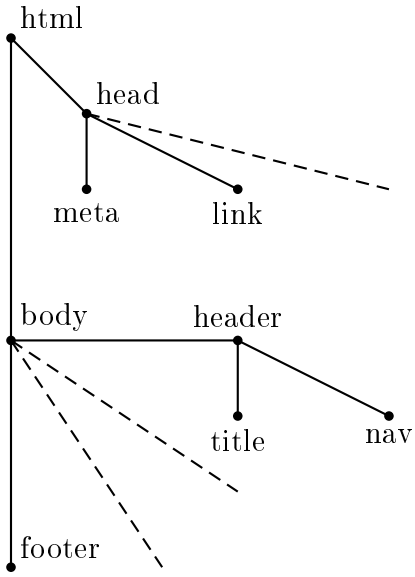
TP n°4 : Introduction à JavaScript pour le Web

1 Document Object Model

Le **Document Object Model (DOM)** est une représentation d'une page web qui permet d'accéder aux éléments de cette page et d'interagir avec eux grâce au langage JavaScript. On peut notamment

- modifier le contenu d'un élément ;
- modifier le style d'un élément ;
- créer ou supprimer un élément ;
- etc.

Le DOM peut être vu comme un arbre où chaque élément peut avoir plusieurs enfants et ainsi de suite. Chaque élément du DOM est un objet JavaScript avec ses propriétés et ses fonctions afin d'interagir avec lui.



2 Accéder aux éléments du DOM

2.1 Recherche à partir du document

Le `document` représente l'objet page Web en JavaScript. C'est donc le point de départ du DOM et c'est lui qui contient les fonctions nécessaires à la recherche de ses éléments.

Nous allons voir quatre méthodes de recherche d'éléments du DOM.

Recherche	Commande
par id	`document.getElementById()`
par classe	`document.getElementsByClassName()`
par balise	`document.getElementsByTagName()`
avancée	`document.querySelector()`

2.1.1 `document.getElementById()`

`document.getElementById()` permet de rechercher un objet par son attribut identifiant. Le résultat est donc un objet unique puisque les identifiants sont uniques.

Exemple : Si on a l'élément HTML

```
<h1 id="titre-principal">Titre principal</h1>
```

On va pouvoir le récupérer grâce à

```
let titrePrincipal=document.getElementById("titre-principal")
```

On remarquera que l'on a crée une variable pour stocker le résultat de la recherche.

Exercice 1.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Repérer dans le fichier HTML l'identifiant du lien et pourquoi ce dernier ne fonctionne pas.
3. Modifier le fichier JavaScript afin d'y déclarer une variable `lien` contenant le résultat de la recherche par identifiant sur notre lien HTML.

2.1.2 `document.getElementsByClassName()`

`document.getElementsByClassName()` permet de rechercher un objet par son attribut de classe. Comme plusieurs éléments peuvent avoir la même classe, le résultat est donc une liste d'objets.

Exemple : Si on a le code HTML

```
<p class="paragrapheA">...</p>
<p class="paragrapheB">...</p>
<p class="paragrapheA">...</p>
<p class="paragrapheB">...</p>
```

On va pouvoir récupérer les paragraphes A grâce à

```
let paragraphesA=document.getElementsByClassName("paragrapheA")
```

Exercice 2.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Repérer dans le fichier HTML les différentes classes de section.
3. Modifier le fichier JavaScript afin d'y déclarer des variables `sectionsA` et `sectionsB` contenant les résultats des recherche par classes sur nos sections HTML.

2.1.3 `document.getElementsByTagName()`

`document.getElementsByTagName()` permet de rechercher un objet par sa balise. Le résultat est donc ici aussi une liste d'objets.

Exemple : L'instruction suivante permet de récupérer la liste de tous les paragraphes.

```
let paragraphes=document.getElementsByTagName("p")
```

2.1.4 `document.querySelector()`

`document.querySelector()` permet d'effectuer des recherches avancées et peut donc s'avérer complexe à utiliser.

Exemples : On considère le code HTML suivant :

```
<section>
    <a href="#">Lien 1</a>
</section>
<section id="identifiant">
    <p>
        <a href="#">Lien 2</a>
    </p>
    <p class="paragrapheA">
        <div><a href="#">Lien 3</a></div>
        <a href="#">Lien 4</a>
    </p>
    <a href="#">Lien 5</a>
</section>
```

- L'instruction `document.querySelector("#identifiant > a")` permet d'obtenir le premier lien qui est un enfant direct de l'objet ayant pour identifiant `identifiant`, ici le lien 5.
- L'instruction `document.querySelector("#identifiant p.paragrapheA > a")` permet d'obtenir le premier lien qui est un enfant direct du paragraphe de classe `paragrapheA` de l'objet ayant pour identifiant `identifiant`, ici le lien 4.

Remarque : `document.querySelector()` renvoie le premier élément à satisfaire la recherche. Si l'on veut tous les éléments la satisfaisant, on peut utiliser `document.querySelectorAll()`.

Exercice 3. On reprend l'exemple précédent.

1. Quel lien est le résultat de la recherche `document.querySelector("#identifiant p > a")` ?
2. Quelle recherche permet d'obtenir le lien 1 ?

2.2 Recherche à partir d'un élément

Comme chaque élément HTML est un objet JavaScript, il est possible de rechercher des éléments à partir de l'un d'entre eux. En voici quatre méthodes de recherche.

Recherche	Commande
des enfants	`element.children`
du parent	`element.parentElement`
de l'élément de même niveau précédent	`element.previousElementSibling`
de l'élément de même niveau suivant	`element.nextElementSibling`

Exemple : On considère le code HTML suivant :

```
<section id="section1">
     <p>Paragraphe 1</p>
     <p>Paragraphe 2</p>
</section>
<section id="section2">
     <p>Paragraphe 3</p>
     <p>Paragraphe 4</p>
</section>
<section id="section3">
     <p>Paragraphe 5</p>
     <p>Paragraphe 6</p>
</section>
```

La commande `let s2=document.getElementById("section2")` permettra d'obtenir la section ayant cet identifiant. `s2.children` renverra les paragraphes trois et quatre ; `s2.previousElementSibling` renverra la section une et `s2.nextElementSibling` la troisième.

Exercice 4.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Repérer dans le fichier HTML l'identifiant du paragraphe de départ.
3. Modifier le fichier JavaScript afin d'y déclarer les variables trouvées à partir des recherches :
   (a) du paragraphe susmentionné ;
   (b) de l'élément parent de ce paragraphe : `elementParent` ;
   (c) des éléments précédent et suivant cet élément parent : `elementParentPrecedent` et `elementParentSuivant`.

3 Modifier le DOM

3.1 Modifier le contenu d'un élément

Dans les deux cas suivants, il est à noter que la modification écrase le contenu précédent.

3.1.1 Contenu interprété en HTML

Il est possible d'intégrer du contenu qui sera interprété comme du HTML grâce à la commande `innerHTML`. Par exemple, si on entre en JavaScript :

```
element.innerHTML="<em>exemple</em>";
```

nous obtiendrons à l'affichage de la page HTML :

exemple.

3.1.2 Contenu non interprété en HTML

Inversement, il est possible d'intégrer du contenu qui ne sera pas interprété comme du HTML grâce à la commande `textContent`. Par exemple, si on entre en JavaScript :

```
element.textContent="<em>exemple</em>";
```

nous obtiendrons à l'affichage de la page HTML :

exemple.

En effet, cette fois-ci, les balises ne seront pas interprétées comme du HTML et seront affichées comme du texte normal.

Exercice 5.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Repérer dans le fichier HTML les identifiants des sections qui contiendront du HTML interprété ou non.
3. Écrire un programme JavaScript afin de modifier le contenu de nos deux sections. Dans les deux cas, on devra remplacer le contenu existant par le suivant :

   ```
   Contenu <strong>modifié</strong> à l'aide de JavaScript.
   ```

   On fera correspondre l'interprétation de ce nouveau contenu à la section modifiée.

3.2 Modifier les attributs d'un élément

Il est possible de modifier les attributs d'un élément principalement grâce aux commandes suivantes :

Action	Commande
obtention	`element.getAttribute("attribut")`
modification	`element.setAttribute("attribut","valeur")`
suppression	`element.removeAttribute("attribut")`

Exemple : si l'on considère le champ HTML suivant :

```
<input type="password" name="motDePasse" id="motDePasse" required/>
```

Alors,

```
let motDePasse=document.getElementById("motDePasse");
motDePasse.getAttribute("type");
```

permet d'obtenir le type de notre champ. Si on ajoute maintenant

```
motDePasse.setAttribute("type","text");
motDePasse.removeAttribute("required");
```

notre champ n'est alors plus obligatoire et n'est plus celui d'un mot de passe (il sera donc affiché en clair à la complétion).

Exercice 6.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire un programme JavaScript afin que tous les liens de la page Web pointent vers l'adresse :

   https ://auvraymath.net/

Exercice 7.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Écrire un programme JavaScript modifiant tous les champs de mot de passe en champs de texte sans utiliser les identifiants.
3. Modifier le programme précédent afin de faire en sorte qu'aucun champ ne soit obligatoire.
4. Inverser les textes affichés dans nos deux boutons.

3.3 Créer, ajouter, remplacer, supprimer des éléments

Il est possible de créer, ajouter, remplacer, supprimer des éléments grâce aux commandes suivantes :

Action	Commande
création	`document.createElement("element")`
ajout	`elementParent.appendChild(elementEnfant)`
remplacement	`elementParent.replaceChild(ancienEnfant,nouvelEnfant)`
suppression	`elementParent.removeChild(elementEnfant)`

Remarques :

- La création d'un élément n'inclue pas son placement au sein du DOM. Pour qu'il soit intégré à celui-ci, il faut l'ajouter à un élément parent.
- L'ajout, le remplacement et la suppression sont des actions relatives à un élément parent. Elles tiennent donc nécessairement compte de la structure du DOM.

Exercice 8.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Créer à l'aide de JavaScript une section contenant dix paragraphes contenant chacun : « Paragraphe numéroParagraphe ».
3. Même question en créant cette fois-ci dix sections contenant dix paragraphes contenant chacun : « Paragraphe numéroSection.numéroParagraphe ».

4 Écoute d'événements

4.1 Événements

Un événement est une réaction à une action de l'utilisateur tel que le passage ou clic de la souris sur un élément ou la saisie d'un texte dans un champ. En JavaScript, il existe de nombreux types d'événements (dont on trouvera une liste ici) ; nous nous intéressons principalement au clic de la souris (`click`), à ses mouvements (`mousemove`, `mouseover/mouseout`, etc) et à l'entrée d'un contenu dans un formulaire (`input`).

L'événement déclenche l'appel d'une fonction à chaque fois que celui-ci se produit. La fonction JavaScript permettant ceci est

```
addEventListener('événement',fonction)
```

Exemples : comme l'événement se déclenche lors de l'interaction de l'utilisateur avec un élément (ici appelé `element`, ce qui nécessite donc de l'avoir d'abord récupéré grâce aux commandes vues plus haut), nous aurons les commandes suivantes :

- `element.addEventListener('click',fonction)` ;
- `element.addEventListener('mousemove',fonction)` ;
- `element.addEventListener('input',fonction)`.

4.2 Comportement par défaut et propagation

4.2.1 Comportement par défaut

Les éléments ont des comportements par défaut que l'on peut souhaiter annuler dans certaines situations. Cela peut par exemple être un lien dont on ne veut qu'il nous envoie vers une nouvelle page, un bouton dont il faudrait annuler la fonction, etc. Pour cela, nous avons la fonction `preventDefault()` qui empêche précisément le comportement par défaut de l'élément.

Exemple : on considère une page HTML contenant un lien ayant pour identifiant `lien` et le code JavaScript suivant :

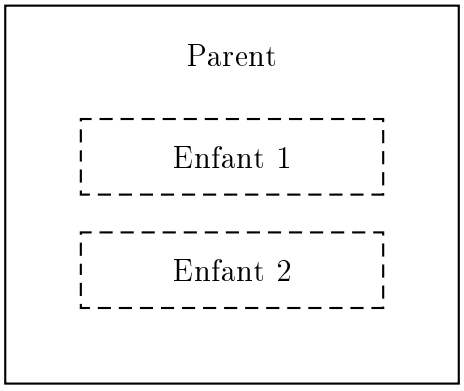
```
// récupération de l'élément sur lequel on va détecter le clic
const lien=document.getElementById("lien");

// définition de la fonction qui sera appelée lors du clic,
// evenement en est un paramètre
function clicReaction(evenement) {
    evenement.preventDefault();
}

lien.addEventListener("click",clicReaction);
```

La dernière ligne nous indique que lorsqu'on clique sur le lien, la fonction `clicReaction` est appelée. Celle-ci va alors empêcher le comportement par défaut du lien et aucune page ne sera chargée.

4.2.2 Propagation

Par défaut, un événement sur un élément se propage vers son élément parent. Par exemple, si l'on clique sur l'une des zones enfants ci-contre, comme celles-ci sont incluses dans la zone parent, on aura aussi cliqué sur la zone parent. Cela se traduit par une propagation de l'événement `click` de l'élément enfant à l'élément parent. Cela n'est pas forcément souhaitable et cela peut être empêché grâce à la fonction

`stopPropagation()`.



En reprenant l'exemple précédent, on aurait `evenement.stopPropagation();` à la place ou en ajout de `evenement.preventDefault();`

Exercice 9.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Ouvrir le fichier HTML dans le navigateur. Le but de cet exercice est de faire en sorte que le compteur parent, resp. enfant, augmente de un à chaque fois que l'utilisateur clique dans la zone délimitée par le rectangle, resp. sur le lien.
   (a) Écouter les événements `click` depuis l'élément parent puis affichez le nombre de clics dans le compteur associé.
   (b) Faire la même chose avec l'élément enfant. Que constatez-vous ? Comment remédier à cela ?
   (c) Pourquoi lorsqu'on clique sur le lien, aucune page ne se charge ? Quel autre moyen aurait-on pour empêcher le chargement d'une nouvelle page ?

5 Récupération de données

Lors du déclenchement d'un événement, il peut être nécessaire de récupérer des données entrées par l'utilisateur comme le contenu d'un champ de formulaire par exemple ou la position de la souris. Nous allons nous intéresser à deux cas particuliers issus des ces exemples.

5.1 Lire le contenu d'un champ

Il est possible de lire le contenu d'un champ au moment où l'utilisateur le remplit ou lorsqu'il clique sur un bouton de soumission par exemple. Pour cela on utilise l'instruction

```
evenement.target.value
```

Celle-ci permet de cibler (`target`) l'élément sur lequel a lieu l'événement et récupère sa valeur (`value`). Si on a déjà identifié et chargé un élément dont on voudrait récupérer la valeur, on peut aussi faire directement

```
element.value
```

Exemple : considérons que l'on ait un formulaire contenant un champ d'identifiant `champ` et un bouton d'identifiant `bouton`. Le programme JavaScript suivant permet de récupérer la valeur du champ lorsqu'on clique sur le bouton.

```
const champ=getElementById("champ");
const bouton=getElementById("bouton");

function recuperation() {
     let valeurChamp=champ.value;
     ...
}

bouton.addEventListener("click",recuperation);
```

Si on veut récupérer la valeur du champ au moment de sa saisie, on utilisera en lieu et place de la dernière ligne de l'exemple ci-dessus :

```
champ.addEventListener("input",recuperation);
```

Dans ce dernier cas, un autre code possible serait

```
const champ=document.getElementById("champ");

function recuperation(evenement) {
     let valeurChamp=evenement.target.value;
     ...
}

champ.addEventListener("input",recuperation);
```

Exercice 10.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Ouvrir le fichier HTML dans le navigateur. Le but de cet exercice est de vérifier que le mot de passe entré dans le champ de confirmation est bien identique à celui du dessus.
   (a) Écouter les événements `input` depuis le champ de confirmation afin de récupérer sa valeur et celle du mot de passe au dessus.
   (b) Faire en sorte que tant que les deux mots de passe sont différents, le message « mots de passe différents » soit affiché dans l'élément d'identifiant message. Le message s'effacera dès que les mots de passe seront identiques.
   (c) Toujours tant que les deux mots de passe sont différents, faire en sorte que le bouton de création du compte soit désactivé (on utilisera l'attribut `disabled`). Dès qu'ils seront identiques, il devra redevenir actif.

5.2 Lire les données de la souris

Il est possible d'obtenir des données à partir de la souris comme ses coordonnées dans la page, les clics, sa présence sur élément, etc. Nous allons nous intéresser ici à l'écoute de la présence de celle-ci sur un élément grâce à l'événement `mouseover`. Pour cela, il suffit d'utiliser l'instruction

```
element.addEventListener("mouseover",fonction);
```

Exercice 11.

1. Télécharger les fichiers correspondants à cet exercice présents dans ce dossier.
2. Ouvrir le fichier HTML dans le navigateur. Le but de cet exercice est de compter le nombre de fois où la souris passe sur l'image de Tux le manchot de Linux. Écrire un programme JavaScript incrémentant le compteur sous Tux à chaque que la souris passe sur celui-ci.

6 Projet Web : partie 4

Il s'agit d'ajouter des scripts à votre site. La page de formulaire doit faire appel à au moins un script effectuant les actions suivantes :

- si au moins un des champs requis n'est pas rempli, le message d'alerte « Tous les champs requis n'ont pas été complétés » doit s'afficher avant la soumission du formulaire et le bouton de soumission est désactivé ;
- des champs de confirmation du mail et du mot de passe doit être ajoutés ; si les contenus ne sont pas identiques, un message d'alerte doit s'afficher en précisant quel champ à un problème et le bouton de soumission est désactivé ;
- si tout le formulaire est valide, un message de validation et de confirmation doit apparaître en dessous du formulaire à sa soumission.

Il est possible d'aller plus loin de faire appel à d'autres scripts sur le site. On pourra penser à des fonctionnalités telles que :

- un compteur de passages ou de clics de la souris un média ;
- un mode sombre / clair pour l'apparence du site sélectionnable à l'aide de boutons ;
- etc.