La programmation événementielle

Table des matières

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Pré-requis : Connaissance du HTML, du CSS et des bases de JavaScript

Contexte

La programmation événementielle est définie par l'interaction d'un utilisateur ou du programme lui-même avec les éléments ou objets de l'application. Ils communiquent en réponse à des événements qui peuvent être le clic sur un bouton, le passage de la souris sur une zone délimitée, une sélection dans une liste déroulante, ou encore un minuteur réglé toutes les trente secondes.

Tous les langages de programmation permettent de réaliser de la programmation événementielle :

- Java
- C++
- Python
- JavaScript

II. Introduction à la programmation événementielle en JavaScript

Objectifs

- Comprendre la mécanique d'un évènement en javaScript

Mise en situation

En JavaScript, pour réaliser de la programmation événementielle, nous serons amenés à manipuler différents objets, éléments, événements et différentes fonctions. Ces notions devront nous être familières pour la mise en place d'interactions entre l'utilisateur et notre application. Nous parlerons alors d'interface web réactive.

Méthode

Un événement est une action qui se produit dans le navigateur web, que celui-ci vous renvoie afin que vous puissiez y répondre.

Par exemple, lorsque les utilisateurs cliquent sur un bouton d'une page web, vous souhaiterez peut-être répondre à cet événement de 'click' en affichant une boîte de dialogue.

Chaque événement « click » va donc déclencher une fonction affichant une boîte de dialogue.

Le lien entre l'événement et la fonction déclenchée qui se fait par l'intermédiaire d'un gestionnaire d'événements est également appelé écouteur d'événements (event listener). Il écoute l'événement et s'exécute lorsque celui-ci se produit.

Supposons que vous ayez le code suivant :

```
<code>
let btn = document.querySelector('#btn');

function display() {
    alert('Je suis une boîte de dialogue qui s'ouvre suite à un clic');
}

```

```
btn.addEventListener('click',display);
<code>
```

Fonctionnement :

- Première ligne : on sélectionne l'élément du DOM que l'on souhaite écouter, celui qui va être à l'origine de l'événement, dans notre cas le bouton.
- Puis on crée une fonction nommée display, utilisant une méthode `alert()`afin d'afficher une boîte de dialogue
- Enfin on lie notre élément du DOM et la fonction à l'aide du gestionnaire d'événement `addEventListener()`, ce gestionnaire prend ici 2 paramètres, le premier le type d'évènement sur lequel il doit se déclencher, ici un click et le second paramètre, le nom de la fonction qu'il déclenche display.

Fonctionne aussi avec une fonction anonyme.

```
<code>
let btn = document.querySelector('#btn');

btn.addEventListener('click',function() {
    alert('Je suis une boîte de dialogue qui s'ouvre suite à un clic');
});
<code>
```

Syntaxe **À retenir**

En javascript la programmation événementielle est composé au minimum de trois éléments:

- La sélection d'un élément du DOM
- Une fonction
- Un gestionnaire d'événement pour lier l'élément du DOM et la fonction à un événement.

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°1 p.25]

En vous basant sur le code HTML correspondant au panier d'un site e-commerce, créez un code JavaScript qui vous permettra de récupérer l'élément panier et qui affiche une alerte lorsque l'on clique dessus.

Vous utiliserez la méthode `addEventListener()` vue dans le cours pour réaliser cette opération.

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Blog</title>
  <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
  <style>
```

1 https://repl.it/

```
    nav {
      width: 100%;
      height: 50px;
      display: flex;
      flex-direction: row;
      justify-content: space-between;
    }

    .logo {
      width: 20%
    }

    .shopping {
      width: 20%;
      display: flex;
      flex-direction: column;
      justify-content: center;
      text-align: right;
      font-size: 30px;
      margin-right: 10px;
    }

    .shopping:hover {
      cursor: pointer;
    }

  </style>
</head>
<body>
  <nav>
    <div class="logo"></div>
    <div id="shopping" class="shopping">
      <i class="fas fa-shopping-cart"></i>
    </div>
  </nav>
<script src="script.js"></script>
</body>
</html>
```

IV. Ajouter/Supprimer un événement au DOM

Objectifs

- Ajouter un événement à la liste des écouteurs d'événements du DOM
- Supprimer un événement de la liste des écouteurs d'événements du DOM

Mise en situation

Selon les besoins de notre application et les fonctionnalités que nous aurons à développer, nous devons ajouter ou supprimer des événements. Nous allons voir quelles méthodes JavaScript sont à votre disposition pour manipuler des événements.

Méthode

Pour créer un événement sur un élément du DOM, il faut commencer par cibler cet élément. Comme nous l'avons vu, des méthodes sont disponibles sur l'objet `document`, appelées des **sélecteurs**.

- `document.getElementById('title')` : renvoie l'élément de DOM qui possède l'id `#title`. Pour rappel, un identifiant doit être unique dans la page. Dans le cas de multiples éléments qui auraient le même id, le navigateur affichera un warning dans la console et `getElementById` aura un résultat imprédictible.
- `document.getElementsByName('email-input')` : renvoie un tableau d'éléments de DOM qui possèdent un attribut `name email-input`. Comme le laisse supposer le nom de la méthode avec le mot "`element`" au pluriel, dans le cas où un seul élément est trouvé par le sélecteur, cet élément sera tout de même inclus dans un tableau.
- `document.getElementsByTagName('p')` : renvoie un tableau d'éléments de DOM qui correspondent à la balise `p`.
- `document.getElementsByClassName('navbar')` : renvoie un tableau d'éléments de DOM qui correspondent à la classe CSS `navbar`.

Ces 4 méthodes sont performantes mais limitées avec des pages ou applications complexes. Elles se révèlent peu souples car nécessitent de faire appel à des boucles, diverses opérations algorithmiques et des filtres sur les résultats obtenus pour pouvoir obtenir une liste d'éléments.

Avec deux nouvelles fonctions introduites par l'API Selectors, la syntaxe permet de faire appel aux **sélecteurs CSS**. On applique en général ces deux méthodes à partir de la racine document.

- `document.querySelector('h3.subtitle')` : renvoie l'élément DOM trouvé qui correspond au sélecteur CSS `h3.subtitle`. Ce sélecteur renverra `null` si aucune valeur n'est trouvée. Dans le cas où le sélecteur trouverait plusieurs résultats, le premier résultat trouvé sera renvoyé.

Méthode la plus recommandée

- `document.querySelector('h3.subtitle')` : renvoie le premier élément DOM trouvé qui correspond au sélecteur CSS h3.subtitle. Ce sélecteur renverra null si aucune valeur n'est trouvée. Dans le cas où le sélecteur trouverait plusieurs résultats, le premier résultat trouvé sera renvoyé.

Méthode la moins recommandée:

- `querySelectorAll('h3.subtitle')` : retourne tous les éléments satisfaisant au sélecteur `h3.subtitle`, dans l'ordre dans lesquels ils apparaissent dans l'arbre du document.

Le type de retour est une `NodeList`, qui est vide si rien n'est trouvé.

Fondamental

Les sélecteurs vont nous permettre de sélectionner un ou plusieurs éléments du DOM. Il doivent être utilisés sur l'objet JavaScript `document`, qui est le point d'entrée de notre page web.

Exemple

```html
<button id="button">Bouton</button>
<div name="text">Une zone de texte</div>
<span>Balise span<span>
<h1 class="title">Titre</h1>
<p class="class-css">classe css</p>
```

```javascript
// Sélection de l'élément ayant pour attribut id="button"
var element = document.getElementById('button');

// Sélection de l'élément ayant pour attribut name="text"
var element = document.getElementsByName('text');

```

```
//Sélection de l'élément étant une balise HTML span
var element = document.getElementsByTagName('span');

//Sélection de l'élément ayant pour attribut class="title"
var element = document.getElementsByClassName('title');

//Sélection de l'ément ayant pour classe css .class-css
var element = document.querySelector('.class-css');
```

Complément

Nous pouvons également enchaîner les sélecteurs :

```
<div id="myDiv">
  <p>un paragraphe</p>
  <span>un span</span>
</div>
```

```
document.getElementById('myDiv').getElementsByTagName('p')
```

AddEventListener

Une fois notre élément de DOM sélectionné, nous pouvons appeler la méthode `addEventListener()` pour lui lier un événement. Comme nous l'avons vu, cette méthode prend généralement deux paramètres : le type d'événement et une fonction, qui sera appelée au moment du déclenchement de cet événement.

Exemple

```
const button = document.getElementById('button')

button.addEventListener('click', maFonction)
```

Paramètres obligatoires :

- `type` : une chaîne de caractères représentant le type d'événement à écouter (`click`, `mouseenter`, `keyup`, etc.).
- `listener` : notre fonction JavaScript qui sera exécutée lors de l'émission de l'événement.

Paramètres facultatifs :

- `options` : un objet qui spécifie les caractéristiques de l'écouteur d'événements.
  - `capture` : un booléen indiquant le sens de propagation de l'événement : parent puis élément, ou élément puis parent.
  - `once` : un booléen indiquant si l'événement doit être supprimé après son appel, si la valeur est *true* alors il ne sera émis qu'une seule fois.
  - `passive` : un booléen indiquant que l'événement ne sera jamais émis.

RemoveEventListener

Nous pouvons supprimer un événement en invoquant la fonction `removeEventListener()`.

Supprimer un `eventListener` quand on en a plus besoin est une excellente pratique, car il peut être coûteux pour le navigateur qui doit observer un nœud de DOM supplémentaire.

Nous devinerons aisément que, plus il y a d'éléments à charger dans le DOM, plus notre page web sera lente à s'afficher.

Exemple

```
const button = document.getElementById('button')

button.removeEventListener('click', maFonction)
```

À noter que `removeEventListener` prend les mêmes paramètres que la fonction d'ajout. Pour réaliser la suppression, les paramètres `type` et `listener` doivent obligatoirement être identiques à la création et à la suppression.

Exemple

```
element.addEventListener("click", clickMe);



element.removeEventListener("click", myFunction);    // Ne fonctionnera pas car le paramètre listener n'est pas le même
element.removeEventListener("mousedown", clickMe);   // Ne fonctionnera pas car le paramètre type n'est pas le même
element.removeEventListener("click", clickMe);       // Fonctionnera
```

Complément **La délégation d'événement**

La délégation d'événement est une pratique consistant à lier un événement non pas sur l'élément ciblé, mais sur l'un de ses ancêtres. La délégation est très utile dans le cas où nous souhaitons lier un événement sur un élément de DOM qui n'existe pas encore au moment du chargement de la page, comme par exemple un bouton qui serait ajouté dynamiquement au DOM par une action de script. Créer un événement sur cet élément à l'aide de `addEventListener` au chargement de la page ne fonctionnera pas. Il est alors nécessaire de modifier la syntaxe de `addEventListener` pour créer une délégation.

Exemple **Considérez le code HTML ci-dessous**

```
<body>
    <div id="event">
        <p>délégation des événements</p>
    </div>
</body>
```

Si nous plaçons un événement `click` sur la `div`, le clic sur le texte du paragraphe `<p>` déclenchera l'événement, bien que ce dernier cache la `div`. Ceci est rendu possible grâce à la délégation d'événement.

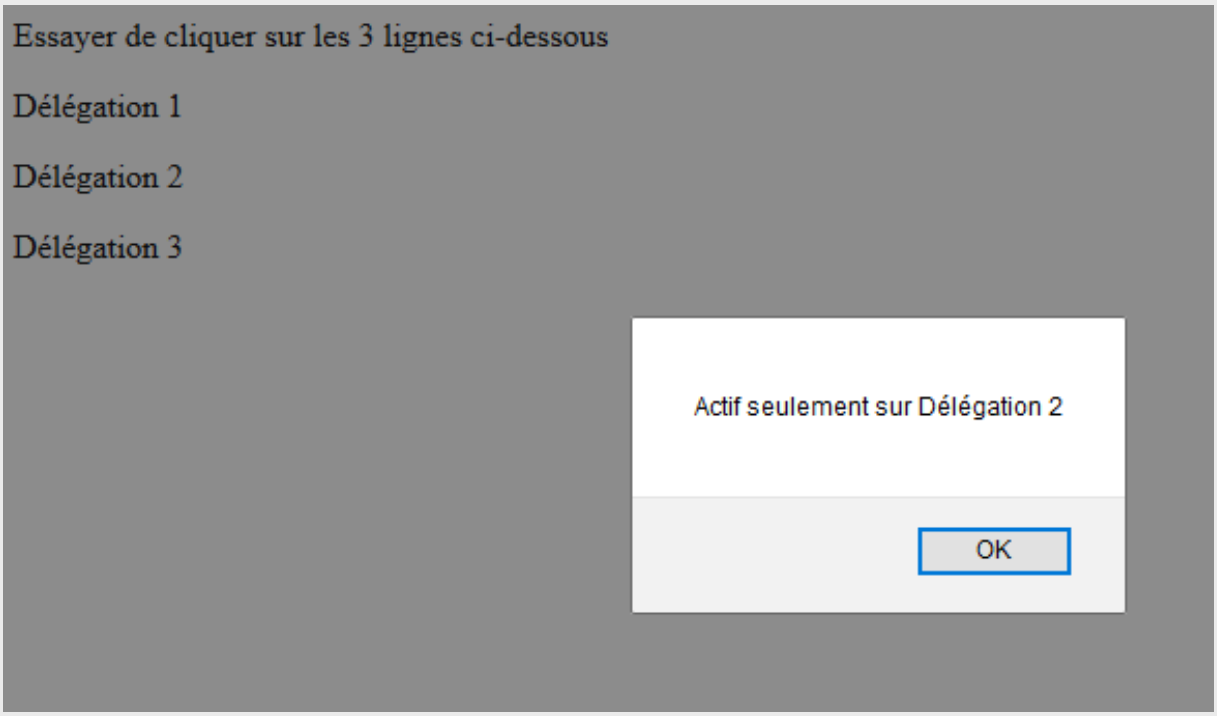
Imaginons que l'on souhaite définir un événement en cliquant sur l'un des paragraphes de la `div`. Au lieu de lier autant d'événements que de `<p>`, nous pourrons n'en définir qu'un seul et le placer directement sur la `div`.

La propriété *target* obtient l'élément sur lequel l'événement s'est produit à l'origine.

```
<body>
    <div id="event">
        <p class="no-event">délégation des événements 1</p>
        <p>délégation des événements 2</p>
        <p class="no-event">délégation des événements 3</p>
    </div>
</body>
```

```
var element = document.getElementById('event')

element.addEventListener('click', function(e) {
  var initElem = e.target;
    if(initElem.className == 'no-event'){
        return;
```

```
    }
    alert("Actif seulement sur Délégation 2")
})
```



Syntaxe **À retenir**

- La méthode d'ajout d'un écouteur d'événements est la signature de celui-ci. Pour le supprimer, nous devons obligatoirement renseigner les mêmes paramètres.

```
element.addEventListener('click', functionEvent, true)
element.removeEventListener('click', functionEvent, true)
```

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

1 https://repl.it/

Question

[solution n°2 p.26]

À présent que nous avons créé le bouton de connexion de notre site e-commerce, nous allons créer une nouvelle page en gardant le menu de navigation HTML de l'exercice 1. Cette page contiendra deux boutons permettant d'ajouter ou d'enlever un article.

- Ajoutez un bouton **Ajouter un article** et un bouton **Supprimer un article**.
- Le bouton **Ajouter** aura un fond vert et le bouton **Supprimer**, un fond rouge.
- S'il y a 0 article ajouté, le panier ne sera pas visible. S'il y a au moins 1 article ajouté, le panier sera visible. Le nombre d'articles sera toujours visible.
- Au clic sur le panier, on supprimera les événements d'ajout et de suppression.

Vous vous baserez sur le code HTML ci-dessous. Le menu a été modifié par rapport à l'exercice 1 afin d'ajouter le nombre d'articles. La modification du texte d'un élément HTML se fait via la méthode `element.textContent`. https://developer.mozilla.org/fr/docs/Web/API/Node/textContent

Pour afficher le panier, vous utiliserez la méthode `element.style.display = 'block'`.

Pour cacher le panier, vous utiliserez la méthode `element.style.display = 'none'`.

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Blog</title>
    <link rel="stylesheet"  href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css" integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p" crossorigin="anonymous"/>
    <style>
      nav {
        width: 100%;
        height: 50px;
        display: flex;
        flex-direction: row;
        justify-content: space-between;
      }

      .shopping {
        width: auto;
        display: flex;
        flex-direction: row;
        align-items: center;
        justify-content: center;
        text-align: right;
        margin-right: 10px;
      }

      .shop {
        width: auto;
        text-align: right;
        font-size: 30px;
        margin-right: 10px;
        padding-top: 10px;
      }

      #shop {
        display: none;
      }

      #shop:hover {
```

```
        cursor: pointer;
      }

      .article {
        width: auto;
        text-align: right;
        margin: 0 10px;
      }
    </style>
  </head>
  <body>
    <nav>
      <div class="logo"></div>
      <div class="shopping">
        <div class="article">
          <p id="number-article">0</p>
        </div>
        <div class="shop">
          <i id="shop" class="fas fa-shopping-cart"></i>
        </div>
      </div>
    </nav>
    <script src="script.js"></script>
  </body>
</html>

```

VI. Les événements

Objectifs

- Comprendre l'objet `Event`
- Connaître les types d'événements existants

Mise en situation

Nous avons vu qu'un événement se fixe sur un élément de DOM et écoute une action utilisateur bien précise pour exécuter du code. Nous allons maintenant voir les différents types d'événements mis à notre disposition par JavaScript et comment les utiliser.

Complément

Nous pouvons consulter l'ensemble des propriétés et méthodes dans la documentation officielle[1].

Les types d'événements

Les événements globaux ne sont pas directement liés à une interaction avec l'utilisateur, mais plutôt à une action de script.

L'événement `load`, par exemple, est automatiquement déclenché par le script quand toutes les ressources de la page sont chargées (images, feuille de style CSS, script JavaScript, etc.).

1 https://developer.mozilla.org/fr/docs/Web/API/Event

Exemple

```
<script>
  window.addEventListener("load", function(event) {
    console.log("Les ressources de la page ont été chargées !");
  });
</script>
```

Complément

Voici quelques-uns de ces événements globaux :

- `offline` : automatiquement déclenché quand le navigateur a perdu sa connexion à Internet. Il peut servir à avertir l'utilisateur ou effectuer une action de stockage local, par exemple.
- `abort` : déclenché quand le chargement d'une ressource a été interrompu par l'utilisateur. Il peut servir à lui indiquer quelle ressource est manquante et que la page web pourra se comporter de manière inattendue.
- `ended` : déclenché à la fin de la lecture d'un média ou lorsque la source du média ne permet plus l'envoi des données. Il peut servir à indiquer à l'utilisateur la raison de l'arrêt du média.
- `success` : déclenché quand une requête HTTP s'est déroulée correctement. Il peut servir à afficher un message à l'utilisateur pour lui indiquer le résultat de la requête.

Les événements souris permettent d'écouter un ensemble d'interactions que l'utilisateur peut faire avec sa souris. Comme nous l'avons vu précédemment, l'événement `click` est déclenché au moment de cliquer sur un élément du DOM.

Exemple

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

function evenement() {
  alert('Vous avez cliqué sur le bouton');
}

button.addEventListener('click', evenement);
```

Complément

Voici quelques-uns de ces événements souris :

- `click` : déclenché quand l'utilisateur clique sur un élément. Il peut servir à demander une confirmation de l'action à effectuer (exemple : *Êtes-vous sûr de vouloir supprimer cet item ?*).
- `mouseenter` : déclenché quand l'utilisateur pointe la souris sur un élément.
- `mouseleave` : déclenché quand l'utilisateur déplace le pointeur de la souris en dehors d'un élément.
- `mouseup` : déclenché quand l'utilisateur relâche un bouton de la souris lorsqu'il se trouve sur un élément.

Attention **Mauvaises pratiques**

Évitez autant que possible l'utilisation des événements `mouseover` et `mouseout`. Ces événements seront reçus par tous les descendants de l'élément auquel ils sont liés. Ils seront donc déclenchés autant de fois qu'il y a de descendants. Nous utiliserons plutôt `mouseenter` ou `mouseleave` qui, eux, ne seront déclenchés que pour l'élément auquel ils sont associés.

Les événements claviers sont liés à toute utilisation du clavier par l'utilisateur. Par exemple, avec l'événement `keydown`, le code ci-dessous sera déclenché lorsque la touche du clavier sera enfoncée.

Exemple

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

function keyboard() {
  alert('Vous venez d\'appuyer sur une touche du clavier');
}

key.addEventListener('keydown', keyboard);
```

Complément

Voici quelques-uns de ces événements :

- `keyup` : cet événement est déclenché quand l'utilisateur relâche une touche du clavier. Il peut servir à contrôler une saisie utilisateur en cours.
- `keydown` : cet événement est déclenché quand l'utilisateur maintient enfoncée une touche du clavier. Il peut servir à interdire l'utilisation d'une touche en temps réel.

Attention **Mauvaises pratiques**

L'utilisation de l'événement `keypress` est à proscrire, car il ne fonctionne pas pour toutes les touches du clavier (Alt, Ctrl, Shift, par exemple). Il est indiqué comme déprécié dans la documentation et ne sera bientôt plus supporté par les navigateurs. Préférez l'utilisation des événements `keydown` ou `keyup`.

Les événements de formulaires sont, par définition, liés à toutes interactions avec un élément de formulaire HTML. L'événement `submit` est, par exemple, déclenché quand un élément `<form>` est soumis, que ce soit au clic sur un élément `<button>` ou `<submit>`, ou même par une action de script.

Exemple

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

function form() {
  alert('Vous êtes sur le point de soumettre le formulaire');
}

form.addEventListener('submit', form);
```

L'objet Event

L'objet Event représente un événement JavaScript. Il contient un ensemble de propriétés et de méthodes communes à tous les événements. Nous pourrons les manipuler à notre guise pour modifier les caractéristiques de notre événement.

Lorsque l'événement se produit, le navigateur Web transmet un objet event au gestionnaire d'événements :

```
<code>
let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {
    console.log(event.type); //retourne 'click'
});
</code>
```

Notez que l'objet event n'est accessible qu'à l'intérieur du gestionnaire d'événements. Une fois que tous les gestionnaires d'événements ont été exécutés, l'objet événement est automatiquement détruit.

En voici quelques-unes des plus utilisées :

Propriétés :

- Event.target : cible de l'événement
- Event.defaultPrevented : indique si le comportement par défaut de l'interaction doit être annulé (par exemple, la validation d'un formulaire, la redirection vers une autre page pour un lien)
- Event.bubbles : indique le sens de propagation de l'événement

Méthodes :

- Event.createEvent() : crée un nouvel l'événement.
- Event.preventDefault() : annule le comportement par défaut de l'événement.
- Event.stopPropagation() : arrête la propagation de l'événement.

Exemple

Pour empêcher le comportement par défaut d'un événement, vous utilisez la méthode preventDefault().Par exemple, lorsque vous cliquez sur un lien, le navigateur vous redirige vers l'URL spécifiée dans l' attribut href :

```
<code>
<a href="https://www.google.com/">Google</a>
</code>
```

Cependant, vous pouvez empêcher ce comportement en utilisant la méthode `preventDefault()` de l'objet event.

```
<code>

let link = document.querySelector('a');

link.addEventListener('click',function(event) {
    console.log('clicked');
    event.preventDefault();
});
</code>
```

Syntaxe **À retenir**

Comme pour la manipulation de notre page web, nous pourrons jouer avec nos événements en utilisant les propriétés et méthodes de l'objet `Event`.

- Propriétés
  - `event.target` : représente l'élément qui a déclenché l'événement. Elle permet de manipuler l'élément en question.
  - `event.bubbles` : permet de définir si un événement se propage ou non. Nous pouvons nous en servir pour autoriser la propagation d'un événement.

- Méthodes
  - `event.stopPropagation()` : permet d'interrompre la propagation de l'événement à travers le DOM. Elle peut servir à éviter de déclencher des événements plusieurs fois au cours de l'exécution.
  - `event.preventDefault()` : permet d'annuler l'action par défaut d'un événement. Elle peut servir, par exemple, à annuler l'effet du clic sur lien.

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Vous aurez aussi besoin de la méthode `element.style()` pour changer la couleur du bouton : https://developer.mozilla.org/fr/docs/Web/API/HTMLElement/style.

Question

[solution n°3 p.28]

Pour les besoins de notre site e-commerce, nous avons besoin de créer un bouton de connexion. En utilisant uniquement le JavaScript :

- Créez un bouton HTML **Connexion** avec un fond vert et une couleur de texte blanche.
- Au clic, affichez un message dans une popup : *Vous avez cliqué sur le bouton de connexion*.
- Au passage de la souris, changez la couleur du bouton en rouge.
- Lorsque la souris sort du bouton, réinitialisez la couleur d'origine du bouton.

VIII. La propagation des événements

Objectif

- Comprendre et maîtriser la propagation des événements

Capture et bouillonnement

En JavaScript les événements sont soumis à deux phases : la phase de **capture** et la phase de **bouillonnement** (*bubbling*).

Lorsqu'un événement est déclenché, JavaScript parcourt l'ensemble du code en partant de l'ancêtre le plus haut (html) et descend jusqu'à l'élément ayant déclenché l'événement. On appelle ça la phase de capture.

Une fois la phase de capture finie, JavaScript parcourt le code en sens inverse, et ce n'est qu'à ce moment qu'il déclenche l'événement lorsqu'il le croise. On appelle cette action la phase de bouillonnement.

1 https://repl.it/

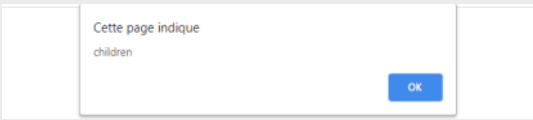
Exemple

Considérons le code suivant :

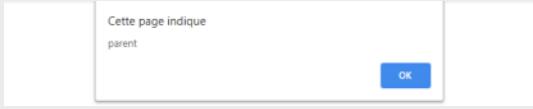
```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>propagation</title>
</head>

  <body>
    <div id="parent">
      <p>parent</p>
      <div id="children">
        <p>enfant</p>
      </div>
    </div>

  </body>

  <script>
    const parent = document.getElementById('parent');
    const children = document.getElementById('children');

    parent.addEventListener('click', () => {
      alert('parent')
    })

    children.addEventListener('click', () => {
      alert('children')
    })

  </script>
</html>

```

Nous avons relié un événement `click` à `parent` et à `children`. Que se passe t-il lorsque l'on `click` sur `children` ?

La première alerte qui s'affiche et celle provoquée par `children`. La phase de **capture** est descendue jusqu'à l'élément écoutant l'événement. Au début de la phase de **bouillonnement**, JavaScript a exécuté l'événement.



Une deuxième alerte s'affiche. Durant la phase de **bouillonnement**, JavaScript a exécuté l'événement écouté par le `parent`.



Modifier le comportement d'un événement

Il existe deux méthodes pour modifier ce comportement :

- La première appartenant à l'objet `Event` : `Event.stopPropagation()`,
- La deuxième consistant à modifier le troisième paramètre optionnel de la méthode `document.addEventListener()`.

Méthode **Event.stopPropagation()**

Cette méthode vient se positionner à l'intérieur de la fonction de `callback` de l'élément écouteur. La méthode `stopPropagation()` va annuler la phase de bouillonnement à partir du moment où le code rencontrera l'instruction. Ainsi, l'événement ne remontera pas jusqu'à la div `parent`.

Exemple

```
const parent = document.getElementById('parent');
const children = document.getElementById('children');

parent.addEventListener('click', () => {
   alert('parent')
})

// event en parametre de la fonction de callback représente l'évènement.
children.addEventListener('click', (event) => {
   event.stopPropagation();
   alert('children')
})
```

Méthode **addEventListener('event', callback, capture?)**

Cette solution consiste à modifier le paramètre optionnel de la méthode `addEventListener()` et de la passer à *true*. Dans ce cas, le code exécutera l'événement durant la phase de capture et annulera la phase de bouillonnement.

Exemple

```
const parent = document.getElementById('parent');
const children = document.getElementById('children');

parent.addEventListener('click', () => {
   alert('parent')
}, true)

children.addEventListener('click', () => {
   alert('children')
}, true)
```

Dans le cas présent, on ignore la phase de bouillonnement et les événements sont exécutés pendant la phase de capture. Ainsi, en cliquant sur `children`, l'alerte `parent` s'affichera en premier, puis l'alerte `children` en second. Le comportement est inversé.

Syntaxe **À retenir**

- Il existe deux phases lorsqu'un événement est déclenché : la phase de capture et la phase de bouillonnement.
- La phase de capture part de l'ancêtre le plus haut vers l'élément écouteur.
- La phase de bouillonnement fait le chemin en sens inverse et exécute les événements.
- `stopPropagation()` permet de stopper la phase de bouillonnement.
- Le troisième paramètre de la méthode `addEventListener()` délègue l'exécution des événements à la phase de capture et annule la phase de bouillonnement.

IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

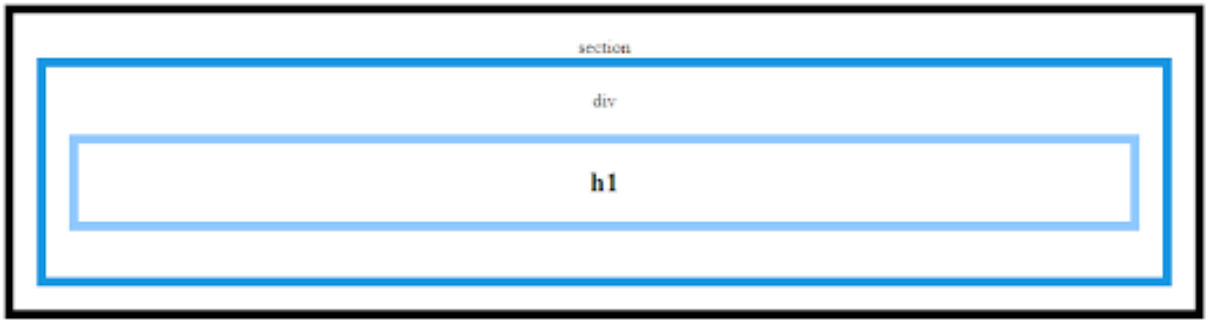
Question

[solution n°4 p.29]

On vous donne le code html de départ suivant :

```
<code>
<section>
  section
<div>
      div
      <h1>h1</h1>

</div>
</section>
<style>
.border-padding, h1, div, section {
  border: 8px solid transparent;
  padding: 20px;
  text-align: center;
}

section {
  border-color: black;
}

div {
  border-color: #1b97e1;
}

h1 {
  border-color: rgba(0, 133, 254, 0.44);
}

</style>

<code>
```

1 https://repl.it/



Réaliser le code JavaScript permettant d'afficher dans une fenêtre de dialogue le code JavaScript permettant d'afficher le tagName de chaque section indépendamment. Un clic doit ouvrir une seule boîte de dialogue.

Plusieurs réponses possibles.

X. Les événements personnalisés

Objectif

- Apprendre à créer des événements personnalisés

Mise en situation

Il se peut, lors de développement futurs, que les événements natifs de JavaScript ne puissent subvenir à nos besoins. Heureusement, le langage nous laisse la possibilité de créer nos propres événements.

Méthode

Deux nouvelles étapes viennent s'ajouter lors de la création d'un événement personnalisé : la création de l'événement en elle-même, et le déclenchement de ce dernier quand bon nous semblera. Cela pourrait être au chargement de la page, ou pourquoi pas lors de l'émission d'un autre événement.

Création avec le constructeur `Event` :

```
const myEvent = new Event('personnalEvent');
```

Déclenchement programmatique de l'événement :

```
element.dispatchEvent(myEvent);
```

Exemple

Nous pouvons également ajouter des données personnalisées avec le paramètre `detail` du constructeur `CustomEvent()` :

```
const event = new CustomEvent('personnalEvent', { detail : { 'myData': 'mon événement perso' } })
const element = document.getElementById('navbar')

element.addEventListener('personnalEvent', displayMe)

element.dispatchEvent(event)

function displayMe(e) {
  console.log(e.detail.myData)
}
```

Si vous souhaitez tester ce code, il est important de vous rappeler de créer le html qui va avec le script.

Rappel

De la même manière que pour les événements natifs, c'est une bonne pratique de supprimer nos événements personnalisés quand nous n'en n'avons plus l'utilité.

Syntaxe **À retenir**

- Nous pouvons créer des événements personnalisés de deux manières différentes :
  - Avec le constructeur `Event` :

```
new Event('monEvenement')
```

- Avec le constructeur `CustomEvent` :

```
new CustomEvent('monEvenementCustom', { detail : { 'tag': 'événement custom' } })
```

XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°5 p.29]

Dans cet exercice on va chercher à lier l'action de deux fonctions grâce à un évènement personnalisé.

Ci-dessous se trouve le code d'une page Html, qui dispose d'une méthode highlight().

Cette méthode est appelée au chargement de la page et permet de changer la couleur du background de notre div.

Vous devez compléter la fonction highlight() pour qu'une fois appelée elle crée un évènement personnalisé nommé backgroundYellow.

Puis vous devez mettre en place l'écoute de cet évènement sur la div pour qu'une fois celui ci réalisé il déclenche la fonction addBorder().

```
<code>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript Custom Event</title>
</head>
<body>
    <div class="note">JS Custom Event</div>
    <script>
        function highlight(elem) {
            const bgColor = 'yellow';
            elem.style.backgroundColor = bgColor;

            // create the event


            // dispatch the event
```

1 https://repl.it/

```
20
21          }
22
23          // Select the div element
24          let div = document.querySelector('.note');
25
26          // Add border style
27          function addBorder(elem) {
28              elem.style.border = "solid 1px red";
29          }
30
31          // Listen to the backgroundYellow event
32
33
34          // highlight div element
35          highlight(div);
36      </script>
37  </body>
38  </html>
39
40
41  </code>
```

XII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.30]

Exercice

Quel objet permet de manipuler les événements du DOM ?

- ○ Target
- ○ Event

Exercice

Quelle méthode permet d'annuler le comportement par défaut d'un événement ?

- ○ preventDefault()
- ○ stopPropagation()
- ○ initEvent()

Exercice

Cochez les bonnes pratiques de la programmation événementielle en JavaScript.

- ☐ Ne surtout pas supprimer ses événements
- ☐ Limiter le déclenchement de certains événements
- ☐ Vérifier qu'un événement soit compatible sur tous les navigateurs
- ☐ Utiliser la délégation d'événement

Exercice

Quel(s) événement(s) n'existe(nt) pas ?

- [ ] keyup
- [ ] mousepress
- [ ] stop
- [ ] focus
- [ ] input

Exercice

Sélectionnez les éléments HTML `span`.

- ○ const span = document.getElementById('span')
- ○ const span = document.getElementsByName('span')
- ○ const span = document.getElementsByTagName('span')

Exercice

Ajoutez un écouteur d'événement `click`.

- ○ span.createEventListener('click', myFunction)
- ○ span.addEventListener('click')
- ○ span.addEventListener('click', myFunction)
- ○ span.addEventListener(myFunction, 'click')

Exercice

Écrivez le code JavaScript permettant de supprimer l'écouteur d'événement créé à la question précédente.

Exercice

Affichez dans la console la cible de l'événement.

- ○ function myFunction() {
  console.log(e.target)
  }
- ○ function myFunction(e) {
  console.log(e.target)
  }
- ○ function myFunction() {
  console.log(e.cible)
  }
- ○ function myFunction(e) {
  alert(e.target)
  }

Exercice

Créez l'événement `monEvenement`.

- ☐ const event = new Event('monEvenement');
- ☐ const event = new CustomEvent('monEvenement', { detail : { 'color': 'green' })
- ☐ const event = monEvenement(Event);

B. Exercice : Défi

Pour cet exercice, vous allez simuler le comportement d'un *chat*.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°7 p.32]

Vous allez devoir créer une page HTML comportant :

- un formulaire pour le premier participant avec les champs **pseudo** et **message**, ainsi qu'un bouton **envoyer**.
- un formulaire identique pour le second participant.
- une zone où les messages seront affichés.

Un script JavaScript permettant :

- d'afficher les messages dans l'ordre de la saisie dans la zone prévue à cet effet.
- de rafraîchir l'affichage de la zone de texte.
- de bloquer la possibilité à l'un des deux participants d'envoyer des messages.

Vous pouvez varier les types d'événements à utiliser.

Solutions des exercices

1 https://repl.it/

p. 4 **Solution n°1**

```
const shop = document.getElementById('shopping');

const goToShop = () => {
   alert('Votre liste d\'achat');
}

shop.addEventListener('click', goToShop);
```

Remarque

Il est très probable que vous rencontriez une autre façon de créer des événements via des attributs directement dans le HTML (`onclick`, `onmouseover`, etc.). Ceci n'est cependant pas une méthode conseillée, car, bien qu'elle paraisse plus facile à utiliser, elle complique la compréhension du code sur le long terme.

Voici à quoi ressemblerait notre code si nous avions utilisé cette méthode :

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Blog</title>
  <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
  <style>
    nav {
      width: 100%;
      height: 50px;
      display: flex;
      flex-direction: row;
      justify-content: space-between;
    }

    .logo {
      width: 20%
    }

    .shopping {
      width: 20%;
      display: flex;
      flex-direction: column;
      justify-content: center;
      text-align: right;
      font-size: 30px;
      margin-right: 10px;
    }

    .shopping:hover {
      cursor: pointer;
    }

  </style>
</head>
<body>
  <nav>
    <div class="logo"></div>
```

```
    <div id="shopping" class="shopping">
      <!--ajout de l'évènement click -->
      <i class="fas fa-shopping-cart" onclick="goToShop()"></i>
    </div>
  </nav>
  <script src="script.js"></script>
</body>
</html>
```

```
const goToShop = () => {
   alert('Votre liste d\'achat');
}
```

p. 10 **Solution n°2**

Tout d'abord, ajoutez les deux boutons et donnez-leur une couleur de fond :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Blog</title>
    <link rel="stylesheet"  href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css"
integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKxVfELGroGkvsg+p"
crossorigin="anonymous"/>
    <style>
      nav {
        width: 100%;
        height: 50px;
        display: flex;
        flex-direction: row;
        justify-content: space-between;
      }

      .shopping {
        width: auto;
        display: flex;
        flex-direction: row;
        align-items: center;
        justify-content: center;
        text-align: right;
        margin-right: 10px;
      }

      .shop {
        width: auto;
        text-align: right;
        font-size: 30px;
        margin-right: 10px;
        padding-top: 10px;
      }

      #shop {
        display: none;
      }

      #shop:hover {
        cursor: pointer;
```

```
      }

      .article {
        width: auto;
        text-align: right;
        margin: 0 10px;
      }

      #add {
        background: green;
      }

      #delete {
        background: red;
      }
    </style>
  </head>
  <body>
    <nav>
      <div class="logo"></div>
      <div class="shopping">
        <div class="actions">
          <button id="add">Ajouter un article</button>
          <button id="delete">Supprimer un article</button>
        </div>
        <div class="article">
          <p id="number-article">0</p>
        </div>
        <div class="shop">
          <i id="shop" class="fas fa-shopping-cart"></i>
        </div>
      </div>
    </nav>
    <script src="script.js"></script>
  </body>
</html>

```

Puis ajouter le code JavaScript dans le fichier `script.js` :

```
let articleNumber = 0

const shop = document.getElementById('shop')
const btnAddArticle = document.getElementById('add')
const btnDeleteArticle = document.getElementById('delete')
const numberArticle = document.getElementById('number-article')

const addArticle = () => {
  articleNumber++
  numberArticle.textContent = articleNumber
  if (articleNumber > 0) {
    shop.style.display = "block"
  }
}

const deleteArticle = () => {
  if (articleNumber > 0) {
    articleNumber--
    numberArticle.textContent = articleNumber
```

```
    if (articleNumber === 0) {
      shop.style.display = "none"
    }
  }
}

const goToShop = () => {
  btnAddArticle.removeEventListener('click', addArticle)
  btnDeleteArticle.removeEventListener('click', deleteArticle)
}

shop.addEventListener('click', goToShop)
btnAddArticle.addEventListener('click', addArticle)
btnDeleteArticle.addEventListener('click', deleteArticle)
```

p. 15 Solution n°3

HTML :

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Blog</title>

  <style>
    #connexion {
      background: green;
      color: white;
    }
  </style>
</head>

  <body>
    <button id="connexion" type="button">Connexion</button>
<script src="script.js"></script>
  </body>
</html>
```

JavaScript :

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

const getMessage = () => {
   alert('Vous avez cliqué sur le bouton de connexion');
}

const setColorGreen = () => {
   connexion.style.background = 'green'
}

const setColorRed = () => {
   connexion.style.background = 'red'
}

connexion.addEventListener('click', getMessage);
connexion.addEventListener('mouseenter', setColorRed);
```

```
connexion.addEventListener('mouseleave', setColorGreen);
```

Remarque

Pour les besoins de l'exercice, nous avons utilisé du JavaScript pour modifier le style d'un élément. Toutefois, en règle générale, nous serons plus susceptibles d'utiliser le pseudo-élément CSS `:hover` pour réaliser un changement de style au passage de la souris.

```css
#connexion {
    background: green;
    color: white;
}

/* indique le style de l'élément au passage de la souris */
#connexion:hover {
    background: red;
    color: white;
}
```

p. 18 **Solution n°4**

Solution la plus optimum :

```
<code>
for (let element of document.querySelectorAll('*')) {
   element.addEventListener("click", (event) => {
     event.stopPropagation();
     alert(` Capturing the element: ${element.tagName}`)
   }, false);
}

</code>
```

p. 20 **Solution n°5**

```
<code>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript Custom Event</title>
</head>
<body>
    <div class="note">JS Custom Event</div>
    <script>
        function highlight(elem) {
            const bgColor = 'yellow';
            elem.style.backgroundColor = bgColor;

            // create the event
            let event = new CustomEvent('backgroundYellow', {
                detail: {
                    backgroundColor: bgColor
                }
            });
```

```
            // dispatch the event
            elem.dispatchEvent(event);
        }

        // Select the div element
        let div = document.querySelector('.note');

        // Add border style
        function addBorder(elem) {
            elem.style.border = "solid 1px red";
        }

        // Listen to the highlight event
        div.addEventListener('backgroundYellow', function (e) {
            addBorder(this);

            // examine the background
            console.log(e.detail);
        });

        // highlight div element
        highlight(div);
    </script>
</body>
</html>

</code>
```

Exercice p. 21 **Solution n°6**

Exercice

Quel objet permet de manipuler les événements du DOM ?

- ○ Target
- ◉ Event

Exercice

Quelle méthode permet d'annuler le comportement par défaut d'un événement ?

- ◉ preventDefault()
- ○ stopPropagation()
- ○ initEvent()

Exercice

Cochez les bonnes pratiques de la programmation événementielle en JavaScript.

- ☐ Ne surtout pas supprimer ses événements
- ☑ Limiter le déclenchement de certains événements
- ☑ Vérifier qu'un événement soit compatible sur tous les navigateurs
- ☑ Utiliser la délégation d'événement

Exercice

Quel(s) événement(s) n'existe(nt) pas ?

- [ ] keyup
- [x] mousepress
- [x] stop
- [ ] focus
- [ ] input

Exercice

Sélectionnez les éléments HTML `span`.

- ○ const span = document.getElementById('span')
- ○ const span = document.getElementsByName('span')
- ◉ const span = document.getElementsByTagName('span')

Exercice

Ajoutez un écouteur d'événement `click`.

- ○ span.createEventListener('click', myFunction)
- ○ span.addEventListener('click')
- ◉ span.addEventListener('click', myFunction)
- ○ span.addEventListener(myFunction, 'click')

Exercice

Écrivez le code JavaScript permettant de supprimer l'écouteur d'événement créé à la question précédente.

span.removeEventListener('click', myFunction)

Exercice

Affichez dans la console la cible de l'événement.

- ○ function myFunction() {
  console.log(e.target)
  }
- ◉ function myFunction(e) {
  console.log(e.target)
  }
- ○ function myFunction() {
  console.log(e.cible)
  }
- ○ function myFunction(e) {
  alert(e.target)
  }

Exercice

Créez l'événement `monEvenement`.

- [x] const event = new Event('monEvenement');
- [x] const event = new CustomEvent('monEvenement', { detail : { 'color': 'green' })
- [ ] const event = monEvenement(Event);

p. 23 **Solution n°7**

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>repl.it</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <div>
      <label for="login1">Pseudo1</label>
      <input type="text" id="login1" name="login1">
      <label for="message1">Message</label>
      <input type="text" id="message1" name="message1">
      <button id="btn-send1">Envoyer</button>
    </div>
    <div>
      <label for="login2">Pseudo2</label>
      <input type="text" id="login2" name="field2">
      <label for="message2">Message</label>
      <input type="text" id="message2" name="message2">
      <button id="btn-send2">Envoyer</button>
      <button id="stop">stop</button>
    </div>
    <button id="reset">Rafraichir</button>
    <p id="display-text"></p>
    <script src="script.js"></script>
  </body>
</html>
```

```javascript
// Participant 1
const btnsend1 = document.getElementById('btn-send1')
const login1 = document.querySelector('#login1')
const  message1 = document.querySelector('#message1')

// Participant 2
const btnsend2 = document.getElementById('btn-send2')
const login2 = document.querySelector('#login2')
const message2 = document.querySelector('#message2')

// Zone d'affichage
const displayText = document.getElementById('display-text')

// Boutons d'actions
const btnreset = document.getElementById('reset')
const stop = document.getElementById('stop')

```

```
btnsend1.addEventListener('click', ()=> setTimeout(function(e) {
  displayMessage1(e)
}, 2000))

btnsend2.addEventListener('click', ()=> setTimeout(function(e) {
  displayMessage2(e)
}, 2000))

function displayMessage1(e) {
  displayText.innerHTML += '<p class="message">'+login1.value+' dit '+message1.value+'</p>'
}

function displayMessage2(e) {
  displayText.innerHTML += '<p class="message">'+login2.value+' dit '+message2.value+'</p>'
}

btnreset.addEventListener('click', reset)

function reset() {
  displayText.innerHTML = ''
}

stop.addEventListener('click', stopMessage)

function stopMessage(e) {
  e.preventDefault()
  btnsend2.setAttribute('disabled','disabled')
  btnsend2.removeEventListener('click', displayMessage2)
}
```