Fondamentaux Javascript

Maxence Cuingnet

DevOps 2021

Une introduction à JavaScript

JavaScript et Java sont des langages complètement différents, à la fois dans le concept et dans la conception.

JavaScript a été inventé par Brendan Eich en 1995 et est devenu une norme ECMA en 1997.

ECMA-262 est le nom officiel de la norme. ECMAScript est le nom officiel de la langue.

JavaScript est un langage interprété. La définition de la langue est plus large que le mot interprété. Vous pouvez le définir comme un langage procédural faiblement typé, basé sur des prototypes, dynamique et impératif.
JavaScript est développé et implémenté sous la forme d'un langage de programmation côté client. Il fait partie du navigateur Web et est considéré comme quelque chose qui peut aider les développeurs à mettre en œuvre des fonctionnalités dynamiques et à créer et intégrer une interface utilisateur sur les pages Web. Vous pouvez également trouver une implémentation de JavaScript côté serveur.

Cependant, la popularité du langage de programmation JavaScript repose en grande partie sur ses applications côté client. Vous pouvez également voir des applications JavaScript en dehors des applications Web. Par exemple, vous pouvez les trouver utilisés pour ajouter une certaine interactivité aux widgets de bureau et aux documents au format PDF

JavaScript a été initialement conçu dans le même moule que le langage C, mais il prend également des noms de langage de programmation Java. Vous ne devez pas confondre Java avec JavaScript car les deux sont intrinsèquement différents. Ils ont également des objectifs et une sémantique différents.

Les cas d'utilisation de JavaScript

JavaScript est à l'œuvre du côté client de la plupart des pages Web du monde entier. Lorsque vous ouvrez une page Web et qu'elle a changé par rapport au moment où vous l'avez ouverte précédemment, ces modifications auraient pu être apportées à l'aide de JavaScript.

Ouvrez une page Web sur l'écran de votre ordinateur. Cliquez avec le bouton droit sur la page Web, puis cliquez sur Afficher la source. Vous verrez le mot JavaScript dans le code de la page.

Cependant, JavaScript vous permet de modifier le texte, d'ajouter des messages contextuels et de valider le texte que vous avez saisi via le codage HTML. HTML et JavaScript marchent côte à côte. Là où JavaScript devance le HTML, c'est la façon dont il rend la page dynamique et interactive. Cela permettra à vos utilisateurs de cliquer sur certains éléments et d'ouvrir de nouvelles pages à partir de la page existante.

Avec l'aide de JavaScript, vous pouvez créer des interfaces réactives qui affinent l'expérience utilisateur et offrent des fonctionnalités extrêmement dynamiques.

Les cas d'utilisation de JavaScript évoluent avec le changement des temps. Les utilisateurs doivent changer l'interface Web éprouvée, testée et ennuyeuse consistant à cliquer sur des liens, à remplir des informations sur une page Web et à envoyer un tas de formulaires.

Ils cherchent à rendre les mêmes processus plus modernes et plus conviviaux. JavaScript a permis aux utilisateurs de créer un formulaire d'inscription qui a la capacité de vérifier si un nom d'utilisateur est déjà pris ou est disponible. Cela empêche les utilisateurs de recharger leurs pages, ce qui peut être très frustrant. Un champ de recherche vous montre les résultats les plus pertinents et suggérés lorsque vous tapez quelque chose. Il montre quelque chose qui est basé sur vos entrées. Le modèle est connu sous le nom de saisie semi-automatique - sans JavaScript, cela n'aurait pas été possible.

Les informations qui doivent changer après des heures définies sans aucune entrée de l'utilisateur sont possibles à l'aide de JavaScript. JavaScript peut tenter de résoudre les problèmes de mise en page. En utilisant JavaScript, vous pouvez trouver la zone et la position exactes de n'importe quel élément de votre page Web. Vous pouvez calculer les dimensions de la fenêtre de votre navigateur. En collectant et en utilisant ces informations, vous pouvez aider à éviter le chevauchement des éléments de menu et des barres de défilement en vérifiant l'espace pour le sous-menu. La meilleure

façon dont JavaScript peut vous aider est peut-être d'améliorer l'interface fournie par HTML.

L'avènement de l'ES6 a encore facilité l'utilisation de la métaprogrammation dans le monde de la programmation JavaScript. Vous pouvez utiliser des proxies et d'autres fonctionnalités similaires. Les proxys d'ES6 ont tendance à faciliter la redéfinition des différentes opérations dans les objets.

Premier script :

```
<!DOCTYPE html>
<html>
<body>
<p>JavaScript rests in the Body</p>
<p id="demo12"></p>
<script>
document.getElementById("demo12").innerHTML = "This is your First
JavaScript";
</script>
</body>
</html>
```

Il n'est plus nécessaire d'écrire **type="text/javascript"** car le langage de script par défaut dans HTML est dorénavant JS

JS6 - Les Fondamentaux

JavaScript peut changer le contenu HTML

Une des nombreuses méthodes HTML JavaScript est `getElementById ()`.

L'exemple ci-dessous "trouve" un élément HTML (avec id = "demo") et change le contenu de l'élément (innerHTML) en "Hello JavaScript":

```
<h2>Que peut faire JS?</h2>

<p id="demo">JavaScript peut changer le contenu d'une page HTML</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>
```

Il est possible de mettre les guillemets ou les apostrophes.

JavaScript peut modifier les valeurs d'attribut HTML

Dans cet exemple, JavaScript modifie la valeur de l'attribut **src** (source) d'une balise **<img>**:

```
<button onclick="document.getElementById('myImage').src='MonImage2'">Turn on the light</button>

<img id="myImage" src="MonImage1" style="width:100px">

<button onclick="document.getElementById('myImage').src='MonImage1'">Turn off the light</button>
```

JavaScript peut changer les styles HTML (CSS)

```
<p id="demo">JavaScript peut changer le style d’éléments</p>

<button type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">Cl
ique ici!</button>
```

JavaScript peut masquer les éléments HTML

Le masquage des éléments HTML peut être effectué en modifiant le style d'affichage:

```
<p id="demo">JavaScript peut cacher des éléments.</p>

<button type="button"
onclick="document.getElementById('demo').style.display='none'">Cli
que ici!</button>
```

et donc inversement :

JavaScript peut afficher des éléments HTML

L'affichage des éléments HTML masqués peut également être effectué en modifiant le style d'affichage:

```
<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button"
onclick="document.getElementById('demo').style.display='block'">Cl
ique ici!</button>
```

Où implémenter le code JS ?

La balise <script>

En HTML, le code JavaScript est inséré entre les balises <script> et </script>.

JavaScript dans <head> ou <body>

Vous pouvez placer n'importe quel nombre de scripts dans un document HTML.

Les scripts peuvent être placés dans le <body>, ou dans la section <head> d'une page HTML, ou dans les deux.

JavaScript dans <head>

Dans cet exemple, une fonction JavaScript est placée dans la section <head> d'une page HTML.

Une fonction est invoquée lorsqu'un bouton est cliqué. Nous verrons plus tard les fonctions. Elles peuvent être appelées lors d'un événement, lorsqu'un utilisateur clique par exemple sur un bouton.

```
<!DOCTYPE html>
<html>
<head>
    <script>
    function myFunction() {
     document.getElementById("demo").innerHTML = "Le paragraphe a
changé";
}
</script>
</head>
<body>
<h1>Page Web</h1>
<p id="demo">Mon Paragraphe</p>
<button type="button" onclick="myFunction()">Essayez le</button>
</body>
</html>
```

JavaScript dans <body>

Dans cet exemple, une fonction JavaScript est placée dans la section <body> d'une page HTML.

La fonction est invoquée (appelée) lorsqu'un bouton est cliqué:

```
<!DOCTYPE html>
<html>
<body>
<h1>Page Web</h1>
<p id="demo">Mon Paragraphe</p>
<button type="button" onclick="myFunction()">Essayez</button>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Le paragraphe a
changé";
}
</script>
</body>
</html>
```

Placer des scripts en bas de l'élément <body> améliore la vitesse d'affichage, car l'interprétation des scripts ralentit l'affichage de la page HTML.

External JavaScript

Les scripts peuvent aussi être placés dans des fichiers externes:

Faites un fichier script.js :

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraphe
changé.";
}
```

Les scripts externes sont pratiques lorsque le même code est utilisé dans de nombreuses pages Web différentes.

Les fichiers JavaScript ont l'extension de fichier .js.

Pour utiliser un script externe, placez le nom du fichier de script dans l'attribut src (source) d'une balise <script>:

```
<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called
"script.js")</p>

<script src="script.js"></script>
```

Vous pouvez placer une référence de script externe dans <head> ou <body> comme vous le souhaitez.

Le script se comportera comme s'il se trouvait exactement à l'emplacement de la balise <script>.

Les scripts externes ne peuvent pas contenir de balises <script>.

Avantages Scripts externes

Placer des scripts dans des fichiers externes présente certains avantages:

Il sépare HTML et code
Il facilite la lecture et la maintenance du HTML et du JavaScript
Les fichiers JavaScript mis en cache peuvent accélérer le chargement des pages
Pour ajouter plusieurs fichiers de script à une page, utilisez plusieurs balises de script:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Références Externes

Les scripts externes peuvent être référencés avec une URL complète ou avec un chemin relatif à la page Web actuelle.

Cet exemple utilise une URL complète pour créer un lien vers un script:

```
<script src="https://www.uneadresse.com/script.js"></script>
```

On peut bien sûr mettre une url locale

Possibilités d'affichage JavaScript

JavaScript peut "afficher" les données de différentes manières:

- ➔ Écriture dans un élément HTML, en utilisant innerHTML.
- ➔ Écriture dans la sortie HTML à l'aide de document.write().
- ➔ Écrire dans une boîte d'alerte, en utilisant window.alert().
- ➔ Écriture dans la console du navigateur, en utilisant console.log().

Utilisation de innerHTML

Pour accéder à un élément HTML, JavaScript peut utiliser la méthode **document.getElementById(id)**.

L'attribut id définit l'élément HTML. La propriété innerHTML définit le contenu HTML.

La modification de la propriété innerHTML d'un élément HTML est un moyen courant d'afficher des données en HTML.

Utilisation de document.write()

À des fins de test, il est pratique d'utiliser document.write():

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>
```

Attention : l'utilisation de document.write() après le chargement d'un document HTML supprimera tout le HTML existant:

```
<h1> Ma première page Web </h1>
<p> Mon premier paragraphe. </p>

<button type = "button" onclick = "document.write(5 + 6)"> Essayer
</button>
```

La méthode document.write() ne doit être utilisée que pour les tests.

Window.alert()

Vous pouvez utiliser une boîte d'alerte pour afficher des données:

```
<script>
window.alert(5 + 6);
</script>
```

Vous pouvez ignorer le mot-clé window.

En JavaScript, l'objet window est l'objet de portée globale, ce qui signifie que les variables, propriétés et méthodes appartiennent par défaut à l'objet window. Cela signifie également que la spécification du mot-clé window est facultative:

```
<script>
alert(5 + 6);
</script>
```

Console.log()

À des fins de débogage, vous pouvez appeler la méthode console.log() dans le navigateur pour afficher les données.

```
<script>
console.log(5 + 6); console.log("Bonjour");
</script>
```

Impression JavaScript

JavaScript n'a pas d'objet d'impression ni de méthode d'impression.

Vous ne pouvez pas accéder aux périphériques de sortie à partir de JavaScript.

La seule exception est que vous pouvez appeler la méthode window.print() dans le navigateur pour imprimer le contenu de la fenêtre courante.

```
<button onclick="window.print()">Print this page</button>
```

Déclarations JavaScript

```
<p id="demo"></p>

<script>
var x, y, z;  // Déclaration 1
x = 5;        // Déclaration 2
y = 6;        // Déclaration 3
z = x + y;    // Déclaration 4

document.getElementById("demo").innerHTML =
"La valeur de z est " + z + ".";
</script>
```

En HTML, les programmes JavaScript sont exécutés par le navigateur Web.

Les instructions JavaScript sont composées de:

Valeurs, opérateurs, expressions, mots clés et commentaires.

Cette instruction indique au navigateur d'écrire «Salut tout le monde». dans un élément HTML avec id = "demo":

```
 document.getElementById("demo").innerHTML = "Salut tout le
monde";
```

La plupart des programmes JavaScript contiennent de nombreuses instructions JavaScript.

Les instructions sont exécutées, une par une, dans le même ordre où elles sont écrites.

Les programmes JavaScript (et les instructions JavaScript) sont souvent appelés code JavaScript.

Points-virgules :

Les points-virgules séparent les instructions JavaScript.

Ajoutez un point-virgule à la fin de chaque instruction exécutable

Lorsqu'elles sont séparées par des points-virgules, plusieurs instructions sur une ligne sont autorisées:

```
a = 5; b = 6; c = a + b;
```

Sur le Web, vous pouvez voir des exemples sans point-virgule.
La fin des instructions par un point-virgule n'est pas obligatoire, mais fortement recommandée.

Espace

JavaScript ignore plusieurs espaces. Vous pouvez ajouter un espace blanc à votre script pour le rendre plus lisible.

Les lignes suivantes sont équivalentes:

```
var person = "Jojo";
var person="Jojo";
```

Une bonne pratique consiste à placer des espaces autour des opérateurs (= + - * /):

```
var x = y + z;
```

Longueur de ligne et sauts de ligne JavaScript

Pour une meilleure lisibilité, les programmeurs aiment souvent éviter les lignes de code de plus de 80 caractères.

Si une instruction JavaScript ne tient pas sur une ligne, le meilleur endroit pour la casser est après un opérateur:

```
document.getElementById("demo").innerHTML =
"Hello Jacques";
```

Blocs de code JavaScript

Les instructions JavaScript peuvent être regroupées en blocs de code, entre accolades {...}.

Le but des blocs de code est de définir des instructions à exécuter ensemble.

Un endroit où vous trouverez des instructions regroupées en blocs, est dans les fonctions JavaScript:

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Bonjour!";
  document.getElementById("demo2").innerHTML = "Comment ça va?";
}
```

Mots-clés JavaScript

Les instructions JavaScript commencent souvent par un mot-clé pour identifier l'action JavaScript à effectuer.

Voici une liste de certains des mots-clés que vous découvrirez dans ce didacticiel:

Description de quelques mots-clés

break	Termine un interrupteur ou une boucle
continue	Saute hors d'une boucle et commence par le haut
debugger	Arrête l'exécution de JavaScript et appelle (si disponible) la fonction de débogage
do ... while	Exécute un bloc d'instructions et répète le bloc, tant qu'une condition est vraie
for	Marque un bloc d'instructions à exécuter, tant qu'une condition est vraie
function	Déclare une fonction
if ... else	Marque un bloc d'instructions à exécuter, en fonction d'une condition
return	Quitte une fonction
switch	Marque un bloc d'instructions à exécuter, selon les cas
try ... catch	Implémente la gestion des erreurs dans un bloc d'instructions
var	Déclare une variable

Les mots clés JavaScript sont des mots réservés. Les mots réservés ne peuvent pas être utilisés comme noms de variables.

Voici un lien avec les mots réservés : https://developer.mozilla.org/fr/docs/conflicting/Web/JavaScript/Reference/Lexical_grammar

Syntaxe Javascript

La syntaxe JavaScript est l'ensemble des règles, comment les programmes JavaScript sont construits:

```
var x, y, z;     // Déclarer les Variables
x = 5; y = 6;    // Assigner leurs valeurs
z = x + y;       // Les utiliser
```

La syntaxe JavaScript définit deux types de valeurs:

- **Valeurs fixes**
- **Valeurs variables**

Les valeurs fixes sont appelées littéraux.

Les valeurs de variable sont appelées des variables. C'est une bonne pratique de programmation de déclarer toutes les variables au début d'un script.

Littéraux JavaScript

Les deux règles de syntaxe les plus importantes pour les valeurs fixes sont:

1. Les nombres sont écrits avec ou sans décimales: 2165 ou 56.23
2. Les chaînes de caractère sont écrites entre guillemets doubles ou simples

“Bonjour tout le monde” == ‘Bonjour tout le monde’

Commentaires JavaScript

Toutes les instructions JavaScript ne sont pas exécutées.

Le code après des doubles barres obliques // ou entre / * et * / est traité comme un commentaire.

Les commentaires sont ignorés et ne seront pas exécutés:

```
var x = 5;   // Je serai exécuté

// var x = 6;   je ne serai pas exécuté
```

Identificateurs JavaScript

Les identificateurs sont des noms.

En JavaScript, les identificateurs sont utilisés pour nommer des variables (et des mots clés, des fonctions et des étiquettes).

Les règles relatives aux noms légaux sont sensiblement les mêmes dans la plupart des langages de programmation.

En JavaScript, le premier caractère doit être une lettre, un trait de soulignement (_) ou un signe dollar ($).

Les caractères suivants peuvent être des lettres, des chiffres, des traits de soulignement ou des signes dollar.

Les nombres ne sont pas autorisés comme premier caractère.
De cette façon, JavaScript peut facilement distinguer les identifiants des nombres.

JavaScript est sensible à la casse

Tous les identifiants JavaScript sont sensibles à la casse.

Les variables lastName et lastname, sont deux variables différentes:

```
var lastname, lastName;
lastName = "André";
lastname = "De la Ville";
```

Camel Case

Historiquement, les programmeurs ont utilisé différentes façons de joindre plusieurs mots en un seul nom de variable:

Trait d'union:

prénom, nom, master-card, inter-ville.

Les traits d'union ne sont pas autorisés dans JavaScript. Ils sont réservés aux soustractions.

Underscore:

prénom, nom, master_card, inter_city.

UpperCase (Pascal Case):

Prénom, Nom, MasterCard, InterCity.

Lower Camel Case:

Les programmeurs JavaScript ont tendance à utiliser la casse camel qui commence par une lettre minuscule:

firstName, lastName, masterCard, interCity.

Utilisation de let et const (ES6)

Avant 2015, l'utilisation du mot-clé var était le seul moyen de déclarer une variable JavaScript.

La version 2015 de JavaScript (ES6) permet d'utiliser le mot-clé **const** pour définir une variable qui ne peut pas être réaffectée, et le mot-clé **let** pour définir une variable à portée restreinte.

Comme il est un peu compliqué de décrire la différence entre ces mots-clés, et parce qu'ils ne sont pas pris en charge dans les anciens navigateurs, la première partie de ce didacticiel utilisera le plus souvent var.

Une déclaration, plusieurs variables

Vous pouvez déclarer plusieurs variables dans une seule instruction.

Démarrez l'instruction avec var et séparez les variables par une virgule:

```
<p id="demo"></p>

<script>
var person = "Henri", carName = "Saab", price = 200000000;
document.getElementById("demo").innerHTML = carName;
</script>
```

ou en plusieurs lignes suivies d’un point-virgule.

Valeur = indéfinie

Dans les programmes informatiques, les variables sont souvent déclarées sans valeur. La valeur peut être quelque chose qui doit être calculé, ou quelque chose qui sera fourni plus tard, comme une entrée utilisateur.

Une variable déclarée sans valeur aura une valeur indéfinie.

La variable carName aura la valeur indéfinie après l'exécution de cette instruction:

```
var carName;
```

Re-déclarer des variables JavaScript

Si vous re-déclarez une variable JavaScript, elle ne perdra pas sa valeur.

La variable **carName** aura toujours la valeur "BMW" après l'exécution de ces instructions:

```
<p id="demo"></p>

<script>
var carName = "Volvo";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>
```

Signe dollar $

N'oubliez pas que les identifiants JavaScript (noms) doivent commencer par:

- Une lettre (A-Z ou a-z)
- Un signe dollar ($)
- Ou un underscore (_)

Puisque JavaScript traite un signe dollar comme une lettre, les identificateurs contenant $ sont des noms de variables valides:

```
<p id="demo"></p>

<script>
var $ = 2;
var $myMoney = 5;
document.getElementById("demo").innerHTML = $ + $myMoney;
</script>
```

L'utilisation du signe dollar n'est pas très courante en JavaScript, mais les programmeurs professionnels l'utilisent souvent comme alias pour la fonction principale d'une bibliothèque JavaScript.

Dans la bibliothèque JavaScript jQuery, par exemple, la fonction principale $ est utilisée pour sélectionner des éléments HTML.
Dans jQuery $("p"); signifie "sélectionner tous les p éléments".

Underscore (_)

Puisque JavaScript traite le trait de soulignement comme une lettre, les identificateurs contenant _ sont des noms de variables valides:

```
var _lastName = "bouteille";
var _x = 2;
var _100 = 5;
```

L'utilisation de l'underscore n'est pas très courante en JavaScript, mais plutôt une convention parmi les programmeurs, c' est de l'utiliser comme alias pour les variables «privées (cachées)».

ECMAScript 2015

ES2015 a introduit deux nouveaux mots clés JavaScript importants: let et const.

Ces deux mots clés fournissent des variables (et des constantes) Block Scope en JavaScript.

Avant ES2015, JavaScript n'avait que deux types de portée: la portée globale et la portée de la fonction: **Global Scope** and **Function Scope**

Global Scope

Les variables déclarées globalement (en dehors de toute fonction) ont une portée globale.

```
<p id="demo"></p>

<script>
var carName = "Porsche";
myFunction();

function myFunction() {
  document.getElementById("demo").innerHTML = "J’affiche " +
carName;
}
</script>
```

Les variables globales sont accessibles de n'importe où dans un programme JavaScript.

Function Scope

Les variables déclarées localement (à l'intérieur d'une fonction) ont une portée uniquement dans la fonction:

```
<p id="demo1"></p>

<p id="demo2"></p>

<script>
myFunction();

function myFunction() {
  var carName = "Porsche";
  document.getElementById("demo1").innerHTML = typeof carName +
"  " + carName;
}
document.getElementById("demo2").innerHTML = typeof carName;
</script>
```

Les variables locales ne sont accessibles que depuis l'intérieur de la fonction où elles sont déclarées.

Block Scope

Les variables déclarées avec le mot-clé var ne peuvent pas avoir d'étendue de bloc : Block Scope
Les variables déclarées à l'intérieur d'un bloc {} sont accessibles depuis l'extérieur du bloc.

```
{
  var x = 2;
}
// x PEUT être utilisé ici
```

Avant ES2015, JavaScript n'avait pas de Block Scope

Les variables déclarées avec le mot-clé **let** peuvent avoir une portée de bloc.

Les variables déclarées à l'intérieur d'un bloc {} ne sont pas accessibles depuis l'extérieur du bloc:

```
{
  let x = 2;
}
// x NE PEUT PAS être utilisé ici
```

Conséquence :

Redéclarer une variable à l'aide du mot-clé var peut poser des problèmes.

Redéclarer une variable à l'intérieur d'un bloc redéclarera également la variable à l'extérieur du bloc:

```
<p id="demo"></p>

<script>
var  x = 10;
// Ici X vaut 10
{
  var x = 2;
  // Ici X vaut 2
}
// Ici X vaut 2
document.getElementById("demo").innerHTML = x;
</script>
```

Ainsi redéclarer une variable à l'aide du mot clé **let** peut résoudre ce problème.

Redéclarer une variable à l'intérieur d'un bloc ne redéclarera pas la variable à l'extérieur du bloc:

```
<p id="demo"></p>

<script>
var  x = 10;
// Ici X vaut 10
{
  let x = 2;
  // Ici X vaut 2
}
// Ici X vaut 10
document.getElementById("demo").innerHTML = x;
</script>
```

Loop Scope

Utilisation de **var** dans une boucle :

```
<p id="demo"></p>

<script>
var i = 5;
for (var i = 0; i < 10; i++) {
  // some statements
}
document.getElementById("demo").innerHTML = i;
</script>
```

Utilisation de **let** dans une boucle :

```
<p id="demo"></p>

<script>
let i = 5;
for (let i = 0; i < 10; i++) {
  // some statements
}

document.getElementById("demo").innerHTML = i;
</script>
```

Dans le premier exemple, en utilisant var, la variable déclarée dans la boucle redéclare la variable en dehors de la boucle.

Dans le deuxième exemple, en utilisant let, la variable déclarée dans la boucle ne redéclarera pas la variable en dehors de la boucle.

Lorsque let est utilisé pour déclarer la variable i dans une boucle, la variable i ne sera visible que dans la boucle.

Function et Global Scope

Les variables déclarées avec var et let sont assez similaires lorsqu'elles sont déclarées dans une fonction.

Elles auront toutes les deux une portée de fonction.

Les variables déclarées avec var et let sont assez similaires lorsqu'elles sont déclarées en dehors d'un bloc.

Elles auront toutes deux une portée globale.

Attention : redéclarer une variable var avec let, dans la même portée, ou dans le même bloc, n'est pas autorisé.

```
var x = 2;       // Autorisé
let x = 3;        // Pas autorisé

{
  var x = 4;    // Autorisé
  let x = 5    // Pas autorisé
}
```

Tout comme redéfinir une variable let dans un même scope n’est pas non plus autorisé :

```
let x = 2;       // OUI
let x = 3;        // NON

{
  let x = 4;    // OUI
  let x = 5;    // NON
}
```

Mais la redéclarer dans un autre scope est autorisé :

```
let x = 2;       // OUI

{
  let x = 3;    // OUI
}

{
  let x = 4;    // OUI
}
```

ES2015 a introduit deux nouveaux mots clés JavaScript importants: let et const.

Les variables définies avec const se comportent comme des variables let, sauf qu'elles ne peuvent pas être réaffectées:

```
<p id="demo"></p>

<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>
```

Portée du bloc

Déclarer une variable avec const est similaire à let quand il s'agit de Block Scope.

Le x déclaré dans le bloc, dans cet exemple, n'est pas le même que le x déclaré à l'extérieur du bloc:

```
var x = 10;
// Ici x vaut 10
{
  const x = 2;
  // Ici x vaut 2
}
// Ici x vaut 10
```

Attribué une fois déclaré

Les variables JavaScript const doivent recevoir directement une valeur lorsqu'elles sont déclarées:

Incorrect :

```
const PI;
PI = 3.14159265359;
```

Correct :

```
const PI = 3.14159265359;
```

Pas de vraies constantes

Le mot-clé const est un peu trompeur.

Il ne définit PAS une valeur constante. Il définit une référence constante à une valeur.

Pour cette raison, nous ne pouvons pas modifier les valeurs primitives constantes, mais nous pouvons modifier les propriétés des objets constants.

Valeurs primitives

Si nous attribuons une valeur primitive à une constante, nous ne pouvons pas modifier la valeur primitive:

```
<p id="demo"></p>

<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>
```

Les objets constants peuvent changer

Vous pouvez modifier les propriétés d'un objet constant:

```
<p>La déclaration d'un objet constant ne rend PAS les propriétés
des objets immuables:</p>

<p id="demo"></p>

<script>
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};

// Changer une propriété:
car.color = "red";

// Ajouter une propriété:
car.owner = "Johnson";

// Afficher une propriété:
document.getElementById("demo").innerHTML = "Car owner is " +
car.owner;
</script>
```

Mais vous ne pouvez PAS réaffecter un objet constant:

```
<p>Mais vous ne pouvez PAS réaffecter un objet constant:</p>

<p id="demo"></p>

<script>
try {
  const car = {type:"Fiat", model:"500", color:"white"};
  car = {type:"Volvo", model:"EX60", color:"red"};
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>
```

Les tableaux constants peuvent changer

Vous pouvez modifier les éléments d'un tableau constant:

```
<p>Déclarer un tableau constant ne rend PAS les éléments
immuables:</p>

<p id="demo"></p>

<script>
// Création d’un tableau
const cars = ["Saab", "Volvo", "BMW"];

// Changer un élément
cars[0] = "Toyota";

// Ajouter un élément
cars.push("Audi");

// Afficher le tableau
document.getElementById("demo").innerHTML = cars;
</script>
```

Mais vous ne pouvez PAS réaffecter un tableau constant:

```
<p id="demo"></p>

<script>
try {
  const cars = ["Saab", "Volvo", "BMW"];
  cars = ["Toyota", "Volvo", "Audi"];
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>
```

Type de données en Javascript

Les variables JavaScript peuvent contenir de nombreux types de données: nombres, chaînes, objets et plus:

```
var length = 16;                               // Nombres
var lastName = "Johnson";                      // Chaînes
var x = {firstName:"John", lastName:"Doe"}; // Objet
```

Lors de l'ajout d'un nombre et d'une chaîne, JavaScript traitera le nombre comme une chaîne.

```
<p id="demo"></p>

<script>
var x = 16 + "Volvo";
document.getElementById("demo").innerHTML = typeof x;
</script>
```

JavaScript évalue les expressions de gauche à droite. Différentes séquences peuvent produire des résultats différents.

```
var x = 16 + 4 + "Volvo";

var x = "Volvo" + 16 + 4;
```

Dans le premier exemple, JavaScript traite 16 et 4 comme des nombres, jusqu'à ce qu'il atteigne «Volvo».

Dans le deuxième exemple, puisque le premier opérande est une chaîne, tous les opérandes sont traités comme des chaînes.

Nombres :

JavaScript n'a qu'un seul type de nombres.

Les nombres peuvent être écrits avec ou sans décimales et les nombres très grands ou très petits peuvent être écrits avec une notation scientifique (exponentielle):

```
<p id="demo"></p>

<script>
var y = 123e5;
var z = 123e-5;

document.getElementById("demo").innerHTML =
y + "<br>" + z;
</script>
```

Booleans:

```
<p id="demo"></p>

<script>
var x = 5;
var y = 5;
var z = 6;
document.getElementById("demo").innerHTML =
(x == y) + "<br>" + (x == z);
</script>
```

Tableaux JavaScript

Les tableaux JavaScript sont écrits entre crochets.

Les éléments du tableau sont séparés par des virgules.

Le code suivant déclare (crée) un tableau appelé voitures, contenant trois éléments (noms de voiture):

```
<p>Les index de tableau sont de base zéro, ce qui signifie que le premier élément est [0].</p>

<p id="demo"></p>

<script>
var cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[0];
</script>
```

Objets JavaScript

Les objets JavaScript sont écrits avec des accolades {}.
Les propriétés de l'objet sont écrites sous forme de paires nom: valeur, séparées par des virgules.

```
<p id="demo"></p>

<script>
var person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " a " + person.age + " ans.";
</script>
```

L'opérateur typeof

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof "Jacques" + "<br>" +
typeof ["Saab","Volvo","BMW"];
</script>
```

Indéfinie

En JavaScript, une variable sans valeur a la valeur indéfinie. Le type est également indéfini.

```
<p id="demo"></p>

<script>
var car;
document.getElementById("demo").innerHTML =
car + "<br>" + typeof car;
</script>
```

Toute variable peut être vidée, en définissant la valeur sur undefined. Le type sera également indéfini.

```
<p id="demo"></p>

<script>
var car = "Volvo";
car = undefined;
document.getElementById("demo").innerHTML =
car + "<br>" + typeof car;
</script>
```

Valeurs Vides

Une valeur vide n'a rien à voir avec undefined.

Une chaîne vide a à la fois une valeur légale et un type.

```
<script>
var car = "";
document.getElementById("demo").innerHTML =
"The value is: " +
car + "<br>" +
"The type is: " + typeof car;
</script>
```

Null

En JavaScript, null c’est «rien». C'est censé être quelque chose qui n'existe pas.

Mais, en JavaScript, le type de données null est un objet.

Vous pouvez considérer comme un bug en JavaScript que typeof null est un objet. Il doit être nul.

Vous pouvez vider un objet en le définissant sur null:

```
<p id="demo"></p>

<script>
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
person = null;
document.getElementById("demo").innerHTML = typeof person;
</script>
```

Vous pouvez également vider un objet en le définissant sur undefined.

Différence entre undefined et null

undefined et null sont égaux en valeur mais de type différent:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof undefined + "<br>" +
typeof null + "<br><br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>
```

Données complexes

L'opérateur typeof peut renvoyer l'un des deux types complexes:

- fonction
- objet

L'opérateur typeof renvoie «objet» pour les objets, les tableaux et la valeur null.

L'opérateur typeof ne renvoie pas "objet" pour les fonctions.

Fonctions Javascript

Une fonction JavaScript est un bloc de code conçu pour effectuer une tâche particulière.

Une fonction JavaScript est exécutée lorsque "quelque chose" l'appelle.

```
<p id="demo"></p>

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
console.log(myFunction(4, 3));
</script>
```

```
var myFunction = function(p1,p2) {            //ES5
      return p1 * p2;
}

const myFunction = (p1,p2) => p1*p2;          //ES6
```

Syntaxe de la fonction JavaScript

Une fonction JavaScript est définie avec le mot-clé **function**, suivi d'un nom, suivi de parenthèses ().

Les noms de fonction peuvent contenir des lettres, des chiffres, des traits de soulignement et des signes dollar (mêmes règles que les variables).

Les parenthèses peuvent inclure des noms de paramètres séparés par

des virgules:
(paramètre1, paramètre2, ...)

Le code à exécuter, par la fonction, est placé entre accolades: {}

Les paramètres de fonction sont répertoriés entre parenthèses () dans la définition de fonction.

Les arguments de fonction sont les valeurs reçues par la fonction lorsqu'elle est appelée.

À l'intérieur de la fonction, les arguments (les paramètres) se comportent comme des variables locales.

Une fonction est sensiblement la même qu'une procédure ou un sous-programme, dans d'autres langages de programmation.

Appel de fonction

Le code à l'intérieur de la fonction s'exécutera lorsque "quelque chose" appellera la fonction:

- Lorsqu'un événement se produit (lorsqu'un utilisateur clique sur un bouton)
- Lorsqu'il est invoqué (appelé) à partir du code JavaScript
- Automatiquement (auto-invoqué)

Seulement, la fonction auto-invoquée s’exécute immédiatement et on n’a donc pas de flexibilité par rapport à cela : une fonction auto-invoquée s’exécutera toujours juste après sa déclaration.

Retour de fonction

Lorsque JavaScript atteint une instruction return, la fonction s'arrête de s'exécuter.

Si la fonction a été appelée à partir d'une instruction, JavaScript retournera pour exécuter le code après l'instruction d'appel.

Les fonctions calculent souvent une valeur de retour. La valeur de retour est "renvoyée" à l'appelant":

```
<p>Cet exemple appelle une fonction qui effectue un calcul et
renvoie le résultat:</p>

<p id="demo"></p>

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>
```

L'opérateur () appelle la fonction

En utilisant l'exemple ci-dessous, toCelsius fait référence à l'objet fonction et toCelsius () fait référence au résultat de la fonction.

L'accès à une fonction sans () renverra l'objet fonction au lieu du résultat de la fonction.

```
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius;
console.log(toCelsius);
</script>
```

Essayez également avec un paramètre ajouté à la fonction.

Fonctions utilisées comme valeurs de variable

Les fonctions peuvent être utilisées de la même manière que vous utilisez des variables, dans tous les types de formules, d'affectations et de calculs.

Au lieu d'utiliser une variable pour stocker la valeur de retour d'une fonction:

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
```

Vous pouvez utiliser la fonction directement, comme valeur de variable:

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Variables locales

Les variables déclarées dans une fonction JavaScript deviennent LOCALES pour la fonction.

Les variables locales ne sont accessibles qu'à partir de la fonction.

```
<p>En dehors de myFunction() carName n'est pas défini.</p>

<p id="demo1"></p>

<p id="demo2"></p>

<script>
myFunction();

function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML =
  typeof carName + " " + carName;
}

document.getElementById("demo2").innerHTML =
typeof carName;
</script>
```

Comme les variables locales ne sont reconnues que dans leurs fonctions, les variables portant le même nom peuvent être utilisées dans différentes fonctions.

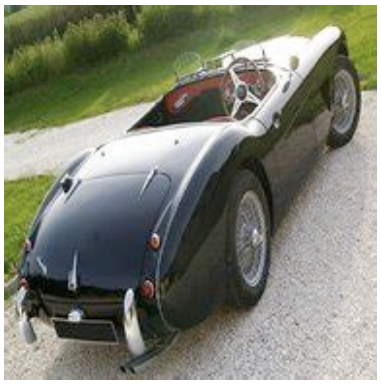
Les variables locales sont créées au démarrage d'une fonction et supprimées lorsque la fonction est terminée.

Objets Javascript

Objets, propriétés et méthodes de la vie réelle

Dans la vraie vie, une voiture est un objet.

Une voiture a des propriétés telles que le poids et la couleur, ainsi que des méthodes telles que le démarrage et l'arrêt:

Objet	Propriétés	Méthodes
	car.name = Aston	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = black	car.stop()

Toutes les voitures ont les mêmes propriétés, mais les valeurs des propriétés diffèrent d'une voiture à l'autre.

Toutes les voitures ont les mêmes méthodes, mais les méthodes sont effectuées à des moments différents.

Objets JavaScript

Vous avez déjà appris que les variables JavaScript sont des conteneurs pour les valeurs de données.

Ce code attribue une valeur simple (Aston) à une variable nommée voiture:

```
<p id="demo"></p>
<script>
// Créer et afficher une variable:
var car = "Fiat";
document.getElementById("demo").innerHTML = car;
</script>
```

Les objets sont aussi des variables. Mais les objets peuvent contenir de nombreuses valeurs.

Ce code attribue de nombreuses valeurs (Aston, 500, Noir) à une variable nommée voiture:

```
<p id="demo"></p>

<script>
// Crée un objet:
var car = {type:"Fiat", model:"500", color:"white"};

// Afficher des informations de l'objet:
document.getElementById("demo").innerHTML = "The car type is " +
car.type;
</script>
```

Les valeurs sont écrites sous forme de paires nom: valeur (nom et valeur séparés par deux points).

Les objets JavaScript sont des conteneurs pour des valeurs appelées propriétés ou méthodes.

Définition d'objet

Vous définissez (et créez) un objet JavaScript:

```
<p id="demo"></p>

<script>
// Crée un objet:
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};

// Afficher des informations de l’objet:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

Les espaces et les sauts de ligne ne sont pas importants. Une définition d'objet peut s'étendre sur plusieurs lignes:

```
<p id="demo"></p>

<script>

var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

Comment accéder aux propriétés de l'objet:

Vous pouvez accéder aux propriétés des objets de deux manières:

objectName.propertyName

ou

objectName["propertyName"]

```
<p id="demo"></p>

<script>

var person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566
};

document.getElementById("demo").innerHTML =
person.firstName + " " + person.lastName;
</script>
```

Remplacer `person.lastName` par `person["lastName"]`

Méthodes d'objet

Les objets peuvent également avoir des méthodes.

Les méthodes sont des actions qui peuvent être effectuées sur des objets.

Les méthodes sont stockées dans les propriétés en tant que définitions de fonction. Une méthode est une fonction stockée en tant que propriété.

Propriété : nom_complet
Valeur de la propriété :
`function() {return this.firstName + " " + this.lastName;}`

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
      return this.firstName + " " + this.lastName;
  }
```

Accès aux méthodes d'objets

Vous accédez à une méthode objet avec la syntaxe suivante:

objectName.methodName()

```
<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id     : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>
```

Si vous accédez à une méthode sans les parenthèses (), elle renverra la définition de la fonction:

essayer le code précédent mais en enlevant les parenthèses de la fonction.

Ne déclarez pas les chaînes, les nombres et les booléens en tant qu'objets!

Lorsqu'une variable JavaScript est déclarée avec le mot-clé "**new**", la variable est créée en tant qu'objet:

```
var x = new String();        // Déclare x comme object string
var y = new Number();        // Déclare y comme objet Number
var z = new Boolean();       // Déclare z comme un objet Boolean
```

Évitez les objets String, Number et Boolean. Ils compliquent votre code et ralentissent la vitesse d'exécution.

Evénements Javascript

Les événements HTML sont des «choses» qui arrivent aux éléments HTML.

Lorsque JavaScript est utilisé dans les pages HTML, JavaScript peut "réagir" à ces événements.

Événements HTML

Un événement HTML peut être une action du navigateur ou une action d'un utilisateur.

Voici quelques exemples d'événements HTML:

- ➔ Une page Web HTML a fini de se charger
- ➔ Un champ de saisie HTML a été modifié
- ➔ Un bouton HTML a été cliqué

Souvent, lorsque des événements se produisent, vous voudrez peut-être faire quelque chose.

JavaScript vous permet d'exécuter du code lorsque des événements sont détectés.

HTML permet d'ajouter des attributs de gestionnaire d'événements, avec du code JavaScript, aux éléments HTML.

Avec des guillemets simples:

```
<element event = 'un peu de JavaScript'>
```

Avec des guillemets doubles:

```
<element event = "un peu de JavaScript">
```

Dans l'exemple suivant, un attribut onclick (avec code) est ajouté à un élément <button>:

```
<button
onclick="document.getElementById('demo').innerHTML=Date()">L’heure
est?</button>

<p id="demo"></p>
```

Dans l'exemple ci-dessus, le code JavaScript modifie le contenu de l'élément avec id = "demo".

Dans l'exemple suivant, le code modifie le contenu de son propre élément (en utilisant this.innerHTML):

```
<button onclick="this.innerHTML=Date()">L’heure est?</button>
```

Le code JavaScript comporte souvent plusieurs lignes. Il est plus courant de voir des attributs d'événement appelant des fonctions:

```
<p>Cliquez sur le bouton pour afficher la date et l’heure</p>

<button onclick="displayDate()">Date et Heure actuelles?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
ou
var displayDate = () => {
     document.getElementById("demo").innerHTML = Date();
}
```

Événements HTML courants

Voici une liste de quelques événements HTML courants:

onchange	Un élément HTML a été modifié
onclick	L'utilisateur clique sur un élément HTML
onmouseover	L'utilisateur déplace la souris sur un élément HTML
onmouseout	L'utilisateur éloigne la souris d'un élément HTML
onkeydown	L'utilisateur appuie sur une touche du clavier
onload	Le navigateur a fini de charger la page

La liste est beaucoup plus longue en réalité bien sûr.

Que peut faire JavaScript?

Les gestionnaires d'événements peuvent être utilisés pour gérer et vérifier les entrées utilisateur, les actions utilisateur et les actions du navigateur:

- ➔ Ce qui doit être fait à chaque fois qu'une page se charge
- ➔ Choses à faire lorsque la page est fermée
- ➔ Action à effectuer lorsqu'un utilisateur clique sur un bouton
- ➔ Contenu à vérifier lorsqu'un utilisateur entre des données
- ➔ Et plus ...

De nombreuses méthodes différentes peuvent être utilisées pour laisser JavaScript fonctionner avec les événements:

- ➔ Les attributs d'événement HTML peuvent exécuter directement du code JavaScript
- ➔ Les attributs d'événement HTML peuvent appeler des fonctions JavaScript
- ➔ Vous pouvez attribuer vos propres fonctions de gestionnaire d'événements aux éléments HTML
- ➔ Vous pouvez empêcher l'envoi ou la gestion des événements
- ➔ Et plus ...

Vous en apprendrez beaucoup plus sur les événements et les gestionnaires d'événements dans les chapitres HTML DOM.

Méthodes Javascript pour les chaînes de caractères

Les méthodes String vous aident à travailler avec des chaînes de caractères.

Méthodes et propriétés de chaîne

Les valeurs primitives, comme "John Doe", ne peuvent pas avoir de propriétés ou de méthodes (car ce ne sont pas des objets).

Mais avec JavaScript, les méthodes et les propriétés sont également disponibles pour les valeurs primitives, car JavaScript traite les valeurs primitives comme des objets lors de l'exécution de méthodes et de propriétés.

Longueur de chaîne

La propriété length renvoie la longueur d'une chaîne:

```
<p id="demo"></p>

<script>
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
document.getElementById("demo").innerHTML = sln;
</script>
```

Chercher une chaîne dans une autre chaîne

La méthode indexOf() renvoie l'index de (la position de) la première occurrence d'un texte spécifié dans une chaîne:

```
<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>
```

JavaScript compte les positions à partir de zéro.

La méthode lastIndexOf () renvoie l'index de la dernière occurrence d'un texte spécifié dans une chaîne:

```
<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
document.getElementById("demo").innerHTML = pos;
</script>
```

Les deux indexOf() et lastIndexOf() renvoient **-1** si le texte est introuvable.

```
<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("John");
document.getElementById("demo").innerHTML = pos;
</script>
```

Les deux méthodes acceptent un deuxième paramètre comme position de départ de la recherche:

```
<p id="demo"></p>
<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate",15);
document.getElementById("demo").innerHTML = pos;
</script>
```

Méthode search()

La méthode search () recherche une chaîne pour une valeur spécifiée et renvoie la position de la correspondance:

```
<p id="demo"></p>

<script>
var str = "Please locate where 'locate' occurs!";
var pos = str.search(/regex/i);
document.getElementById("demo").innerHTML = pos;
</script>
```

Avez-vous remarqué une différence entre les deux méthodes, indexOf () et search ()?

Elles acceptent les mêmes arguments (paramètres), et renvoient la même valeur?

Les deux méthodes ne sont PAS égales. Voici les différences:

- ➔ La méthode search () ne peut pas prendre un deuxième argument de position de départ.
- ➔ La méthode indexOf() ne peut pas prendre de valeurs de recherche puissantes (expressions régulières).

Extraction de parties de chaîne

Il existe 3 méthodes pour extraire une partie d'une chaîne:

- substring(*start*, *end*)
- substr(*start*, *length*)
- slice(*start*, *end*)

La méthode slice ()

slice () extrait une partie d'une chaîne et renvoie la partie extraite dans une nouvelle chaîne.

La méthode prend 2 paramètres: la position de départ et la position de fin (fin non incluse).

Cet exemple découpe une partie d'une chaîne de la position 7 à la position 12 (13-1):

```
<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
document.getElementById("demo").innerHTML = res;
</script>
```

Si un paramètre est négatif, la position est comptée à partir de la fin de la chaîne.
Cet exemple découpe une partie d'une chaîne de la position -12 à la position -6:

```
<p id="demo"></p>
<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);
document.getElementById("demo").innerHTML = res;
</script>
```

Si vous omettez le deuxième paramètre, la méthode découpera le reste de la chaîne:

```
<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.slice(7);
document.getElementById("demo").innerHTML = res;
</script>
```

ou en comptant à partir de la fin :

```
var res = str.slice(-12);
```

La méthode substring ()

substring () est similaire à slice ().

La différence est que substring () ne peut pas accepter les index négatifs.

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

Et pareil, si vous omettez le deuxième paramètre, substring () découpera le reste de la chaîne.

La méthode substr()

substr() est similaire à slice().

La différence est que le deuxième paramètre spécifie la longueur de la pièce extraite.

```
<p id="demo"></p>

<script>
var str = "Apple, Banana, Kiwi";
var res = str.substr(7,6);
document.getElementById("demo").innerHTML = res;
</script>
```

Si vous omettez le deuxième paramètre, substr () découpera le reste de la chaîne également.

Si le premier paramètre est négatif, la position compte à partir de la fin de la chaîne.

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(-4);
```

Le résultat sera:
Kiwi

Remplacer le contenu d'une chaîne de caractère

```
<button onclick="myFunction()">Essayer</button>

<p id="demo">Please visit Microsoft!</p>

<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace("Microsoft","Linux");
  document.getElementById("demo").innerHTML = txt;
}
</script>
```

La méthode replace() ne modifie pas la chaîne sur laquelle elle est appelée. Elle renvoie une nouvelle chaîne.

Par défaut, la méthode replace () est sensible à la casse.
L'écriture de MICROSOFT (avec des majuscules) ne fonctionnera pas.

Pour un remplacement insensible à la casse, utilisez une expression régulière avec un indicateur /i (insensible)

Notez que les expressions régulières sont écrites sans guillemets.

Pour remplacer toutes les correspondances, utilisez une expression régulière avec un indicateur /g (correspondance globale):

```
<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft and Microsoft!</p>

<script>
function myFunction() {
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace(/Microsoft/g,"Linux");
  document.getElementById("demo").innerHTML = txt;
}
</script>
```

Conversion en majuscules et minuscules

Une chaîne est convertie en majuscules avec toUpperCase() et est convertie en minuscules avec toLowerCase():

```
<p id="demo">Hello World!</p>

<script>
function myFunction() {
  var text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML = text.toUpperCase();
  document.getElementById("demo").innerHTML = text.toLowerCase();
}
</script>
```

Alternez les deux méthodes dans le code sinon la seconde effacera le résultat de la première.

La méthode concat()

concat() joint une ou plusieurs chaîne(s) de caractères.

```
<p id="demo"></p>

<script>
var text1 = "Hello";
var text2 = "World!";
var text3 = text1.concat(" ",text2);
document.getElementById("demo").innerHTML = text3;
</script>
```

La méthode concat () peut être utilisée à la place de l'opérateur plus.

Toutes les méthodes de chaîne renvoient une nouvelle chaîne. Ils ne modifient pas la chaîne d'origine.
Formellement dit: les chaînes sont immuables: les chaînes ne peuvent pas être modifiées, seulement remplacées.

Il est également possible d’accéder à un élément de la chaîne de caractères avec un index :

```
var str = "HELLO WORLD";
str[0];                    // donne H
```

Comment convertir une chaîne de caractère en tableau :

Une chaîne peut être convertie en tableau avec la méthode split():

```
<p id="demo"></p>

<script>
function myFunction() {
  var str = "a,b,c,d,e,f";
  var arr = str.split(",");
  document.getElementById("demo").innerHTML = arr[0];
}
</script>
```

Si le séparateur est omis, le tableau retourné contiendra la chaîne entière dans l'index [0].

Si le séparateur est "", le tableau renvoyé sera un tableau de caractères uniques:

```
<p id="demo"></p>

<script>
var str = "Hello";
var arr = str.split("");
var text = "";
var i;
for (i = 0; i < arr.length; i++) {
  text += arr[i] + "<br>"
}
document.getElementById("demo").innerHTML = text;
</script>
```

Méthodes de nombres en Javascript

Conversion de variables en nombres

Il existe 3 méthodes JavaScript qui peuvent être utilisées pour convertir des variables en nombres:

- ➔ La méthode Number ()
- ➔ La méthode parseInt ()
- ➔ La méthode parseFloat ()

Ces méthodes ne sont pas des méthodes numériques, mais des méthodes JavaScript globales.

Méthodes JavaScript globales

Les méthodes globales JavaScript peuvent être utilisées sur tous les types de données JavaScript.

Voici les méthodes les plus pertinentes lorsque vous travaillez avec des nombres:

- ➔ **Number()** Renvoie un nombre, converti à partir de son argument.
- ➔ **parseFloat()** Analyse son argument et renvoie un nombre à virgule flottante
- ➔ **parseInt()** Analyse son argument et retourne un entier

Number() peut être utilisé pour convertir des variables JavaScript en nombres:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  Number(true) + "<br>" +
  Number(false) + "<br>" +
  Number("10") + "<br>" +
  Number("  10") + "<br>" +
  Number("10  ") + "<br>" +
  Number(" 10  ") + "<br>" +
  Number("10.33") + "<br>" +
  Number("10,33") + "<br>" +
  Number("10 33") + "<br>" +
  Number("John");
</script>
```

Si le nombre ne peut pas être converti, NaN (Not a Number) est renvoyé.

La méthode parseInt()

parseInt() analyse une chaîne et renvoie un nombre entier. Les espaces sont autorisés. Seul le premier nombre est renvoyé:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  parseInt("10") + "<br>" +
  parseInt("10.33") + "<br>" +
  parseInt("10 6") + "<br>" +
  parseInt("10 years") + "<br>" +
  parseInt("years 10");
</script>
```

La méthode parseFloat()

parseFloat() analyse une chaîne et renvoie un nombre. Les espaces sont autorisés. Seul le premier nombre est renvoyé:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  parseFloat("10") + "<br>" +
  parseFloat("10.33") + "<br>" +
  parseFloat("10 6") + "<br>" +
  parseFloat("10 years") + "<br>" +
  parseFloat("years 10");
</script>
```

Il existe plusieurs autres méthodes sur les propriétés des nombres comme les max et min bien sûr.

Tableaux en Javascript

Créer un tableau

```
<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

Les espaces et les sauts de ligne ne sont pas importants. Une déclaration peut s'étendre sur plusieurs lignes:

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

On peut également utiliser le mot-clé Array :

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Les deux exemples ci-dessus font exactement la même chose. Il n'est pas nécessaire d'utiliser new Array ().
Pour plus de simplicité, de lisibilité et de rapidité d'exécution, utilisez le premier (méthode littérale de tableau).

Vous accédez à un élément de tableau en vous référant au numéro d'index.

Cette instruction accède à la valeur du premier élément dans les voitures:

```
<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>
```

Changer un élément de tableau :

```
<p>Les éléments du tableau JavaScript sont accessibles à l'aide d'index numériques (à partir de 0).</p>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
```

Les tableaux sont des objets, si vous leur appliquer typeof(), il retournera object.

Et les éléments des tableaux peuvent être des objets :

Les tableaux sont des types d'objets spéciaux.

Pour cette raison, vous pouvez avoir des variables de types différents dans le même tableau.

Vous pouvez avoir des objets dans un tableau. Vous pouvez avoir des fonctions dans un tableau. Vous pouvez avoir des tableaux dans un tableau:

```
myArray [0] = Date.now;
myArray [1] = maFonction;
myArray [2] = myCars;
```

Propriétés et méthodes des tableaux

La vraie force des tableaux JavaScript réside dans les propriétés et méthodes de tableau intégrées:

La propriété **length** d'un tableau renvoie la longueur d'un tableau (le nombre d'éléments du tableau).

Cependant, la propriété **length** retourne toujours un de plus que l'index de tableau le plus élevé.

```
<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.length;
</script>
```

Accéder aux éléments du tableau avec une boucle :

Le moyen le plus sûr de parcourir un tableau est d'utiliser une boucle for:

```
<p id="demo"></p>

<script>
var fruits, text, fLen, i;
fruits = ["Banane", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>
```

Vous pouvez également utiliser la fonction **Array.forEach()**:

```
<p id="demo"></p>

<script>
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";
document.getElementById("demo").innerHTML = text;

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
</script>
```

Ajouter des éléments dans un tableau:

Le moyen le plus simple d'ajouter un nouvel élément à un tableau consiste à utiliser la méthode push():

```
<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>
```

Un nouvel élément peut également être ajouté à un tableau à l'aide de la propriété length:

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.length;
</script>
```

Pour accéder à un élément du tableau :

```
var first = fruits[0];
```

Pour accéder au dernier élément du tableau :

```
var last = fruits[fruits.length - 1];
```

ATTENTION ! L'ajout d'éléments avec des index élevés peut créer des «trous» indéfinis dans un tableau:

```
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";

fLen = fruits.length;
text = "";
for (i = 0; i < fLen; i++) {
  text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
```

Tableaux associatifs

De nombreux langages de programmation prennent en charge les tableaux avec des index nommés.

Les tableaux avec des index nommés sont appelés tableaux associatifs (ou hachages).

JavaScript ne prend pas en charge les tableaux avec des index nommés.

En JavaScript, les tableaux utilisent toujours des index numérotés.

```
<p id="demo"></p>

<script>
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
```

ATTENTION !!
Si vous utilisez des index nommés, JavaScript redéfinira le tableau en un objet standard.
Après cela, certaines méthodes et propriétés de tableau produiront des résultats incorrects.

```
<p id="demo"></p>

<script>
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
```