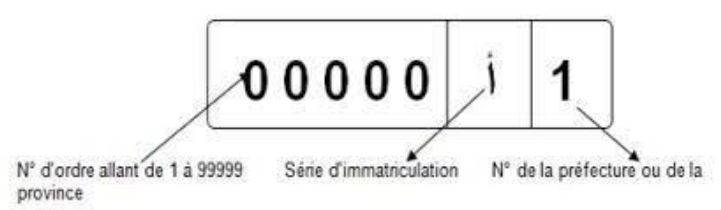
Javascript

Les expressions régulières

Les expressions régulières

- Les expressions régulières, s'appellent aussi les **RegExp**
- Les expressions régulière est un système très puissant permettant de vérifier le **format** d'une chaîne de caractères.
- On utilise pour cela un modèle et on compare la chaîne à ce modèle. Ce modèle on l'appel le **patron** ou ***pattern*** en anglais.



Modèle de matricule de voiture

Modèle ISBN

Les expressions régulières

- Pour travailler avec les expressions régulières, il faut deux choses:
  1. On doit d'abord **créer le modèle** d'une manière correcte
  2. Puis on passe **à vérifier notre chaine** de caractère **si elle correspond au modèle où non.**

Créer une expression régulière

- ***1 ère méthode:***
  Pour créer une expression régulière en JavaScript, il faut entourer le patron (**pattern**) par les caractères (/) :

```
let Expr = /modele/modificateur ;
```

- Exemple :

```
let Expr = /ofppt/i;
```

- ***2ème méthode:***
- **Ou bien** en utilisant le constructeur **RegExp** de JavaScript :

```
let Expr = new RegExp('ofppt');
```

Pattern ou modèle

Les modificateurs

- Les modificateurs de modèle sont placés directement après l'expression régulière. Par exemple, si vous souhaitez rechercher un modèle sans tenir compte de la casse, vous pouvez utiliser le modificateur i, comme ceci : /modèle/i.
- Les modificateurs les plus utilisés sont:

Modificateur	Description
g	Effectue une correspondance globale, c'est-à-dire qu'elle trouve toutes les occurrences.
i	Rend la correspondance insensible à la casse (c'est à dire ne distinguant pas majuscules et minuscules).
m	Modifie le comportement de ^ et $ pour qu'ils correspondent à une limite de nouvelle ligne (c'est-à-dire le début ou la fin de chaque ligne dans une chaîne multiligne), au lieu d'une limite de chaîne.

Les modificateurs

- Exemple 1:

```
var  chaine="Bonjour  mes amis, bonjour à tous les présents!"
var R=chaine.match(/bonjour/i)
console.log(R)
```

Résultat: `['Bonjour', index: 0,`

- Exemple 2:

```
var  chaine="bonjour  mes amis, bonjour à tous les présents!"
var R=chaine.match(/bonjour/g)
console.log(R)
```

Résultat : `['bonjour', 'bonjour']`

- Exemple 3:

```
var  chaine="Bonjour  mes amis,\n bonjour à tous les présents !"
var R=chaine.match(/bonjour/m)
console.log(R)
```

Résultat: `['bonjour', index: 20,`

Construction du modèle (pattern)

Pour vérifier si une chaîne de caractères correspond ou non à un **modèle**, il faut **placer le modèle entre des symboles spéciaux** . Chaque symbole a une signification.

Voila un tableau qui contient ces symboles:

Symboles utilisés par les expressions régulières

Symbole	Correspondance	Exemple
^	Début de ligne	**^b :** commence par b
$	Fin de ligne	**er$** : finit par "er"
.	N'importe quel caractère	**^.$:** contient un seul caractère quelconque
\	Caractère d'échappement	[\.] :contient un "." [\\] contient "\" [\-] contient -
|	Alternative	**^(a|A)** :commence par a ou A
( )	Groupement	**^((a)|(er))** commence par a ou er
-	Intervalle de caractères	**^[a-d]** : commence par a,b,c ou d
[ ]	Ensemble de caractères	**[0-9]** : contient un chiffre 0 ou 1 ou....9
[ag]	Ensemble de carctères	Soit **a** soit **g**
[a-g]		Tous les caractères **de a jusqu'à g**
[^]	Tout sauf un ensemble de caractères	**^[^a] :**ne commence pas par a
+	1 fois ou plus	**^(a)+** : commence par **un ou plusieurs a**
?	0 ou 1 fois	**^(a)?** :commence ou non par un a
*	0 fois ou plus	**^(a)*** : peut ou non commencer par a
{x}	x fois exactement	**a{2}** : deux fois "a"
{x,}	x fois au moins	**a{2,} :** deux fois "a" au moins
{x, y}	x fois minimum, y maximum	**a{2,4}** :deux, trois ou quatre fois "a"

Symboles utilisés par les expressions régulières

Alias	Correspondance	Equivalence
\\d	Un chiffre	[0-9]
\\D	Tout sauf un chiffre	[^0-9]
\\s	Un caractère d'espacement	\n ,\t , \f …
\\w	Un caractère alphanumérique	[a-zA-Z0-9_]
\\W	Tout sauf un caractère alphanumérique	[^a-zA-Z0-9_]

Liste d'exemples pour les pattern

- **Exemple 1:**

On peut que le nom de d'une filière soit de la forme suivante:

TS-TDM

TS-TDI

TS-TRI

.....

Vous remarques que les noms des filières commencent toutes par "TS-" suivi du nom de la filière.

Donc ce qu'on ici c'est un format ou modèle.

Comment écrire ce modèle?

Solution : "^TS\\-[A-Z]{2,}$"

Liste d'exemples pour les pattern

- Exemple 2:

Le code d'un livre doit être composé du mot: ISBN suivi d'un tiret suivi de 10 ou 13 chiffres.

ISBN-8975432451 est un code valide.

Comment écrire ce modèle?

Solution : "^ISBN\\-([0-9]{10}|[0-9]{13})$" ou bien

"^ISBN\\-(\\d{10}|\\d{13})$"

Liste d'exemples pour les pattern

- Exemple 2:

Le code d'un livre doit être composé du mot: ISBN suivi d'un tiret suivi de 10 ou 13 chiffres.

ISBN-8975432451 est un code valide.

Comment écrire ce modèle?

Solution : "^ISBN\\-([0-9]{10}|[0-9]{13})$" ou bien

"^ISBN\\-(\\d{10}|\\d{13})$"

Validation d'une chaine de caractère

- Une fois le modèle de l'expression régulière est construit, on peut on peut vérifier la validité d'une chaine de caractère à l'aide de la fonction test:

test(chaîne): cette méthode retourne **true** si "chaîne" respecte le format (modèle) de l'expression régulières , sinon elle retourne **false**.

Remarque: la chaine peut être délimitée entre guillemets ou entre //

Validation d'une chaine de caractère

***Exemple*:**

```
var maChaine=document.getElementById("z").value;

var modele=  "^ISBN\-([0-9]{10}|[0-9]{13})$" ;

// var modele=  /^ISBN\-([0-9]{10}|[0-9]{13})$/ ;

var reg=new RegExp(modele);

if (reg.test(maChaine)==false){
    alert("la chaine ne correspond pas au modèle !!");
}
```

La méthode exec()

- La méthode **exec()** de RegExp va retournera **la première occurrence** trouvée dans la chaîne traitée correspondant au motif., ou **null** dans le cas contraire.

```
var  chaine="bonbon"

var expression=new RegExp("^b")
if(expression.exec(chaine)!=null){
    alert("correspondance")
}
else{
    alert("pas de correspondance")
}
```

La méthode search()

- La méthode search() permet d'effectuer une recherche dans une chaine de caractères à partir d'une expression régulière fournie en argument.
- Cette méthode va retourner **la position** à laquelle a été trouvée la première occurrence de l'expression recherchée dans une chaîne de caractères ou **-1** si rien n'est trouvé.

```
var  chaine="bonbon"

var expression=new RegExp("^b")
if(chaine.search(expression)!=-1){
    alert("correspondance")
}
else{
    alert("pas de correspondance")
}
```

La fonction match()

- La méthode match() permet de renvoyer **toutes les occurrences trouvées** dans la chaîne de caractères qui correspondent au motif.
- S'il n'y a pas de correspondance elle retourne **null**.