Javascript : le DOM

Sommaire

8. Exercices

9. Conclusion et importance de la manipulation du DOM dans le développement web.

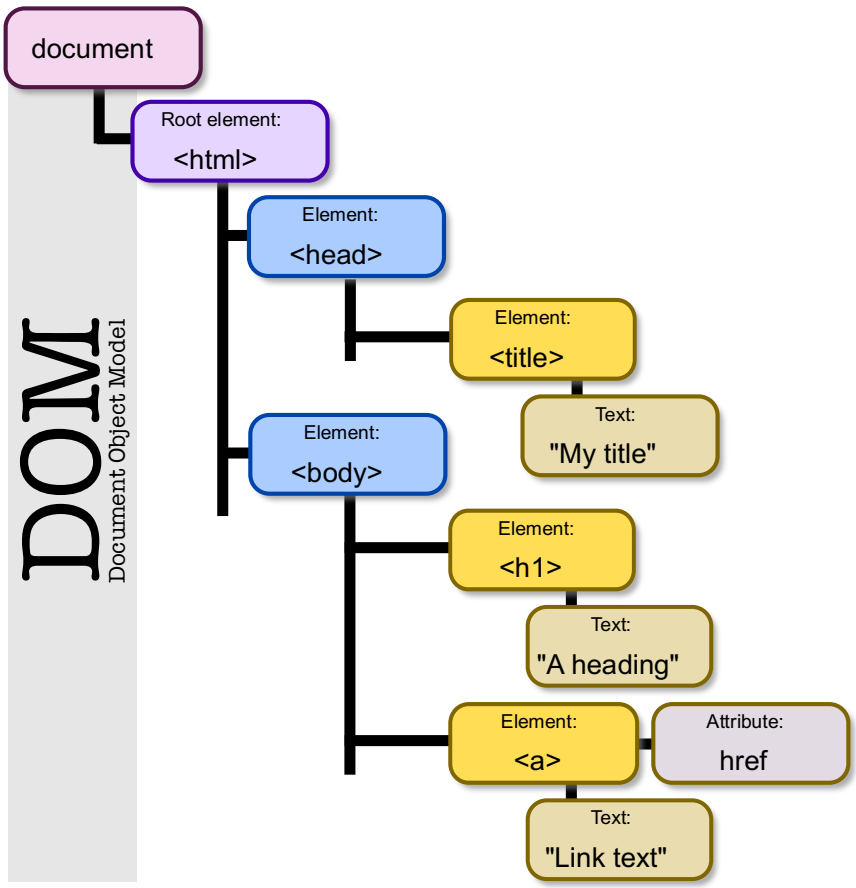
1. Introduction au DOM

Qu'est-ce que le DOM ?

Le Document Object Model (DOM) est une représentation hiérarchique en arbre de la structure d'une page web. Il s'agit d'une interface de programmation qui permet aux langages de programmation, notamment JavaScript, de manipuler et d'interagir avec les éléments d'une page web. Le DOM représente chaque élément de la page (comme les balises HTML) en tant qu'objet dans une structure arborescente, où chaque objet correspond à un nœud dans cet arbre.

Structure de l'Arbre DOM

Le DOM organise les éléments HTML en une structure en arbre, où chaque élément est un nœud. Les nœuds sont organisés en une hiérarchie, avec le nœud racine représentant le document entier et les nœuds enfants représentant les éléments HTML.



Relation entre le DOM, HTML et CSS

Le HTML définit la structure et le contenu d'une page web, tandis que le CSS gère son style et sa mise en forme. Le DOM agit comme une interface de programmation qui permet de modifier dynamiquement le contenu et le style d'une page en utilisant JavaScript.

Rôle du JavaScript dans la Manipulation du DOM

JavaScript permet d'accéder aux éléments du DOM, de les modifier, d'ajouter ou de supprimer des éléments, et de réagir aux interactions de l'utilisateur en ajoutant des écouteurs d'événements. Cela permet de créer des pages web dynamiques et interactives.

Voici un exemple concret en code qui permet de modifier dynamiquement le contenu et le style de la page en utilisant JavaScript.

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemple DOM</title>
  <link rel="stylesheet" type="text/css" href="dom.css">
</head>
<body>
  <header>
    <h1 id="main-heading">Bienvenue sur mon site</h1>
  </header>
  <main>
    <p>Ceci est un exemple de page web.</p>
    <p>Le bouton va changer le texte du H1 et sa couleur lorsqu'on va
cliquer dessus</p>
    <button id="change-text">Changer le texte</button>
  </main>
  <script src="dom.js"></script>
</body>
</html>
```

CSS :

```
body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
}

header {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 10px;
}

main {
```

```
    padding: 20px;
}
```

JS :

```
const heading = document.getElementById("main-heading");
const button = document.getElementById("change-text");

button.addEventListener("click", () => {
  heading.textContent = "Nouveau Titre";
  heading.style.color = "red";
});
```

Le fichier JavaScript utilise le DOM pour sélectionner l'élément du titre (<h1>) par son ID (précisé dans le html) en utilisant la fonction getElementById() de l'objet document et on stocke cet élément dans une variable (ici heading). L'objet document représentant votre document html. Nous faisons la même chose avec le bouton. Et pour ajouter un écouteur d'événements, nous allons utiliser addEventListener. Nous reviendrons sur cette notion d'écouteur d'evenements dans un futur chapitre mais voici une brève explication de ce que fait la fonction d'écoute addEventListener().

La méthode addEventListener est une fonction JavaScript qui permet de créer un lien entre un élément du DOM (comme un bouton, une image, un lien, etc.) et une fonction JavaScript spécifiée. C'est une fonction qui va ajouter un évènement à l'écoute d'un autre élément. Cette fonction sera exécutée chaque fois qu'un événement spécifié se produit sur l'élément cible. En d'autres termes, addEventListener permet de réagir aux interactions de l'utilisateur ou à d'autres événements sur la page (comme le clic, le hover, la sélection d'un élément dans une liste, cocher une checkbox etc…)

Et donc nous sélectionnons le bouton par sa variable button et nous utilisons la fonction d'écoute addEventListener qui modifie le contenu du titre et son style lorsque le bouton est cliqué.

2. Sélection des Éléments

Méthodes de Sélection du DOM

Les méthodes de sélection du DOM permettent de cibler et de récupérer des éléments spécifiques à partir de la structure de l'arbre DOM. Voici les principales méthodes de sélection du DOM en JavaScript :

- getElementById: Sélectionne un élément par son attribut id.

```
const element = document.getElementById("elementId");
```

- getElementsByClassName: Sélectionne les éléments par leur classe.

```
const elements = document.getElementsByClassName("className");
```

- getElementsByTagName: Sélectionne les éléments par leur balise.

```
const elements = document.getElementsByTagName("tagName");
```

- querySelector: Sélectionne le premier élément correspondant au sélecteur CSS.

```
const element = document.querySelector("selector");
```

- querySelectorAll: Sélectionne tous les éléments correspondant au sélecteur CSS.

```
const elements = document.querySelectorAll("selector");
```

- hasFocus : La méthode document.hasFocus() est utilisée pour déterminer si le document actuel a le focus. C'est un boolean qui sera renvoyé.

```
if (document.hasFocus()) {
    console.log("Le document a le focus.");
} else {
    console.log("Le document n'a pas le focus.");
}
```

Cette méthode renverra true si le document actuel a le focus et false s'il ne l'a pas. Vous pouvez l'utiliser pour déclencher des actions ou des conditions en fonction de si le document a le focus ou non, par exemple pour gérer les interactions lorsque l'utilisateur change de fenêtre ou d'onglet dans le navigateur.

Voici un exemple de code HTML qui correspond aux exemples de sélection du DOM :

```
<!DOCTYPE html>
<html>
<head>
  <title>Sélection du DOM</title>
</head>
<body>
  <h1 id="title">Titre Principal</h1>
  <div class="container">
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
  </div>
  <ul class="list">
    <li>Élément 1</li>
    <li>Élément 2</li>
    <li>Élément 3</li>
  </ul>
  <button id="myButton">Cliquez-moi</button>
  <script src="dom.js"></script>
</body>
```

```
</html>
```

JS:

```
// Sélection par ID
const titleElement = document.getElementById("title");
console.log("Sélection par ID:", titleElement);

// Sélection par classe
const containerElements = document.getElementsByClassName("container");
console.log("Sélection par classe:", containerElements);

// Sélection par balise
const listItems = document.getElementsByTagName("li");
console.log("Sélection par balise:", listItems);

// Sélection avec querySelector
const firstItem = document.querySelector(".list li");
console.log("Premier élément de la liste:", firstItem.textContent);

// Sélection avec querySelectorAll
const paragraphs = document.querySelectorAll(".container p");
console.log("Sélection avec querySelectorAll:", paragraphs);
```

Parcours des Nœuds DOM

Le parcours des nœuds DOM vous permet de naviguer à travers la structure arborescente des éléments HTML dans le Document Object Model (DOM). Il existe plusieurs propriétés et méthodes qui facilitent ce parcours. Voici comment vous pouvez parcourir les nœuds DOM :

- parentNode : Cette propriété renvoie l'élément parent direct d'un nœud.

```
const childElement = document.getElementById("child");
const parentElement = childElement.parentNode;
console.log(parentElement);
```

- childNodes : Cette propriété renvoie une liste des nœuds enfants d'un élément, y compris les nœuds de texte et les nœuds commentaires.

```
const parentElement = document.getElementById("parent");
const children = parentElement.childNodes;
console.log(children);
```

- firstChild : Cette propriété renvoie le premier nœud enfant d'un élément.

```
const parentElement = document.getElementById("parent");
const firstChild = parentElement.firstChild;
console.log(firstChild);
```

- lastChild : Cette propriété renvoie le dernier nœud enfant d'un élément.

```
const parentElement = document.getElementById("parent");
const lastChild = parentElement.lastChild;
console.log(lastChild);
```

- nextSibling : Cette propriété renvoie le nœud suivant qui est un enfant du même parent.

```
const firstChild = document.getElementById("first");
const nextSibling = firstChild.nextSibling;
console.log(nextSibling);
```

- previousSibling : Cette propriété renvoie le nœud précédent qui est un enfant du même parent.

```
const secondChild = document.getElementById("second");
const previousSibling = secondChild.previousSibling;
console.log(previousSibling);
```

- nextElementSibling : Cette propriété renvoie le prochain élément (nœud) qui est un enfant du même parent, en excluant les nœuds de texte et les nœuds commentaires.

```
const firstElement = document.getElementById("firstElement");
const nextElement = firstElement.nextElementSibling;
console.log(nextElement);
```

- previousElementSibling : Cette propriété renvoie l'élément précédent qui est un enfant du même parent, en excluant les nœuds de texte et les nœuds commentaires.

```
const secondElement = document.getElementById("secondElement");
const previousElement = secondElement.previousElementSibling;
console.log(previousElement);
```

Ces propriétés et méthodes de parcours des nœuds DOM vous permettent d'accéder et de manipuler les éléments et leurs relations dans la structure arborescente du DOM.

Ces propriétés sont utiles pour naviguer plus facilement dans le DOM sans être obligé de surcharger le code avec des classes. Lorsque vous mettez une classe sur un élément, vous savez que vous pouvez aussi accéder à tout ce qui l'entoure sans être obligé de mettre des classes supplémentaires sur chaque ligne de votre code html.

3. Modification du Contenu et des Attributs

La modification du contenu et des attributs des éléments du DOM est une partie essentielle de la manipulation des pages web en JavaScript. Voici comment vous pouvez réaliser ces modifications, ce sont quelques exemples :

Modification du Contenu avec textContent:

La propriété textContent permet de modifier ou de récupérer le contenu texte d'un élément du DOM.

Voici deux exemples, un exemple avec la récupération du contenu et un autre exemple avec la modification du contenu :

Récupération du contenu d'un élément du DOM

HTML :

```
<p id="myParagraph">
    Lorem ipsum dolor, sit amet consectetur adipisicing elit.
    Beatae iusto consequuntur culpa repudiandae odit voluptatibus
    reiciendis velit aperiam, ut tempore, atque ad laudantium qui.
</p>
<button id="myButton">Obtenir le contenu</button>
```

JS:

```
const button = document.getElementById("myButton");
button.addEventListener("click", () => {
  const paragraph = document.getElementById("myParagraph");
  const content = paragraph.textContent;
  alert("Contenu du paragraphe:", content);
});
```

Modification du contenu texte d'un élément du DOM :

HTML:

```
<p id="myParagraph2">Contenu initial</p>
<button id="myButton2">Modifier le contenu</button>
```

JS:

```
const button2 = document.getElementById("myButton2");
button2.addEventListener("click", () => {
  const paragraph2 = document.getElementById("myParagraph2");
  paragraph2.textContent = "Hommage à Bernard : \
  Moitié homme, moitié robot, \
  Le plus valeureux des héros \
  Bioman, Bioman \
  Défenseur de la Terre \
  Comme un arc-en-ciel courageux \
  Rouge, rose, vert, jaune et bleu \
  Bioman, Bioman \
  Héros de l Univers";
});
```

Modification du Contenu avec innerHTML:

La propriété innerHTML permet de définir ou de récupérer le contenu HTML d'un élément du DOM.

```
const element = document.getElementById("myElement");
element.innerHTML = "<strong>Nouveau contenu en
gras</strong>";
```

Exercice :
Ajoutez un paragraphe avec du texte à l'intérieur. Ajoutez un bouton pour afficher le contenu html de l'élément dans la console.

Ajoutez un deuxième paragraphe avec du texte souligné à l'intérieur. Ajoutez un bouton pour modifier le contenu html de cet élément, pour que le texte ne soit plus souligné mais en gras.

Modification des Attributs avec setAttribute:

La méthode setAttribute permet de définir ou de modifier les attributs d'un élément du DOM.

```
const link = document.getElementById("myLink");
link.setAttribute("href", "https://www.example.com");
```

Exercice :
Ajoutez une div avec un texte directement à l'intérieur.

```
<div id="myDiv">Je suis un texte dans une div</div>
```

Ajoutez un bouton qui définit le style de la div, en ajoutant un arrière-plan de couleur bleue, une couleur blanche pour le texte, et une taille de 16px.

Suppression des Attributs avec removeAttribute:

La méthode removeAttribute permet de supprimer un attribut spécifique d'un élément du DOM.

```
const image = document.getElementById("myImage");
image.removeAttribute("src");
```

Exercice :
Ajoutez un champ (input) pour récupérer le prénom de l'utilisateur.
Ajoutez un bouton de validation pour afficher le prénom avec une alert.
Le bouton doit être caché par défaut. (hidden)
On affichera le bouton uniquement si le champ est rempli.

Pour cela on va écouter l'input et on interviendra lorsque le champ est rempli pour afficher le bouton.

Voici le début de code, il faut modifier les deux lignes sous les commentaires en utilisant setAttribute() et removeAttribute(). On verra en détail par la suite l'utilisation de "value" qui est ici utilisé avec myInput.value. C'est ce qui permet de récupérer la valeur de l'input donc ce qui est saisi par l'utilisateur.
HTML :

```
<input type="text" id="myInput" placeholder="Entrez votre prénom">
<button id="submitButton" hidden>Soumettre</button>
```

JS :

```
myInput.addEventListener("input", () => {
  if (myInput.value !== "") {
    //à remplir
  } else {
    //à remplir
  }
});
```

Modification du contenu avec outerHTML :

Récupère ou définit le code HTML complet de l'élément, y compris l'élément lui-même.

```
const element = document.getElementById("myElement");
console.log(element.outerHTML); // Affiche le code HTML complet (faites une
comparaison avec innerHTML pour voir la différence
```

Modification du contenu avec insertAdjacentHTML :

Insère du code HTML à un emplacement spécifié par rapport à l'élément cible.

```
const element = document.getElementById("myElement");
element.insertAdjacentHTML("beforeend", "<p>Nouveau paragraphe</p>");
```

4. Manipulation de Classes CSS

Ces méthodes vous permettent de manipuler le contenu, les attributs et le style des éléments du DOM en fonction de vos besoins pour créer des pages web interactives et dynamiques.

Utilisation de classList et ses méthodes.

La manipulation de classes CSS à l'aide de la propriété classList est une méthode courante pour gérer la présentation et le style des éléments du DOM en JavaScript.

Cela vous permettra de modifier directement du style en fonction des interactions de l'utilisateur sur votre page web.

Voici comment vous pouvez utiliser classList pour ajouter, supprimer et basculer des classes, ainsi que pour modifier le style d'éléments via les classes. La propriété classList d'un élément fournit un objet qui contient plusieurs méthodes utiles pour gérer les classes CSS de l'élément :

1. add: Ajoute une ou plusieurs classes à l'élément.
2. remove: Supprime une ou plusieurs classes de l'élément.
3. toggle: Ajoute la classe si elle n'est pas déjà présente, sinon la supprime.
4. contains: Vérifie si l'élément possède une classe spécifique.
5. item: Récupère la classe d'index spécifié.
6. replace: Remplace une classe par une autre.
7. toggle: Ajoute ou supprime une classe en fonction d'une condition.

Exemple: Utilisation de classList et ses méthodes:

JS:

```
const myDiv2 = document.getElementById("myDiv2");
const addClassButton = document.getElementById("addClassButton");
const removeClassButton = document.getElementById("removeClassButton");
const toggleButton = document.getElementById("toggleButton");
const getClassButton = document.getElementById("getClassButton");
const replaceClassButton = document.getElementById("replaceClassButton");

addClassButton.addEventListener("click", () => {
  //add pour ajouter une classe
  myDiv2.classList.add("highlight");
  myDiv2.textContent = "J'ai la classe highlight red";
});

removeClassButton.addEventListener("click", () => {
  //remove pour retirer la classe
  myDiv2.classList.remove("box");
});

toggleButton.addEventListener("click", () => {
  //toggle (bacule en  anglaise) pour basculer d'un état à un autre, c'est
comme un interrupteur
  myDiv2.classList.toggle("highlight");
  // on utilse contains pour verifier si la class est présente
  if(myDiv2.classList.contains("highlight")) {
    myDiv2.textContent = "J'ai la classe highlight red";
  } else {
    myDiv2.textContent = "Je n'ai pas la classe highlight";
  }
});
```

HTML :

```
<div id="myDiv2" class="box">Contenu initial</div>
<button id="addClassButton">Ajouter une classe</button>
<button id="removeClassButton">Supprimer une classe</button>
<button id="toggleButton">Basculer la classe</button>
<br>
<br>
<div id="myDiv3" class="class1 class2 class3">Contenu</div>
<button id="getClassButton">Obtenir la classe</button>
```

Information sur className

L'utilisation de className est une méthode plus ancienne pour manipuler les classes CSS d'un élément du DOM. Bien qu'elle soit toujours valide et largement prise en charge, la propriété classList offre une meilleure approche car elle fournit des méthodes plus puissantes et flexibles pour gérer les classes. Cependant, pour vous montrer la différence, voici comment vous pourriez utiliser className pour accomplir des tâches similaires :

Utilisation de className **pour Ajouter et Supprimer des classes**:

JS:

```
const myDivClassName = document.getElementById("myDivClassName");
const addClassNameButton = document.getElementById("addClassNameButton");
const removeClassNameButton =
document.getElementById("removeClassNameButton");

addClassButton.addEventListener("click", () => {
  //Pour ajouter on manipule la chaine de caractères avec =+
  //il ne faut pas oublier l'espace avant highlight ou vos classes seront
collées et cela ne fonctionnera pas
  myDivClassName.className += " highlight";
});

removeClassNameButton.addEventListener("click", () => {
  //il n'y a pas de fonction remove avec className, on passe donc par
  //replace avec className. Replace utilise deux attributs, le premier est
  //celui qu'on va remplacer le deuxième et celui qui prendre la place
  //Ici on "triche" en lui disant de le remplacer par rien avec les doubles
  //cotes vide pour le supprimer.zsz
  myDivClassName.className = myDivClassName.className.replace("box", "");
});
```

HTML :

```
<div id="myDivClassName" class="box">Contenu initial d'une div que l'on
va manipuler avec className</div>
<button id="addClassNameButton">Ajouter une classe avec
className</button>
<button id="removeClassNameButton">Supprimer une classe avec
className</button>
```

Modification du style d'éléments via className:

CSS:

```
.highlight3 {
    background-color: yellow;
    color: black;
}
```

JS:

```
const myDivClassName2 = document.getElementById("myDivClassName2");
const toggleStyleButtonClassName =
document.getElementById("toggleStyleButtonClassName");

toggleStyleButtonClassName.addEventListener("click", () => {
  if (myDivClassName2.className.includes("highlight3")) {
    myDivClassName2.className = myDivClassName2.className.replace("
highlight3", "");
  } else {
    myDivClassName2.className += " highlight3";
  }
});
```

HTML:

```
<div id="myDivClassName2" class="box">Contenu initial d'une deuxieme div
avec className</div>
<br>
<button id="toggleStyleButtonClassName">Modifier le style avec
className</button>
```

En utilisant className, vous modifiez directement la chaîne de caractères de classes d'un élément. Cependant, classList offre des méthodes plus conviviales pour effectuer les mêmes actions et pour gérer les classes de manière plus sûre, en évitant les erreurs liées à la manipulation de chaînes.

Par conséquent, l'utilisation de classList est recommandée pour la manipulation des classes CSS. Vous devez savoir qu'il existe className car vous serez amené à travailler sur des projets plus anciens qui utiliseront certainement className. Mais cette méthode est dépréciée, il ne faut donc plus l'utiliser quand c'est possible. (deprecated en anglais)

5. Les différentes propriétés pour gérer les interactions avec les DOM

Dans cette section, nous allons plonger plus profondément dans la manipulation du Document Object Model (DOM) en explorant des propriétés avancées qui vous permettront de tirer le meilleur parti de vos interactions avec les éléments d'une page web.

Value

La propriété value est une propriété couramment utilisée pour accéder à la valeur d'un élément de formulaire dans le DOM, principalement pour les éléments <input>, <textarea> et <select>. Elle vous permet de récupérer la valeur saisie ou sélectionnée par l'utilisateur dans ces éléments.

Voyons un exemple concret :

HTML

```
<input type="text" id="nameInput" placeholder="Entrez votre nom">
<button id="submitButton">Soumettre</button>
```

JS:

```
const nameInput = document.getElementById("nameInput");
const submitButton = document.getElementById("submitButton");

submitButton.addEventListener("click", () => {
  const enteredName = nameInput.value;
  alert("Nom saisi : " + enteredName);
});
```

Explication :

- L'élément nameInput est sélectionné à l'aide de getElementById.
- Lorsque le bouton submitButton est cliqué, l'événement déclenche la fonction qui récupère la valeur de nameInput.value.
- L'alerte affiche ensuite le nom saisi par l'utilisateur.

Une chose importante qu'il faut savoir avec Value c'est qu'on peut le "parser" (analyser).

"Parser" (mot anglais) est le processus de conversion d'une chaîne de caractères en un autre type de données, tel que nombre, date, etc. Cela est particulièrement utile lorsque vous avez besoin de traiter les données saisies par l'utilisateur d'une manière spécifique.

Voici quelques exemples de méthodes de parsing couramment utilisées :

parseInt() et **parseFloat()**
Ces méthodes sont utilisées pour analyser une chaîne de caractères et la convertir en nombre entier (parseInt()) ou en nombre décimal (parseFloat()). Par exemple :

```
<input type="text" id="numericInput">
<button id="parseNumericButton">Parser en Nombre</button>
```

```
const numericInput = document.getElementById("numericInput");
const parseNumericButton =
document.getElementById("parseNumericButton");

parseNumericButton.addEventListener("click", () => {
```

```
  const numericValue = numericInput.value;
  const integerValue = parseInt(numericValue);
  const floatValue = parseFloat(numericValue);

  console.log("Entier : " + integerValue);
  console.log("Flottant : " + floatValue);
});
```

new Date()

Lorsque vous traitez des dates, vous pouvez utiliser le constructeur Date pour analyser une chaîne de caractères et créer un objet Date. Par exemple :

```
<input type="text" id="dateInput">
<button id="parseDateButton">Parser la Date</button>
```

```
const dateInput = document.getElementById("dateInput");
const parseDateButton = document.getElementById("parseDateButton");

parseDateButton.addEventListener("click", () => {
  const dateString = dateInput.value;
  const dateObject = new Date(dateString);

  console.log("Date analysée : " + dateObject);
});
```

JSON.parse()

Si vous avez une chaîne de caractères JSON, vous pouvez utiliser JSON.parse() pour la convertir en un objet JavaScript. Par exemple :

```
<textarea id="jsonInput"></textarea>
<button id="parseJSONButton">Parser JSON</button>
```

```
const jsonInput = document.getElementById("jsonInput");
const parseJSONButton = document.getElementById("parseJSONButton");

parseJSONButton.addEventListener("click", () => {
  const jsonString = jsonInput.value;
  const jsonObject = JSON.parse(jsonString);

  console.log("Nom : " + jsonObject.name);
  console.log("Âge : " + jsonObject.age);
});
```

Boolean()

Pour analyser une chaîne en un booléen, vous pouvez utiliser le constructeur Boolean(). Cela est utile pour convertir des valeurs comme "true" et "false". Par exemple :

```
<select id="booleanSelect">
  <option value="true">Vrai</option>
  <option value="false">Faux</option>
```

```
</select>
<button id="parseBooleanButton">Parser Booléen</button>
```

```
const booleanSelect = document.getElementById("booleanSelect");
const parseBooleanButton = document.getElementById("parseBooleanButton");

parseBooleanButton.addEventListener("click", () => {
  const booleanValue = booleanSelect.value;
  const parsedBoolean = Boolean(booleanValue);

  console.log("Valeur booléenne : " + parsedBoolean);
});
```

Gardez à l'esprit que lors de l'analyse, assurez-vous que la chaîne de caractères est dans un format attendu pour obtenir des résultats précis.

Checked, **selected**, **disabled** et **readonly**

Ces propriétés sont essentielles pour interagir avec les éléments de formulaire, tels que les cases à cocher, les boutons radio, et les options de sélection. Illustrons cela avec un exemple :

```
<input type="checkbox" id="checkBox"> Case à Cocher
<select id="colorSelect">
  <option value="red">Rouge</option>
  <option value="green">Vert</option>
  <option value="blue">Bleu</option>
</select>
<button id="submitButton">Soumettre</button>
```

```
const checkBox = document.getElementById("checkBox");
const colorSelect = document.getElementById("colorSelect");
const submitButton = document.getElementById("submitButton");

submitButton.addEventListener("click", () => {
  const isCheckBoxChecked = checkBox.checked;
  const selectedColor = colorSelect.selectedOptions[0].value;

  if (isCheckBoxChecked) {
    alert("Case à cocher cochée. Couleur sélectionnée : " + selectedColor);
  } else {
    alert("Case à cocher non cochée. Aucune couleur sélectionnée.");
  }
});
```

Explication :

- La propriété checked est utilisée pour vérifier si la case à cocher est cochée.

- La propriété selected permet de déterminer si une option d'un élément <select> est sélectionnée.

Dans l'exemple, l'alerte affiche l'état de la case à cocher et la couleur sélectionnée dans le menu déroulant.

Il existe aussi les attributs disabled et readonly qui sont utilisés dans les éléments de formulaire pour contrôler l'interactivité et l'édition. L'attribut disabled désactive complètement l'élément, tandis que l'attribut readonly permet uniquement la visualisation sans modification de l'élément.

Style

La propriété style permet d'accéder et de modifier les propriétés de style en ligne d'un élément. Voici un exemple pratique :

```
<div id="myDiv" style="width: 100px; height: 100px; background-color:
blue;"></div>
<button id="changeColorButton">Changer la Couleur</button>
```

```
const myDiv = document.getElementById("myDiv");
const changeColorButton = document.getElementById("changeColorButton");

changeColorButton.addEventListener("click", () => {
  myDiv.style.backgroundColor = "red";
});
```

Explication :

- L'élément myDiv a des styles définis en ligne.
- Lorsque le bouton changeColorButton est cliqué, la fonction change la couleur d'arrière-plan de myDiv en rouge.

Dataset

La propriété dataset vous permet d'interagir avec les attributs data-* d'un élément. Ces attributs sont couramment utilisés pour stocker des données personnalisées. Voici un exemple :

```
<div id="user" data-id="123" data-name="John Doe" data-age="30"></div>
<button id="showUserDataButton">Afficher les Données de
l'Utilisateur</button>
```

```
const userElement = document.getElementById("user");
const showUserDataButton = document.getElementById("showUserDataButton");

showUserDataButton.addEventListener("click", () => {
  const userId = userElement.dataset.id;
  const userName = userElement.dataset.name;
  const userAge = userElement.dataset.age;
```

```
  alert(`ID: ${userId}, Nom: ${userName}, Âge: ${userAge}`);
});
```

Explication :

- L'élément user possède des attributs data-* pour stocker des informations spécifiques à l'utilisateur.
- Lorsque le bouton "Afficher les Données de l'Utilisateur" est cliqué, les données sont extraites à partir de userElement.dataset et affichées dans une alerte.

Src et href

Les propriétés src et href sont utilisées pour accéder aux URLs d'images et de liens respectivement. Voici comment les utiliser :

```
<img id="imageElement" src="image.jpg" alt="Image">
<a id="linkElement" href="https://www.example.com">Lien</a>
```

```
const imageElement = document.getElementById("imageElement");
const linkElement = document.getElementById("linkElement");

const imageURL = imageElement.src;
const linkURL = linkElement.href;

console.log("URL de l'image : " + imageURL);
console.log("URL du lien : " + linkURL);
```

Explication :

- La propriété src permet d'accéder à l'URL de l'image.
- La propriété href permet d'accéder à l'URL du lien.

Il existe bien entendu de nombreuses propriétés pour interagir et récupérer des informations que l'on peut encore manipuler par la suite. En voici quelques-unes de plus sur le comportement des pages web:

- offsetWidth et offsetHeight : Ces propriétés fournissent la largeur et la hauteur totales de l'élément, y compris les bordures et la défilement interne (scrollbar) si elles sont présentes.
- clientWidth et clientHeight : Ces propriétés représentent la largeur et la hauteur de la zone d'affichage de l'élément, excluant les bordures, le padding et la scrollbar.
- scrollWidth et scrollHeight : Ces propriétés indiquent la largeur et la hauteur totales de l'élément, incluant les parties non visibles qui nécessitent un défilement.
- scrollTop et scrollLeft : Ces propriétés déterminent la quantité de défilement vertical (scrollTop) et horizontal (scrollLeft) appliquée à l'élément.

Ces propriétés sont utiles pour récupérer et manipuler des informations spécifiques sur les éléments et pour contrôler leur comportement dans la page.

6. Écouteurs d'Événements

Introduction aux événements.

Les événements sont des actions ou des occurrences qui se produisent dans le navigateur, comme un clic de souris, une pression de touche ou le chargement d'une page. Le DOM permet de capturer ces événements et de réagir à eux en exécutant du code JavaScript.

Utilisation d'addEventListener.

La méthode addEventListener est utilisée pour attacher un gestionnaire d'événements à un élément du DOM. Ce gestionnaire d'événements sera exécuté lorsque l'événement spécifié se produit sur l'élément cible.

En terme simple, nous allons "écouter" un évènement (event) et nous lui donnerons des instructions lorsque cet évènement se produit. Un évènement est une interaction avec la page web. Par exemple, lorsque l'utilisateur clique sur un bouton, lorsqu'il appuie sur une touche, lorsqu'il survole quelque chose avec sa souris etc..

Voici la syntaxe de la méthode addEventListener de l'objet élément :

```
element.addEventListener(eventType, eventHandler, useCapture);
```

- element: L'élément DOM auquel vous souhaitez attacher le gestionnaire d'événements.
- eventType: Une chaîne de caractères représentant le type d'événement que vous souhaitez écouter (par exemple, "click", "mouseover", "submit", etc.).
- eventHandler: Une fonction qui sera exécutée lorsque l'événement se produit.
- useCapture (facultatif) : Un booléen qui indique si l'événement doit être capturé dans la phase de capture (true) ou dans la phase de bouillonnement (false, valeur par défaut).

Nous déjà beaucoup utilisé addEventListener, vous pouvez donc comparer cette syntaxe avec les exemples précédents. Voici un autre exemple avant de voir une liste des différents évènements les plus utilisés.

```
// Sélection de l'élément bouton
const buttonEvent = document.getElementById("myButtonEvent");
// Fonction gestionnaire d'événements
function handleClick() {
    alert("Le bouton a été cliqué !");
}
// Ajout du gestionnaire d'événements au clic sur le bouton
// Nous écoutons l'event click et nous lui donnons l'instruction
// de lancer la fonction handleClick si le bouton est cliqué
buttonEvent.addEventListener("click", handleClick);
```

Ici nous utilisons une ancienne façon d'ecrire du javascript mais qui fonctionne parfaitement. Avec ES6, qui est une des dernières mises à jour de javascript, nous utiliserons une syntaxe plus courte et plus rapide avec des fonctions fléchées :

```
buttonEvent.addEventListener("click", () => {
    alert("Le bouton a été cliqué !!!!!!");
});
```

N'oubliez pas de mettre en commentaire la fonction handleClick() et l'autre addEventListener pour que la fonction fléchée fonctionne.

Les événements courants

Voici une liste des principaux événements avec des exemples concrets d'utilisation pour chaque événement. Vous pouvez survoler le code rapidement, ce n'est que de la théorie, vous allez y revenir avec une liste d'exercices. Cependant, lisez bien les différents évènements pour que vous sachiez ce qui existe. N'oubliez pas que ce sont les plus utilisés, il en existe d'autres.

Clic de souris (click):
Action : Clic gauche de la souris sur un élément.
Exemple :

```
const button = document.getElementById("myButton");
button.addEventListener("click", () => {
  alert("Le bouton a été cliqué !");
});
```

Double-clic de souris (dblclick):
Action : Double-clic gauche de la souris sur un élément.
Exemple :

```
const element = document.getElementById("myElement");
element.addEventListener("dblclick", () => {
  alert("Double-clic sur l'élément !");
});
```

Survole de la souris (mouseover et mouseout):
Action : Passage de la souris sur un élément (entrée ou sortie).
Exemple :

```
const element = document.getElementById("myElement");
element.addEventListener("mouseover", () => {
  element.style.backgroundColor = "yellow";
});
element.addEventListener("mouseout", () => {
  element.style.backgroundColor = "";
});
```

Chargement de la page (load):
Action : La page a terminé de se charger.

Exemple :

```
window.addEventListener("load", () => {
  alert("La page est complètement chargée !");
});
```

Soumission de formulaire (submit):
Action : Soumission d'un formulaire.
Exemple :

```
const form = document.getElementById("myForm");
form.addEventListener("submit", (event) => {
  event.preventDefault(); // Empêche la soumission normale du formulaire
  alert("Formulaire soumis !");
});
```

Pression de touche (keydown, keyup, keypress):
Action : Appui, relâchement ou maintien d'une touche du clavier.
Exemple :

```
document.addEventListener("keydown", (event) => {
  if (event.key === "Enter") {
    alert("Touche Entrée pressée !");
  }
});
```

Changement de valeur d'entrée (input, change):
Action : Changement de la valeur d'un champ de texte ou d'un élément de formulaire.
Exemple :

```
const input = document.getElementById("myInput");
input.addEventListener("input", () => {
  console.log("La valeur a changé :", input.value);
});
```

Redimensionnement de la fenêtre (resize):
Action : Redimensionnement de la fenêtre du navigateur.
Exemple :

```
window.addEventListener("resize", () => {
  alert("La fenêtre a été redimensionnée !");
});
```

Focus sur un élément (focus):
Action : Un élément obtient le focus (devient actif pour les entrées).
Exemple :

```
const input = document.getElementById("myInput");
input.addEventListener("focus", () => {
  input.style.backgroundColor = "lightblue";
});
```

Perte du focus sur un élément (blur):
Action : Un élément perd le focus (n'est plus actif pour les entrées).
Exemple :

```
const input = document.getElementById("myInput");
input.addEventListener("blur", () => {
  input.style.backgroundColor = "";
});
```

Changement de valeur sélectionnée (change):
Action : Changement de la valeur sélectionnée dans un élément de formulaire.
Exemple :

```
const select = document.getElementById("mySelect");
select.addEventListener("change", () => {
  alert("Option sélectionnée : " + select.value);
});
```

Sélection de texte (select):
Action : Sélection de texte dans un élément de formulaire.
Exemple :

```
const input = document.getElementById("myInput");
input.addEventListener("select", () => {
  alert("Texte sélectionné : " +
input.value.substring(input.selectionStart, input.selectionEnd));
});
```

Ces exemples illustrent comment utiliser les principaux événements du DOM en JavaScript pour ajouter de l'interactivité et réagir aux actions de l'utilisateur sur une page web.

Exercices sur les évènements :

- Exercice 1: Clic sur un Élément

Créez un bouton et ajoutez un gestionnaire d'événements pour afficher une alerte lorsque le bouton est cliqué.

- Exercice 2: Survole d'un Élément

Créez un paragraphe et ajoutez un gestionnaire d'événements pour changer sa couleur de fond lorsque la souris le survole.

- Exercice 3: Pression de Touche Spécifique

Lorsque l'utilisateur appuie sur la touche "Enter", affichez une alerte.

- Exercice 4: Bouton d'Annulation

Créez un formulaire avec un champ de texte et un bouton "Annuler". Lorsque l'utilisateur clique sur "Annuler", réinitialisez la valeur du champ de texte.

- Exercice 5: Clic droit personnalisé

Empêchez le menu contextuel d'apparaître lorsque l'utilisateur effectue un clic droit sur la page.

- Exercice 6: Modification de la Sélection

Créez un champ de texte et affichez l'index de début et de fin de la sélection de texte dans la console lorsque l'utilisateur sélectionne du texte.

- Exercice 7: Formulaire de Confirmation

Créez un formulaire de soumission avec un bouton "Soumettre". Lorsque l'utilisateur clique sur "Soumettre", affichez une boîte de dialogue de confirmation. Si l'utilisateur confirme, affichez une alerte de succès.

- Exercice 8: Focus et Blur

Créez un champ de texte et changez sa couleur de fond lorsque l'élément obtient le focus, puis rétablissez-la lorsque l'élément perd le focus.

- Exercice 9: Changement de Valeur d'Entrée

Créez un champ de texte et affichez dans la console le contenu de ce champ chaque fois que sa valeur change.

- Exercice 10: Bouton de Toggle

ÉCréez un bouton "Basculer" et une zone de texte vide. Lorsque l'utilisateur clique sur "Basculer", affichez ou masquez la zone de texte en fonction de son état actuel.

7. Création d'Éléments et Gestion des Parents/Enfants

Création d'Éléments avec createElement

La méthode createElement vous permet de créer de nouveaux éléments HTML en utilisant JavaScript. Vous spécifiez le nom de la balise de l'élément que vous souhaitez créer, puis vous pouvez l'ajouter au DOM.
Exemple :

```
const newParagraph = document.createElement("p");
newParagraph.textContent = "Nouveau paragraphe créé avec createElement.";
document.body.appendChild(newParagraph);
```

Ajout et Suppression d'Éléments avec appendChild **et** removeChild

La méthode appendChild vous permet d'ajouter un nouvel élément en tant qu'enfant d'un élément existant. La méthode removeChild vous permet de supprimer un enfant spécifique d'un élément parent.
Exemple :

```
const parentElement = document.getElementById("parent");
```

```
const newChild = document.createElement("div");
parentElement.appendChild(newChild); // Ajoute un nouvel enfant
parentElement.removeChild(newChild); // Supprime l'enfant nouvellement
ajouté
```

Traverser et Manipuler la Hiérarchie Parent/Enfants

Vous pouvez accéder aux éléments parent et enfants d'un élément donné en utilisant les propriétés parentNode, childNodes, firstChild, lastChild etc… (déjà vu précédemment dans le parcours des nœuds du DOM).

Mais cela existe aussi par rapport **aux éléments** directement :

- **parentElement :**

parentElement est une propriété spécifique aux éléments HTML et représente l'élément parent immédiat d'un élément donné. Cette propriété retourne directement l'élément parent sous forme d'objet.
Exemple :

HTML :

```
<div id="parent">
  <p id="child">Ceci est un paragraphe.</p>
</div>
```

JS :

```
const childElement = document.getElementById("child");
const parentDiv = childElement.parentElement; // Renvoie l'élément <div
id="parent">
```

Différence avec parentNode :

```
const childElement = document.getElementById("child");
const parentDivNode = childElement.parentNode; // Renvoie le nœud de type
élément <div id="parent">
```

La principale différence réside dans le fait que parentElement est spécifique aux éléments HTML, tandis que parentNode est une propriété plus générale qui s'applique à tous les types de nœuds du DOM. Pour manipuler les éléments HTML spécifiquement, il est recommandé d'utiliser parentElement pour accéder à l'élément parent.

En voici d'autres pour sélectionner des éléments :

- **children :**

La propriété children d'un élément renvoie une collection en lecture seule des éléments enfants qui sont des nœuds éléments (balises HTML), excluant les nœuds de texte et les nœuds de commentaire.

```
<div id="parent">
```

```
  <p>Paragraphe 1</p>
  <p>Paragraphe 2</p>
</div>
```

```
const parentElement = document.getElementById("parent");
const childrenElements = parentElement.children; // Renvoie une collection
d'éléments <p>
// Donc renvoie un tableau d'élèments
```

- **firstElementChild :**

La propriété firstElementChild renvoie le premier élément enfant d'un élément donné, excluant les nœuds de texte et les nœuds de commentaire.

```
const firstChildElement = parentElement.firstElementChild; // Renvoie
l'élément <p> "Paragraphe 1"
```

- **lastElementChild :**

La propriété lastElementChild renvoie le dernier élément enfant d'un élément donné, excluant les nœuds de texte et les nœuds de commentaire.

```
const lastChildElement = parentElement.lastElementChild; // Renvoie
l'élément <p> "Paragraphe 2"
```

- **previousElementSibling :**

La propriété previousElementSibling renvoie l'élément précédent (frère) d'un élément donné, excluant les nœuds de texte et les nœuds de commentaire.

- **nextElementSibling :**

La propriété nextElementSibling renvoie l'élément suivant (frère) d'un élément donné, excluant les nœuds de texte et les nœuds de commentaire.

```
const firstChild = document.querySelector("p:first-child");
const nextSibling = firstChild.nextElementSibling; // Renvoie l'élément <p>
"Paragraphe 2"
```

IMPORTANT :

Il est aussi possible de multiplier les sélections. Par exemple pour sélectionner le parent d'un parent. C'est très utilisé.

Supposons que nous ayons la structure HTML suivante :

```
<div id="grandParent">
  <div id="parent">
    <p>Contenu du parent</p>
  </div>
</div>
```

Et voici comment nous pouvons utiliser parentElement.parentElement pour accéder au grand-parent de l'élément <p> :

```
const paragraphElement = document.querySelector("p");
const parentElement = paragraphElement.parentElement; // Renvoie l'élément
<div id="parent">
const grandparentElement = parentElement.parentElement; // Renvoie
l'élément <div id="grandparent">
```

Dans la pratique nous écrirons plutôt :

```
const paragraphElementGrandParent = document.querySelector("p").parentElement.parentElement;
```

Dans cet exemple, paragraphElement est l'élément <p> contenu à l'intérieur de l'élément parent . En utilisant parentElement.parentElement, nous pouvons accéder à l'élément grand-parent .
Cela peut être utile lorsque vous avez besoin d'accéder à des éléments dans des niveaux hiérarchiques plus élevés dans la structure du DOM.

8. Exercices

Faites un document HTML avec un paragraphe. Il doit contenir ce texte :
" Le nouveau roi du dessin animé, c'est Bob. Et ça dure depuis 1999. Pour ceux qui ont vécu dans une caverne sous-marine loin, très loin de la bonne ville de Bikini Bottom, Bob est une joyeuse éponge de mer qui vit dans une jolie cité des profondeurs de l'océan Pacifique et travaille comme cuisinier dans un snack aquatique."

Exercice 1: Sélection par ID et Modification de Texte
Sélectionnez l'élément avec l'ID "target-paragraph" et modifiez son texte pour mieux rendre hommage à Bob l'éponge et afficher :
"Bob l'éponge est un personnage dynamique et optimiste mais également très naïf. L'un de ses passe-temps préférés est la chasse à la méduse, une activité similaire à l'observation ornithologique et à la collection d'insectes, et la création de bulles en compagnie de Patrick. Il ne sait pas à quel point il agace Carlo.".

Vous devez également mettre les noms de personnage entre balise avec une classe "highlight". Les noms de personnages doivent être en gras et en bleu.

Exercice 2: Sélection par Classe et Modification de Style
Sélectionnez tous les éléments avec la classe "highlight" et changez leur couleur de texte en rouge et soulignez-les.

Exercice 3: Parcours des Nœuds Enfants et Modification d'Attributs
Dans votre document HTML vous devez rajouter une liste avec 5 éléments à cette liste avec 5 qualités de Bob l'éponge.

Sélectionnez l'élément avec l'ID "list-container", parcourez ses nœuds enfants (éléments de la liste) et ajoutez un attribut "data-index" à chaque élément, en numérotant les éléments de 1 à N.

Exercice 4: Sélection par Balise et Suppression d'Élément
Sélectionnez tous les éléments <li> dans la liste avec l'ID "target-list" et supprimez le dernier élément de la liste.

Exercice 5: Parcours des Nœuds Parents et Modifications Multiples
Sélectionnez un élément de votre choix, puis utilisez le parcours des nœuds parents pour accéder à son parent direct. Ensuite, modifiez le contenu du parent pour afficher "Contenu modifié" et ajoutez une classe "modified" au parent.

N'oubliez pas de créer un fichier JavaScript séparé pour chaque exercice et de lier ces fichiers à votre page HTML pour tester les résultats.

9. Conclusion

La manipulation du DOM est une compétence essentielle pour les développeurs web, car elle permet de créer des interactions dynamiques et réactives sur les pages web. Grâce à la manipulation du DOM, les sites web peuvent répondre aux actions des utilisateurs en temps réel, offrant ainsi une expérience utilisateur plus fluide et agréable.

En utilisant JavaScript pour modifier le contenu, le style et le comportement des éléments dans le DOM, les développeurs peuvent créer des fonctionnalités telles que des formulaires interactifs, des mises à jour en direct, des galeries d'images dynamiques et bien plus encore.

En résumé, la manipulation du DOM est un élément clé du développement web moderne, permettant de transformer des pages statiques en applications web interactives et engageantes. En comprenant les concepts et les techniques que nous avons explorés, vous serez mieux équipé pour créer des expériences web dynamiques et engageantes pour les utilisateurs.

Nous allons finir d'aborder la manipulation du DOM avec un TP qui sera fourni en annexe. Ce TP reprendra certaines parties de ce cours mais aussi ce que vous avez vu jusqu'à maintenant. Il faudra donc créer une page web avec un fichier HTML, une page CSS, et une page JS. Ce sera détaillé dans le TP mais nous utiliserons des boucles, des fonctions fléchées, des fonctions normales, des ternaires, des booléens, du css, de la manipulation du DOM etc…

Ce TP vous permettra de vous rendre compte de ce que l'on peut faire en regroupant une bonne partie de ce que vous apprenez depuis le début. Vous allez aussi sans vous en rendre compte directement manipuler des Objets.

Le prochain cours reviendra sur les objets en détail, mais avec ce que vous aurez vu jusqu'ici, ainsi qu'avec l'initiation sur l'objet que nous avions fait avec les héros (sur

lequel nous reviendrons), le cours du l'objet vous semblera plus abordable et plus concret. Nous passerons ensuite sur le REACT JS.