiée. Un tableau concernant la ou les identifications dans le cas contraire.

*** replace (*expression régulière, chaîne de remplacement*)**

Remplace la ou les sous-chaînes identifiée par la chaîne de remplacement.

*** search(*expression régulière*)**

Retourne la position de la première sous-chaînes identifiées ; **-1** en cas d'échec.

2. Définition de la chaîne de motif.

2.1. caractères littéraux.

Tous les caractères ont vocation à se représenter eux mêmes.

exemple : ***/papa est en voyage/*** est un objet **RegExp**, formé de 18 caractères «ordinaires». Une action classique dans les traitements de texte consiste à rechercher (ou à chercher/remplacer) la première occurrence d'une chaîne dans un texte (où l'occurrence à partir d'un certain endroit du texte). La chaîne à recherchée peut être une chaîne de motif de **RegExp**.

caractères non graphiques : un certain nombre de caractères n'ont pas de glyphe. Par exemple la tabulation, la fin de ligne etc. Dans ces cas on doit utiliser un caractère d'échappement pour les introduire dans une chaîne de motif. Le caractère d'échappement en javaScript est **l'antislash **

Les caractères sont alors représentés par deux caractères graphique, le premier étant ****

	quel caractère ?	remarques	unicode
\0	caractère NUL		**\u0000**
\t	caractère de tabulation	**TAB**	**\u0009**
\n	caractère de retour à la ligne	**LF**	**\u000A**

\v	caractère tabulation verticale		**\u000B**
\f	caractère saut de page	**FF**	**\u000C**
\r	retour chariot	**LF**	**\u000D**
\x*nn*	caractère latin de code hexadécimal *nn*	les caractères hex **a-f** en majuscules ou minuscules	
\uxxxx	caractère unicode de code hexadécimal *xxxx*		
\cX	caractère de contrôle		

caractères interdits :

Certains caractère **ne se représentent pas eux mêmes**, car ils vont avoir une signification particulière. Ce sont des caractères de ponctuation ou des caractères ayant un usage spécifique :

^ $. * + ? = ! : | \ / ( ) [ ] { }

Noter que dans ces caractères, il n'y a pa le moins **-**, les quotes simples ou doubles **" '** , l'espace, la virgule ou le point-virgule **;** le dièse, **#**, l''esperluette **&**, l'arrobase @, l'underscore_

Si on introduit un caractère interdit, il faut l'échapper : **\(** pour la parenthèse, **\.** pour le point et bien évidemment **** pour l'antislash. Ainsi le littéral **RegExp** pour l'antislash est **/\\/**

Si on utilise la syntaxe littérale, **il n'y a pas lieu d'échapper les quotes**. Un antislash abusif est en général sans conséquence sauf dans les cas où l'échappement donne un sens particulier (**\i** , avec **i** entier par exemple).

2.2. les classes de caractères.

On a souvent besoin d'exprimer un caractère appartenant à une classe de caractères : un chiffre, tout sauf un chiffre, un caractère valide dans un identificateur, un ASCII majuscule...

le caractère	correspondance	équivalent	exemple
[caractères]	un des caractères entre crochets		[abc] soit a, soit b, soit c
[^caractères]	tout sauf un caractère entre crochets		[^abc] tout sauf a, b, c
[x-y]	tout caractère entre x et y		[A-Z] majuscule ASCII
[x-yz-t]	tout caractère entre x et y ou entre z et t		[A-Za-z_0-9]
.	le point : tout caractère sauf un caractère de fin de ligne.		
\w	tout ASCII pour identificateur	[A-Za-z_0-9]	
\W	tout sauf ASCII pour identificateur	[^A-Za-z_0-9]	
\s	tout caractère d'espacement		
\S	tout caractère non espacement		
\d	tout chiffre	[0-9]	
\D	tout sauf un chiffre	[^0-9]	

* les caractères entre crochet peuvent être échappés : **/[\s\d]/** désigne tout caractère qui est soit un espacement (espace, tab etc) soir un chiffre.

* il existe un échappement spécial : **\b** ; il désigne le back-space **si on le trouve entre crochets** :

/[\b]/ est valide pour le back-space.

* quelques exemples :

* **/\d\d/** signifie une suite de deux chiffres et **/\d\d\d\d/** une suite de quatre chiffres.

* ***/ /*** signifie un espace, ce qui est souvent source d'erreur, surtout si on a tendance à éclaircir les sources pour les rendre plus lisibles : les espaces appartiennent au motif !

* /**[0-9a-fA-F]**/désigne un chiffre hexadécimal ; faire attention qu'un espace ne se glisse pas entre les crochets !

exemple : test sur \w

```
1.<script type="text/javascript">
2.    var chnEntree ="une chaîne-123&leçon";
3.    var motif = /\W/g;
4.    var chnSortie = chnEntree.replace (motif, "^^^");
5.    alert (chnSortie) ;
6.</script>
7.<!-- h1401_re.html -->
```

une chaîne-123&leçon
une^^^cha^^^ne^^^123^^^le^^^on

exemple : test sur **\s** (espaces)

```
1.<script type="text/javascript">
2.    var chnEntree ="une chaîne          (2 tab)\n-123\xA0&\u00a0leçon";
3.    var motif = /\s/g;
4.    var chnSortie = chnEntree.replace (motif, "^^^");
5.    alert (chnEntree+"\n"+chnSortie) ;
6.</script>
7.<!-- h1402_re.html →
```

une chaîne (2 tab)
-123 & leçon
une^^^chaîne^^^^^^(2^^^tab)^^^-123^^^&^^^leçon

2.3. les répétiteurs.

Un répétiteur est un moyen de décrire la répétition d'un caractère (ou d'un groupe de caractère)

exemples : autant de fin de lignes successives que l'on veut ;
deux espaces au moins de suite ;
entre 3 et 5 chiffres...

répétiteur	signification	exemple
*	zéro, un ou plusieurs	**a*** : 1, n... ou 0 caractère(s) **a** en suivant
+	au moins un	**\d+** au moins un chiffre
?	facultatif	**a? a peut être présent ou non, pris en considération ou non (3 cas posibles)**
		[0-9a-fA-F]{4} un hexadécimal à 4 chiffres

{n}	exactement n fois	
{n,}	au moins n fois	
{n,m}	au moins n fois, pas plus de m fois	

exemples :

* **/\d{2,4}/** une suite de 2 à 4 chiffres : 00, 595, 9874, 0001

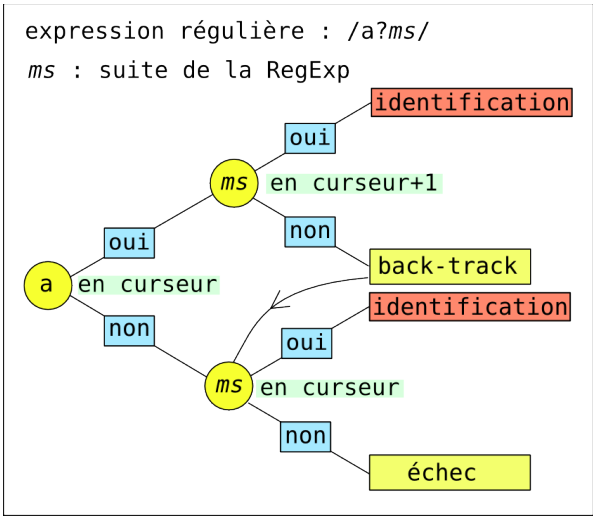
* **/\s+java\s+/** le mot java encadré obligatoirement par des espaces ou assimilés.

* **/"[^"]*"/** une quote, éventuellement des caractères, sauf une quote, puis une quote comme dans **"abcd efgh"**, ou **""**

attention : les répétiteurs **?** et ***** peuvent se révéler peu compréhensibles et doivent être utilisés avec précaution.

```
1.<script type="text/javascript">
2.    var chnEntree ="CaaabBbBaVa"; ;
3.    var motif = /a?[^A-Z]/g;
4.    var chnSortie = chnEntree.replace (motif, "§");
5.    alert (chnEntree+"\n"+chnSortie) ;
6.</script>
7.<!-- h1403_re.html -->
```

le motif **/a?[^A-Z]/** décrit : 0 ou 1 caractère **a** non suivi par une majuscule. On examine d'abord si un **a** convient ; en cas d'échec, on prend l'absence de **a** en considération.



CaaabBbBaVa

C§§B§B§V§

2.4. gourmandise.

L'utilisation des répétiteurs peut poser problème : Par exemple, on peut demander si le motif **/ba+/** décrit une sous chaîne de **baaaaaaaaaab** et quelle est cette sous chaîne. La solution courte se serait **ba** , la solution longue **baaaaaaaaaa** et tous les intermédiaires sont possibles. En fait on ne retient que les deux extrêmes.

Mais il reste à la discriminer : **par défaut, les répétiteurs sont gourmands** (avides), c'est-à-dire qu'il décrivent la solution la plus longue. On peut **les forcer** à décrire la solution la plus courte ; dans ce cas on leur adjoint un point d'interrogation : **+? *? {n,m}?**

Attention : supposons que l'on ait /**a*?b/** Comment se fait la description ? Il faut rechercher **a.** Si on le trouve, on continue avec la recherche de **b**, mais en égrenant les **a**. Donc le motif décrit bien la sous-chaîne **aaab** par exemple dans la chaîne **cccaaabbb**. Par contre dans la chaîne **cccbeee**, seule la sous-chaîne **b** est décrite.

```
1.<script type="text/javascript">
2.    var chnEntree ="CaaabBBBaVaaaabDbbEbF";
3.    var motif = /a*?b/g;
4.    var chnSortie = chnEntree.replace (motif, "§");
5.    alert (chnEntree+"\n"+chnSortie) ;
6.</script>
7.<!-- h1403_re.html →
```

CaaabBBBaVaaaabDbbEbF

C§BBBaV§D§§E§F

2.5. Les attributs ou drapeaux.

Les attributs modulent la façon dont la recherche de sous-chaîne décrite par le motif doit être faite.

* attribut **i** : (ignoreCase) ne pas distinguer majuscules et minuscules ;

* attribut **g** : (global) rechercher dans tout le texte et ne pas s'arrêter à la première identification ;

* attribut **m** : recherche multiligne ; permet d'identifier les débuts et fins de lignes (voir Ancrage)

Les attributs (appelés aussi drapeaux) se placent après le slash final des littéraux, ou en second argument du constructeur.

***/chaîne de motif/*img**

new RegExp (*chaîne de motif, attributs*)

Les attributs sont des propriétés des objets **RegExp** : **global, multiline, ignoreCase.** Attention, ces propriétés sont **en lecture seule** ! Ces attributs ne sont pas modifiables : ils appartiennent à la **RegExp** ; et si dans un script on veut utiliser le même motif plusieurs fois avec des attributs différents, il faut créer autant de **RegExp** différentes.

exemple d'utilisation :

```
motif = new RegExp (chn, "g")
```

alert (motif.global+" "+motif.multiline+" "+motif.ignoreCase);

3. Ancrage des RegExp.

3.1. La syntaxe.

Les textes sont en général considérés comme ayant une double structure : en mots et en lignes (ce que les traitements de textes nomment paragraphes !).

Les ancrages servent à préciser où, dans un texte, il faut chercher les sous-chaînes décrites par un motif.

ancrages	explications
^	le motif doit décrire une sous-chaîne en début de chaîne ou de ligne
$	le motif doit décrire une sous-chaîne en fin de chaîne ou de ligne
\b	limite d'un mot. Encadrement par **\b** : le motif doit décrire un mot

\B	la position n'est pas limite d'un mot
(?=*chaîne de motif*)	les caractères qui suivent recherchent le motif, mais ne l'incluent pas dans la sous-chaîne identifiée.
(?!*chaîne de motif*)	les caractères suivants excluent le motif.

3.2. Début et fin de la chaîne de recherche.

Les ancres **^**et **$** fonctionnent sur la chaîne à analyser si le mode est monoligne et sur les lignes lorsque l'attribut m est posé (mode multiligne).

exemple : supprimer les blancs (espaces, tabulations) en début de chaîne (ligne) ou fin de chaîne (ligne).

```
1.<script type="text/javascript">
2.    var chnEntree ="       papa est en voyage       ";
3.    var motif1 = /^\s*/ , motif2 = /\s*$/ ;
4.    var resultat = (chnEntree.replace(motif1,"")).replace(motif2,"") ;
5.    alert ("***"+chnEntree+"***\n"+"***"+resultat+"***") ;
6.</script>
7.<!-- h1405_re.html →
```

```
***       papa est en voyage      ***
***papa est en voyage***
```

3.3. notion de mots.

la forme d'échappement **\b** présente quelques difficultés : dans la chaîne de motif, elle peut représenter aussi la tabulation ; on a vu qu'il faut la crocheter ! Elle se révèle incommode dès que l'on travaille sur des textes : elle considère comme début ou fin de mot tout ce qui appartient à **\W** et donc les caractères accentués, la ponctuation (sauf l'underscore) etc. Il convient donc souvent de définir «à la main» ce que l'on considère comme un mot.

exemple 1 :

```
1.<script type="text/javascript">
2.  var chnEntree ="Java apparaît en 1993, JavaScript un an plus tard.\n"+
3.    "Il s'appelle LiveScript, et tente de profiter de la notoriété de
Java,\n"+
4.    "non pas java, le «petit noir», ni «lajavanaise» de Gainsbourg \n"+
5.    "ni l'île de Java, mais le nouveau langage dont on commence à parler.";
6.  var motif = /\bjava\b/gmi;
7.  var resultat = chnEntree.replace(motif, "§§§§") ;
8.  var alert (chnEntree+"\n\n"+resultat) ;
9.</script>
10.<!-- h1406_re.html -->
```

```
Java apparaît en 1993, JavaScript un an plus tard.
Il s'appelle LiveScript, et tente de profiter de la notoriété de Java,
```

non pas java, le «petit noir», ni «lajavanaise» de Gainsbourg
ni l'île de Java, mais le nouveau langage dont on commence à parler.

§§§§ apparaît en 1993, JavaScript un an plus tard.
Il s'appelle LiveScript, et tente de profiter de la notoriété de §§§§,
non pas §§§§, le «petit noir», ni «lajavanaise» de Gainsbourg
ni l'île de §§§§, mais le nouveau langage dont on commence à parler.

exemple 2 :

Voici un exemple qui peut être affiné (il y a quelques caractères parasites et il manque les ligatures) . On recherche des mots en langues ouest européennes :

```
1.<script type="text/javascript">
2.    var chnEntree ="aaa Ami avecéàaze qsdf%sdf,vcx";
3.    var motif = /[a-zA-Z\u00C0-\u00FF]+/g;
4.    var resultat = chnEntree.match(motif) ;
5.    alert (resultat) ;
6.</script>
7.<!-- h1407_re.html -->
```

0	"aaa"
1	"Ami"
2	"avecéàaze"
3	"qsdf"
4	"sdf"
5	"vcx"

3.4. assertion vers l'avant.

```
1.<script type="text/javascript">
2.  var chnEntree ="Java apparaît en 1993, JavaScript un an plus tard.\n"+
3.    "Il s'appelle LiveScript, et tente de profiter de la notoriété de
Java,\n"+
4.    "non pas java, le «petit noir», ni «lajavanaise» de Gainsbourg \n"+
5.    "ni l'île de Java, mais le nouveau langage dont on commence à parler.";
6.  var motif = /java(?!script)/gmi;
7.  var resultat = chnEntree.replace(motif, "§§§§") ;
8.  alert (chnEntree+"\n\n"+resultat) ;
9.</script>
10.<!-- h1408_re.html -->
```

`/java(?!script)/gmi;` le mot **java** est recherché, mais non suivi de **script** ; la recherche est globale

(tout est recherché), multi-ligne, et indifférente à la casse.

Java apparaît en 1993, JavaScript un an plus tard.
Il s'appelle LiveScript, et tente de profiter de la notoriété de Java,
non pas java, le «petit noir», ni «lajavanaise» de Gainsbourg
ni l'île de Java, mais le nouveau langage dont on commence à parler.

§§§§ apparaît en 1993, JavaScript un an plus tard.
Il s'appelle LiveScript, et tente de profiter de la notoriété de §§§§,
non pas §§§§, le «petit noir», ni «la§§§§naise» de Gainsbourg
ni de l'île de §§§§, mais le nouveau langage dont on commence à parler.

4. Expressions parenthésées (ou groupements).

4.1. Une chaîne de motif peut être parenthésée.

Le système de parenthèses (évidemment bien balancé) peut fonctionner comme en mathématiques : il isole un segment du motif, un groupement, sur lequel par exemple appliquer un répétiteur (ou ne rien faire). Les parenthèses peuvent être imbriquées.

Une chaîne de motif qui comporte des parenthèses définit des sous-motifs. Il est utile de **numéroter les parenthèses ouvrantes, de gauche à droite, en commençant à 1 et pas à zéro !** Les sous-chaîne reconnues sont numérotées de la même façon, que celles-ci soient imbriquées ou non. La chaîne entière est numérotée 0. Si on veut qu'une parenthèse de groupement ne soit pas numérotée, on utilise **(?:...)**. Si un groupe est numéroté avec la valeur **i**, alors **\i** permet de rechercher les mêmes caractères que ceux trouvés lorsque le numéro de groupe **i** a été identifié.

Lors d'une recherche ou d'un remplacement, les sous-chaînes identifiées sont désignées par **$** suivi du numéro afférent au sous-motif correspondant (pour la chaîne complète, **$&**, attention : **$&** et non **$0**) On peut donc réutiliser une sous-chaîne identifiée dans une chaîne de remplacement. On numéroter jusque 99 parenthèses ouvrantes.

exemple 1 :

On cherche à vérifier si une adresse mail usuelle est correcte ou non. Pour des raisons liées à l'exposé, on a utilisé la méthode **match()** et on a surparenthésé ce qui permet de décomposer la reconnaissance.

On rappelle qu'on considère une adresse mail valide par le fait qu'elle est constituée des «segments d'identification» commençant par une lettre ASCII et que les chiffres, l'undescore, le tiret sont ensuite acceptés. L'adresse se termine par un indicatif de pays (**.fr**) ou de genre (**.com**), avec au moins deux caractères et constitué de lettres.

```
1.<script type="text/javascript">
2.    var chnEntree ="jean-fr.mercier@orange.fr" ;
3.    var chnOblg = "([a-z_][\\w-]*)";
4.    chnFac = "((\\.([a-z_][\\w-]*))*)";
5.    chnPays = "(\\.[a-z]{2,})";
6.    chn = "^"+chnOblg+chnFac+"@"+chnOblg+chnFac+chnPays+"$";
7.    var motif= new RegExp (chn);
8.    alert (motif.source);
9.    alert (chnEntree.match(motif));
```

```
10./script>
11.<!-- h14015_re.html -->
```

```
^([a-z_][\w\-]*)((\.([a-z_][\w\-]*))*)@([a-z_][\w\-]*)((\.([a-z_][\w\-]*))*)(\.[a-z]{2,})$
```

```
0       "jean-fr.mercier@orange.fr"
1       "jean-fr"
2       ".mercier"
3       ".mercier"
4       "mercier"
5       "orange"
6       ""
7       undefined
8       undefined
9       ".fr"
index   0
input   "jean-fr.mercier@orange.fr"
```

* noter le double slash pour un premier échappement de l'antislash dans la chaîne argument. Par contre, le signe tiret **-** n'a pas à être échappé.

exemple 2 : mode de résolution de **\i**

```
1.<script type="text/javascript">
2.    var chnEntree =
3."un \"fleuriste\" qui n'aime par la \"rose\" ni"   l'\"églantine\" sauvage";
4.    var motif= /(['"])(.*?)\1/g;
5.    alert (chnEntree.replace(motif,"«$2»"));
6.</script>
7.<!-- h14016_re.html -->
```

```
un «fleuriste» qui n«aime par la "rose" ni l»«églantine» sauvage
```

* ligne 4 : noter en premier lieu la recherche non gourmande. Sinon, la résolution de **\1** se serait faite sur la quote double après **églantine**. La quote simple de **n'aime** est résolue avec la quote simple de **l'églantine**, et la chaîne intermédiaire est reprise à l'identique.

4.2. la méthode de chaîne `replace` avec utilisation des sous-chaînes trouvées.

Le second argument de **replace** peut comporter des sous-chaînes reconnues, mais en plus peut être une fonction qui calcule la chaîne de remplacement.

exemple :

Remplacer dans un texte les guillemets droits (quote double) par les guillemets français.

```
1.<script type="text/javascript">
2.    var chnEntree ='"Java" apparaît en 1993, "JavaScript" un an plus tard.\n'+
```

```
3.'Il s\'appelle "LiveScript", et tente de profiter de la notoriété de
"Java",\n'+
4.'non pas "java", le «petit noir», ni «lajavanaise» de Gainsbourg \n'+
5.'ni l\'île de "Java", mais le nouveau langage dont on commence à parler.';
6.    var motif = /"([^"]*)"/gmi;
7.    alert (chnEntree+"\n\n"+resultat) ;
8.</script>
9.<!-- h1409_re.html →
10.    var resultat = chnEntree.replace(motif, "«$1»") ;
```

"Java" apparaît en 1993, "JavaScript" un an plus tard.
Il s'appelle "LiveScript", et tente de profiter de la notoriété de "Java",
non pas "java", le «petit noir», ni «lajavanaise» de Gainsbourg
ni l'île de "Java", mais le nouveau langage dont on commence à parler.

«Java» apparaît en 1993, «JavaScript» un an plus tard.
Il s'appelle «LiveScript», et tente de profiter de la notoriété de «Java»,
non pas «java», le «petit noir», ni «lajavanaise» de Gainsbourg
ni l'île de «Java», mais le nouveau langage dont on commence à parler.

4.3. utilisation du dollar dans les chaînes `replace`.

$1, $2 … $99	sous -chaînes d'ordre 1, 2, … 99 ; **$** est supposé suivi de un ou deux chiffres.
$&	sous-chaîne correspondant à la RegExp
$`	texte à gauche de la sous-chaîne
$'	texte à droite
$$	dollar littéral

5. Les méthodes.

5.1. La méthode match() des instances de String sans l'attribut g

Recherche la première correspondance avec la RegExp.

Retourne un objet contenant les résultats de la correspondance ; **null** si rien n'est trouvé

Les propriétés **"0", "1", "2"** et correspondent à **$&, $1, $2** etc.

La propriété **index** donne la position de la sous-chaîne trouvée.

La propriété **input** donne la chaîne de recherche.

Il y a la propriété de tableau prédéfinie **length**.

exemple :

```
1.<script type="text/javascript">
2.    var chnEntree = "azertaaabbbccdddddeeeqsdfg" ;
3.    var motif = /a+(b+(c+)(d+))e+/ ;
4.    var resultat = chnEntree.match(motif) ;
5.    for(x in resultat)
6.        alert (x+"  "+resultat[x]);
7.</script>
8.<!-- h14010_re.html -->
```

```
0           aaabbbccdddddeee
1           bbbccddddd
2           cc
3           ddddd
index             5
input             azertaaabbbccdddddeeeqsdfg
```

5.2. La méthode `match()` des instances de String avec l'attribut g

Recherche les correspondances successives dans toute la chaîne.

Retourne un tableau des sous-chaînes trouvées, mais **pas les détails** correspondant aux parties parenthésées. Il n'y a pas de propriété **index** ou **input**. Il y a toujours la propriété prédéfinie **length**.

exemple :

```
1.<script type="text/javascript">
2.    var chnEntree ="CaaabBbBaVa";
3.    var motif = /a?[^A-Z]/g;
4.    chnSortie = chnEntree.match(motif);
5.    alert (chnSortie) ;
6.</script>
7.<!-- h1403_re.html →
```

```
0     "aa"
1     "ab"
2     "b"
3     "a"
4     "a"
```

5.3. La méthode `search()` des instances de `String`.

Retourne la position de la première sous-chaîne trouvée ; **-1** si rien n'est trouvé.

La méthode ignore l'attribut **g**.

5.4. La méthode `split()` des instances de `String`.

La méthode accepte comme premier argument une chaîne, mais aussi une expression régulière.

Attention à la particularité suivante : les sous-motifs parenthésés se retrouvent dans le tableau :

exemple :

```
1.<script type="text/javascript">
2.    var chnEntree ="bonjour <b>le monde</b>, Bonjour <i>JoJo</i>";
3.    var motif = /(<[^>]+>)/;
4.    var chnSortie = chnEntree.split(motif);
5.    alert (chnSortie) ;
6.</script>
7.<!-- h1411_re.html →
```

```
0     "bonjour "
1     "<B>"
2     "le monde"
3     "</B>"
4     ", Bonjour "
5     "<i>"
6     "JoJo"
7     "</i>"
8     ""
```

5.5. La méthode `replace()` des instances de `String`.

Le second argument peut être une fonction, son paramètre étant la sous-chaîne trouvée.

exemple :

Mettre en majuscules tous les mots d'un texte (ouest européen).

Les attributs sont **g** (recherche dans tout le texte) et **i** (ne pas tenir compte de la casse).

```
1.<script type="text/javascript">
2.    var chnEntree ='"java" apparaît en 1993, "JavaScript" un an plus tard.\n'+
3.'Il s\'énonce "LiveScript", et tente de profiter de la notoriété de
"java",\n'+
4.'non pas "java", le «petit noir», ni «lajavanaise» de Gainsbourg \n'+
5.'ni l\'île de "Java", mais le nouveau langage dont on commence à parler.';
6.    var motif = /[A-Z\u00C0-\u00FF]+/gi;
7.    var fnRpl = function(chn){
8.       return chn.substring(0,1).toUpperCase()+chn.substring(1)
9.    };
10.   var resultat = chnEntree.replace(motif, fnRpl) ;
11.   alert (resultat) ;
12.</script>
13.<!-- h1412_re.html -->
```

"Java" Apparaît En 1993, "JavaScript" Un An Plus Tard.
Il S'Énonce "LiveScript", Et Tente De Profiter De La Notoriété De "Java",
Non Pas "Java", Le «Petit Noir», Ni «Lajavanaise» De Gainsbourg
Ni L'Île De "Java", Mais Le Nouveau Langage Dont On Commence À Parler.

5.6. La méthode `test()` des instances de `RegExp`.

syntaxe : **expRegExp.test (chaîne).**

Retourne **true** si l'expression régulière peut être identifiée dans chaîne, **false** sinon.

5.7. La méthode `exec()` des instances de `RegExp`.

syntaxe : **expRegExp.exec (chaîne).**

exec() est la méthode à tout faire des Expressions Régulières, mais elle est plus complexe à utiliser que les méthodes de chaîne. En effet, elle est destinée à être appelée de façon itérée, avec un drapeau **g** posé ; à chaque itération, un paramètre, **lastIndex**, se met à jour qui indique où peut commencer la recherche suivante.

* **si l'attribut g n'est pas posé** :

une seule recherche est effectuée ; si aucune correspondance n'est trouvée la fonction retourne **null**. Sinon, elle retourne un objet **Array** qui est similaire à celui de la méthode **match()** avec une recherche non globale.

exemple :

```
1.<script type="text/javascript">
2.    var chnEntree = "000abbbccdddddeee222aaabbcddeee6666";
3.    var motif = /a+(b+(c+)(d+))e+/;
4.    var resultat = motif.exec(chnEntree);
5.    for(x in resultat)
6.    alert (x+"  "+resultat[x]);
7.</script>
8.<!-- h14013_re.html -->
```

```
0           abbbccdddddeee
1           bbbccddddd
2            cc
3           ddddd
index             3
input             000abbbccdddddeee222aaabbcddeee6666
```

*** si l'attribut g est posé :**

exemple :

```
1.<script type="text/javascript">
2.    var chnEntree = "000abbbccdddddeee222aaabbcddeee6666";
3.    var motif = new RegExp ("a+(b+(c+)(d+))e+","g");
4.    while (true) {
5.        var resultat = motif.exec (chnEntree) ;
```

```
6.        if (resultat == null) break ;
7.        var chn ="lastIndex"+" \t"+motif["lastIndex"]+"\n" ;
8.        for(var x in resultat)
9.            chn+=x+" \t\t"+resultat[x]+"\n";
10.       alert (chn) ;
11.</script>
12.<!-- h14014_re.html -->
```

```
lastIndex   17
0           abbbccdddddeee
1           bbbccddddd
2           cc
3           ddddd
index             3
input             000abbbccdddddeee222aaabbcddeee6666
```

```
lastIndex   31
0           aaabbcddeee
1           bbcdd
2           c
3           dd
index             20
input             000abbbccdddddeee222aaabbcddeee6666
```

* Les propriétés énumérables sont identiques au cas précédent ;

* la différence essentielle est la propriété prédéfinie **lastIndex** de l'objet **RegExp.** Dans le cas où l'attribut **g** est posé, **cette propriété est gérée** :

- à chacun des appels de **exec()**, elle indique sur quel caractère commencer la recherche ;

- à la fin d'une recherche, **lastIndex** indique une position après le dernier caractère reconnu. C'est le caractère 0 en cas d'échec.

- la propriété appartient à l'objet **RegExp** : par défaut, lors de la création de l'objet, il est mis à 0. Mais, si on utilise le même objet **RegExp** appelant **exec()**, la mise à jour est automatique. Il ne faut donc pas oublier, le cas échéant, **d'initialiser cette propriété** (si on veut commencer l'examen ailleurs qu'au début de chaîne, ou si une analyse précédente n'a pas été menée au terme).

6. L'alternative.

6.1. l'un ou l'autre.

Deux sous-motifs peuvent être choisis en alternative : si le premier ne peut être identifié à une sous-chaîne dans une chaîne, on essaie alors le second. Le signe de l'alternative est **|**

6.2. exemple.

```
1. <script type="text/javascript">
2.     alert = console.log ;
3.     var chnEntree ='Java apparaît en 1993, JavaScript un an plus tard.\n'+
4. 'Il s\'appelle LiveScript, et tente de profiter de la notoriété de Java,\n'+
```

```
5. 'non pas java, le \"petit noir\", ni lajavanaise de Gainsbourg \n'+
6. 'ni l\'île de Java, mais du nouveau langage dont on commence à parler.';
7.     var motif= /\bjava\w*?|\w*?Script\b/ig;
8.     var chnSortie = chnEntree.replace(motif,"«$&»")
9.     alert (chnEntree+"\n\n"+chnSortie);
10.</script>
11.<!-- h14017_re.html -->
```

```
Java apparaît en 1993, JavaScript un an plus tard.
Il s'appelle LiveScript, et tente de profiter de la notoriété de Java,
non pas java, le "petit noir", ni lajavanaise de Gainsbourg
ni l'île de Java, mais du nouveau langage dont on commence à parler.

«Java» apparaît en 1993, «Java»«Script» un an plus tard.
Il s'appelle «LiveScript», et tente de profiter de la notoriété de «Java»,
non pas «java», le "petit noir", ni lajavanaise de Gainsbourg
ni l'île de «Java», mais du nouveau langage dont on commence à parler.
```