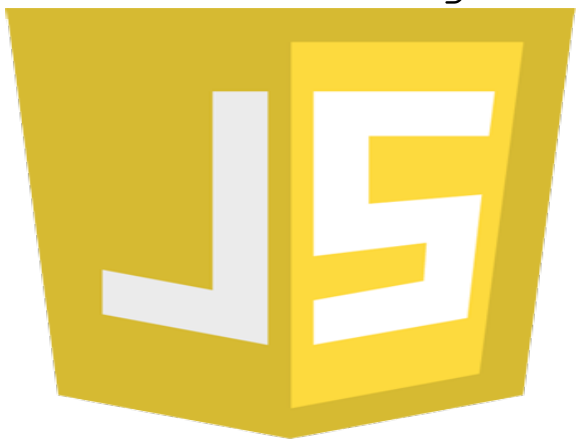
JavaScript : concepts avancés

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

1. Opérateurs
   - L'opérateur `in`
   - L'opérateur `delete`
   - L'opérateur `instanceof`
2. Exceptions
3. Hoisting
4. Expressions régulières
5. Quelques erreurs JavaScript
6. Conventions et bonnes pratiques

JavaScript

Quelques opérateurs

- `typeof` : pour obtenir le type d'une variable
- `new` : pour créer un objet en utilisant un constructeur
- `this` : pour désigner l'objet courant

JavaScript

Quelques opérateurs

- `typeof` : pour obtenir le type d'une variable
- `new` : pour créer un objet en utilisant un constructeur
- `this` : pour désigner l'objet courant

Autres opérateurs

- `in`
- `delete`
- `instanceof`
- `...`

JavaScript

L'opérateur `in`

Il permet de tester si

- un indice est dans le tableau (inférieur à la taille de ce dernier)
- une méthode appartient à un objet
- ...

JavaScript

Exemple avec un tableau

```
var tab = [2,3,5];
if (2 in tab)
  console.log("oui");
// affiche oui
```

JavaScript

Exemple avec un tableau

```
var tab = [2,3,5];
if (2 in tab)
  console.log("oui");
// affiche oui
```

Exemple avec un objet de type chaîne de caractère

```
var str = new String("bonjour");
if ("length" in str)
  console.log("oui");
// affiche oui
```

JavaScript

Exemple avec un tableau

```
var tab = [2,3,5];
if (2 in tab)
  console.log("oui");
// affiche oui
```

Exemple avec un objet de type chaîne de caractère

```
var str = new String("bonjour");
if ("length" in str)
  console.log("oui");
// affiche oui
```

Exemple avec un autre type d'objet

```
var perso = { nom: 'wick', prenom: 'john' };
if ("nom" in perso)
  console.log("oui");
// affiche oui
```

JavaScript

L'opérateur `delete`

- Il permet de supprimer
  - une variables non déclarée explicitement (avec `var`)
  - un attribut d'objet
  - un élément de tableau
- retourne true si la suppression se termine correctement, `false` sinon.

JavaScript

Exemple avec une variable de type `object`

```
perso = { nom: 'wick', prenom: 'john' };
console.log(perso);
// affiche { nom: 'wick', prenom: 'john' }

delete perso.prenom ;
console.log(perso);
// affiche { nom: 'wick' }
```

JavaScript

Exemple avec une variable de type `object`

```
perso = { nom: 'wick', prenom: 'john' };
console.log(perso);
// affiche { nom: 'wick', prenom: 'john' }

delete perso.prenom ;
console.log(perso);
// affiche { nom: 'wick' }
```

Ceci génère une erreur car l'objet a été supprimé

```
perso = { nom: 'wick', prenom: 'john' };
delete perso;
console.log(perso);
```

JavaScript

Exemple avec une variable de type `object`

```
perso = { nom: 'wick', prenom: 'john' };
console.log(perso);
// affiche { nom: 'wick', prenom: 'john' }

delete perso.prenom ;
console.log(perso);
// affiche { nom: 'wick' }
```

Ceci génère une erreur car l'objet a été supprimé

```
perso = { nom: 'wick', prenom: 'john' };
delete perso;
console.log(perso);
```

Une variable déclarée avec var ne sera pas supprimée

```
var perso = { nom: 'wick', prenom: 'john' };
delete perso;
console.log(perso);
// affiche { nom: 'wick', prenom: 'john' }
```

JavaScript

L'opérateur `instanceof`

retourne `true` si l'objet donné est du type spécifié, `false` sinon.

JavaScript

L'opérateur `instanceof`

retourne `true` si l'objet donné est du type spécifié, `false` sinon.

Exemple

```
var jour = new Date(2019, 06, 06);
if (jour instanceof Date) {
  console.log("oui");
}
// affiche oui
```

JavaScript

Considérons la fonction suivante

```
function produit (a, b) {
  return a * b;
}
```

JavaScript

Considérons la fonction suivante

```
function produit (a, b) {
  return a * b;
}
```

Appeler la fonction avec des nombres donne le résultat suivant

```
console.log(produit (2, 3));
// affiche 6
```

JavaScript

Considérons la fonction suivante

```
function produit (a, b) {
  return a * b;
}
```

Appeler la fonction avec des nombres donne le résultat suivant

```
console.log(produit (2, 3));
// affiche 6
```

Appeler la fonction avec un nombre et une chaîne de caractère donne le résultat suivant

```
console.log(produit (2, "a"));
// affiche NaN
```

JavaScript

On peut lancer une exception si un des paramètres n'est pas de type `number`

```
function produit (a, b) {
  if (isNaN (a) || isNaN (b))
    throw "Les paramètres doivent être de type number";
  return a * b;
}
```

JavaScript

On peut lancer une exception si un des paramètres n'est pas de type `number`

```
function produit (a, b) {
  if (isNaN (a) || isNaN (b))
    throw "Les paramètres doivent être de type number";
  return a * b;
}
```

L'appel de la fonction doit être entouré par un bloc `try ... catch` **pour capturer et traiter l'éventuelle exception lancée**

```
try {
  console.log(produit (2, "a"));
}
catch(e) {
  console.error(e);
}
```

`console.error()` **permet d'afficher dans la console une erreur en rouge.**

JavaScript

En exécutant le code précédent, le résultat est

Les paramètres doivent être de type number

JavaScript

En exécutant le code précédent, le résultat est

Les paramètres doivent être de type number

Remarque

On peut lancer une exception de type `string`, `number`, `boolean`...

JavaScript

Hoisting (la remontée)

- Possibilité d'utiliser une variable avant de la déclarer
- Le compilateur commence par lire toutes les déclarations, ensuite il traite le reste du code
- Les constantes et les variables déclarées avec `let` ne supporte pas le **hoisting**

JavaScript

Hoisting (la remontée)

- Possibilité d'utiliser une variable avant de la déclarer
- Le compilateur commence par lire toutes les déclarations, ensuite il traite le reste du code
- Les constantes et les variables déclarées avec `let` ne supporte pas le **hoisting**

Malgré l'utilisation du mode strict, l'affichage d'une variable non-déclarée ne génère pas d'erreur grâce à la remontée

```
"use strict";
x = 7;
console.log(x);
var x;
```

JavaScript

Expressions régulières

- outil de recherche puissant adopté par la plupart des langages de programmation
- facilitant la recherche dans les chaînes de caractères (et donc le remplacement, le calcul de nombre d'occurrences...)
- permettent de vérifier si des chaînes de caractères respectent certains formats (email, numéro de téléphone...)
- syntaxe plus ou moins proche pour tous les langages de programmation

JavaScript

Plusieurs méthodes de recherche disponibles

- `chaine1.search(chaine2)` : permet de chercher et retourner la position de la première occurrence de `chaine2` dans `chaine1`
- `chaine1.test(chaine2)` : permet de chercher et retourner `true` si `chaine2` contient `chaine1`, `false` sinon.
- `chaine1.replace(chaine2, chaine3)` : permet de remplacer la première occurrence de `chaine2` dans `chaine1` par `chaine3`
- `chaine1.exec(chaine2)` : permet de chercher `chaine1` dans `chaine2` et retourner un objet contenant plusieurs données sur la recherche (position de la première occurrence, chaîne recherchée...).
- `chaine1.match(chaine2)` : permet de chercher `chaine1` dans `chaine2` et retourner un tableau contenant toutes les occurrences.

JavaScript

Recherche d'une sous-chaîne dans une chaîne de caractère

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

JavaScript

Recherche d'une sous-chaîne dans une chaîne de caractère

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

Recherche avec une expression régulière insensible à la casse

```
var str = "Bonjour tout le monde";
var pos = str.search(/Tout/i);
console.log(pos); // affiche 8
```

JavaScript

Recherche d'une sous-chaîne dans une chaîne de caractère

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

Recherche avec une expression régulière insensible à la casse

```
var str = "Bonjour tout le monde";
var pos = str.search(/Tout/i);
console.log(pos); // affiche 8
```

Retourne -1 si la sous-chaîne n'existe pas

```
var str = "Bonjour tout le monde";
var pos = str.search(/c/i);
console.log(pos); // affiche -1
```

JavaScript

On peut aussi utiliser la fonction de recherche `test` qui retourne `true` si la sous-chaine existe, `false` sinon.

```
var str = /AB/i;
var result = str.test("ababaabbaaab");
console.log(result); // affiche true
```

JavaScript

Remplacer la première occurrence d'une sous-chaîne

```
var chaine = "ababaabbaaab";
var txt = chaine.replace(/ab/,"c");
console.log(txt);
// affiche cabaabbaaab
```

JavaScript

Remplacer la première occurrence d'une sous-chaîne

```
var chaine = "ababaabbaaab";
var txt = chaine.replace(/ab/,"c");
console.log(txt);
// affiche cabaabbaaab
```

Remplacer toutes les occurrences d'une sous-chaîne

```
var chaine = "ababaabbaaab";
var txt = chaine.replace(/ab/g,"c");
console.log(txt);
// affiche ccacbaac
```

JavaScript

Utiliser `exec` pour avoir un résultat sous forme d'un objet

```
var chaine = "abababaabbbaaab";
var str = /AA/i;
var resultat = str.exec(chaine);

console.log("chaîne trouvée : " + resultat[0]);
// affiche chaîne trouvée : aa

console.log("indice de la première occurrence : " +
   resultat.index);
// affiche indice de la première occurrence : 4

console.log("texte complet : " + resultat.input);
// affiche texte complet : abababaabbbaaab
```

JavaScript

Pour trouver toutes les occurrences avec `exec`, il faut utiliser une boucle et ajouter le paramètre `g`

```
var chaine = "abababaabbaaab";
var str = /AA/gi;
var resultat;
while (resultat = str.exec(chaine)) {
        console.log("motif recherché : " + resultat
           [0]);
        console.log("indice de la première
           occurrence : " + resultat.index);
        console.log("texte complet : " + resultat.
           input);
}
```

JavaScript

On peut aussi utiliser `match` qui retourne un tableau contenant toutes occurrences sans utiliser de boucles

```
var chaine = "abababaabbaaab";
var str = /AA/gi;
var resultat;
console.log(chaine.match(str));
// affiche (2) ["aa", "aa"]
```

Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$: minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$: exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$: minimum `n` occurrences de `a` consécutives

Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$: minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$: exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$: minimum `n` occurrences de `a` consécutives


```
var str = /ba?c/i;
console.log(str.test("bac")); // true
```

Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$: minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$: exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$: minimum `n` occurrences de `a` consécutives


```
var str = /ba?c/i;
console.log(str.test("bac")); // true
```

```
var str = /ba?c/i;
console.log(str.test("baac")); // false
```

JavaScript

Contraintes exprimées avec les expressions régulières

- $a \mid b$: `a` ou `b`
- `^` : commence par
- `$` : se termine par
- `()` : le groupe
- `a(?!b)` : `a` non suivi de `b`
- `a(?=b)` : `a` suivi de `b`
- `(?<!a)b` : `b` non précédé par `a`

JavaScript

Contraintes exprimées avec les expressions régulières

- `.` : n'importe quel caractère
- `\d` : un chiffre
- `\D` : tout sauf un chiffre
- `\w` : un caractère alphanumérique
- `\W` : tout sauf un caractère alphanumérique
- `\t` : un caractère de tabulation
- `\n` : un caractère de retour à la ligne
- `\s` : un espace
- ...

JavaScript

Contraintes exprimées avec les expressions régulières

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule ou en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- . . .

JavaScript

Contraintes exprimées avec les expressions régulières

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule ou en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- . . .

**Pour utiliser un caractère réservé (^, $...) dans une expression régulière, il faut le précéder par **

JavaScript

Exercice 1

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère contient 3 occurrences (pas forcement consécutives) de la sous-chaîne `ab`.

- `true` pour `abacccababcc`
- `true` pour `abababccccab`
- `false` pour `baaaaabaccccba`

JavaScript

Exercice 2

Trouver une expression régulière qui permet de déterminer si une chaîne commence par la lettre `a`, se termine par la lettre `b` et pour chaque `x` suivi de `y` (pas forcement consécutives), il existe un `z` situé entre `x` et `y`.

- `true` pour `ab`
- `true` pour `abxyb`
- `true` pour `abxazyb`
- `false` pour `abxuyb`
- `false` pour `abxazyxyb`

JavaScript

Exercice 3

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à une adresse e-mail.

JavaScript

Exercice 3

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à une adresse e-mail.

Exercice 4

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à un numéro de téléphone français.

JavaScript

Quelques erreurs JavaScript

- TypeError : lorsqu'on appelle une méthode sur une variable de type `undefined`
- SyntaxError : lorsqu'on rencontre un symbole inattendu
  ```
  var x = 2 + 3,;
  ```
- ReferenceError : lorsqu'on utilise une variable non-déclarée et non-initialisée
- EvalError : lorsque l'expression passé à la fonction `eval` est mal formulée
- RangeError : lorsqu'une variable de type `number` reçoit une valeur supérieure à sa limite autorisée
- ...

JavaScript

Conventions et bonnes pratiques

- Utilisation du **camelCase** pour les variables (sauf pour les constantes et variables globales) et fonctions
- Terminer chaque instruction par `;`
- Placer un espace avant et après chaque opérateur ou accolade
- Bien indenter son code et ne pas utiliser la touche de tabulation
- Ne pas dépasser 80 caractères par ligne
- Utiliser les valeurs par défaut pour les paramètres d'une fonction
- Avoir un bloc `default` dans `switch`

JavaScript

Conventions et bonnes pratiques

- Éviter les variables globales
- Déclarer les variables au tout début du bloc
- Utiliser les types primitifs et éviter les types objets
- Éviter l'utilisation de `eval()` pour la sécurité de votre programme
- Définir un bloc default dans le `switch`
- Utiliser \ pour indiquer le retour à la ligne pour les chaines de caractères :

```
var str = "Bonjour \
tout le monde"
```