TP de Javascript portant sur les aspects Gestion d'événement et Interactions Utilisateur.

Ce TP doit être fait en une durée de 3h.

Première partie : Prérequis.

- Connaissances de base des langages HTML et CSS.
- Notion fondamentales de JavaScript (variables, fonctions, conditions, boucles, etc.)

Quelques rappels techninques :

Insertion d'un script dans une page HTML : plusieurs manières possibles.

- Entre les balises `<script></script>`
- Relié à un document .js

```
<DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Ma Page Web</title>
</head>
<body>
    <script>
        // Ici vous pouvez écrire votre code JavaScript dans une page
HTML
    </script>
    <script src="script.js"></script>
    <!-- Ici vous pouvez écrire votre script dans un fichié séparé -->
</body>
</html>
```

Le DOM (Document Object Model)

Le DOM représente la structure d'un document HTML comme un **arbre d'objets**.
JavaScript permet de manipuler cet arbre pour **modifier le contenu, la structure et le style** des éléments.

Sélection d'Éléments

- `document.getElementById('id')`
  Sélectionne un élément par son **ID unique**.

- `document.querySelector('selecteurCSS')`
  Sélectionne le **premier élément** correspondant au sélecteur CSS.

- `document.querySelectorAll('selecteurCSS')`
  Sélectionne **tous les éléments** correspondant au sélecteur CSS (retourne une **NodeList**).
- `document.getElementsByClassName('nomClasse')`
  Sélectionne les éléments par leur **nom de classe** (retourne une **HTMLCollection**).
- `document.getElementsByTagName('nomTag')`
  Sélectionne les éléments par leur **nom de balise** (retourne une **HTMLCollection**).

La Gestion d'Événements

Les **événements** sont des actions ou des occurrences qui se produisent dans le navigateur (ex : clic de souris, appui sur une touche, chargement de page, soumission de formulaire).

Écouteurs d'Événements (Event Listeners)

Fonctions qui "écoutent" la survenance d'un événement sur un élément spécifique et exécutent du code en réponse.

Syntaxe de base :

```
element.addEventListener('typeEvenement', fonctionGestionnaire, [options]);
```

- `typeEvenement` : Type de l'événement (ex : `'click'`, `'mouseover'`, `'keydown'`, `'submit'`)
- `fonctionGestionnaire` : Fonction exécutée lorsque l'événement se produit
- `options` *(facultatif)* : Par exemple :
  - `{ capture: true }` pour la phase de capture
  - `{ once: true }` pour que l'écouteur ne s'exécute qu'une seule fois

L'objet `Event`

Lorsqu'un événement se déclenche, une instance de l'objet `Event` est passée comme argument à la fonction gestionnaire.

Propriétés courantes :

- `event.target` : Élément **sur lequel** l'événement s'est produit
- `event.currentTarget` : Élément **auquel** l'écouteur est attaché
- `event.type` : Type de l'événement (ex : `'click'`)
- `event.clientX`, `event.clientY` : Coordonnées X et Y de la souris
- `event.keyCode`, `event.key` : Code et nom de la touche pressée (clavier)

Méthodes courantes :

- `event.preventDefault()`
  Annule l'action par défaut (ex : empêcher la soumission d'un formulaire)

- `event.stopPropagation()`
  Empêche la propagation de l'événement dans la hiérarchie du DOM (effet "bulle")

Délégation d'Événements

Technique consistant à attacher un **seul écouteur** à un élément parent au lieu d'un écouteur par enfant.

- Utile pour des **listes dynamiques** ou des éléments **ajoutés dynamiquement**
- Utilise `event.target` pour déterminer **quel enfant** a déclenché l'événement

Exercice 1 : Les Bases de la Gestion d'Événements (30-45 minutes)

Objectif : Comprendre les différentes manières d'attacher des événements et l'objet `Event`.

1. Configuration initiale :
   - Créez un fichier `index.html` avec la structure de base (DOCTYPE, head, body).
   - Créez un fichier `style.css` lié à `index.html`.
   - Créez un fichier `script.js` lié à `index.html` (juste avant la balise `</body>` fermante).
2. HTML (`index.html`) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>TP Événements JavaScript</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Gestion d'Événements</h1>

    <button id="monBoutonHTML" onclick="alert('Bouton cliqué via
attribut HTML!')">Cliquez-moi (HTML)</button>

    <button id="monBoutonDOM0">Cliquez-moi (DOM Level 0)</button>

    <button id="monBoutonDOM2">Cliquez-moi (DOM Level 2)</button>

    <p id="messageInteraction">Interaction : Aucune</p>

    <div id="zoneSurvol">
        Survolez-moi
    </div>

    <p id="coordonneesSouris">X: 0, Y: 0</p>
```

```
    <input type="text" id="champTexte" placeholder="Tapez du texte
ici...">
    <p id="affichageTouche">Dernière touche pressée : </p>

    <script src="script.js"></script>
</body>
</html>
```

1. CSS (`style.css`):

```
<style>
body {
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 15px;
    margin-top: 50px;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
}

#zoneSurvol {
    width: 200px;
    height: 100px;
    background-color: lightblue;
    border: 1px solid blue;
    display: flex;
    justify-content: center;
    align-items: center;
    font-weight: bold;
}

input[type="text"] {
    padding: 8px;
    font-size: 16px;
    width: 300px;
}
</style>
```

1. JavaScript (`script.js`) - Partie 1 : Attacher des événements
   - **Attribut HTML** `onclick` : (Déjà fait dans `index.html`) C'est la méthode la plus ancienne et la moins recommandée car elle mélange HTML et JS.
   - **DOM Level 0 (`on` property)** :

- Sélectionnez le bouton avec l'ID `monBoutonDOM0`.
- Attachez-lui un gestionnaire d'événement pour le `click` en utilisant la propriété `onclick`.
- Ce gestionnaire devra afficher une alerte ou modifier le texte du paragraphe `messageInteraction`.

– **DOM Level 2 (`addEventListener`)** :

- Sélectionnez le bouton avec l'ID `monBoutonDOM2`.
- Attachez-lui un gestionnaire d'événement pour le `click` en utilisant `addEventListener`.
- Ce gestionnaire devra afficher une alerte ou modifier le texte du paragraphe `messageInteraction`.

Question : Quelle est la principale différence entre `onproperty` et `addEventListener` concernant l'ajout de multiples gestionnaires pour le même événement ?

```
// script.js - Partie 1

// 1. DOM Level 0 - onproperty
const boutonDOM0 = document.getElementById('monBoutonDOM0');
const messageInteraction =
document.getElementById('messageInteraction');

boutonDOM0.onclick = function() {
    messageInteraction.textContent = 'Interaction : Bouton DOM Level 0
cliqué !';
    console.log('Bouton DOM Level 0 cliqué !');
};

// 2. DOM Level 2 - addEventListener
const boutonDOM2 = document.getElementById('monBoutonDOM2');

function gererClicBoutonDOM2(event) {
    messageInteraction.textContent = 'Interaction : Bouton DOM Level 2
cliqué !';
    console.log('Bouton DOM Level 2 cliqué !', event.target); //
event.target est le bouton
}

boutonDOM2.addEventListener('click', gererClicBoutonDOM2);

// Ajout d'un deuxième écouteur pour démontrer la différence avec DOM
Level 0
boutonDOM2.addEventListener('click', function() {
    console.log('Deuxième écouteur pour bouton DOM Level 2 !');
});

// Question : Quelle est la principale différence ?
// Réponse attendue : `addEventListener` permet d'attacher plusieurs
```

```
gestionnaires au même événement sur le même élément,
// tandis que `onproperty` écrase le gestionnaire précédent.
```

1. JavaScript (`script.js`) - Partie 2 : L'objet `Event` et différents types d'événements
   - **Événements de souris (`mouseover`, `mouseout`, `mousemove`)** :
     - Sélectionnez l'élément `zoneSurvol`.
     - Lorsqu'on survole (`mouseover`) la zone, changez son arrière-plan en vert clair.
     - Lorsqu'on quitte (`mouseout`) la zone, restaurez son arrière-plan en bleu clair.
     - Lorsqu'on déplace la souris (`mousemove`) à l'intérieur de la zone, affichez les coordonnées `clientX` et `clientY` de la souris dans le paragraphe `coordonneesSouris`.
   - **Événements de clavier (`keydown`, `keyup`, `input`)** :
     - Sélectionnez le champ de texte avec l'ID `champTexte`.
     - Lorsqu'une touche est pressée (`keydown`) dans le champ, affichez dans le paragraphe `affichageTouche` le nom de la touche (`event.key`) et son code (`event.keyCode` - bien que `event.key` soit plus moderne).

Bonus : Utilisez l'événement `input` pour afficher en temps réel le contenu du champ de texte dans un autre paragraphe.

```
// script.js - Partie 2

// Événements de souris
const zoneSurvol = document.getElementById('zoneSurvol');
const coordonneesSouris =
document.getElementById('coordonneesSouris');

zoneSurvol.addEventListener('mouseover', function() {
    zoneSurvol.style.backgroundColor = 'lightgreen';
});

zoneSurvol.addEventListener('mouseout', function() {
    zoneSurvol.style.backgroundColor = 'lightblue';
});

zoneSurvol.addEventListener('mousemove', function(event) {
    coordonneesSouris.textContent = `X: ${event.clientX}, Y: $
{event.clientY}`;
});

// Événements de clavier
const champTexte = document.getElementById('champTexte');
const affichageTouche = document.getElementById('affichageTouche');

champTexte.addEventListener('keydown', function(event) {
```

```
    affichageTouche.textContent = `Dernière touche pressée : $
{event.key} (Code: ${event.keyCode})`;
});

// Bonus : Affichage en temps réel du contenu du champ
// Ajout d'un paragraphe pour le bonus dans index.html: <p
id="contenuChamp">Contenu : </p>
const contenuChamp = document.getElementById('contenuChamp'); //
Ajoutez cette ligne si vous faites le bonus

champTexte.addEventListener('input', function(event) {
    // contenuChamp.textContent = `Contenu : $
{event.target.value}`; // Activez si vous faites le bonus
});
```

Exercice 2 : Interactions Avancées et Prévention du Comportement par Défaut (45-60 minutes)

Objectif : Maîtriser la prévention du comportement par défaut et créer des interactions plus complexes.

1. HTML (`index.html`) - Ajoutez à la suite du contenu existant :

```
<h2>Interactions Avancées</h2>

    <form id="monFormulaire">
        <label for="nom">Nom :</label>
        <input type="text" id="nom" name="nom" required>
        <button type="submit">Soumettre</button>
        <p id="messageForm"></p>
    </form>

    <a href="https://www.google.com" id="monLien">Lien vers Google
(cliquez pour annuler)</a>
    <p id="messageLien"></p>

    <h2>Liste Interactive</h2>
    <ul id="listeDynamique">
        <li>Élément 1</li>
        <li>Élément 2</li>
        <li>Élément 3</li>
    </ul>
    <button id="ajouterElement">Ajouter un élément</button>
```

1. JavaScript (`script.js`) - Partie 3 : Prévention du comportement par défaut
   - **Formulaire (`submit`)** :
     - Sélectionnez le formulaire avec l'ID `monFormulaire`.
     - Ajoutez un écouteur d'événement pour le type `submit`.

        - À l'intérieur du gestionnaire, utilisez `event.preventDefault()` pour empêcher la soumission par défaut du formulaire (qui rechargerait la page).
        - Affichez un message de confirmation ou d'erreur dans le paragraphe `messageForm` (ex: "Formulaire soumis via JavaScript !" ou "Veuillez remplir le champ nom.").
    - **Lien (`click`)** :
        - Sélectionnez le lien avec l'ID `monLien`.
        - Ajoutez un écouteur d'événement pour le type `click`.
        - À l'intérieur du gestionnaire, utilisez `event.preventDefault()` pour empêcher le lien de rediriger vers Google.
        - Affichez un message dans le paragraphe `messageLien` (ex: "Redirection annulée !").

```
// script.js - Partie 3

// Prévention du comportement par défaut
const monFormulaire = document.getElementById('monFormulaire');
const messageForm = document.getElementById('messageForm');

monFormulaire.addEventListener('submit', function(event) {
    event.preventDefault(); // Empêche la soumission par défaut du
formulaire

    const nomInput = document.getElementById('nom');
    if (nomInput.value.trim() === '') {
        messageForm.textContent = 'Veuillez remplir le champ Nom !';
        messageForm.style.color = 'red';
    } else {
        messageForm.textContent = `Formulaire soumis par $
{nomInput.value} (via JavaScript) !`;
        messageForm.style.color = 'green';
        // Ici, vous enverriez normalement les données à un serveur
via Fetch API ou XMLHttpRequest
    }
});

const monLien = document.getElementById('monLien');
const messageLien = document.getElementById('messageLien');

monLien.addEventListener('click', function(event) {
    event.preventDefault(); // Empêche la redirection par défaut du
lien
    messageLien.textContent = 'Redirection vers Google annulée !';
    messageLien.style.color = 'orange';
});
```

1. JavaScript (`script.js`) - Partie 4 : Délégation d'événements et ajout/suppression dynamique

- **Liste interactive (`click`)** :
  - Sélectionnez la liste `listeDynamique`.
  - Au lieu d'ajouter un écouteur de clic à chaque `<li>`, ajoutez un seul écouteur de clic à la liste `<ul>` elle-même.
  - Dans le gestionnaire, utilisez `event.target` pour déterminer quel élément `<li>` a été cliqué.
  - Si l'élément cliqué est un `<li>`, changez son couleur de texte en rouge et affichez son contenu dans la console.
- **Ajout dynamique d'éléments** :
  - Sélectionnez le bouton `ajouterElement`.
  - Lorsqu'on clique sur ce bouton, ajoutez un nouvel élément `<li>` à la `listeDynamique`. Le nouvel élément doit également répondre aux clics grâce à la délégation d'événements.

```
// script.js - Partie 4

// Délégation d'événements
const listeDynamique = document.getElementById('listeDynamique');
const ajouterElementBouton =
document.getElementById('ajouterElement');
let compteurElements = 3; // Pour les nouveaux éléments

// Écouteur sur l'élément parent (<ul>)
listeDynamique.addEventListener('click', function(event) {
    // Vérifie si l'élément cliqué est bien un <li>
    if (event.target.tagName === 'LI') {
        console.log('Élément cliqué :', event.target.textContent);
        event.target.style.color = 'red';
        // Pour le remettre à la normale après un court délai
(optionnel)
        setTimeout(() => {
            event.target.style.color = 'initial';
        }, 500);
    }
});

// Ajout dynamique d'éléments
ajouterElementBouton.addEventListener('click', function() {
    compteurElements++;
    const nouvelElement = document.createElement('li');
    nouvelElement.textContent = `Nouvel Élément ${compteurElements}`;
    listeDynamique.appendChild(nouvelElement);
    console.log('Nouvel élément ajouté :', nouvelElement.textContent);
});
```

Exercice 3 : Mini-projet - Création d'un "Détecteur de Forme" (60-75 minutes)

Objectif : Mettre en pratique l'ensemble des concepts pour créer une interaction utilisateur ludique et plus complexe.

1. **Idée** : Créer une zone sur laquelle l'utilisateur dessine avec la souris (en maintenant le clic). Le système doit détecter la forme approximative (par exemple, un carré ou une ligne) basée sur la direction et la distance du mouvement. Pour simplifier, nous nous contenterons de détecter si le mouvement est majoritairement horizontal ou vertical.

2. **HTML (`index.html`) - Ajoutez à la suite** :

```
<h2>Détecteur de Mouvement</h2>
<div id="zoneDessin"></div>
<p id="resultatDetection">Détection : Aucune</p>
<button id="resetDetection">Réinitialiser</button>
```

1. **CSS (`style.css`) - Ajoutez** :

```
<style>
#zoneDessin {
    width: 400px;
    height: 250px;
    border: 2px solid #333;
    background-color: #f0f0f0;
    cursor: crosshair;
    position: relative; /* Important pour les points */
}

.point {
    position: absolute;
    width: 5px;
    height: 5px;
    background-color: blue;
    border-radius: 50%;
}
</style>
```

1. **JavaScript (`script.js`) - Partie 5 : Détecteur de Mouvement**
   - **Variables d'état** :
     - `isDrawing = false` : Un booléen pour savoir si l'utilisateur est en train de cliquer et glisser.
     - `startX`, `startY` : Coordonnées du point de départ du clic.
     - `pathPoints = []` : Un tableau pour stocker les points du chemin dessiné.
   - **Événements sur `zoneDessin`** :
     - `mousedown` :
       - Mettre `isDrawing` à `true`.
       - Enregistrer `event.clientX` et `event.clientY` comme `startX` et `startY`.

        - Effacer les points précédents et le texte de détection.
        - Ajouter un point de départ visuel.
    - `mousemove` :
        - Si `isDrawing` est `true` :
            - Ajouter le point actuel (`event.clientX`, `event.clientY`) au tableau `pathPoints`.
            - Ajouter un petit point visuel (un `div` avec la classe `point`) à la `zoneDessin` pour "dessiner" le chemin.
    - `mouseup` :
        - Mettre `isDrawing` à `false`.
        - Appeler une fonction `detecterForme()` pour analyser `pathPoints`.
- **Fonction `detecterForme()`** :
    - Si `pathPoints.length` est suffisant (ex: > 10 points) :
        - Calculer la différence totale en X (`deltaXTotal`) et en Y (`deltaYTotal`) entre le premier et le dernier point, ou entre tous les points pour une meilleure robustesse.
        - Comparer `Math.abs(deltaXTotal)` et `Math.abs(deltaYTotal)` :
            - Si `deltaXTotal` est significativement plus grand que `deltaYTotal`, c'est une ligne horizontale.
            - Si `deltaYTotal` est significativement plus grand que `deltaXTotal`, c'est une ligne verticale.
            - Sinon, c'est "Autre forme" ou "Mouvement complexe".
        - Mettre à jour le texte du paragraphe `resultatDetection`.
    - Si pas assez de points, afficher "Mouvement trop court".
- **Bouton `resetDetection`** :
    - Lorsque cliqué, effacer tous les points visuels de la `zoneDessin`, réinitialiser `pathPoints`, et mettre `resultatDetection` à "Détection : Aucune".

```
// script.js - Partie 5

const zoneDessin = document.getElementById('zoneDessin');
const resultatDetection =
document.getElementById('resultatDetection');
const resetDetectionBtn = document.getElementById('resetDetection');

let isDrawing = false;
let startX, startY;
let pathPoints = [];

// Fonction pour ajouter un point visuel
function addPointToDrawing(x, y) {
    const pointDiv = document.createElement('div');
```

```
    pointDiv.classList.add('point');
    // Les coordonnées doivent être relatives à la zoneDessin
    const rect = zoneDessin.getBoundingClientRect();
    pointDiv.style.left = `${x - rect.left - 2.5}px`; // -2.5 pour
centrer le point de 5px
    pointDiv.style.top = `${y - rect.top - 2.5}px`;
    zoneDessin.appendChild(pointDiv);
}

// Efface tous les points visuels
function clearDrawing() {
    const points = zoneDessin.querySelectorAll('.point');
    points.forEach(point => point.remove());
}

zoneDessin.addEventListener('mousedown', function(event) {
    isDrawing = true;
    startX = event.clientX;
    startY = event.clientY;
    pathPoints = [{ x: startX, y: startY }];
    clearDrawing();
    addPointToDrawing(startX, startY); // Ajoute le point de départ
    resultatDetection.textContent = 'Détection : Dessin en cours...';
});

zoneDessin.addEventListener('mousemove', function(event) {
    if (isDrawing) {
        pathPoints.push({ x: event.clientX, y: event.clientY });
        addPointToDrawing(event.clientX, event.clientY);
    }
});

zoneDessin.addEventListener('mouseup', function() {
    isDrawing = false;
    if (pathPoints.length > 10) { // Assez de points pour une
détection significative
        detecterForme();
    } else {
        resultatDetection.textContent = 'Détection : Mouvement trop
court.';
    }
});

// Gérer le cas où la souris sort de la zone de dessin avant mouseup
zoneDessin.addEventListener('mouseleave', function() {
    if (isDrawing) {
        isDrawing = false;
        if (pathPoints.length > 10) {
            detecterForme();
        } else {
```

```
            resultatDetection.textContent = 'Détection : Mouvement
trop court.';
        }
    }
});


function detecterForme() {
    if (pathPoints.length < 2) {
        resultatDetection.textContent = 'Détection : Pas assez de
points pour analyser.';
        return;
    }

    const firstPoint = pathPoints[0];
    const lastPoint = pathPoints[pathPoints.length - 1];

    // Calculer la différence totale en X et Y
    const deltaX = lastPoint.x - firstPoint.x;
    const deltaY = lastPoint.y - firstPoint.y;

    // Pour une détection plus robuste, on peut sommer les différences
absolues entre chaque point successif
    let totalDeltaXAbs = 0;
    let totalDeltaYAbs = 0;
    for (let i = 1; i < pathPoints.length; i++) {
        totalDeltaXAbs += Math.abs(pathPoints[i].x - pathPoints[i-
1].x);
        totalDeltaYAbs += Math.abs(pathPoints[i].y - pathPoints[i-
1].y);
    }


    // Détecter si c'est majoritairement horizontal ou vertical
    const tolerance = 0.5; // Tolérance pour considérer un mouvement
comme dominant
    if (totalDeltaXAbs > totalDeltaYAbs * (1 / tolerance)) { //
Beaucoup plus de mouvement horizontal que vertical
        resultatDetection.textContent = 'Détection : Ligne Horizontale
!';
        resultatDetection.style.color = 'blue';
    } else if (totalDeltaYAbs > totalDeltaXAbs * (1 / tolerance)) { //
Beaucoup plus de mouvement vertical que horizontal
        resultatDetection.textContent = 'Détection : Ligne
Verticale !';
        resultatDetection.style.color = 'green';
    } else if (Math.abs(deltaX) < 20 && Math.abs(deltaY) < 20 &&
pathPoints.length > 10) {
         // Si le mouvement est court mais contient beaucoup de
points, c'est peut-être un point ou un petit gribouillis
```

```
        resultatDetection.textContent = 'Détection : Gribouillis / Point';
        resultatDetection.style.color = 'purple';
    }
    else {
        resultatDetection.textContent = 'Détection : Autre forme (mouvement complexe)';
        resultatDetection.style.color = 'red';
    }
}

resetDetectionBtn.addEventListener('click', function() {
    clearDrawing();
    pathPoints = [];
    resultatDetection.textContent = 'Détection : Aucune';
    resultatDetection.style.color = 'initial';
});
```