



PROGRAMMATION WEB POUR LA VISUALISATION

SD BUT3 R5.VCOD.07
Yacine Ouzrout



iut.univ-lyon2.fr

@IUTLumiereLyon2

OBJECTIF

3.3.7. Ressource R5.VCOD.07 : Programmation web pour la visualisation

Compétence ciblée :

- Développer un outil décisionnel

SAÉ au sein desquelles la ressource peut être mobilisée et combinée :

- SAÉ 5.VCOD.01 | Analyse et conception d'un outil décisionnel

Descriptif :

L'objectif de cette ressource est de découvrir les éléments utiles pour la réalisation de reporting et de dataviz via des pages web, à partir de données stockées sur un serveur, en anticipant les éventuelles erreurs (connexion impossible, fichiers non trouvés, ...)

Contenus :

- HTML, CSS, Javascript
- Utilisation de librairies dédiées
- Importation de données stockées
- Automatisation de l'importation de données (gestion d'erreurs, ...)
- Tests et recettes de la solution créée
- Comparaison d'outils et de méthodes

Les étudiants sont ici formés à des techniques complémentaires en datavisualisation dans un contexte web. Les éléments pour choisir la solution la plus adaptée à un problème ou une demande particulière sont discutés. Les étudiants sont également sensibilisés aux problèmes pouvant être rencontrés (problèmes de connexion, de disponibilité des données, ...) et à la façon de les gérer.

Apprentissages critiques ciblés :

- AC34.01VCOD | Prendre conscience de la nécessité d'utiliser des moyens spécifiques pour exploiter les Big Data ou les flux de données
- AC34.02VCOD | Défendre ses choix de solution par un argumentaire éclairé
- AC34.04VCOD | Apprécier l'intérêt de l'utilisation d'un gestionnaire de versions de code

Mots clés :

Librairies dédiées – Données stockées – Comparaison d'outils

Volume horaire :

Volume horaire défini nationalement : 35 heures dont 20 heures de TP

ORGANISATION DU COURS

7 TD et 10 TP

MCCC

- 1 TD ordinateur (0,5)
- 1 DS ordinateur (0,5)

- **Partie 1 : Approfondissement Javascript**
 - Programmation Objet en JS
 - Structuration des pages Web (DOM)
- **Partie 2 : Traitement des données Json**
 - Les formats de transfert données XML, Json...
 - Automatisation du traitement des données avec Json
 - Exploitation des données en JS
- **Partie 3 : Visualisation avec D3.js** <https://observablehq.com/@d3/gallery>
 - Les librairies JS pour la visualisation
 - Développement de graphiques avec D3.js
- **Partie 4 : Le javascript côté serveur**
 - Découverte de la plateforme NodeJS
 - Interaction avec une base de données
 - Développement d'une application web avec le Framework Express.js
 - API REST

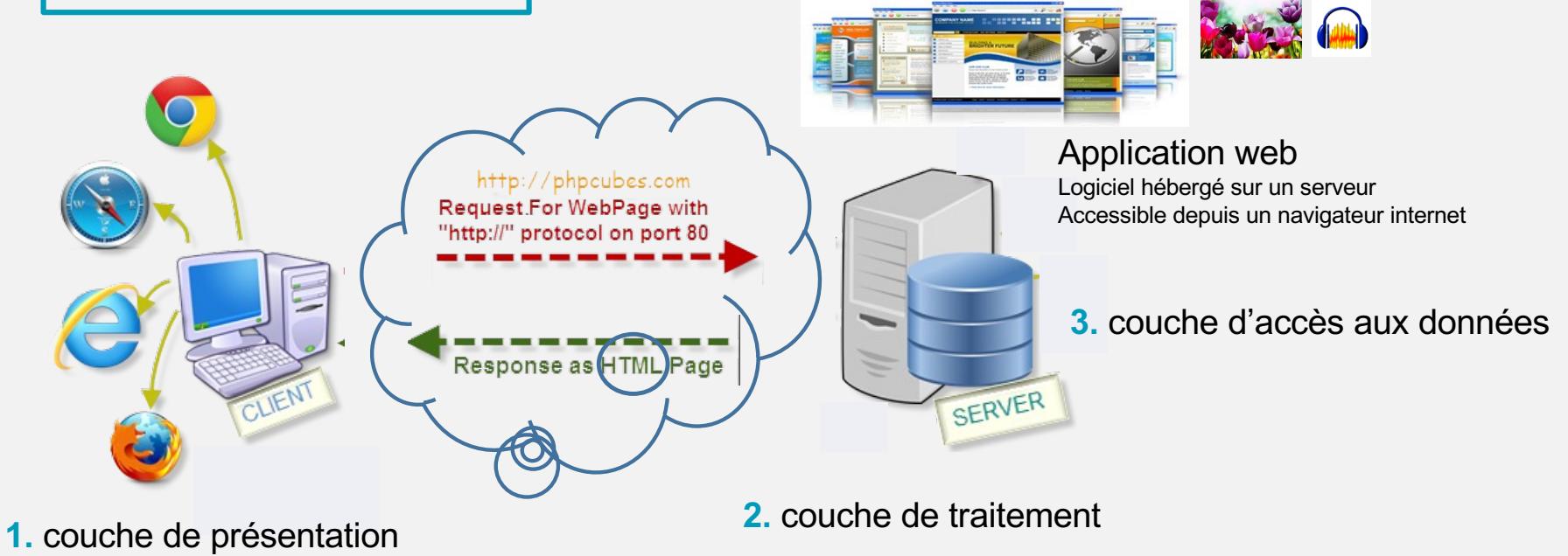
PARTIE 1 : APPROFONDISSEMENT JS

Javascript

- Langage de programmation lié à HTML
- Code JavaScript intégré aux pages HTML
- Code interprété par le navigateur client
- Différent du code PHP qui est interprété du côté serveur

Environnement Client-Serveur

Architecture 3 tiers



LES TECHNOLOGIES WEB

- Coté Client



- Coté Serveur



Langages Web dynamiques

Du côté client *front-end : la partie visible d'un site, les interfaces*

- *HTML/CSS*
- *Script client JavaScript*
- *Frameworks et bibliothèques JavaScript : Angular, React, Vue.js, Jquery ...*

Du côté serveur *back-end : fonctionnalités du site, gestion des données...*

- Langage de script **PHP** (*Hypertext Preprocessor*) (libre)
- *JavaScript*
- *Python*
- *Frameworks : Django (Python), Node.js (JavaScript), Express (Node.js), Laravel (PHP), Ruby on Rails, Noyau ASP.net, GateauPHP (PHP)...*
- **SQL**

LIBRAIRIES DE CRÉATION DE PAGES WEB

Des librairies, *frameworks*, **CSS** gratuits ou payants :

- Bootstrap
- Foundation
- Materialize
- Bilma
- Ukit
- Skeleton
- ...

<https://www.codeur.com/blog/frameworks-css/>

Des librairies, *frameworks*, **Javascript** gratuits ou payants :

- Angular.js (*création de pages*)
- React.js (*faire des interfaces*)
- Vue.js (*faire des interfaces*)
- Jquery (*manipuler les données*)
- **D3.js** (*visualisation des données*)
- Chart.js (*faire des graphiques*)
- Anime.js (*faire des animations*)
- Bideo.js (*intégrer des vidéos*)
- ...

<https://kinsta.com/fr/blog/bibliotheques-javascript/>

https://alokai-com.translate.goog/blog/vue-vs-react?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=rq

Des **CMS** (Content Management System) :

- WordPress (*open source*)
- Joomla (*open source*)
- Drupal (*open source*)
- Jimdo
- Site123
- ...

<https://www.iphon.fr/hebergeur/faq/cms>

Javascript

- JavaScript est un langage
 - à **prototype** (par opposition à un langage basé sur les classes et sous classes pour réaliser l'héritage). Tout objet JavaScript est doté d'une propriété **prototype**, qui représente le modèle de l'objet.
 - **événementiel** (association d'actions aux événements déclenchés par l'utilisateur (passage de souris, clic, saisie clavier, etc...)).
- Javascript renferme une panoplie d'objets prédéfinies (appelés prototypes) qui peuvent donner naissance à d'autres objets que l'on souhaite créer.
- Javascript est standardisé par un comité spécialisé, l'ECMA (*European Computer Manufacturers Association*).

Les objets

Un objet est un élément nommé ayant des

- **Propriétés** : paramètres que vous pouvez vérifier et modifier.
- **Méthodes** : actions que l'objet est capable d'effectuer.
- **Événements** : actions qui arrivent à l'objet, auxquelles celui-ci peut répondre automatiquement par une action.

Les Objets

- Les objets de JavaScript, sont soit des entités prédéfinies du langage, soit créés par le programmeur :
 - Le navigateur est un objet qui s'appelle **navigator**
 - La fenêtre du navigateur est un objet qui se nomme **window**
 - La page HTML est un autre objet, que l'on appelle **document**
 - Un formulaire **Form** à l'intérieur d'un **document**, est aussi un objet
 - Un lien hypertexte **link** dans une page HTML, est encore un autre objet
 - etc...
- Les objets javascript peuvent réagir à des "Evénements »
 - `<input type="button" value="Valider" onclick="generer()">`
- Accès aux propriétés/attributs d'un objet
 - `voiture.couleur.value = "verte »;`
 - `var Var1 = voiture.couleur.value;`

Les objets

- Les objets Javascript qui dépendent du contenu d'un document HTML sont structurés hiérarchiquement. C'est à dire que des objets s'emboîtent dans d'autres jusqu'à atteindre l'objet le plus élevé à l'échelle de la hiérarchie.
- L'objet le plus élevé est la fenêtre du navigateur, il est identifié par le nom **window**. Il peut être manipulé directement à travers des fonctions qui lui sont prédéfinies. On appelle ces fonctions des **méthodes**.
- Exemple : méthode **alert()** qui affiche les messages est une méthode de l'objet window, on peut écrire : **window.alert()**
- Le point « **.** » permet:
 - de passer d'un objet à une de ces méthodes
 - d'un objet parent à un autre objet enfant
 - d'un objet à une variable qui lui est propre ; on appelle cette variable un attribut.

DOM : DOCUMENT OBJECT MODEL

DOM

- Le **DOM** (**D**ocument **O**bject **M**odel) est un standard W3C qui décrit une API orientée objet permettant de représenter un document HTML en mémoire afin de la manipuler avec un langage de programmation.
- Une **API** (**A**pplication **P**rogramming **I**nterface ou « interface de programmation d'application ») est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

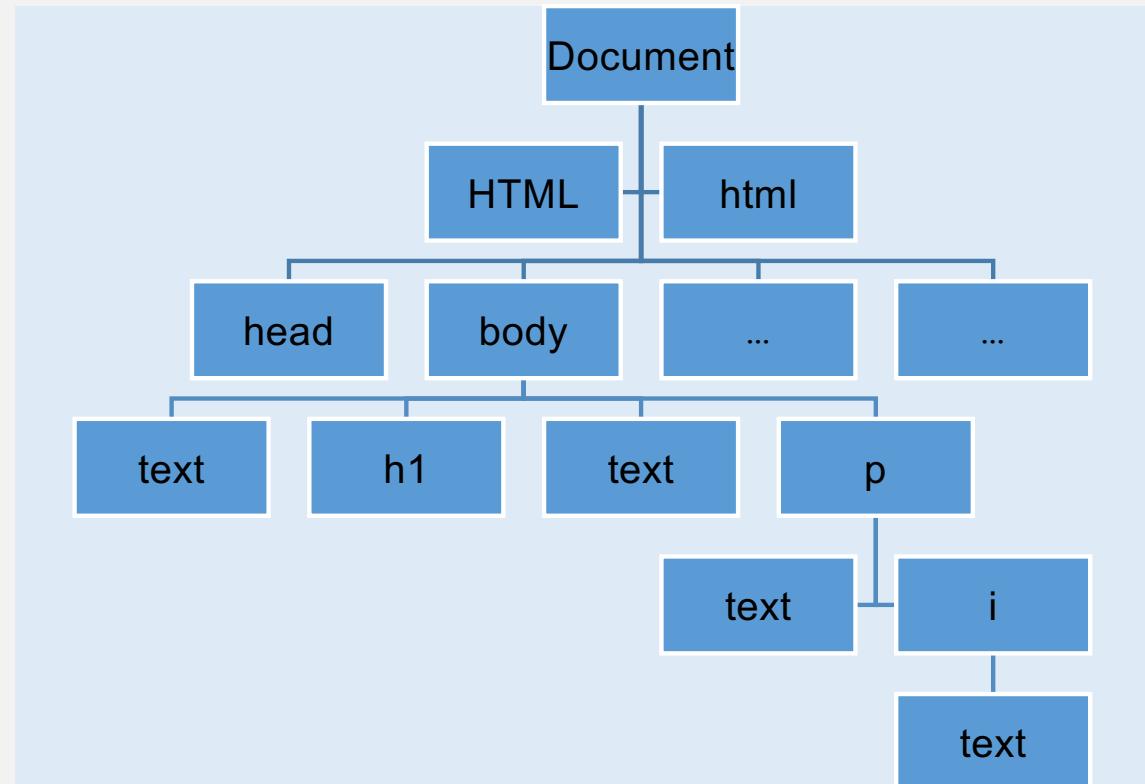
[W3C : World Wide Web Consortium](#), est un organisme international qui développe des standards pour le Web

DOM

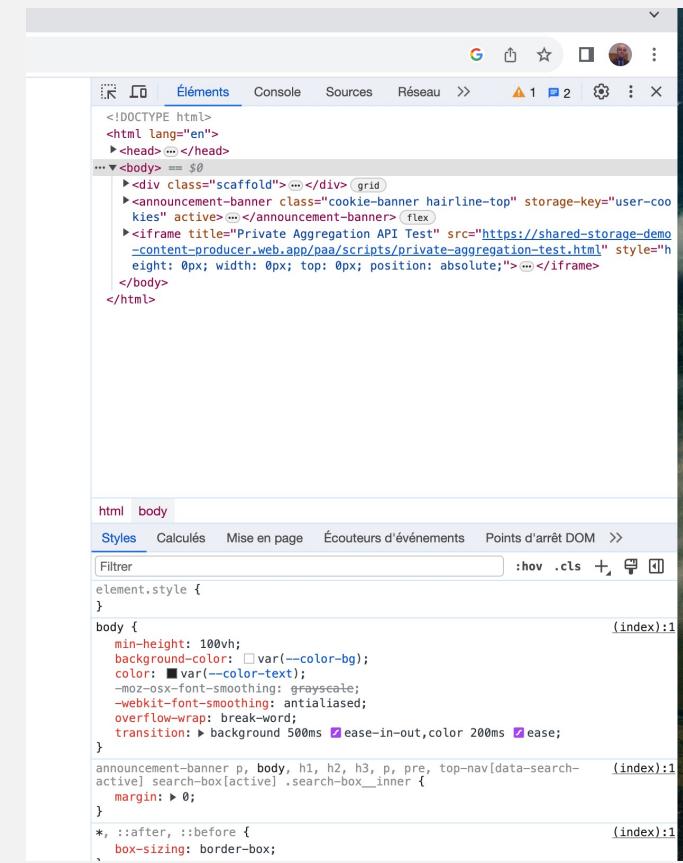
- Lorsqu'on demande à un navigateur d'afficher une page Web, celui-ci va automatiquement créer un modèle objet de la page ou du document.
- Ce modèle objet correspond à une autre représentation de la page sous forme d'arborescence contenant des objets qui sont de type **Node** (nœuds).
- Nous allons utiliser ce modèle en utilisant un langage de script, le JavaScript, pour manipuler les propriétés et les méthodes des objets de la page web
- Les objets de la page web peuvent être identifiés par un attribut de nom ou d'ID qui lui donne son adresse unique et en fait un objet
 - <p id="Par1"> Bonjour tout le monde </p>

DOM : un arbre avec des nœuds (nodes)

- Le DOM est structuré comme un arbre et suit la structure hiérarchique du code HTML.
- L'arbre contient des nœuds, les nœuds peuvent avoir des fils, et tous les nœuds ont un parent (sauf la racine).



Inspecteur DOM : inclus dans Firefox, Chrome, I.E...



Exemple

JS-Exemp-1.html

```
<HTML>
<HEAD>
<TITLE>Introduction au DOM</TITLE>
</HEAD>

<BODY>
    <h1> Titre principal </h1>
    <p> Un paragraphe </p>
</BODY>
</HTML>
```

Titre principal

Un paragraphe

Clique droit → Inspecter



The screenshot shows a browser window with developer tools open. The tab bar at the top includes 'Inspecteur' (highlighted in blue), 'Console', 'Débogueur', 'Réseau', 'Éditeur de style', and 'Pe'. Below the tab bar is a search bar labeled 'Rechercher dans le HTML'. The main area displays the DOM tree:

```
<html>
  > <head> ... </head>
  > <body>
    > <h1>Titre principal</h1>
    > <p>Un paragraphe</p>
  > </body>
</html>
```

The node `<p>Un paragraphe</p>` is highlighted with a blue background, and a blue downward arrow points to the 'Inspecteur' tab.

Exemple

JS-Exemp-1.html

```
<HTML>
  <HEAD>
    <TITLE>Introduction au DOM</TITLE>
  </HEAD>

  <BODY>
    <h1> Titre principal </h1>
    <p> Un paragraphe </p>
  </BODY>
</HTML>
```

The screenshot shows a browser window with the URL `file:///Users/youzrout/Documents/Yacine/IUT/YO-CoursIUT/SD/JS-Exemp-1.html`. The page content is "Titre principal" followed by "Un paragraphe". A blue arrow points from the text "Clique droit → Inspector" to the context menu in the browser. Another blue arrow points from the text "Clique droit → Afficher les propriétés DOM" to the expanded DOM tree in the developer tools.

Titre principal

Un paragraphe

Clique droit → Inspector

Clique droit → Afficher les propriétés DOM

html > head

childElementCount: 2

childNodes: NodeList(5) [#text "\n\t" , h1 , #text "\n\t" , ...]

0: #text "\n\t"

1: <h1>

2: #text "\n\t"

3: <p>

4: #text "\n\n \n"

length: 5

<prototype>: NodeListPrototype { item: item(), keys: keys(), values: values(), ... }

children: HTMLCollection { 0: h1 , 1: p , length: 2 }

classList: DOMTokenList []

className: ""

Les propriétés du DOM : les Node

- C'est une des interfaces incontournables du modèle.
- Les différents types de nœuds reflètent les différentes catégories de balisage d'un document : éléments, attributs, commentaires, textes.
- Tous ces types de nœuds partagent un ensemble de propriétés et fonctions héritées de leur type générique le nœud.

Ainsi, toutes les interfaces sur les nœuds disposent des propriétés et méthodes de **Node**.

Exemple (2)

```
1 <HTML>
2 <HEAD>
3   <TITLE>Introduction au DOM</TITLE>
4 </HEAD>
5
6 <BODY>
7   <h1> Titre principal </h1>
8   <p> Un paragraphe </p>
9
10  <div>
11    <p> paragraphe 2 </p>
12    <ol>
13      <li> titre 1 </li>
14      <li> titre 2 </li>
15      <li> titre 3 </li>
16    </ol>
17  </BODY>
18 </HTML>
```



JS-Exemp-2.html

Titre principal

Un paragraphe

paragraphe 2

- 1. titre 1
- 2. titre 2
- 3. titre 3

The screenshot shows the browser's developer tools with the "Inspecteur" tab selected. The element being inspected is an ordered list () located within a div. The list contains three items, each represented by a blue-highlighted element. The browser's address bar at the top shows the file path: file:///Users/youzrout/Documents/Yacine/IUT/YO-Cor.

```
html > body > div > ol
  ↵ <ol> ⚡
    accessKey: ""
    accessKeyLabel: ""
    assignedSlot: null
    ▶ attributes: NamedNodeMap []
    autocapitalize: ""
    autofocus: false
    baseURI: "file:///Users/youzrout/Documents/Yacine/IUT/YO-CoursIUT/SD/_Cours/__2023_Programmation%20Web%20Visualisa"
    childElementCount: 3
    ▶ childNodes: NodeList(7) [ #text ⚡, li ⚡, #text ⚡, ... ]
      ▶ 0: #text "\n\t\t\t"
      ▶ 1: <li> ⚡
        ▶ 2: #text "\n\t\t\t"
        ▶ 3: <li> ⚡
        ▶ 4: #text "\n\t\t\t"
        ▶ 5: <li> ⚡
        ▶ 6: #text "\n\t\t"
        ▶ length: 7
      ▶ <prototype>: NodeListPrototype { item: item(), keys: keys(), values: values(), ... }
    ▶ children: HTMLCollection { 0: li ⚡, 1: li ⚡, length: 3, ... }
      ▶ 0: <li> ⚡
      ▶ 1: <li> ⚡
      ▶ 2: <li> ⚡
      ▶ length: 3
      ▶ <prototype>: HTMLCollectionPrototype { item: item(), namedItem: namedItem(), length: Getter, ... }
    ▶ classList: DOMTokenList ⚡
```

Autres Exemples

- DOM : application d'événements sur des objets du DOM [JS-Exemp-3.html](#)
- DOM : Propriétés des nœuds des objets du DOM [JS-Exemp-4.html](#)

Propriétés de parcours du DOM

JS-Exemp-5.html

```
<html>
<body>
  <h1 id="vTitre">Les légumes </h1>
  <ul>
    <li id="a">Artichaud</li>
    <li id="n">Aubergine</li>
    <li id="c">carotte </li>
    <li id="m">Mangue</li>
    <li id="p">pomme de terre </li>
  </ul>
</body>
</html>
```

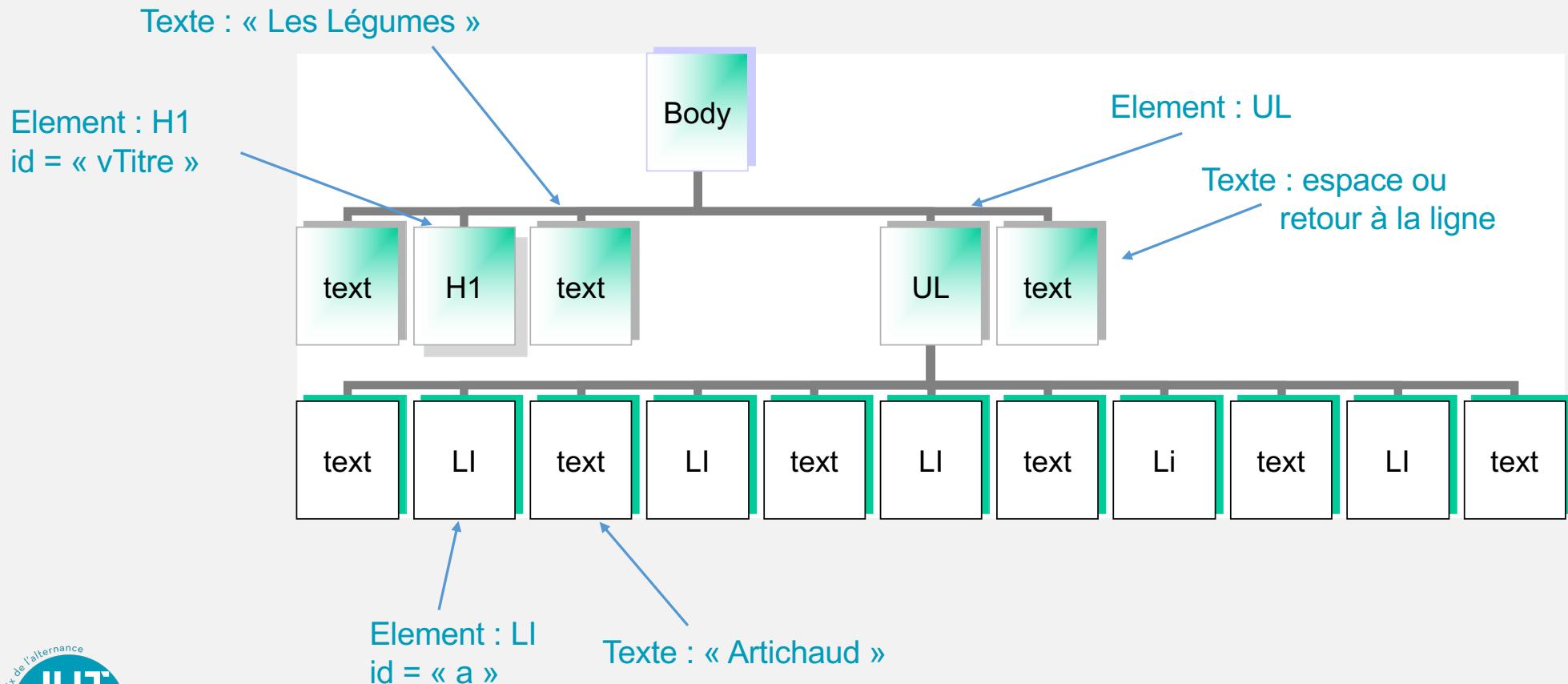
Les légumes

- Artichaud
- Aubergine
- carotte
- Mangue
- pomme de terre

The screenshot shows the DOM tree of the page. The root node is the HTML element, which contains a head section and a body section. The body section contains a title element with the text "Inspection du DOM", a script element with some JavaScript code, and a main content area. The main content area contains an h1 element with the id "vTitre" and the text "Les légumes", followed by a ul element. The ul element has five li children with ids "a", "n", "c", "m", and "p", each containing a piece of fruit or vegetable names.

```
<!DOCTYPE html>
<html> [event] [défilable]
  <head></head>
  <body> [débordement]
    <meta content="text/html; charset=UTF-8" http-equiv="content-type">
    <title>Inspection du DOM</title>
    <h1 id="vTitre">...</h1>
    <ul>
      <li id="a">
        ::marker
        Artichaud
      </li>
      <li id="n">...</li>
      <li id="c">...</li>
      <li id="m">...</li>
      <li id="p">...</li>
    </ul>
    <script language="JavaScript">...</script>
  </body>
</html>
```

Propriétés du DOM



```

<html>
<body>
  <h1 id="vTitre">Les légumes </h1>
  <ul>
    <li id="a">Artichaud</li>
    <li id="n">Aubergine</li>
    <li id="c">carotte </li>
    <li id="m">Mangue</li>
    <li id="p">pomme de terre </li>
  </ul>
</body>
</html>

```

← → ⌂ ⌂

file:///Users/youzrout/Documents/Yacine/IUT/YO-CoursIUT/SD/_C

Propriétés de parcours du DOM

Les légumes

- Artichaud
- Aubergine
- carotte
- Mangue

html > body > ul

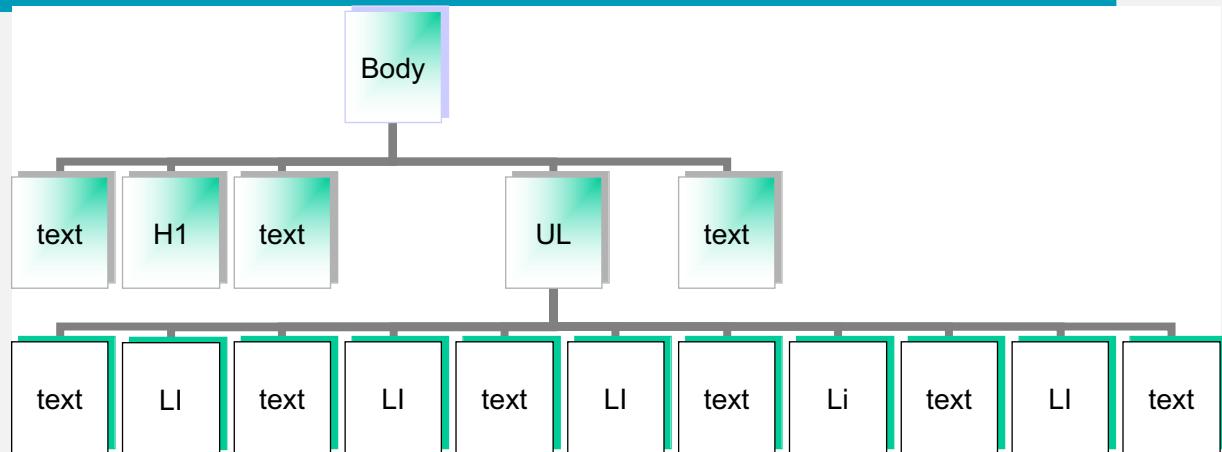
⋮ Filtrer

```

>> inspect($0, true)
<- <ul> ⓘ
  accessKey: ""
  accessKeyLabel: ""
  assignedSlot: null
  attributes: NamedNodeMap []
  autocapitalize: ""
  autofocus: false
  baseURI: "file:///Users/youzrout/Documents/Yacine/IUT/YO-CoursIUT/SD/_Cours/__2023_Programmation%20Web%20Visualisati"
  childElementCount: 5
  ▾ childNodes: NodeList(11) [ #text ⓘ , li#a ⓘ , #text ⓘ , ... ]
    ▶ 0: #text "\u000b\t" ⓘ
    ▶ 1: <li id="a"> ⓘ
    ▶ 2: #text "\u000b\t" ⓘ
    ▶ 3: <li id="n"> ⓘ
    ▶ 4: #text "\u000b\t" ⓘ
    ▶ 5: <li id="c"> ⓘ
    ▶ 6: #text "\u000b\t" ⓘ
    ▶ 7: <li id="m"> ⓘ
    ▶ 8: #text "\u000b\t" ⓘ
    ▶ 9: <li id="p"> ⓘ
    ▶ 10: #text "\u000b" ⓘ
    length: 11
  > <prototype>: NodeListPrototype { item: item(), keys: keys(), values: values(), ... }
  ▾ children: HTMLCollection { 0: li#a ⓘ , 1: li#n ⓘ , length: 5, ... }

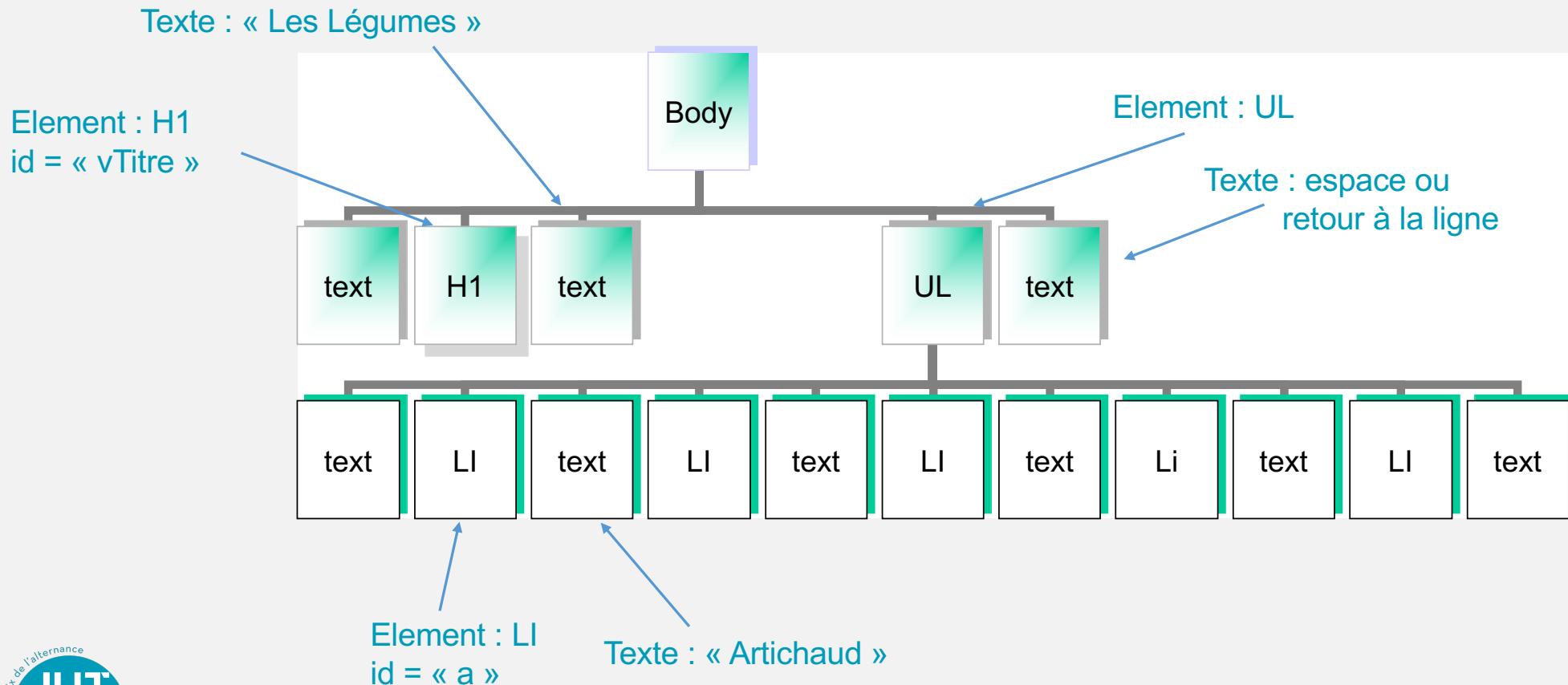
```

Propriétés du DOM



Propriétés	description
parentNode	le nœud père
childNodes	nœud fil du nœud courant
firstChild	le premier nœud fils
lastChild	le dernier nœud fils
previousSibling	le nœud frère précédent
nextSibling	nœud frère suivant

Propriétés du DOM



Exemple

JS-Exemp-5.html

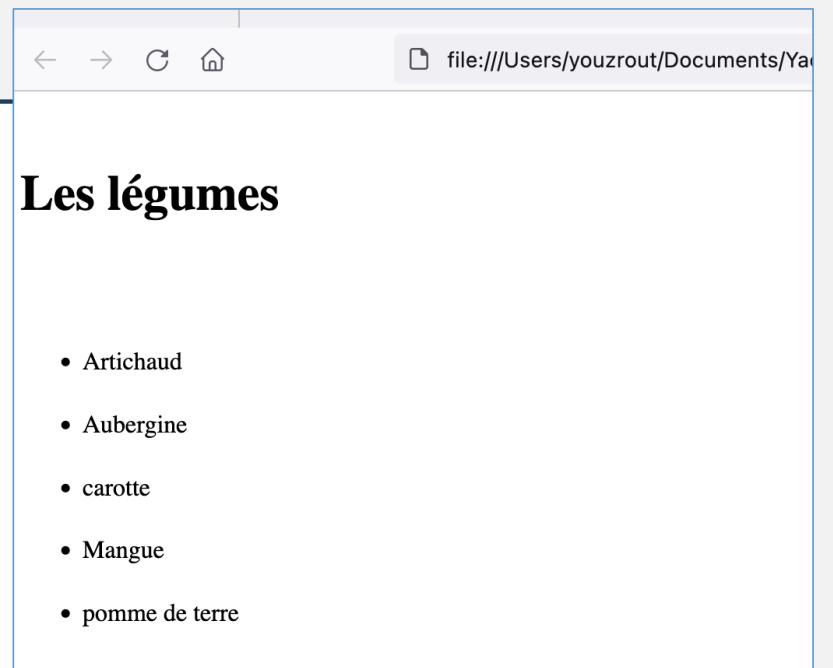
```
<body>
<h1 id="vTitre">Les légumes </h1>
<ul>
    <li id="a">Artichaud</li>
    <li id="n">Aubergine</li>
    <li id="c">carotte </li>
    <li id="m">Mangue</li>
    <li id="p">pomme de terre </li>
</ul>

<script language =JavaScript>
window.onload=function()
{
    var el = document.getElementById("vTitre");
    alert(el.nodeName);
    alert(el.InnerText);

    //Afficher le nodeName du parent de l'élément qui a pour ID = "a "
    alert(document.getElementById( "a").parentNode.nodeName);

    alert(document.getElementById( »n").parentNode.previousSibling.previousSibling.innerHTML);

}
</script>
</body>
```



Exemple

JS-Exemp-5.html

```
<body>
<h1 id="vTitre">Les légumes </h1>
<ul>
    <li id="a">Artichaud</li>
    <li id="n">Aubergine</li>
    <li id="c">carotte </li>
    <li id="m">Mangue</li>
    <li id="p">pomme de terre </li>
</ul>

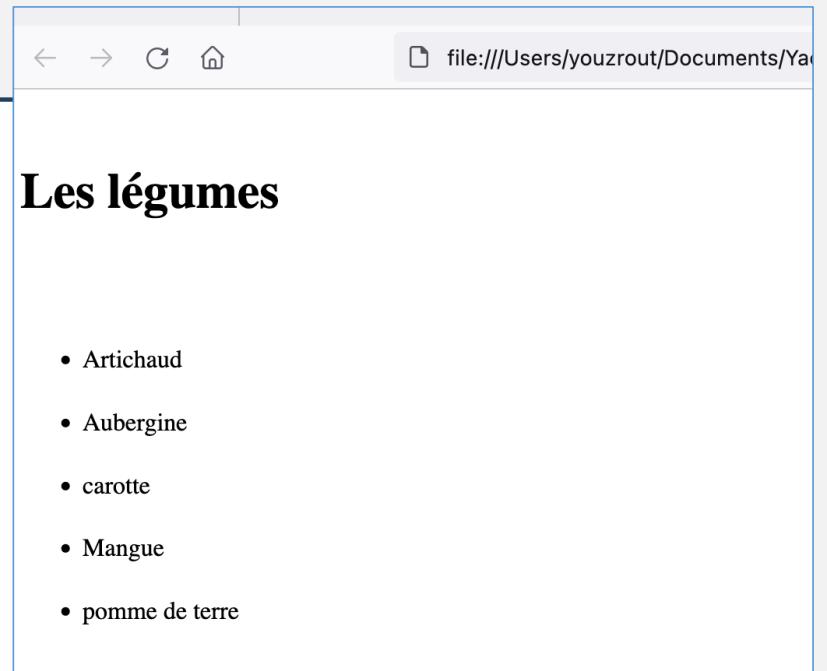
<script language =JavaScript>
//fonction qui s'exécute à l'ouverture de la page
window.onload=function()  {

    var el=document.getElementById("vTitre");
    alert(el.nodeName);
    alert(document.getElementById( "a").parentNode.nodeName);

    alert(document.getElementById( "n").innerHTML);
    alert(document.getElementById( "n").textContent);

    alert(n.parentNode.previousSibling.previousSibling.innerHTML);

    alert(a.parentNode.nextSibling.nextSibling.nodeName);
}
</script>
</body>
```



Modèle Objet de Document (DOM)

- Quid ?
 - Modèle associé à un l'environnement client
- But
 - Permettre la manipulation des objets :
 - de l'interface
 - du document
 - (objets créés automatiquement par le navigateur)
- Types d'objets
 - **Window, Document, Form, Browser , ...**

Description des tous les objets du DOM :

https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model 32

Modèle Objet de Document (DOM)

Les événements

Événements page et fenêtre [\[modifier \]](#) [\[modifier le code \]](#)

- `onabort` — s'il y a une interruption de chargement
- `onerror` — en cas d'erreur pendant le chargement de la page
- `onload` — après la fin du chargement de la page
- `onbeforeunload` — se produit juste avant de décharger la page en cours (par changement de page, en quittant)
- `onunload` — se produit lors du déchargement de la page (par changement de page, en quittant)
- `onresize` — quand la fenêtre est redimensionnée

Événements clavier [\[modifier \]](#) [\[modifier le code \]](#)

- `onkeydown` — lorsqu'une touche est enfoncee
- `onkeypress` — lorsqu'une touche est pressée et relâchée
- `onkeyup` — lorsqu'une touche est relâchée

Événements formulaire [\[modifier \]](#) [\[modifier le code \]](#)

- `onblur` — à la perte du focus
- `onchange` — à la perte du focus, si la valeur a changé
- `onfocus` — lorsque l'élément obtient le focus (ou devient actif)
- `onreset` — lors de la remise à zéro du formulaire via un bouton ou une fonction `reset()`
- `onselect` — quand du texte est sélectionné
- `onsubmit` — quand le formulaire est validé via un bouton ou une fonction `submit()`

Événements souris [\[modifier \]](#) [\[modifier le code \]](#)

- `onclick` — sur un simple clic
- `ondblclick` — sur un double clic
- `onmousedown` — lorsque le bouton de la souris est enfoncé, sans forcément le relâcher
- `onmousemove` — lorsque le curseur est déplacé
- `onmouseout` — lorsque le curseur sort de l'élément
- `onmouseover` — lorsque le curseur se trouve sur l'élément
- `onmouseup` — lorsque le bouton de la souris est relâché
- `onscroll` — lorsque le scroll de la souris est utilisé

DOM : 1. Objet Window

Propriétés

- `frames[]` : tableau de frames
- `frames.length` : nombre de frames
- `self` : fenêtre courante
- `opener` : la fenêtre (si elle existe) qui a ouvert la fenêtre courante
- `parent` : parent de la fenêtre courante, si la fenêtre courante est une sous-partie d'un frameset
- `top` : fenêtre principale (qui a créé toutes les fenêtres)
- `status` : message dans la barre de statut
- `defaultstatus` : message par défaut de la barre de statut
- `name` : nom de la fenêtre

DOM : 1. Objet Window

Méthodes

- `alert(string)` : ouvre une boîte de dialogue avec le message passé en paramètre
- `confirm` : ouvre une boîte de dialogue avec les boutons Ok et cancel
- `blur()` : enlève le focus de la fenêtre
- `focus()` : donne le focus à la fenêtre
- `prompt(string)` : affiche une fenêtre de saisie
- `scroll(int x, int y)` : positionnement aux coordonnées (x,y)
- `open(URL, string name, string options)` : ouvre une nouvelle fenêtre contenant le document identifié par l'URL
- `close()` : ferme la fenêtre

DOM : 2. Objet Document

Propriétés

- `title` : titre du document
- `location` : URL du document
- `lastModified` : date de dernière modification
- `referrer` : URL de la page d'où arrive l'utilisateur
- `fgColor` : couleur du texte
- `linkColor`
`vlinkColor`
`alinkColor` couleurs utilisées pour les liens hypertextes

Ex: `document.fgColor="red";`

DOM : 2. Objet Document

Propriétés

- `forms[]` : tableau des formulaires de la page
- `forms.length` : nombre de formulaire(s) de la page
- `links[]` : tableau des liens de la page
- `links.length` : nombre de lien(s) de la page
- `anchors[]` : tableau des ancre(s) internes (``)
- `anchors.length` : nombre de d'ancre(s) interne(s)
- `images[]`
- `applets[]`
- `embeds[]`

Remarque : les tableaux contiennent les éléments dans l'ordre de leur apparition dans le code HTML

DOM : 2. Objet Document

Méthodes

- `write(string)` : écrit une chaîne dans le document
- `writeln(string)` : idem + caractère de fin de ligne
- `clear()` : efface le document
- `close()` : ferme le document

Ex: `document.write("Bonjour !");`

DOM : 3. Objet Form

[JS-Exemp-6.html](#)

Propriétés

- **name** : nom (unique) du formulaire
- **method** : méthode de soumission (0=GET, 1=POST)
- **action** : action déclenchée par la validation du formulaire
- **target** : fenêtre de destination de la réponse
(si elle existe)
- **elements[]** : tableau des éléments du formulaires
- **length** : nombre d'éléments du formulaire

DOM : 3. Objet Form

- **Méthodes**

- `submit()` : soumet le formulaire
- `reset()` : ré-initialise le formulaire

- **Événements**

- `onSubmit(method)` : action à réaliser lorsque le formulaire est soumis
- `onReset(method)` : action à réaliser lorsque le formulaire est réinitialisé

DOM : 4. Objet Navigator

Propriétés

- `appCodeName` : nom de code interne du navigateur
- `appName` : nom réel du navigateur
- `appVersion` : version du navigateur
- `userAgent` : objet complexe contenant des détails sur :
 - l'`appCodeName`,
 - l'`appVersion`
 - le système d'exploitation utilisé
- `plugins[]` : tableau des plugins installés chez le client
- `mimeTypes[]` : tableau des types MIME supportés par le navigateur

Exemple : ajouter des éléments dans le DOM

```
<html>                                                 Hello.html
  <head>
    <link rel='stylesheet' type='text/css' href='hello.css' />
    <script type='text/javascript' src='hello.js'></script>
  </head>
  <body>
    <p id='hello'> hello </p>
    <div id='empty'> </div>
  </body>
</html>
```

hello.js

Hello.js

```
window.onload=function() {
    var vemp=document.getElementById('empty');
    addNode(vemp,"Contenu");
    addNode(vemp,"Cours JS & DOM!");
    var children= vemp.childNodes;
    for (var i=0;i<children.length;i++){
        children[i].className='Nouvelle';
    }
    vemp.style.border='solid green 2px';
    vemp.style.width="200px";
}

function addNode(el,text) {
    var childEl=document.createElement("div");
    el.appendChild(childEl);
    var txtNode=document.createTextNode(text);
    childEl.appendChild(txtNode);
}
```



hello.js

```
window.onload=function() {
    var vemp=document.getElementById('empty');
    addNode(vemp,"Contenu");
    addNode(vemp,"Cours JS & DOM!");
    var children= vemp.childNodes;
    for (var i=0;i<children.length;i++){
        children[i].className=' Nouvelle';
    }
    vemp.style.border='solid green 2px';
    vemp.style.width="200px";
}

function addNode(el,text) {
    var childEl=document.createElement("div");
    el.appendChild(childEl);
    var txtNode=document.createTextNode(text);
    childEl.appendChild(txtNode);
}
```

Syntaxe

Object document.getElementById(String id)

Description

Retourne un objet HTML à partir de son id, défini dans la propriété id de la balise de l'objet.

hello.js

```
window.onload=function() {
    var vemp=document.getElementById('empty');
    addNode(vemp,"Contenu");
    addNode(vemp,"Cours JS & DOM!");
    var children= vemp.childNodes;
    for (var i=0;i<children.length;i++){
        children[i].className=' Nouvelle';
    }
    vemp.style.border='solid green 2px';
    vemp.style.width="200px";
}

function addNode(el,text) {
    var childEl=document.createElement("div");
    el.appendChild(childEl);
    var txtNode=document.createTextNode(text);
    childEl.appendChild(txtNode);
}
```

Syntaxe

Object document.createElement(String id)

Description

Créer un nouvel élément HTML
en prenant le type de balise en argument.

hello.js

```
window.onload=function() {
    var vemp=document.getElementById('empty');
    addNode(vemp,"Contenu");
    addNode(vemp,"Cours JS & DOM!");
    var children= vemp.childNodes;
    for (var i=0;i<children.length;i++){
        children[i].className='Nouvelle';
    }
    vemp.style.border='solid green 2px';
    vemp.style.width="200px";
}

function addNode(el,text) {
    var childEl=document.createElement("div");
    el.appendChild(childEl);
    var txtNode=document.createTextNode(text);
    childEl.appendChild(txtNode);
}
```

Syntaxe

Object `el.appendChild(fils)`

Description

ajoute un nœud fils au noeud el.

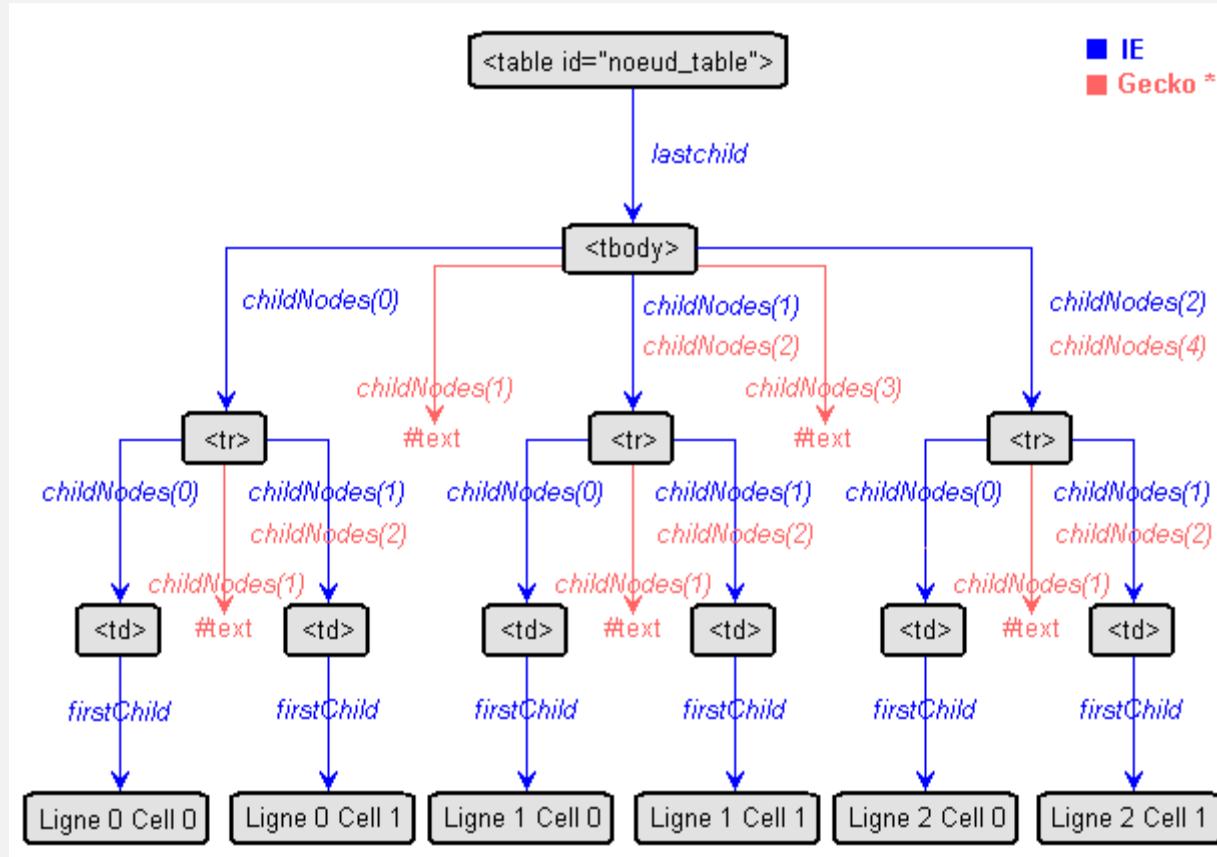
EXEMPLE MANIPULATION D'UN TABLEAU AVEC LE DOM

Modifier un tableau avec DOM

```
<table id=" noeud_table " border="1" cellpadding="4" cellspacing="0" summary="Tableau à 3 lignes et 2 colonnes">
    <tr>
        <td>Ligne 0 Cellule 0</td>
        <td>Ligne 0 Cellule 1</td>
    </tr>
    <tr>
        <td>Ligne 1 Cellule 0</td>
        <td>Ligne 1 Cellule 1</td>
    </tr>
    <tr>
        <td>Ligne 2 Cellule 0</td>
        <td>Ligne 2 Cellule 1</td>
    </tr>
</table>
```

[JS-Exemp-7.html](#)

Modifier un tableau avec DOM



Modifier un tableau avec DOM

Récupérer toutes les lignes du tableau

- var tableau = document.getElementById("noeud_table");
- var lignes = tableau.getElementsByTagName("TR");

où **lignes** est un tableau contenant tous les nœuds TR.

Modifier un tableau avec DOM

- Accéder à la 2^{ème} cellule de la 1^{ère} ligne

- var **tableau** = document.getElementById("noeud_table");
- var **lignes** = **tableau**.getElementsByTagName("TR");
- var cells = **lignes[0]**.getElementsByTagName("TD");
- var cell_1_2 = cells[1];
La variable cell_1_2 contient l'objet correspondant à la 2^{ème} cellule de la 1^{ère} ligne du tableau (pas la valeur de la cellule)

Modifier un tableau avec DOM

On peut aller un peu plus vite ainsi :

- var tableau = document.getElementById("noeud_table");
- var ligne_1 = tableau.getElementsByTagName("TR")[0];
- var cell_1_2 = ligne1.getElementsByTagName("TD")[1];

Modifier un tableau avec DOM

On peut ainsi changer le contenu de cette cellule :

- **cell_1_2.firstChild.nodeValue = 'blabla';**

Modifier l'attribut de cellules

Supposons qu'on veuille mettre cette 2^{ème} cellule de la 1^{ère} ligne en gras.
Il faudra y ajouter un attribut style :

- **cell_1_2.setAttribute('style', 'font-weight:bold');**

Modifier un tableau avec DOM

Ajouter une nouvelle ligne

```
var tableau = document.getElementById("noeud_table");
var dernière_ligne = document.createElement("TR");
Var tbody = tableau.lastChild;
for (i=0;i<2;i++) {
    var cell = document.createElement("TD");
    var texte = document.createTextNode(cellules[i]);
    cell.appendChild(texte);
    dernière_ligne.appendChild(cell);
}
tbody.appendChild(dernière_ligne);
```

Modifier un tableau avec DOM

Supprimer une ligne

Maintenant on va supprimer la 1^{ère} ligne :

- var tableau = document.getElementById("noeud_table");
- var premiere_ligne = tableau.getElementsByTagName("TR")[0];
- **tableau.removeChild(premiere_ligne);**

Le code **parent.removeChild(enfant)** permet de supprimer l'élément, ainsi que les éléments que celui-ci contient.

Javascript & DOM

Pour en savoir plus :

Description des tous les objets du DOM :

https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model

Cours complet Javascript + DOM :

<https://www.pierre-giraud.com/javascript-apprendre-coder-cours/presentation-dom/>

Tutorial Javascript + DOM :

<https://codes-sources.commentcamarche.net/faq/11071-manipuler-le-document-object-model-dom>

Cours avec exemples W3 school :

https://www.w3schools.com/jsref/prop_html_innerhtml.asp

Exercices DOM :

<https://www.codingame.com/playgrounds/53589/exercices-dom/1--selection-suppression-delement>