HTML DOM

Document Object Model

Qu'est-ce que le DOM ?

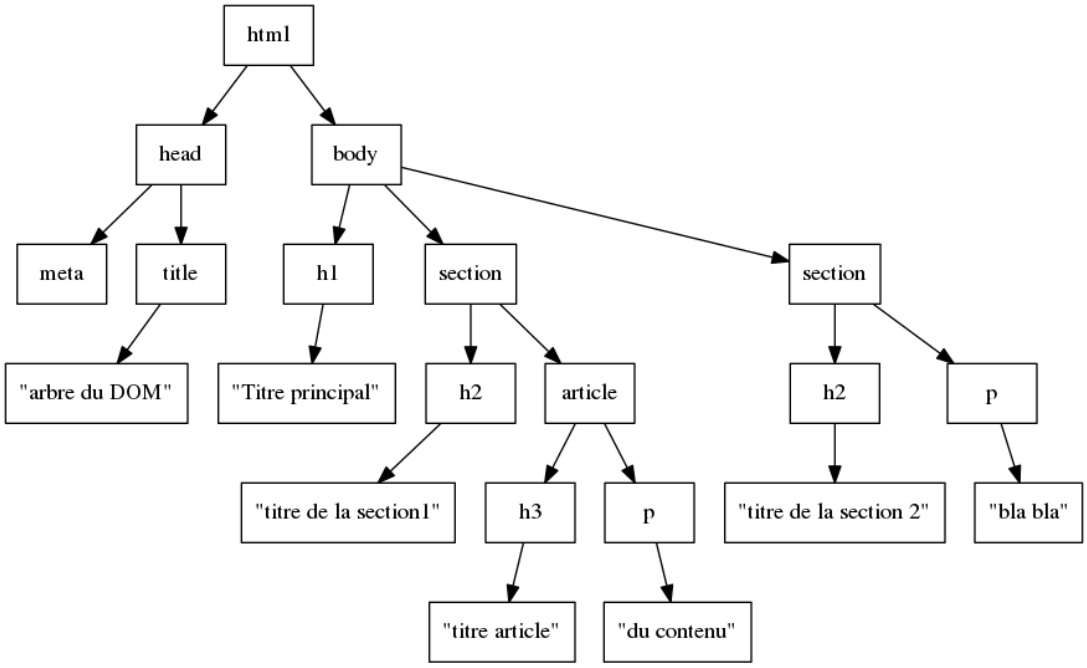
Le **DOM** (*Document Object Model*, modèle objet de document) est la représentation qu'un navigateur construit en mémoire à partir d'une page HTML : un **arbre** dont chaque balise, chaque texte et chaque attribut devient un objet. C'est par cet arbre qu'un programme JavaScript lit la page affichée et la modifie. Le DOM n'est pas un langage de programmation : c'est une interface, identique dans tous les navigateurs, que JavaScript utilise pour agir sur la page.

L'arbre d'une page

Soit le fichier HTML ci-dessous.

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title> arbre du DOM </title>
    </head>
    <body>
        <h1> Titre principal </h1>
        <section>
            <h2> titre de la section 1 </h2>
            <article>
                <h3> titre article </h3>
                <p> du contenu </p>
            </article>
        </section>
        <section>
            <h2> titre de la section 2 </h2>
            <p> bla bla </p>
        </section>
    </body>
</html>
```

L'arbre associé à cette page est le suivant :



Chaque case de cet arbre est un **nœud**. Un document HTML contient trois sortes de nœuds :

— les **éléments**, un par balise (`html`, `body`, `h1`, `p`…) ;

— les **textes**, écrits entre les balises ;

— les **attributs** (`class`, `id`, `src`…), rattachés à leur élément.

On emploie le vocabulaire des arbres généalogiques : `html` est la **racine** ; `head` et `body` sont ses **enfants** ; `body` est le **parent** de `h1` et des deux `section` ; les deux `section` sont des **frères**.

Exemple 1. Dans la construction `<h1 class="centre"> Bonjour <i> tout le monde </i> ! </h1>`, l'élément `h1` a trois enfants : le texte « Bonjour », l'élément `i` (qui a lui-même un enfant, le texte « tout le monde ») et le texte « ! ». L'attribut `class` appartient à `h1` mais n'est pas un enfant dans l'arbre.

Nous allons dans la suite écrire des scripts en JavaScript qui utilisent le DOM pour lire et modifier la page où ils se trouvent.

Premiers pas en JavaScript

JavaScript est le langage de programmation des navigateurs. Dans une page HTML, un script s'écrit entre les balises `<script>` et `</script>`, placées après les éléments qu'il manipule (en fin de `body`) pour que ces éléments existent déjà quand le script s'exécute. Sa syntaxe diffère de celle de Python sur les points suivants :

— les blocs sont délimités par des accolades `{ }` et non par l'indentation, qui ne sert qu'à la lisibilité ; chaque instruction se termine par un point-virgule ;

— une variable se déclare avec `let` si sa valeur doit changer, ou avec `const` si elle ne sera jamais réaffectée ;

— `console.log(...)` affiche une valeur dans la console du navigateur (outils de développement, touche F12), comme `print` en Python ;

— `Number("12")` convertit une chaîne en nombre, comme `int` ou `float` en Python, et `+` entre deux chaînes les met bout à bout ;

— une fonction se définit avec le mot `function` ; elle peut aussi être écrite sans nom, directement là où on la transmet (`function (event) { ... }`), ce que l'on fera avec les événements.

Python	JavaScript	Rôle
`x = 0`	`let x = 0;`	variable dont la valeur peut changer
`TAUX = 20`	`const TAUX = 20;`	variable qui ne sera pas réaffectée
`True`, `False`	`true`, `false`	booléens
`a == b`	`a === b`	égalité
`a != b`	`a !== b`	différence
`a and b`	`a && b`	ET logique
`a or b`	`a || b`	OU logique
`not a`	`!a`	NON logique
`if ...:` puis `else:`	`if (...) { ... } else { ... }`	instruction conditionnelle
`for x in liste:`	`for (const x of liste) { ... }`	parcours d'une liste
`def f(a):`	`function f(a) { ... }`	définition de fonction
`print(x)`	`console.log(x);`	affichage

JavaScript possède aussi `==` et `!=`, qui convertissent les valeurs avant de les comparer (`"1" == 1` vaut `true`) : on utilise toujours `===` et `!==`, qui comparent sans conversion (`"1" === 1` vaut `false`).

```javascript
const ages = [8, 15, 70];
let total = 0;
for (const age of ages) {
  if (age < 12 || age >= 65) {
    total = total + 5;
  } else {
    total = total + 10;
  }
}
console.log("Prix total : " + total);   // affiche Prix total : 20

function peutEntrer(age, aUnBillet) {
  return age >= 18 && aUnBillet;
}
const reponse = "oui";
if (reponse === "oui" && !peutEntrer(16, true)) {
  console.log("Il faut être majeur.");  // affiché, car peutEntrer(16, true) vaut false
}
if (reponse !== "non") {
  console.log("Réponse enregistrée.");  // affiché
}
```

Accéder aux éléments

L'objet `document` représente la page entière. Quatre de ses méthodes permettent d'atteindre un ou plusieurs éléments.

— `document.getElementById("intro")` renvoie l'élément dont l'attribut `id` vaut `intro` (un identifiant est unique dans la page) ;

— `document.querySelector(".machin")` renvoie le **premier** élément qui correspond à un sélecteur CSS (`#truc`, `.machin`, `p`, `nav a`...) ;

— `document.querySelectorAll("p")` renvoie **tous** les éléments qui correspondent au sélecteur, dans une liste (`NodeList`) que l'on parcourt avec `for ... of` ;

— `document.getElementsByTagName("a")` renvoie la collection des éléments portant ce nom de balise ; appliquée à un élément (`elt.getElementsByTagName("a")`), elle ne renvoie que ses descendants.

```
const intro = document.getElementById("intro");
const premierLien = document.querySelector("nav a");
for (const p of document.querySelectorAll("p")) {
  console.log(p.textContent);
}
```

Une fois l'élément obtenu, on lit son contenu et ses attributs :

— `elt.textContent` : le texte contenu dans l'élément (sans les balises) ;

— `elt.innerHTML` : le code HTML contenu dans l'élément ;

— `elt.getAttribute("href")` : la valeur d'un attribut ; les attributs courants sont aussi des propriétés (`elt.id`, `elt.src`, `elt.value` pour un champ de saisie).

Se déplacer dans l'arbre

À partir d'un élément `elt`, on atteint ses voisins sans connaître leur identifiant :

— `elt.children` : la liste de ses éléments enfants ; `elt.firstElementChild` et `elt.lastElementChild` : le premier et le dernier ;

— `elt.childNodes` : la liste de **tous** ses nœuds enfants, textes compris (les retours à la ligne du code HTML comptent comme des nœuds texte) ;

— `elt.parentNode` : son parent ;

— `elt.nextElementSibling` et `elt.previousElementSibling` : ses frères suivant et précédent.

Deux propriétés renseignent sur la nature d'un nœud : `nodeName` renvoie son nom (`"H1"`, `"#text"`...) et `nodeType` un code numérique.

Type de nœud	`nodeType`
élément	1
texte	3
commentaire	8
document	9

Modifier la page

Le DOM permet surtout d'**écrire** dans la page. Tout changement est immédiatement affiché.

— `elt.textContent = "Nouveau texte"` remplace le texte ;

— `elt.innerHTML = "<b>gras</b>"` remplace le contenu par du HTML ;

— `elt.setAttribute("src", "photo.png")` modifie un attribut ;

— `elt.style.color = "red"` modifie le style ; `elt.classList.add("important")` ajoute une classe CSS ;

— `document.createElement("li")` crée un élément en mémoire, que `parent.appendChild(nouveau)` insère dans la page ;

— `elt.remove()` retire l'élément de la page.

Exemple 2 (Un compteur).

```
<p id="compteur">0</p>
<button onclick="incrementer()">+1</button>

<script>
  function incrementer() {
    const p = document.getElementById("compteur");
    p.textContent = Number(p.textContent) + 1;
  }
</script>
```

Exemple 3 (Ajouter un élément à une liste).

```
<ul id="courses">
  <li>Pain</li>
</ul>
<input type="text" id="article">
<button onclick="ajouter()">Ajouter</button>

<script>
  function ajouter() {
    const li = document.createElement("li");
    li.textContent = document.getElementById("article").value;
    document.getElementById("courses").appendChild(li);
  }
</script>
```

Réagir aux événements

Un **événement** est un fait qui se produit dans la page : un clic, une touche enfoncée, une saisie dans un champ, la fin du chargement. Un **gestionnaire d'événement** est une fonction que le navigateur exécute quand l'événement se produit. Deux écritures sont possibles.

1. Dans le HTML, avec un attribut `onclick`, `oninput`… : `<button onclick="incrementer()">`. C'est l'écriture des exemples ci-dessus.
2. Dans le script, avec `addEventListener` : le HTML ne contient plus de code, et un même élément peut avoir plusieurs gestionnaires.

```html
<input type="text" id="prenom">
<p id="salut"></p>

<script>
  const champ = document.getElementById("prenom");
  champ.addEventListener("input", function (event) {
    document.getElementById("salut").textContent = "Bonjour " + event.target.value;
  });
</script>
```

La fonction reçoit un objet `event` qui décrit l'événement : `event.target` est l'élément concerné, `event.key` la touche enfoncée, et `event.preventDefault()` annule l'action par défaut (par exemple l'envoi d'un formulaire).

Événement	**Se produit quand...**
`click`	l'utilisateur clique sur l'élément
`input`	le contenu d'un champ de saisie change
`change`	une sélection ou une saisie est validée
`submit`	un formulaire est envoyé
`keydown`	une touche du clavier est enfoncée
`mouseover`	la souris entre dans l'élément
`load`	la page a fini de se charger