Semaine 4 : Les boites de dialogues et la validation des données avec JavaScripts

Cours_ UVCI
PRW2103-1

DIABATE Nabègna,
Université Virtuelle de Côte d'Ivoire

Juillet 2018
1.0

Table des matières

I - Section 1 : Introduction aux événements **3**

1. 1. Qu'est ce qu'un événement 3

2. 2. Gestion et mise en œuvre des événements en JavaScript 4

3. Exercice 6

II - Section 2 : Les boites de dialogue **7**

1. 1. Les boites d'alerte 7

2. 2. les boites de confirmation 7

3. 2. Les boites de saisie 8

4. Exercice 9

III - Section 3 - Validation des données de formulaire avec JavaScript **10**

1. 1. L'accès aux éléments 10

2. 2. Utilisation de JavaScript pour valider les données 11

3. 3. Cas pratique de validation d'un champ en JavaScript 15

4. Exercice 20

Section 1 : Introduction aux événements

1. 1. Qu'est ce qu'un événement

Javascript est un *langage événementiel*. Donc les scripts déclarés dans la page ne sont pas destinés à s'exécuter séquentiellement, linge après ligne, du début à la fin. Mais ils attendent jusqu'à ce qu'un événement soit détecté pour qu'une partie du code s'exécute.

Un événement est une *action* de l'utilisateur qui peut se produire sur le *navigateur*. Le plus célèbre est sans doute le `clic` qui peut survenir sur un objet quelconque (hyperlien, image, bouton...). Le navigateur écoute les événements qui se produisent sur la page Web puis les transmet au script par le biais de ce que l'on appelle *le gestionnaire d' événements*.

C'est par exemple le cas lorsque vous cliquez sur un lien, que vous fermez une fenêtre, que vous cliquez sur un bouton ou encore lorsque tout bêtement la page se charge. Toute cette liste sont des événements qu'on peut relever en JavaScript (pour ne pas dire DHTML évidemment).

La question désormais qui se pose est *comment détecter un nouvel événement sur la page.*

La réponse est assez simple, en effet, le langage JavaScript est doté de plusieurs *écouteurs d'événements*. Chacun d'eux permet de détecter un type d'événement. Et tous ces écouteurs s'incrustent directement dans les balises HTML concernées. Tous les événements commencent par le préfixe `on` suivit de l'action qui s'exécute. Voici la liste des événements les plus récurrents :

Évènement JS	Description / Rôle
onabort	Se déclenche lorsque l'utilisateur stoppe le chargement de la page.
onblur	Se déclenche lorsqu'un élément de la page perd le focus (lorsque l'utilisateur clique ailleurs).
onchange	Se déclenche à chaque fois que la valeur d'un champ de donnée voit sa valeur modifiée.
onclick	Se déclenche lorsque l'utilisateur clique sur l'élément en question.
ondblclick	Se déclenche lorsque l'utilisateur double-clique sur l'élément en question.
ondragdrop	Se déclenche lorsque l'utilisateur glisse et dépose (drag and drop) un objet.
onerror	Se déclenche lorsqu'une erreur apparaît sur la page (JavaScript 1.1).
onfocus	Se déclenche lorsque l'utilisateur donne le focus à un élément (il clique dessus).
onkeydown	Se déclenche lorsque l'utilisateur appuie une touche du clavier (JavaScript 1.2).
onkeypress	Se déclenche lorsque l'utilisateur maintient une touche du clavier enfoncée.
onkeyup	Se déclenche lorsque l'utilisateur relâche une touche du clavier (JavaScript 1.2).
onload	Se déclenche lorsque l'utilisateur charge la page.
onmousedown	Se déclenche lorsque l'utilisateur appuie sur un bouton de sa souris au-dessus d'un élément de la page.
onmouseout	Se déclenche lorsque l'utilisateur ne survole plus un élément de la page (JavaScript 1.1).
onmouseover	Se déclenche lorsque l'utilisateur survole un élément de la page.
onmouseup	Se déclenche lorsque l'utilisateur arrête d'appuyer sur un bouton de sa souris au-dessus d'un élément de la page.
onreset	Se déclenche lorsque l'utilisateur clique sur un input de type reset (réinitialisation d'un formulaire).
onresize	Se déclenche lorsque l'utilisateur redimensionne la fenêtre actuelle.
onselect	Se déclenche lorsque l'utilisateur sélectionne un texte dans un champ de type text ou textarea.
onsubmit	Se déclenche lorsque l'utilisateur soumet un formulaire (soit par un bouton à cliquer soit avec la touche [enter]).
onunload	Se déclenche lorsque l'utilisateur quitte la page actuelle.

Source : https://fr.wikiversity.org/wiki/JavaScript/Introduction_au_DHTML

2. 2. Gestion et mise en œuvre des événements en JavaScript

Le *gestionnaire d'événements* permet d'associer un traitement à un événement.

Sa syntaxe respecte la forme suivante : *onEvénement = "code Javascript édité directement ici, ou fonction appelée à cet endroit"*

Il faut noter que *onEvénement* est un attribut HTML, et ce qui est entre les cottes est sa valeur. Donc la ligne précédente est intégrée directement dans une balise HTML. Cependant, même si ce qui se trouve entre les cottes est purement du code Javascript, nous ne le mettons pas dans les tags de script (<script> et </script>).

Il est très rare qu'on déclare directement la suite d'instructions JavaScript à exécuter entre les cottes. En général, on y appelle une fonction qu'on a déjà déclaré avant dans la balise *<head>*. Ainsi nous séparons les code HTML et JavaScript ce qui rend le code plus lisible et plus facile maintenir.

Faisons un exemple pour mieux comprendre comment associer une action JavaScript à l'événement *click* capturé sur un bouton HTML :

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <meta charset="UTF-8" />
5       <script language="javascript">
6          function fMessage(){
7             alert("Vous avez bien cliqué sur le bouton!");
8          }
9       </script>
10   </head>
11   <body>
12      <input type="button" value="Cliquez ici" onClick="fMessage()" />
13   </body>
14 </html>
```

Le fait de cliquer sur le bouton donne naissance à l'événement *click* qui appellera, grâce au gestionnaire d'événements, la fonction *fMessage()* définie dans le *<head>*. On aura donc le message "*Vous avez bien cliqué sur le bouton!*" qui s'affichera dans une boite de dialogue *alert()*.

3. Exercice

Exercice

1) Lequel de ces types d'événements n'existe pas ?

- ◯ blur
- ◯ load
- ◯ mouseclick
- ◯ mouseout

Exercice

2) A quel événement peut on faire appel pour changer la couleur de fond d'un texte lorsque le souris survole ce texte ?

Exercice

3) Quel événement JavaScript est contraire à l'événement `onload` *?*

Exercice

4) Diane et étudiante a l'UVCI, et pendant la lecture de de son cours, elle perd la connexion à la plate-forme campus. Lorsque la connexion, est rétablie, elle décide d'actualiser la page campus afin de retrouver son cours. Quels sont les événement qui se déclencherons lorsqu'elle actualisera la page ?

- ☐ onload
- ☐ onreset
- ☐ onchange
- ☐ onerror
- ☐ onsubmit

Section 2 : Les boites de dialogue

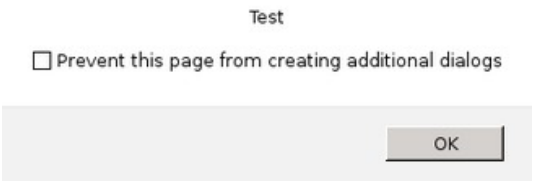
II

Une boîte de dialogue est une petite fenêtre rectangulaire qu'est capable d'afficher n'importe quel navigateur pourvu de JavaScript. Cette boîte permet soit de demander une confirmation au visiteur (*confirm*), soit de prévenir le visiteur (*alert*) ou soit de demander quelque chose au visiteur (*prompt*). Une telle boîte n'est nullement compliquée à mettre en place. Chacune de ces boîtes s'ouvre en faisant appel à des fonctions de JavaScript.

Notez aussi que chacune des boîtes de dialogue de JavaScript a le focus bloqué par le navigateur. C'est-à-dire qu'il faudra cliquer sur un des boutons pour pouvoir continuer.

1. 1. Les boites d'alerte

Les boites d'alerte ont pour principal objectif d'alerter le visiteur de quelque chose. Pour continuer, le visiteur devra juste appuyé sur un bouton du genre "Continuer", "OK", etc.



Caractéristiques :

Nom : `alert()`

Boutons : "`OK`" (et la croix de fermeture)

Code : *`window.alert (texte d'alerte);`*

Le *`window`* devant la fonction *`alert`* est facultatif. Exemple :

```
1 alert('Bonjour, bienvenue sur mon site.\nIl est tout neuf !');
2 // est pareil que
3 windows.alert('Bonjour, bienvenue sur mon site.\nIl est tout neuf !');
```

Notez l'emploi du caractère spécial *\n* pour revenir à la ligne (retour de chariot) dans une alerte JavaScript.

2. 2. les boites de confirmation

JavaScript met aussi a disposition des codeurs une fonction nommée *confirm* qui permet d'afficher une boîte de dialogue demandant une confirmation ou une annulation au visiteur. Par exemple : *Voulez-vous supprimer tous les messages* ?.

Caractéristiques :

Nom: *`confirm()`*

Boutons : "*`OK`*", "*`ANNULER`*" (et la croix de fermeture)

Code : *`window.confirm(texte de confirmation);`*

Exemple :

```
1 var choix = confirm("Voulez-vous supprimer tous les messages ?");
2 if (choix) alert("Vous avez cliqué sur OK");
3 else alert("Vous avez cliqué sur ANNULER ou vous avez fermé");
```

On comprend donc sur cet exemple que *confirm* peut renvoyer deux valeurs :

TRUE si le visiteur a cliqué sur *OK*.

FALSE si le visiteur a cliqué sur *ANNULER* ou a cliqué sur la croix rouge de fermeture.

3. 2. Les boites de saisie

C'est la dernière boîte de dialogue que nous allons voir. Celle-ci permet de demander une saisie au visiteur. Elle contient donc un champs de texte.

Caractéristiques :

Nom: *prompt*

Boutons : "*OK*", "*ANNULER*" (et la croix de fermeture)

Code : *window.prompt(texte de demande, valeur par défaut du textfield);*

Exemple :

```
1 do
2 {
3   choix = prompt("Veuillez entrer un nombre supérieur à zéro :", 0);
4 } while (isNaN(choix) || !choix || Number(choix) < 0);
5
6 document.write("Le nombre que vous avez entré est : "+choix);
```

Le code est certes plus compliqué que précédemment mais il est tout à fait compréhensible. Explication :

- En premier lieu, on utilise la boucle *do-while* car nous voulons être sûr que celle-ci sera parcourue au moins une fois.
- Dans la boucle, on demande d'entrer un nombre positif via le *prompt*.
- Ensuite, il y a le *while* et sa condition : les || veulent dire *OU*. *isNaN* (is Not a Number) permet de vérifier si la saisie est bien un nombre. Puis on vérifie que le visiteur n'a pas cliqué sur *ANNULER* (!choix ou choix == false) et enfin, on vérifie que le nombre est supérieur à 0 (avec Number(choix) > 0).

4. Exercice

Exercice

1) Quelle est la bonne syntaxe ?

- ◯ alert('Hello world!')
- ◯ alert('Hello world!');
- ◯ alert(Hello world!);

Exercice

2) Complétez le code de la boite de dialogue ci-dessous

```
if (          ( "Message à afficher" ) ) {
// Code à exécuter
}      {
// Code à exécuter
}
```

Exercice

3) Quelle(s) boîte(s) de dialogue qui permet(tent) d'afficher un texte et un bouton « OK » ?

- ☐ Alert
- ☐ Confirm
- ☐ Prompt

Exercice

4) On vous demande de concevoir une page web réservé aux adultes de plus de 21 ans. Pour cela, lorsqu'un visiteur entre sur la page, vous devez demander sa date de naissance. S'il a plus de 21 ans, il continue et affiche page, sinon, on on ferme la page et il n'y a pas accès.

Quel(s) boite(s) de dialogues pouvez-vous utiliser ici ?

- ☐ Alert
- ☐ Confirm
- ☐ Prompt

Section 3 - Validation des données de formulaire avec JavaScript

1. 1. L'accès aux éléments

Pour agir sur les éléments d'une page Web, tels que les champs d'un formulaire par exemple, afin d'en contrôler le contenu, il convient de pouvoir accéder à ces différents éléments, de pouvoir cibler de manière spécifique chacun d'entre eux. Voici trois méthodes intéressantes pour cibler un élément du *DOM* (*https://fr.wikipedia.org/wiki/Document_Object_Model*) dans votre page Web.

- `getElementById`
- `getElementsByName`
- `getElementsByTagName`

- getElementById

La traduction littérale donnerait "*Accéder à l'élément a partir de son Id*".

Cette méthode permet d'accéder aux propriétés d'un élément de votre page (par exemple un lien, une image ou un paragraphe) juste en fournissant l'`id` de celui-ci. Voyons un exemple simple :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="iso-8859-1" />
    <script type="text/javascript">
      function supprimerTexte (id_zone) {
        if (document.getElementById) {
        document.getElementById(id_zone).innerHTML = '';
        }
      }
    </script>
  </head>

  <body>
    <span id="mon_texte">Bonjour le monde...</span>
    <p align="center">
    <a href="javascript: supprimerTexte('mon_texte');">Supprimer le texte de cette page</a>
    </p>
  </body>
</html>
```

La propriété `innerHTML` vous permet d'éditer le contenu de l'élément de la page que vous avez

désigné. Cependant, si vous voulez éditer un champ de texte, vous devrez utiliser directement l'attribut *value* ou encore si vous souhaitez cocher une case, vous utiliserez l'attribut *checked*.

Vous pouvez absolument récupérer le contenu de toute balise du code HTML avec *getElementById* et en modifier les attributs : `style, valeur`, etc. Par exemple, pour changer l'image affichée par une balise *img*, vous n'aurez qu'à utiliser :

```
1 var image = document.getElementById('mon_image');
2 // Mise en cache de la nouvelle image
3 var new_image = new Image();
4 new_image.src = './dossier_images/my_picture.jpg';
5 image.src = new_image.src;
```

Donc si vous prenez ce code et ce que vous avez appris sur les événements vous pouvez aisément faire un système permettant de changer une image au passage de la souris (`onMouseOver`, `onMouseOut`).

- *getElementsByName*

Cette méthode est semblable à *getElementById* bien qu'elle soit largement moins utilisée que la première.

En fait, celle-ci fonctionne comme *getElementById* sauf qu'elle se base sur l'attribut *name* des balises au lieu de se baser sur leur *id*. Comme plusieurs éléments de la page Web peuvent avoir la même valeur d'attribut *name*, la méthode ne retournera pas un seul élément, mais une collection d'éléments sous forme de tableau (`array`). Ceci même dans le cas où un seul élément porte le nom recherché (tableau avec une seule valeur). Pour se souvenir de ce comportement, notez bien le "*s*" de "*Elements*" dans le nom de la méthode.

En reprenant l'exemple précédent, on aurait :

```
1 // On recherche tous les éléments dont la valeur de l'attribut "name" est égale à
  "mes_images"
2 var images = document.getElementsByName('mes_images');
3 var old_image = images[0]; // si l'on prend la première de la collection
4
5 // Mise en cache de la nouvelle image
6 var new_image = new Image();
7 new_image.src = './dossier_images/my_picture.jpg';
8 old_image.src = new_image.src;
```

- *getElementsByTagName*

Cette méthode fonctionne comme "*getElementsByName*" sauf qu'elle se base sur la balise (HTML) des éléments recherchés, et non sur la valeur d'un attribut.

Avec le même exemple, on aurait :

```
1 // On recherche toutes les balises "IMG" (nota: on peut aussi écrire "img")
2 var images = document.getElementsByTagName('IMG');
3 var old_image = images[0]; // si l'on prend la première de la collection
4
5 // Mise en cache de la nouvelle image
6 var new_image = new Image();
7 new_image.src = './dossier_images/my_picture.jpg';
8 old_image.src = new_image.src;
```

2. 2. Utilisation de JavaScript pour valider les données

Le JavaScript va nous être un bon moyen d'effectuer une validation puissante des données utilisateurs côté client car il possède des propriétés spécialement créées pour la validation d'éléments

de formulaire.

Cependant, puisque l'utilisateur peut désactiver JavaScript sur son navigateur, le JavaScript sera utilisé pour avertir les utilisateurs bienveillants d'un oubli ou d'une erreur lors du remplissage d'un formulaire en direct plus que pour empêcher des utilisateurs malveillants d'exploiter nos formulaires.

Il serait aussi possible, comme nous l'avons déjà vu, d'afficher le formulaire que si l'utilisateur n'a pas désactivé JavaScript (Vous avez surement déjà vu ce message sur certains sites web : "*Vous devez activer JavaScript pour continuer la navigation sur ce site !*").

Avec les formulaires, JavaScript :

- Permet de valider les formulaires côté client. Attention : ne dispense pas de la validation côté serveur, qui est indispensable (sécurité).
- On peut accéder aux formulaires et au champs par leurs *id* si on leur en donne.
- On peut accéder aux formulaires par leur nom (*name*); de même pour les champs.
- Principe: on lie des fonctions JavaScript à certains événements (*on quitte un champ, on valide le formulaire, on presse une touche...*)

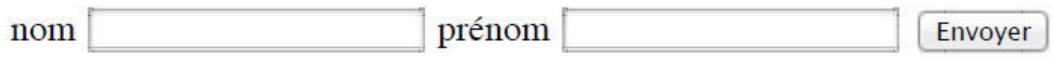
On peut tester la validité d'un formulaire à trois (03) moments bien précis :

- A l'expédition du formulaire
- Au cours de l'édition des champs de saisie
- Lors de la frappe dans un champs de saisie

a) Vérification à l'expédition

- On veut que le formulaire soit vérifié juste avant d'être expédié
- On utilise l'événement `onsubmit` de la balise `form`
- Quand le formulaire va être envoyé, on appelle le code JavaScript dans `onsubmit`
- Si la valeur de `onsubmit` a la forme `return` `nomDeFonction`(): Si la fonction renvoie `true` , l'action est effectuée. Sinon, le formulaire n'est pas expédié

Si on veut par exemple tester que le nom et le prénom fournis par l'utilisateur ne sont pas vides.




```
<script type="text/javascript">
  function verifChampNonVideVersion0(id) {
    var result= true;
    field= document.getElementById(id);
    // On accède à la valeur du champ par la propriété javascript value,
    // et pas par l'attribut DOM value !!!
    var value= field.value;
    if (value == "") {
      result= false;
    }
    return result;
    }
  function verifNomPrenomPasVides() {
    var r= verifChampNonVideVersion0('formEx0Nom');
    if (r == true)
      r= verifChampNonVideVersion0('formEx0Prenom');
    if (! r)
      alert("erreur: champ non renseigné");
    return r;
    }
</script>

<form action="traitement.php" onsubmit="return verifNomPrenomPasVides()">
```

```
  <p>nom <input type="text" name="nom" id="formEx0Nom">
  prénom <input type="text" name="prenom" id="formEx0Prenom">
  <input type="submit" value="Envoyer"></p>
</form>
```

Cette méthode est critiquée pour certaines raisons :

- `alert()` : message d'erreur peu précis
- disparaît quand on veut corriger
- agressif (nouvelle fenêtre)

On utilisera, pour palier à cela, plutôt la manipulation du DOM pour écrire les messages d'erreur directement sur la page.


```

  <script type="text/javascript">
    //<![CDATA[
    // La ligne qui précède est un code XML/SGML qui protège le
          // javascript de toute interprétation.
          // n'est plus utile en HTML5
  function verifNomPrenomPasVides1() {
           var r0= verifChampNonVideVersion0('formEx1Nom');
    var r1= verifChampNonVideVersion0('formEx1Prenom');
    var msg0= "";
    var msg1= "";
    if (! r0)
      msg0= "champ vide";
    if (! r1)
      msg1= "champ vide";
    document.getElementById('Ex1ErreurNom').innerHTML= msg0;
    document.getElementById('Ex1ErreurPrenom').innerHTML= msg1;
    return r0 && r1;
  }
  //]]>
  </script>

  <form name="formEx1" action="test.php" onsubmit="return
verifNomPrenomPasVides1()">
          <table>
    <tbody><tr>
      <td>nom</td>
      <td><input name="nom" id="formEx1Nom" type="text"> <span id="Ex1ErreurNom"
class="error"></span></td>
    </tr>
    <tr>
      <td>prénom</td>
      <td><input name="prenom" id="formEx1Prenom" type="text"> <span id=
"Ex1ErreurPrenom" class="error"></span></td>
    </tr>
    </tbody></table>
    <br><input type="submit" value="Envoyer">
  </form>
```

b) Traitement des champs en cours d'édition

Le plus souvent, on utilise l'événement *onblur* ou *onchange*

- Avec *onblur*, le code est appelé quand le curseur quitte le champ (il perd le "focus"),
- Avec *onchange*, le code est appelé quand le curseur quitte le champ et que la valeur a changé;
- *onchange* s'utilise aussi pour un champ de type select, auquel cas il permet tout simplement de savoir que la sélection a été modifiée.
- S'utilise en complément de *onsubmit*

On n'a pas besoin de retourner une valeur


```
<script type="text/javascript">

  function verifChampNonVideVersion2(idChamp,idErreur) {
    var result= true;
    var msg="";
    if (document.getElementById(idChamp).value == "") {
      result= false;
      msg= "champ vide";
    }
    document.getElementById(idErreur).innerHTML= msg;
    return result;
  }

  function verifNomPrenomPasVides2() {
    var r0= verifChampNonVideVersion0('formEx2Nom', 'Ex2ErreurNom');
    var r1= verifChampNonVideVersion0('formEx2Prenom', 'Ex1ErreurNom');
    document.getElementById('Ex1ErreurNom').innerHTML= msg0;
    document.getElementById('Ex1ErreurPrenom').innerHTML= msg1;
    return r0 && r1;
  }

 </script>

<form name="formEx2" action="test.php" onsubmit="return verifNomPrenomPasVides1()"
>
  <table>
    <tbody><tr>
      <td>nom</td>
      <td><input name="nom" id="formEx2Nom" onblur=
"verifChampNonVideVersion2('formEx2Nom', 'Ex2ErreurNom')" type="text">
  <span id="Ex2ErreurNom" class="error"></span></td>
    </tr>
    <tr>
      <td>prénom</td>
      <td><input name="prenom" id="formEx2Prenom" onblur=
"verifChampNonVideVersion2('formEx2Prenom', 'Ex2ErreurPrenom')" type="text">
  <span id="Ex2ErreurPrenom" class="error"></span></td>
    </tr>
  </tbody></table>
```

```
  <br><input type="submit">
</form>
```

c) Vérification lors de la frappe

On peut utiliser *onkeydown* pour un traitement avant modification du champ ou *onkeyup* pour un traitement a posteriori.

Noter l'utilisation de *return* (ne fonctionne qu'avec onkeydown).

```
<script type="text/javascript">
// inspiré de http://www.w3schools.com/jsref/jsref_onkeypress.asp
function filtreNombre(e)
{
var keynum;

if(window.event) // IE
  {
  keynum = e.keyCode;
  }
else if(e.which) // autres
  {
  keynum = e.which;
  }
var s = String.fromCharCode(keynum)

return "0123456789".indexOf(s) == -1;
}
</script>

<p>Tapez du texte (mais pas de chiffre)
<input onkeydown="return filtreNombre(event)" type="text">
</p>
```

Complément

Vous pouvez avoir un complément d'informations sur la validation des formulaires avec JavaScript sur *OpenWeb* en cliquant sur le lien suivant :

https://openweb.eu.org/articles/validation_formulaire

3. 3. Cas pratique de validation d'un champ en JavaScript

Sur un champ de type texte, nous pouvons vérifier la pertinence des données renseignées dans le champ « prenom » d'un formulaire par exemple.

Nous allons vouloir vérifier différentes choses :

- Le champ n'est pas vide ;
- Le champ ne contient que des lettres (accentuées ou pas), des tirets et des espaces (pour gérer les prénoms composés) ;
- Le champ contient entre 5 et 30 caractères.

Nous allons également vouloir afficher un message d'erreur lorsque le champ n'est pas bien rempli avec une indication sur l'erreur rencontrée.

- Vérifier la présence d'une valeur

Pour simplifier un peu notre script final, nous allons effectuer les vérifications des données lors du clic sur le bouton « *Valider* » de notre formulaire, c'est-à-dire lorsque l'utilisateur tente d'envoyer le formulaire.

Si un champ est vide, le formulaire ne doit pas être envoyé et un message d'erreur doit être affiché là où la valeur est manquante.

Pour faire cela, nous allons devoir créer un gestionnaire d'événement *click* contenant notre fonction de validation.

Je vous conseille de rajouter un *id* à votre input type="*submit*" afin de pouvoir y accéder plus facilement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Les formulaires Avec JavaScript</title>
        <meta charset="utf-8">

        <style>
            label{display: inline-block;min-width: 200px;}
        </style>
    </head>

    <body>
        <h1>Les formulaires Avec JavaScript</h1>

        <form method="post" action="traitement.php">

            <label for='prenom'>Entrez votre prénom svp : </label>
            <input type='text' name='prenom' id='prenom' maxlength='20' required>
            <span id='missPrenom'></span><br>

            <label for='mail'>Entrez votre mail : </label>
            <input type='email' name='mail' id='mail' required><br>

            <label for='tel'>Numéro de téléphone :</label>
            <input type='tel' name='tel' id='tel' required><br>

            <input type='submit' value='Valider' id='bouton_envoi'>
        </form>

        <script>
            var formValid = document.getElementById('bouton_envoi');

            formValid.addEventListener('click', validation);

            function validation(){

            }
        </script>
    </body>
</html>


```

Ensuite, pour vérifier la présence de données en soi, nous allons utiliser la propriété *validity* de l'objet *Element*.

Cette propriété contient elle même différentes propriétés permettant de vérifier des données et qui vont renvoyer un booléen.

Pour vérifier qu'un champ n'est pas vide, nous allons utiliser la propriété *valueMissing*, qui renvoie *true* si un champ possédant un attribut *required* est vide.

Dans le cas où *valueMissing* renvoie *true*, nous allons donc bloquer l'envoi du formulaire et

renvoyer un message d'erreur.

Pour bloquer l'envoi du formulaire, nous allons utiliser la méthode *preventDefault*() si aucune valeur n'est envoyée.

Pour rappel, *preventDefault()* est une méthode de l'objet *Event* qui va annuler le déclenchent d'un événement si celui-ci est annulable.

Finalement, nous allons vouloir afficher un message décrivant l'erreur à côté du champ. Pour cela, nous allons rajouter un élément *span* avec un *id* dans notre formulaire.

Nous allons ensuite utiliser la propriété *textContent* pour afficher l'erreur.

Voici donc le code auquel vous devriez arriver :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Les formulaires</title>
        <meta charset="utf-8">

        <style>
            label{display: inline-block;min-width: 200px;}
        </style>
    </head>

    <body>
        <h1>Les formulaires HTML</h1>

        <form method="post" action="traitement.php">

            <label for='prenom'>Entrez votre prénom svp : </label>
            <input type='text' name='prenom' id='prenom' maxlength='20' required>
            <span id='missPrenom'></span><br>

            <label for='mail'>Entrez votre mail : </label>
            <input type='email' name='mail' id='mail' required><br>

            <label for='tel'>Numéro de téléphone :</label>
            <input type='tel' name='tel' id='tel' required><br>

            <input type='submit' value='Valider' id='bouton_envoi'>
        </form>

        <script>
            var formValid = document.getElementById('bouton_envoi');
            var prenom = document.getElementById('prenom');
            var missPrenom = document.getElementById('missPrenom');

            formValid.addEventListener('click', validation);

            function validation(event){
                //Si le champ est vide
                if (prenom.validity.valueMissing){
                    event.preventDefault();
                    missPrenom.textContent = 'Prénom manquant';
                    missPrenom.style.color = 'red';
                }
            }
        </script>
    </body>
```

```
47 </html>
48
49
```

Dans le code ci-dessus, si le champ prénom est vide, alors *`valueMissing`* renvoie *`true`* et on exécute le code dans le if, à savoir :

- Empêcher l'envoi du formulaire avec *`preventDefault()`* ;
- Créer et afficher un message d'erreur.

- *Vérifier la qualité des données envoyées*

La première partie du travail est faite pour notre champ prénom. Maintenant, nous voulons également vérifier que l'utilisateur n'envoie pas n'importe quoi.

Nous n'allons autoriser que les lettres, les apostrophes, les tirets et les espaces pour ce champ.

De plus, nous n'autoriserons l'utilisation de majuscule qu'en début de mot et nous n'autoriserons pas les tirets en début ou en fin de chaîne.

La meilleure façon de faire tout cela est, vous l'avez certainement deviné, d'utiliser les expressions régulières.

Je vous invite à faire une recherche sur les *RegEx* JavaScript pour pouvoir aborder cette partie du cours.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Les formulaires</title>
5         <meta charset="utf-8">
6         <style>label{display: inline-block;min-width: 200px;}</style>
7     </head>
8
9     <body>
10        <h1>Les formulaires HTML</h1>
11        <form method="post" action="traitement.php">
12            <label for='prenom'>Entrez votre prénom svp : </label>
13            <input type='text' name='prenom' id='prenom' maxlength='20' required>
14            <span id='missPrenom'></span><br>
15
16            <label for='mail'>Entrez votre mail : </label>
17            <input type='email' name='mail' id='mail' required><br>
18            <label for='tel'>Numéro de téléphone :</label>
19            <input type='tel' name='tel' id='tel' required><br>
20            <input type='submit' value='Valider' id='bouton_envoi'>
21        </form>
22
23        <script>
24            var formValid = document.getElementById('bouton_envoi');
25            var prenom = document.getElementById('prenom');
```

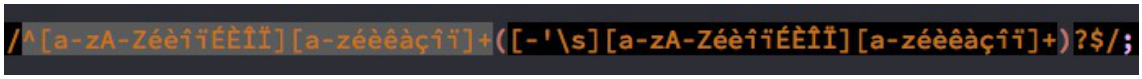
```
            var missPrenom = document.getElementById('missPrenom');
            var prenomValid =
/^[a-zA-ZéèîïÉÈÎÏ][a-zéèêàçîï]+([-'\s][a-zA-ZéèîïÉÈÎÏ][a-zéèêàçîï]+)?$/;

            formValid.addEventListener('click', validation);

            function validation(event){
                //Si le champ est vide
                if (prenom.validity.valueMissing){
                    event.preventDefault();
                    missPrenom.textContent = 'Prénom manquant';
                    missPrenom.style.color = 'red';
                //Si le format de données est incorrect
                }else if (prenomValid.test(prenom.value) == false){
                    event.preventDefault();
                    missPrenom.textContent = 'Format incorrect';
                    missPrenom.style.color = 'orange';
                }else{
                }
            }
        </script>
    </body>
</html>

```

Une petite explication de la *regex* ci-dessus s'impose. Nous allons la décortiquer partie par partie.



Dans la première partie (partie surlignée en plus clair ci-dessus), nous demandons tout simplement à l'utilisateur de commencer soit par une lettre non accentuée en majuscule ou en minuscule, soit par l'un des caractères suivants : « *éèîïÉÈÎÏ* ».

On demande ensuite que cette première lettre soit suivie par au moins une autre lettre en minuscule ou par l'un des caractères suivants : « *éèêàçîï* ». Notez l'utilisation du signe `+` pour demander « au moins une autre... ».

La deuxième partie de notre *regex* concerne le second prénom (dans le cas d'un prénom composé par exemple, ou d'un prénom contenant un apostrophe).

Ce second prénom doit être facultatif. Nous avons donc utilisé le signe `?` sur toute cette partie en l'entourant au préalable par des parenthèses pour faire porter le point d'interrogation sur toute la partie de la *regex*.

Cette deuxième partie commence soit par un apostrophe, un tiret ou un espace `\s`, puis se poursuit par le deuxième prénom en soi (on réutilise la première partie de la regex ici).

Si vous vous posez des questions sur la pertinence de cette *regex*, gardez bien à l'esprit qu'aucune *regex* n'est jamais parfaite.

Par exemple, cette regex va accepter des prénoms comme « Pç » et refuser les prénoms commençant par un « È » par exemple.

Tout dépend du niveau de restriction que vous souhaitez appliquer à votre champ. Dans mon cas, je voulais une *regex* pas trop complexe qui éliminerait la majorité des données incohérentes tout en acceptant tous les prénoms communs.

Notez encore une fois qu'une *regex* n'est jamais qu'un compromis entre ce qu'on souhaite accepter et ce qu'on veut refuser : plus votre *regex* va être stricte, plus vous risquez de refuser des données à priori « bonnes ».

Quoiqu'il en soit, la validation de notre champ prénom est désormais terminée, n'hésitez pas à essayer ce script ou à le compléter selon vos besoins !

4. Exercice

Exercice

1) Quelle méthode n'existe pas dans le DOM ?

- ◯ document.getElementsByClassName
- ◯ document.getElementsByTagName
- ◯ document.getElementsByAttribute
- ◯ document.getElementById

Exercice : Faites un peu de recherche !

2) L'instruction `var px = window.event.x;` permet de :

- ☐ connaître la position du point d'insertion
- ☐ stocker la position du curseur
- ☐ stocker la position en x du curseur de la souris

Exercice

3) Quels événement peuvent être utilisé pour tester la validité d'un champ de formulaire en cours d'édition ?

- ☐ onblur
- ☐ onfocus
- ☐ onchange
- ☐ onsubmit

Exercice

4) l'événement `onclick` peut-il être utilisé pour vérifier un formulaire à l'expédition ?

- ◯ Oui
- ◯ Non