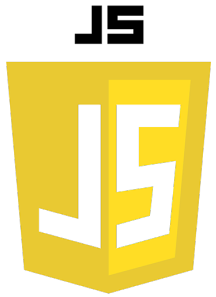


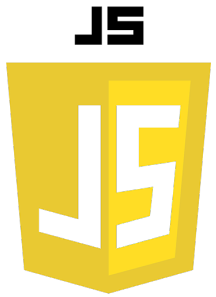
LES REGEX

Une expression régulière (aussi abrégée regex) est une séquence de caractères qui définit un schéma de recherche.

Les expressions régulières vont nous permettre de vérifier la présence de certains caractères dans une chaine de caractères, en comparant l'expression régulière à une chaine de caractères.

Nous allons très souvent utiliser les expressions régulières pour filtrer et vérifier la validité des données envoyées par les utilisateurs via des formulaires par exemple.

Notez que les expressions régulières n'appartiennent pas au Javascript mais constituent un langage en soi. Cependant, le javascript a créé des méthodes qui vont nous permettre d'exploiter toute la puissance des expressions régulières.



Syntaxe de base :

Une expression régulière est définie entre deux barres obliques / /.

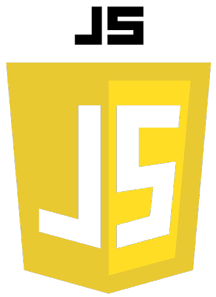
```
const regex = /pattern/;
```

Drapeaux (Flags) :

Les drapeaux sont utilisés pour indiquer diverses options pour l'expression régulière.

- g: global
- i: insensible à la casse
- m: multiligne

```
const regex = /pattern/gi;
```



Méthodes utilisant des regex :

test() :

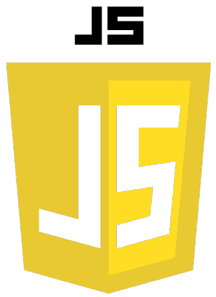
La méthode test() retourne un booléen indiquant si le motif est trouvé dans la chaîne.

```
const regex = /hello/;
console.log(regex.test("hello world")); // true
```

replace() :

La méthode replace() remplace le texte qui correspond à l'expression régulière.

```
const str = "apple";
console.log(str.replace(/a/g, "b")); // "bpple"
```



match() :

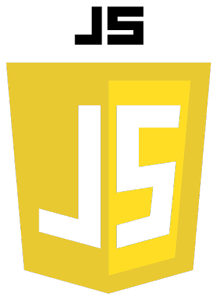
La méthode match() est une méthode de chaîne qui retourne un tableau contenant toutes les correspondances.

```
const str = "apple";
console.log(str.match(/a/g)); // ["a"]
```

exec() :

La méthode exec() retourne un tableau contenant toutes les occurrences trouvées.

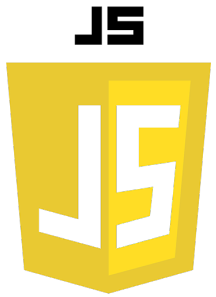
```
const regex = /a/g;
console.log(regex.exec("apple")); // ["a"]
```



```
const str = "J'aime jouer de la guitare.";
const regex = /guitare/;

if (regex.test(str)) {
  console.log('VRAI');
} else {
  console.log('FAUX');
}
```

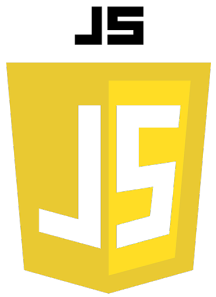
Si vous exécutez ce code, vous verrez qu'il affiche VRAI parce que le mot « guitare » a été trouvé dans la phrase « J'aime jouer de la guitare. ».
Retenez bien ce petit bout de code. Nous allons le garder un moment en changeant parfois la regex, parfois la phrase dans laquelle on fait la recherche.



Chaîne	Regex	Résultat
J'aime jouer de la guitare	/guitare/	VRAI
J'aime jouer de la guitare	/piano/	FAUX

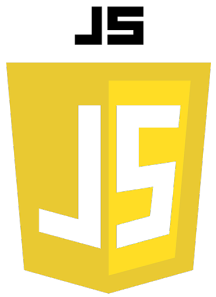
les regex font la différence entre majuscules et minuscules ; on dit qu'elles sont « sensibles à la casse »

Chaîne	Regex	Résultat
J'aime jouer de la guitare	/Guitare/	FAUX
J'aime jouer de la guitare	/GUITARE/	FAUX



On va justement utiliser l’option i :

Chaîne	Regex	Résultat
J'aime jouer de la guitare	/Guitare/i	VRAI
Vive la GUITARE !	/guitare/i	VRAI
Vive la GUITARE !	/guitare/	FAUX



On va maintenant utiliser le symbole OU, que vous avez déjà vu dans le chapitre sur les conditions : c'est la barre verticale « | ».

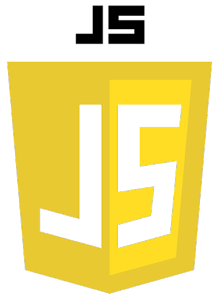
Grâce à elle, vous allez pouvoir laisser plusieurs possibilités à votre regex. Ainsi, si vous tapez :

`/guitare|piano/`

cela veut dire que vous cherchez soit le mot « guitare », soit le mot « piano ». Si un des deux mots est trouvé, la regex répond VRAI.

Voici quelques exemples :

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	/guitare | piano/	VRAI
J'aime jouer du piano.	/guitare | piano/	VRAI
J'aime jouer du banjo.	/guitare | piano/	FAUX
J'aime jouer du banjo.	/guitare | piano | banjo/	VRAI



Début et fin de chaîne

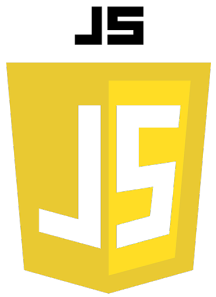
Les regex permettent d'être très précis. Jusqu'ici en effet le mot pouvait se trouver n'importe où. Mais supposons que l'on veuille que la phrase commence ou se termine par ce mot.
Nous allons avoir besoin des deux symboles suivants, retenez-les :

- ^(accent circonflexe) : indique le début d'une chaine ;
- $(dollar) : indique la fin d'une chaine.

Ainsi, si vous voulez qu'une chaîne commence par « Bonjour », il faudra utiliser la regex : **`/^Bonjour/`**

Si vous placez le symbole « ^ » devant le mot, alors ce mot devra obligatoirement se trouver au début de la chaîne, sinon on vous répondra FAUX.
De même, si on veut vérifier que la chaîne se termine par « zéro », on écrira cette regex :
`/zéro$/`

Chaîne	Regex	Résultat
Bonjour petit zéro	/^Bonjour/	VRAI
Bonjour petit zéro	/zéro$/	VRAI
Bonjour petit zéro	/^zéro/	FAUX
Bonjour petit zéro !!!	/zéro$/	FAUX

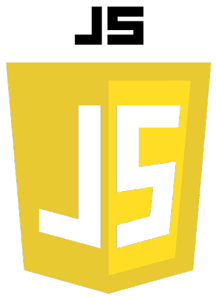


Les classes de caractère :

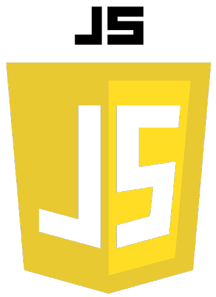
Grâce à ce qu'on appelle les classes de caractères, on peut faire varier énormément les possibilités de recherche.
Tout cela tourne autour des crochets. On place une classe de caractères entre crochets dans une regex.
Cela nous permet de tester beaucoup de possibilités de recherche à la fois, tout en étant très précis.

```
/gr[io]s/
```

Entre crochets, c'est ce qu'on appelle une classe de caractères. Cela signifie qu'une des lettres à l'intérieur peut convenir.
Dans le cas présent, notre regex reconnait deux mots : « gris » et « gros ». C'est un peu comme le OU qu'on a appris tout à l'heure, sauf que ça s'applique ici à une lettre et non pas à un mot.



Chaîne	Regex	Résultat
La nuit, tous les chats sont gris	/gr[aoi]s/	VRAI
Berk, c'est trop gras comme nourriture	/gr[aoi]s/	VRAI
Berk, c'est trop gras comme nourriture	/gr[aoi]s$/	FAUX
Je suis un vrai zéro	/[aeiouy]$/	VRAI
Je suis un vrai zéro	/^[aeiouy]/	FAUX



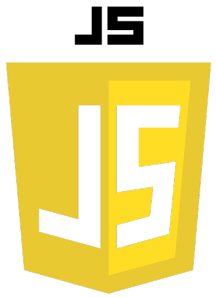
Les intervalles de classe

Grâce au symbole « - » (le tiret), on peut autoriser toute une plage de caractères.
Par exemple, [abcdefghijklmnopqrstuvwxyz]. Vous avez le droit d'écrire : [a-z]

En plus, ça fonctionne aussi avec les chiffres :[0-9]

Vous pouvez écrire deux plages à la fois dans une classe :[a-z0-9]. Cela signifie « N'importe quelle lettre (minuscule) OU un chiffre ».

Chaîne	Regex	Résultat
Cette phrase contient une lettre	/[a-z]/	VRAI
cette phrase ne comporte ni majuscule ni chiffre	/[A-Z0-9]/	FAUX
Je vis au 21e siècle	/^[0-9]/	FAUX
Une balise de titre HTML	/<h[1-6]>/	VRAI



Et pour dire que je n'en veux pas ?
Si vous ne voulez PAS des caractères que vous énumérez dans votre classe, vous devrez placer le symbole « ^ » au début de la classe.

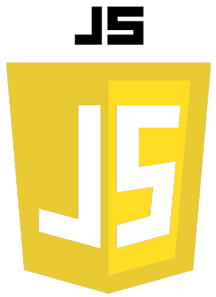
En effet, « ^ »placé à l'intérieur d'une classe, cela sert à dire que vous ne VOULEZ PAS de ce qui se trouve à l'intérieur de cette classe.

Ainsi, la regex suivante :

`/[^0-9]/`

… signifie que vous voulez que votre chaîne comporte au moins un caractère qui ne soit pas un chiffre.

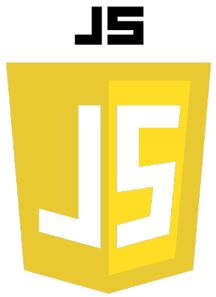
Chaîne	**Regex**	**Résultat**
Cette phrase contient autre chose que des chiffres	/[^0-9]/	VRAI
cette phrase contient autre chose que des majuscules et des chiffres	/[^A-Z0-9]/	VRAI
Cette phrase ne commence pas par une minuscule	/^[^a-z]/	VRAI
Cette phrase ne se termine pas par une voyelle	/[^aeiouy]$/	FAUX
ScrrmmmblllGnngngnngnMmmmmffff	/[^aeiouy]/	VRAI



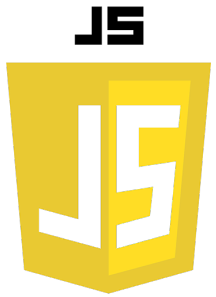
Les quantificateurs :

Les quantificateurs sont des symboles qui permettent de dire combien de fois peuvent se répéter un caractère ou une suite de caractères.
Par exemple, pour reconnaître une adresse e-mail comme francois@free.fr, il va falloir dire : « Elle commence par une ou plusieurs lettres, suivie(s) d'une@(arobase), suivie de deux lettres au moins, elles-mêmes suivies d'un point, et enfin de deux à quatre lettres (pour le.fr,.com, mais aussi.info.

- ? (point d'interrogation) : ce symbole indique que la lettre est facultative. **Elle peut y être 0 ou 1 fois**. Ainsi, /a?/ reconnaît 0 ou 1 « a ».
- + (signe plus) : la lettre est obligatoire. **Elle peut apparaître 1 ou plusieurs fois**. Ainsi, /a+/ reconnaît « a », « aa », « aaa », « aaaa », etc. ;
- * (étoile) : la lettre est facultative. **Elle peut apparaître 0, 1 ou plusieurs fois**. Ainsi, /a*/ reconnaît « a », « aa », « aaa », « aaaa », etc. Mais s'il n'y a pas de « a », ça fonctionne aussi !



Chaîne	Regex	Résultat
Eeeee	/e+/	VRAI
Ooo	/u?/	VRAI
Magnifique	/[0-9]+/	FAUX
Yahooooooo	/^Yaho+$/	VRAI
Yahooooooo c'est génial !	/^Yaho+$/	FAUX
Blablablablabla	/^Bla(bla)*$/	VRAI



Être plus précis grâce aux accolades

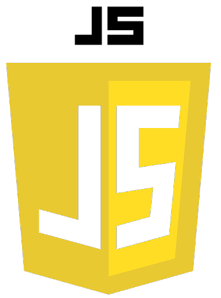
Parfois on aimerait indiquer que la lettre peut être répétée quatre fois, ou de quatre à six fois… bref, on aimerait être plus précis sur le nombre de répétitions.
C'est là qu'entrent en jeu les accolades.

Il y a trois façons d'utiliser les accolades :

- {3} : si on met juste un nombre, cela veut dire que la lettre (ou le groupe de lettres s'il est entre parenthèses) doit être répétée **3 fois exactement**.
  /a{3}/ fonctionne donc pour la chaîne « aaa ».
- {3,5} : ici, on a plusieurs possibilités. On peut avoir la lettre de **3 à 5 fois**.
  /a{3,5}/ fonctionne pour « aaa », « aaaa », « aaaaa ».
- {3,} : si vous mettez une virgule, mais pas de 2e nombre, ça veut dire qu'il peut y en avoir jusqu'à l'infini. Ici, cela signifie « **3 fois ou plus** ».
  /a{3,}/ fonctionne pour « aaa », « aaaa », « aaaaa », « aaaaaa », etc. Je ne vais pas tous les écrire, ça serait un peu long.

Chaîne	Regex	Résultat
Eeeee	/e{2,}/	VRAI
Blablablabla	/^Bla(bla){4}$/	FAUX
546781	/^[0-9]{6}$/	VRAI

les métacaractères



Les métacaractères qu'il faut connaître sont les suivants :

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

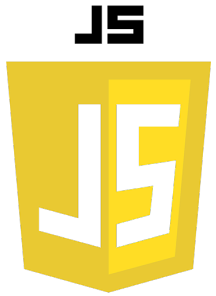
Pour la plupart d'entre eux, vous les connaissez déjà.
Ainsi, le dollar « $ » est un caractère spécial parce qu'il permet d'indiquer une fin de chaîne.
De même pour l'accent circonflexe, les parenthèses, les crochets, les accolades et les symboles « ? + * ».
Pour le point « . » et l'antislash « \ », vous ne les connaissez pas mais vous n'allez pas tarder à les apprendre.

Comment écrire la regex : /Quoi ?/

Le point d'interrogation, vous le savez, sert à dire que la lettre juste avant est facultative (elle peut apparaître 0 ou 1 fois). Ici, l'espace devant le point d'interrogation serait donc facultatif, mais ce n'est pas ce qu'on veut faire !
Alors, comment faire pour faire comprendre qu'on recherche « Quoi ? » alors que le point d'interrogation a déjà une signification ?
Il va falloir **l'échapper**. Cela signifie que vous devez placer en fait un antislash « \ » devant un caractère spécial.
Ainsi, la bonne regex serait :

/Quoi \?/

Ici, l'antislash sert à dire que le point d'interrogation juste après n'est pas un symbole spécial, mais bel et bien une lettre comme une autre.



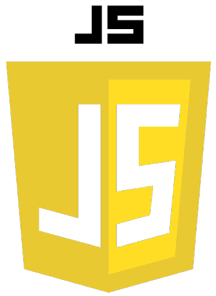
Chaîne	regex	Résultat
Je suis impatient !	/impatient \!/	VRAI
Je suis (très) fatigué	/\(très\) fatigué/	VRAI
J'ai sommeil…	/sommeil\.\.\./	VRAI
Le smiley :-\	/:-\\/	VRAI

Le cas des classes

Pas besoin de l'échapper : à l'intérieur de crochets les métacaractères… ne comptent plus
Ainsi, cette regex marche très bien :
/[a-z?+*{}]/
Elle signifie qu'on a le droit de mettre une lettre, un point d'interrogation, un signe+, etc

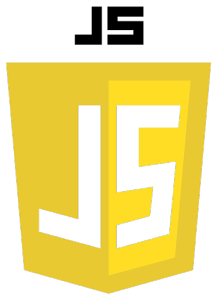
3 cas particuliers, cependant.

- « / » : il sert toujours à indiquer la fin de la regex. Pour l'utiliser, vous DEVEZ mettre un antislash devant, même dans une classe de caractères.
- «] » (crochet fermant) : normalement, le crochet fermant indique la fin de la classe. Si vous voulez vous en servir comme d'un caractère que vous recherchez, il faut là aussi mettre un antislash devant.
- « - » (tiret) : encore un cas un peu particulier. Le tiret – vous le savez – sert à définir un **intervalle de classe** (comme[a-z]). Et si vous voulez ajouter le tiret dans la liste des caractères possibles ? Eh bien il suffit de le mettre soit au début de la classe, soit à la fin. Par exemple :[a-z0-9-]permet de chercher une lettre, un chiffre ou un tiret.



Les classes abrégées

Raccourci	Signification
\d	Indique un chiffre. Ça revient exactement à taper[0-9]
\D	Indique ce qui n'est PAS un chiffre. Ça revient à taper[^0-9]
\w	Indique un caractère alphanumérique ou un tiret de soulignement. Cela correspond à[a-zA-Z0-9_]
\W	Indique ce qui n'est PAS un mot. Si vous avez suivi, ça revient à taper[^a-zA-Z0-9_]
\t	Indique une tabulation
\n	Indique une nouvelle ligne
\r	Indique un retour chariot
\s	Indique un espace blanc
\S	Indique ce qui n'est PAS un espace blanc (\t \n \r)
.	Indique n'importe quel caractère. Il autorise donc tout !

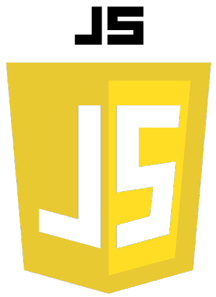


Regex d’un numéro de téléphone :

C’est un numéro de 10 chiffres.

- le premier chiffre est TOUJOURS un 0 ;
- le second chiffre va de 1 à 6 (1 pour la région parisienne… 6 pour les téléphones portables), mais il y a aussi le 8 (ce sont des numéros spéciaux). À noter que le 7 et le 9 sont utilisés mais que nous ne les prendrons pas en compte dans nos exemples ;
- ensuite viennent les 8 chiffres restants (ils peuvent aller de 0 à 9 sans problème).

Pour commencer, et pour faire simple, on va supposer que l'utilisateur entre le numéro de téléphone sans mettre d'espace ni quoi que ce soit (mais on complique juste après, et vous verrez que c'est là le véritable intérêt des regex).
Ainsi, le numéro de téléphone doit ressembler à ça : « 0153789999 ». Comment écrire une regex qui corresponde à un numéro de téléphone comme celui-ci ?



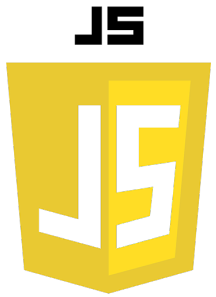
Voici comment je procède, dans l'ordre, pour construire cette regex.

1. Primo, on veut qu'il y ait UNIQUEMENT le numéro de téléphone. On va donc commencer par mettre les symboles ^ et $ pour indiquer un début et une fin de chaîne:/^$/
2. Continuons. On sait que le premier caractère est forcément un 0. On tape donc :
   /^0$/
3. Le 0 est suivi d'un nombre allant de 1 à 6, sans oublier le 8 pour les numéros spéciaux. Il faut donc utiliser la classe[1-68], qui signifie « Un nombre de 1 à 6 OU le 8 » :
   /^0[1-68]$/
4. Ensuite, viennent les 8 chiffres restants, pouvant aller de 0 à 9. Il nous suffit donc d'écrire[0-9]{8}pour indiquer que l'on veut 8 chiffres. Au final, ça nous donne cette regex :
   /^0[1-68][0-9]{8}$/

Maintenant, on va supposer que la personne peut taper un espace tous les deux chiffres (comme c'est courant de le faire en France), mais aussi un point ou un tiret. Notre regex devra donc accepter les numéros de téléphone suivants :

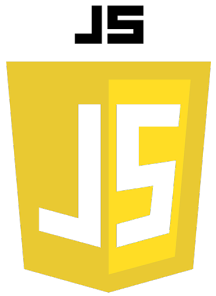
0153789999, 01 53 78 99 99, 01-53-78-99-99, 01.53.78.99.99, 0153 78 99 99, 0153.78 99-99

- etc.



On reprend donc la création de notre regex.

1. Primo, le 0 puis le chiffre de 1 à 6 sans oublier le 8. Ça, ça ne change pas : /^0[1-68]$/
2. Après ces deux premiers chiffres, il peut y avoir soit un espace, soit un tiret, soit un point, soit rien du tout (si les chiffres sont attachés). On va donc utiliser la classe[-.](tiret, point, espace). Mais comment faire pour dire que le point (ou le tiret, ou l'espace) n'est pas obligatoire ? Avec le point d'interrogation, bien sûr ! Ça nous donne:/^0[1-68][-.]?$/
3. Après le premier tiret (ou point, ou espace, ou rien), on a les deux chiffres suivants. On doit donc rajouter[0-9]{2}à notre regex.
   /^0[1-68][-.]?[0-9]{2}$/
4. Et maintenant, réfléchissez. Il y a moyen de terminer rapidement : on a juste besoin de dire que « [-.]?[0-9]{2} » doit être répété quatre fois, et notre regex est terminée ! On va se servir des parenthèses pour entourer le tout, et placer un {4} juste après pour indiquer que tout ça doit se répéter quatre fois. Ce qui nous donne finalement :
   /^0[1-68]([-.]?[0-9]{2}){4}$/



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Validation du numéro de téléphone</title>
</head>
<body>

<p id="result"></p>

<form id="phoneForm">
    <p>
        <label for="telephone">Votre téléphone ?</label>
        <input id="telephone" name="telephone" /><br />
        <input type="submit" value="Vérifier le numéro" />
    </p>
</form>
<script src= »script.js"></script>
</body>
</html>
```

```
let phoneForm = document.getElementById('phoneForm’);
phoneForm.addEventListener('submit', function(event) {
        event.preventDefault(); // Empêche la soumission réelle du formulaire
        let phone = document.getElementById('telephone').value;
        phone = phone.replace(/<[^>]*>/g, ""); // on remplace les balises html par un espace
        if (/^0[1-68]([-. ]?[0-9]{2}){4}$/.test(phone)) {
            document.getElementById('result').innerHTML = 'Le numéro ' + phone + ' est un numéro <strong>valide</strong> !';
        } else {
            document.getElementById('result').innerHTML = 'Le numéro ' + phone + ' n\'est pas valide, recommencez !';
        }
    });
```