Maîtriser les Expressions Régulières

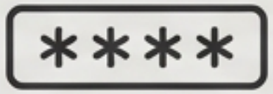
L'art de la recherche de motifs en JavaScript

Le Défi Quotidien : La Validation des Données

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

Comment s’assurer qu'un numéro de téléphone est au bon format ?

✓ ✗



Le mot de passe respecte-t-il les critères de sécurité ?

✓ ✗

Comment filtrer efficacement des informations dans une longue chaîne de caractères ?

✓ ✗

La Solution : Un Langage de Schémas de Recherche

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

Point Clé: Les expressions régulières ne sont pas du JavaScript, mais un langage en soi. JavaScript nous donne simplement les méthodes pour les utiliser.

Syntaxe Fondamentale:

```
const regex = /votre_pattern/flags;
```

Les "Flags" (Drapeaux): Options qui modifient la recherche.

- `i` : Insensible à la casse (majuscules/minuscules).
- `g`: Global (trouve toutes les occurrences, pas seulement la première).
- `m`: Multiligne.

L'Interaction avec JavaScript : 4 Méthodes Essentielles

`test()` : Le Détective

Retourne un booléen (`true` ou `false`) si le motif est trouvé. Idéal pour la validation.

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

`replace()` : Le Chirurgien

Cherche un motif et le remplace.

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

`match()` : Le Collectionneur

Retourne un tableau contenant toutes les correspondances trouvées.

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

`exec()` : L'Enquêteur

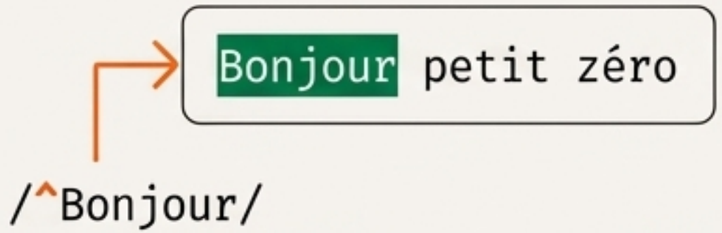
Similaire à `match()`, retourne un tableau détaillé de la première occurrence.

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

Outil N°1 - La Précision : Ancres et Sensibilité à la Casse

Ancre de Début `^`

Le motif doit se trouver au tout début de la chaîne.

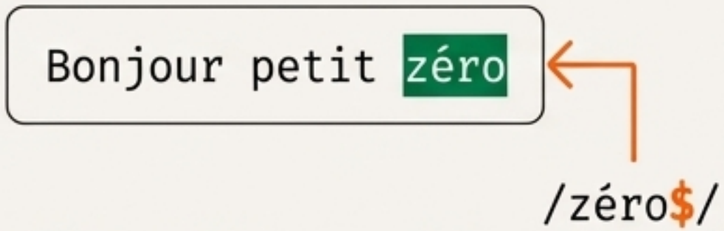



```
const regex = /^Bonjour/;

regex.test("Bonjour petit
zéro"); // VRAI
```

Ancre de Fin `$`

Le motif doit se trouver à la toute fin de la chaîne.

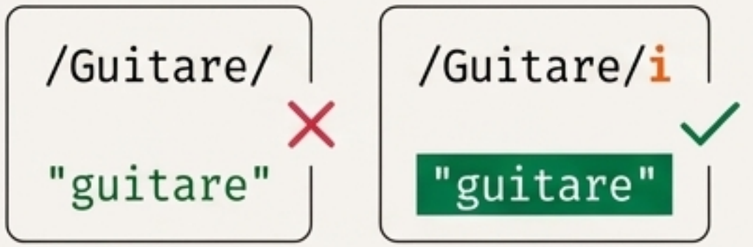



```
const regex = /zéro$/;

regex.test("Bonjour petit
zéro"); // VRAI
```

Le Flag `i`

Pour ignorer la différence entre majuscules et minuscules. Par défaut, les regex sont 'sensibles à la casse'.



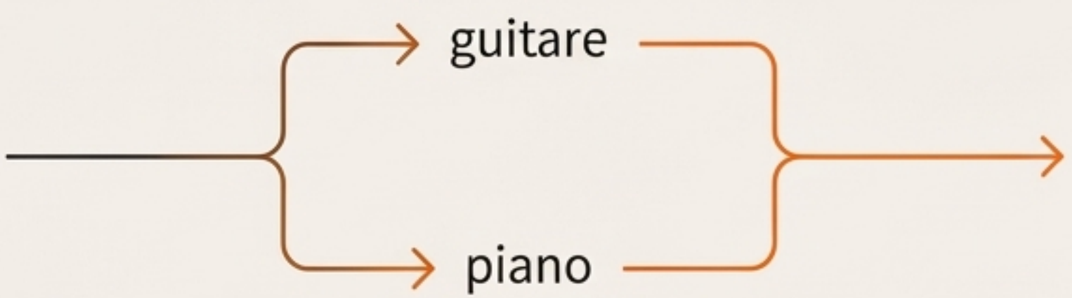

```
const regex = /Guitare/i;

regex.test("J'aime jouer de
la guitare"); // VRAI
```

Outil N°2 - La Flexibilité : Le 'OU' et les Classes de Caractères

L'Opérateur "OU" `|`

Chercher un mot OU un autre.

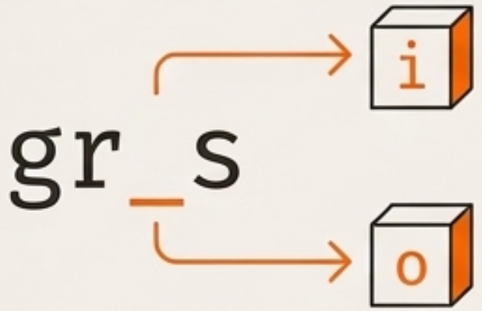



```
const regex = /guitare|piano/

regex.test("J'aime jouer du piano."); // VRAI
regex.test("J'aime jouer du banjo."); // FAUX
```

Les Classes de Caractères `[]`

Chercher un caractère PARMI plusieurs.
C'est un 'OU' au niveau de la lettre.



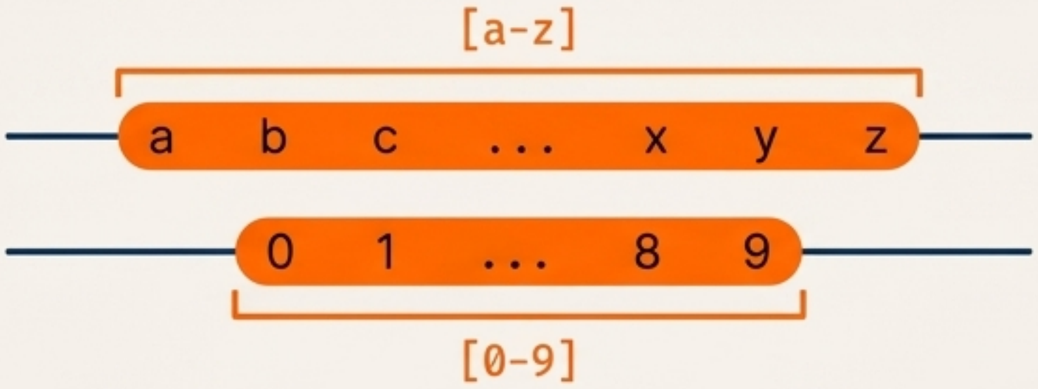

```
const regex = /gr[io]s/;

regex.test("La nuit, tous les chats sont gris"); // VRAI
```

Outil N°3 - L'Efficacité : Intervalles et Négation

Les Intervalles `-`

Définir une plage de caractères.



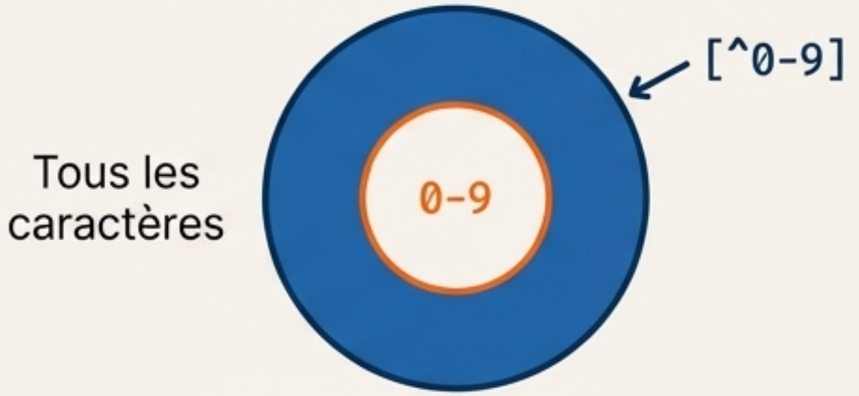
- `[a-z]` : N'importe quelle lettre minuscule.
- `[0-9]` : N'importe quel chiffre.
- On peut les combiner : `[A-Z0-9]` (une majuscule ou un chiffre).

`/<h[1-6]>/` "reconnaît les balises `<h1>`, `<h2>`… jusqu'à `<h6>`.

La Négation `^` (dans une classe)

Exclure des caractères.

Attention : `^` à l'intérieur d'une classe `[]` signifie 'tout SAUF'.




```
const regex = /[^0-9]/;
// Signifie : 'au moins un caractère qui n'est pas un chiffre'.
regex.test("Cette phrase contient autre chose que des chiffres"); // VRAI
```

Outil N°4 - La Puissance : Les Quantificateurs

Les quantificateurs sont des symboles qui permettent de dire combien de fois peut se répéter un caractère ou une suite de caractères.

? (Optionnel)

0 ou 1 fois.

/a?/

reconnaît 0 ou 1 'a'.

+ (Au moins un)

1 ou plusieurs fois.

/a+/

reconnaît 'a', 'aa', 'aaa', etc.

* (Zéro ou plus)

0, 1 ou plusieurs fois.

/a*/

reconnaît une chaîne vide, 'a', 'aa', etc.

Outil N°5 - Le Contrôle : Préciser la Quantité avec les Accolades

{n} (Quantité exacte)

Le caractère ou groupe doit être répété exactement **n** fois.

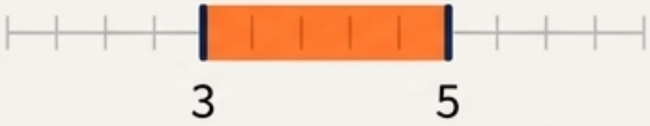
```
/[0-9]{6}/
```

3 7 1 9 5 2 ✓

/[0-9]{6}/ "recherche une suite de 6 chiffres."

{n,m} (Intervalle)

Le caractère ou groupe peut apparaître entre **n** et **m** fois.




```
/a{3,5}/
```

`aaa` ✓	`aaaaa` ✓
`aaaa` ✓	`aa` ✗
`aaaaa` ✓	`aaaaaa` ✗

/a{3,5}/ "reconnaît 'aaa', 'aaaa', ou 'aaaaa'."

{n,} (Quantité minimale)

Le caractère ou groupe doit apparaître au moins **n** fois.


```
/e{2,}/
```

`ee` ✓
`eee` ✓
`e` ✗

/e{2,}/ "reconnaît 'ee', 'eee', etc."

Les Règles du Jeu : Échapper les Méta-caractères

Le Problème

Certains caractères ont un pouvoir spécial :

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

La Solution

Pour chercher le caractère littéral lui-même, il faut l' 'échapper' avec un antislash `\`.

Incorrect

```
/Quoi ?/
```

signifie 'Quo' suivi d'un 'i' optionnel

Correct

```
/Quoi \?/
```

signifie 'Quoi' suivi d'un point d'interrogation littéral

Masterclass - Le Projet : Valider un Numéro de Téléphone Français

Les Règles

Le Défi

Construire une regex qui valide un numéro de téléphone français en respectant toutes ses variations.

- (10) Doit contenir 10 chiffres.
- (0) Le premier chiffre est toujours `0`.
- (1-6,8) Le second chiffre est `1`, `2`, `3`, `4`, `5`, `6` ou `8`.
- (...) Les 8 chiffres restants peuvent être de `0` à `9`.

La Complexité

L'utilisateur peut insérer des séparateurs optionnels (`.`, `-`, ou `␣`) tous les deux chiffres.

Exemples de Formats

Formats Valides à Accepter

- ✓ `0123456789`
- ✓ `01 53 78 99 99`
- ✓ `01-53-78-99-99`
- ✓ `01.53.78.99.99`
- ✓ `0153.78 99-99`
  (Format mixte)

La Construction, Étape par Étape

↳ 01.23.45.67.89 ↵

1. Le Cadre :
La chaîne doit être uniquement un numéro de téléphone.

`/^...$/`

01.23.45.67.89

2. Le Début :
Le premier chiffre est `0`, le second est `[1-6]` ou `8`.

`/^0[1-68]...$/`

01.23.45.67.89

3. Le Motif Répétitif :
Un séparateur optionnel (`-`, `.`, ou ``) suivi de deux chiffres.

`[-.]?`	`.`
`[0-9]{2}`	01.23

`[-.]?[0-9]{2}`

4. L'Assemblage :
On groupe le motif `(...)` et on le répète 4 fois `{4}`.

`/^0[1-68]([-.]?[0-9]{2}){4}$/` → 01.23.45.67.89

La Solution Finale : Élégante et Robuste

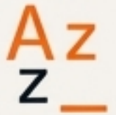
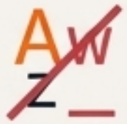
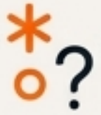
```
/^0[1-68]([-.]?[0-9]{2}){4}$/
```

Mise en Pratique (Code JavaScript)

```javascript
// Extrait du code de validation
let phone = document.getElementById('telephone').value;
const regex = /^0[1-68]([-.]?[0-9]{2}){4}$/;

if (regex.test(phone)) {
    // Numéro valide
    result.innerHTML = 'Le numéro est <strong>valide</strong> !';
} else {
    // Numéro non valide
    result.innerHTML = 'Le numéro n\'est pas valide.';
}
```

Votre Aide-Mémoire : Les Classes Abrégées

Raccourci	Icône	Signification
`\d`		Un chiffre (`[0-9]`)
`\D`		Tout sauf un chiffre (`[^0-9]`)
`\w`		Un caractère alphanumérique ou underscore (`[a-zA-Z0-9_]`)
`\W`		Tout sauf un caractère de mot (`[^a-zA-Z0-9_]`)
`\s`		Un espace blanc (espace, tabulation `\t`, etc.)
`\S`		Tout sauf un espace blanc
`.`		N’importe quel caractère (sauf la nouvelle ligne)

Vous Avez la Boîte à Outils. Maintenant, Construisez !

Résumé du Parcours

Vous avez appris à décomposer un problème complexe et à assembler une solution précise et puissante avec les expressions régulières.

Appel à l'Action

La maîtrise vient avec la pratique. Essayez de :

- Valider une adresse e-mail.
- Créer une regex pour une URL.
- Analyser des fichiers de log pour trouver des erreurs spécifiques.

Ressources Recommandées

Pour tester et visualiser vos regex en direct :

Regex101.com →

RegExr.com →