10/08/2022

JS DOM

WAHIM POUR INSY2S

Qu’est-ce que le DOM?

Le terme DOM (Document Object Model) est un terme standardisé et donc défini de manière officielle.

Le DOM est une interface de programmation pour des documents HTML ou XML qui représente le document (la page web) sous une forme qui permet aux langages de script comme le JavaScript d’y accéder et d’en manipuler le contenu et les styles.

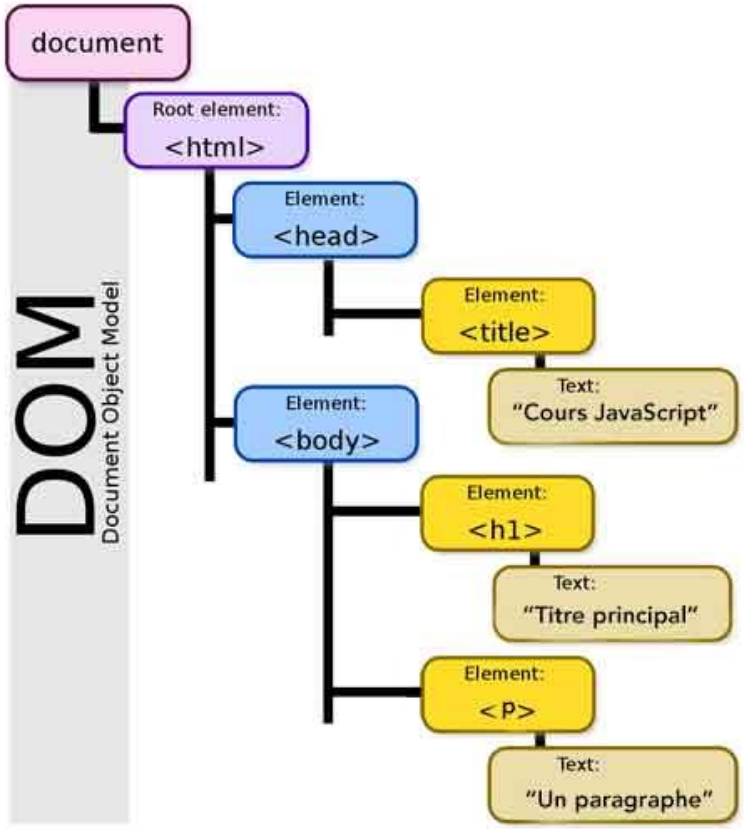
Le DOM est ainsi une représentation structurée du document sous forme **« d’arbre »** crée automatiquement par le navigateur.

Chaque branche de cet arbre se termine par ce qu’on appelle un nœud qui va contenir des objets.

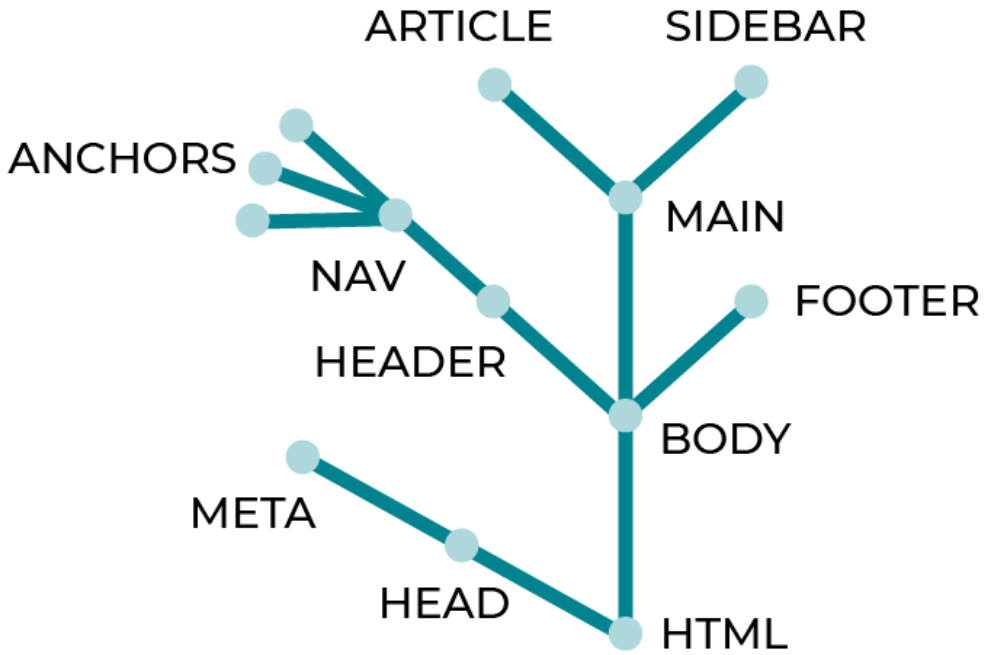
Qu’est-ce que le DOM?

Dans le DOM, on commence toujours par un élément racine qui est le point de départ du document : la balise ***<html>*** . Celle-ci a pour enfants les balises ***<head>*** et ***<body>*** qui ont donc un parent commun : la balise ***<html>*** ! Vous trouverez ensuite le contenu de votre page dans la balise ***<body>*** sous forme de liens, boutons, blocs, etc.

Qu’est-ce que le DOM?



Source: https://www.pierre-giraud.com



Source: https://openclassrooms.com/fr

Qu’est-ce que le DOM?

Avec une interface de programmation nous permettant de parcourir le DOM, nous allons pouvoir interagir avec lui. Ces interactions comprennent :

- La modification du contenu d'un élément précis ;
- La modification du style d'un élément ;
- La création ou la suppression d'éléments ;
- L'interaction avec les utilisateurs, afin de repérer des clics sur un élément ou encore de récupérer leur nom dans un formulaire ;
- Etc.

Accéder aux éléments du DOM

Chaque élément du DOM est un **objet** JavaScript avec ses propriétés et ses fonctions pour le manipuler.

Tout commence avec le ***document*** .

Cet **objet**, auquel vous avez directement accès dans votre code JavaScript, est le point de départ du DOM.

Il représente votre page (votre document) entière.

C'est donc lui qui contient les **fonctions** dont vous aurez besoin pour retrouver les éléments que vous cherchez.

Accéder aux éléments du DOM

- `document.getElementById();`

C’est sans doute la méthode la plus utilisée pour retrouver un **élément précis**.

Comme son nom l'indique, elle va rechercher un élément grâce à son ***id*** , or, rappelez-vous qu'il ne doit y avoir qu'un seul élément avec un ***id*** donné dans le html, cette méthode est donc une candidate parfaite pour retrouver un élément particulier.

getElementById(« valeur de l’id ») prend en paramètre l' ***id*** de l'élément que vous recherchez et vous retournera cet élément s'il a été trouvé.

Accéder aux éléments du DOM

Exemple : Prenons ce contenu HTML

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no" />
    <title>Cours JS</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <h1 class="titre">Cours JS</h1>
    <p class="para1">
      Lorem ipsum dolor sit, amet consectetur
adipisicing elit. Illo maxime
      explicabo cupiditate ipsam autem.
Estnecessitatibus, architecto esse
      quaerat hic repellat libero nisi ad eos ex vero
at in velit.
    </p>
```

```
<section class="container">
      <form id="my-form">
        <h1>Add User</h1>
        <div class="msg"></div>
        <div>
          <label for="name">Name:</label>
          <input type="text" id="name" name="name" />
        </div>
        <div>
          <label for="email">Email:</label>
          <input type="text" id="email" name="email" />
        </div>
        <input class="btn" type="submit" value="Submit"
/>
      </form>
      <ul id="users"></ul>
    </section>
    <script src="main.js"></script>
  </body>
</html>
```

Accéder aux éléments du DOM

Créons maintenant un fichier main.js et ajoutons le code suivant :

```
const form=document.getElementById("my-form");
console.log(form);
```

En observant la console de votre navigateur, vous constaterez que celle-ci a été récupérer l'élément du DOM qui contient l'attribut ***id*** avec la valeur « ***my-form*** » pour l'afficher.

D'autres exemples : https://developer.mozilla.org/fr/docs/Web/API/Document/getElementById

Accéder aux éléments du DOM

- `document.getElementsByClassName();`

Cette méthode fonctionne de la même manière que la précédente, mais fera sa recherche sur la ***class*** des éléments et retournera la liste des éléments qui correspondent.

Exemple :

```
const container = document.getElementsByClassName("container");
console.log(container);
```

Ceci récupère tous les éléments qui ont l'attribut ***class*** avec la valeur « ***container*** »

D'autres exemples :
https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByClassName

Accéder aux éléments du DOM

- `document.getElementsByName();`

Avec cette méthode, vous rechercherez tous les éléments avec l'attribut ***name*** ayant la valeur précisée dans la parenthèse.

Exemple :

```
const inputMail = document.getElementsByName("email");
console.log(inputMail);
```

D'autres exemples :
https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByName

Accéder aux éléments du DOM

- `document.getElementsByTagName();`

Pour cette méthode, vous rechercherez tous les éléments avec un nom de balise bien précis (par exemple tous les liens (***a***), tous les boutons (***button***)...).

De la même manière que la méthode précédente, vous récupérerez la liste des éléments correspondants.

Exemple :

```
const ul = document.getElementsByTagName("ul");
console.log(ul);
```

D'autres exemples :
https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByTagName

Accéder aux éléments du DOM

- `document.querySelector();`

Cette méthode est plus complexe, mais aussi beaucoup plus puissante.

Elle renvoie le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés dans le document.

Exemple :

```
const labelFor = document.querySelector("label[for=email]");
console.log(labelFor);
```

La valeur dans la parenthèse est un sélecteur CSS, celle de l'exemple permet d'aller récupérer l'élément dont la balise est ***label***, dont l'attribut est ***for*** et dont la valeur de ***for*** est ***email***.

Accéder aux éléments du DOM

- `document.querySelectorAll();`

La méthode querySelector() ne renvoie que le premier élément qui correspond aux sélecteurs spécifiés.

Pour renvoyer toutes les correspondances, utilisez plutôt la méthode querySelectorAll().

Exemple :

```
const allInputs = document.querySelectorAll("input");
console.log(allInputs);
```

D'autres exemples : https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector

https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll

Modifier le DOM

➢Modifiez le contenu d'un élément

Pour commencer, voyons déjà les propriétés permettant de modifier directement le contenu de notre élément.

Les deux principales sont : ***innerHTML*** et ***textContent*** .

innerHTML demande à ce que vous entriez du texte représentant un contenu HTML.

Exemple :

```
document.querySelector(".para1").innerHTML = "<span class='p2'>para de classe p2 </span>";
```

Modifier le DOM

La propriété ***textContent*** , quant à elle, demande un simple texte qui ne sera pas interprété comme étant du HTML.

Exemple :

```
document.querySelector(".para1").textContent = "para modifié";
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/Element/innerHTML

https://developer.mozilla.org/fr/docs/Web/API/Node/textContent

Modifier le DOM

➢Modifiez des classes

Il est aussi possible d'accéder directement à la liste des classes d'un élément avec la propriété ***classList*** .

Cette propriété ***classList*** fournit aussi une série de fonctions permettant de modifier cette liste de classes.

En voici quelques-unes :

- ***add*** => Ajoute les classes spécifiées
- ***remove*** => Supprime les classes spécifiées
- ***toggle*** => change la présence d'une classe dans la liste. Si la classe existe, alors la supprime et renvoie false, dans le cas inverse, ajoute cette classe et retourne true.

Modifier le DOM

Exemples :

```
document.querySelector("p").classList.add("paragraphe"); // Ajoute l'attribut class et sa valeur paragraphe à la balise p
document.querySelector("p").classList.remove("paragraphe"); // Supprime la valeur de class paragraphe à la balise p
document.querySelector("p").classList.toggle("paragraphe"); // si « paragraphe » est défini, le supprime, sinon, l'ajoute
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/Element/classList#Méthodes

Modifier le DOM

➢**Changez les styles d'un élément**

Avec la propriété ***style*** , vous pouvez récupérer et modifier les différents styles d'un élément.

style est un objet qui a une propriété pour chaque style existant.

Par exemple, pour modifier la couleur d'arrière-plan d'un élément, vous ferez :

```
document.getElementById('my-form').style.backgroundColor = '#ccc';
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/HTMLElement/style

Modifier le DOM

➢Modifiez les attributs

Pour définir ou remplacer les attributs d'un élément, vous pouvez utiliser la fonction ***setAttribute*** .

Exemples :

```
document.getElementById("name").setAttribute("type", "password");
// Change le type de l'input en un type password
document.getElementById("name").setAttribute("name", "my-password");
// Change le nom de l'input en my-password
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/Element/setAttribute

Modifier le DOM

➢**Créez de nouveaux éléments et les ajouter en tant qu'enfants**

La fonction ***document.createElement*** permet de créer un nouvel élément du type spécifié que nous pourrons insérer dans notre DOM.

Exemples :

```
const newElt = document.createElement("div");
```

Avec le code ci-dessus, nous venons de créer un nouvel élément de type ***div*** , mais qui n'est pas encore rattaché au DOM.

Nous allons ensuite récupéré l'élément ayant pour id ***my-form*** .

```
let elt = document.getElementById("my-form");
```

Modifier le DOM

Enfin, nous allons ajouté notre nouvel élément dans les enfants de l'élément ***#my-form*** .

```
elt.appendChild(newElt);
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/Node/appendChild

Modifier le DOM

➢Supprimez et remplacez des éléments

Il existe les fonctions ***removeChild*** et ***replaceChild*** , afin de respectivement supprimer et remplacer un élément.

Exemple :

Reprenons notre code précédent :

```
const newElt = document.createElement("div");
let elt = document.getElementById("my-form");
elt.appendChild(newElt);
```

Ajoutons ce qui suit :

```
elt.removeChild(newElt);
// Supprime l'élément newElt de l'élément elt
elt.replaceChild(document.createElement("article"), newElt);
// Remplace l'élément newElt par un nouvel élément de type article
```

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/Node/removeChild

Écouter des events (événements)

➢**Qu'est-ce qu'un événement ?**

Un **événement** ou **event** est une réaction à une action émise par l'utilisateur, comme le clic sur un bouton ou la saisie d'un texte dans un formulaire.

Un **événement** en JavaScript est représenté par un nom (***click*** , ***mousemove*** ...) et une fonction que l'on nomme une **callback** .

Un événement est par défaut propagé, c'est-à-dire que si nous n'indiquons pas à l'événement que nous le traitons, il sera transmis à l'élément parent, et ainsi de suite jusqu'à l'élément racine.

Écouter des events (événements)

Cette fonction **callback** , c'est nous qui allons la spécifier.

Elle sera appelée à chaque fois que l'action que l'on désire suivre est exécutée.

Cela signifie que si l'on désire suivre le clic sur un élément, notre fonction sera appelée à chaque fois que l'utilisateur cliquera sur cet élément.

Écouter des events (événements)

➢ **Événement lors d'un clic sur un élément**

Afin de réagir lors d'un **clic** sur un élément, il faut *écouter* cet **événement**.

Pour cela, nous avons à notre disposition la fonction ***addEventListener()*** .

Cette fonction nous permet d'écouter tous types d'événements (pas que le clic).

Réagir à un **événement**, c'est faire une action lorsque celui-ci se déclenche.
Écouter, c'est vouloir être averti quand **l'événement** se déclenche.

Écouter des events (événements)

Exemple :

```
const element = document.getElementsByClassName("btn");
// On récupère l'élément sur lequel on veut détecter le clic
element[0].addEventListener("click", function (e) {
// On écoute l'événement click
  e.preventDefault();
  document.getElementById("users").innerHTML = "<li>Hello World</li>";
// On change le contenu de notre élément pour afficher « Hello World"
});
```

Écouter des events (événements)

```
const element = document.getElementsByClassName("btn");
// On récupère l'élément sur lequel on veut détecter le clic
element[0].addEventListener("click", function (event) {
// On écoute l'événement click
  event.preventDefault();
  document.getElementById("users").innerHTML = "<li>Hello World</li>";
// On change le contenu de notre élément pour afficher « Hello World"
});
```

N.B.: *Rappelez-vous, l'attribut **class et sa valeur** ne sont pas forcément uniques dans notre **DOM,** de plus la méthode **getElementsByClassName** renvoie un objet de type tableau (**array** vu en JS pure) , il est donc indispensable de préciser que nous souhaitons récupérer l'élément en première position (**indice 0** dans un **array**) dans notre **DOM** même si il est seul.*
*En appelant ici la fonction **preventDefault()** dans notre **callback**, nous demandons au gestionnaire des événements de ne pas exécuter le comportement par défaut de notre élément (qui est la redirection vers une autre page pour un lien).*

Écouter des events (événements)

Pour en savoir plus : https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener

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

Exercice 1

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>Débuter Javascript</title>

  <link rel="stylesheet"
href="style.css">
</head>
```

```
<body>
    <div class="clickEvent">
        <button id="btn">Appuyez
!</button>
        <br>
        <img src="./456536.jpg" id="img"
alt="New York">
    </div>
    <script src="main.js"></script>
</body>
</html>
```

Exercice 1

```css
* {
  border-radius: 30px;
  padding: 8px;
  text-align: center;
}
body {
  min-height: 100vh;
  transition: .5s;
}
```

```css
img {
  margin: 0 auto;
  height: 200px;
  visibility: hidden;
}
.show {
  visibility: visible;
  opacity: 1;
}
```

Exercice 1

Utilisez ce que vous avez découvert dans ce cours sur le JS DOM pour faire en sorte que le bouton « Appuyer ! » de l'exercice fasse apparaître l'image de New-York quand celle-ci n'est pas visible et disparaître quand l'image est visible.

Exercice 2

```html
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>Débuter Javascript</title>
  <link rel="stylesheet"
href="style.css">
</head>
```

```html
<body>
  <div class="themeContainer">
    <div class="theme" id="dark"></div>
    <div class="theme"
id="yellow"></div>
    <div class="theme" id="green"></div>
  </div>
  <script src="main.js"></script>
</body>
</html>
```

Exercice 2

```
* {
  border-radius:
30px;
  padding: 8px;
}
.themeContainer {
  position: fixed;
  top: 10px;
}
.theme {
  height: 40px;
  width: 40px;
  cursor: pointer;
}
```

```
#dark {
  background: black;
}
#yellow {
  background: yellow;
}
#green {
  background: green;
}
```

```
.darkTheme {
  background: black;
  color: white;
}
.yellowTheme {
  background: yellow;
  color: white;
}
.greenTheme {
  background: green;
  color: white;
}
```

Exercice 2

Utilisez ce que vous avez découvert dans ce cours sur le JS DOM ainsi que dans le cours sur le JS classique pour faire en sorte que chaque pastille de couleur modifie le fond de la page pour qu'elle prenne la couleur choisie.

Un indice : revoir le mot clé ***switch*** dans le JS classique ou « Vanilla ».