Intégration Web

Programmation événementielle en Javascript

Pré-requis

Être capable d'insérer du code javascript :

- dans une balise script
- dans un fichier JS

Savoir utiliser :

- les variables
- les structures de contrôle
- parcourir des tableaux

Avoir des notions d'anglais technique pour lire de la documentation.

Les événements par l'exemple

Exemples simples

Click sur un menu

https://jsfiddle.net/ychdvf6q/1/

> Décrire le comportement du programme

Réponse :

- Cible : élément de menu
- Événement : click

- Action : affichage du sous-menu

Click sur une image

https://jsfiddle.net/ec9xoagt/

Exercice 1 sur Moodle : décrire le comportement du programme

Réponse :

- Cible : l'image
- Événement : click
- Action : affichage d'un message

Click sur un bouton

https://jsfiddle.net/x1rnkvd5/

Exercice 2 sur Moodle : décrire le comportement du programme

Réponse :

- Cible : bouton
- Événement : click
- Action : incrémentation de valeurs

La notion d'événements

Dans les exemples précédents, lorsque l'utilisateur a cliqué sur le menu, le sous-menu s'est affiché. Le navigateur a réagi à l'événement « click » : il a modifié le code HTML de la page :

- Modification du texte
- Ajout de classes
- Affichage d'un message

- etc

Code source et inspecteur

Pour travailler efficacement, il faut prendre l'habitude d'utiliser les outils de développement. En premier lieu : « l'inspecteur ». L'inspecteur renseigne sur le code html et le code css. Il indique également les événements qui sont écoutés par le navigateur.

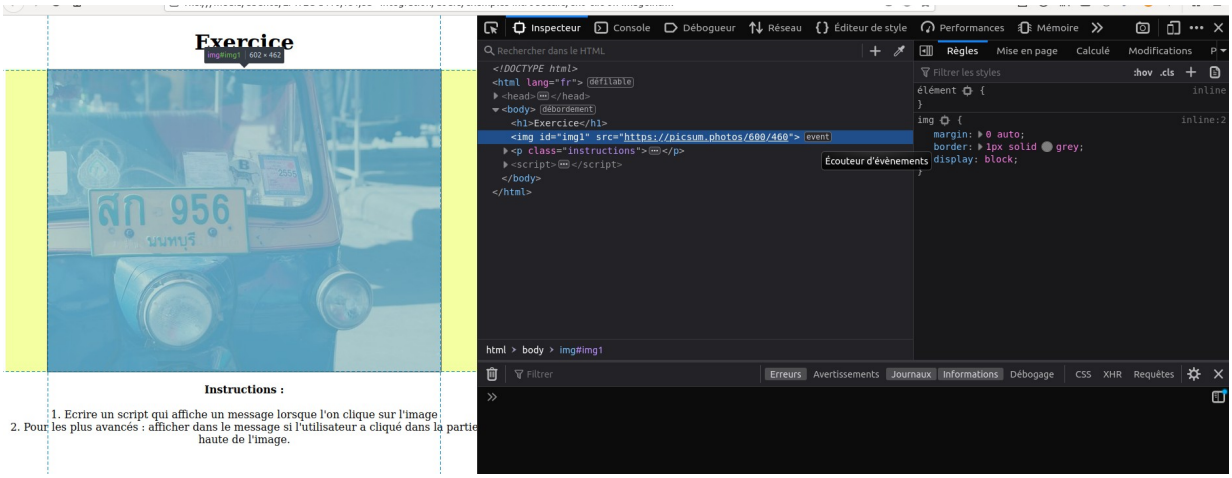


Figure 1 : l'outil de développement « inspecteur »

Le DOM

Le langage Javascript est un langage capable de modifier le code HTML de la page, en temps réel, lorsque certains événements se produisent. Pour cela, Javascript manipule le DOM (*Document Object Model*). Le DOM est une représentation de la structure de la page HTML.

le DOM

Un peu de vocabulaire ...

Une page html a une structure arborescente. On trouve notamment les balises html, head, body. Dans la balise body, on trouve ensuite d'autres balises qui elles-mêmes contiennent une ou plusieurs autres balises.

Ces balises sont des nœuds du DOM. Le terme « nœud » est un terme générique qui sert à désigner tous les objets contenus dans le DOM.

Il existe plusieurs types de nœud :

- **nœud élément**

  Ce sont les balises html.

  Ex : div, form, span, p, ...

- **nœud attribut**

  Ce sont les attributs html.

  Ex : href, src, class, id, style, type, ...

- **nœud texte**

  Ce sont les contenus des éléments html

  Ex : le texte d'un paragraphe ou d'un span…

Recherche dans le DOM

La notion de « **méthode** » fréquemment utilisée à partir de maintenant est en quelque sorte une « fonction » appliquée à des objets.
Les objets ont des « **attributs** » et des « **méthodes** »

Recherche d'éléments

Méthode getElementById()

getElementById() prend en paramètre un id et retourne l'élément html ayant cet id.

Étant donné que les id d'éléments doivent être uniques, ils constituent un moyen utile d'accéder rapidement à un élément spécifique.

Par exemple :

```
<body>
      <h1 id='titre'>Titre principal</h1>
      <p>Un paragraphe</p>
 </body>
```

On pourra retrouver l'élément ayant l'id titre avec le code JavaScript suivant :

```
const titre = document.getElementById('titre') ;
```

Méthode getElementsByClassName()

La méthode getElementsByClassName() prend en paramètre une classe et retourne une liste d'éléments ayant cette classe.

Par exemple :

```
<h1 id='titre'>Titre principal</h1>
<p class='text'>Un paragraphe</p>
<p class='text'>Un paragraphe</p>
```

On pourra retrouver la liste des éléments de classe « text » avec le code JavaScript suivant :

```
const paragraphes = document.getElementsByClassName('text');
const paragraphe = paragraphes[1];
```

Méthode getElementsByTagName()

La méthode getElementsByTagName() prend en paramètre un nom de balise et retourne la liste des éléments html ayant cette balise.

Exemple :

```
<div>
    <article>Contenu 1</article>
    <article>Contenu 2</article>
    <article>Contenu 3</article>
</div>
```

On pourra retrouver la liste des balises « article » avec le code JavaScript suivant :

```
const articles = document.getElementsByTagName('article');
const thirdArticle = articles[2];
```

Méthode querySelector()

La méthode querySelector() prend en paramètre un sélecteur et retourne **le premier** élément qui correspond à la recherche. Elle ne renvoie **pas** une liste des résultats.

Par exemple :

```
<p class='text'>Paragraphe 1</p>
<div id='container' class='main'>
    <h1 id='titre'>Titre principal</h1>
     <p class='text'>Paragraphe 2</p>
     <p class='text'>Paragraphe 3</p>
</div>
<p class='text'>Un paragraphe</p>
```

On peut retrouver le premier élément correspondant au sélecteur « #container p.text » avec le code JavaScript suivant :

```
const paragraphe = document.querySelector('#container p.text') ;
console.log(paragraphe.textContent) ; // Affiche Paragraphe 2 ;
```

Méthode querySelectorAll()

Pour retourner **une liste de résultats** qui correspondent à la recherche que vous souhaitez faire il faut utiliser la fonction querySelectorAll(), qui fonctionne de la même manière

On peut retrouver tous les éléments correspondant au sélecteur « #container p.text » avec le code JavaScript suivant :

```
const paragraphes = document.querySelectorAll('#container p.text') ;
```

Exercice sur Moodle : associer chaque méthode à ce qu'elles font.

Noeuds adjacents, parents et enfants

Une page html est une arborescence d'éléments (body, div, p, a, span, etc.)

Lorsqu'on récupère un élément html dans une variable au moyen de l'une des méthodes citées ci-dessus, il est possible de récupérer les éléments adjacents, parents ou enfants.

On utilise les propriétés suivantes :

element.children : cette propriété retourne la liste des enfants de cet élément ;

element.parentElement : cette propriété retourne l'élément parent de celui-ci ;

element.nextElementSibling /**element.previousElementSibling** : ces propriétés permettent de naviguer vers l'élément suivant / précédent de même niveau que notre élément.

```
<div id='parent'>
    <div id='previous'>Précédent</div>
    <div id='main'>
        <p>Paragraphe 1</p>
        <p>Paragraphe 2</p>
    </div>
    <div id='next'>Suivant</div>
</div>
```

et si l'on considère que nous avons le code JavaScript suivant :

```
const elt = document.getElementById('main');
```

nous aurons ceci :

```
console.log(elt.children) ; // retourne les éléments de type  p  qui sont les enfants de
l'élément #main
console.log(elt.parentElement) ; // retourne la  div  qui a l'id  parent
elt.nextElementSibling retourne l'élément qui a l'id  next
elt.previousElementSibling retournera l'élément qui a l'id  previous
```

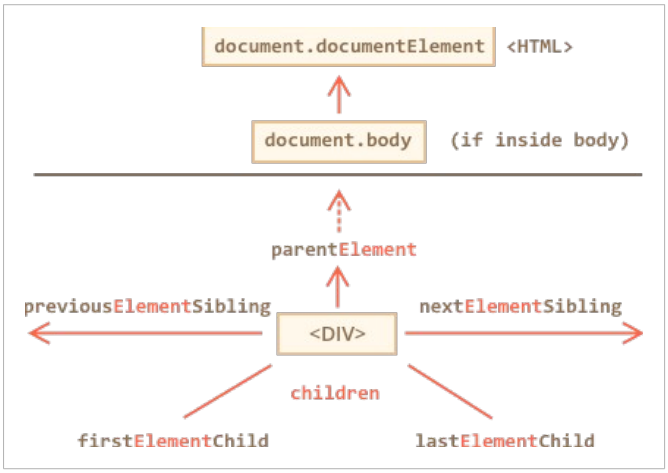


figure 2 : les différents nœuds

Le contenu des éléments

On peut récupérer le contenu d'un élément avec la propriété innerHTML ou textContent. Par exemple :

```
<body>
      <h1 id='titre'>Titre <span>principal</span></h1>
      <p>Un paragraphe</p>
</body>
```

On peut récupérer le contenu du titre avec le code JavaScript suivant :

```
const titre = document.getElementById('titre') ;
const contenu = titre.innerHTML ; // Retourne « Titre <span>principal</span> »
const contenu = titre.textContent ; // Retourne « Titre principal »
```

La propriété innerHTML conserve le balisage html alors que la propriété textContent ne le conserve pas.

Parcours des listes d'éléments

Chaque liste récupérée par les méthodes getElementsByTagName(), getElementsByClassName() et querySelectorAll() sont des listes de « nœuds » (HTMLCollection ou NodeList). Par exemple :

```
<div id="main">
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
</div>
```

Pour connaître le nombre d'éléments de cette liste, on utilise la propriété length :

```
const liste = document.getElementsByTagName('p');
const nombre = liste.length ;
```

Pour parcourir tous les paragraphes de la liste, on utilise une boucle for :

```
for (let i = 0; i < items.length; i++) {
  console.log(items[i]);
}
```

Modification du DOM

Modification du contenu

textContent

Par exemple, avec le code JavaScript suivant :

```
let elt = document.getElementById('main');
elt.textContent = "Une ligne de texte";
```

l'élément qui a l'id 'main' aura un nouveau contenu ; le HTML deviendra donc :

```
<div id="main">Une ligne de texte</div>
```

innerHTML

Par exemple, avec le code JavaScript suivant :

```
let elt = document.getElementById('main');
elt.innerHTML = '<ul><li>Elément 1</li><li>Elément 2</li></ul>';
```

l'élément qui a l'id 'main' aura un nouveau contenu ; le HTML deviendra donc :

```
<div id='main'>
    <ul>
        <li>Elément 1</li>
        <li>Elément 2</li>
    </ul>
</div>
```

Ajout / Suppression de classes

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

Partie html

```
<div class="header top-box green element">
```

Partie Js :

```
const elt = querySelector('div') ;
```

Voici quelques méthodes possibles :

```
elt.classList.add('nouvelleClasse');// Ajoute la classe nouvelleClasse à l'élément
elt.classList.remove('nouvelleClasse'); // Supprime la classe nouvelleClasse
elt.classList.contains('nouvelleClasse');   // Retourne false
elt.classList.replace('green', 'blue'): // Remplace green par blue
```

Modification de styles

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

Pour modifier la couleur d'un élément :

```
elt.style.color = '#fff';      // Change la couleur du texte de l'élément à blanche
```

Pour modifier la couleur d'arrière-plan d'un élément :

```
elt.style.backgroundColor = '#000';
```

Pour modifier la graisse d'un élément :

```
elt.style.fontWeight = 'bold'; // Met le texte de l'élément en gras
```

Les propriétés utilisables correspondent aux propriété css,
en adoptant l'écriture « CamelCase »

CSS	JS
background-color	backgroundColor
font-weight	fontWeight

Ajout / Suppression d'attributs

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

element.setAttribute(name, value) prend en paramètres le nom de l'attribut et sa valeur et ne retourne rien.

Vous pouvez utiliser les fonctions **getAttribute** et **removeAttribute** pour avoir encore plus de contrôle sur les attributs.

Voici quelques exemples avec elt faisant référence à un élément de type input :

```
elt.setAttribute('type', 'password');   // Change le type de l'input en un type password
elt.setAttribute('name', 'my-password');    // Change le nom de l'input en my-password
elt.getAttribute('name');               // Retourne my-password
```

Ajout d'éléments

L'ajout d'éléments dans le DOM se fait en 2 étapes :

La fonction **document.createElement** permet de créer un nouvel élément. Cette méthode prend en paramètre le nom de la balise de notre élément et renvoie l'élément créé.

Exemple :

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

Un élément créé avec cette fonction ne fait pas encore partie du document, vous ne le verrez donc pas sur votre page. Pour le voir, il va d'abord falloir l'ajouter en tant qu'enfant à un élément.

Il existe plusieurs façons d'ajouter un élément dans notre page. La plus connue est **appendChild**. Cette méthode permet d'ajouter un élément à la liste des enfants du parent depuis lequel la fonction est appelée.

Voici un exemple :

```
const newElt = document.createElement('div');
let elt = document.getElementById('main');
elt.appendChild(newElt);
```

appendChild est une méthode qui s'applique sur l'élément parent (ici : **elt**)

elt.appendChild(element) prend en paramètre l'élément à ajouter en tant qu'enfant.

Suppression d'éléments

Quelques exemples

```
const newElt = document.createElement('div');
let elt = document.getElementById('main');
elt.appendChild(newElt);
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
```

Les événements

Un clic sur un bouton est un événement initié par l'utilisateur ; la saisie d'un texte dans un formulaire également.

L'affichage d'une image sur une page web est aussi un événement ; un événement autonome. Le chargement complet d'une page web est aussi un événement.

Gestionnaires d'événements

Syntaxe

Exemple : Dans le code html suivant, on souhaite afficher un message lorsque l'image est cliquée

```
<body>
    <h1 class='titre'>Gestion d'événements</h1>
    <img id='img1' src='https://picsum.photos/300/230'/>
</body>
```

En Javascript :

Etape 1. On définit la cible

```
const target = document.getElementById('img1');
```

Etape 2. On positionne un écouteur

```
target.addEventListener('click', action);
```

La méthode addEventListener a deux paramètres :

- le premier paramètre est l'événement qui est écouté (click, mouseover, keypress, submit, ...)
- le second paramètre est le nom de l'action qui sera effectuée lorsque l'événement se produira. On peut choisir le nom que l'on veut.

Etape 3. On code l'action

```
function action(event) {
    alert("image cliquée");
}
```

L'action exécutée lorsque l'événement se produira est une fonction. Son nom correspond au nom défini à l'étape 2.

on appelle cette fonction, la fonction de callback et qu'elle a toujours un paramètre (objet event)

Références :

https://developer.mozilla.org/fr/docs/Learn/JavaScript/Building_blocks/Events

Bonne pratique

Il est déconseillé de placer votre code js sur vos éléments html.

Ex : <button onclick="bgChange()">Press me</button>

C'est une pratique qui était à la mode, mais qui est maintenant **déconseillée**. Cela correspond à une bonne pratique de développement qui est la « séparation des préoccupations » (separation of concerns).

A vous de pratiquer

Exercice sur Moodle : compléter le code pour le rendre fonctionnel

Liste des principaux événements en JS

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

Souris

Click : https://developer.mozilla.org/fr/docs/Web/API/Element/click_event

mouseover : https://developer.mozilla.org/fr/docs/Web/API/Element/mouseover_event

mousemove :https://developer.mozilla.org/fr/docs/Web/API/Element/mousemove_event

Formulaire

Change : https://developer.mozilla.org/fr/docs/Web/API/HTMLElement/change_event

submit : https://developer.mozilla.org/fr/docs/Web/API/HTMLFormElement/submit_event

Clavier

Keyup & keydown : https://developer.mozilla.org/fr/docs/Web/API/KeyboardEvent/code

Utilisation des outils de développement

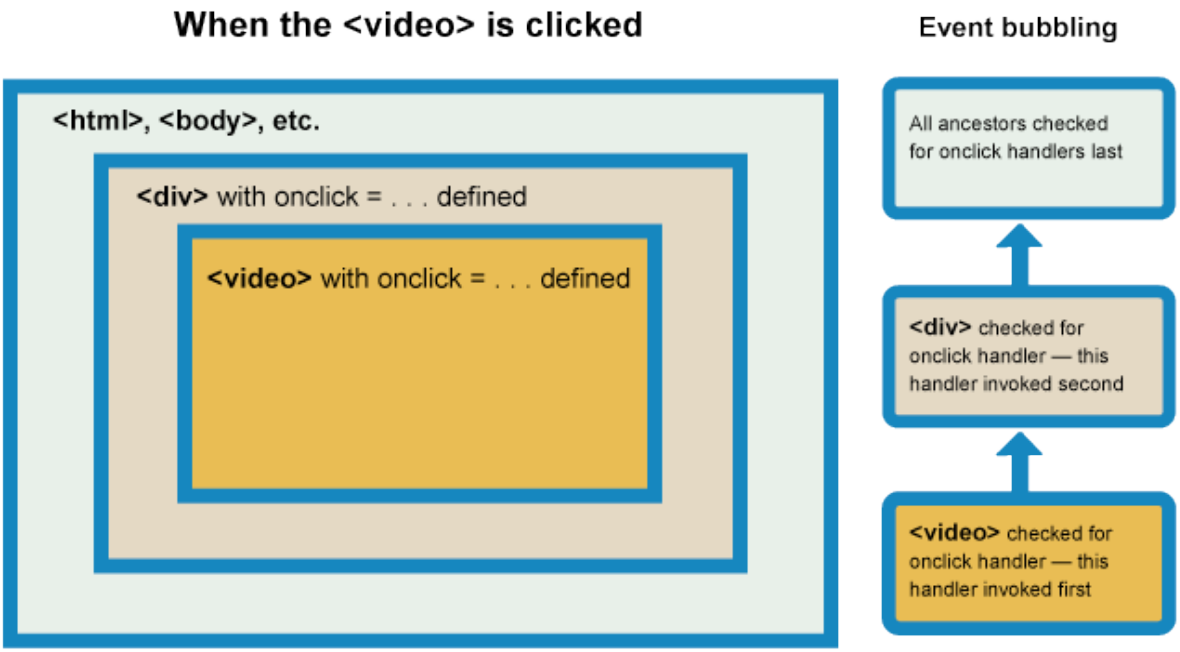
A l'issue des TP vous devrez savoir utiliser efficacement :

- La console
- Le debogueur

Attributs et méthodes des événements

e.stopPropagation()

Lorsqu'un événement est déclenché, il est d'abord reçu par l'élément cible, mais il est ensuite « remonté » vers les éléments parents.



Avec stopPropagation(), vous pouvez ainsi empêcher que les éléments parents reçoivent l'événement.

Admettons par exemple que nous ayons un élément pour lequel nous voulons afficher un message lorsque l'on clique dessus. Mais à l'intérieur de cet élément, nous avons aussi un autre élément qui doit nous afficher un autre message lorsque l'on clique dessus.

Par défaut, si nous cliquons dans l'élément intérieur, le message va s'afficher, puis notre élément parent va lui aussi recevoir l'événement du clic et encore changer le message. Pour éviter cela, nous devons stopper la propagation de l'événement.

Ainsi, dans l'élément intérieur, nous ferons ceci :

```
elementInterieur.addEventListener('click', function(event) {
    event.stopPropagation();
    elementAvecMessage.innerHTML = "Message de l'élément intérieur";
```

```
});
```

De cette manière, lorsque l'on clique sur l'élément intérieur, l'élément parent ne recevra plus le clic, et seul l'élément intérieur affichera son message. En revanche, en cliquant directement dans l'élément parent, sans être dans l'élément intérieur, l'élément parent recevra bien l'événement et affichera son message.

e.preventDefault()

La méthode preventDefault permet d'empécher l'action « naturelle » du navigateur (soumission de formulaire, clic sur un lien, etc.)

Admettons que vous vouliez réagir au clic sur un lien. Vous pouvez mettre en place un gestionnaire d'événement comme nous l'avons vu précédemment.

Exemple :

```
<body>
    <a id="lien" href="http://webmmi.iut-tlse3.fr">IUT</h1>
</body>
```

En Javascript :

```
const target = document.getElementById('lien');
target.addEventListener('click', action);
function action(event) {
    event.preventDefault();
    alert("lien cliqué");
}
```

On ajoute la méthode preventDefault(), pour empêcher l'internaute d'être dirigé sur l'url du lien. Le navigateur reste sur la page et le message d'alerte s'affiche bien.

e.target

e.target est un objet qui fait référence à la cible sur laquelle est positionné l'écouteur d'événement.

target est une propriété utile lorsque vous voulez définir le même gestionnaire d'événements sur plusieurs éléments et affecter une action à chacun d'entre eux quand un événement se produit sur eux.

https://jsfiddle.net/x3gpahyt/1/

Plus généralement, target est une propriété utile pour recueillir des informations sur l'événement qui vient de se produire.