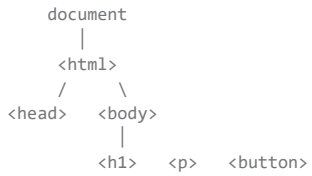
JavaScript - Manipuler le DOM

Le DOM (*Document Object Model*) est la représentation de la page HTML que JavaScript voit. Pour rendre une page dynamique, il faut savoir **sélectionner** des éléments, **lire** leur contenu et le **modifier**. C'est la fondation de toute interactivité côté navigateur.

4TTR 5TTR 6TTR niveau

C'est quoi, le DOM ?

Quand le navigateur charge une page HTML, il en construit une représentation interne : un arbre d'objets, où chaque balise devient un nœud. Cet arbre est le **DOM**.



JavaScript peut **lire et modifier** cet arbre — et le navigateur met à jour l'affichage en conséquence. C'est ce qui rend les pages dynamiques.

L'objet global qui te donne accès à tout ça : `document`.

Sélectionner un élément

Par identifiant — `getElementById`

L'utilisation la plus simple et la plus rapide.

```html
<p id="message">Salut !</p>
```

```js
const p = document.getElementById("message");
```

Renvoie l'élément, ou `null` si aucun élément n'a cet `id`. Un `id` doit être **unique** dans la page.

Par sélecteur CSS — `querySelector` / `querySelectorAll`

Plus flexible : on utilise la syntaxe des sélecteurs CSS.

```js
// Premier élément qui correspond
document.querySelector("#message");        // par id
document.querySelector(".surbrillance");   // par classe
document.querySelector("p");               // première balise <p>
document.querySelector("ul li.actif");     // sélecteur combiné

// Tous les éléments qui correspondent
document.querySelectorAll(".item");        // une NodeList (≈ tableau)
```

Quand utiliser quoi ?

- `getElementById` — un seul élément précis, par son id. Le plus rapide.

© 2025 Mr. Lambrechts

- `querySelector` — un seul élément, sélecteur CSS quelconque.
- `querySelectorAll` — plusieurs éléments. Renvoie une `NodeList` qu'on parcourt comme un tableau.

`querySelector` couvre 90% des cas dès qu'on dépasse les `id`.

Lire et modifier le contenu

Une fois qu'on a un élément, on peut accéder à son contenu :

```
const titre = document.querySelector("h1");

// Lire
console.log(titre.textContent);    // le texte brut
console.log(titre.innerHTML);      // le HTML interne (balises incluses)

// Modifier
titre.textContent = "Nouveau titre";
titre.innerHTML = "Titre <em>en italique</em>";
```

`textContent` ou `innerHTML` ?

Propriété	Effet
`textContent`	Texte brut. Les `<` et `>` deviennent du texte affiché. **À privilégier**.
`innerHTML`	Du HTML. Les balises sont interprétées. Puissant **mais risqué** si on injecte du texte venant de l'utilisateur (faille XSS).

Règle simple : `textContent` **par défaut**. N'utilise `innerHTML` que si tu insères du HTML que **tu as écrit toi-même**, jamais du texte tapé par un utilisateur.

Modifier les attributs

Les attributs HTML (`src`, `href`, `alt`, `disabled`…) sont accessibles directement sur l'élément.

```
const img = document.querySelector("#photo");

img.src = "chat.jpg";
img.alt = "Un chat";
img.width = 300;
```

Pour les attributs personnalisés (`data-*`), on passe par `getAttribute` / `setAttribute` :

```
const carte = document.querySelector(".carte");

carte.setAttribute("data-id", 42);
console.log(carte.getAttribute("data-id"));   // "42"
```

Modifier les classes CSS

© 2025 Mr. Lambrechts

Plutôt que de toucher au style directement, on **bascule des classes** définies dans le CSS — c'est plus propre et plus maintenable.

```
const bouton = document.querySelector("#valider");

bouton.classList.add("actif");          // ajoute la classe
bouton.classList.remove("actif");       // retire la classe
bouton.classList.toggle("actif");       // ajoute si absente, retire si présente
bouton.classList.contains("actif");     // true / false
```

```
/* dans le CSS */
.actif {
    background-color: green;
    color: white;
}
```

Pour la modification ponctuelle du style en JS (couleur en direct, transition…) voir l'article suivant sur le style.

Créer et insérer des éléments

Pour ajouter dynamiquement des éléments à la page :

```
// 1. Créer
const nouveauP = document.createElement("p");
nouveauP.textContent = "Paragraphe ajouté en JS";
nouveauP.classList.add("note");

// 2. Insérer dans le DOM
document.body.appendChild(nouveauP);     // à la fin du body
```

Variantes :

```
parent.appendChild(enfant);         // ajoute à la fin
parent.prepend(enfant);              // ajoute au début
parent.insertBefore(enfant, autre);  // insère avant un autre élément
enfant.remove();                     // retire du DOM
```

Exemple : remplir une liste

```
<ul id="fruits"></ul>
```

```
const fruits = ["pomme", "banane", "cerise"];
const liste = document.getElementById("fruits");

for (let fruit of fruits) {
    const li = document.createElement("li");
    li.textContent = fruit;
    liste.appendChild(li);
}
```

Trois fruits dans le tableau → trois `<li>` dans la page, sans toucher au HTML.

Lire la valeur d'un champ `<input>`

Pour les formulaires, ce ne sont pas `textContent` ni `innerHTML` qu'on utilise, mais la propriété `.value`.

```html
<input id="nom" type="text">
<button id="valider">OK</button>
```

```js
const champ = document.getElementById("nom");
const bouton = document.getElementById("valider");

bouton.addEventListener("click", () => {
    const saisie = champ.value;
    console.log(`L'utilisateur a tapé : ${saisie}`);
});
```

`.value` fonctionne pour `<input>`, `<textarea>` et `<select>`. C'est la manière propre de récupérer une saisie utilisateur sur un vrai site (plutôt que `prompt`).

Le piège : exécuter avant le chargement

```html
<head>
<script>
        const titre = document.querySelector("h1");
        titre.textContent = "Nouveau titre";  // ❌ titre est null !
</script>
</head>
<body>
    <h1>Titre original</h1>
</body>
```

Le script s'exécute **avant** que `<h1>` n'existe dans le DOM → `querySelector` renvoie `null` → erreur en accédant à `.textContent`.

Deux solutions :

1. Mettre le `<script>` **à la fin du** `<body>`, juste avant `</body>`. C'est le réflexe simple.
2. Attendre l'événement `DOMContentLoaded` :

```js
document.addEventListener("DOMContentLoaded", () => {
    const titre = document.querySelector("h1");
    titre.textContent = "Nouveau titre";
});
```

À retenir

- Le DOM = arbre d'objets que JavaScript voit, accessible via `document`.
- **Sélection** : `getElementById("id")`, `querySelector("css")`, `querySelectorAll("css")`.
- **Contenu** : `textContent` (sûr, par défaut) vs `innerHTML` (puissant, risqué).
- **Attributs** : `.src`, `.href`, `.value`… ou `getAttribute` / `setAttribute`.
- **Classes** : `classList.add` / `remove` / `toggle` — préférer les classes CSS au style en dur.
- **Création** : `createElement` + `appendChild` pour ajouter des éléments dynamiquement.
- **Champs de formulaire** : `.value` pour lire une saisie.
- **Toujours** placer le `<script>` à la fin du `<body>` (ou utiliser `DOMContentLoaded`).

© 2025 Mr. Lambrechts