Cours pour debutant en javascript

Le but de ce tutoriel est de guider toute personne qui aborde pour la première fois ou toute personne souhaitant développer des compétences en Javascript

Introduction

1. Les fondamentaux en javascript

1.1 Ecrire ton premier script

Bonjour le monde! Cette partie du didacticiel concerne le noyau JavaScript, le langage lui-même.

Mais nous avons besoin d'un environnement de travail pour exécuter nos scripts et, puisque ce livre est en ligne, le navigateur est un bon choix. Nous allons réduire au minimum le nombre de commandes spécifiques au navigateur (comme alert) afin que vous ne passiez pas de temps dessus si vous prévoyez de vous concentrer sur un autre environnement (comme Node.js). Nous nous concentrerons sur JavaScript dans le navigateur dans la prochaine partie du didacticiel.

Voyons d'abord comment nous attachons un script à une page Web. Pour les environnements côté serveur (comme Node.js), vous pouvez exécuter le script avec une commande comme "node my.js".

La balise « script »

Les programmes JavaScript peuvent être insérés presque n'importe où dans un document HTML à l'aide de la balise suivant:

```
    <script>
```

Par exemple:

```
    <!DOCTYPE HTML>
    <html>

        <body>

            <p>Before the script...</p>

            <script>
                alert( 'Hello, world!' );
            </script>

            <p>...After the script.</p>

        </body>

    </html>
```

Vous pouvez exécuter l'exemple en cliquant sur le bouton « Play » dans le coin supérieur droit de la boîte ci-dessus.

La balise `<script>` contient du code JavaScript qui est automatiquement exécuté lorsque le navigateur traite la balise.

Balisage moderne

La `<script>`balise a quelques attributs qui sont rarement utilisés de nos jours mais qui peuvent encore être trouvés dans l'ancien code :

L' typeattribut :`<script type=…>` L'ancienne norme HTML, HTML4, nécessitait un script pour avoir un fichier type. Habituellement, c'était le cas type="text/javascript". Ce n'est plus nécessaire. De plus, le standard HTML moderne a totalement changé la signification de cet attribut. Maintenant, il peut être utilisé pour les modules JavaScript. Mais c'est un sujet avancé, nous parlerons des modules dans une autre partie du tutoriel.

L' languageattribut :`<script language=…>` Cet attribut était destiné à montrer la langue du script. Cet attribut n'a plus de sens car JavaScript est le langage par défaut. Il n'est pas nécessaire de l'utiliser.

Commentaires avant et après les scripts. Dans les livres et les guides très anciens, vous pouvez trouver des commentaires à l'intérieur des `<script>`balises, comme ceci :

```
    <script type="text/javascript">
        <!--...//-->
    </script>
```

Cette astuce n'est pas utilisée dans le JavaScript moderne. Ces commentaires masquent le code JavaScript des anciens navigateurs qui ne savaient pas comment traiter la `<script>`balise. Étant donné que les navigateurs publiés au cours des 15 dernières années n'ont pas ce problème, ce type de commentaire peut vous aider à identifier un code vraiment ancien.

Scripts externes

Si nous avons beaucoup de code JavaScript, nous pouvons le mettre dans un fichier séparé.

Les fichiers de script sont attachés au HTML avec l' srcattribut :

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

Voici `/path/to/script.js` un chemin absolu vers le script à partir de la racine du site. On peut également fournir un chemin relatif à partir de la page courante. Par exemple, `src="script.js"`, tout comme `src="./script.js"`, signifierait un fichier "script.js"dans le dossier actuel.

Nous pouvons également donner une URL complète. Par exemple:

```
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js">
</script>
```

Pour attacher plusieurs scripts, utilisez plusieurs balises :

```
    <script src="/js/script1.js"></script>
    <script src="/js/script2.js"></script>
```

```
## Veuillez noter:

En règle générale, seuls les scripts les plus simples sont mis en HTML. Les
plus complexes résident dans des fichiers séparés.

L'avantage d'un fichier séparé est que le navigateur le télécharge et le
stocke dans son cache .

Les autres pages qui font référence au même script le prendront dans le
cache au lieu de le télécharger, de sorte que le fichier n'est en fait
téléchargé qu'une seule fois.

Cela réduit le trafic et rend les pages plus rapides.
```

```
Si src est défini, le contenu du script est ignoré.
Une seule <script> balise ne peut pas contenir à la fois l' srcattribut et
le code.

Cela ne fonctionnera pas :

<script src="file.js">
  alert(1);
</script>
```

Nous devons choisir soit un externe, `<script src="…">`soit un régulier `<script>`avec code.

L'exemple ci-dessus peut être divisé en deux scripts pour fonctionner :

```
<script src="file.js"></script>
<script>
  alert(1);
</script>
```

1.2 Les Variables

La plupart du temps, une application JavaScript doit travailler avec des informations. Voici deux exemples :

- Une boutique en ligne – les informations peuvent inclure des produits vendus et un panier.
- Une application de chat - les informations peuvent inclure des utilisateurs, des messages et bien plus encore. Des variables sont utilisées pour stocker ces informations.

Une variable

Une variable est un « stockage nommé » pour les données. Nous pouvons utiliser des variables pour stocker des goodies, des visiteurs et d'autres données.

Pour créer une variable en JavaScript, utilisez le letmot - clé.

L'instruction ci-dessous crée (en d'autres termes : déclare) une variable avec le nom « message » :

```
    let message;
```

Maintenant, nous pouvons y mettre des données en utilisant l'opérateur d'affectation égale(=)

```
let message;

message = 'Hello';
```

La chaîne est maintenant enregistrée dans la zone mémoire associée à la variable. Nous pouvons y accéder en utilisant le nom de la variable :

```
let message;
message = 'Hello';

alert(message);
```

Pour être concis, nous pouvons combiner la déclaration et l'affectation de la variable en une seule ligne :

```
let message = 'Hello';

alert(message);
```

Nous pouvons également déclarer plusieurs variables sur une seule ligne :

```
let user = 'John', age = 25, message = 'Hello';
```

1.3 Les types de variables

Une valeur en JavaScript est toujours d'un certain type. Par exemple, une chaîne ou un nombre.

Il existe huit types de données de base en JavaScript. Ici, nous les couvrirons en général et dans les prochains chapitres, nous parlerons de chacun d'eux en détail.

Nous pouvons mettre n'importe quel type dans une variable. Par exemple, une variable peut à un moment être une chaîne puis stocker un nombre :

```
let message = "hello";
message = 123456;
```

Les langages de programmation qui permettent de telles choses, comme JavaScript, sont appelés « typés dynamiquement », ce qui signifie qu'il existe des types de données, mais que les variables ne sont liées à aucun d'entre eux.

Le type nombre

```
let n = 123;
n = 12.345;
```

Le type de nombre représente à la fois des nombres entiers et des nombres à virgule flottante.

Il existe de nombreuses opérations pour les nombres, par exemple la multiplication *, la division /, l'addition +, la soustraction -, etc.

Outre les nombres normaux, il existe des "valeurs numériques spéciales" qui appartiennent également à ce type de données : Infinity, -Infinityet NaN.

- Infinityreprésente l' infini mathématique . C'est une valeur spéciale qui est supérieure à n'importe quel nombre.

Nous pouvons l'obtenir en résultat de la division par zéro :

```
alert( 1 / 0 ); //Infinity
```

Ou simplement le référencer directement :

```
alert(Infinity); //Infinity
```

- NaNreprésente une erreur de calcul. C'est le résultat d'une opération mathématique incorrecte ou indéfinie, par exemple :

```
alert( "une chaine de caratere" / 2 ); //NaN
```

NaNest collant. Toute autre opération sur les NaNretours NaN:

```
alert( "une chaine de caratere" / 2 + 5 ); //NaN
```

Donc, s'il y a NaNquelque part dans une expression mathématique, il se propage à l'ensemble du résultat.

```
Les opérations mathématiques sont sûres

Faire des mathématiques est « sûr » en JavaScript. On peut tout faire :
diviser par zéro, traiter les chaînes non numériques comme des nombres,
etc.

Le script ne s'arrêtera jamais avec une erreur fatale (« die »). Au pire,
nous aurons NaNcomme résultat.
```

Les valeurs numériques spéciales appartiennent formellement au type « nombre ». Bien sûr, ce ne sont pas des nombres au sens commun du terme.

Nous verrons plus sur le travail avec les nombres dans le chapitre Nombres .

Le type BigInt

En JavaScript, le type « nombre » ne peut pas représenter des valeurs entières supérieures à (c'est-à-dire), ou inférieures à pour les négatifs. C'est une limitation technique causée par leur représentation interne. `(253-1)9007199254740991-(253-1)`

Dans la plupart des cas, cela suffit, mais parfois nous avons besoin de très gros nombres, par exemple pour la cryptographie ou les horodatages à la microseconde.

BigInt type a été récemment ajouté au langage pour représenter des entiers de longueur arbitraire.

Une **BigInt** valeur est créée en ajoutant nà la fin d'un entier :

```
const bigInt = 1234567890123456789012345678901234567890n;
```

Comme les BigIntchiffres sont rarement nécessaires, nous ne les couvrons pas ici, mais leur consacrons un chapitre séparé BigInt . Lisez-le quand vous avez besoin de si gros chiffres.

```
Problèmes de compatibilité

À l'heure actuelle, BigIntest pris en charge dans
Firefox/Chrome/Edge/Safari, mais pas dans IE.
```

Vous pouvez consulter le tableau de compatibilité MDN BigInt pour savoir quelles versions d'un navigateur sont prises en charge.

Type Chaîne de caractères

Une chaîne en JavaScript doit être entourée de guillemets.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

En JavaScript, il existe 3 types de citations.

1. Guillemets doubles : "Hello".
2. Guillemets simples : 'Hello'.
3. Backticks : `Hello`.

Les guillemets simples et doubles sont des guillemets « simples ». Il n'y a pratiquement aucune différence entre eux en JavaScript.

Les backticks sont des citations de « fonctionnalité étendue ». Ils nous permettent d'intégrer des variables et des expressions dans une chaîne en les enveloppant dans ${...}, par exemple :

```
let name = "John";

// afficher la valeur d'une chaine de caractere
alert( `Hello, ${name}!` ); // Hello, John!

// afficher la valeur d'un calul
alert( `le resultat est ${1 + 2}` ); // le resultat est 3
```

L'expression à l'intérieur ${...}est évaluée et le résultat devient une partie de la chaîne. On peut y mettre n'importe quoi : une variable comme nameou une expression arithmétique comme 1 + 2ou quelque chose de plus complexe.

Veuillez noter que cela ne peut être fait que par backticks. D'autres citations n'ont pas cette fonctionnalité d'intégration !

```
alert( "le resultat est ${1 + 2}"); // le resultat est ${1 + 2}
```

Type Booléen (type logique)

Le type booléen n'a que deux valeurs : trueet false.

Ce type est couramment utilisé pour stocker des valeurs oui/non : truesignifie « oui, correct » et falsesignifie « non, incorrect ».

Par exemple:

```
let nameFieldChecked = true;
let ageFieldChecked = false;
```

Les valeurs booléennes résultent également de comparaisons :

```
let isGreater = 4 > 1;
alert( isGreater ); // affichera true (car 4 est vraiment superieur a 1)
```

Nous aborderons plus en détail les booléens dans le chapitre Opérateurs logiques .

1.4 Les operations

Nous connaissons de nombreux opérateurs de l'école. Ce sont des choses comme l'addition +, la multiplication *, la soustraction -, etc.

Pour ajouter deux variables, utilisez le signe + :

```
let x = 8;
let y = 5;
let total = x + y; // le resultat sera total = 13
```

À l'inverse, la soustraction utilise le signe - :

```
let x = 10;
let y = 2;
let total = x - y; // le resultat sera total = 8
```

Pour ajouter ou soustraire un nombre d'une variable, vous pouvez utiliser les opérateurs += et -= :

```
let x = 10;

x -= 2;  //le resultat sera x = 8

x += 12; // il y a maintenant 20 cookies dans la boîte
```

Enfin, vous pouvez utiliser ++ ou -- pour ajouter ou soustraire 1 (incrément ou décrément) :

```
let numberOfLikes = 10;
numberOfLikes++;  // cela fait 11
numberOfLikes--; // et on revient à 10...qui n'a pas aimé mon article ?
```

Les opérations de multiplication et de division utilisent les opérateurs * et / :

```
let x = 20;
let y = 5;
let z = x * y;
let  a = x / y;
```

Comme pour l'addition et la soustraction, il existe aussi les opérateurs *= et /= pour multiplier ou diviser un nombre :

```
let x = 2;
x *= 6;  // x vaut maintenant 2*6 = 12;
x /= 3;  // x vaut maintenant 12/3 = 4;
```

Les constantes

Dans de nombreux programmes, certaines données ne seront pas modifiées pendant l'exécution du programme. C'est le cas par exemple du nom d'une entreprise, de la date de naissance d'un utilisateur, ou du nombre d'heures dans une journée. Pour s'assurer de ne pas réaffecter par inadvertance de nouvelles valeurs à ces données, vous allez utiliser des constantes.

Ce sont simplement des variables qui ne seront pas mutables. On donnera une valeur de départ et on ne pourra plus changer la valeur par la suite. Ainsi s'il y a une erreur de logique dans votre code changeant la valeur du variable (constante) qui ne devait pas changer, javascript retournera une erreur.

Par exemple :

```
const x = 20;
x = 30; // Retournera une erreur dans la console car on ne peut plus
changer sa valeur
```

1.5 Les conditions

Parfois, nous devons effectuer différentes actions en fonction de différentes conditions.

Pour ce faire, nous pouvons utiliser l' if instruction et l'opérateur conditionnel ?, également appelé opérateur « point d'interrogation ».

L'énoncé « if/else »

En JavaScript, si on utilise des boolean (bouléens, en français) simples pour les instructions if / else , la syntaxe se présente comme suit :

```
if (ecrire votre condition ici) {
// réaction à la valeur vraie de myBoolean
} else {
// réaction à la valeur faux de myBoolean
}
```

Donc, pour vérifier si vous etes un majeur, vous pouvez procéder comme suit en supposant que l'age minimal pour etre majeur est 18 ans:

```
// on decare votre une variale qui contient votre age
let age = 45;

// Ensuite on verifie si vous etes majeur ou non
if(age > 18){
    alert("vous etes majeur")
}else{
    alert("vous etes mineur")
}
```

Utilisez des expressions

Plutôt qu'une simple variable logique dans une condition if / else , vous pouvez aussi utiliser des expressions de comparaison, qui comparent des valeurs entre elles.

Les expressions de comparaison vous permettent de comparer deux valeurs par les opérateurs suivants :

- `<` inférieur à ;
- `<=` inférieur ou égal à ;
- `==` égal à ;
- `>=` supérieur ou égal à ;
- `>` supérieur à ;
- `!=` différent de.

Par exemple :

```
const x = 30;
const y = 25;

if (x < y) {
// x est plus petit
} else {
// x est plus grand
}
```

Vous pouvez aussi chaîner les instructions if / else pour réagir à des conditions potentielles multiples :

```
if (numberOfGuests == numberOfSeats) {
// tous les sièges sont occupés
} else if (numberOfGuests < numberOfSeats) {
// autoriser plus d'invités
} else {
// ne pas autoriser de nouveaux invités
}
```

Le chaînage d'instructions permet de prévoir différents résultats en fonction des différentes situations.

Les conditions multiple