

# JavaScript Events

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Les événements</b>	<b>3</b>
A. La définition d'un événement.....	3
B. Supprimer un gestionnaire d'événements .....	4
C. Les différents événements .....	5
D. Créer ses propres événements .....	6
E. Exercice : Quiz .....	7
<b>III. Le fonctionnement avancé des événements</b>	<b>8</b>
A. L'objet Event .....	8
B. La propagation des événements .....	9
C. La capture d'événements .....	9
D. Empêcher la propagation d'événements.....	11
E. PreventDefault.....	11
F. Exercice : Quiz .....	12
<b>IV. Essentiel</b>	<b>13</b>
<b>V. Auto-évaluation</b>	<b>13</b>
A. Exercice .....	13
B. Test.....	15
<b>Solutions des exercices</b>	<b>16</b>

## I. Contexte

**Durée :** 1 heure

**Environnement de travail :** VS CODE

### Contexte

La programmation événementielle en JavaScript est une approche qui permet de gérer les événements déclenchés par l'utilisateur ou par le système, en leur attribuant des actions à réaliser en réponse. Cette approche est couramment utilisée dans le développement web pour créer des interfaces utilisateur interactives.

La programmation événementielle en JavaScript peut sembler complexe, mais en réalité elle est assez simple à comprendre. Il suffit de lier des fonctions à des événements pour leur donner une action à réaliser en réponse. Un événement est donc lié à un code à exécuter. JavaScript fournit des fonctions intégrées pour la gestion des événements, ce qui facilite grandement le travail des développeurs.

Dans ce cours, nous allons explorer les bases de la programmation événementielle en JavaScript, en expliquant les concepts clés tels que les événements, les écouteurs d'événements et la façon de les utiliser pour créer des interfaces utilisateur interactives.

### Attention

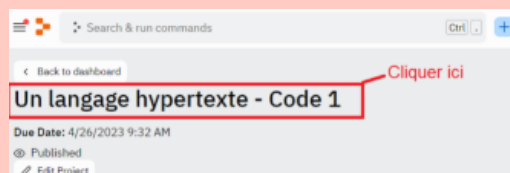
Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

- 1) Vous connecter à votre compte sur <https://replit.com/> (ou créer gratuitement votre compte)
- 2) Rejoindre la Team Code Studi du module via ce lien : <https://replit.com/teams/join/mmurvlgippxuasordloklllbqskoim-programmer-avec-javascript>

Une fois ces étapes effectuées, nous vous conseillons de rafraîchir votre navigateur si le code ne s'affiche pas.

En cas de problème, redémarrez votre navigateur et vérifiez que vous avez bien accepté les cookies de connexion nécessaires avant de recommencer la procédure.

Pour accéder au code dans votre cours, cliquez sur le nom du lien Replit dans la fenêtre. Par exemple :



## II. Les événements

### A. La définition d'un événement

#### Définition

Les événements sont des actions qui se produisent sur une page web, tels que le clic d'un bouton ou la soumission d'un formulaire. En JavaScript, nous pouvons détecter ces événements et y répondre en exécutant une fonction. Les événements peuvent être déclenchés par l'utilisateur, le navigateur ou le code JavaScript lui-même.

Il existe de nombreux types d'événements en JavaScript, tels que le clic, le chargement de la page, la soumission de formulaire, le déplacement de la souris, etc. Chaque type d'événement est associé à un objet Event qui contient des informations sur l'événement, telles que la cible de l'événement (l'élément HTML qui a déclenché l'événement), le type d'événement et les données spécifiques à l'événement.

Des événements se produisent en permanence sur une page web, mais si on veut exécuter du code après un événement, on doit « écouter un événement ».

#### Méthode

Pour écouter un événement en JavaScript, il est possible d'utiliser la méthode `addEventListener()`. Cette méthode prend 2 arguments : le type d'événement à écouter et la fonction à exécuter lorsque l'événement se produit. Voici un exemple de code qui écoute le clic d'un bouton et exécute une fonction lorsque le clic se produit :

```
1 const button = document.querySelector('button');
2
3 button.addEventListener('click', () => {
4   alert('Bouton cliqué');
5 });
```

[cf. ]

Dans cet exemple, la méthode `querySelector()` est utilisée pour sélectionner le premier élément de type « *button* » dans le document HTML. Ensuite, la méthode `addEventListener()` est utilisée pour écouter le clic de ce bouton et exécuter une fonction anonyme qui affiche une alerte avec le texte « *Bouton cliqué* ».

## B. Supprimer un gestionnaire d'événements

#### Méthode

La méthode `removeEventListener()` est utilisée pour supprimer un gestionnaire d'événements d'un élément. Cette méthode prend également 2 arguments : le type d'événement et une fonction de gestionnaire d'événements.

```
1 const button = document.querySelector('button');
2
3 function handleClick(event) {
4   console.log('Bouton cliqué');
5 }
6
7 button.addEventListener('click', handleClick);
8
9 // Supprime le gestionnaire d'événements handleClick
10 button.removeEventListener('click', handleClick);
```

[cf. ]

Dans ce code, si on clique sur le bouton, la méthode `handleClick` ne sera pas appelée, car le `removeEventListener` annule le `addEventListener`.

### C. Les différents événements

Événement	Description
click	Déclenché lorsque l'utilisateur clique sur un élément de la page.
keyup	Déclenché lorsque l'utilisateur relâche une touche de clavier.
submit	Déclenché lorsque l'utilisateur soumet un formulaire.
load	Déclenché lorsque la page web a fini de charger.
mouseover	Déclenché lorsque le curseur de la souris se déplace sur un élément.
mouseout	Déclenché lorsque le curseur de la souris quitte un élément.
scroll	Déclenché lorsque l'utilisateur fait défiler la page.
resize	Déclenché lorsque la taille de la fenêtre du navigateur est modifiée.
contextmenu	Déclenché lorsque l'utilisateur fait un clic droit sur un élément.
mousedown	Déclenché lorsque l'utilisateur enfonce un bouton de la souris sur un élément.
mouseup	Déclenché lorsque l'utilisateur relâche un bouton de la souris sur un élément.
touchstart	Déclenché lorsque l'utilisateur touche l'écran tactile.
touchend	Déclenché lorsque l'utilisateur relâche l'écran tactile.
touchmove	Déclenché lorsque l'utilisateur déplace son doigt sur l'écran tactile.
error	Déclenché lorsqu'une erreur se produit lors du chargement de la page ou d'une ressource.
change	Déclenché lorsqu'un élément de formulaire (ex. : input) est modifié.

Cette liste n'est pas exhaustive, mais elle donne une idée des événements les plus couramment utilisés en JavaScript. Les développeurs peuvent également créer leurs propres événements personnalisés pour répondre aux besoins spécifiques de leur application.

## D. Créer ses propres événements

Outre les événements intégrés à JavaScript, il est également possible de créer des événements personnalisés pour répondre aux besoins spécifiques de votre application. Les événements personnalisés peuvent être déclenchés à n'importe quel moment du code, et les écouteurs d'événements peuvent leur être attachés de la même manière que pour les événements intégrés.

### Méthode

Pour créer un événement personnalisé en JavaScript, vous devez utiliser l'objet `CustomEvent`. Voici un exemple de code qui crée un événement personnalisé nommé `monEvenement` :

```
1 // Création de l'événement personnalisé
2 const monEvenement = new CustomEvent('monEvenement');
3
4 // Ajout d'un écouteur d'événement pour l'événement personnalisé
5 document.addEventListener('monEvenement', () => {
6   console.log('L\'événement personnalisé a été déclenché !');
7 });
8
9 // Déclenchement de l'événement personnalisé
10 document.dispatchEvent(monEvenement);
```

[cf. ]

Dans cet exemple, nous avons créé un objet `CustomEvent` avec le nom `monEvenement`. Ensuite, nous avons ajouté un écouteur d'événement pour cet événement personnalisé en utilisant la méthode `addEventListener` de l'objet `document`. Finalement, nous avons déclenché l'événement personnalisé en utilisant la méthode `dispatchEvent` de l'objet `document`.

### Complément

Il est également possible de passer des données avec l'événement personnalisé en utilisant l'objet `CustomEvent`. Voici un exemple de code qui crée un événement personnalisé nommé `monEvenement` avec des données :

```
1 // Création de l'événement personnalisé avec des données
2 const monEvenement = new CustomEvent('monEvenement', {
3   detail: {
4     message: 'Bonjour tout le monde !'
5   }
6 });
7
8 // Ajout d'un écouteur d'événement pour l'événement personnalisé avec des données
9 document.addEventListener('monEvenement', (event) => {
10   console.log(`Message reçu : ${event.detail.message}`);
11 });
12
13 // Déclenchement de l'événement personnalisé avec des données
14 document.dispatchEvent(monEvenement);
```

[cf. ]

Dans cet exemple, nous avons créé un objet `CustomEvent` nommé `monEvenement` avec l'option `detail` pour y inclure des données. Ensuite, nous avons ajouté un écouteur d'événement pour cet événement personnalisé en utilisant la méthode `addEventListener` de l'objet `document`. Finalement, nous avons déclenché l'événement personnalisé en utilisant la méthode `dispatchEvent` de l'objet `document`. Nous aurons donc ce message dans la console : « Message reçu : Bonjour tout le monde ! ».

**Attention** Utiliser les événements depuis le HTML

```
1 <button onclick="maFonction()">cliquez ici</button>
```

[cf. ]

L'utilisation de l'attribut « *onclick* » (ou autre) dans les balises HTML est considérée comme une technique obsolète en JavaScript. Cela est dû à plusieurs raisons, notamment :

- L'attribut « *onclick* » mélange la logique de présentation (HTML) et la logique de comportement (JavaScript), ce qui peut rendre le code difficile à maintenir et à déboguer.
- L'utilisation de l'attribut « *onclick* » peut conduire à des problèmes de sécurité, comme l'injection de code malveillant.
- L'ajout d'événements directement dans le code HTML limite la flexibilité et la réutilisabilité du code.

Au lieu d'utiliser l'attribut « *onclick* » dans les balises HTML, il est préférable d'utiliser les écouteurs d'événements JavaScript pour gérer les événements. Les écouteurs d'événements permettent une séparation plus claire entre la logique de présentation et la logique de comportement. Cela rend le code plus facile à maintenir, à déboguer et à réutiliser.

**E. Exercice : Quiz**

[solution n°1 p.17]

**Question 1**

Quel est l'objet utilisé pour créer des événements personnalisés en JavaScript ?

- ☐ Event
- ☐ CustomEvent
- ☐ MouseEvent

**Question 2**

Comment ajoute-t-on un écouteur d'événement pour un événement personnalisé ?

- ☐ `addEventListener('nomEvenement', function() {})`
- ☐ `on('nomEvenement', function() {})`
- ☐ `attachEvent('nomEvenement', function() {})`

**Question 3**

Comment déclenche-t-on un événement personnalisé ?

- ☐ `dispatch('nomEvenement')`
- ☐ `trigger('nomEvenement')`
- ☐ `dispatchEvent('nomEvenement')`

**Question 4**

Peut-on inclure des données avec un événement personnalisé ?

- ☐ Non, ce n'est pas possible
- ☐ Oui, en utilisant l'objet `CustomEvent`
- ☐ Oui, en utilisant l'objet `Event`

Question 5

Quel est l'événement JavaScript qui est déclenché lorsqu'un utilisateur appuie sur une touche du clavier ?

- ☐ `onscroll`
- ☐ `onkeydown`
- ☐ `onmouseover`

### III. Le fonctionnement avancé des événements

#### A. L'objet `Event`

##### Définition

L'objet `Event` en JavaScript représente un événement qui se produit dans une page web, tel qu'un clic de souris, une pression de touche, un chargement de page, etc. Cet objet contient des informations sur l'événement, telles que le type d'événement, la cible de l'événement, les coordonnées de la souris, etc. Nous recevons une instance de cet objet lors du déclenchement de l'écouteur d'un événement.

Voici quelques propriétés et méthodes utiles de l'objet `Event` :

Propriétés :

- `type` : le type d'événement (ex.: « `click` », « `load` », « `keydown` », etc.),
- `target` : l'élément qui a déclenché l'événement,
- `currentTarget` : l'élément qui gère actuellement l'événement,
- `clientX`, `clientY` : les coordonnées de la souris par rapport à la fenêtre,
- `pageX`, `pageY` : les coordonnées de la souris par rapport au document,
- `key` : le code de la touche qui a été pressée (pour les événements de type « `keydown` »).

Méthodes :

- `preventDefault()` : empêche le comportement par défaut de l'événement (ex.: empêche le lien de suivre son URL),
- `stopPropagation()` : empêche la propagation de l'événement aux parents de l'élément cible,
- `stopImmediatePropagation()` : empêche la propagation de l'événement aux parents de l'élément cible et empêche l'exécution d'autres gestionnaires d'événements pour cet événement sur l'élément actuel.



**Méthode**

Voici un exemple de code JavaScript qui écoute l'événement `keyup` pour détecter quand l'utilisateur a appuyé sur la touche « Entrée » :

```
1  const input = document.getElementById('myInput');
2
3  input.addEventListener('keyup', function(event) {
4    if (event.key === 'Enter') {
5      console.log('La touche "Entrée" a été pressée !');
6    }
7  });
```

[cf. ]

On utilise ici la variable `Event`, qui est une instance de la classe `Event`. Depuis cette variable, on peut récupérer la propriété « `key` », soit la touche sur laquelle l'utilisateur a cliqué.

## B. La propagation des événements

### Propagation d'événements

La propagation d'événements est le mécanisme par lequel un événement se propage à travers l'arborescence DOM, de l'élément qui l'a déclenché jusqu'à la racine du document.

Lorsqu'un événement se produit sur un élément, l'agent utilisateur (le navigateur web) vérifie s'il existe un gestionnaire d'événements pour cet événement sur cet élément (le `addEventListener` qu'on aurait pu créer nous-mêmes). Si c'est le cas, il appelle ce gestionnaire d'événements. Ensuite, l'agent utilisateur remonte l'arborescence DOM pour vérifier si d'autres éléments ont également des gestionnaires d'événements pour cet événement. Si c'est le cas, il appelle également ces gestionnaires d'événements, en commençant par l'élément parent immédiat de l'élément d'origine, puis en remontant jusqu'à la racine du document. On va donc de l'élément le plus bas ciblé par l'événement, jusqu'au plus haut, la racine.

**Complément** **Capture et propagation**

La propagation d'événements se produit en 2 phases : la phase de **capture** et la phase de **propagation** (ou de bouillonnement). La phase de capture se produit en premier et consiste à appeler les gestionnaires d'événements des éléments parents, en descendant l'arborescence DOM. Ensuite, la phase de propagation se produit, qui consiste à appeler les gestionnaires d'événements des éléments parents, en remontant l'arborescence DOM.

La propagation d'événements peut être utile pour gérer les événements de manière plus globale, en ajoutant un seul gestionnaire d'événements à un élément parent, plutôt que d'ajouter des gestionnaires d'événements à chaque élément individuel qui a besoin de traiter l'événement. Cela peut simplifier le code et le rendre plus facile à maintenir.

Cependant, la propagation d'événements peut également rendre le code plus difficile à comprendre et à déboguer, car il peut être difficile de savoir quels gestionnaires d'événements seront appelés en premier, en particulier dans les documents complexes.

## C. La capture d'événements

**Définition**

La capture d'événements est un mécanisme de gestion des événements qui permet de gérer un événement à un niveau supérieur dans l'arborescence du Document Object Model (DOM).

En règle générale, lorsque vous cliquez sur un élément dans une page web, l'événement de clic est d'abord géré par l'élément lui-même (par exemple là où vous avez cliqué), puis par ses parents dans l'arborescence DOM, jusqu'à ce que l'événement soit propagé à l'ensemble du document.

#### Méthode

Voici le code à renseigner dans votre fichier HTML :

```
1 <div id="parent">
2   <button id="enfant">Cliquez moi</button>
3 </div>
```

Voici le code à renseigner dans votre fichier JavaScript :

```
1 const parent = document.querySelector('#parent');
2 const enfant = document.querySelector('#enfant');
3
4 parent.addEventListener('click', function() {
5   console.log('Parent');
6 });
7
8 enfant.addEventListener('click', function() {
9   console.log('Enfant');
10 });
```

[cf. ]

Si nous cliquons sur le bouton, nous cliquons aussi sur le div, car le div contient le bouton. Mais, dans ce cas-là, si on clique sur le bouton, nous aurons en console d'abord « *Enfant* », puis « *Parent* ». Car l'événement géré en premier est celui de l'élément le plus bas dans le DOM (ici le button). Après cela, on remonte dans le DOM.

#### Méthode

Avec la capture d'événements, vous pouvez gérer l'événement à un niveau supérieur de l'arborescence DOM avant qu'il ne soit propagé à l'élément cible. Cela signifie que vous pouvez intercepter l'événement et le gérer avant qu'il n'atteigne l'élément qui a déclenché l'événement. Il suffit de passer « *true* » au troisième argument de la méthode `addEventListener` (capture) pour spécifier que la capture doit être utilisée.

```
1 <div id="parent">
2   <button id="enfant">Cliquez moi</button>
3 </div>
4
5 const parent = document.querySelector('#parent');
6 const enfant = document.querySelector('#enfant');
7
8 parent.addEventListener('click', function() {
9   console.log('Parent');
10 }, true);
11
12 enfant.addEventListener('click', function() {
13   console.log('Enfant');
14 });
```

[cf. ]

Dans cet exemple, si vous cliquez sur l'élément enfant, l'événement sera capturé au niveau de l'élément parent avant d'être propagé à l'élément enfant. Vous verrez donc « *Parent* » dans la console avant de voir « *Enfant* ».

En résumé, la capture d'événements permet de gérer les événements à un niveau supérieur de l'arborescence DOM, ce qui vous permet de les intercepter et de les gérer avant qu'ils n'atteignent l'élément cible.

## D. Empêcher la propagation d'événements

### Méthode

Comme nous venons de le voir, lorsqu'un événement est déclenché sur un élément, il se propage aux éléments parents de l'élément. Cela s'appelle la propagation d'événements. Vous pouvez empêcher la propagation d'un événement en appelant la méthode `stopPropagation()` sur l'objet `Event`.

```
1 <div id="parent">
2   <div id="enfant">Cliquez moi</div>
3 </div>

1 const parent = document.querySelector('#parent');
2 const enfant = document.querySelector('#enfant');
3
4 parent.addEventListener('click', function() {
5   console.log('Parent');
6 });
7
8 enfant.addEventListener('click', function(event) {
9   console.log('Enfant');
10  event.stopPropagation();
11 });
```

[cf.]

Dans ce cas, si vous cliquez sur l'élément enfant, l'événement ne sera pas propagé à l'élément parent. Vous verrez donc seulement « *Enfant* » dans la console.

Si nous enlevons la ligne `event.stopPropagation();`, alors nous aurons dans la console « *Enfant* », puis « *Parent* ».

## E. PreventDefault

### Exemple

Supposons que vous avez un formulaire d'inscription sur votre site web. Le formulaire comprend des champs pour le nom, l'adresse e-mail et le mot de passe, ainsi qu'un bouton « *S'inscrire* ».

Lorsqu'un utilisateur remplit le formulaire et clique sur le bouton « *S'inscrire* », vous voulez effectuer une validation de formulaire pour vous assurer que les données sont correctes avant de les envoyer au serveur. Pour cela, vous pouvez écouter l'événement de clic du bouton « *S'inscrire* » et empêcher le comportement par défaut du formulaire en utilisant `event.preventDefault()`.

```
1 const boutonInscription = document.querySelector('#bouton-inscription');
2 boutonInscription.addEventListener('click', function(event) {
3   event.preventDefault();
4   // code pour valider les données du formulaire
5 });
```

Ici, la fonction de gestionnaire d'événements sera appelée chaque fois que l'utilisateur cliquera sur le bouton « *S'inscrire* ». La méthode `event.preventDefault()` empêchera la soumission du formulaire par défaut.

Vous pouvez alors ajouter du code pour valider les données du formulaire. Par exemple, vous pouvez vérifier si l'adresse e-mail est valide ou si le mot de passe est suffisamment sécurisé. Si les données sont valides, vous pouvez les envoyer au serveur *via* une requête AJAX.

Vous pouvez également utiliser des événements pour fournir des commentaires en temps réel à l'utilisateur lorsqu'il remplit le formulaire. Par exemple, vous pouvez ajouter un événement d'entrée pour le champ d'adresse e-mail, qui déclenche une fonction de gestionnaire d'événements pour vérifier si l'adresse e-mail est valide ou non.

```
1 const champEmail = document.querySelector('#champ-email');
2 champEmail.addEventListener('input', function() {
3   // Vérifier si l'adresse e-mail est valide.
4 });
```

Dans ce cas, la fonction de gestionnaire d'événements sera appelée chaque fois que l'utilisateur saisira quelque chose dans le champ d'adresse e-mail. Vous pouvez alors ajouter du code pour vérifier si l'adresse e-mail est valide ou non et fournir un feedback en temps réel à l'utilisateur, par exemple en changeant la couleur du champ en fonction de sa validité.

## F. Exercice : Quiz

[solution n°2 p.18]

### Question 1

Comment fonctionne la capture d'événements en JavaScript ?

- ☐ Les événements sont capturés par les éléments parents avant d'être déclenchés sur l'élément cible
- ☐ Les événements sont capturés par les éléments enfants avant d'être déclenchés sur l'élément parent
- ☐ Les événements sont directement déclenchés sur l'élément cible

### Question 2

L'objet `Event` permet de récupérer des informations sur l'événement.

- ☐ Vrai
- ☐ Faux

### Question 3

Comment empêcher la propagation d'un événement en JavaScript ?

- ☐ En utilisant la méthode `event.preventDefault()`
- ☐ En utilisant la méthode `event.stopPropagation()`
- ☐ En utilisant la méthode `stopPropagation(event)`

### Question 4

Comment capturer un événement en utilisant la phase de capture en JavaScript ?

- ☐ En utilisant la méthode `element.addEventListener()` avec le troisième paramètre « *false* »
- ☐ En utilisant la méthode `element.attachEvent()`
- ☐ En utilisant la méthode `element.addEventListener()` avec le troisième paramètre « *true* »

### Question 5

Comment arrêter la propagation d'un événement pendant la phase de capture en JavaScript ?

- ☐ En utilisant la méthode `event.stopPropagation()` dans un gestionnaire d'événements capturant
- ☐ En utilisant la méthode `event.preventDefault()` dans un gestionnaire d'événements capturant
- ☐ En utilisant la méthode `event.stopImmediatePropagation()` dans un gestionnaire d'événements capturant

## IV. Essentiel

En résumé, les événements sont des actions qui se produisent dans une application web, comme le clic sur un bouton ou le chargement d'une page. Ils sont gérés à l'aide de gestionnaires d'événements, qui sont des fonctions qui sont appelées lorsqu'un événement se produit, et peuvent être déclenchés de plusieurs manières. L'objet `Event` est utilisé pour gérer nos événements et les variables liées à celui-ci. Nous avons parlé de propagation et de capture d'événement qui correspondent à la manière dont les événements se déplacent dans notre page web.

Pour finir, on peut dire que la gestion des événements est une partie fondamentale de la programmation en JavaScript. Comprendre la capture et la propagation d'événements vous permettra de maîtriser votre développement front, et particulièrement de mieux comprendre les bugs liés aux événements, qui sont récurrents et parfois difficiles à résoudre.

## V. Auto-évaluation

### A. Exercice

Dans cet exercice, nous allons créer une application de *to-do list* en utilisant la gestion des événements en JavaScript. Notre application doit permettre aux utilisateurs d'ajouter des tâches à la liste, de cocher les tâches terminées et de supprimer les tâches terminées.

Voici le code HTML de base pour notre application :

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>To-Do List</title>
6     <link rel="stylesheet" href="styles.css"/>
7   </head>
8   <body>
9     <h1>To-Do List</h1>
10    <div class="task-form">
11      <input type="text" id="new-task" placeholder="Add a new task">
12      <button id="add-task">Add Task</button>
13    </div>
14    <ul id="task-list"></ul>
15
16    <script src="script.js"></script>
17  </body>
18 </html>
```

Voici le JavaScript :

```
1 const newTaskInput = document.getElementById('new-task');
2 const addTaskButton = document.getElementById('add-task');
3 const taskList = document.getElementById('task-list');
```

Et enfin le CSS :

```
1 /* Style général de la page */
2 body {
3   font-family: Arial, sans-serif;
4   margin: auto;
5   padding: 0;
6   background-color: #f4f4f4;
7   max-width: 500px;
8 }
9
10 /* Style de l'en-tête */
11 header {
```

```
12 background-color: #333;
13 color: #fff;
14 padding: 15px;
15 }
16
17 header h1 {
18   margin: 0;
19   font-size: 36px;
20 }
21
22 /* Style de la liste de tâches */
23 #task-list {
24   list-style-type: none;
25   padding: 0;
26   margin: 0;
27 }
28
29 #task-list li {
30   background-color: #fff;
31   padding: 10px;
32   border-bottom: 1px solid #ccc;
33   display: flex;
34   justify-content: space-between;
35   align-items: center;
36   margin: 10px;
37 }
38
39 #task-list li:last-child {
40   border-bottom: none;
41 }
42
43 #task-list li .task-title {
44   flex-grow: 1;
45 }
46
47 #task-list li.completed {
48   background-color: #e17777;
49 }
50
51 /* Style du formulaire */
52 .task-form {
53   margin-top: 20px;
54   display: flex;
55   align-items: center;
56 }
57
58 .task-form input[type="text"] {
59   flex-grow: 1;
60   padding: 10px;
61   font-size: 18px;
62   border-radius: 5px;
63   border: none;
64 }
65
66 .task-form button {
67   background-color: #333;
68   color: #fff;
69   padding: 10px 20px;
```

```
70 font-size: 18px;
71 border-radius: 5px;
72 border: none;
73 cursor: pointer;
74 margin-left: 10px;
75 }
```

**Question 1**

[solution n°3 p.19]

Ajoutez le code pour gérer la création d'une tâche. Au clic sur le bouton « add task », il faut créer une tâche en HTML, et l'ajouter au div qui a pour id « *task-list* ».

**Question 2**

[solution n°4 p.19]

Ajoutez le code pour ajouter la classe « *completed* » à une tâche lorsqu'on la coche.

**B. Test****Exercice 1 : Quiz**

[solution n°5 p.19]

**Question 1**

Quelle méthode permet d'ajouter un écouteur d'événement à un élément HTML ?

- ☐ `addEventListener()`
- ☐ `createElement()`
- ☐ `setAttribute()`

**Question 2**

Comment peut-on empêcher la propagation d'un événement dans l'arbre DOM ?

- ☐ `event.preventDefault()`
- ☐ `event.stopPropagation()`
- ☐ `event.stopImmediatePropagation()`

**Question 3**

Comment peut-on créer un événement personnalisé en JavaScript ?

- ☐ En créant une instance de la classe `CustomEvent`
- ☐ `event.dispatch()`
- ☐ `event.trigger()`

**Question 4**

Quelle méthode peut-on utiliser pour retirer un écouteur d'événement d'un élément HTML ?

- ☐ `deleteElement()`
- ☐ `clearAttribute()`
- ☐ `removeEventListener()`

**Question 5**

Lorsqu'on crée un événement en JavaScript, il est immédiatement déclenché.

- ☐ Vrai
- ☐ Faux


## Solutions des exercices



**Exercice p. 7 Solution n°1****Question 1**

Quel est l'objet utilisé pour créer des événements personnalisés en JavaScript ?


- ☐ Event
- ☒ CustomEvent
- ☐ MouseEvent

 La réponse `CustomEvent` est correcte, car c'est l'objet utilisé pour créer des événements personnalisés en JavaScript. L'objet `CustomEvent` est un sous-type de l'objet `Event`, qui permet de définir des propriétés supplémentaires pour l'événement personnalisé.

**Question 2**

Comment ajoute-t-on un écouteur d'événement pour un événement personnalisé ?


- ☒ `addEventListener('nomEvenement', function() {})`
- ☐ `on('nomEvenement', function() {})`
- ☐ `attachEvent('nomEvenement', function() {})`

 La réponse `addEventListener('nomEvenement', function() {})` est correcte, car c'est la méthode utilisée pour ajouter un écouteur d'événement pour un événement personnalisé. La méthode `on()` n'est pas utilisée en JavaScript, et `attachEvent()` est obsolète.

**Question 3**

Comment déclenche-t-on un événement personnalisé ?


- ☐ `dispatch('nomEvenement')`
- ☐ `trigger('nomEvenement')`
- ☒ `dispatchEvent('nomEvenement')`

 `dispatchEvent` est la méthode utilisée pour déclencher un événement personnalisé. Les méthodes `dispatch()` et `trigger()` n'existent pas en JavaScript.

**Question 4**

Peut-on inclure des données avec un événement personnalisé ?

- ☐ Non, ce n'est pas possible
- ☒ Oui, en utilisant l'objet `CustomEvent`
- ☐ Oui, en utilisant l'objet `Event`

 `CustomEvent` est l'objet qui permet d'inclure des données avec un événement personnalisé. L'objet `Event` ne permet pas d'inclure des données supplémentaires.

**Question 5**

Quel est l'événement JavaScript qui est déclenché lorsqu'un utilisateur appuie sur une touche du clavier ?

- ☐ onscroll
- ☒ onkeydown
- ☐ onmouseover

**Q** L'événement « *onkeydown* » en JavaScript est déclenché lorsqu'un utilisateur appuie sur une touche du clavier. Cet événement peut être utilisé pour détecter les touches spécifiques que l'utilisateur a enfoncées et déclencher des actions en conséquence. Par exemple, si vous souhaitez effectuer une action lorsque l'utilisateur appuie sur la touche « Entrée », vous pouvez utiliser la méthode `addEventListener()` pour écouter l'événement « *onkeydown* », puis vérifier si la touche enfoncée est la touche « Entrée » en utilisant la propriété « *key* » de l'objet événement. En résumé, l'événement « *onkeydown* » est un événement clé pour la détection des touches du clavier en JavaScript.

## Exercice p. 12 Solution n°2

### Question 1

Comment fonctionne la capture d'événements en JavaScript ?

- ☒ Les événements sont capturés par les éléments parents avant d'être déclenchés sur l'élément cible
- ☐ Les événements sont capturés par les éléments enfants avant d'être déclenchés sur l'élément parent
- ☐ Les événements sont directement déclenchés sur l'élément cible

**Q** Lors de la phase de capture, l'événement est d'abord capturé par l'élément parent de l'élément cible, puis par les parents de cet élément. Finalement, l'événement atteint l'élément cible et est traité par les gestionnaires d'événements associés à cet élément.

### Question 2

L'objet `Event` permet de récupérer des informations sur l'événement.

- ☒ Vrai
- ☐ Faux

**Q** Grâce à l'objet `Event`, nous allons pouvoir récupérer des informations sur l'événement, notamment la cible (par exemple sur un clic, là où a cliqué l'utilisateur).

### Question 3

Comment empêcher la propagation d'un événement en JavaScript ?

- ☐ En utilisant la méthode `event.preventDefault()`
- ☒ En utilisant la méthode `event.stopPropagation()`
- ☐ En utilisant la méthode `stopPropagation(event)`

**Q** La méthode `event.stopPropagation()` empêche la propagation de l'événement à travers la phase de propagation. Si un gestionnaire d'événements appelle `event.stopPropagation()`, les gestionnaires d'événements associés aux parents de l'élément cible ne seront pas appelés.

### Question 4

Comment capturer un événement en utilisant la phase de capture en JavaScript ?

- ☐ En utilisant la méthode `element.addEventListener()` avec le troisième paramètre « *false* »
  - ☐ En utilisant la méthode `element.attachEvent()`
  - ☒ En utilisant la méthode `element.addEventListener()` avec le troisième paramètre « *true* »
- Q** Le troisième paramètre de la méthode `element.addEventListener()` spécifie si l'événement doit être capturé ou non pendant la phase de capture. En passant « *true* » comme troisième paramètre, vous pouvez capturer l'événement pendant la phase de capture.

### Question 5

Comment arrêter la propagation d'un événement pendant la phase de capture en JavaScript ?

- ☒ En utilisant la méthode `event.stopPropagation()` dans un gestionnaire d'événements capturant
  - ☐ En utilisant la méthode `event.preventDefault()` dans un gestionnaire d'événements capturant
  - ☐ En utilisant la méthode `event.stopImmediatePropagation()` dans un gestionnaire d'événements capturant
- Q** Que vous soyez en phase de capture ou non, l'arrêt de la propagation d'un événement fonctionne de la même manière.

### p. 15 Solution n°3

```
1 addTaskButton.addEventListener('click', function() {
2   const taskText = newTaskInput.value;
3   const taskItem = document.createElement('li');
4   const taskCheckbox = document.createElement('input');
5   const taskLabel = document.createElement('label');
6
7   taskCheckbox.type = 'checkbox';
8   taskLabel.innerText = taskText;
9
10  taskItem.appendChild(taskCheckbox);
11  taskItem.appendChild(taskLabel);
12  taskList.appendChild(taskItem);
13
14  newTaskInput.value = '';
15 });
```

[cf.]

### p. 15 Solution n°4


```
1 taskList.addEventListener('change', function(event) {
2   if (event.target.type === 'checkbox') {
3     const taskItem = event.target.parentElement;
4     taskItem.classList.toggle('completed');
5   }
6 });
```

[cf.]

### Exercice p. 15 Solution n°5


### Question 1

Quelle méthode permet d'ajouter un écouteur d'événement à un élément HTML ?

- ☒ `addEventListener()`
- ☐ `createElement()`
- ☐ `setAttribute()`
-  `addEventListener()` permet d'ajouter un écouteur d'événement à un élément HTML.


### Question 2

Comment peut-on empêcher la propagation d'un événement dans l'arbre DOM ?

- ☐ `event.preventDefault()`
- ☒ `event.stopPropagation()`
- ☐ `event.stopImmediatePropagation()`
-  `event.stopPropagation()` permet d'empêcher la propagation d'un événement dans l'arbre DOM.


### Question 3

Comment peut-on créer un événement personnalisé en JavaScript ?

- ☒ En créant une instance de la classe `CustomEvent`
- ☐ `event.dispatch()`
- ☐ `event.trigger()`
-  La classe `CustomEvent` permet de créer un événement personnalisé en JavaScript. Vous pourrez ensuite déclencher cet événement, et mettre des écouteurs sur cet événement.


### Question 4

Quelle méthode peut-on utiliser pour retirer un écouteur d'événement d'un élément HTML ?

- ☐ `deleteElement()`
- ☐ `clearAttribute()`
- ☒ `removeEventListener()`
-  `removeEventListener()` permet de retirer un écouteur d'événement d'un élément HTML.

### Question 5

Lorsqu'on crée un événement en JavaScript, il est immédiatement déclenché.

- ☐ Vrai
- ☒ Faux
-  Faux. Pour créer un événement, on utilise la méthode `createEvent`. Une fois que l'événement est créé, il faut le déclencher avec la méthode `dispatch`.