



YASMINE BOUAMRA

Contact : yasmine.trainings@gmail.com.



0

1

INITIATION À LA MANIPULATION DU DOM

QU'EST-CE QUE LE DOM ?

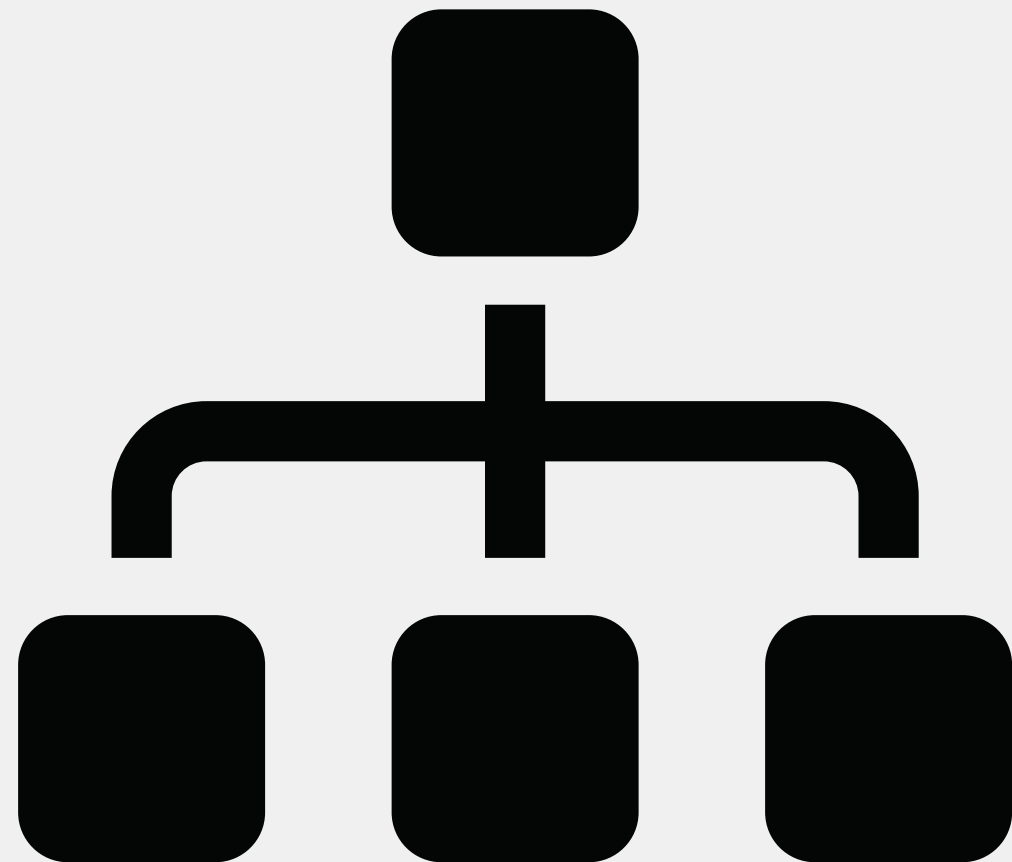
- 02

DOM est l'acronyme pour Document Objet Model. C'est la structure logique d'un fichier HTML (ou XML).

Le DOM se présente sous forme de structure en arbre. Les différents composants du fichier HTML constituent l'arborescence.

QUELLE EST LA STRUCTURE DU DOM ?

03

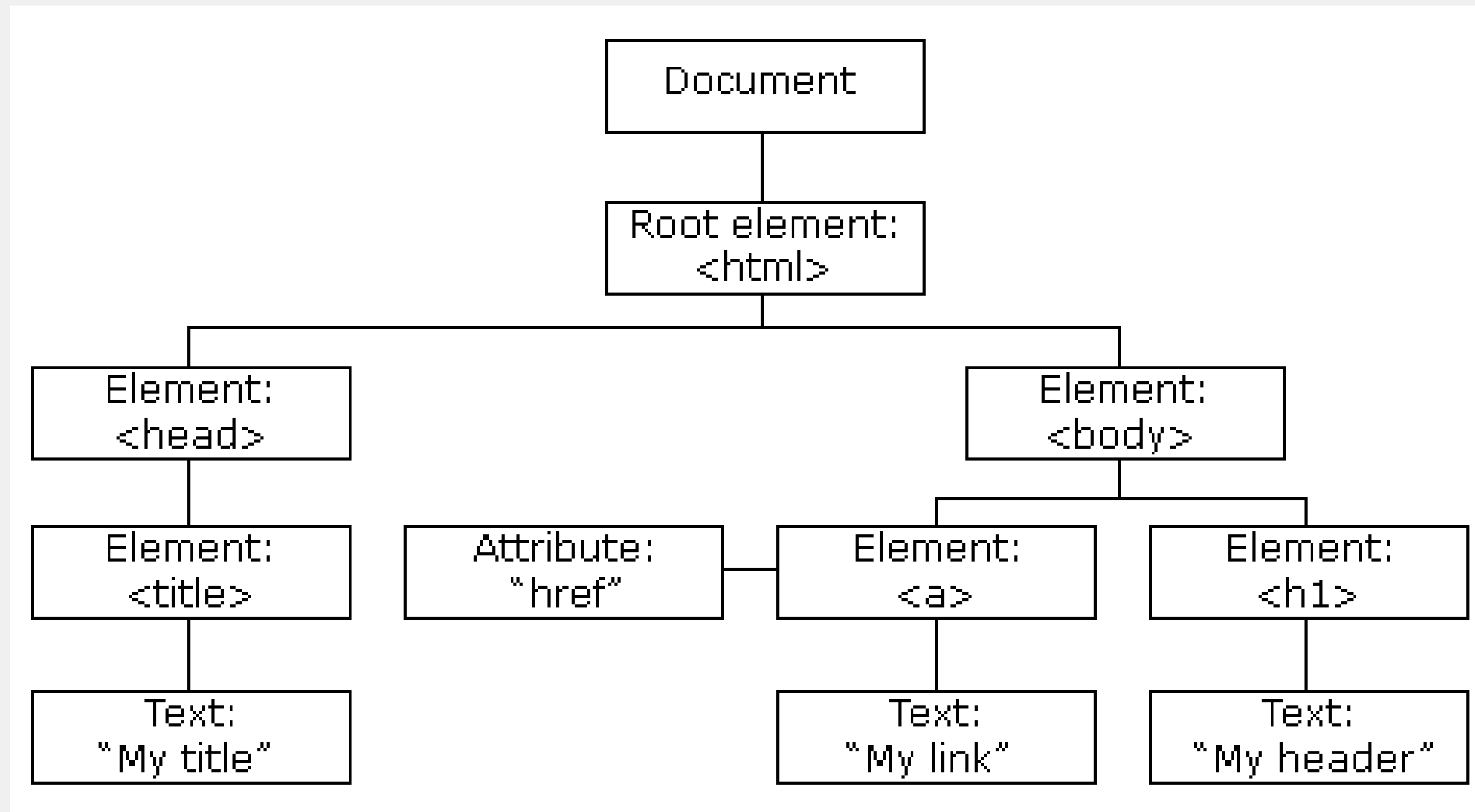


UNE STRUCTURE TREE-LIKE

Les composants du DOM sont représentés sous forme d'arborescence.

DÉROULEMENT D'UN EXEMPLE

04





PLUSIEURS TYPES DE NOEUDS

Nous l'avons donc compris, la structure du DOM se représente comme des relations parent-fils entre différents nœuds. Tous les nœuds n'ont pas le même sens et c'est ce que nous allons voir.

PLUSIEURS TYPES DE NOEUDS

ELEMENT_NODE	Représente un nœud élément (comme p ou div par exemple)
TEXT_NODE	Représente un nœud de type texte
COMMENT_NODE	Représente un nœud commentaire
DOCUMENT_NODE	Représente le nœud formé par le document en soi
DOCUMENT_TYPE_NODE	Représente le nœud doctype


EXERCICE

Représenter graphiquement le DOM de ce fichier HTML.

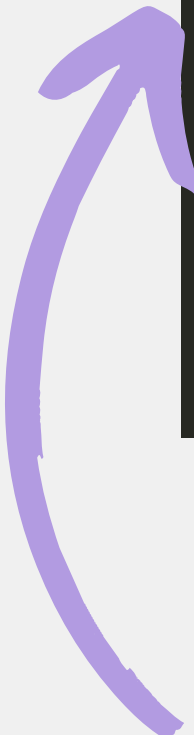
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Cours JavaScript</title>
6  </head>
7  <body>
8      // Nous allons présenter le cours de JavaScript
9      <h1>
10         Initiation à JavaScript
11     </h1>
12     <p>
13         Durant deux jours, nous allons comprendre les bases de JS.
14     </p>
15     <div>
16         <h1>Initiation à la manipulation du DOM</h1>
17         <p>Nous allons comprendre ce qu'est le DOM et comment le manipuler.
18         Vous pouvez consulter le sommaire <a href="#">ici</a>.
19         </p>
20     </div>
21 </body>
22 </html>
```


08

LES NOEUDS TEXTES PARASITES



Pour compléter le graphe représentant le DOM construit à partir du document précédent, nous devons ajouter les nœuds textes parasites. Ces derniers représentent les retours à la ligne ou un espace.



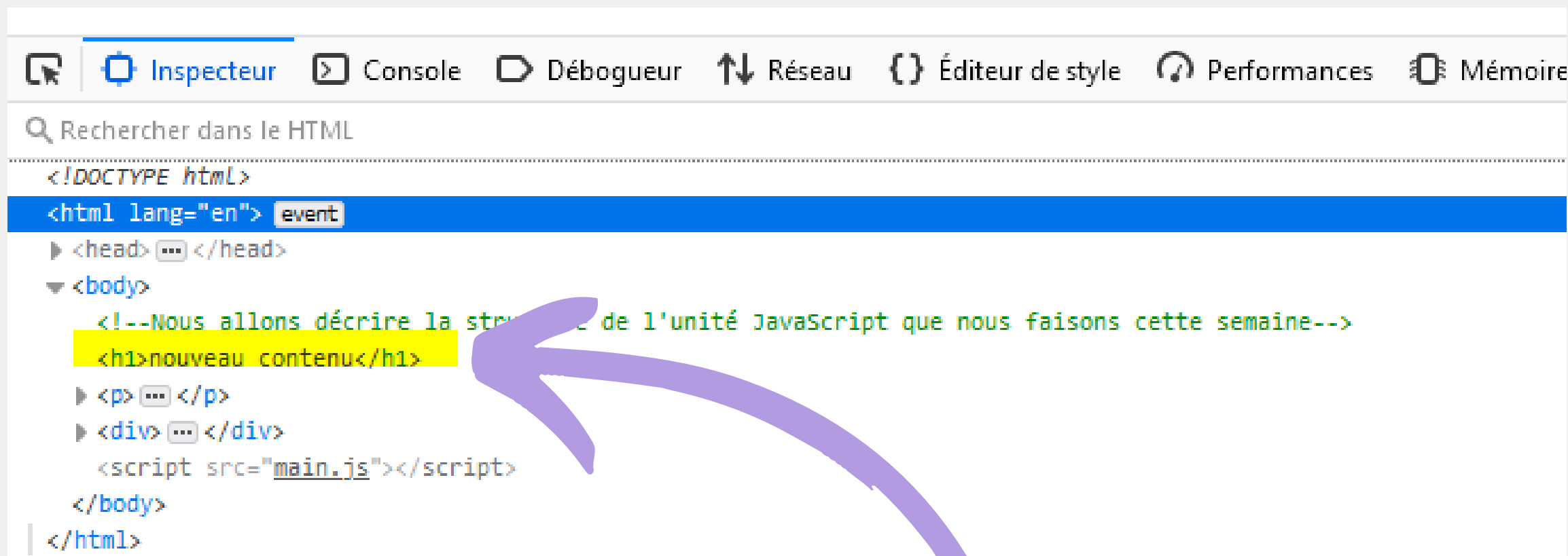
```
<p>
  Durant deux jours, nous allons comprendre
</p>
<div>
  <h1>Initiation à la manipulation du DOM</h1>
  <p>Nous allons comprendre ce qu'est le DOM
  Vous pouvez consulter le sommaire <a href="#">
  </p>
</div>
```

Présence d'un nœud parasite de retour à la ligne.

LA DIFFÉRENCE ENTRE LE DOM ET L'HTML

Le contenu du DOM est dynamique. Quand vous modifiez le contenu d'un nœud texte, il sera modifié dans le DOM.

Le contenu de votre fichier HTML ne changera pas durant l'exécution, quel que soient les modifications créées par les scripts.



```
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body>
    <!--Nous allons décrire la structure de l'unité JavaScript que nous faisons cette semaine-->
    <h1>nouveau contenu</h1>
    <p>
    </p>
    <div>
    </div>
    <script src="main.js"></script>
  </body>
</html>
```



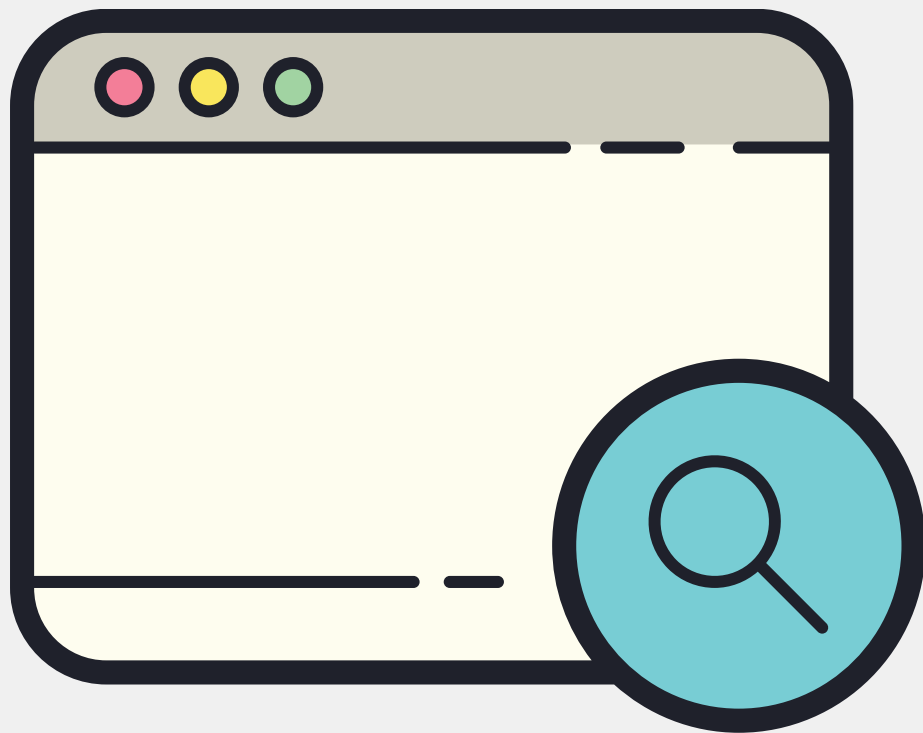
Nous allons modifier dynamiquement le premier H1 du fichier.

Il contenait "Initiation à JavaScript". Nous allons mettre nouveau contenu.

Avec cet exemple, nous pourrions mieux expliquer la différence entre le DOM et le fichier HTML.

Le DOM est un modèle qui se compose de nombreuses interfaces. C'est à travers ces interfaces qu'on va pouvoir manipuler notre HTML.

UNE INTERFACE, C'EST QUOI ?



En JS, les interfaces sont en quelque sorte des collections de fonctions qui demandent que l'objet ait une certaine méthode ou attribut pour être utilisé. Elles utilisent le concept de Duck Typing.

DUCK TYPING

Duck Typing est un concept qui détermine l'adéquation d'un objet pour une fonction. Dans les langages de programmations classique, elle se détermine par le typage. En JavaScript, elle se détermine à travers l'existence des paramètres de la fonction dans la variable passée en argument.

Les interfaces du DOM

- L'interface Event qui représente tout événement qui a lieu dans le DOM
- L'interface EventTarget qui est une interface que vont implémenter les objets qui peuvent recevoir des événements
- L'interface Node qui est l'interface de base pour une grande partie des objets du DOM
- L'interface Document qui représente le document actuel et qui va être l'interface la plus utilisée
- L'interface Element qui est l'interface de base pour tous les objets d'un document ;

MANIPULATION DU DOM

Parlons pratique

Accéder à un élément

Comme nous l'avons expliqué, le DOM va nous permettre de manipuler les éléments de l'HTML et leur style.

Pour cela, nous devons dans un premier temps y accéder.

Pour cela, nous introduisons `querySelector`.

`querySelector` est une méthode implémentée à travers des interfaces `Document` et `Element`.

querySelector()

16

Pour récupérer un élément à travers `querySelector()`, il suffit de le sauvegarder dans une variable.

Par exemple, on suppose un document contenant un paragraphe `p`. On le récupère comme suit.

```
querySelector > exemple > JS main.js > [e] paragraph  
1  let paragraph = document.querySelector("p");
```

EXERCICE

Récupérer les différents éléments du fichier HTML ci-joint.

Imprimer les éléments récupérés dans la console.

Que remarquez-vous ? Quels sont les problèmes soulevés ?

Quelle solution suggérez-vous ?

querySelectorAll()

Lorsque l'on a besoin de sélectionner tous les éléments du même type de balise/de la même classe, il faut utiliser `querySelectorAll()`.

`querySelectorAll()` renvoie une liste de type `NodeList`.
C'est une liste statique (qui n'est pas affectée par les changements dans le DOM) des objets du sélecteur sélectionné.
- On utilise `forEach` pour la parcourir.

querySelectorAll()

```
1 let paragraph = document.querySelectorAll("p");  
2 console.log(paragraph);
```



Console :

```
▼ NodeList [ p#p1 , p#p2 ]  
  ▶ 0: <p id="p1">  
  ▶ 1: <p id="p2">  
    length: 2  
  ▶ <prototype>: NodeListPrototype { item: item(), keys: keys(), values: values(), ... }
```

EXERCICE

- 1) Parcourez la liste des nœuds paragraphes et affichez les ainsi que leurs index.
- 2) Soit le code suivant, si on utilisait `querySelector()` sur les éléments `div`, que contiendrait la liste ? Codez le.

```
<div>  
  <h1>ceci est une div</h1>  
  <p>ceci est un paragraph dans une div</p>  
  <!-- Ceci est un commentaire dans une div -->  
</div>
```

getElementsByClassName

Nous pouvons sélectionner des éléments en utilisant leur classe. Pour cela, on peut soit utiliser `querySelector` comme vu précédemment ou `getElementsByClassName`.

```
let p_div = document.getElementsByClassName("p-div");  
console.log(p_div);
```

```
▼ HTMLCollection { 0: p.p-div , 1: p.p-div , length: 2 }  
  ► 0: <p class="p-div">   
  ► 1: <p class="p-div">   
    length: 2  
  ► <prototype>: HTMLCollectionPrototype { item: item(), namedItem: namedItem(), length: Getter, _ }
```

getElementById

Il est également possible de sélectionner un élément grâce à son ID.

```
let paragraph = document.getElementById("p1");  
console.log(paragraph);
```

```
▶ <p id="p1" class="paragraph"> ☒
```

ACCÉDER AU CONTENU

A présent que l'on sait sélectionner des éléments, nous allons voir comment accéder à leurs contenus. Il existe différents attributs que l'on peut utiliser :

- `innerHTML` pour récupérer le contenu entre les deux balises ouvrantes/fermantes,
- `outerHTML` pour récupérer la totalité du contenu, incluant les balises,
- `textContent` pour récupérer l'intégralité du contenu textuel d'un élément
- `innerText` pour récupérer uniquement le texte visible d'un élément.


```
let p_div1 = document.getElementById("p-div1");
console.log(p_div1.innerHTML);
console.log(p_div1.outerHTML);
console.log(p_div1.innerText);
console.log(p_div1.textContent);
```



Bonjour
 Premier paragraphe de la div

<p class="p-div" id="p-div1">Bonjour
 Premier paragraphe de la div</p>

Bonjour

Premier paragraphe de la div

Bonjour Premier paragraphe de la div

MODIFIER LE STYLE

Il est également possible de modifier les attributs CSS des éléments sélectionnés.

```
p_div1.style.color="blue";  
p_div1.style.fontFamily="Montserrat";
```



Bonjour
Premier paragraphe de la div

EXERCICE

- Récupérez tous les éléments "refrains".
Montrez deux manières de le faire.
- Récupérez le couplet 2 et affichez le dans la console.
- Modifier la couleur des refrains pour bleu.
- Modifier la police des couplets pour la police de votre choix.

EVENTS

Les évènements sont des réactions à des actions prises par l'utilisateur.

Lorsque l'on clique sur un bouton et que quelque chose se produit
-> c'est un évènement.

EventListener

Pour pouvoir créer une réaction à un évènement, il faut savoir qu'il a eu lieu.

Pour cela, on va "écouter" et attendre qu'il se produise. Pour cela, on utilise des EventListener.

l'interface Element met à notre disposition la fonction `addEventListener` qui nous permet de traiter un évènement sur un élément.

SYNTAXE

addEventListener a deux paramètres : le nom de l'évènement et la fonction qui permet de le gérer.

```
let button = document.querySelector("button");  
button.addEventListener("name",function traitement(){  
    // instructions  
})
```

TOUS LES ÉVÈNEMENTS

Il existe de nombreux évènements, on vous invite à les chercher au fur et à mesure, selon vos besoins.

Voici en attendant, quelques évènements communément utilisés :

- onclick : lors qu'un clic se produit .
- keyUp : lors qu'une touche clavier est relâchée.
- keyDown : lorsqu'une touche de clavier est pressée.



EXERCICE

- Mettre les refrains en bleu et les couplets en rouge sur le clic du bouton correspondant.
- Récupérer le deuxième couplet et lui changer de font (mettez la police de votre choix).
- Récupérer le dernier refrain et le mettre en italique.

EXERCICE

Jeu : plus ou moins

Créez une page HTML avec un input-field, un bouton, et du texte expliquant le jeu

- Lorsque la page se charge, un nombre aléatoire est généré (réutiliser la fonction qu'on a utilisé dans l'app Pokemon)
- L'utilisateur rentre ses suppositions dans l'input-field en appuyant sur le bouton.



EXERCICE

Jeu : plus ou moins

- Pour chaque supposition, vous allez tester si elle vaut plus ou moins que le nombre généré et l'afficher sur la page.

- L'affichage doit se faire en dessous de l'input field.

NB : le nombre généré doit être un nombre naturel.

NB2 : les traitements doivent se faire dans des fonctions.



02

NAVIGUER DANS LE DOM

Naviguer dans le DOM

Nous allons à présent introduire les notions de navigation dans le DOM.

Parfois, il n'est pas possible d'accéder à un élément à travers son sélecteur ou ce n'est pas optimisé de le faire. Dans ce cas, il est possible de naviguer de nœuds en nœuds.

PARENT-CHILDREN

Pour cela, nous allons exploiter les relations parents-enfants qui se créent dans le DOM.

L'interface Node offre des méthodes qui permettent d'accéder aux parents d'un élément et à ses enfants.

- parentNode renvoie le parent (ou null si absence de parent)
- childNodes renvoie une liste de noeuds enfants

parentNode()

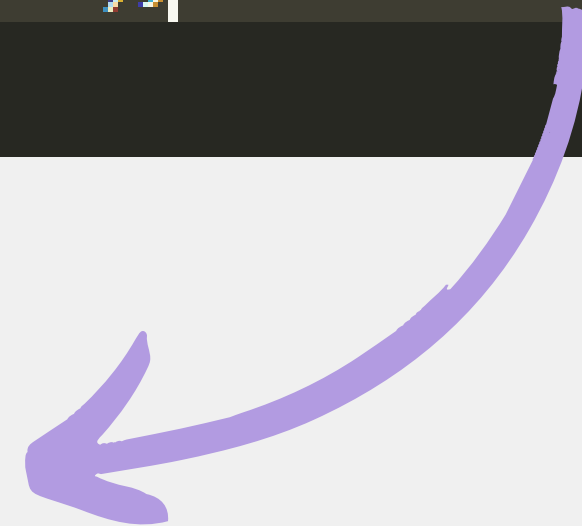
37

parentNode() renvoie le noeud parent quelque soit son type. Par conséquent, si le parent est un documentNode, ça sera le document qui sera renvoyé.

```
getElementBy > exemple > JS main.js > ...  
1   let div = document.querySelector("div");  
2   let parentDiv = div.parentNode;  
3   console.log(parentDiv);
```

▶ <body> 🛠

TSS: hosted page injected



EXERCICE

Que renvoie le `p.parentNode` étant donné la structure HTML suivant ?

```
<div>
  <button>bouton</button>
  <p class="p-div" id="p-div1">Bonjour<br> Premier paragraphe de la div</p>
  <p class="p-div">Second paragraphe de la div
    <div>
      <p id="p-div3">je suis un paragraphe dans la div</p>
    </div>
  </p>
</div>
```

```
let p = document.getElementById("p-div3");
console.log(p.parentNode);
```

parentElement

Dans le cas où l'on souhaite cibler un noeud parent seulement si c'est un noeud élément, on utilise `parentElement`.

```
let div = document.querySelector("div");  
let parentDiv = div.parentElement  
console.log(parentDiv);
```





EXERCICE

Dans une structure HTML de base, quelles balises n'ont pas un parent element.

childNodes

41

On peut accéder aux enfants grâce à l'attribut childNodes. Il renvoie une liste d'enfants que l'on peut parcourir.



```
1 let div = document.querySelector("div");
2 let divChildren = div.childNodes;
3 console.log(divChildren);
4
```

```
▼ NodeList(10) [ #text, button, #text, p#-div1.p-div, #text, p.p-div, div, #text, p, #text ]
  ▶ 0: #text
    "
  ▶ 1: <button>
  ▶ 2: #text
    "
  ▶ 3: <p id="p-div1" class="p-div">
  ▶ 4: #text
    "
  ▶ 5: <p class="p-div">
  ▶ 6: <div>
  ▶ 7: #text
    "
  ▶ 8:
  ▶ 9:
```

EXERCICE


Quel est le nombre d'enfants de la balise body ?

```
9  <body>
10    <h1>Bonjour, bienvenue sur cete page</h1>
11    <p class="paragraph" id="p1">Ceci est le premier paragraph</p>
12    <p class="paragraph" id="p2">Ceci est le second paragraph</p>
13    <div>
14      <button>bouton</button>
15      <p class="p-div" id="p-div1">Bonjour<br> Premier paragraphe de la div</p>
16      <p class="p-div">Second paragraphe de la div
17        <div>
18          <p id="p-div3">je suis un paragraphe dans la div</p>
19        </div>
20      </p>
21    </div>
22    <p class="paragraph" id="p3">Ceci est le dernier paragraph</p>
23    <script src="main.js"></script>
24  </body>
```

CHILDREN

43

Dans le cas où l'on veut uniquement accéder aux enfants noeuds éléments, on utilise l'attribut children.



```
1 let div = document.querySelector("div");
2 let divChildren = div.children;
3 console.log(divChildren);
4
```

```
▼ HTMLCollection { 0: button , 1: p#p-div1.p-div , 2: p.p-div , 3: div , 4: p , length: 5, _ }
  ▶ 0: <button>
  ▶ 1: <p id="p-div1" class="p-div">
  ▶ 2: <p class="p-div">
  ▶ 3: <div>
  ▶ 4: <p>
    length: 5
  ▶ "p-div1": <p id="p-div1" class="p-div">
  ▶ <prototype>: HTMLCollectionPrototype { item: item(), namedItem: namedItem(), length: Getter, _ }
```

EXERCICE

Quel est le nombre d'enfants éléments de body ?

```
<body>
  <h1>Bonjour, bienvenue sur cete page</h1>
  <p class="paragraph" id="p1">Ceci est le premier paragraph</p>
  <p class="paragraph" id="p2">Ceci est le second paragraph</p>
  <div>
    <button>bouton</button>
    <p class="p-div" id="p-div1">Bonjour<br> Premier paragraphe de la div</p>
    <p class="p-div">Second paragraphe de la div
      <div>
        <p id="p-div3">je suis un paragraphe dans la div</p>
      </div>
    </p>
  </div>
  <p class="paragraph" id="p3">Ceci est le dernier paragraph</p>
  <script src="main.js"></script>
</body>
</html>
```

firstChild lastChild

45

Il est possible d'accéder directement au premier ou au dernier noeud d'un noeud cible dans le DOM. Pour cela, on utilise les attributs firstChild et lastChild respectivement pour le premier et le dernier noeud.

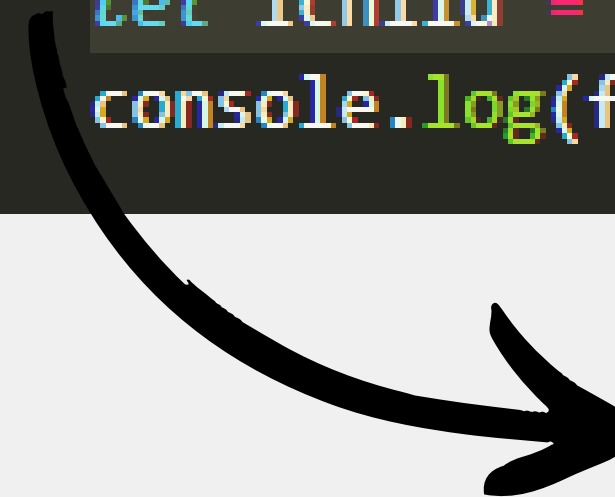
```
1 let div = document.querySelector("div");
2 let fchild = div.firstChild;
3 let lchild = div.lastChild;
4 console.log(fchild, lchild);
```

```
▶ HTMLCollection { 0: button , 1: p#p-div1.p-div , 2: p.p-div , 3: div , 4: p , length: 5, _ }
  ▶ #text "
    "
  ▶ #text "
    "
```

firstElementChild & lastElementChild

Vous aurez remarquer que cela renvoie des éléments #text qui sont des noeuds parasites. Il peut être intéressant dans ce cas là de récupérer le premier et le dernier enfant élément comme suit.

```
1  let div = document.querySelector("div");  
2  let fchild = div.firstElementChild;  
3  let lchild = div.lastElementChild;  
4  console.log(fchild, lchild);  
5
```



► <button> ◻ ► <p> ◻



EXERCICE

Comment parcourir une liste d'enfants sans
utiliser de boucle for ?

Pour finir, on va voir comment
créer, ajouter, et déplacer des
noeuds dans le DOM.

MODIFICATIONS ET AJOUTS

createElement()

Pour commencer, nous allons voir comment créer un élément grâce à JS.
On va utiliser la méthode createElement().

```
1  let newElement = document.createElement("p")
2  // la méthode prend en argument le type d'élément
3  // ici nous avons créé un paragraphe
4  newElement.textContent= "contenu textuel du paragraphe"
5  // nous ajoutons le contenu de l'élément qui est créé vide
6
```

EXERCICE

Créer une page HTML de base.
Ajouter deux paragraphes à la fin de la page en utilisant createElement.
Que remarquez-vous ?
Que feriez-vous pour avoir un noeud texte ou un noeud commentaire ?

PLACER UN NOEUD

Créer un noeud n'est pas suffisant. Il faut le placer dans le DOM pour qu'il s'affiche sur notre page web. Pour cela, nous allons utiliser les méthodes `append` et `prepend`

```
let b = document.body;  
// on va sélectionner le parent où on va placer le nouveau noeud  
// prepend permet de l'ajouter comme premier enfant  
b.prepend(newElement);  
// append permet de l'ajouter comme dernier enfant  
b.append(newElement);
```



EXERCICE

Compléter la question de l'exercice précédent en ajoutant les noeuds à la fin de la page.

Que faire si on souhaite insérer au milieu de la page, avant une div?

INSÉRER DES ÉLÉMENTS ADJACENTS

Il est possible d'insérer des éléments adjacents en sélectionnant un noeud et utiliser la méthode `insertAdjacentElement()`

```
let div = document.getElementById("div");  
div.insertAdjacentElement('beforeend', newElement);
```

Il est possible d'insérer directement du contenu HTML de la même manière en utilisant `insertAdjacentHTML()` !

```
let htmlContent = "<strong>Ceci est un texte ajouté</strong>";  
div.insertAdjacentHTML('beforeend', htmlContent);
```

ARGUMENTS

- **beforebegin** : Insère le ou les nœuds avant l'élément ;
- **afterbegin** : Insère le ou les nœuds avant le premier enfant de l'élément ;
- **beforeend** : Insère le ou les nœuds après le dernier enfant de l'élément ;
- **afterend** : Insère le ou les nœuds après l'élément

REMOVE

Pour finir, il est possible de supprimer des noeuds du DOM en utilisant la méthode `remove()`.

```
newElement.remove();
```

Il est également possible d'utiliser la méthode `removeChild()`. Elle renvoie le noeud supprimé qu'on peut stocker dans une variable.

```
let deleted = div.removeChild(newElement);
```




EXERCICE

- Reprenez l'exemple des refrains et des couplets. Créez un bouton qui permet de remplacer les refrains par les couplets et inversement.
- Créez un bouton qui permet de supprimer les refrains et les couplets.

DES QUESTIONS ? COMMENTAIRES ?

N'hésitez pas à partager vos commentaires.

ADRESSE E-MAIL

yasmine.trainings@gmail.com