Module 9 (laboratoire)
DOM et progra événementielle

Exercice 1 : Le survol des peintures

Le but de cet exercice est d'aborder un premier exemple de programmation événementielle tout en s'exerçant à manipuler les arguments optionnels dans les fonctions.

Étape 1

Créez un document HTML avec le contenu suivant. Observez sa structure.

```
<!DOCTYPE html>
<html>
<head>
  <title>Quelques peintures</title>
  <meta charset="utf-8" />
  <style>
    #peintures {
      border: 1px solid green;
      background-color: lightgreen;
      padding: 0;
      display: flex;
      align-items: center;
      margin-bottom: 0;
    }
    .peinture {
      border: 2px solid black;
      margin: 4px;
      width: 150px;
    }
    #commentaire {
      margin-top: 0;
      border: 1px solid green;
      background-color: lemonchiffon;
      min-height: 80px;
      padding: 8px;
      display: flex;
      flex-direction: column;
      justify-content: center;
    }
  </style>
</head>
<body>
  <div id="peintures"></div>
  <div id="commentaire"><p id="pcommentaire"></p></div>
</body>
</html>
```

Dans le document ci-dessus, le div identifié par « peintures » va recueillir des images de peintures alors que le paragraphe identifié par « pcommentaire », lui, va permettre d'afficher des informations au sujet de ces peintures.

Étape 2

Ajoutez une balise <script> dans la partie <head> du document pour placer la définition des diverses fonctions demandées (à terme, tout le code Javascript devrait s'y trouver).

Créez tout d'abord une fonction `créePeinture(src)` qui va créer et renvoyer un nouvel élément HTML correspondant à une balise <img> associé au fichier dont l'adresse est donnée en argument et possédant la classe `peinture`. Cette fonction se contentera de créer l'élément et de le renvoyer, pas de l'insérer dans le document.

Pour tester votre fonction, vous pouvez ajouter le script suivant dans le document.

```
let elem = créePeinture("escaliers.jpg");
document.getElementById("peintures").appendChild(elem);
```

À quel endroit du document devez-vous ajouter ces deux lignes ? Pourquoi ?

Étape 3

Pour éviter de devoir placer des bouts de code un peu partout dans le document, on peut associer l'exécution de ces bouts de code à un événement qui se produira dans le futur, quand tous les éléments HTML nécessaires auront bien été créés. Dans la plupart des cas, on utilise l'événement « onload » (= chargement terminé) de l'objet « window », qui correspond au document tout entier.

Pour faire en sorte que les lignes de l'étape précédente soient exécutées après la fin du chargement du document, utilisez le code suivant (que vous pouvez ajouter dans le script situé dans la balise <head>.

```
window.onload = function () {
  ... code à exécuter ...
};
```

Étape 4

Vous souvenez-vous du CSS ?

Ajoutez une règle CSS faisant en sorte que, lorsque le curseur de la souris survole une image de class `peinture`, sa bordure (normalement noire) devienne jaune.

Étape 5

Modifiez la fonction `créePeinture` en lui ajoutant deux paramètres. La nouvelle fonction, `créePeinture(src, titre, auteur)` devra renvoyer un élément HTML correspondant à la peinture en question.

Il s'agira toujours d'une image associée au fichier donné comme premier argument mais, désormais, l'élément HTML en question possédera une donnée appelée « info » (voir l'atelier précédent et la propriété `dataset`) et contenant un commentaire formaté comme suit

```
« Maison aux escaliers » par Escher
```

et composé du titre entre guillemets français (entités HTML `&laquo;` et `&raquo;`) puis du mot « par » et du nom de l'auteur.

De plus, l'élément HTML aura une action associée à l'événement « mouseover » qui se déclenche quand la souris passe sur l'image. L'action en question consistera à placer dans le paragraphe identifié par « pcommentaire » un commentaire composé de « Commentaire : » suivi du contenu placé dans « info ».

Ainsi, pour la peinture d'Escher, le commentaire à afficher devrait être

```
Commentaire : « Maison aux escaliers » par Escher
```

Pour tester votre code, modifier le script exécuté après le chargement du document en

```
let elem = créePeinture("escaliers.jpg", "Maison aux escaliers",
"Escher");
```

Étape 6

Votre code est censé fonctionner avec n'importe quelle peinture. Pour tester si c'est le cas, utilisez le code suivant.

```
window.onload = function () {
  let elem = créePeinture("escaliers.jpg", "Maison aux escaliers",
"Escher");
  document.getElementById("peintures").appendChild(elem);
  elem = créePeinture("radeau.jpg", "Radeau de la Méduse",
"Géricault") ;
  document.getElementById("peintures").appendChild(elem);
};
```

Voyez-vous une chose à améliorer dans ce bout de code ?

Étape 7

Il est temps de passer à une version utilisant plus de peintures… Heureusement, tous les détails concernant les peintures sont définis dans le tableau suivant. Copiez-collez sa déclaration au début de votre script.

```
let peintures = [
  ["escaliers.jpg", "Maison aux escaliers", "Escher"],
  ["frappis.jpg", "", "Frappis"],
  ["radeau.jpg", "Radeau de la Méduse", "Géricault"],
  ["joconde.png", "Joconde revisitée"],
  ["fleurs.jpg"],
  ["pipe.jpg", "La trahison des images", "Magritte"],
  ["persistence.jpg", "The Persistence of Memory", "Dali"]
];
```

Premier obstacle… chaque peinture est présentée sous la forme d'un tableau citant, dans l'ordre, le fichier-source, le titre et l'auteur. On ne peut donc pas se contenter d'une boucle telle que

```
for (let peinture of peintures) {
```

```
  ... créePeinture(peinture) ...
}
```

vu que `créePeinture` attend trois arguments donnés un par un plutôt qu'un tableau.

La solution la plus simple consiste sans doute à utiliser l'opérateur « `...` », appelé le « spread operator ». Celui-ci convertit justement un tableau en une suite d'éléments qui peuvent être utilisés (entre autres) comme arguments d'une fonction.

Voici ce que devrait donc devenir le nouveau code pour `window.onload` (y compris l'amélioration que vous avez dû effectuer à l'étape précédente).

```
window.onload = function () {
  let cadre = document.getElementById("peintures");
  for (let peinture of peintures) {
    let elem = créePeinture(...peinture);
    cadre.appendChild(elem);
  }
};
```

Exécutez le tout et observez les commentaires…

Étape 8

Un peu de nettoyage dans les commentaires.

On veut désormais que les commentaires prennent la forme suivante :

- `Commentaire : « Maison aux escaliers » par Escher` *si toutes les informations sont disponibles* ;
- `Commentaire : « Joconde revisitée » (auteur inconnu)` *s'il manque l'auteur* ;
- `Commentaire : aucune information` *si on n'a aucune information*.

Modifiez donc la partie du code qui définit l'information « data » pour tenir compte des consignes ci-dessus.

Au fait, notez que la peinture avec des explosions de couleurs a comme titre « ». Le commentaire correspondant devrait donc être `Commentaire : « » par Frappis`.

Exercice 2 : Sagas sur PC

L'objectif de cet exercice est d'utiliser la programmation événementielle pour rendre visibles ou invisibles certaines parties du document en fonction des actions de l'utilisateur (à la manière des balises « spoiler » qu'on trouve dans certains forums).

Étape 1

Comme d'habitude, commencez par créer un document HTML avec le contenu suivant.

```
<!DOCTYPE html>
<html>
<head>
  <title>Quelques sagas de jeux de rôle sur PC</title>
```

```
  <meta charset="utf-8" />
  <style>
    body {
      background-color: black;
    }
    #wrapper {
      width: 600px;
      margin: 0 auto;
      color: #E5DFC5;
      text-align: center;
      font-family: sans-serif;
    }
    h1 {
      border-bottom: 2px solid #91C7A9;
    }
    h2 {
      border: 2px solid #A92F41;
      background-color: #A92F41;
      color: black;
      padding: 2px;
      margin: 12px 2px 0 2px;
      border-radius: 8px;
    }
    ul.saga {
      background-color: #B48375;
      list-style-type: none;
      margin: 0px 2px;
      padding: 0px;
      border: 2px solid #A92F41;
      border-bottom-left-radius: 8px;
      border-bottom-right-radius: 8px;
    }
    .saga li {
      display: block;
      padding: 2px;
      color: black;
    }
  </style>
  <script>
    window.onload = function () {
      ajouteSaga("Baldur's Gate",
        "Baldur's Gate I (1998)",
        "Baldur's Gate: Tales of the Sword Coast (1999)",
        "Baldur's Gate II: Shadows of Amn (2000)",
        "Baldur's Gate II: Throne of Bhaal (2001)",
        "Baldur's Gate: Enhanced Edition (2012)",
        "Baldur's Gate II: Enhanced Edition (2013)",
        "Baldur's Gate: Enhanced Edition - Siege of Dragonspear
(2016)"
      );
      ajouteSaga("Icewind Dales",
        "Icewind Dales (2000)",
        "Icewind Dales II (2002)",
        "Icewind Dales: Enhanced Edition (2014)"
```

```
      );
      ajouteSaga("Neverwinter Nights",
        "Neverwinter Nights (2002)",
        "Neverwinter Nights: Shadows of Undrentide (2003)",
        "Neverwinter Nights: Hordes of the Underdark (2003)",
        "Neverwinter Nights 2 (2006)",
        "Neverwinter Nights 2: Mask of the Betrayer (2007)",
        "Neverwinter Nights 2: Storm of Zehir (2008)",
        "Neverwinter Nights 2: Mysteries of Westgate (2009)"
      );
      ajouteSaga("Dragon Age",
        "Dragon Age: Origins (2009)",
        "Dragon Age II (2011)",
        "Dragon Age: Inquisition (2014)"
      );
      ajouteSaga("Mass Effect",
        "Mass Effect (2007)",
        "Mass Effect 2 (2010)",
        "Mass Effect 3 (2012)",
        "Mass Effect: Andromeda (2017)"
      );
      ajouteSaga("Fallout",
        "Fallout (1997)",
        "Fallout 2 (1998)",
        "Fallout Tactics (2001)",
        "Fallout: Brotherhood of Steel (2004)",
        "Fallout 3 (2008)",
        "Fallout: New Vegas (2010)",
        "Fallout 4 (2015)"
      );
    }
  </script>
</head>
<body>
  <div id="wrapper">
    <h1>Quelques sagas de JDR sur PC</h1>
    <p>Cliquez sur un titre pour afficher/cacher les détails.</p>
    <div id="sagas"></div>
  </div>
</body>
</html>
```

Étape 2

On a associé à l'événement « onload » du document une série d'actions qui devraient permettre d'ajouter le contenu de la page. Ce contenu parle de diverses sagas de jeux de rôle sur PC, reprenant pour chacune d'entre elles un titre général puis la liste des épisodes correspondants.

Par exemple, la dernière saga, « Fallout », regroupe 7 jeux parus entre 1997 et 2015.

Dans un premier temps (et pour que la page puisse être chargée sans erreur), définissez une fonction `ajouteSaga(saga)` qui se contente d'ajouter le titre de la saga dans le div

identifié par « sagas », sous la forme d'un élément HTML de balise h2. La fonction ne tiendra compte (pour le moment) que du premier argument reçu.

Étape 3

Pour pouvoir ajouter les titres des sagas, vous avez sans doute utilisé `document.getElementById` à un certain moment, afin de pouvoir cibler le div identifié par « sagas ». Vous l'avez sans doute utilisé dans la fonction `ajouteSaga`, ce qui signifie que la recherche s'effectue à chaque appel de la fonction. Il y a moyen de mieux faire…

Que pensez-vous de définir une variable globale définie une fois pour toutes ?

Suffit-il d'ajouter la ligne

```
let cadre = document.getElementById("sagas");
```

au début du script et d'ensuite référencer la variable globale `cadre` depuis la fonction `ajouteSaga` ? Testez cette solution, lisez l'erreur obtenue et assurez-vous que vous en comprenez la raison.

Solutionnez le problème en conservant la déclaration de la variable à cet endroit mais en déplaçant son initialisation vers un endroit/moment où elle pourra se faire…

Étape 4

Revenez à la fonction ajouteSaga et modifiez-là pour qu'en plus d'ajouter le titre de la saga (sous forme d'une balise h2), elle ajoute également la liste des épisodes (format ul, éléments li).

Dans le code existant, la fonction ajouteSaga est appelée plusieurs fois avec des nombres d'arguments différents. Pour la coder, utilisez le procédé suivant.

```
function ajouteSaga (titre, ...épisodes) {
  ... code où « épisodes » s'utilise comme un tableau ...
}
```

L'opérateur « … » utilisé ci-dessus s'appelle le « rest operator » parce qu'il capture tous les arguments restants dans un tableau. À l'intérieur de la fonction, vous pouvez utiliser « épisodes » comme s'il s'agissait d'un tableau reprenant (dans ce cas-ci) le titre des différents épisodes de la saga. Naturellement, on ne peut utiliser qu'un seul « rest operator » par fonction et celui-ci doit obligatoirement être le dernier paramètre formel.

Spread et rest operators. Les deux opérateurs s'écrivent de la même manière mais ils ne s'utilisent pas au même endroit et ont des effets opposés. Le « rest operator » s'utilise dans l'en-tête d'une fonction et signifie « on rassemble le reste des arguments dans le tableau indiqué ». Le « spread operator », lui, s'utilise dans un appel de fonction et signifie « on déballe le tableau donné et utilise ses éléments comme des paramètres effectifs pour la fonction. »

Codez la fonction `ajouteSaga`, n'oubliez pas d'ajouter la classe « saga » à l'élément HTML correspondant à la liste <ul>.

Étape 5

Il faut encore rendre le document HTML-dynamique. Lors de l'affichage de la page, on désire que toutes les sagas soient « refermées » (c'est-à-dire que seul le titre est visible, pas les épisodes). Ensuite, en cliquant sur le titre d'une saga, l'utilisateur peut demander à ce que la liste des épisodes devienne visible. Un second clic la rendra à nouveau invisible.

Voici quelques consignes et conseils quant à la manière d'implémenter tout ça…

1. On peut rendre un élément HTML invisible en modifiant la propriété CSS « display ». Une valeur « none » signifie que l'élément n'est pas affiché ; dans le cas de la liste <ul>, une valeur « block » signifie que l'élément sera affiché (en tant que bloc – voir le cours d'IG1 pour les différences entre block et inline).

   Comme il s'agit d'une propriété CSS, il faut donc passer par la propriété « style » de l'objet DOM. Ainsi, si la variable `liste` cible une liste <ul>, on peut la rendre invisible avec l'instruction `liste.style.display = "none";`

2. Pour savoir quoi faire en cas de clic sur un titre, il faut savoir si la liste associée est visible ou pas… il s'agit donc d'une information à retenir d'une manière ou d'une autre. Cela pourrait se faire en utilisant des variables globales ou encore le dataset des titres… ici, on choisit de le faire en utilisant les classes des titres en prenant la convention suivante.

   Si un titre h2 a la classe « contenu-visible », cela signifie que la liste qui le suit est visible. Sinon, cela signifie que la liste qui le suit est cachée.

3. Au chargement de la page, toutes les listes doivent être cachées. Tenez-en compte dans la fonction `ajouteSaga` en initialisant correctement la propriété « display » pour la liste des épisodes.

Dans un premier temps, créez une fonction `toggleVisListe()` qui contiendra le code à exécuter lorsque l'utilisateur cliquera sur un titre. À l'intérieur de la fonction, vous pouvez utiliser « this » pour faire référence au titre en question. Utilisez les méthodes permettant de parcourir l'arborescence HTML pour pouvoir cibler la liste qui suit. Travaillez votre code jusqu'il soit lisible et bien structuré.

Ensuite, modifiez ajouteSaga pour qu'elle associe cette fonction à l'événement « onclick » sur les titres h2 créés.

Étape 6

Il ne reste plus qu'à « arrondir les angles »… ou plutôt, à les « désarrondir ». Les quatre coins des titres h2 sont arrondis par défaut (voir les règles CSS du document). Ce n'est pas un problème pour les deux coins du haut mais l'arrondi des deux coins du bas est plutôt moche quand la liste est dépliée/visible.

Trouvez une solution pour enlever l'arrondi de ces deux coins-là quand la liste est visible (allez au plus simple).

Exercice 3 : Au saut du

Dans de nombreuses applications, lorsqu'on demande de choisir une ou plusieurs options parmi une liste donnée, on utilise une interface à 2 colonnes : la colonne de gauche reprend la liste de toutes les options et la colonne de droite cite les options que vous avez choisies. D'un simple clic, vous pouvez faire passer une option de la colonne de gauche vers la colonne de droite (c'est-à-dire l'ajouter à votre sélection) ou de la colonne de droite à la colonne de gauche (c'est-à-dire l'enlever de votre sélection).

Le but de cet exercice est d'implémenter une interface de ce genre en utilisant deux listes HTML.

Voici le fichier de départ.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <style>
      ul {
        border: 2px solid #4b3124;
        height: 200px;
        list-style-type: none;
        margin: 8px;
        padding: 0;
      }
      li {
        border: 1px solid #e7dfc6;
        background-color: #f3efe2;
        border-radius: 4px;
        margin: 4px;
        padding: 2px;
        text-align: center;
        cursor: ew-resize;
      }
      li:hover {
        background-color: #e7dfc6;
        border: 1px solid #4b3124;
      }
    </style>
  </head>
  <body>
    <table width="100%">
    <tr><td width="50%">
    <ul id="liste1">
      <li class="elem">Un</li>
      <li class="elem">Deux</li>
      <li class="elem">Trois</li>
      <li class="elem">Quatre</li>
      <li class="elem">Cinq</li>
    </ul>
    </td><td width="50%">
    <ul id="liste2">
```

```
    </ul>
  </body>
</html>
```

Celui-ci affiche deux cadres (représentant les listes). Au départ, 5 éléments se trouvent dans la colonne de gauche.

Pour vous faciliter la tâche, on a attribué les identifiants « `liste1` » et « `liste2` » aux deux listes et la classe « `elem` » à chacun des éléments.

Votre but est d'ajouter du code Javascript pour permettre à l'utilisateur de faire passer un des éléments d'une liste à l'autre en effectuant un simple clic. Ainsi, au départ, si l'utilisateur clique sur un des éléments de la liste, celui-ci devra disparaître de la `liste1` et apparaître dans la `liste2`. S'il clique à nouveau sur cet élément (alors qu'il se trouve dans la seconde liste), l'élément disparaîtra de la `liste2` et réapparaîtra dans la `liste1`. L'ordre des éléments dans les listes n'a pas d'importance.

Note. Faites votre code Javascript de manière indépendante du code HTML. Autrement dit, n'ajoutez pas manuellement des actions à l'événement « onclick » des éléments de la liste. Ces ajouts devront se faire après le chargement de la page, de manière dynamique (c'est-à-dire via du code Javascript). Pour ce faire, créez une fonction (par exemple `init`) qui se chargera de tout mettre en place et ajoutez la ligne « `window.onload = init;` » signifiant que, dès que tous les éléments de la fenêtre auront été chargés, la fonction `init` sera automatiquement exécutée.

Exercice 4 : Boutons réactifs

En manipulant le DOM, on peut modifier les actions associées à certains événements : plutôt que de fixer une fois pour toutes la tâche associée à un bouton, on peut modifier celle-ci à sa guise.

Étape 1

Créez un fichier HTML contenant un bouton dont le texte indique « Cliquez moi ! ».

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
  </head>
  <body>
    <button id="but">Cliquez moi !</button>
  </body>
</html>
```

Étape 2

Dans un premier temps, on se fixe comme but de transformer le texte du bouton en « Merci ! » lorsque l'utilisateur clique sur lui. Considérez les trois solutions et prenez note desquelles fonctionnent et desquelles échouent…

Solution 1. Ajouter au code HTML du bouton l'attribut suivant.

```
onclick="this.innerHTML = 'Merci !';"
```

Solution 2. Ajouter une balise <script> dans la partie <head> avec la définition

```
function action () { this.innerHTML = 'Merci !'; }
```

et, dans la description HTML du bouton, l'attribut

```
onclick="action();"
```

Solution 3. Ne rien ajouter dans la description HTML du bouton mais utiliser le script suivant.

```
function action () { this.innerHTML = 'Merci !'; }
window.onload = function () {
  document.getElementById("but").onclick = action;
};
```

Étape 3

Dans la suite, basez-vous principalement sur la 3e solution proposée ci-dessus.

Modifiez à nouveau le code pour que, après le premier clic, le bouton affiche « Encore » puis, après le second clic, « Merci ».

Concrètement, cela veut dire que le premier clic devra entraîner une modification non seulement du texte du bouton (`b.innerHTML`) mais également de l'événement associé au clic (`b.onclick`).

Étape 4

Au début de votre script Javascript, définissez une variable globale `nbClicksRequis`.

```
const nbClicksRequis = 5;
```

Cette variable représente le nombre de fois qu'il faudra cliquer sur le bouton avant que celui-ci ne dise « Merci ! ».

Dans le cas où `nbClicksRequis = 5`, le bouton affichera donc « Cliquez moi ! » lors du chargement de la page, puis « Encore 4 fois ! » après un premier clic, « Encore 3 fois ! » après un second clic, … « Encore 1 fois ! » après un quatrième clic, et finalement « Merci ! » après le cinquième clic.

Étape 5

Dans l'étape précédente, vous avez sans doute utilisé une variable globale pour mémoriser le nombre de clicks déjà effectués sur le bouton en question.

Pour cette étape, il va sans doute falloir changer d'approche… ici, on vous demande d'ajouter 30 boutons sur la page HTML (via copier/coller ou, mieux, en utilisant Javascript). Chacun de ces 30 boutons aura le même comportement que le bouton de l'étape précédente ; les nombres de clics seront comptabilisés individuellement.

L'objectif est que chacun des 30 boutons utilisent la **même** fonction pour l'événement `onclick` (et pas 30 fonctions différentes utilisant chacune une variable globale différente). On pourrait s'en tirer avec une variable globale de type tableau… mais il y a

plus simple : rappelez-vous que chacun des éléments du DOM représentant un bouton est un objet Javascript…

- à qui on peut donc ajouter des propriétés à la volée et
- qui peut être référencé par le mot-clef `this` à l'intérieur de la fonction associée à l'événement `onclick`.

Tirez avantage de cela en stockant le nombre de clicks déjà effectués sur un bouton en tant que « donnée locale ».