

JavaScript 2^{ème} partie

Toujours selon [@MDN](#):

JavaScript (« JS ») est un langage de script léger, orienté objet. Le code JavaScript est **interprété ou compilé à la volée**. C'est un langage à **objets** disposant d'un **typage faible** et **dynamique**.

Rappel: Dans cet exemple on instancie une variable appelée `car` en lui affectant un objet, puis on lui affecte une chaîne de caractères.

```
let car = {  
  brand: "Reliant"  
  year: 1962  
};  
  
car = "Reliant";
```

Document Object Model (DOM)

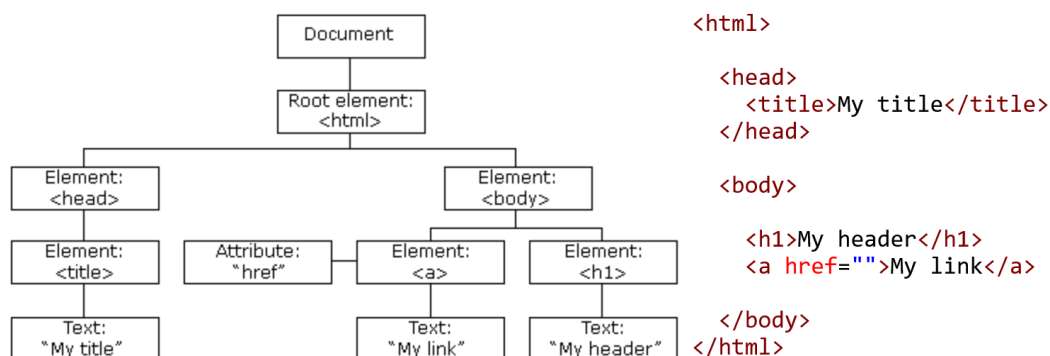
Le DOM est une interface de programmation (API) pour l'HTML, le XML et le SVG. Dans le cadre de ce cours, le DOM sert à "connecter" une page web au JavaScript.

Même si le DOM est essentiellement utilisé en JavaScript, ce n'est pas du JavaScript.

Le DOM doit être distingué du HTML initial, il peut être modifié par le navigateur.

Le DOM:

- fournit une représentation structurée d'un **document** en arbre
- expose des méthodes permettant d'accéder au document et d'y apporter des modifications dans sa **structure**, son **style** et son **contenu**.
- représente le document en **nœuds** et **objets** ayant chacun leurs propriétés
- Permet d'écouter et de gérer des **événements** sur les nœuds.



Voir `dom-01_document.html`

Modification du DOM

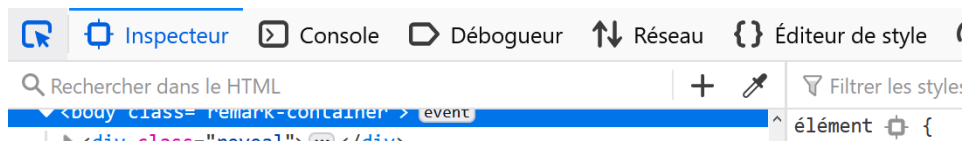
Le DOM peut-être modifié, par exemple en JavaScript:

```
document.getElementById("modification-du-dom").textContent="Le titre de cette page s'est modifié!";
```

- `getElementById` est une méthode de `Document` qui retourne un `Element`.
- `textContent` est un propriété de `Node` héritée par `Element`.

La console ou certains éditeurs de code proposent de l'autocomplétion de méthodes et propriétés de DOM.

Pour naviguer dans le DOM, rien de mieux l'**outil de sélection d'éléments** dans les outils de développement du navigateur!



Voir `dom-02_alert.js`

Sélecteurs les plus courants

Exemples	Signification
<code>.getElementById()</code>	Sélectionne l'élément avec l'id en paramètre
<code>.getElementsByName()</code>	Sélectionne tout les éléments selon le tag donné (par exemple 'p')
<code>.getElementsByClassName()</code>	Sélectionne tout les éléments selon l'attribut name
<code>.children</code>	Sélectionne tout les éléments selon ayant la classe donnée
<code>.textContent</code>	Sélectionne tout les enfants
<code>.querySelector()</code>	Sélectionne le texte à l'intérieur de l'élément
	Sélectionne des éléments selon la syntaxe CSS

- Les sélecteurs peuvent être chaînés, par ex:
`document.getElementById("s-lecteurs-les-plus-courants").children`
- L'autocomplétion dans la console vous aide à connaître les méthodes disponibles sur un élément

Gestion d'événements

Une page HTML se charge → le `document` se créé. Comment savoir quand la page a terminé de se charger?

```
// Création d'un objet qui a deux propriétés de type string
const myContent = {
  alertText: "Le document est chargé",
  alertLink: "https://developer.mozilla.org/fr/docs/web/API"
};

// Utilisation du DOM!
document.onreadystatechange = function() {
  console.log("L'état du document a changé:", document.readyState);
  if (document.readyState === "complete") {
    console.log(myContent.alertText);
  }
}
```

Quelques autres événements

```
<button onclick="console.log('click')">Cliquez-moi!</button>
<input type="checkbox" onchange="console.log(`Est-ce coché? : ${this.checked}`)">
<div class="pink" onmouseover="console.log('ha'.repeat(Math.floor(Math.random() * 10) + 1))"></div>
<input type="text" onselect="console.log('select!')" value="Select me!">
<input type="text" onkeyup="console.log(this.value)">
```

Plein d'autres événements existent:

<https://developer.mozilla.org/fr/docs/Web/Events>

Voir `dom-03_ready.js`

Le mot-clé `this` dans l'HTML

En JavaScript, `this` se réfère au contexte dans lequel on se trouve. [Voir le chapitre sur la portée des variables](#)

Utilisé comme argument dans le *callback* d'un *event*, il permet de passer l'élément actuel au *callback*:

Dans cet HTML, `doSomething()` est la fonction appelée quand `onkeyup` est fait. Autrement dit, `doSomething()` est le *callback* de `onkeyup`:

```
<input type="text" onkeyup="doSomething(this)">
```

En JavaScript

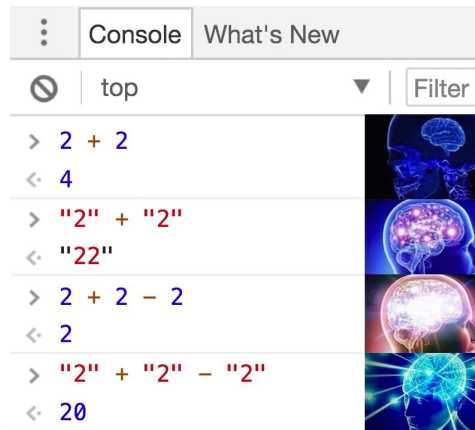
```
function doSomething(someElement) {
  console.log(typeof someElement); // object
  console.log(someElement.value); // le contenu que l'utilisateur tape dans l'input
}
```

Le mot clé `this` peut également être utilisé dans une fonction JavaScript mais cela sort du contexte de ce cours. Si vous souhaitez en savoir plus:

https://www.w3schools.com/js/js_this.asp

Exercices

Faites les exercices dom-01 à dom-04.



```
Console What's New
top Filter
> 2 + 2
< 4
> "2" + "2"
< "22"
> 2 + 2 - 2
< 2
> "2" + "2" - "2"
< 20
```

Pour vous aider, cette page peut servir de "dictionnaire" de tags HTML avec chaque fois un détail des événements et attributs disponibles:

<https://www.w3schools.com/tags/default.asp>

Hors série: qu'est ce que le Path ?


`Path` est une variable d'environnement disponible sur les systèmes Windows, Linux et Unix. Elle permet de mettre au courant le système de l'existence d'un programme ainsi que de l'endroit où il se trouve. Une fois le système au courant, on pourra taper directement le nom dudit programme dans un terminal, sans avoir à taper son chemin complet.

Les programmes concernés dans ce cours sont `npm`, `python` et `psql`.

Sur windows, le `Path` existe à deux niveaux:

- Au niveau du système, pour tous les utilisateurs. Privilèges d'administration nécessaires pour le changer
- Au niveau de l'utilisateur courant.

Pour ajouter un programme au `Path` sur Windows:

1. Cliquer sur le menu démarrer ou presser la touche  (windows)
2. Taper "variables" et choisir "Modifier les variables d'environnement pour votre compte"
3. Cliquer sur `Path` puis "Modifier"
4. Ajouter le chemin du dossier contenant le programme à la fin de la liste.

npm




npm est un gestionnaire de paquets. Il facilite l'installation et la gestion des librairies dont dépend notre projet. Il est courant qu'un projet en JavaScript dépende de plusieurs librairies qui elles-mêmes ont des dépendances résultant parfois sur des milliers de dépendances.

npm s'utilise en tapant `npm` en ligne de commande mais:

- Node.js doit être installé sur la machine
- npm doit être présent dans le `Path` (c'est une question posée à l'installation)

npm init

Pour démarrer un projet avec npm que ce soit dans un répertoire vide ou dans lequel du code est déjà présent, lancez cette commande à la racine de votre projet:

 Si la fenêtre de terminal n'est pas encore affichée dans votre Visual Studio Code: cliquez sur le menu *Terminal > New Terminal*

```
npm init -y
```

L'option `-y` permet de répondre à toutes les questions par oui.


Un fichier `package.json` est créé:

- Il contient toutes les infos nécessaires à publier notre projet en tant que `package`.
- Les dépendances à d'autres librairies y seront listées.

npm install

L'instruction `install` que l'on peut abréger en `i` permet d'installer un paquet.

```
npm i bootstrap@5.3.2 @popperjs/core
```

 Il est recommandé de spécifier une version, ici `@5.3.2`. Cela afin d'éviter des mises à jours automatiques pouvant casser votre projet.

Le fichier `package.json` a été modifié: une section `dependencies` s'est créée

Un fichier `package-lock.json` est arrivé: il contient l'arbre des dépendances

Un dossier `node_modules` s'est créé: il contient les fichiers téléchargés prêts à être utilisés, dans ce cas: bootstrap et popperjs

```
▼ npm-01
  ▼ node_modules
    > @popperjs
    > bootstrap
    {} .package-lock.json
    <> index.html
    {} package-lock.json
    {} package.json
```

npm install --save-dev

L'option `--save-dev` ou `-D` permet d'installer des dépendances pour le développement.

On parle d'outils qui vont nous aider pour le développement mais qui ne seront pas dans le code final en production quand l'application JavaScript sera terminée.

Vite



Vite permet d'utiliser des fonctions JavaScript qui ne sont pas encore supportées par les navigateurs. Vite se chargera de transformer (transpiler) notre code vers un code compréhensible par le navigateur. Il va également réduire la taille du code par deux mécanismes principaux:

- La minification: tous les espaces blancs sont réduits à leur strict minimum et les variables et fonctions sont renommées quand cela est possible
- Seules les parties des librairies qui sont utilisées dans notre code seront importées dans le fichier final.

Pour installer Vite:

```
npm i -D vite
```

Vite - configuration

Dans le fichier `package.json`, remplacer les lignes `"scripts"`

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

par celles-ci:

```
"scripts": {  
  "start": "vite",  
  "build": "vite build",  
  "serve": "vite preview",  
},
```

Ces "scripts" sont des raccourcis de commandes communs à la plupart des projets JavaScript. Ils permettent de lancer les commandes Vite sans avoir à toucher au Path.

Lancer la commande `npm run start` permet de lancer le mini serveur vite qui lui, contrairement au "Go Live" de Visual Studio Code, se chargera de transpiler le code.

Vite - changements par rapport à la méthode traditionnelle

On peut continuer à inclure les CSS et JavaScript avec la méthode traditionnelle mais on ne profitera pas du travail d'optimisation de Vite.

Pour un projet simple:

- Importer les CSS tierces (telles que Bootstrap) dans le fichier CSS `@import`

- Créer un fichier `index.js` qui importe la CSS
- Dans le fichier `index.html`, on n'a plus d'import de CSS dans le `<head>`
- Toujours dans `index.html`, le JavaScript est importé avec l'attribut `module`
- On n'utilise plus "Go Live" mais `npm run start`

```
<script type="module" src="index.js"></script>
```

Exercice

Suivre le "Quick Start" d'OpenLayers:

<https://openlayers.org/doc/quickstart.html>