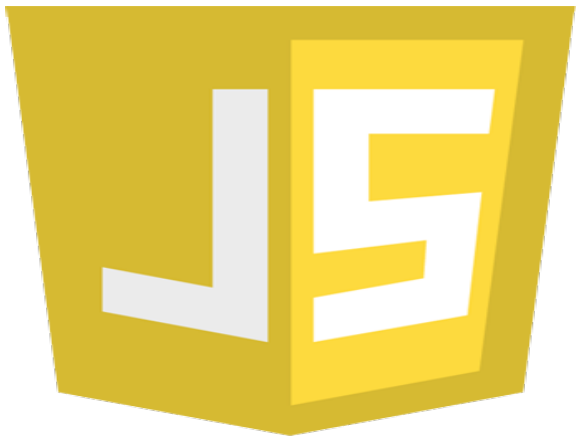
L'algorithmie avec JavaScript



Plan

1. Introduction
2. Intégrer JavaScript dans HTML
3. Console
4. Commentaires
5. Variables
6. Quelques opérations sur les variables
7. Méthodes utiles pour les chaînes de caractères

Plan

8. Conditions et boucles
   - `if`
   - `if ... else`
   - `else if`
   - `switch`
   - Expression ternaire
   - `while`
   - `do ... while`
   - `for`
9. Tableaux
10. Fonctions
    - Fonction de retour (callback)
    - Quelques fonctions connues utilisant le callback
    - Fonction génératrice
    - Closure d'une fonction

Plan

11 Objets
- Attributs
- Méthodes
- Constructeurs
- Prototypes

12 Objets et méthodes prédéfinis

13 Opérateurs
- L'opérateur `in`
- L'opérateur `delete`
- L'opérateur `instanceof`

Plan

14 Constantes

15 Exceptions

16 Hoisting

17 Expressions régulières

18 Promesses
- `async`
- `await`

19 Quelques erreurs JavaScript

20 Conventions et bonnes pratiques

JavaScript

Définition, histoire et rôle

- est un langage de programmation de scripts orienté objet (à prototype)
- créé par Brendan Eich
- presenté par Netscape et Sun Microsystems en décembre 1995
- standardisé par ECMAScript en juin 1997 pour
- permettant de
  - compléter l'aspect algorithmique manquant à HTML (et CSS)
  - rendre plus vivant le site web avec notamment des animations, effets et de l'interaction avec l'internaute (le visiteur, le client...).

JavaScript

Spécificités du JavaScript

- langage faiblement typé
- syntaxe assez proche de celle de Java, C++, C...
- possibilité d'écrire plusieurs instructions sur une seule ligne à condition de les séparer par `;`
- terminer une instruction par `;` est fortement recommandée même si cette dernière est la seule sur la ligne

JavaScript

Autres langages de programmation de scripts

- AppleScript
- JScript, VBScript et TypeScript (Microsoft)
- LiveScript (Netscape) puis JavaScript
- ActionScript (MacroMedia)
- CoffeeScript (Open-source)
- ...

JavaScript

Intégrer JavaScript dans HTML

Trois façons pour définir des scripts JavaScript

- comme valeur d'attribut de n'importe quelle balise HTML
- dans une balise `<script>` de la section `<head>` d'une page HTML
- dans un fichier d'extension `.js` référencé dans une page HTML par la balise `<script>`

JavaScript

Première méthode

```
<button onclick="alert('Hello World');"> Cliquer ici
</button>
```

JavaScript

Deuxième méthode

```
<!DOCTYPE html>
<html>
<head>
<title>First JS Page</title>
<script type="text/javascript">
  function maFonction() {
    alert("Hello World !");
  }
</script>
</head>
<body>
  <button onclick="maFonction()">
    Cliquer ici
  </button>
</body>
</html>
```

JavaScript

Deuxième méthode

```
<!DOCTYPE html>
<html>
<head>
<title>First JS Page</title>
<script type="text/javascript">
  function maFonction() {
    alert("Hello World !");
  }
</script>
</head>
<body>
  <button onclick="maFonction()">
    Cliquer ici
  </button>
</body>
</html>
```

L'attribut `type="text/javascript"` n'est plus nécessaire depuis HTML5.

JavaScript

Troisième méthode

```
<!DOCTYPE html>
<html>
<head>
  <title>First JS Page</title>
  <script src="file.js"></script>
</head>
<body>
  <button onclick="maFonction()"> Cliquer ici</button>
</body>
</html>
```

JavaScript

Troisième méthode

```
<!DOCTYPE html>
<html>
<head>
  <title>First JS Page</title>
  <script src="file.js"></script>
</head>
<body>
  <button onclick="maFonction()"> Cliquer ici</button>
</body>
</html>
```

Contenu du `file.js`

```
function maFonction() {
  alert("Hello World !");
}
```

JavaScript

Pour une box d'affichage avec confirmation

```
var bin = confirm("Press a button!");
alert(bin);
```

JavaScript

Pour une box d'affichage avec confirmation

```
var bin = confirm("Press a button!");
alert(bin);
```

Pour une box d'affichage avec une zone de saisie

```
var str = prompt("Votre nom", "John Wick");
alert(str);
```

JavaScript

La console, pourquoi ?

- permet de contrôler l'avancement de l'exécution d'un programme
  - en affichant le contenu de variables (déboguer)
  - en vérifiant les blocs du code visites (tracer)

JavaScript

La console, pourquoi ?

- permet de contrôler l'avancement de l'exécution d'un programme
  - en affichant le contenu de variables (déboguer)
  - en vérifiant les blocs du code visites (tracer)

Modifions le contenu du `file.js`

```
function maFonction(){
  console.log("Hello World !");
}
```

JavaScript

Où trouve t-on le message ?

- Pour les navigateurs suivants
  - Google chrome
  - Mozilla firefox
  - Internet explorer
- Cliquer sur F12

JavaScript

Existe t-il un autre moyen de tester un programme JS sans passer par un navigateur ?

- Oui, en utilisant **NodeJS** (pour télécharger `https://nodejs.org/en/`)
- Pour tester, utiliser une console telle que
  - Invite de commandes
  - Windows PowerShell
  - Cmder
- Lancer la commande `node nomFichier.js`

JavaScript

Modifions le contenu du `file.js`

```
function maFonction(){
  console.log("Hello World !");
}
maFonction();
```

JavaScript

Modifions le contenu du `file.js`

```
function maFonction(){
  console.log("Hello World !");
}
maFonction();
```

Lancer la commande `node file.js`

JavaScript

Il est aussi possible de définir un raccourci de `console.log`

```
var cl = console.log;
cl("Hello World !");
```

JavaScript

Commentaire sur une seule ligne

```
// commentaire
```

JavaScript

Commentaire sur une seule ligne

```
// commentaire
```

Commentaire sur plusieurs lignes

```
/* le commentaire
la suite
et encore la suite
*/
```

JavaScript

Commentaire sur une seule ligne

```
// commentaire
```

Commentaire sur plusieurs lignes

```
/* le commentaire
la suite
et encore la suite
*/
```

Commentaire pour la documentation

```
/** un commentaire
pour
la documentation
*/
```

JavaScript

Déclaration d'une variable avec le mot clé `var`

```
var x;
```

JavaScript

Déclaration d'une variable avec le mot clé `var`

```
var x;
```

Initialisation

```
x = 0;
```

JavaScript

Déclaration d'une variable avec le mot clé `var`

```
var x;
```

Initialisation

```
x = 0;
```

Déclaration + initialisation

```
var y = 5;
```

JavaScript

Déclaration d'une variable avec le mot clé `var`

```
var x;
```

Initialisation

```
x = 0;
```

Déclaration + initialisation

```
var y = 5;
```

Il est possible de déclarer simultanément plusieurs variables

```
var x = 0, y = 5;
```

JavaScript

Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration

```
z = 5;
```

JavaScript

Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration

```
z = 5;
```

Affectation d'une variable non-déclarée

```
t = x + y;
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée $\Rightarrow$ ReferenceError

```
alert(v);
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée ⇒ ReferenceError

```
alert (v);
```

Une variable déclarée mais non-initialisée a par défaut la valeur `undefined`

```
var w;
alert(w);
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée ⇒ ReferenceError

```
alert (v);
```

Une variable déclarée mais non-initialisée a par défaut la valeur `undefined`

```
var w;
alert (w);
```

`NaN` **: not a number**

```
var x = "bonjour";
n = x * 2;
console.log(n);
```

JavaScript

Quelques types de variable selon la valeur affectée

- number
- string
- boolean
- object
- undefined

JavaScript

Pour récupérer le type de valeur affectée à une variable

```
console.log(typeof 5);
// affiche number

console.log(typeof true);
// affiche boolean

console.log(typeof 5.2);
// affiche number

console.log(typeof "bonjour");
// affiche string

console.log(typeof 'c');
// affiche string

var x;
console.log(typeof x);
// affiche undefined
```

JavaScript

Une variable déclarée avec le mot-clé `var` a une visibilité globale

```
{
  var x = 1;
}
console.log(x);
// affiche 1
```

JavaScript

Une variable déclarée avec le mot-clé `var` a une visibilité globale

```
{
  var x = 1;
}
console.log(x);
// affiche 1
```

Remarque

Une variable déclarée dans un bloc `{ ... }` autre que fonction (`if`, `for`...) a une portée globale.

JavaScript

Addition (qui peut se transformer en concaténation) et conversion

```
var x = 1;
var y = 3;
var z = '8';
var t = "2";
var u = "bonjour";
var v = "3bonjour";
var m;
var n = 2.5;
console.log(x + y); // 4
console.log(x + z); // 18
console.log(x + parseInt(z)); // 9
console.log(x + t); // 12
console.log(x + y + z); // 48
console.log(z + y + x); // 831
console.log(x + u); // 1bonjour
console.log(x + parseInt(u)); // NaN
console.log(x + v); // 13bonjour
console.log(x + parseInt(v)); // 4
console.log(u + v); // bonjour3bonjour
console.log(x + m); // NaN
console.log(x + n); // 3.5
```

JavaScript

Autres opérateurs arithmétiques

- $*$: multiplication
- $-$: soustraction
- $/$: division
- $\%$: reste de la division
- $**$: exponentiel

JavaScript

Attention, l'opérateur + pour les chaînes de caractères est différent de tous les autres opérateurs arithmétiques

```
var a = "2";
var b = '3';
var resultat = a * b;
console.log(resultat);
// affiche 6
```

JavaScript

Attention, l'opérateur `+` pour les chaînes de caractères est différent de tous les autres opérateurs arithmétiques

```
var a = "2";
var b = '3';
var resultat = a * b;
console.log(resultat);
// affiche 6
```

Cependant ce code génère un `NaN`

```
var a = "bon";
var b = 'jour';
var resultat = a * b;
console.log(resultat);
// affiche NaN
console.log(b + parseInt(a));
// affiche jourNaN
```

JavaScript

En JavaScript, une division par 0 ne génère pas d'erreur

```
a = 2;
b = 0;
console.log(a / b);
```

JavaScript

En JavaScript, une division par 0 ne génère pas d'erreur

```
a = 2;
b = 0;
console.log(a / b);
```

Pour convertir une chaîne en entier

```
var a = '4';
b = 2;
console.log(b + parseInt(a));
```

JavaScript

Quelques raccourcis

- `i++;` ≡ `i = i + 1;`
- `i--;` ≡ `i = i - 1;`
- `i += 2;` ≡ `i = i + 2;`
- `i -= 3;` ≡ `i = i - 3;`
- `i *= 2;` ≡ `i = i * 2;`
- `i /= 3;` ≡ `i = i / 3;`
- `i %= 5;` ≡ `i = i % 5;`

JavaScript

Exemple de post-incrémentation

```
var i = 2;
var j = i++;
console.log(i); // affiche 3
console.log(j); // affiche 2
```

JavaScript

Exemple de post-incrémentation

```
var i = 2;
var j = i++;
console.log(i); // affiche 3
console.log(j); // affiche 2
```

Exemple de pre-incrémentation

```
var i = 2;
var j = ++i;
console.log(i); // affiche 3
console.log(j); // affiche 3
```

JavaScript

Pour permuter le contenu de deux variables, on peut utiliser la décomposition

```
a = 2;
b = 0;
[a,b] = [b,a]
```

JavaScript

Pour permuter le contenu de deux variables, on peut utiliser la décomposition

```
a = 2;
b = 0;
[a,b] = [b,a]
```

Pour évaluer une expression arithmétique exprimée sous forme de chaîne de caractères (EvalError si l'expression est mal formulée)

```
var str = "2 + 5 * 3";
console.log(eval(str));
// affiche 17
```

JavaScript

L'opérateur unaire + peut être utilisé pour convertir en nombre

```
console.log(typeof (+"3"));
// affiche number
console.log(+true);
// affiche 1
```

JavaScript

L'opérateur unaire + peut être utilisé pour convertir en nombre

```
console.log(typeof (+"3"));
// affiche number
console.log(+true);
// affiche 1
```

L'opérateur unaire – peut être utilisé pour la négation ou la conversion en nombre

```
console.log(typeof (-"-3"));
// affiche nombre
console.log(-"-3");
// affiche 3
console.log(-true);
// affiche -1
```

JavaScript

Méthodes utiles pour les chaînes de caractères

- `length` : la longueur de la chaîne
- `toUpperCase()` : pour convertir une chaîne de caractères en majuscule
- `toLowerCase()` : pour convertir une chaîne de caractères en minuscule
- `trim()` : pour supprimer les espaces au début et à la fin
- `substr()` : pour extraire une sous-chaîne de caractères
- `indexOf()` : pour retourner la position d'une sous-chaîne dans une chaîne, -1 sinon.
- ...

JavaScript

Pour connaître la longueur d'une chaîne

```
var str = "bonjour";
console.log(str.length);
// affiche 7
```

JavaScript

Pour connaître la longueur d'une chaîne

```
var str = "bonjour";
console.log(str.length);
// affiche 7
```

Pour supprimer les espaces au début et à la fin de la chaîne

```
var str = "  bon jour  ";
console.log(str.length);
// affiche 12

var sansEspace = str.trim();
console.log(sansEspace.length);
// affiche 8
```

JavaScript

Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

JavaScript

Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

On peut aussi préciser le nombre de caractère à extraire

```
var str = "bonjour";
console.log(str.substr(3,2));
// affiche jo
```

JavaScript

Pour extraire une sous-chaîne à partir de l'indice 3 jusqu'à la fin

```
var str = "bonjour";
console.log(str.substr(3));
// affiche jour
```

On peut aussi préciser le nombre de caractère à extraire

```
var str = "bonjour";
console.log(str.substr(3,2));
// affiche jo
```

Pour extraire les trois derniers caractères, on utilise une valeur négative

```
var str = "bonjour";
console.log(str.substr(-3)); //eq substr(4) avec 4 = length - 3
// affiche our
```

JavaScript

Ne pas confondre `substr()` avec `substring()` qui elle prend comme paramètre l'indice de début et l'indice de fin (non-inclus)

```
var str = "bonjour";
console.log(str.substring(1,3));
// affiche on
```

JavaScript

Pour déterminer l'indice d'une sous chaîne dans une chaîne de caractère

```
var str = "bonjour";
console.log(str.indexOf("jour"));
// affiche 3
```

JavaScript

Pour déterminer l'indice d'une sous chaîne dans une chaîne de caractère

```
var str = "bonjour";
console.log(str.indexOf("jour"));
// affiche 3
```

S'il n'y a aucune occurrence, elle retourne -1

```
var str = "bonjour";
console.log(str.indexOf("soir"));
// affiche -1
```

JavaScript

Pour accéder à un caractère d'indice `i` dans une chaîne de caractères

```
// soit directement via l'indice
console.log(str[i]);

// soit en faisant l'extraction d'une sous chaine de
    caractère
console.log(str.substr(i,1));

// soit avec la méthode d'extraction de caractère
console.log(str.charAt(i));
```

JavaScript

Remarque

Appeler une de ces méthodes à partir d'une variable ayant la valeur `undefined` génère une erreur nommée **TypeError**.

JavaScript

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

JavaScript

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

Exemple

```
var x = 3;
if (x > 0)
{
   console.log(x + " est strictement positif");
}
```

JavaScript

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`

```
if (condition1) {
   ...
}
else {
   ...
}
```

JavaScript

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`

```
if (condition1) {
   ...
}
else {
   ...
}
```

Exemple

```
var x = 3;
if (x > 0)
{
   console.log(x + " est strictement positif");
}
else
{
  console.log(x + " est négatif ou nul");
}
```

JavaScript

On peut enchaîner les conditions avec `else if` (et avoir un troisième bloc voire ... un nième)

```
if (condition1)
{
    ...
}
else if (condition2)
{
    ...
}
...
else
{
    ...
}
```

JavaScript

Exemple

```
var x = -3;
if (x > 0)
{
  console.log(x + " est strictement positif");
}
else if (x < 0)
{
  console.log(x + " est strictement négatif");
}
else
{
  console.log(x + " est nul");
}
```

JavaScript

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non

JavaScript

Opérateurs logiques

- && : et
- || : ou
- ! : non

Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3){
...
}
[else ...]
```

JavaScript

Opérateurs logiques

- && : et
- || : ou
- ! : non

Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3){
...
}
[else ...]
```

Pour les conditions, on utilise les opérateurs de comparaison

JavaScript

Opérateurs de comparaison

- == : pour tester l'égalité des valeurs
- != : pour tester l'inégalité des valeurs
- === : pour tester l'égalité des valeurs et des types
- !== : pour tester l'inégalité des valeurs ou des types
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

JavaScript

Structure conditionnelle `switch` : syntaxe

```
switch (nomVariable)
{
  case constante-1:
    groupe-instructions-1;
    break;
  case constante-2:
    groupe-instructions-2;
    break;
  ...
  case constante-N:
    groupe-instructions-N;
    break;
  default:
    groupe-instructions-par-défaut;
}
```

JavaScript

Remarques

- Le `switch` permet **seulement** de tester l'égalité
- Le `break` permet de quitter le `switch` une fois le bloc de case est vérifier
- Il est possible de regrouper plusieurs `case`
- Le bloc `default` est facultatif, il sera exécuter si la valeur de la variable ne correspond à aucune constante de `case`

JavaScript

Structure conditionnelle avec `switch`

```
var x = 5;
switch (x) {
  case 1:
    alert('un');
    break;
  case 2:
    alert('deux');
    break;
  case 3:
    alert('trois');
    break;
  default:
    alert("autre");
}
```

JavaScript

Un multi-case pour un seul traitement

```
var x = 5;
switch (x) {
  case 1:
  case 2:
    alert('un ou deux');
    break;
  case 3:
    alert('trois');
    break;
  case 4:
  case 5:
    alert('quatre ou cinq');
    break;
  default:
    alert("autre");
}
```

JavaScript

La variable dans `switch` peut être

- un nombre
- un caractère
- une chaîne de caractère (contrairement aux C++, C...)

JavaScript

Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

JavaScript

Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

Simplifions l'écriture avec l'expression ternaire

```
var nbr = prompt('Saisir un nombre');
nbr % 2 == 0 ? alert("pair") : alert('impair');
```

JavaScript

Considérons l'exemple suivant

```
var nbr = prompt('Saisir un nombre');
if (nbr % 2 == 0) {
  alert ("pair);
}
else {
  alert ("imapir");
}
```

Simplifions l'écriture avec l'expression ternaire

```
var nbr = prompt('Saisir un nombre');
nbr % 2 == 0 ? alert("pair") : alert('impair');
```

Ou

```
alert(nbr % 2 == 0 ? "pair" : 'impair');
```

JavaScript

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```

JavaScript

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

JavaScript

Exemple

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

JavaScript

Exemple

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

Le résultat est

```
0
1
2
3
4
```

JavaScript

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```

JavaScript

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

JavaScript

Exemple

```
var i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```

JavaScript

Exemple

```
var i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```

Le résultat est

```
0
1
2
3
4
```

JavaScript

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```

JavaScript

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```

Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

JavaScript

Exemple

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

JavaScript

Exemple

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

Le résultat est

```
0
1
2
3
4
```

JavaScript

Exercice

Écrire un code JS qui permet d'afficher les nombres pairs compris entre 0 et 10.

JavaScript

Exercice

Écrire un code JS qui permet d'afficher les nombres pairs compris entre 0 et 10.

Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    console.log(i);
  }
}
```

JavaScript

Exercice

Écrire un code JS qui permet d'afficher les nombres pairs compris entre 0 et 10.

Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    console.log(i);
  }
}
```

Deuxième solution

```
for (var i = 0; i < 10; i += 2) {
  console.log(i);
}
```

JavaScript

Tableau (array)

- une variable pouvant avoir simultanément plusieurs valeurs
- l'accès à chaque valeur s'effectue via son indice et avec l'opérateur `[]` : `[ indice ]`
- la première valeur (on dit aussi élément) du tableau est d'indice 0.
- les valeurs peuvent avoir plusieurs types différents

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

Le raccourci

```
var tab = [value1, value2,... valueN];
```

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

Le raccourci

```
var tab = [value1, value2,... valueN];
```

Accès à l'élément du tableau d'indice `i`

```
tab[i]
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

Pour afficher le premier élément du tableau

```
tab[0]
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

Pour afficher le premier élément du tableau

```
tab[0]
```

Pour afficher le dernier élément du tableau

```
tab[tab.length - 1]
```

JavaScript

Pour ajouter une nouvelle valeur (par exemple 7) au tableau

```
tab[tab.length] = 7;
```

JavaScript

Pour ajouter une nouvelle valeur (par exemple 7) au tableau

```
tab[tab.length] = 7;
```

Ou tout simplement

```
tab.push(7);
```

JavaScript

Pour afficher un tableau

```
console.log(tab);
```

JavaScript

Pour afficher un tableau

```
console.log(tab);
```

Pour parcourir et afficher un tableau

```
for(var i = 0; i < tab.length; i++) {
  console.log(tab[i]);
}
```

JavaScript

Pour afficher un tableau

```
console.log(tab);
```

Pour parcourir et afficher un tableau

```
for(var i = 0; i < tab.length; i++) {
  console.log(tab[i]);
}
```

On peut aussi utiliser la version simplifiée de `for`

```
for(var elt of tab) {
  console.log(elt);
}
```

JavaScript

Opérations sur les tableaux

- `push()` : ajoute un élément passé en paramètre au tableau
- `pop()` : supprime le dernier élément du tableau
- `shift()` : supprime le premier élément du tableau
- `indexOf()` : retourne la position de l'élément, passé en paramètre, dans le tableau, `-1` sinon.
- `reverse()` : inverse l'ordre des éléments du tableau
- `sort()` : trie un tableau
- `splice()` : permet d'extraire, ajouter ou supprimer un ou plusieurs éléments (selon les paramètres, voir slide suivante)
- `includes()` : retourne `true` si la valeur passée en paramètre est dans un tableau, `false` sinon.
- `...`

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];
tab = sports.splice(2, 0, "rugby", "natation");

for(var elt of sports)
  console.log(elt);
// affiche foot, tennis, rugby, natation, basket,
   volley

console.log(tab);
// n'affiche rien
```

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];
tab = sports.splice(2, 0, "rugby", "natation");

for(var elt of sports)
  console.log(elt);
// affiche foot, tennis, rugby, natation, basket,
   volley

console.log(tab);
// n'affiche rien
```

Aucun élément supprimé car le deuxième paramètre = 0

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];
tab = sports.splice(2, 1, "rugby", "natation");

for(var elt of sports)
  console.log(elt);
  // affiche foot, tennis, rugby, natation, volley

console.log(tab);
// affiche basket
```

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];
tab = sports.splice(2, 1, "rugby", "natation");

for(var elt of sports)
  console.log(elt);
  // affiche foot, tennis, rugby, natation, volley

console.log(tab);
// affiche basket
```

Un seul élément supprimé car le deuxième paramètre = 1

JavaScript

À ne pas confondre avec `slice` qui permet d'extraire un sous-tableau sans modifier le tableau d'origine

```
var sports = ["foot", "tennis", "basket", "volley"];
tab = sports.slice(1, 3);

for(var elt of sports)
  console.log(elt);
  // affiche foot, tennis, basket, volley

console.log(tab);
// affiche tennis, basket
```

JavaScript

Il est possible de préciser la taille d'un tableau et d'initialiser ses valeurs avec la méthode `fill`

```
var tab = new Array(3).fill(0);
console.log(tab);
// affiche [ 0, 0, 0 ]
```

JavaScript

Il est possible de préciser la taille d'un tableau et d'initialiser ses valeurs avec la méthode `fill`

```
var tab = new Array(3).fill(0);
console.log(tab);
// affiche [ 0, 0, 0 ]
```

On peut aussi utiliser `fill` pour modifier les valeurs d'une partie du tableau

```
var tab = [0, 1, 2, 3, 4, 5, 6, 7, 8];
tab.fill(9, 2, 5);
console.log(tab);
// affiche [ 0, 1, 9, 9, 9, 5, 6, 7, 8 ]
```

JavaScript

Autres opérations sur les tableaux (ES5)

- `forEach()` : pour parcourir un tableau
- `map()` : pour appliquer une fonction sur les éléments d'un tableau
- `filter()` : pour filtrer les éléments d'un tableau selon un critère défini sous forme d'une fonction anonyme ou fléchée
- `reduce()` : pour réduire tous les éléments d'un tableau en un seul selon une règle définie dans une fonction anonyme ou fléchée
- `...`

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
// affiche 3 et 5
```

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
// affiche 3 et 5
```

On peut permuter map et filter selon le besoin

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
// affiche 3 et 5
```

On peut permuter map et filter selon le besoin
On peut aussi remplacer les fonctions fléchée par des fonctions anonymes

JavaScript

Exemple avec `reduce` : permet de réduire les éléments d'un tableau en une seule valeur

```
var tab =[2, 3, 5];
var somme = tab.map(x => x + 3)
    .filter(x => x > 5)
    .reduce(function(sum, elem){
        return sum + elem;
    });

console.log(somme);
// affiche 14
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme(a, b) {
  return a + b;
}
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme(a, b) {
  return a + b;
}
```

Appeler une fonction

```
var resultat = somme (1, 3);
```

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
  return a + b;
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
  return a + b;
}
```

Appeler une fonction anonyme

```
var resultat2 = somme2 (1, 3);
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return a + b")
   ;
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return a + b")
   ;
```

Appeler une fonction anonyme

```
var resultat4 = somme4 (1, 3);
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

Le resultat

```
NaN
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

Le resultat

```
NaN
```

On peut affecter une valeur par défaut aux paramètres

```
function somme (a=0, b=0) {
  return a + b;
}
```

JavaScript

Et si on veut que la fonction somme retourne la somme quel que soit le nombre de paramètres

```
function somme () {
  result = 0;
  for(var i = 0; i < arguments.length ; i++) {
    result += arguments[i];
  }
  return result;
}
```

JavaScript

Et si on veut que la fonction somme retourne la somme quel que soit le nombre de paramètres

```
function somme () {
  result = 0;
  for(var i = 0; i < arguments.length ; i++) {
    result += arguments[i];
  }
  return result;
}
```

Tous ces appels sont corrects

```
console.log(somme(2, 3, 5));
console.log(somme(2, 3));
console.log(somme(2, 3, 5, 7));
console.log(somme(2));
```

JavaScript

Fonction de retour (callback)

- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée en **JavaScript** (**jQuery** et **nodeJS**) avec les fonctions asynchrones

JavaScript

Fonction de retour (callback)

- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée en **JavaScript** (**jQuery** et **nodeJS**) avec les fonctions asynchrones

Considérons les deux fonctions suivantes

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

JavaScript

Utilisons les fonctions précédentes comme callback d'une fonction `operation()`

```
function operation(a,b,fonction) {
  console.log (fonction(a, b));
}

// appeler la fonction opération
operation (3, 5, somme);
// affiche 8
```

JavaScript

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

JavaScript

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

JavaScript

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

Utilisons cette fonction comme callback dans `setTimeout`

```
function direBonjourApresXSecondes(x) {
  setTimeout(direBonjour, x * 1000);
}
```

JavaScript

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

Utilisons cette fonction comme callback dans `setTimeout`

```
function direBonjourApresXSecondes(x) {
  setTimeout(direBonjour, x * 1000);
}
```

Appelons `direBonjourApresXSecondes` et bonjour sera affiché après

```
direBonjourApresXSecondes(5);
```

JavaScript

Il est aussi possible d'utiliser une fonction anonyme

```
function direBonjourApresXSecondes(x) {
  setTimeout(function() {
    alert("Bonjour");
  },
  x * 1000);
}
direBonjourApresXSecondes(5);
```

JavaScript

Il est aussi possible d'utiliser une fonction anonyme

```
function direBonjourApresXSecondes(x) {
  setTimeout(function() {
    alert("Bonjour");
  },
  x * 1000);
}
direBonjourApresXSecondes(5);
```

La fonction `setTimeout()` ne permet pas de répéter l'affichage toutes les `x` secondes.

JavaScript

Il est aussi d'annuler l'exécution de la fonction callback de `setTimeout` avec `clearTimeout` si cette dernière est appelée avant la fin de la durée de `setTimeout`

```
var timeout;
function direBonjourApresXSecondes(x) {
  timeout = setTimeout(function(){
    alert("Bonjour");
  },
  x * 1000);
}

direBonjourApresXSecondes(5);

clearTimeout(timeout);
// Bonjour ne sera jamais affiché
```

JavaScript

Pour afficher un message toutes les `X` secondes, on peut utiliser `setInterval` qui a la même signature que `setTimeout`

```
function direBonjourToutesXSecondes(x) {
  setInterval(direBonjour, x * 1000);
}

function direBonjour() {
  alert("Bonjour");
}

direBonjourToutesXSecondes(5);
```

JavaScript

Pour afficher un message toutes les `X` secondes, on peut utiliser `setInterval` qui a la même signature que `setTimeout`

```
function direBonjourToutesXSecondes(x) {
  setInterval(direBonjour, x * 1000);
}

function direBonjour() {
  alert("Bonjour");
}

direBonjourToutesXSecondes(5);
```

Comme `clearTimeout()`, il existe une fonction `clearInterval()`.

JavaScript

Remarque

Si la fonction callback nécessite de paramètres, alors il est possible de les préciser après la durée dans `setInterval` et `setTimeout`.

JavaScript

Fonction génératrice

- déclarée avec `function*`
- utilise le mot-clé `yield` pour générer plusieurs valeurs

JavaScript

Fonction génératrice

- déclarée avec `function*`
- utilise le mot-clé `yield` pour générer plusieurs valeurs

Exemple

```
function* generateur() {
  for (let i = 0; i < 3; i++) {
    yield i;
  }
}
```

JavaScript

Pour appeler la fonction

```
var f = generateur();
```

JavaScript

Pour appeler la fonction

```
var f = generateur();
```

Pour générer des valeurs

```
console.log(f.next().value);
// affiche 0

console.log(f.next().value);
// affiche 1

console.log(f.next().value);
// affiche 2

console.log(f.next().value);
// affiche undefined
```

JavaScript

Considérons les deux fonctions suivantes

```
var i = 0;
function f() {
  i++;
}
f();
console.log(i); // affiche 1
function g() {
  var j = 5;
}
g();
console.log(j); // génère une erreur
```

JavaScript

Considérons les deux fonctions suivantes

```
var i = 0;
function f() {
  i++;
}
f();
console.log(i); // affiche 1
function g() {
  var j = 5;
}
g();
console.log(j); // génère une erreur
```

Remarque

Une variable déclarée dans une fonction avec le mot-clé `var` n'a pas une portée globale. Donc, elle est locale.

JavaScript

La closure : déclarer une variable locale dans une fonction qui existe en globale

```
var i = 0;
function f() {
var i = 5;
i++;
console.log(i);
}
f(); // affiche 6
console.log(i); // affiche 0
```

JavaScript

La closure : déclarer une variable locale dans une fonction qui existe en globale

```
var i = 0;
function f() {
var i = 5;
i++;
console.log(i);
}
f(); // affiche 6
console.log(i); // affiche 0
```

Remarque

La variable `i` affichée à la dernière instruction est la variable déclarée à la première ligne.

JavaScript

Comment sauvegarder toutes ces donnees dans une seule variable ?

nom	wick
prénom	john
⋮	⋮

JavaScript

Comment sauvegarder toutes ces donnees dans une seule variable ?

nom	wick
prénom	john
⋮	⋮

Comment faire ?

- Les tableaux indexés ne le permettent pas
- Les tableaux associatifs ou les objets

JavaScript

Objet ?

un ensemble de

- attributs (variables, champs) : clé + valeur[s]
- méthodes : fonctions

JavaScript

Création d'un objet

```
var obj = {
  nom: "wick",
  prenom: "john"
};
```

JavaScript

Création d'un objet

```
var obj = {
  nom: "wick",
  prenom: "john"
};
```

Accès à un attribut de l'objet

```
console.log(obj.nom);
console.log(obj["prenom"]);
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj)
  console.log(elt);
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj)
  console.log(elt);
```

Il est itérable avec `for ... in`

```
for(var key in obj)
  console.log(key + " : " + obj[key]);
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj)
  console.log(elt);
```

Il est itérable avec `for ... in`

```
for(var key in obj)
  console.log(key + " : " + obj[key]);
```

Un objet peut aussi être créé ainsi

```
var obj = new Object();
obj.nom = "wick";
obj.prenom = "john";
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un $\Rightarrow$ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un $\Rightarrow$ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

Ajouter un nouvel attribut à un objet existant

```
obj2.age = 35;
```

JavaScript

Pour cloner un objet

```
function clone(obj){
  var obj2 = {};
  for (key in obj)
    obj2[key] = obj[key];
  return obj2;
}

var obj = {nom: "wick", prenom: "john"};
var obj2 = clone(obj);
obj2.nom = "abruzzi";

console.log(obj);
// affiche wick

console.log(obj2);
// affiche abruzzi
```

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);
console.log(obj);
console.log(obj2);
```

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);
console.log(obj);
console.log(obj2);
```

La méthode `Object.create()` permet de copier un objet.

JavaScript

Pour récupérer l'ensemble des valeurs d'un objet dans un tableau

```
var obj = {nom: "wick", prenom: "john"};
var values = Object.values(obj);

for (value of values)
  console.log(value);

// affiche wick ensuite john
```

JavaScript

Pour récupérer l'ensemble des clés d'un objet dans un tableau

```
var keys = Object.keys(obj);

for (key of keys)
  console.log(key + " : " + obj[key]);

// affiche nom : wick
// prenom : john
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

Déclarons une variable `name` qui reçoit sa valeur d'un attribut d'`obj`

```
var name = obj.nom;
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

Déclarons une variable `name` qui reçoit sa valeur d'un attribut d'`obj`

```
var name = obj.nom;
```

Modifions la valeur d'un n'affecte pas l'autre

```
name = "travolta";
console.log(obj.nom);
// affiche wick

obj.nom = "abruzzi";
console.log(name);
// affiche travolta
```

JavaScript

Pour transformer un objet en chaîne de caractère

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

JavaScript

Pour transformer un objet en chaîne de caractère

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

Pour transformer une chaîne de caractère en objet

```
var p = JSON.parse(str);
console.log(p.nom + " " + p.prenom);
// affiche wick john
```

JavaScript

Un objet avec attributs et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```

JavaScript

Un objet avec attributs et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```

Pour appeler la méthode `direBonjour`

```
obj.direBonjour();
```

JavaScript

Un objet avec attributs et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```

Pour appeler la méthode `direBonjour`

```
obj.direBonjour();
```

Comment afficher le nom dans la méthode ?

JavaScript

Ceci génère une erreur

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + nom);
  }
};
obj.direBonjour();
```

JavaScript

Ceci génère une erreur

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + nom);
  }
};
obj.direBonjour();
```

Il faut utiliser l'opérateur `this`

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + this.nom);
  }
};
obj.direBonjour();
```

JavaScript

Constructeurs ?

- Besoin d'un moule pour créer des objets (comme les classes en Java, C#, PHP...)
- Tous les objets JS sont de type `Object`
- Il est possible de créer ou de cloner un objet a‘ partir d'un autre en utilisant des méthodes d'`Object`
- Et si on veut créer un modèle d'objet (comme la classe), on peut utiliser les constructeurs

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
}
```

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
}
```

Explication

- Un constructeur d'objet permettant d'initialiser la valeur de deux attributs `nom` et `prenom`
- L'appel de ce constructeur avec l'opérateur `new` permet de créer plusieurs objets ayant tous les attributs `nom` et `prenom`
- C'est comme une classe dans un LOO à classes

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

Afficher les valeurs d'un objet

```
console.log(perso);
```

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

Afficher les valeurs d'un objet

```
console.log(perso);
```

Pour vérifier qu'il s'agit bien d'un objet

```
console.log(typeof perso);
```

JavaScript

Déclarer un constructeur contenant une méthode

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
  this.direBonjour = function (){
    console.log("bonjour " + this.nom);
  }
}
```

JavaScript

Déclarer un constructeur contenant une méthode

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
  this.direBonjour = function (){
    console.log("bonjour " + this.nom);
  }
}
```

Pour appeler cette méthode

```
var perso = new Personne("wick","john");
perso.direBonjour();
```

JavaScript

Une fois le constructeur déclaré, impossible de lui ajouter de nouvel attribut de cette manière

```
Personne.age = 0;
```

JavaScript

Une fois le constructeur déclaré, impossible de lui ajouter de nouvel attribut de cette manière

```
Personne.age = 0;
```

Pour cela, il faut utiliser le prototype

```
Personne.prototype.age = 0;
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

Appeler les méthodes ajoutées

```
perso.setAge(45);
console.log(perso.getAge());
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

Appeler les méthodes ajoutées

```
perso.setAge(45);
console.log(perso.getAge());
```

Remarque

Ne jamais modifier le prototype d'un objet prédéfini.

JavaScript

Objets prédéfinis

- `Math` : contenant de méthodes permettant de réaliser des opérations mathématiques sur les nombres
- `Date` : permet de manipuler des dates
- `...`

JavaScript

Exemple de méthodes de l'objet `Math`

```
Math.round(2.1); // retourne 2
Math.round(2.9); // retourne 3
Math.pow(2, 3); // retourne 8
Math.sqrt(4); // retourne 2
Math.abs(-2); // retourne 2
Math.min(0, 1, 4, 2, -4, -5); // retourne -5
Math.max(0, 1, 4, 2, -4, -5); // retourne 4
Math.random(); // retourne un nombre aléatoire
Math.floor(Math.random() * 10);
// retourne un entier compris entre 0 et 9
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date);
// affiche Fri Aug 30 1985 05:50:00 GMT+0200
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date);
// affiche Fri Aug 30 1985 05:50:00 GMT+0200
```

Créer une date à partir d'un nombre de millisecondes

```
var date = new Date(100000000000);
console.log(date);
// affiche Sat Mar 03 1973 10:46:40 GMT+0100
```

JavaScript

Exemple de méthodes pour les dates

- `getFullYear()` : retourne l'annee
- `getMonth()` : retourne le mois [0-11]
- `getDate()` : retourne le jour [1-31]
- `getHours()` : retourne l'heure [0-23]
- `getMinutes()` : retourne les minutes [0-59]
- `getSeconds()` : retourne les secondes [0-59]
- `getMilliseconds()` : retourne les millisecondes [0-999]
- `getTime()` : retourne le nombre de millisecondes depuis le 1 Janvier 1970 (Avec ES5, on peut utiliser `Date.now()`)

JavaScript

Exemple de méthodes pour les dates

- `getFullYear()` : retourne l'annee
- `getMonth()` : retourne le mois [0-11]
- `getDate()` : retourne le jour [1-31]
- `getHours()` : retourne l'heure [0-23]
- `getMinutes()` : retourne les minutes [0-59]
- `getSeconds()` : retourne les secondes [0-59]
- `getMilliseconds()` : retourne les millisecondes [0-999]
- `getTime()` : retourne le nombre de millisecondes depuis le 1 Janvier 1970 (Avec ES5, on peut utiliser `Date.now()`)

On peut aussi modifier les attributs de l'objet Date en utilisant les set.

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

Une constante

un élément qui ne peut changer de valeur

JavaScript

Une constante

un élément qui ne peut changer de valeur

Pour déclarer une constante

il faut utiliser le mot-clé `const`

JavaScript

Une constante

un élément qui ne peut changer de valeur

Pour déclarer une constante

il faut utiliser le mot-clé `const`

Déclaration d'une constante

```
const PI = 3.1415;
```

JavaScript

Une constante

un élément qui ne peut changer de valeur

Pour déclarer une constante

il faut utiliser le mot-clé `const`

Déclaration d'une constante

```
const PI = 3.1415;
```

L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)

```
PI = 5;
```

JavaScript

Considérons l'objet suivant déclaré avec le mot-clé `const`

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

JavaScript

Considérons l'objet suivant déclaré avec le mot-clé `const`

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)

```
obj = {};
```

JavaScript

Considérons l'objet suivant déclaré avec le mot-clé `const`

```
const obj = {
    nom: "wick",
    prenom: "john"
};
```

L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)

```
obj = {};
```

Cependant, l'instruction suivante est correcte et ne lève donc pas d'exception

```
obj.nom ="travolta";
obj['prenom'] = "denzel";
obj.age = 50;

console.log(obj);
// affiche {nom: "travolta", prenom: "denzel", age: 50}
```

JavaScript

Idem pour les tableaux

```
const tableau = [2, 3, 8];
```

JavaScript

Idem pour les tableaux

```
const tableau = [2, 3, 8];
```

L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)

```
tableau = [];
```

JavaScript

Idem pour les tableaux

```
const tableau = [2, 3, 8];
```

L'instruction suivante lève une exception (`Uncaught TypeError: Assignment to constant variable`)

```
tableau = [];
```

Cependant, l'instruction suivante est correcte et ne lève donc pas d'exception

```
tableau[2] = 1;
tableau[0] = 9;

console.log(tableau);
// affiche [9, 3, 1]
```

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 = "ababaabbbaaab";
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 : ababaabbbaaab
```

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"]
```

JavaScript

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

JavaScript

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
```

JavaScript

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

Promesses

- un objet JavaScript utilisé souvent pour réaliser des traitements sur un résultat suite à une opération asynchrone
- disposant d'une première méthode `then()` permettant de traiter le résultat une fois l'opération accomplie
- disposant d'une deuxième méthode `catch()` qui sera exécutée en cas d'échec de l'opération
- composé de deux parties : déclaration et utilisation

JavaScript

Considérons la déclaration suivante d'une promesse

```
var test = true;
var promesse = new Promise((resolve, reject) => {
  if(test)
    resolve();
  else
    reject();
});
```

JavaScript

Considérons la déclaration suivante d'une promesse

```
var test = true;
var promesse = new Promise((resolve, reject) => {
  if(test)
    resolve();
  else
    reject();
});
```

Dans la partie utilisation, on doit indiquer ce qu'il faut faire dans les deux cas : réussite (`resolve`) ou échec (`reject`)

```
promesse.then(() => console.log("test réussi") )
        .catch(() => console.log("erreur") );
// affiche test réussi car test = true
```

JavaScript

Considérons la déclaration suivante d'une promesse

```
var test = true;
var promesse = new Promise((resolve, reject) => {
  if(test)
    resolve();
  else
    reject();
});
```

Dans la partie utilisation, on doit indiquer ce qu'il faut faire dans les deux cas : réussite (`resolve`) ou échec (`reject`)

```
promesse.then(() => console.log("test réussi") )
        .catch(() => console.log("erreur") );
// affiche test réussi car test = true
```

Le `catch()` n'est pas obligatoire

JavaScript

La déclaration précédente peut être écrite ainsi

```
var test = true;
var promesse = () => {
  return new Promise((resolve, reject) => {
    if(test)
      resolve();
    else
      reject();
  });
};
```

JavaScript

La déclaration précédente peut être écrite ainsi

```
var test = true;
var promesse = () => {
  return new Promise((resolve, reject) => {
    if(test)
      resolve();
    else
      reject();
  });
};
```

Pour l'utilisation, il faut appeler promesse comme une fonction

```
promesse().then(() => console.log("test réussi") )
          .catch(() => console.log("erreur") );
```

JavaScript

Remarque

Une promesse peut recevoir des paramètres et retourner un résultat

JavaScript

Exemple : déclaration

```
var division = (a, b) => {
  return new Promise((resolve, reject) => {
    if(b != 0)
      resolve(a / b);
    else
      reject("erreur : division par zéro");
  });
};
```

JavaScript

Exemple : déclaration

```
var division = (a, b) => {
  return new Promise((resolve, reject) => {
    if(b != 0)
      resolve(a / b);
    else
      reject("erreur : division par zéro");
  });
};
```

L'utilisation

```
division(10, 2).then((res) => console.log("résultat : " + res))
               .catch((error) => console.log(error));
// affiche résultat : 5

division(5, 0).then((res) => console.log("résultat : " + res))
              .catch((error) => console.log(error));
// affiche erreur : division par zéro
```

Remarque : les promesses s'exécutent d'une manière asynchrone

```
var division = (a, b) => {
  return new Promise((resolve, reject) => {
    if(b != 0)
      resolve(a / b);
    else
      reject("erreur : division par zéro");
  });
};

division(10, 2).then((res) => console.log("résultat : " + res))
               .catch((error) => console.log(error));

division(5, 0).then((res) => console.log("résultat : " + res))
              .catch((error) => console.log(error));

console.log("fin");
```

Le résultat est :

```
fin
résultat : 5
erreur : division par zéro
```

JavaScript

Le mot-clé `async`

Il permet de transformer une fonction JavaScript en promesse

JavaScript

Le mot-clé `async`

Il permet de transformer une fonction JavaScript en promesse

Considérons la fonction `somme()` suivante

```
let somme = (a, b) => a + b;
```

JavaScript

Le mot-clé `async`

Il permet de transformer une fonction JavaScript en promesse

Considérons la fonction `somme()` suivante

```
let somme = (a, b) => a + b;
```

Le résultat est :

```
console.log(somme(2, 3));
// affiche 5
```

JavaScript

Pour transformer la fonction `somme()` en promesse, on ajoute le mot-clé `async` à sa déclaration

```
let somme = async (a, b) => a + b;
```

Maintenant, on peut l'utiliser comme une promesse

```
somme(2, 3).then(result => console.log(result));
// affiche 5
```

JavaScript

Pour transformer la fonction `somme()` en promesse, on ajoute le mot-clé `async` à sa déclaration

```
let somme = async (a, b) => a + b;
```

Maintenant, on peut l'utiliser comme une promesse

```
somme(2, 3).then(result => console.log(result));
// affiche 5
```

Appeler `somme()` comme une simple fonction JavaScript n'affiche pas le résultat

```
console.log(somme(2, 3));
// affiche qu'une promise a été résolue et que sa
   valeur de retour est 5
```

JavaScript

Le mot-clé `await`

- utilisable seulement dans des environnements asynchrones
- permettant d'interrompre l'exécution de la fonction asynchrone et attendre la résolution de la promesse

JavaScript

Le mot-clé `await`

- utilisable seulement dans des environnements asynchrones
- permettant d'interrompre l'exécution de la fonction asynchrone et attendre la résolution de la promesse

Considérons la promesse `somme()` qui attend 2 secondes pour retourner un résultat

```
let somme =  (a, b) => {
        return new Promise((resolve) => {
        setTimeout(() => { resolve(a + b) }, 2000);
  });
};
```

JavaScript

Le mot-clé `await`

- utilisable seulement dans des environnements asynchrones
- permettant d'interrompre l'exécution de la fonction asynchrone et attendre la résolution de la promesse

Considérons la promesse `somme()` qui attend 2 secondes pour retourner un résultat

```
let somme =  (a, b) => {
        return new Promise((resolve) => {
        setTimeout(() => { resolve(a + b) }, 2000);
  });
};
```

On veut implémenter une promesse `sommeCarre()` qui utilise la promesse `somme()`

```
let sommeCarre = async (a, b) => {
  let s = somme(a, b).then(result => result);
  let result = Math.pow(s, 2);
  return result;
};
```

JavaScript

Pour utiliser la promesse avec les valeurs `2` et `3`

```
sommeCarre(2, 3).then(result => console.log(result));
// affiche NaN car on n'a pas obtenu le résultat de somme
   lorsqu'on a calculé le carré
```

Solution, utiliser `await` pour attendre la fin de la première promesse

```
let sommeCarre = async (a, b) => {
  let s = await somme(a, b).then(result => result);
  let result = Math.pow(s, 2);
  return result;
};
```

JavaScript

Pour utiliser la promesse avec les valeurs `2` et `3`

```
sommeCarre(2, 3).then(result => console.log(result));
// affiche NaN car on n'a pas obtenu le résultat de somme
   lorsqu'on a calculé le carré
```

Solution, utiliser `await` pour attendre la fin de la première promesse

```
let sommeCarre = async (a, b) => {
  let s = await somme(a, b).then(result => result);
  let result = Math.pow(s, 2);
  return result;
};
```

Si on teste maintenant

```
sommeCarre(2, 3).then(result => console.log(result));
// affiche 25
```

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

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"
```