JavaScript pour le Web : DOM et Événements

Interactivité et Manipulation du Navigateur

Pour les étudiants de 1ère année Supinfo

Au Programme

1. Contexte - Rappel de JavaScript pour le web interactif
2. DOM - Manipuler la structure et le contenu de la page
3. Événements - Réagir aux actions de l'utilisateur

Contexte : Rappel et Utilité

RAPPEL : POURQUOI JAVASCRIPT ?

JavaScript a été **conçu pour rendre le web plus interactif**. Il devait être **exécuté côté client** (dans le navigateur) et être **facile à apprendre**. Aujourd'hui, c'est le langage de programmation incontournable du web.

UTILITÉ DE JAVASCRIPT

- **Stocker des valeurs** dans des variables et les manipuler
- **Faire des opérations** sur les données (strings, nombres, etc.)
- **Exécuter du code** en réponse à des événements (clics, saisies)
- **Modifier la structure** de la page avec le DOM
- **Créer des animations** et des interactions avancées

API NAVIGATEURS

Le navigateur propose de nombreuses **API (Application Programming Interfaces)** pour faire des choses complexes : DOM (manipuler HTML/CSS), géolocalisation, canvas/WebGL (animations 2D/3D), Audio/Video, et bien d'autres.

API Navigateurs : Le DOM

QU'EST-CE QUE LE DOM ?

Le **Document Object Model (DOM)** est une **API qui représente la page web** de sorte à ce que les programmes JavaScript puissent la comprendre et la modifier. C'est l'interface principale pour interagir avec le contenu HTML et CSS d'une page.

UTILITÉ DU DOM

- **Accéder** aux éléments HTML de la page
- **Modifier** le contenu, le style et les attributs
- **Créer et supprimer** des éléments dynamiquement
- **Réagir** aux événements utilisateur

AUTRES API NAVIGATEURS

- **Géolocalisation** - Accéder à la position de l'utilisateur
- **Canvas et WebGL** - Créer des animations 2D/3D
- **Audio et Video** - Contrôler les médias
- **Fetch API** - Faire des requêtes réseau

DOM : L'Arbre de Nœuds

QU'EST-CE QU'UN NŒUD ?

Le DOM représente la page HTML comme un **arbre de nœuds**. Chaque **nœud** représente une partie du document : un élément HTML, du texte, un attribut, ou le document lui-même. Les nœuds sont organisés en une **hiérarchie parent-enfant**.

VISUALISATION DE L'ARBRE DOM

```
document
├── html
│   ├── head
│   │   ├── title
│   │   └── meta
│   └── body
│       ├── div (id="container")
│       │   ├── h1 (Titre)
│       │   └── p (Paragraphe)
│       ├── button (Bouton)
│       └── script
```

CONCEPTS CLÉS

- **Nœud parent** : L'élément qui contient d'autres éléments (ex: body contient div)
- **Nœuds enfants** : Les éléments contenus dans un parent (ex: h1 et p sont enfants de div)

DOM : Sélectionner des Éléments

CONCEPT GÉNÉRAL

Pour manipuler un élément HTML avec JavaScript, il faut d'abord le **sélectionner**. On récupère l'élément parent (généralement **document** pour la page entière) et on utilise des **sélecteurs CSS** (`.` pour les classes, `#` pour les IDs) pour trouver le(s) élément(s) voulu(s).

MÉTHODES PRINCIPALES

querySelector (un seul)

```
// Sélectionne le premier élément
const btn = document
.querySelector(".btn");

const title = document
.querySelector("#title");
```

querySelectorAll (tous)

```
// Sélectionne tous les éléments
const btns = document
.querySelectorAll(".btn");

// Boucler sur les résultats
btns.forEach(btn => {
// faire quelque chose
});
```

AUTRES MÉTHODES

- **getElementById** - Sélectionner par ID (plus ancien, moins flexible)
- **getElementsByClassName** - Sélectionner par classe (retourne une collection)
- **Conseil** - Préférez querySelector et querySelectorAll pour plus de flexibilité

DOM : Modifier le Contenu

textContent

Modifie le **contenu texte uniquement** de l'élément. Les balises HTML ne sont pas interprétées.

- Texte brut uniquement
- Les balises HTML sont affichées comme du texte
- Plus sûr (pas de risque XSS)
- Plus performant

```
const elem = document.querySelector('p');
elem.textContent = 'Bonjour <b>monde</b>';

// Affiche:
// Bonjour <b>monde</b>
```

innerHTML

Modifie le **contenu HTML** de l'élément. Les balises HTML sont interprétées et exécutées.

- HTML et texte
- Les balises HTML sont interprétées
- Risque de sécurité (XSS) si contenu non validé
- Plus flexible pour le contenu complexe

```
const elem = document.querySelector('p');
elem.innerHTML = 'Bonjour <b>monde</b>';

// Affiche:
// Bonjour monde
```

💡 **Conseil**

Utilisez **textContent** par défaut pour des raisons de sécurité et de performance. Utilisez **innerHTML** uniquement quand vous avez besoin de contenu HTML et que vous êtes sûr que le contenu provient d'une source fiable.

DOM : Modifier Style et Attributs

Propriété style

Modifier directement les styles CSS d'un élément via la propriété **style**. Les noms des propriétés CSS sont en camelCase.

```
const el = document
.querySelector('#box');

el.style.display = 'none';
el.style.backgroundColor
= 'red';
el.style.fontSize = '20px';
```

classList

Gérer les classes CSS d'un élément via le tableau **classList**. Permet d'ajouter, supprimer ou basculer des classes.

```
const el = document
.querySelector('#box');

el.classList.add('active');
el.classList.remove('hidden');
el.classList.toggle('selected');
el.classList.contains('active');
```

setAttribute

Modifier les attributs HTML d'un élément avec **setAttribute**. Utile pour les attributs data, href, src, etc.

```
const el = document
.querySelector('a');

el.setAttribute('href',
'https://example.com');
el.setAttribute('data-id',
'123');
```

💡 Quand utiliser quoi ?

style pour les styles simples et ponctuels • **classList** pour gérer les états (actif, caché, sélectionné) avec des classes CSS • **setAttribute** pour modifier les attributs HTML (href, src, data-*)

DOM : Créer et Ajouter des Éléments

CRÉER UN ÉLÉMENT

Pour créer un nouvel élément HTML, utilisez **document.createElement()** en spécifiant le type d'élément (ex: "div", "p", "button"). L'élément est créé mais pas encore ajouté à la page.

```
const newDiv = document.createElement("div");
newDiv.textContent = "Contenu du div";
newDiv.classList.add("ma-classe");
```

AJOUTER UN ÉLÉMENT À LA PAGE

Pour ajouter un élément créé à la page, utilisez **parentElement.appendChild(childElement)**. L'élément enfant est ajouté à la fin de la liste des enfants du parent.

```
const container = document.getElementById("container");
container.appendChild(newDiv);
// newDiv est maintenant visible sur la page
```

SUPPRIMER UN ÉLÉMENT

Pour supprimer un élément de la page, utilisez **parentElement.removeChild(childElement)**. L'élément est retiré du DOM.

```
container.removeChild(newDiv);
// newDiv est supprimé de la page
```

RÉSUMÉ DES 3 OPÉRATIONS

Créer

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

Ajouter

```
parent.appendChild(el);
```

Supprimer

```
parent.removeChild(el);
```

Événements : Définition et Types

QU'EST-CE QU'UN ÉVÉNEMENT ?

Un **événement** est une **action ou une occurrence** qui se produit dans le navigateur. Les événements permettent de **réagir aux actions de l'utilisateur** comme les clics, les saisies au clavier, les mouvements de souris, ou les soumissions de formulaires. JavaScript peut "écouter" ces événements et exécuter du code en réponse.

TYPES D'ÉVÉNEMENTS

Événements de Souris

- **click** - Clic sur un élément
- **mouseover** - Survol d'un élément
- **mouseout** - Sortie de la souris
- **mousemove** - Mouvement de la souris

Événements de Clavier

- **keydown** - Touche enfoncée
- **keyup** - Touche relâchée
- **keypress** - Touche pressée (caractère)

Événements de Formulaire

- **submit** - Envoi du formulaire
- **change** - Changement de valeur
- **focus** - Élément en focus
- **blur** - Élément perd le focus

Événements de Fenêtre

- **load** - Page chargée
- **resize** - Redimensionnement
- **scroll** - Défilement
- **unload** - Fermeture de la page

Événements : addEventListener

SYNTAXE DE ADDEVENTLISTENER

La méthode **addEventListener** est le moyen standard d'attacher un gestionnaire d'événements à un élément. Elle prend deux paramètres : le **type d'événement** (ex: "click", "keydown") et la **fonction callback** à exécuter.

```
element.addEventListener(typeEvenement, fonction);

// Exemple
const btn = document.querySelector("button");
btn.addEventListener("click", function() {
console.log("Bouton cliqué!");
});
```

SUPPRIMER UN GESTIONNAIRE

Pour retirer un écouteur, utilisez **removeEventListener** avec la même signature que lors de l'ajout. C'est pourquoi il est important d'utiliser une fonction nommée.

```
// Supprimer le gestionnaire
btn.removeEventListener("click", handleClick);
```

MAUVAISE VS BONNE PRATIQUE

❌ Mauvaise Pratique

```
// Fonction anonyme
btn.addEventListener("click",
function() {
console.log("Cliqué");
}
);

// Impossible à supprimer
// plus tard!
```

✅ Bonne Pratique

```
// Fonction nommée
function handleClick() {
console.log("Cliqué");
}

btn.addEventListener("click",
handleClick
);

// Peut être supprimée!
```

💡 Conseil : utilisez une fonction nommée pour pouvoir la supprimer plus tard avec removeEventListener.

Événements : preventDefault

DÉFINITION

La méthode **preventDefault()** permet d'**empêcher le comportement par défaut** d'un événement. Par exemple, empêcher un formulaire de se soumettre et recharger la page, ou empêcher un lien de naviguer vers une nouvelle page.

SYNTAXE

```
element.addEventListener('event', (event) => {
event.preventDefault();
// Votre code personnalisé
});
```

CAS D'USAGE COURANTS

- **Formulaires** - Empêcher le rechargement de page lors de la soumission (submit)
- **Liens** - Empêcher la navigation vers une nouvelle page (click)
- **Clavier** - Empêcher les raccourcis clavier par défaut (keydown)
- **Glisser-déposer** - Empêcher le comportement par défaut du drag-drop

EXEMPLE : FORMULAIRE

```
const form = document.querySelector('form');
form.addEventListener('submit', (event) => {
event.preventDefault(); // Empêche le rechargement
const nom = document.querySelector('input[name="nom"]').value;
console.log('Nom saisi:', nom);
// Traiter les données avec JS
});
```

Événements : Traitement de Formulaire

CONCEPT

Un formulaire HTML émet un événement **submit** lors de son envoi. Par défaut, cet événement recharge la page. Pour traiter les données avec JavaScript, il faut **empêcher ce rechargement** avec **preventDefault()**, puis récupérer et traiter les valeurs.

ÉTAPES CLÉS

1. **Sélectionner** le formulaire avec querySelector
2. **Écouter** l'événement submit avec addEventListener
3. **Empêcher** le rechargement avec preventDefault()
4. **Récupérer** les valeurs avec la propriété value
5. **Traiter** les données avec JavaScript

EXEMPLE COMPLET

HTML

```
<form id="myForm">
<input type="text"
name="nom"
required>
<input type="email"
name="email"
required>
<button type="submit">
Envoyer
</button>
</form>
```

JavaScript

```
const form = document
.querySelector('#myForm');

form.addEventListener
('submit', (event) => {
event.preventDefault();
const nom = document
.querySelector
('input[name="nom"]')
.value;
const email = document
.querySelector
('input[name="email"]')
.value;
console.log(nom, email);
});
```

Conclusion

5 CONCEPTS CLÉS À RETENIR

1

DOM : Représentation de la page

2

Sélectionner : querySelector

3

Modifier : textContent, innerHTML

4

Créer : createElement, appendChild

5

Événements : addEventListener

RESSOURCES POUR APPROFONDIR

MDN Web Docs - Documentation complète sur le DOM et les événements

JavaScript.info - Tutoriels interactifs sur le DOM et les événements

Eloquent JavaScript - Livre en ligne gratuit sur JavaScript avancé

Web APIs - Référence complète des API du navigateur

Continuez à pratiquer et à explorer le DOM et les événements. C'est la base de toute interactivité web moderne !