

DÉFINITION DU DOM

QU'EST-CE QUE LE DOM ?

Le **DOM (Document Object Model)** est une représentation structurée du contenu HTML d'une page web. Il permet d'accéder et de manipuler l'arborescence du document avec des langages de programmation comme **JavaScript**.

SIGNIFICATION DE L'ACRONYME

- **D** : Document (HTML ou XML)
- **O** : Object (dans le sens programmation orientée objet)
- **M** : Model (modèle structuré et manipulable)

RÔLE ET UTILITÉ DANS UN NAVIGATEUR WEB

Le **DOM** est utilisé par le navigateur pour :

- Créer la **structure** de la page web
- Afficher et **animer** le contenu
- Gérer les **interactions** utilisateur

STRUCTURE DU DOM

Le **DOM** représente les éléments **HTML** sous forme d'objet, organisés en **arborescence** et **hiérarchie**, reflétant la structure du document.

ÉLÉMENTS HTML

Chaque élément HTML (balise, texte, attribut, etc.) est représenté par un **objet (noeud)** dans le **DOM**.

ARBORESCENCE ET HIÉRARCHIE

Le **DOM** suit une structure en **arbre**, où chaque objet est un **noeud**, possédant un ou plusieurs **noeuds enfants**, et dépendant d'un **noeud parent**.

MANIPULATION DU DOM

Grâce à **JavaScript**, il est possible de manipuler le **DOM** :

- Sélectionner des éléments : `document.querySelector()`, `document.getElementById()`
- Modifier des éléments : `element.innerHTML`, `element.style`
- Ajouter/supprimer des éléments : `element.appendChild()`, `element.removeChild()`
- Traverser le DOM (naviguer entre les noeuds) : `element.nextSibling`, `element.parentNode`

SÉLECTION DES ÉLÉMENTS

On peut sélectionner des **éléments du DOM** avec des méthodes telles que :

- getElementById
- getElementsByTagName
- querySelector

MODIFICATION DES ATTRIBUTS ET DU CONTENU

Il est possible de modifier les **attributs** et le **contenu** des éléments du DOM avec des méthodes comme
setAttribute, innerHTML, appendChild...

```
// Modifier un attribut d'un élément
element.setAttribute('attribute', 'value');

// Modifier le contenu HTML d'un élément
element.innerHTML = '<h1>Hello World</h1>';

// Ajouter un élément enfant
var newElement = document.createElement('p');
newElement.innerHTML = 'Un nouveau paragraphe';
element.appendChild(newElement);
```

INTERACTION AVEC LE DOM

LECTURE ET MODIFICATION DU CONTENU

innerHTML

Permet de **lire** ou **modifier** le contenu **HTML** d'un élément.

```
let element = document.getElementById("monElement");
element.innerHTML = "<span>Nouveau contenu</span>";
```

textContent

Permet de lire ou modifier le contenu en **texte brut** d'un élément.

```
let element = document.getElementById("monElement");
element.textContent = "Nouveau contenu";
```

nodeValue

Permet de lire ou modifier la valeur **textuelle** d'un nœud **texte**.

```
let texteNode = document.createTextNode("Texte");
texteNode.nodeValue = "Nouveau texte";
```

SÉLECTION D'ÉLÉMENTS

getElementsBy

Permet de sélectionner un **élément** par son **identifiant**.

```
let element = document.getElementById("monElement");
```

getElementsByClassName

Permet de **sélectionner des éléments** par leur **classe**.

```
let elements = document.getElementsByClassName("maClasse");
```

getElementsByName

Permet de sélectionner des **éléments** par leur **nom de balise**.

```
let elements = document.getElementsByName ("p");
```

querySelector

Permet de **sélectionner un élément** en utilisant un **sélecteur CSS**.

```
let element = document.querySelector("#monElement");
```

querySelectorAll

Permet de sélectionner tous les **éléments** correspondant à un **sélecteur CSS**.

```
let elements = document.querySelectorAll(".maClasse");
```

MANIPULATION DES ATTRIBUTS

setAttribute

Permet de définir la valeur d'un **attribut** pour un élément.

```
element.setAttribute("monAttribut", "maValeur");
```

getAttribute

Permet de récupérer la valeur d'un **attribut** d'un élément.

```
let valeur = element.getAttribute("monAttribut");
```

removeAttribute

Permet de **supprimer un attribut** d'un élément.

```
element.removeAttribute("monAttribut");
```

CRÉATION ET SUPPRESSION D'ÉLÉMENTS

createElement

La fonction `createElement` permet de créer un nouvel élément **HTML** à partir de la chaîne spécifiant le nom de l'élément.

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

SYNTAXE

```
let element = document.createElement(tagName);
```

EXEMPLES D'UTILISATION

```
let div = document.createElement('div');
let p = document.createElement('p');
```

appendChild

La méthode `appendChild` permet d'ajouter un élément **HTML** en tant qu'enfant du **nœud** spécifié.

```
var parent = document.getElementById("parent");
var enfant = document.createElement("div");

parent.appendChild(enfant);
```

SYNTAXE

```
parentNode.appendChild(childNode);
```

EXEMPLES D'UTILISATION

```
let ul = document.createElement('ul');
let li = document.createElement('li');
ul.appendChild(li);
```

insertBefore

La méthode `insertBefore` permet d'insérer un **nœud** avant un autre nœud, en tant qu'enfant du nœud spécifié.

```
parentElement.insertBefore(newElement, referenceElement);
```

Paramètre	Description
newElement	Le nouvel élément à insérer
referenceElement	L'élément avant lequel le nouvel élément doit être inséré

SYNTAXE

```
parentNode.insertBefore(newNode, referenceNode);
```

EXEMPLES D'UTILISATION

```
let ul = document.querySelector('ul');
let li = document.createElement('li');
ul.insertBefore(li, ul.firstChild);
```

removeChild

La méthode **removeChild** permet de supprimer un nœud enfant d'un nœud parent spécifié.

```
parentElement.removeChild(childElement);
```

- parentElement: L'élément parent duquel on veut supprimer l'enfant.
- childElement: L'élément enfant à supprimer du parent.

SYNTAXE

```
parentNode.removeChild(childNode);
```

EXEMPLES D'UTILISATION

```
let ul = document.querySelector('ul');
let li = ul.querySelector('li');
ul.removeChild(li);
```

ÉVÉNEMENTS EN JAVASCRIPT

INTRODUCTION AUX ÉVÉNEMENTS

Les **événements** sont des actions ou occurrences qui se produisent dans le navigateur, généralement en réponse à l'interaction de l'utilisateur.

- Exemples d'événements :
 - Clic sur un bouton
 - Appui sur une touche du clavier
 - Chargement d'une page

DÉFINITION ET UTILITÉ

Les **événements** permettent à notre code **JavaScript** de répondre aux actions des utilisateurs, telles que les clics, les mouvements de souris, etc.

EXEMPLES D'ÉVÉNEMENTS COURANTS

- **click** : L'utilisateur clique sur un élément
- **mousemove** : La souris de l'utilisateur se déplace sur un élément
- **keydown** : L'utilisateur appuie sur une touche du clavier
- **submit** : Un formulaire est soumis

GESTION DES ÉVÉNEMENTS

ADDEVENTLISTENER

La méthode `addEventListener` permet d'attacher un **gestionnaire d'événements** à un élément.

```
element.addEventListener('type', function (event) {  
    // Gestionnaire de l'événement  
});
```

- **élément** : l'objet cible sur lequel l'événement doit être écouté
- **type** : une chaîne représentant le type d'événement à écouter (par exemple, 'click')
- **fonction** : la fonction de rappel qui sera exécutée lorsque l'événement sera déclenché

SYNTAXE

```
element.addEventListener(event, function, useCapture);
```

EXEMPLES D'UTILISATION

```
document.querySelector("button").addEventListener("click", function() {  
    alert("Le bouton a été cliqué !");  
});
```

REMOVEEVENTLISTENER

La méthode `removeEventListener` permet de retirer un **gestionnaire d'événements** attaché précédemment avec `addEventListener`.

```
function monGestionnaire() {  
    console.log("Événement détecté");  
}  
  
// Ajouter le gestionnaire d'événements  
element.addEventListener("click", monGestionnaire);  
  
// Retirer le gestionnaire d'événements  
element.removeEventListener("click", monGestionnaire);
```

SYNTAXE

```
element.removeEventListener(event, function, useCapture);
```

EXEMPLES D'UTILISATION

```
function clicAlert() {  
    alert("Le bouton a été cliqué !");  
}  
  
document.querySelector("button").addEventListener("click", clicAlert);  
document.querySelector("button").removeEventListener("click", clicAlert);
```

PRATIQUE D'UTILISATION DES GESTIONNAIRES D'ÉVÉNEMENTS

- Utilisez `addEventListener` pour ajouter des gestionnaires d'événements
- Essayez de gérer différents types d'événements
- Testez la combinaison de plusieurs gestionnaires d'événements sur un seul élément
- Expérimenez avec `removeEventListener` pour détacher les gestionnaires d'événements

PROPAGATION DES ÉVÉNEMENTS

PHASE DE CAPTURE

La phase de capture commence au **nœud racine** du document et arrive jusqu'à l'**élément cible** en parcourant l'**arborescence du DOM**.

```
element.addEventListener("click", function(event) {  
  console.log("Phase de capture");  
, true);
```

PHASE DE CIBLAGE

Étape où l'événement atteint l'élément **cible** (émetteur de l'événement) et où l'événement est déclenché.

```
element.addEventListener("click", function(event) {  
  console.log("Phase de ciblage");  
});
```

PHASE DE DIFFUSION

La phase de diffusion commence après la phase de **ciblage** et remonte jusqu'au **nœud racine**.

```
element.addEventListener("click", function(event) {  
  console.log("Phase de diffusion");  
, false);
```

GÉRER LA PROPAGATION DES ÉVÉNEMENTS

EVENT.STOPPROPAGATION

Arrête la **propagation** de l'événement dans les phases de **capture** et de **diffusion**.

```
element.addEventListener("click", function(event) {  
  console.log("Stop propagation");  
  event.stopPropagation();  
});
```

EVENT.STOPIMMEDIATEPROPAGATION

Arrête la **propagation** de l'événement pour les autres **gestionnaires d'événements** sur le même élément.

```
element.addEventListener("click", function(event) {  
  console.log("Stop immediate propagation");  
  event.stopImmediatePropagation();  
});
```

EVENT.PREVENTDEFAULT

Empêche le **comportement par défaut** de l'élément lors de l'événement.

```
element.addEventListener("click", function(event) {  
  console.log("Prevent default");  
  event.preventDefault();  
});
```

ÉVÉNEMENTS ET FORMULAIRES

GESTION DES ÉVÉNEMENTS DE FORMULAIRE

Les **événements de formulaire** permettent d'interagir avec les éléments d'un **formulaire HTML** pour réaliser des actions comme la **validation**, les **calculs dynamiques**, etc.

onsubmit

L'événement `onsubmit` est déclenché lors de la soumission d'un **formulaire**. Il peut être utilisé pour **valider** les données avant soumission.

```
document.querySelector('form').addEventListener('submit', function(event) {
  event.preventDefault(); // Annule la soumission du formulaire

  // Validation des données et soumission manuelle si valides
  if (isValidData()) {
    this.submit();
  }
});
```

SYNTAXE

```
<form onsubmit="return maFonction()">
```

EXEMPLES D'UTILISATION

```
function maFonction() {  
    if (document.getElementById("inputNom").value === "") {  
        alert("Nom requis");  
        return false;  
    }  
    return true;  
}
```

onchange

L'événement **onchange** est déclenché lorsqu'un élément de **formulaire** change de valeur.

SYNTAXE

```
<input type="text" id="inputNom" onchange="maFonction()">
```

EXEMPLES D'UTILISATION

```
function maFonction() {  
    var inputNom = document.getElementById("inputNom").value;  
    document.getElementById("affichageNom").innerHTML = inputNom;  
}
```

oninput

L'événement `oninput` est déclenché lorsqu'un utilisateur modifie la valeur d'un **élément de formulaire**.

Exemple :

```
<input type="text" id="myInput" oninput="myFunction () ">

<script>
function myFunction () {
  console.log("La valeur du champ de texte a été modifiée.");
}
</script>
```

SYNTAXE

```
<input type="text" id="inputPrix" oninput="maFonction()">
```

EXEMPLES D'UTILISATION

```
function maFonction() {  
    var inputPrix = document.getElementById("inputPrix").value;  
    var taxe = inputPrix * 0.15;  
    document.getElementById("affichageTaxe").innerHTML = taxe.toFixed(2);  
}
```

