



## Partie 3 – Manipulation du DOM #1

---

### Table des matières

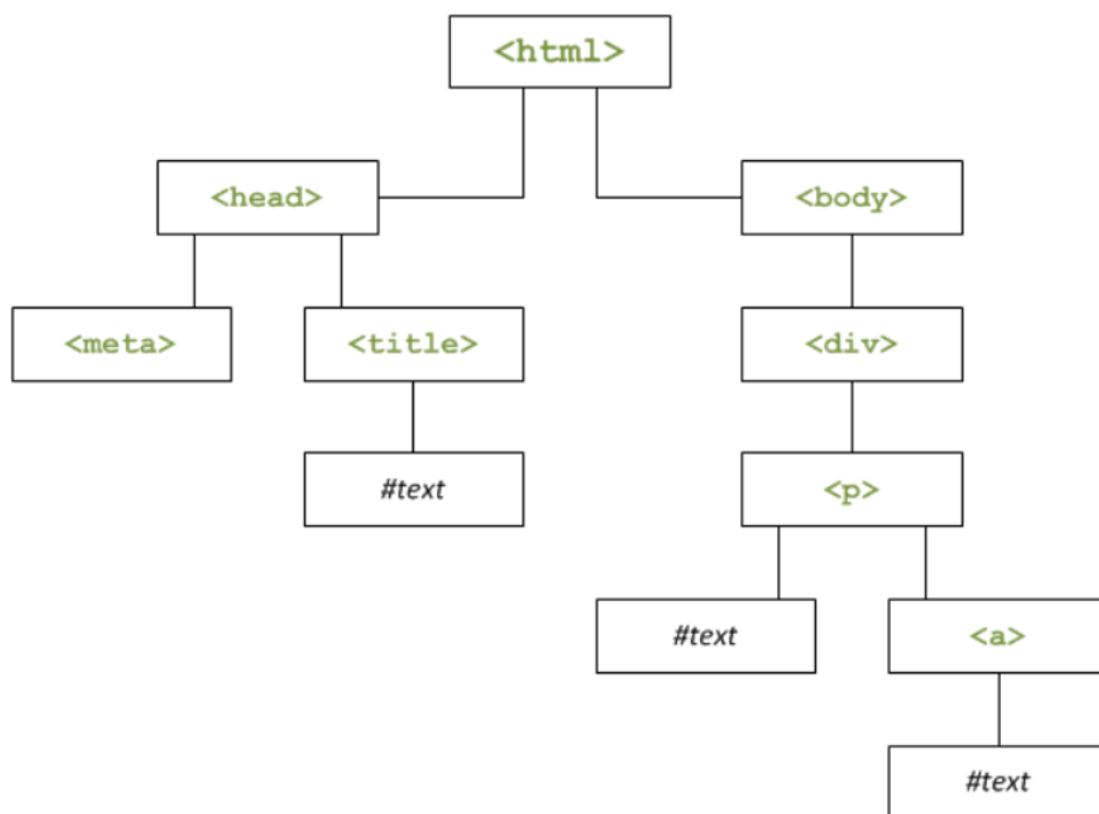
1.	Introduction au <b>DOM</b> .....	2
2.	L'objet <b>DOCUMENT</b> .....	3
3.	Accéder aux éléments .....	4
3.1	Méthode <b>document.getElementById</b> .....	4
3.2	Méthode <b>document.getElementsByTagName</b> .....	5
3.3	Méthode <b>document.getElementsByName</b> .....	6
3.3	Méthode <b>document.getElementsByClassName</b> .....	6
4.	Lire et modifier le contenu des nœuds .....	7
4.1	Propriété <b>innerHTML</b> .....	7
4.2	Propriété <b>textContent</b> .....	8
5.	Deux nouvelles méthodes de sélection : <b>querySelector</b> .....	9
	Choix de la méthode de sélection .....	10
6.	Les propriétés d'attributs et de classe .....	11
	Accéder à la valeur d'attribut : <b>getAttribute()</b> .....	11
	Certains attributs sont directement accessibles : <b>id</b> , <b>href</b> et <b>value</b> .....	11
	Pour vérifier la présence d'un attribut sur un élément : <b>hasAttribute()</b> .....	11
	Pour récupérer la liste des classes d'un élément du DOM : <b>classList</b> .....	11
	Pour tester la présence d'une classe sur un élément : <b>contains()</b> .....	11

# 1. INTRODUCTION AU DOM

---

Une page web n'est rien d'autre qu'un ensemble de balises imbriquées les unes dans les autres. On peut la représenter sous une forme hiérarchisée appelée une arborescence.

L'élément `<html>` en constitue la racine et contient deux éléments : `<head>` et `<body>`, qui contiennent eux-mêmes plusieurs sous-éléments :



---

Une page Web peut être vue comme un arbre

Chaque entité de l'arborescence est appelée un **nœud**. On distingue deux types de nœuds :

- Ceux qui correspondent à des éléments HTML, comme `<body>`, `<p>` ou `<a>`. Ces nœuds peuvent avoir des sous-nœuds, appelés fils ou enfants(children).
- Les nœuds textuels. Ceux-ci ne peuvent pas avoir de fils.

Cette représentation de la structure d'une page web offerte par un navigateur et exploitable via JavaScript est appelée **DOM**, pour **Document Object Model**.

Le DOM représente une page web sous la forme d'une hiérarchie d'objets, où chaque objet correspond à un nœud de l'arborescence de la page.

Les objets du DOM disposent de propriétés et de méthodes permettant de les manipuler avec JavaScript.

## 2. L'OBJET DOCUMENT

---

Une page web n'est rien d'autre qu'un ensemble

Lorsqu'un programme JavaScript s'exécute dans le contexte d'un navigateur web, il peut accéder à la racine du DOM en utilisant la variable **document**.

La variable document correspond à l'élément **<html>**.

Cette variable est un objet et dispose des propriétés **head** et **body** qui permettent d'accéder respectivement aux éléments **<head>** et **<body>** de la page.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Le titre de la page</title>
</head>
<body>
  <div>
    <p>Un peu de texte et <a>un lien</a></p>
  </div>
</body>
</html>
```

Dans cet exemple, l'élément **<html>** contient deux éléments, appelés **enfants** : **<head>** et **<body>**. Pour ces deux enfants, **<html>** est l'élément **parent**.

Chaque élément est appelé **nœud** (node en anglais).

Le nœud **<head>** contient lui aussi 3 enfants : deux **<meta>** et un **<title>**.

**<meta>** ne contient pas de nœud enfant tandis que **<title>** en contient un qui s'appelle **#text**. Un **#text** est un simple élément texte qui ne peut contenir des nœuds enfant.

### 3. ACCEDER AUX ELEMENTS

---

L'objet document possède nativement 3 méthodes principales :

- `document.getElementById()`
- `document.getElementsByTagName()`
- `document.getElementsByName()`

Récemment, ont été ajoutés 2 méthodes supplémentaires :

- `document.querySelector()`
- `document.querySelectorAll()`

#### 3.1 Méthode `document.getElementById`

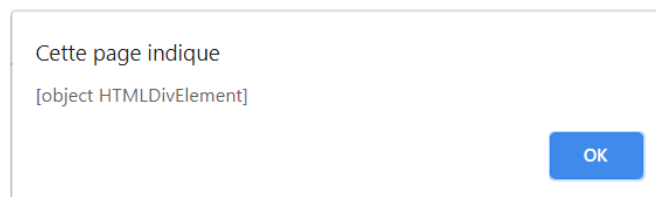
---

Cette méthode permet d'accéder directement à un élément en connaissant son ID :

```
<div id="myDiv">
  <p>Un peu de texte et <a>un lien</a></p>
</div>

<script>
  let div = document.getElementById('myDiv');
  alert(div);
  console.log(div);
</script>
```

On obtient :



Ce qui nous dit qu'on a bien récupéré un élément de type HTMLDivElement 😊

## 3.2 Méthode `document.getElementsByTagName`



**Attention à cette méthode : il y a un « s » à Elements.**

Cette méthode permet de récupérer dans un tableau, tous les éléments d'une même balise html. Exemple, pour récupérer toutes les `<div>` :

```
<div id="myDiv">
  <p>Un peu de texte et <a>un lien</a></p>
  <p>bla bla bla</p>
</div>
<p>bla bla bla</p>
<p>bla bla bla</p>

<script>
  let parags = document.getElementsByTagName('p');
  for (let i = 0; i < parags.length; i++) {
    console.log("Element n°", i, ":" + parags[i]);
  }
</script>
```

La méthode retourne bien un tableau d'éléments : il est nécessaire de parcourir le tableau avec une boucle !

On peut appliquer cette méthode sur n'importe quel élément HTML, pas seulement sur l'objet document.

```
let div = document.getElementById('myDiv');
let parags = div.getElementsByTagName('p');
for (let i = 0; i < parags.length; i++) {
  console.log("Element n°", i, ":" + parags[i]);
}
```

On peut aussi récupérer toutes les balises du document ou d'une zone avec \*

```
let div = document.getElementById('myDiv');
let items = div.getElementsByTagName('*');
for (let i = 0; i < items.length; i++) {
  console.log("Element n°", i, ":" + items[i]);
}
```

### 3.3 Méthode `document.getElementsByName`



Ici aussi : il y a un « s » à Elements.

Cette méthode est semblable à `getElementsByTagName` et permet de récupérer les éléments qui possèdent l'attribut `name` spécifié.

L'attribut `name` n'est utilisé qu'au sein des formulaires et est déprécié depuis la spécification HTML5 dans tout autre élément que celui d'un formulaire. Donc, on l'utilise au sein d'un `<input>` mais pas pour un élément `<map>` ...

```
<form>
  <input type="radio" name="choix" id="choix1" />
  <input type="radio" name="choix" id="choix2" checked="true" />
  <input type="radio" name="choix" id="choix3" />
</form>
<script>
  let items = document.getElementsByName('choix');
  for (let i = 0; i < items.length; i++) {
    console.log("Radio bouton n°", i, ":" + items[i].checked);
  }
</script>
```

### 3.3 Méthode `document.getElementsByClassName`



Ici aussi : il y a un « s » à Elements.

Cette méthode fonctionne de la même manière, mais permet cette fois de récupérer les éléments par un nom de classe :

```
<div id="myDiv">
  <p class="bleu">Un peu de texte et <a>un lien</a></p>
  <p class="rouge">bla bla bla</p>
</div>
<p class="bleu">bla bla bla</p>
<p class="bleu">bla bla bla</p>

<script>
  let items = document.getElementsByClassName('bleu');
  for (let i = 0; i < items.length; i++) {
    console.log("Element n°", i, ":" + items[i]);
  }
</script>
```

## 4. LIRE ET MODIFIER LE CONTENU DES NŒUDS

---

En Javascript, on utilise principalement 2 méthodes :

- `innerHTML` : pour manipuler du contenu Html
- `textContent` : pour manipuler du contenu texte

### 4.1 Propriété `innerHTML`

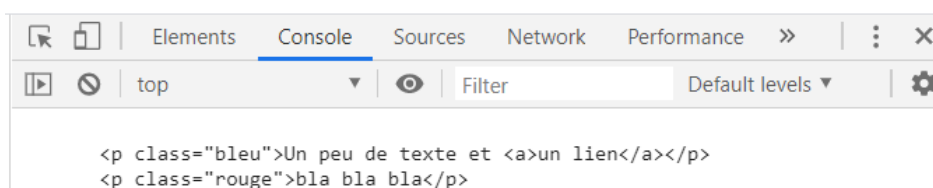
---

La propriété `innerHTML` a été créée par Microsoft pour Internet Explorer puis a été normalisé au sein du HTML5 : elle est devenue un standard parce que tous les navigateurs la supportaient déjà !

`innerHTML` permet de récupérer tout le code html enfant d'un élément dans une chaîne :

```
<div id="myDiv">
  <p class="bleu">Un peu de texte et <a>un lien</a></p>
  <p class="rouge">bla bla bla</p>
</div>

<script>
  let div = document.getElementById('myDiv');
  console.log(div.innerHTML);
</script>
```



Cette propriété fonctionne dans les 2 sens : en lecture et en écriture.

```
let div = document.getElementById('myDiv');
div.innerHTML = "<blockquote>Une citation à la place du paragraphe</blockquote>";
```

Dans ce cas, on remplace le contenu.

Si on veut ajouter au lieu de remplacer, on utilise la concaténation `+=`

```
let div = document.getElementById('myDiv');
div.innerHTML += "<strong>Ajout d'une portion mise en emphase</strong>";
```

## 4.2 Propriété `textContent`

La propriété `innerText`, créée par Microsoft pour Internet Explorer, n'a pas été standardisée et n'est pas supportée par tous les navigateurs.

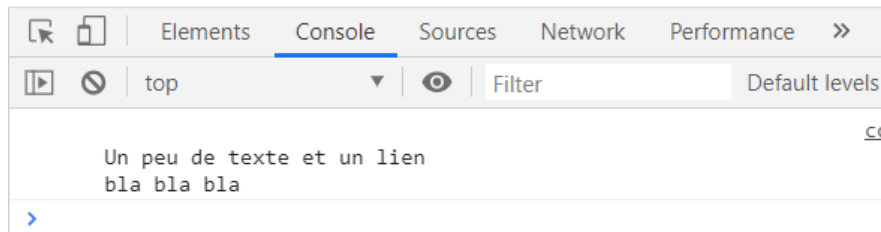
La propriété `textContent` est la version standardisée d'`innerText` et reconnue par tous les navigateurs.

Ces deux propriétés fonctionnent de la même manière que `innerHTML`, excepté le fait que seul le text est récupéré et non les balises.

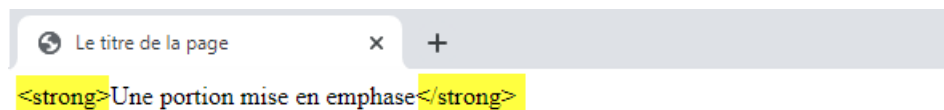
```
<div id="myDiv">
  <p class="bleu">Un peu de texte et <a>un lien</a></p>
  <p class="rouge">bla bla bla</p>
</div>

<script>
  let div = document.getElementById('myDiv');
  console.log(div.textContent);
</script>
```

Les balises ne sont pas interprétées, cette fois, par la page :



Ni en lecture, ni en écriture :





## 5. DEUX NOUVELLES METHODES DE SELECTION : QUERYSELECTOR

---

Les 4 méthodes natives vues jusqu'ici sont performantes mais limitées.

Avec des pages ou applications complexes, elles se révèlent peu souples car nécessitent de faire appel à des boucles, diverses opérations algorithmiques et des filtres sur les résultats obtenus pour pouvoir obtenir une liste d'éléments.

Avec deux nouvelles fonctions introduites par l'[API Selectors](#), la syntaxe permet de faire appel aux sélecteurs CSS, généralement à partir de la racine document.

- `querySelector()` : retourne le premier élément trouvé satisfaisant au sélecteur
- `querySelectorAll()` : retourne tous les éléments satisfaisant au sélecteur

**Le sélecteur comprend tous les sélecteurs CSS 😊**

Exemples `querySelectorAll()`:

```
// Tous les paragraphes
console.log(document.querySelectorAll("p").length); // Affiche 3

// Tous les paragraphes à l'intérieur de l'élément identifié par "contenu"
console.log(document.querySelectorAll("#contenu p").length); // Affiche 2

// Tous les éléments ayant la classe "existe"
console.log(document.querySelectorAll(".existe").length); // Affiche 8
```

```
// Tous les éléments fils de l'élément identifié par "antiques" ayant la classe "existe"
console.log(document.querySelectorAll("#antiques > .existe").length); // Affiche 1
```

Exemples `querySelector()`:

Identique à `querySelectorAll()` mais renvoie uniquement le 1<sup>er</sup> élément correspondant :

```
// Le premier paragraphe
console.log(document.querySelector("p"));
```

Un article intéressant sur le sujet :

<https://www.alsacreations.com/article/lire/1445-dom-queryselector-queryselectorall-selectors-api.html>

## Choix de la méthode de sélection

---

En théorie, il serait possible d'utiliser systématiquement les méthodes `querySelectorAll` et `querySelector`.

Cependant, celles-ci souffrent d'un déficit de performances par rapport aux méthodes `getElementsByTagName`, `getElementsByClassName` et `getElementById`.

Approche pragmatique conseillée :

Nombre d'éléments à obtenir	Critère de sélection	Méthode à utiliser
Plusieurs	Par balise	<code>getElementsByTagName</code>
Plusieurs	Par classe	<code>getElementsByClassName</code>
Plusieurs	Autre que par balise ou par classe	<code>querySelectorAll</code>
Un seul	Par identifiant	<code>getElementById</code>
Un seul (le premier)	Autre que par identifiant	<code>querySelector</code>

Un benchmark intéressant sur le sujet : <https://jsperf.com/jquery-selectors-vs-native-api>

## 6. LES PROPRIETES D'ATTRIBUTS ET DE CLASSE

---

Il est souvent très utile, en plus que de manipuler le contenu textuel ou html d'un nœud, de pouvoir accéder aux attributs d'un nœud.

C'est ce que permettent les fonctions ci-dessous.

Accéder à la valeur d'attribut : **getAttribute()**

---

```
// L'attribut href du premier lien
console.log(document.querySelector("a").getAttribute("href"));
```

Certains attributs sont directement accessibles : **id**, **href** et **value**

---

```
// L'identifiant de la première liste
console.log(document.querySelector("ul").id);

// L'attribut href du premier lien
console.log(document.querySelector("a").href);
```

Pour vérifier la présence d'un attribut sur un élément : **hasAttribute()**

---

```
if (document.querySelector("a").hasAttribute("target")) {
    console.log("Le premier lien possède l'attribut target");
} else {
    console.log("Le premier lien ne possède pas l'attribut target");
}
```

Pour récupérer la liste des classes d'un élément du DOM : **classList**

---

Dans une page web, une balise peut posséder plusieurs classes.

```
// Liste des classes de l'élément identifié par "antiques"
var classes = document.getElementById("antiques").classList;
console.log(classes.length); // Affiche 1 : l'élément possède une seule classe
console.log(classes[0]); // Affiche "merveilles"
```

Pour tester la présence d'une classe sur un élément : **contains()**

---

```
if (document.getElementById("antiques").classList.contains("merveille")) {
    console.log("L'élément identifié par antiques possède la classe merveille");
} else {
    console.log("L'élément identifié par antiques ne possède pas la classe merveille");
}
```