Développer en JavaScript

Les bases du langage aux technologies avancées

Instructeur : Yousssouf SAGAF
sagafysf@gmail.com
02-03/07/2023 et 05-06/07/2023

1. Introduction:

JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation de Node.js. Il est un des trois langages fondamentaux des technologies web avec HTML et CSS.

1.1. Historique:

JavaScript a été créé en 1995 par Brendan Eich alors qu'il travaillait à Netscape Communications Corporation. Le langage a été initialement nommé Mocha, renommé LiveScript, puis JavaScript comme nous le connaissons aujourd'hui. Netscape a ensuite soumis JavaScript à ECMA International pour obtenir un standard, ce qui a conduit à la version officielle : ECMAScript.

1.2. ECMAScript :

ECMAScript est le standard sur lequel JavaScript est basé. Il a été créé pour standardiser JavaScript afin d'assurer l'interopérabilité entre les différentes plates-formes qui l'utilisent. Le nom ECMAScript vient de l'organisation qui a standardisé le langage, l'ECMA (*European Computer Manufacturers Association*), même si aujourd'hui elle se fait appeler Ecma pour éviter toute confusion géographique.

JavaScript est la mise en œuvre la plus connue de la norme ECMAScript, bien qu'il existe d'autres implémentations, comme JScript de Microsoft ou ActionScript utilisé par Flash.

ECMAScript définit la syntaxe et les fonctionnalités du langage, mais ne se préoccupe pas de la manière dont il interagit avec l'environnement d'exécution (le navigateur, dans notre cas). Par exemple, la manipulation du DOM ou l'envoi de requêtes HTTP ne sont pas spécifiées par ECMAScript, mais sont généralement fournies par l'environnement d'exécution.

Il y a eu plusieurs versions d'ECMAScript depuis sa première publication en 1997 :

- **ES1, ES2 et ES3** : Ces versions initiales ont établi le langage et ajouté des fonctionnalités telles que les exceptions try/catch.
- **ES4** : Cette version a été abandonnée en raison de désaccords sur les fonctionnalités à inclure.
- **ES5** : Publié en 2009, ES5 a ajouté de nombreuses fonctionnalités utiles comme le mode strict, les accessors et JSON.
- **ES6 / ES2015** : C'est une mise à jour majeure qui a ajouté de nombreuses nouvelles fonctionnalités comme les classes, les modules, les flèches, les promesses, etc.
- **ES7 / ES2016 à ES12 / ES2021** : Ces versions annuelles ont ajouté de nouvelles fonctionnalités au langage de manière incrémentielle.

Les navigateurs modernes supportent généralement ES6/ES2015 et les versions ultérieures, bien qu'il puisse y avoir quelques différences et incohérences. Pour s'assurer que le code JavaScript est compatible avec tous les navigateurs, les développeurs peuvent utiliser des transpilateurs comme Babel pour convertir leur code ES6+ en ES5.

Un autre point important est que JavaScript est un langage évolutif. Cela signifie que de nouvelles fonctionnalités sont constamment ajoutées et améliorées. C'est pourquoi il est crucial pour les développeurs JavaScript de se tenir au courant des dernières versions d'ECMAScript et de comprendre comment ces nouvelles fonctionnalités peuvent améliorer leur code.

1.3. JavaScript et HTML:

JavaScript et HTML interagissent étroitement sur le web. HTML est utilisé pour créer la structure de base de nos pages web, tandis que JavaScript est utilisé pour créer des interactions dynamiques sur ces pages.

Par exemple, vous pouvez avoir un élément HTML comme ceci:

```
<button id="myButton">Cliquez ici</button>
```

Et vous pouvez utiliser JavaScript pour écouter le clic de ce bouton et effectuer une action, comme ceci:

```
document.getElementById('myButton').addEventListener('click', function() {
    alert('Le bouton a été cliqué!');
});
```

Dans cet exemple, lorsque l'utilisateur clique sur le bouton, une boîte d'alerte apparaîtra avec le message "Le bouton a été cliqué!".

Il est également important de noter que JavaScript peut manipuler le DOM (Document Object Model). Le DOM est une interface de programmation pour les documents HTML et XML. Il représente la structure d'un document et permet aux programmes de manipuler le contenu, la structure et le style d'un document. En d'autres termes, le DOM est une représentation de votre page web qui peut être manipulée avec JavaScript.

1.4. Intégration du JS dans le HTML

L'intégration de JavaScript dans un code HTML est une étape cruciale pour tout développeur web. Il existe trois manières principales d'incorporer du JavaScript dans une page HTML : en ligne (inline), interne et externe.

- **Intégration en ligne de JavaScript**

L'intégration en ligne implique d'écrire du JavaScript directement dans les attributs des balises HTML. C'est une approche pratique pour de petits scripts spécifiques à un élément HTML.

- **Intégration interne de JavaScript**

L'intégration interne de JavaScript consiste à écrire le JavaScript dans une balise **<script>** dans le code HTML. Cela convient pour les scripts un peu plus longs qui sont spécifiques à une seule page.

- **Intégration externe de JavaScript**

L'intégration externe de JavaScript est la méthode la plus courante pour les grands projets. Le JavaScript est écrit dans un fichier séparé (généralement avec l'extension **.js**) et est ensuite lié à la page HTML en utilisant une balise **<script>**.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ma page</title>

    <!-- Intégration externe de JavaScript -->
    <script src="monscript.js"></script>
</head>

<body>
    <h1>Bienvenue sur ma page!</h1>

    <!-- Intégration interne de JavaScript -->
    <script>
    function afficheBonjour() {
        alert('Bonjour, monde!');
    }
    </script>

    <!-- Intégration en ligne de JavaScript -->
    <button onclick="afficheBonjour()">Cliquez-moi!</button>
</body>
</html>
```

Dans cet exemple, on intégre le JavaScript de trois manières différentes :

1. **Intégration externe** : Le fichier **monscript.js** est lié à la page HTML dans la balise **<head>**.
2. **Intégration interne** : La fonction **afficheBonjour()** est définie dans une balise **<script>** dans le **<body>** de la page HTML.
3. **Intégration en ligne** : Lorsqu'on clique sur le bouton, la fonction **afficheBonjour()** est exécutée, affichant une boîte d'alerte avec le message "Bonjour, monde!".

1.5. La gestion des packages avec npm

npm, qui signifie Node Package Manager, est un gestionnaire de paquets pour le langage de programmation JavaScript. Il est le moyen par défaut de gérer les packages dans l'environnement Node.js, qui est une plateforme permettant d'exécuter du JavaScript côté serveur.

Un package est une bibliothèque de code que vous pouvez inclure dans différents projets. Il peut s'agir d'une petite bibliothèque, telle que Lodash, qui fournit des utilitaires pour travailler avec des tableaux et des objets, ou d'une bibliothèque plus conséquente comme Express.js, qui est un framework pour la construction d'applications web en Node.js.

- **Installation de npm**

Etapes à suivre pour installer Node.js et npm :

1. **Allez sur le site web de Node.js** : Accédez à https://nodejs.org/.
2. **Téléchargez l'installateur** : Sur la page d'accueil de Node.js, vous verrez des options pour télécharger l'installateur. Il existe des versions pour Windows, MacOS et Linux. Vous pouvez choisir entre la dernière version LTS (Long Term Support), qui est la plus stable, ou la version actuelle, qui contient les dernières fonctionnalités.
3. **Installez Node.js et npm** : Une fois que vous avez téléchargé l'installateur, ouvrez-le et suivez les instructions pour installer Node.js et npm sur votre ordinateur.

Une fois l'installation terminée, vous pouvez vérifier que Node.js et npm ont été correctement installés en ouvrant une fenêtre de terminal et en tapant les commandes suivantes:

```
node -v
npm -v
```

Si ces deux commandes retournent une version, alors Node.js et npm sont correctement installés sur votre machine.

- **Installation de packages**

Pour installer un package, vous pouvez utiliser la commande **npm install**. Par exemple, pour installer Express.js, vous pouvez utiliser la commande suivante :

```
npm install express
```

Quand vous travaillez sur un projet JavaScript, il est courant d'avoir un fichier **package.json** dans votre répertoire de projet. Ce fichier sert à deux choses principales :

1. Il spécifie les dépendances de votre projet. Lorsque vous exécutez **npm install** sans arguments, npm regarde le fichier **package.json** et installe toutes les dépendances qui y sont listées.
2. Il peut contenir diverses "scripts" qui peuvent être exécutés avec **npm run**. Par exemple, vous pourriez avoir un script pour démarrer votre application, un autre pour la tester, etc.

Pour créer un fichier **package.json**, vous pouvez utiliser la commande **npm init**. Cela vous guidera à travers la création de ce fichier.

- **Mise à jour et désinstallation de packages**

Vous pouvez mettre à jour vos packages avec la commande **npm update**. Si vous voulez désinstaller un package, vous pouvez utiliser la commande **npm uninstall**.

- **npm et la communauté open source**

L'un des avantages de npm est qu'il donne accès à une immense bibliothèque de packages open source, qui ont été créés et maintenus par la communauté des développeurs JavaScript. Cela signifie que vous n'avez souvent pas besoin de réinventer la roue. Si vous avez besoin d'une certaine fonctionnalité, il est probable qu'il existe déjà un package npm que vous pouvez utiliser.

2. Les bases du langage javascript

2.1. Maîtriser les bases : variables, expressions, opérateurs et structures de contrôle

En JavaScript, vous pouvez déclarer des variables à l'aide des mots-clés **var**, **let** ou **const**. Les variables déclarées avec **var** ont une portée de fonction, tandis que celles déclarées avec **let** et **const** ont une portée de bloc. La différence entre **let** et **const** est que les variables déclarées avec **const** ne peuvent pas être réaffectées.

```
let name = 'John';
const age = 30;
```

Une expression est une combinaison de valeurs, de variables et d'opérateurs qui sont interprétés et donnent une valeur. Par exemple, **1 + 2** est une expression.

JavaScript a plusieurs types d'opérateurs :

- arithmétique (**+**, **-**, *****, **/**, **%**),
- d'affectation (**=**, **+=**, **-=**, etc.),
- de comparaison (**==**, **!=**, **===**, **!==**, **<**, **>**, **<=**, **>=**),
- logiques (**&&**, **||**, **!**), et plus encore.

Les structures de contrôle comprennent les instructions **if**, **switch**, **for**, **while**, **do while**, **break**, **continue**, et d'autres. Par exemple :

```
if (age > 18) {
    console.log('Vous êtes majeur.');
} else {
    console.log('Vous êtes mineur.');
}
```

2.2. Accéder aux ressources du navigateur

En JavaScript, vous pouvez accéder à de nombreuses ressources du navigateur à travers l'objet **window** et le Document Object Model (DOM).

Par exemple, vous pouvez manipuler des éléments HTML :

```
document.getElementById('myElement').innerText = 'Bonjour!';
```

Ou encore écouter des événements :

```
window.addEventListener('load', function() {
    console.log('La page a fini de charger.');
});
```

2.3. Définir des fonctions

Une fonction est un bloc de code conçu pour effectuer une tâche particulière. Une fonction est exécutée lorsque vous l'appelez (ou l'invoquez).

Voici comment définir une fonction en JavaScript :

```
function sayHello(name) {
    console.log('Bonjour ' + name + '!');
}
sayHello('John');
```

Il est également possible de définir des fonctions anonymes et des fonctions fléchées :

```
let sayHello = function(name) {
    console.log('Bonjour ' + name + '!');
}

// Ou avec une fonction fléchée :
let sayHello = (name) => {
    console.log('Bonjour ' + name + '!');
}
```

2.4. Les objets de JavaScript

JavaScript est un langage orienté objet, ce qui signifie que presque tout est un objet. Un objet est une collection de propriétés, et une propriété est une association entre un nom (ou une clé) et une valeur. Une valeur de propriété peut être une fonction, qui est alors considérée comme une méthode de l'objet.

Voici quelques-uns des objets intégrés que vous trouverez couramment en JavaScript :

- **String** : Il s'agit de l'objet global qui est le constructeur des chaînes de caractères, ou une séquence de caractères.

```
let str = new Strig("Ceci est une chaîne de caractères");
console.log(str);
```

- **Math** : Il s'agit d'un objet intégré qui possède des propriétés et des méthodes pour les constantes et fonctions mathématiques.

```
let number = Math.sqrt(16);
console.log(number);
```

- **Date :** Il s'agit de l'objet de date de JavaScript, qui peut être utilisé pour obtenir l'année, le mois et le jour actuels, et bien plus encore.

```
let date = new Date();
console.log(date);
```

- **Array :** Les tableaux de JavaScript sont des objets de haut niveau, de type liste.

```
let fruits = ['Apple', 'Banana', 'Mango'];
console.log(fruits[0]);
```

- **window :** C'est l'objet global en JavaScript dans le contexte d'un navigateur. Il représente la fenêtre du navigateur.

```
let w = window.innerWidth;
console.log(w);
```

- **navigator :** C'est un autre objet intégré en JavaScript qui peut être utilisé pour obtenir des informations sur le navigateur de l'utilisateur.

```
let browser = navigator.appName;
console.log(browser);
```

2.5. Les collections Set et Map

Les collections en JavaScript comme Set et Map sont également des types d'objets :

Set : C'est un type d'objet qui stocke des valeurs uniques, qu'il s'agisse de valeurs primitives ou de références d'objets.

```
let set = new Set();
set.add(1);
set.add(2);
set.add(3);
console.log(set);
```

Map : Les objets **Map** sont des collections simples de clés/valeurs. N'importe quelle valeur (objet et valeurs primitives) peut être utilisée comme une clé ou une valeur.

```
let map = new Map();
map.set('name', 'John');
map.set('age', 30);
console.log(map);
```

3. La gestion de l'interactivité des pages

3.1. Gérer les événements fenêtre : load et unload

L'événement **load** est déclenché lorsque tout le contenu de la page a été complètement chargé. Cela comprend les images externes, les styles et les scripts. Voici un exemple d'utilisation de l'événement **load** :

```
window.addEventListener('load', function() {
    console.log('La page est complètement chargée');
});
```

L'événement **unload** est déclenché juste avant qu'une page soit complètement déchargée. Cet événement peut être utilisé pour effectuer des opérations de nettoyage.

```
window.addEventListener('unload', function() {
    console.log('La page est sur le point d\'être déchargée');
});
```

load et **unload** peuvent être utilisés pour effectuer des actions spécifiques lors du chargement ou du déchargement de la page. Par exemple, vous pouvez initialiser des variables, charger des données supplémentaires, effectuer des appels de suivi ou de journalisation, ou nettoyer des ressources avant que la page ne se ferme.

Veuillez noter que l'événement **unload** peut être limité dans certaines situations en raison de restrictions de sécurité du navigateur pour empêcher les actions malveillantes. Par conséquent, il est préférable d'utiliser l'événement **beforeunload** si vous avez besoin d'exécuter un code avant que la page ne soit déchargée.

3.2. Gérer les événements clavier et souris : focus, blur, change, clic, mouseover, mouseout et submit

JavaScript permet également de réagir à de nombreux événements générés par l'utilisateur, qu'ils soient liés au clavier ou à la souris. Voici un exemple d'évènement sur un **button** :

```
<!DOCTYPE html>
<html>
<head><title>Exemple de bouton avec événement de clic</title></head>
<body>
  <button>Click Me</button>
  <script>
    let button = document.querySelector('button');
    button.addEventListener('click', function() {
        console.log('Le bouton a été cliqué');
    });
  </script>
</body>
</html>
```

Ce code sélectionne le premier bouton trouvé dans le document HTML à l'aide de **document.querySelector('button')**. Ensuite, il ajoute un écouteur d'événement de clic sur le bouton. Lorsque le bouton est cliqué, la fonction callback est exécutée, et dans cet exemple, elle affiche simplement le message "Le bouton a été cliqué" dans la console.

Voici quelques exemples de gestionnaires d'événements :

```
button.addEventListener('mouseover', function() {
    console.log('La souris est passée sur le bouton');
});

button.addEventListener('mouseout', function() {
    console.log('La souris est sortie du bouton');
});

button.addEventListener('focus', function() {
    console.log('Le bouton a le focus');
});

button.addEventListener('blur', function() {
    console.log('Le bouton a perdu le focus');
});

let input = document.querySelector('input');
input.addEventListener('change', function() {
    console.log('La valeur de l\'input a changé');
});

let form = document.querySelector('form');
form.addEventListener('submit', function(e) {
    e.preventDefault();
            console.log('Le formulaire a été soumis');
```

3.3. Déclencher par rapport au temps : setInterval et setTimeout

setTimeout permet de déclencher une action une fois après un certain temps, tandis que **setInterval** permet de déclencher une action régulièrement à intervalles réguliers.

```
setTimeout(function() {
    console.log('2 secondes se sont écoulées');
}, 2000);

setInterval(function() {
    console.log('Une autre seconde s\'est écoulée');
}, 1000);
```

Pour stopper l'exécution d'une fonction lancée par **setInterval**, vous pouvez utiliser la fonction **clearInterval** :

```
let intervalId = setInterval(function() {
    console.log('Une autre seconde s\'est écoulée');
}, 1000);

// Stoppe l'exécution après 5 secondes
setTimeout(function() {
    clearInterval(intervalId);
}, 5000);
```

3.4. Gérer les rollovers, les zooms, les diaporamas…

Les événements JavaScript peuvent être utilisés pour réaliser des effets plus complexes comme les rollovers, les zooms ou les diaporamas. Par exemple, voici comment vous pourriez réaliser un effet de rollover en utilisant les événements **mouseover** et **mouseout** :

```
<!DOCTYPE html
<html>
<head>
  <title>Exemple de changement d'image au survol</title>
</head>
<body>
  <img src="image-originale.jpg" alt="Image originale">

  <script>
    let image = document.querySelector('img');
    image.addEventListener('mouseover', function() {
        image.src = 'image-survol.jpg';
    });
    image.addEventListener('mouseout', function() {
        image.src = 'image-originale.jpg';
    });
  </script>
</body>
</html>
```

Cet exemple suppose que vous avez deux images : '*image-originale.jpg*' et '*image-survol.jpg*'. Lorsque l'utilisateur passe la souris sur l'image, l'image de survol est affichée. Lorsque l'utilisateur sort la souris de l'image, l'image originale est réaffichée.

3.5. Gestion du zoom

Pour un effet de zoom sur une image, on peut utiliser l'événement **mouseover** et manipuler la largeur et la hauteur de l'image. Voyons un exemple simple :

HTML :

```
<img id="myImage" src="image.jpg" style="width:200px;height:200px;">
```

JS :

```
let myImage = document.querySelector('#myImage');

myImage.addEventListener('mouseover', function() {
    myImage.style.width = '400px';
    myImage.style.height = '400px';
});

myImage.addEventListener('mouseout', function() {
    myImage.style.width = '200px';
    myImage.style.height = '200px';
});
```

Dans cet exemple, lorsque vous survolez l'image avec votre souris, sa taille change pour 400x400 pixels, et quand votre souris sort de l'image, sa taille revient à 200x200 pixels.

3.6. Gestion d'un diaporama

Pour gérer un diaporama, vous pouvez utiliser un tableau d'images et l'objet **setInterval** pour changer l'image affichée à intervalles réguliers.

HTML :

```
<img id="diaporama" src="image1.jpg">
```

JS :

```
let diaporama = document.querySelector('#diaporama');
let images = ['image1.jpg', 'image2.jpg', 'image3.jpg'];
let index = 0;

setInterval(function() {
    index++;
    if (index === images.length) {
        index = 0;
    }
    diaporama.src = images[index];
}, 2000);
```

Dans cet exemple, l'image du diaporama change toutes les 2 secondes. Lorsque toutes les images ont été affichées, le diaporama revient au début.

3.7. Gérer les interactions avec addEventListener

La méthode **addEventListener** est une méthode fondamentale pour gérer les événements en JavaScript. Elle permet d'ajouter une fonction à exécuter lorsque l'événement spécifié est déclenché.

```
let button = document.querySelector('button');

button.addEventListener('click', function() {
    console.log('Le bouton a été cliqué');
});
```

Dans cet exemple, la fonction passée à **addEventListener** sera exécutée chaque fois que l'utilisateur clique sur le bouton. Vous pouvez ajouter autant de gestionnaires d'événements que vous le souhaitez à un même élément pour un même événement. Ils seront exécutés dans l'ordre de leur ajout.

3.8. Accéder aux éléments du document HTML via DOM

On peut accéder aux éléments HTML à l'aide de différentes méthodes, parmi lesquelles : **getElementById**, **getElementsByClassName**, **getElementsByTagName**, **querySelector** et **querySelectorAll**.

Exemple :

```
<!DOCTYPE html>
<html>
<head> <title>Illustration HTML et CSS</title> <style>
   .myClass {
      background-color: lightblue;
      padding: 10px;
      margin-bottom: 10px; }
   #myId {
      font-weight: bold; }
  </style>
</head>
<body>
  <div id="myId" class="myClass">Bonjour le monde !</div>
  <div class="myClass">Bonjour encore !</div>
</body>
</html>
```

JS:

```
let elementById = document.getElementById('myId');
let elementsByClassName = document.getElementsByClassName('myClass');
let elementsByTagName = document.getElementsByTagName('div');
let elementByQuerySelector = document.querySelector('.myClass');
let elementsByQuerySelectorAll = document.querySelectorAll('.myClass');
```

Ces méthodes de sélection d'éléments en JavaScript vous permettent de cibler spécifiquement des éléments HTML dans votre document en utilisant des sélecteurs basés sur l'**id**, la classe ou le type d'élément.

- **document.getElementById('myId')** sélectionne un élément par son **id** et le stocke dans **elementById**.
- **document.getElementsByClassName('myClass')** sélectionne tous les éléments par leur classe (ici **myClass**) et les stocke dans **elementsByClassName**.
- **document.getElementsByTagName('div')** sélectionne tous les éléments ** ** et les stocke dans **elementsByTagName**.
- **document.querySelector('.myClass')** sélectionne le premier élément correspondant au sélecteur CSS et le stocke dans **elementByQuerySelector**.
- **document.querySelectorAll('.myClass')** sélectionne tous les éléments correspondants au sélecteur CSS et les stocke dans **elementsByQuerySelectorAll**.

3.9. Modifier, masquer et afficher des objets HTML

Pour modifier le contenu HTML d'un élément, on peut utiliser la propriété **innerHTML**. Pour masquer un élément, on peut utiliser la propriété **style.display**.

Exemple de code :

HTML:

```
<div id="myId">Hello World!</div>
```

JS:

```
let element = document.getElementById('myId');
// Modifier
element.innerHTML = 'Nouveau contenu';

// Masquer
element.style.display = 'none';

// Afficher
element.style.display = 'block';
```

3.10. Modifier les attributs des éléments d'interface (police, couleur…)

Pour modifier les attributs d'un élément, on peut utiliser la méthode **setAttribute**. Pour modifier le style d'un élément, on peut utiliser la propriété **style**.

Exemple de code:

HTML:

```
<div id="myId">Hello World!</div>
```

JS:

```
let element = document.getElementById('myId');

// Modifier un attribut
element.setAttribute('align', 'center');

// Modifier le style
element.style.color = 'red';
element.style.fontFamily = 'Arial';
```

3.11. Déplacer du texte, des images

Pour déplacer du texte ou des images, on peut modifier la position d'un élément à l'aide de la propriété **style**.

Exemple de code:

HTML:

```
<div id="myId">Hello World!</div>
```

JS:

```
let element = document.getElementById('myId');

// Déplacer l'élément
element.style.position = 'absolute';
element.style.top = '100px';
element.style.left = '200px';
```

3.12. Gérer un menu dynamiquement

Pour gérer un menu dynamiquement, on peut utiliser des événements pour montrer ou cacher des éléments.

Exemple de code:

HTML:

```
<button id="boutonMenu">Toggle Menu</button>
<div id="menu" style="display: none;">
    <a href="#">Link 1</a>
    <a href="#">Link 2</a>
    <a href="#">Link 3</a>
</div>
```

JS:

```
let bouton = document.getElementById('boutonMenu');
let menu = document.getElementById('menu');

bouton.addEventListener('click', function() {
```

```
    if (menu.style.display === 'none') {
        menu.style.display = 'block';
    } else {
        menu.style.display = 'none';
    }
});
```

3.13. La manipulation de la page avec querySelectorAll

La méthode **querySelectorAll** renvoie tous les éléments qui correspondent à un sélecteur CSS donné.

Exemple de code:

HTML:

```
<div class="maClasse">Hello World!</div>
<div class="maClasse">Hello Again!</div>
```

JS:

```
let elements = document.querySelectorAll('.maClasse');
elements.forEach(function(element) {
    element.style.color = 'red';
});
```

4. Gestion des données de formulaires

La gestion des données de formulaire en JavaScript consiste à manipuler divers types d'éléments de formulaire, y compris les champs de texte, les cases à cocher, les boutons radio, les boutons et les listes déroulantes.

4.1. Gérer les objets Form, Text, Checkbox, Radio, Button

Les objets de formulaire peuvent être accédés et manipulés en utilisant JavaScript. Par exemple, vous pouvez obtenir la valeur d'un champ de texte, vérifier si une case à cocher est cochée, etc.

HTML

```
<form id="myForm">
    <input type="text" id="myText" value="Hello World">
    <input type="checkbox" id="myCheckbox">
    <input type="radio" id="myRadio">
    <input type="button" id="myButton" value="Click Me">
</form>
```

JS

```
<script>
    var form = document.getElementById('myForm');
    var text = document.getElementById('myText').value; // get the value of the text field
    var checkbox = document.getElementById('myCheckbox').checked; // check if the checkbox is checked
    var radio = document.getElementById('myRadio').checked; // check if the radio button is selected
</script>
```

4.2. Utiliser les listes : Select, Option

Vous pouvez également manipuler les listes déroulantes (éléments select) en JavaScript. Par exemple, vous pouvez obtenir l'option actuellement sélectionnée, changer la sélection, etc.

Exemple:

HTML

```
<select id="mySelect">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```

JS

```
<script>
    var select = document.getElementById('mySelect');
    var selectedOption = select.options[select.selectedIndex].value; // get the value of the selected
option
</script>
```

4.3. Utiliser les expressions régulières

Les expressions régulières peuvent être utilisées pour valider et manipuler les données du formulaire. Par exemple, vous pouvez utiliser une expression régulière pour vérifier si le contenu d'un champ de texte correspond à un modèle spécifique.

HTML

```
<form id="myForm">
    <input type="text" id="myText" value="Hello World">
</form>
```

JS

```
<script>
    var text = document.getElementById('myText').value;
    var pattern = /World/; // regular expression pattern
    var result = pattern.test(text); // check if the text matches the pattern
    console.log(result); // outputs: true
</script>
```

Dans cet exemple, l'expression régulière **/World/** vérifie si le texte contient le mot "World". La méthode **test()** est une méthode intégrée de l'objet RegExp en JavaScript. Elle est utilisée pour tester s'il y a une correspondance entre une expression régulière et une chaîne de caractères spécifiée. La méthode renvoie **true** si une correspondance est trouvée et **false** dans le cas contraire.

5. Introduction à la programmation AJAX

AJAX signifie Asynchronous JavaScript and XML. C'est un ensemble de technologies Web utilisées pour créer des applications Web interactives.

5.1. Les apports d'AJAX

AJAX permet aux pages Web de mettre à jour de manière asynchrone en échangeant des données avec un serveur Web. Cela signifie qu'il est possible de mettre à jour certaines parties d'une page Web sans recharger toute la page.

Les avantages de l'utilisation d'AJAX sont nombreux :

- Améliore l'expérience utilisateur en rendant les pages plus réactives.
- Réduit le trafic entre le client et le serveur, car seules certaines parties de la page doivent être mises à jour.
- Permet des mises à jour de la page Web en arrière-plan, sans interaction de l'utilisateur.

5.2. L'objet XMLHttpRequest

L'objet XMLHttpRequest est l'épine dorsale de toute interaction AJAX avec le serveur. Il a des méthodes pour l'ouverture d'une connexion (**open()**), l'envoi d'une requête (**send()**), et la définition d'une fonction à exécuter lorsque la réponse est reçue (**onreadystatechange**). Il a également des propriétés pour vérifier l'état de la requête (**readyState**) et pour lire la réponse du serveur (**responseText** ou **responseXML**).

Cet exemple montre comment créer un nouvel objet XMLHttpRequest, envoyer une requête GET à un serveur et afficher la réponse:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", 'https://api.example.com/data', true);
xhr.onreadystatechange = function () {
  if (xhr.readyState == 4 && xhr.status == 200)
    console.log(xhr.responseText);
}
xhr.send();
```

5.3. Les promesses (Promise)

Une promesse est un objet JavaScript qui lie l'aboutissement d'un travail asynchrone, qui est généralement une requête AJAX. Une promesse peut être dans l'un des trois états : en attente (pending), résolue (fullfiled) ou rejetté (rejected). Voici un exemple d'utilisation d'une promesse pour exécuter une opération asynchrone:

```
var promise = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('Success!');
  }, 250);
});

promise.then(function(value) {
  console.log(value);
});
```

5.4. La structuration de données en JSON

JSON (JavaScript Object Notation) est un format de données standard pour le transfert de données entre le client et le serveur dans les applications Web. Il est léger, facile à comprendre et à écrire, et il est intégré nativement à JavaScript, ce qui signifie que vous n'avez pas besoin d'une bibliothèque externe pour le manipuler. JSON est souvent utilisé pour encapsuler les données qui sont envoyées au serveur via AJAX, ou les données qui sont renvoyées par le serveur. Dans le contexte d'AJAX, vous utiliseriez généralement les méthodes intégrées **JSON.stringify()** pour convertir un objet JavaScript en une chaîne JSON, et **JSON.parse()** pour convertir une chaîne JSON en un objet JavaScript.

Voici un exemple de conversion d'un objet JavaScript en une chaîne JSON, puis de sa reconversion en un objet JavaScript :

```
var obj = {name: "John", age: 30, city: "New York"};
var myJSON = JSON.stringify(obj);
console.log(myJSON);

var myObj = JSON.parse(myJSON);
console.log(myObj);
```

5.5. APIs pour les applications

La gestion de l'historique du navigateur permet d'interagir avec l'historique de session du navigateur grâce à l'API de l'historique. Il existe plusieurs méthodes pour cela, dont :

- **window.history.back()** : permet de revenir à la page précédente.
- **window.history.forward()** : permet de se déplacer vers la page suivante.
- **window.history.go(n)** : permet de se déplacer de 'n' entrées dans l'historique.
- **window.history.pushState()** : permet d'ajouter une entrée à l'historique.

Exemple de code:

```
// Utilise l'API History pour naviguer à la page précédente de l'historique du navigateur.
window.history.back();

// Ajoute une nouvelle entrée à l'historique de navigation.
// Le premier argument est un objet qui représente l'état, le deuxième argument est le titre de la nouvelle entrée (non vraiment utilisé à ce jour),
// et le troisième argument est l'URL de la nouvelle entrée.
window.history.pushState({page: 1}, "title 1", "?page=1");
```

5.6. Stockage local :

Le stockage local fait partie de l'API de stockage Web et permet de stocker des données dans le navigateur de l'utilisateur de manière persistante. Il est similaire aux cookies, mais peut contenir plus de données et ne sont pas envoyées au serveur à chaque requête HTTP.

- **localStorage.setItem('key', 'value')** : permet de stocker une donnée.
- **localStorage.getItem('key')** : permet de récupérer une donnée.
- **localStorage.removeItem('key')** : permet de supprimer une donnée.
- **localStorage.clear()** : permet de vider tout le stockage local.

Exemple de code:

```
// Utilise l'API localStorage pour stocker une paire clé-valeur dans le stockage Web du navigateur.
// 'name' est la clé et 'John' est la valeur.
localStorage.setItem('name', 'John');

// Récupère la valeur de 'name' du stockage Web et l'affiche dans la console.
console.log(localStorage.getItem('name'));
// Supprime l'élément 'name' du stockage Web.
localStorage.removeItem('name');
```

5.7. WebSockets :

Les WebSockets sont une technologie avancée qui permet d'ouvrir une session de communication interactive entre le navigateur de l'utilisateur et un serveur. Avec cette API, vous pouvez envoyer des

messages à un serveur et recevoir des réponses orientées événement sans avoir à interroger le serveur pour une réponse.

Exemple de code :

```
// Crée un nouveau WebSocket qui se connecte à un serveur WebSocket à l'URL spécifiée.
var socket = new WebSocket('ws://localhost:8080');

// Écoute l'événement 'onopen', qui est déclenché lorsque la connexion est établie avec succès.
// Envoie un message au serveur une fois la connexion ouverte.
socket.onopen = function(event) {
  socket.send('Hello Server!');
};

// Écoute l'événement 'onmessage', qui est déclenché lorsque le serveur envoie un message.
// Affiche le message du serveur dans la console.
socket.onmessage = function(event) {
  console.log('Server: ' + event.data);
};

// Écoute l'événement 'onclose', qui est déclenché lorsque la connexion est fermée.
// Affiche un message dans la console lorsque la connexion est fermée.
socket.onclose = function(event) {
  console.log('Connection closed');
};
```

6. Les modules ES6

Les modules ES6 sont une fonctionnalité du langage JavaScript qui permet d'organiser et de structurer notre code en utilisant des modules autonomes. L'un des avantages des modules ES6 est la gestion efficace de l'espace de noms, ce qui évite les collisions de noms et les écrasements d'événements.

6.1. Problématique : collision de noms, écrasement d'événements

Lorsque on travaille avec du code JavaScript, il peut arriver que l'on utilise des noms de variables, de fonctions ou d'événements qui se répètent dans différentes parties de notre code. Cela peut entraîner des conflits, des écrasements involontaires et des erreurs difficiles à diagnostiquer. Par exemple, si on a deux fichiers JavaScript distincts qui utilisent tous les deux une fonction appelée "**calculerMoyenne**", cela pourrait entraîner des problèmes lorsqu'ils sont tous les deux chargés dans la même page HTML.

6.2. Espace de noms

Les modules ES6 résolvent ce problème en fournissant un mécanisme pour créer des espaces de noms distincts. Chaque module a son propre espace de noms isolé, ce qui signifie que les noms de variables, de fonctions et d'événements utilisés à l'intérieur d'un module ne seront pas en conflit avec les noms utilisés dans d'autres modules. Chaque module peut avoir des variables et des fonctions portant le même nom sans provoquer de collisions. Par exemple, si on a deux modules A et B, et que les deux modules ont une fonction "**calculerMoyenne**", ces deux fonctions peuvent coexister sans conflit, car elles sont définies dans des espaces de noms séparés.

Cela permet d'organiser le code de manière modulaire, en découpant l'application en modules indépendants et réutilisables, tout en évitant les conflits de noms et les écrasements involontaires.

Pour utiliser les modules ES6, on doit exporter les éléments (**variables**, **fonctions**, **classes**) que l'on souhaite rendre accessibles depuis d'autres modules en utilisant le mot-clé **export**, et les importer dans d'autres modules à l'aide du mot-clé **import**.

Par exemple, si on a un module A qui définit une fonction **calculerMoyenne** et un module B qui souhaite utiliser cette fonction, on peut exporter la fonction **calculerMoyenne** dans le module A en utilisant **export**, puis l'importer dans le module B en utilisant **import**.

Voici un exemple de syntaxe pour l'exportation et l'importation d'une fonction :

Dans le module A :

```
// module A
export function calculerMoyenne(notes) {
  // Code pour calculer la moyenne des notes
}
```

Dans le module B :

```
// module B

import { calculerMoyenne } from './moduleA.js';

// Utilisation de la fonction calculerMoyenne
const notes = [15, 18, 16];
const moyenne = calculerMoyenne(notes);
console.log(moyenne);
```

7. La programmation orientée objet en ES6

La programmation orientée objet (POO) est un paradigme de programmation qui permet d'organiser notre code en utilisant des concepts tels que les classes, l'héritage, les objets et les méthodes. En ES6, on dispose de fonctionnalités spécifiques pour mettre en œuvre la POO de manière plus concise et plus claire.

7.1. Les classes et héritages

Les classes sont des modèles pour la création d'objets. Elles définissent les propriétés et les comportements communs à un ensemble d'objets. Pour définir une classe en ES6, on utilise le mot-clé **class** suivi du nom de la classe. Les objets créés à partir d'une classe sont appelés des instances.
L'héritage permet à une classe d'hériter des propriétés et des méthodes d'une autre classe. On utilise le mot-clé **extends** pour créer une classe enfant qui hérite des propriétés et des méthodes de la classe parent.

Exemple de code :

```
class Animal {
  constructor(nom) {
    this.nom = nom;
  }

  direBonjour() {
    console.log(`Bonjour, je suis ${this.nom}.`);
  }
}

class Chien extends Animal {
  aboyer() {
    console.log("Wouaf !");
  }
}
```

```
const monChien = new Chien("Max");
monChien.direBonjour(); // Bonjour, je suis Max.
monChien.aboyer(); // Wouaf !
```

7.2. Le contexte

Le contexte en ES6 se réfère à la valeur spéciale **this** qui fait référence à l'objet courant sur lequel une méthode est appelée. Dans le contexte d'une classe, **this** fait référence à l'instance de l'objet créée à partir de cette classe. Dans une méthode d'une classe, **this** est utilisé pour accéder aux propriétés et aux méthodes de l'instance en cours.

Exemple de code :

```
class Personne {
  constructor(nom) {
    this.nom = nom;
}

  sePresenter() {
    console.log(`Bonjour, je suis ${this.nom}.`);
  }
}

const personne1 = new Personne("Alice");
personne1.sePresenter();
```

7.3. Getter et setter

Les getters et les setters sont des méthodes spéciales qui nous permettent d'accéder et de modifier les propriétés d'un objet de manière contrôlée. Les getters sont utilisés pour obtenir la valeur d'une propriété, tandis que les setters sont utilisés pour définir la valeur d'une propriété. En ES6, on peut utiliser les mots-clés **get** et **set** pour définir des getters et des setters dans une classe.

- Les getters et les setters permettent de contrôler l'accès aux propriétés d'un objet.
- Un getter est une méthode qui renvoie la valeur d'une propriété.
- Un setter est une méthode qui modifie la valeur d'une propriété.

Exemple de code :

```
class CompteBancaire {

  constructor(solde) {
    this._solde = solde;
}

  get solde() {
    return this._solde;
}

  set solde(nouveauSolde) {
    if (nouveauSolde >= 0) {
      this._solde = nouveauSolde;
    }
  }
}

const compte = new CompteBancaire(1000);
console.log(compte.solde); // 1000
```

```
compte.solde = 2000;
console.log(compte.solde); // 2000
compte.solde = -500; // Le solde ne peut pas être négatif
console.log(compte.solde); // 200
```

7.4. Les méthodes statiques

- Une méthode statique est une méthode qui est attachée à la classe elle-même plutôt qu'aux instances de la classe.
- Les méthodes statiques sont appelées sur la classe elle-même, pas sur les instances.

Exemple de code :

```
class MathUtils {

  static carre(nombre) {
    return nombre * nombre;
  }
}

console.log(MathUtils.carre(5)); // 25
```

8. La présentation de la librairie jQuery

Le but de jQuery est de faciliter l'utilisation de JavaScript sur votre site web. jQuery est une bibliothèque JavaScript légère, qui se présente sous le slogan "Écrire moins, en faire plus". Le but de jQuery est de faciliter l'utilisation de JavaScript sur votre site web.

jQuery simplifie de nombreuses tâches courantes qui nécessitent normalement plusieurs lignes de code JavaScript, en les encapsulant dans des méthodes que vous pouvez appeler en une seule ligne de code.

jQuery simplifie également de nombreux aspects complexes de JavaScript, tels que les appels AJAX et la manipulation du DOM. La bibliothèque jQuery comprend les fonctionnalités suivantes :

- Manipulation HTML/DOM
- Manipulation CSS
- Méthodes d'événements HTML
- Effets et animations
- AJAX

Tips : De plus, jQuery dispose de nombreux plugins pour presque toutes les tâches que vous pouvez rencontrer.

Il existe plusieurs façons de commencer à utiliser jQuery sur votre site web. Vous pouvez :

- Télécharger la bibliothèque jQuery depuis *jQuery.com*.
- Inclure jQuery à partir d'un CDN, tel que Google.

Il existe deux versions de jQuery disponibles en téléchargement :

1. **Version de production** : cette version est destinée à votre site web en direct, car elle a été minifiée et compressée.
2. **Version de développement** : cette version est destinée aux tests et au développement (code non compressé et lisible). Les deux versions peuvent être téléchargées depuis jQuery.com.

La sélection et manipulation du DOM

La sélection et manipulation du DOM :

- jQuery offre une syntaxe simple et concise pour sélectionner et manipuler les éléments du DOM.
- La sélection des éléments se fait en utilisant les sélecteurs CSS, ce qui permet de cibler facilement les éléments souhaités.
- Une fois les éléments sélectionnés, vous pouvez les manipuler en utilisant les méthodes fournies par jQuery, telles que **addClass**, **removeClass**, **attr**, **text**, etc.

Exemple de code :

```
// Sélectionner tous les paragraphes et ajouter une classe 'highlight'
$('p').addClass('highlight');

// Modifier le contenu textuel d'un élément avec l'id 'myElement'
$('#myElement').text('Nouveau contenu');
```

Les événements

- jQuery facilite la gestion des événements en fournissant des méthodes pour attacher des gestionnaires d'événements à des éléments du DOM.
- Vous pouvez attacher des gestionnaires d'événements à des événements courants tels que **click**, **hover**, **submit**, **keyup**, etc.
- Les gestionnaires d'événements peuvent être attachés directement aux éléments sélectionnés ou utilisés avec des délégués d'événements pour les éléments futurs ajoutés dynamiquement.

Exemple de code :

```
// Attacher un gestionnaire d'événement au clic sur un bouton
$('#myButton').click(function() {
  alert('Le bouton a été cliqué !');
});

// Utiliser un délégué d'événement pour les éléments futurs ajoutés
$('#myContainer').on('click', 'button', function() {
  alert('Un bouton a été cliqué dans le conteneur !');
});
```

AJAX avec jQuery

- jQuery facilite l'envoi et la réception de requêtes AJAX.
- Vous pouvez effectuer des requêtes HTTP GET, POST, PUT, DELETE, etc. en utilisant les méthodes fournies par jQuery, telles que **$.ajax**, **$.get**, **$.post**, etc.
- Vous pouvez également manipuler les données renvoyées par la requête AJAX et mettre à jour dynamiquement votre page en fonction des résultats.

Exemple de code :

```
// Effectuer une requête GET pour récupérer des données JSON
$.get('https://api.example.com/data', function(data) {
  // Manipuler les données renvoyées
  console.log(data);
});
```