Web Developer
Scripts Clients

09 – Le DOM

A. Martel 2025-2026

9.1. DOM – Introduction

Le Document Object Model est une API (Application Programming Interface) de manipulation des documents XML et HTML.

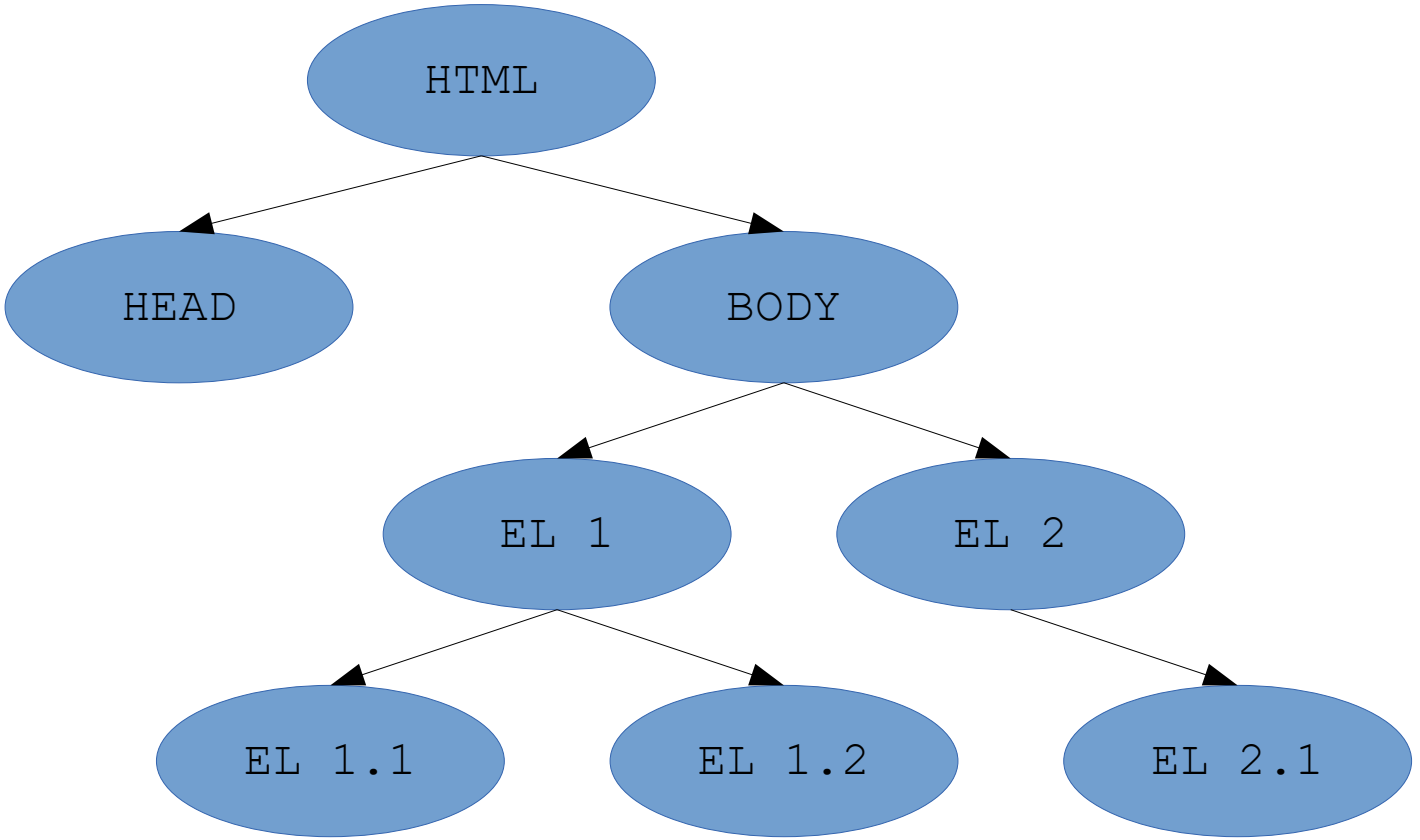
Grâce à lui, nous pourrons accéder à n'importe quelle partie de notre page web pour :

- Ajouter un élément
- Modifier un élément
- Supprimer un élément

Ces éléments peuvent être des balises HTML et/ou des propriétés CSS.

9.1. DOM – Introduction

Le DOM représente la page sous-forme d'un arbre d'éléments :



9.2. DOM – Window

L'objet « racine » du DOM est l'objet window.

Il représente la fenêtre du navigateur.

C'est dans cet objet que sont notamment stockées les variables et méthodes globales.

9.2. DOM – Window

Exemple 1 (portée des variables var **via** window**) :**

```
var valeur = 'globale';

(function () {
  var valeur = 'locale';

  console.log('Accès en local : ' + valeur);
  console.log('Accès en global : ' + window.valeur);
})();
```

9.2. DOM – Window

Exemple 1 (portée des variables let **via** window**) :**

```
let valeur = 'globale';

(function () {
  let valeur = 'locale';

  console.log('Accès en local : ' + valeur);
  console.log('Accès en global : ' + window.valeur);
})();
```

Une variable let **n'est pas accessible en global.**

9.2. DOM – Window

Exemple 2 (portée des variables via window **– utilité du mot-clé** var**) :**

```
(function () {
   valeur = 'locale';
   console.log('Accès en local : ' + valeur);
})();

console.log('Accès en global : ' + window.valeur);
```

9.2. DOM – Window

Exemple 2 (portée des variables via window **– utilité du mot-clé** var**) :**

```
(function () {
   var valeur = 'locale';
   console.log('Accès en local : ' + valeur);
})();

console.log('Accès en global : ' + window.valeur);
```

9.3. DOM – Document

L'objet document, objet enfant de window, représente la balise html de la page web.

C'est donc au travers de cet objet que nous pourrons accéder à tous les éléments de la page.

Il existe cinq méthodes d'accès aux éléments d'une page :

1. getElementById : très souvent utilisé
2. getElementsByTagName : rarement, voire jamais plus utilisé
3. getElementsByName : rarement, voire jamais plus utilisé
4. querySelector : recommandé pour un seul élément
5. querySelectorAll : recommandé pour un tableau d'éléments

9.3. DOM – Document

La première méthode d'accès à un élément est getElementById à laquelle on passe en paramètre l'id de l'élément voulu :

```
// Exemple pour accéder
let le_contenu = document.getElementById('contenu');

console.log(le_contenu);
```

Cette méthode retourne un objet de type « element ».

9.3. DOM – Document

L'objet « element » comprend les propriétés suivantes (liste non-exhaustive) :

- attributes
- childNodes
- id
- innerHtml
- name
- textContent

9.3. DOM – Document

La deuxième méthode d'accès à un élément est getElementsByTagName à laquelle on passe en paramètre soit le type des éléments voulus (une balise par exemple), soit une * pour tous les éléments :

```
// Sélection de tous les éléments de type DIV
let elements = document.getElementsByTagName('div');
let element;

// Boucle, élément par élément, sur tous les DIV sélectionnés
for (element of elements) {
   console.log(element);
}
```

9.3. DOM – Document

Il est également possible d'appeler cette méthode sur un autre élément que document.

```
let elements = document.getElementsByTagName('div');
let element;

for (element of elements) {
   let sous_elements = element.getElementsByTagName('*');
   let sous_element;
   for (sous_element of sous_elements) {
      console.log(sous_element);
   }
}
```

9.3. DOM – Document

La troisième méthode d'accès à un élément est getElementsByName à laquelle on passe en paramètre la valeur d'un attribut name. Par conséquent, elle ne devrait être utilisée qu'en vue de sélectionner des éléments d'un formulaire.

9.3. DOM – Document

La quatrième méthode d'accès à un élément est querySelector à laquelle on passe un sélecteur de type « CSS3 ». Retourne le premier élément correspondant au sélecteur passé en paramètre.

```
let element = document.querySelector('.test1');

console.log(element.textContent);
```

9.3. DOM – Document

La cinquième méthode est querySelectorAll qui retourne tous les éléments qui correspondent au sélecteur passé en paramètre.

```
let elements = document.querySelectorAll('.test1');
let element;

for (element of elements) {
   console.log(element.textContent);
}
```

9.4. DOM – Exercices

Exercice 1

Essayez d'accéder à différents éléments en fonction de sélecteurs CSS :

- Balise
- Classe
- Identifiant
- Balise imbriquée

9.5. DOM – Contenu

On peut maintenant accéder en lecture aux éléments de notre page.

Voyons comment y accéder en écriture.

9.5. DOM – Contenu HTML

Pour modifier le code HTML d'un élément, nous utiliserons la propriété innerHTML :

```
let element = document.querySelector('#id1');

element.innerHTML = '<strong>Nouveau contenu</strong>';
```

9.5. DOM – Contenu HTML

Attention lors de l'utilisation de cette méthode, car :

- Elle interprète et exécute le HTML inséré (y compris du JS via les événements) → c'est ce qu'on appelle une faille XSS (Cross-Site Scripting)
- Elle supprime les event listeners (que nous verrons plus tard) existants → tous les anciens nœuds sont détruits ainsi que tous les événements attachés
- Elle est peu performante pour les grosses structures :
  - Détruit l'ancien contenu
  - Crée l'intégralité des nouveaux nœuds
  - Force un recalcul du layout
  - → Inefficace si seule une petite partie du DOM doit changer

9.5. DOM – Contenu HTML

Il est préférable, pour ajouter du HTML sans effacer les events listeners, d'utiliser la méthode insertAdjacentHTML.

Elle prend deux paramètres :

1) La position à laquelle ajouter un élément :
    1) beforebegin : avant l'élément
    2) afterbegin : au début de l'élément (comme prepend)
    3) beforeend : à la fin de l'élément (comme append)
    4) afterend : après l'élément
2) Le code HTML à ajouter

NB : Comme innerHTML, insertAdjacentHTML ne doit jamais recevoir un texte provenant de l'utilisateur ou d'une API non maîtrisée

9.5. DOM – Contenu HTML

beforebegin

<element>

  afterbegin

  [contenu initial]

  beforeend

</element>

afterend

9.5. DOM – Contenu HTML

Exemple pour ajouter un élément en fin d'une liste :

```
liste.insertAdjacentHTML("beforeend", "<li>Nouvel élément</li>");
```

9.5. DOM – Contenu

Pour modifier le texte d'un élément, nous utiliserons la propriété textContent :

```
let element = document.querySelector('#id1');
let new_content = '<strong>Nouveau contenu</strong>';

element.textContent = new_content;
```

9.6. DOM – Attribut

Pour accéder à un attribut d'un élément, nous utiliserons les méthodes getAttribute et setAttribute :

```
let element = document.querySelector('#id1');

element.setAttribute('class', 'rouge');
// mais
element.className = 'rouge';
// et non pas element.class car class est un mot réservé en JS
```

Rem : il existe un autre attribut qui est un mot réservé en JS, il s'agit de for. Pour l'utiliser en propriété, vous écrirez, par exemple, element.htmlFor

9.6. DOM – Attribut

Pour manipuler les classes, le plus efficace est d'utiliser l'attribut classList qui contient plusieurs méthodes :

- add : permet d'ajouter une classe à la liste
- remove : permet de retirer une classe de la liste
- toggle : permet de retirer une classe si elle est présente ou de l'ajouter si elle ne l'est pas
- contains : permet de vérifier si une classe est présente dans la liste

9.6. DOM – Attribut

Dans un formulaire, vous pouvez accéder à la valeur d'un champ grâce à la propriété value :

```
let nom = document.getElementById('nom').value;
```

9.7. DOM – Exercices

Exercice 2

Vous devez créer un formulaire reprenant le nom, le prénom, la date de naissance et l'email d'un élève.

Si une valeur n'est pas valide (à contrôler à l'aide de regex : longueurs du nom et du prénom, format date et email), vous afficherez un message (via console.log).

Lorsque toutes les valeurs sont valides, vous sauvegardez celles-ci dans un objet Eleve que vous afficherez ensuite dans la console.

9.7. DOM – Exercices

Exercice 3

Idem Exercice 2, mais :

- N'utilisez plus de « console.log »
- Modifiez l'aspect du champ qui est en erreur
- Affichez le message d'erreur à côté du champ

9.8. DOM – Nœuds

Avec les propriétés suivantes, nous allons voir comment nous déplacer à partir d'un élément de référence, mais également comment ajouter, déplacer ou supprimer un élément.

Précision importante, le DOM est constitué de nœuds dont il existe plusieurs types, par exemple :

Nom (nodeName)	Valeur (nodeType)
ELEMENT_NODE	1
TEXT_NODE	3
COMMENT_NODE	8

9.8. DOM – Nœuds

Dès lors, on comprend que tous les éléments sont des nœuds, mais que tous les nœuds ne sont pas des éléments.

9.8. DOM – Nœuds

La propriété parentNode permet d'accéder au parent de l'élément courant :

```
let enfant = document.getElementById('enfant_id');
let parent = enfant.parentNode;


console.log(parent.id);
```

Les propriétés nodeName et nodeType permettent d'obtenir des informations sur le type d'élément.

Le type étant un code, voir la documentation Mozilla pour plus d'informations

9.8. DOM – Nœuds

La méthode qui permet de déterminer si un élément possède un (ou des) enfant est hasChildNodes **:**

```
let parent = document.getElementById('parent_id');
let a_des_enfants = parent.hasChildNodes();

console.log(a_des_enfants);
```

9.8. DOM – Nœuds

Les propriétés firstChild et lastChild permettent d'accéder respectivement au premier et dernier enfant de l'élément courant :

```
let parent = document.getElementById('parent_id');
let premier_enfant = parent.firstChild;
let dernier_enfant = parent.lastChild;

console.log(premier_enfant.id);
console.log(dernier_enfant.id);
```

Avec ces propriétés, on accède aux nœuds, quel qu'en soit le type. Pour n'accéder qu'aux éléments HTML, on utilisera les propriétés firstElementChild et lastElementChild.

9.8. DOM – Nœuds

La propriété childNodes permet de récupérer tous les enfants de l'élément courant :

```
let parent = document.getElementById('parent_id');
let enfants = parent.childNodes;

console.log(premier_enfant.id);
console.log(dernier_enfant.id);
```

9.8. DOM – Nœuds

Les propriétés previousSibling et nextSibling permettent d'accéder au nœud qui précède ou qui suit l'élément courant :

```
let enfant2 = document.getElementById('enfant_2_id');
let enfant1 = enfant2.previousSibling;
let enfant3 = enfant2.nextSibling;

console.log(enfant1.id);
console.log(enfant3.id);
```

9.8. DOM – Nœuds

Pour accéder à l'élément HTML qui précède ou qui suit, on utilisera les propriétés suivantes :

- previousElementSibling
- nextElementSibling

9.9. DOM – Ajout d'un nœud

Pour ajouter un nœud, nous devons respecter la démarche suivante : création, définition et ajout. C'est la méthode la plus sûre pour éviter les failles XSS.

La méthode, de document, d'ajout d'un nœud de type élément est createElement et celle d'ajout d'un nœud de type texte est createTextNode.

```
// Sélection de l'élément cible
let element_cible = document.getElementById('parent_id');
// Création du nouvel élément
let nouvel_element = document.createElement('h1');
// Création du texte de ce nouvel élément
let texte_nouvel_element = document.createTextNode('Titre du parent');
```

9.9. DOM – Ajout d'un nœud

L'élément étant créé, il faut l'ajouter à l'élément parent.

La première méthode d'ajout d'un nœud est appendChild qui ajoutera le nouveau nœud à la fin du parent :

```
// Ajout du texte au nouvel élément
nouvel_element.appendChild(texte_nouvel_element);

// Ajout du nouvel élément dans l'élément cible
element_cible.appendChild(nouvel_element);
```

9.9. DOM – Ajout d'un nœud

La seconde méthode d'ajout est insertBefore qui ajoutera le nouveau nœud avant un nœud cible du parent :

```
// Ajout du texte au nouvel élément
nouvel_element.appendChild(texte_nouvel_element);

// Ajout du nouvel élément dans l'élément cible avant
// un repère, qui est ici le premier nœud enfant
element_cible.insertBefore(nouvel_element, element_cible.firstChild);
```

9.9. DOM – Ajout d'un nœud

En résumé, pour ajouter du contenu à un nœud, voici les méthodes selon les cas d'utilisation :

- Pour du texte : textContent
- Pour ajouter du contenu : insertAdjacentHTML
- Pour remplacer complètement un « petit » bloc sûr : innerHTML
- Pour contrôler totalement le contenu : createElement, etc.

Attention, ne jamais insérer de HTML provenant de l'utilisateur ou d'une API non contrôlée.

9.10. DOM – Dupliquer un nœud

Pour dupliquer un nœud, on utilisera la méthode cloneNode. On pourra préciser si on veut également dupliquer le contenu du nœud (paramètre à true ou false) :

```
let element_cible = document.getElementById('parent_id');
let nouveau_noeud = element_cible.cloneNode(true);

element_cible.parentNode.appendChild(nouveau_noeud);
```

9.11. DOM – Exercices

Exercice

La méthode d'ajout insertAfter n'existant pas en JS, il vous est demandé de l'implémenter.

9.12. DOM – Remplacer un nœud

Pour remplacer un nœud, on utilisera la méthode replaceChild. En paramètre on trouvera le nouveau nœud suivi du nœud à remplacer :

```
let element_cible = document.getElementById('parent_id');
let nouvel_element = document.createElement('div');
let texte_nouvel_element = document.createTextNode('Je suis le nouvel élément');

nouvel_element.appendChild(texte_nouvel_element);
element_cible.parentNode.replaceChild(nouvel_element, element_cible);
```

9.13. DOM – Supprimer un nœud

Pour supprimer un nœud, on utilisera la méthode removeChild. En paramètre on trouvera le nœud à supprimer :

```
let element_cible = document.getElementById('enfant_1_id');

element_cible.parentNode.removeChild(element_cible);
```

Rem : cette méthode retourne le nœud supprimé. Ce qui signifie qu'on pourrait le récupérer pour l'insérer ailleurs par exemple.

9.14. DOM – Exercices

2) Vous devez afficher dans la console l'arborescence de la page dom-ex-02.html (voir Moodle)

3) Vous devez charger le contenu de la page dom-ex-02.html, ajouter un bouton dans le header. Lorsque vous cliquez sur ce bouton, et à l'aide des propriétés et méthodes vues dans ce chapitre, vous transformez le contenu de la page pour qu'il ressemble au rendu de dom-ex-03.html

9.15. DOM – Formulaires

Plusieurs propriétés sont disponibles sur un objet formulaire pour accéder aux valeurs des différents éléments :

- value : valeur d'un input
- disabled : permet de lire ou d'écrire dans l'état actif ou non d'un élément
- checked : valeur d'une checkbox ou d'un radiobutton
- readonly : permet de lire ou d'écrire dans l'état « en lecture seule » d'un élément
- selectedIndex : permet d'accéder à l'index de l'élément sélectionné d'un select
- options : liste des options d'un select

9.15. DOM – Formulaires

Exemple

```
<form onsubmit="return testForm()">
 <input id="texte" type="text" /><br/>
 <label>Test checkbox</label><input id="check" type="checkbox"/><br/>
 <label>Test radio</label><input id="radio" type="radio"
value="radio_ok"/><br/>
 <select id="selection">
  <option value="valeur1">Choix 1</option>
  <option value="valeur2">Choix 2</option>
  <option value="valeur3">Choix 3</option>
 </select><br/>
 <input type="submit" value="Test"/>
</form>
```

9.15. DOM – Formulaires

Exemple

```
function testForm() {
    let texte = document.getElementById('texte');
    let check = document.getElementById('check');
    let radio = document.getElementById('radio');
    let selection = document.getElementById('selection');

    console.log(texte.value);
    console.log(check.checked);
    console.log(radio.checked);
    console.log(selection.selectedIndex);

    return false;
}
```

9.16. DOM – Exercices

Exercice 4

Tester les propriétés disabled et readonly, en lecture et en écriture, et options pour afficher toutes les options d'un select et pour en ajouter une.