DOM – Histoire et Manipulation

1WEBD – Javascript Web Development

Sommaire

1. Introduction.
2. Interagir avec des éléments
3. Gestion d’Evénements
4. Formulaires

1. Introduction

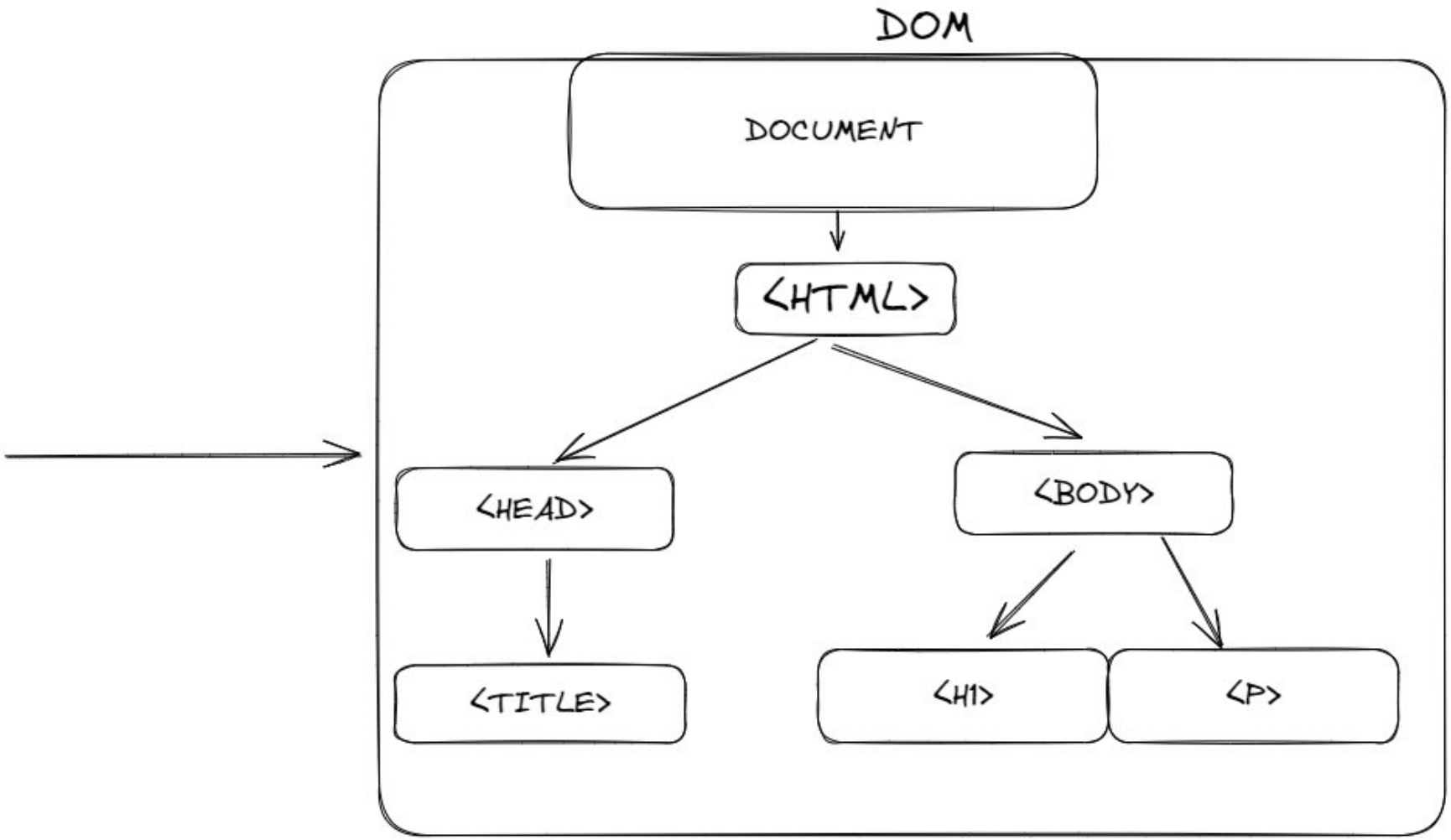
1. Introduction

Introduction

- Document Object Model (DOM)
- Représente la page de sorte à ce que les programmes puissent la comprendre et la modifier
- Représente la page comme un arbre de nœud
- Nœud = partie du document ou élément HTML

1. Introduction

```
<html>
    <head>
        <title>Exemple</title>
    </head>
    <body>
        <h1>un titre</h1>
        <p>un paragraphe</p>
    </body>
</html>
```



1. Introduction

2. Interagir avec des éléments

2. Interagir avec des éléments

Sélectionner des Eléments

- On récupère l'élément parent de celui qu'on veut sélectionner (pour la page, on peut utiliser **document**)
- On utilise des sélecteurs css (**.** pour les classes, **#** pour les ids etc) pour récupérer le(s) élément(s)
- Pour un seul élément, on peut utiliser **.querySelector**
- Pour plusieurs, on peut utiliser **.querySelectorAll**

2. Interagir avec des éléments

Sélectionner des Eléments

```
// HTML pour l'exemple
/*
<div class="container">
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
    <p class="special">Paragraphe 3</p>
</div>
*/

// Sélection du premier paragraphe dans le div avec la classe 'container'
const premierParagraphe = document.querySelector('.container p');
console.log(premierParagraphe.textContent); // Affiche le texte de "Paragraphe 1"

// Sélection du premier paragraphe avec la classe 'special'
const paragrapheSpecial = document.querySelector('.special');
console.log(paragrapheSpecial.textContent); // Affiche le texte de "Paragraphe 3"

// Sélection de tous les paragraphes dans le div avec la classe 'container'
const tousLesParagraphes = document.querySelectorAll('.container p');

// Affichage de chaque paragraphe
tousLesParagraphes.forEach(paragraphe => {
    console.log(paragraphe.textContent);
});
```

2. Interagir avec des éléments

Sélectionner des Eléments - Autres Méthodes

- getElementById
- getElementByClassName

2. Interagir avec des éléments

Modifier les Eléments - contenu

- textContent peut changer le contenu de l'élément
  - mais ce n'est que du texte
- innerHTML peut changer le contenu de l'élément et est interprété comme de l'html

2. Interagir avec des éléments

Modifier les Eléments - Style et Classes

- Pour modifier un style en particulier (ex: display) on peut aller modifier les valeurs dans style
- Pour modifier une classe dans un élément, on peut utiliser le tableau classList

2. Interagir avec des éléments

Modifier les Eléments

```
// HTML pour l'exemple
/*
<div id="monDiv">Contenu de mon div</div>
*/

// Sélection de l'élément
const monDiv = document.getElementById('monDiv');

// Modification des styles CSS
monDiv.style.color = 'blue'; // Change la couleur du texte en bleu
monDiv.style.backgroundColor = 'yellow'; // Change la couleur de fond en jaune
monDiv.style.padding = '10px'; // Ajoute un padding
monDiv.style.border = '1px solid black'; // Ajoute une bordure

// Ajout d'une classe
monDiv.classList.add('maClasse'); // Ajoute la classe 'maClasse' à l'élément

// Vérification de la présence d'une classe
if (monDiv.classList.contains('maClasse')) {
    console.log('La classe maClasse est présente');
}

// Suppression d'une classe
monDiv.classList.remove('maClasse'); // Supprime la classe 'maClasse'

// Alternance d'une classe
monDiv.classList.toggle('autreClasse'); // Ajoute 'autreClasse' si elle n'est pas présente,
la supprime sinon
```

2. Interagir avec des éléments

Modifier les Eléments - Attribut

- On peut modifier les attributs d'un élément avec la méthode **setAttribute**

2. Interagir avec des éléments

Modifier les Eléments - Attribut

```
// HTML pour l'exemple
/*
<button id="monBouton">Cliquez-moi</button>
*/

// Sélection de l'élément
const monBouton = document.getElementById('monBouton');

// Utilisation de setAttribute pour changer l'attribut 'type'
monBouton.setAttribute('type', 'button');

// Utilisation de setAttribute pour ajouter un nouvel attribut 'name'
monBouton.setAttribute('name', 'boutonEnvoyer');

// Utilisation de setAttribute pour ajouter un style inline
monBouton.setAttribute('style', 'background-color: green; color: white;');

// Ajout d'un événement 'click' via setAttribute
// Note: Il est généralement préférable d'utiliser addEventListener pour les événements.
monBouton.setAttribute('onclick', "alert('Bouton cliqué!')");
```

2. Interagir avec des éléments

Ajouter des Eléments – Création

- Pour créer un élément, on peut utiliser **document.createElement**
- Il faut spécifier quel élément on veut créer

2. Interagir avec des éléments

Ajouter des Eléments – Création

```
// HTML pour l'exemple
/*
<div id="container"></div>
*/

// Sélection de l'élément conteneur
const container = document.getElementById('container');

// Création d'un nouvel élément de paragraphe
const nouveauParagraphe = document.createElement('p');

// Ajout de texte au paragraphe
nouveauParagraphe.textContent = "Ceci est un nouveau paragraphe ajouté par JavaScript.";
```

2. Interagir avec des éléments

Ajouter des Eléments - Création - Ajout à la page

- Pour ajouter un élément sur la page, on peut utiliser la méthode **appendChild** de l'élément parent (le nœud au-dessus dans l'arbre DOM)
- Dans ce cas de figure, la variable **document** peut être considérée comme un élément

2. Interagir avec des éléments

Ajouter des Eléments - Création - Ajout à la page

```
// HTML pour l'exemple
/*
<div id="container"></div>
*/

// Sélection de l'élément conteneur
const container = document.getElementById('container');

// Création d'un nouvel élément de paragraphe
const nouveauParagraphe = document.createElement('p');

// Ajout de texte au paragraphe
nouveauParagraphe.textContent = "Ceci est un nouveau paragraphe ajouté par JavaScript.";

// Ajout du paragraphe au conteneur
container.appendChild(nouveauParagraphe);
```

2. Interagir avec des éléments

Ajouter des Eléments - Création - Ajout à la page

- Pour supprimer un élément, on peut utiliser la méthode **removeChild** de l'élément parent
- Dans ce cas de figure, la variable **document** peut être considérée comme un élément

2. Interagir avec des éléments

Ajouter des Eléments - Création - Ajout à la page

```
// HTML pour l'exemple
/*
<div id="container">
    <p id="paragrapheASupprimer">Ce paragraphe sera supprimé.</p>
    <p>Autre paragraphe qui reste.</p>
</div>
*/

// Sélection de l'élément parent (le conteneur)
const container = document.getElementById('container');

// Sélection de l'élément à supprimer
const paragrapheASupprimer = document.getElementById('paragrapheASupprimer');

// Suppression de l'élément sélectionné du conteneur
container.removeChild(paragrapheASupprimer);
```

2. Interagir avec des éléments

3. Gestion d’Evénements

3. Gestion d’Evénements

- Evènements = action ou occurrences qui se produisent dans le navigateur
- Permettent de réagir a des actions de l'utilisateur (click, touches du clavier, mouvement de souris)

3. Gestion d’Evénements

Types

- **Événements de Souris**: **click**, **mouseover**, **mouseout**, **mousemove**, etc.
- **Événements de Clavier**: **keydown**, **keyup**, **keypress**
- **Événements de Formulaire**: **submit**, **change, focus**, **blur**
- **Événements de Fenêtre**: **load**, **resize**, **scroll**, **unload**
- **Evénements de input**: **change**
- La liste est incomplète: plus d'infos ici
- On peut aussi définir ses propres évènements

3. Gestion d’Evénements

Ajout

- La méthode **addEventListener** est un moyen standard d'attacher un gestionnaire d'événements à un élément
- Conseil: déclarez une fonction séparée au lieu de le faire dans l'appel à **addEventListener**; cela permet de le supprimer plus tard

3. Gestion d’Evénements

Ajout

```
// Sélection de l'élément
const monBouton = document.getElementById('monBouton');

// Ajout d'un écouteur d'événement
monBouton.addEventListener('click', function() {
    alert('Bouton cliqué!');
});
```

3. Gestion d’Evénements

Prévention du Comportement par Défaut

- Pour empêcher le comportement par défaut d'un événement, utilisez **preventDefault**

3. Gestion d’Evénements

Prévention du Comportement par Défaut

```
const monFormulaire = document.getElementById('monFormulaire');
monFormulaire.addEventListener('submit', function(e) {
    e.preventDefault(); // Empêche l'envoi du formulaire
});
```

3. Gestion d’Evénements

Suppression du Gestionnaire

- Pour retirer un écouteur, utilisez **removeEventListener** avec la même signature que lors de l'ajout

3. Gestion d’Evénements

Suppression du Gestionnaire

```
// Fonction gestionnaire d'événement
function clicHandler() {
    alert('Bouton cliqué!');
}

// Ajout de l'écouteur
monBouton.addEventListener('click', clicHandler);

// Suppression de l'écouteur
monBouton.removeEventListener('click', clicHandler);
```

3. Gestion d’Evénements

4. Formulaires

4. Formulaires

- Rappel: utiliser les règles de validation html5 (type, format, required)
- On veut éviter de recharger la page à chaque envoie de formulaire
- On va donc utiliser un **preventDefault** sur le formulaire
- Le comportement suite à l'envoie du formulaire va être défini dans le callback du event listener

4. Formulaires

Traitement Formulaire

- Quand un formulaire est envoyé, il émet un evenement **submit**
- Il entraine un rechargement de la page (pour envoyer les données)
- On doit empêcher ce rechargement pour traiter les données avec JS

4. Formulaires

Traitement Formulaire

```
const monFormulaire = document.getElementById('monFormulaire');
monFormulaire.addEventListener('submit', function(e) {
    e.preventDefault(); // Empêche l'envoi du formulaire
});
```

4. Formulaires

Récupérer les valeurs entrées

- On peut récupérer le contenu actuel d'un input avec la propriété **value** de l’élément

4. Formulaires

Récupérer les valeurs entrées

```
// Sélection de l'élément input et du bouton
const monInput = document.getElementById('monInput');
const texteSaisi = monInput.value;

// Affichage de la valeur dans la console
console.log("Texte saisi :", texteSaisi);
```

4. Formulaires