

BLONDAT Cyprien

ESIREM 3A IT

COMPTE-RENDU DÉVELOPPEMENT WEB

Sommaire :

- I) **Introduction**
- II) **Exercice 1**
- III) **Exercice 2**
- IV) **Exercice 3**
- V) **Exercice 4**
- VI) **Exercice 5**
- VII) **Conclusion**

INTRODUCTION

Ce compte-rendu portera sur la réalisation de 5 exercices de développement web, avec les langages HTML et CSS. Nous allons pouvoir réaliser nos premières pages WEB grâce au langage HTML, et nous pourrons agencer ces dernières grâce au CSS.

Nous travaillerons avec le HTML5, la dernière version, ainsi que le CSS3.



EXERCICE 1

Nous commençons par ouvrir le fichier `etiquettes_formulaire.html`, fourni.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemple etiquettes de formulaire</title>
5   <meta name="viewport" content="width=device-width" />
6 </head>
7 <body>
8   <h1>Etiquettes de formulaire flexibles avec CSS</h1>
9   <form action="" method="get">
10    <p><label for="username">Nom d'utilisateur:</label> <input type="text" /></p>
11    <p><label for="password">Mot de passe:</label> <input type="password" /></p>
12  </form>
13 </body>
14 </html>
15
```

Image 1 : etiquettes_formulaire.html

Puis, nous créons un fichier css, baptisé « `fichier.css` » qui sera notre document servant à établir l'agencement de notre page WEB, et nous le lions a notre fichier HTML, via la commande suivante :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple etiquettes de formulaire</title>
5      <meta name="viewport" content="width=device-width" />
6      <link rel="stylesheet" href="fichier.css" />
7  </head>
8  <body>
9      <h1>Etiquettes de formulaire flexibles avec CSS</h1>
10     <form action="" method="get">
11         <p><label for="username">Nom d'utilisateur:</label> <input type="text" /></p>
12         <p><label for="password">Mot de passe:</label> <input type="password" /></p>
13     </form>
14 </body>
15 </html>
16

```

Image 2 : Lien entre le fichier HTML et CSS

Et nous remplissons notre fichier CSS des lignes de codes suivantes :

```

1  h1 {font-size: 18pt;}
2  label
3  {
4      width: 100px;
5      text-align: right;
6      display: inline-block;
7      vertical-align: baseline;
8  }

```

Image 3 : Début du fichier CSS

Ces commandes ont différentes utilités. Ainsi « label » et « h1 » sont des noms de balises, et toutes les balises qui portent ces noms seront modifiées. H1 correspond au titre.

Font-size : La taille du texte. Donc le titre verra sa taille modifiée.

Width : La largeur, de 100 pixels ici.

Text-align : Aligné le texte, ici à droite.

Display : Type de mise en page, ici en « inline-block »

Grâce à ce type de mise en page, nous pouvons utiliser « vertical-align » pour aligner le texte verticalement.

Nous avons le résultat suivant :

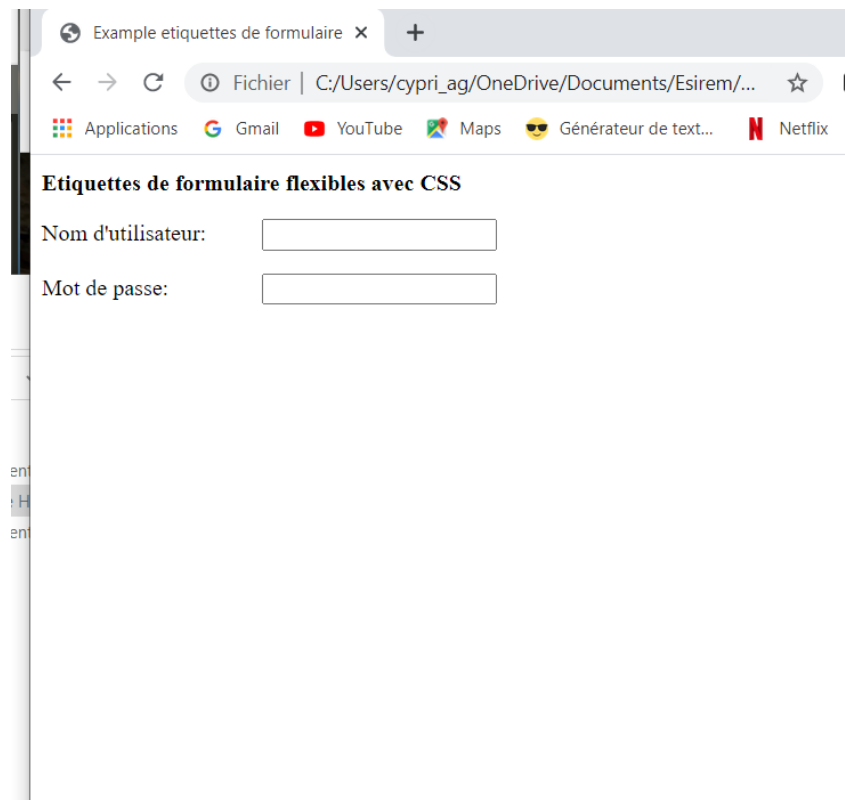


Image 4 : Affichage de la page Web etiquettes_formulaire

Nous souhaitons désormais afficher les étiquettes au-dessus des champs du formulaires, pour les terminaux de moins de 480 pixels.

Nous rajoutons dans le fichier CSS une requête média :

```
@media screen and (max-width: 480px){  
  label  
  {  
    width: 100%;  
  }  
  input  
  {  
    width: 100%;  
  }  
}
```

Image 5 : Requête média

Cette requête concerne les écrans dont la largeur est max de 480 pixels (max-width : 480px). Et stipule que les éléments « label » occupent une largeur de 100%, et les éléments

input également. Ainsi, les champs, et les noms occuperont chacun une ligne, permettant aux noms d’être au-dessus des champs.

Etiquettes de formulaire flexibles avec CSS

Nom d'utilisateur:

Mot de passe:

Image 6 : Nouvel affichage avec la requête média

EXERCICE 2

On commence par récupérer le fichier `media_queries.html` et on l’affiche dans le navigateur.

Voici un aperçu en plein écran :

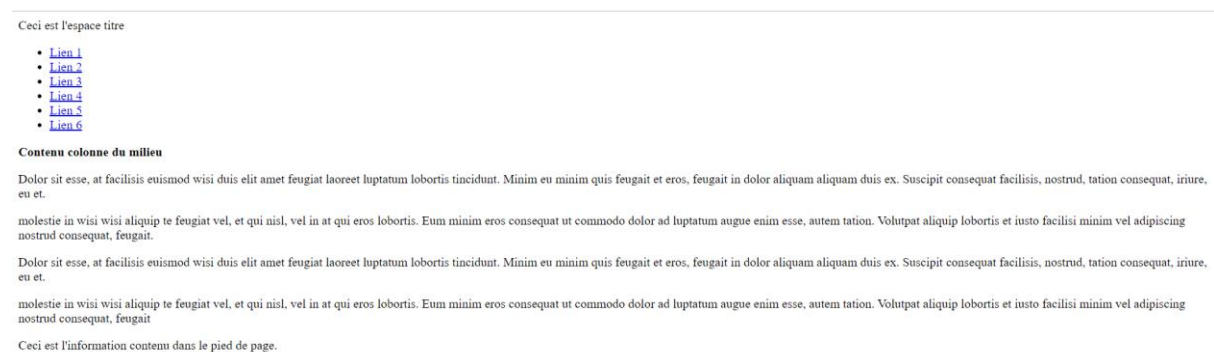


Image 7 : Media_queries.html en plein écran

On redimensionne la taille de la fenêtre :

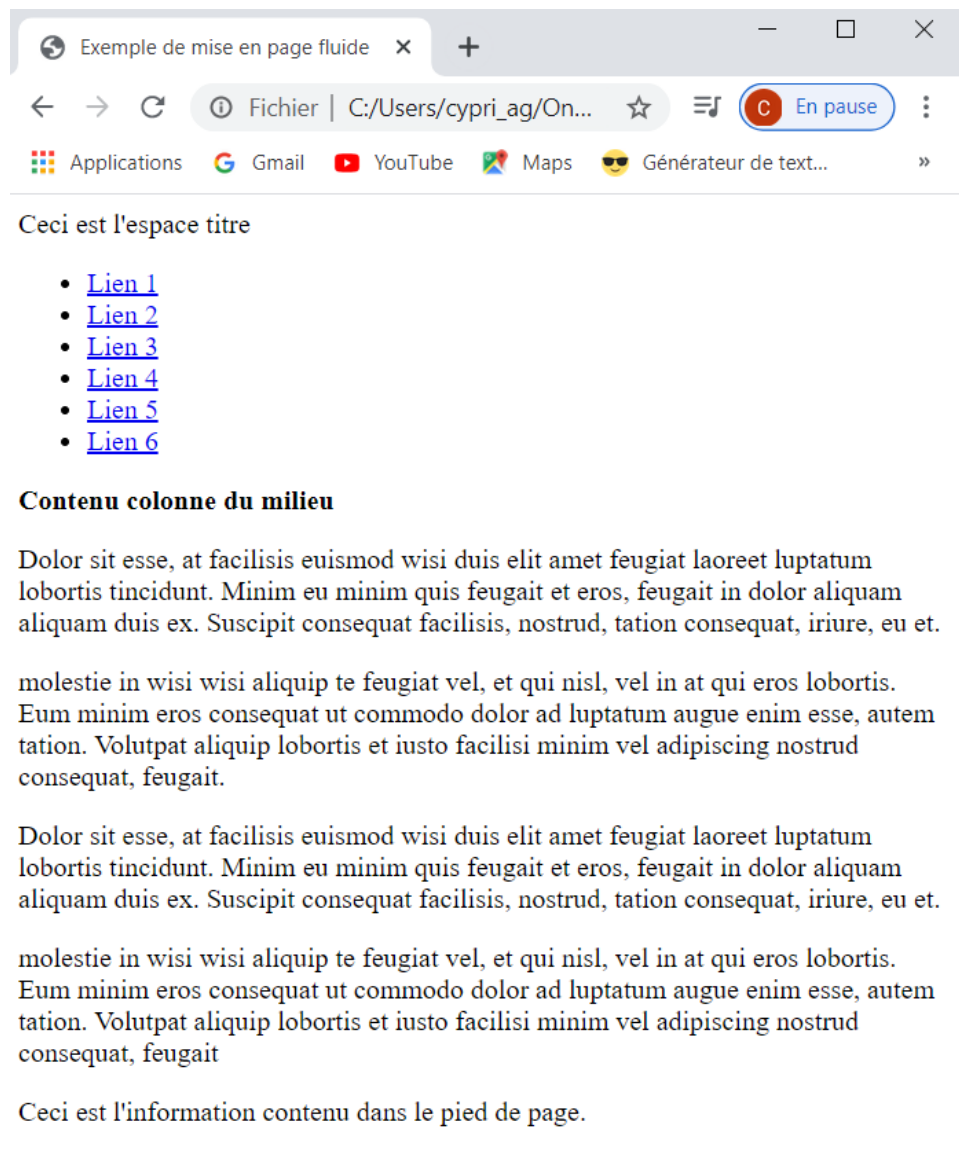


Image 8 : media_queries.html sur un écran plus réduit

On remarque que l’affichage du texte s’adapte à la taille de l’écran. Ainsi, sur un écran plus réduit, le nombre de lignes augmentent pour combler la place perdue sur la longueur.

Nous récupérons maintenant media_queries.css, le fichier CSS correspondant au fichier HTML, et nous y incluons le code dans une requête média pour les écrans de plus de 801 pixels :

```
@media screen and (min-width: 801px){  
  #titrePage {  
    font-size:36pt;  
    font-family:"Times New Roman", Times, serif;  
    background-color:#9999FF  
  }  
  
  #container {  
    font-size: 16pt;  
    position: relative;  
    width: 100%;  
  }  
  
  #colonneG {  
    width: 200px;  
    height: 100%;  
    float:left;  
  }  
  
  #contenuPage {  
    margin-left: 210px;  
  }  
  
  #footer {  
    border: 2px gray solid;  
    padding: 5pt;  
    margin-top: 5pt;  
  }  
}
```

Image 9 : media_queries.css

Nous affichons à nouveau la page Web et nous obtenons le résultat suivant :

Ceci est l'espace titre

- [Lien 1](#)
- [Lien 2](#)
- [Lien 3](#)
- [Lien 4](#)
- [Lien 5](#)
- [Lien 6](#)

Contenu colonne du milieu

Dolor sit esse, at facilisis euismod wisi duis elit amet feugiat laoreet luptatum lobortis tincidunt. Minim eu minim quis feugait et eros, feugait in dolor aliquam aliquam duis ex. Suscipit consequat facilisis, nostrud, tation consequat, iriure, eu et.

molestie in wisi wisi aliquip te feugiat vel, et qui nisl, vel in at qui eros lobortis. Eum minim eros consequat ut commodo dolor ad luptatum augue enim esse, autem tation. Volutpat aliquip lobortis et iusto facilisi minim vel adipiscing nostrud consequat, feugait.

Dolor sit esse, at facilisis euismod wisi duis elit amet feugiat laoreet luptatum lobortis tincidunt. Minim eu minim quis feugait et eros, feugait in dolor aliquam aliquam duis ex. Suscipit consequat facilisis, nostrud, tation consequat, iriure, eu et.

molestie in wisi wisi aliquip te feugiat vel, et qui nisl, vel in at qui eros lobortis. Eum minim eros consequat ut commodo dolor ad luptatum augue enim esse, autem tation. Volutpat aliquip lobortis et iusto facilisi minim vel adipiscing nostrud consequat, feugait

Ceci est l'information contenu dans le pied de page.

Image 10 : *media_queries.html* avec le fichier CSS correspondant

Puis on y inclut une requête média pour les écrans de moins de 800 pixels, dans lequel nous souhaitons aligner le titre à droite, supprimer les puces, ou encore changer la couleur de fond du titre :

```
@media screen and (max-width: 800px){
  #titrePage{
    font-size: 36pt;
    font-family: "Times New Roman", Times, serif;
    background-color: #808000;
    text-align: right;
  }

  #colonneG{
    width: 100%;
    list-style: none;
  }
  ul{
    margin: 0;
    padding: 0;
    list-style-type: none;
  }
}
```

Image 11 : Ajout au fichier *media_queries.css*

Nous changeons le code hexadécimal de la couleur, pour obtenir du kaki, et nous alignons le texte a droite avec la commande « text-align ».

Puis nous modifions la balise « ul » qui gère les puces et numérotations, en retirant le type de numérotations, tout simplement.

Nous obtenons le résultat suivant :



Image 12 : Nouvelle page web media_queries

EXERCICE 3

On commence par récupérer le fichier modernizr.js, qui est un fichier javascript. Puis on ouvre le fichier detection_de_fonctionnalites.html, afin de l'examiner.

```

<!DOCTYPE html>
<html>
<head>
  <title>Utiliser la détection de fonctionnalités côté client</title>
  <meta charset="UTF-8"/>
  <meta name="viewport" content="width=device-width" />
</head>
<body>
<h1>Utiliser la détection de fonctionnalités côté client</h1>
<p>Cet exemple illustre comment utiliser Modernizr pour détecter la prise
en charge de fonctionnalités dans un navigateur donné.
  L'ensemble du code est exécuté côté client, et utilise à la fois du
  JavaScript et du CSS pour détecter la prise en charge
  de fonctionnalités individuelles, au lieu d'employer du code côté
  serveur.</p>
<h3>Fonctionnalités JavaScript</h3>
<p>Geolocalisation: <span id="geoloc"></span></p>
<p>Evenements tactiles: <span id="touch"></span></p>
<h3>Fonctionnalités HTML5</h3>
<p>SVG: <span id="svg"></span></p>
<p>Canvas: <span id="canvas"></span></p>
<h3>Fonctionnalités CSS3</h3>
<p class="animtest">animations CSS prises en charge</p>
<p class="noanimtest">animations CSS non prises en charge</p>
</body>
</html>

```

Image 13 : *detection_de_fonctionnalites.html*

On remarque que le texte affiché est une succession de titres (les balises h1, h3, etc) et de texte (balise p). Certains morceaux du texte ont une classe associée, il est donc probable que ces parties de texte soient agencés de différentes manières par la suite. Cependant, pour l'instant, notre page ressemble à ça :

Utiliser la détection de fonctionnalités côté client

Cet exemple illustre comment utiliser Modernizr pour détecter la prise en charge de fonctionnalités dans un navigateur donné. L'ensemble du code est exécuté côté client, et utilise à la fois du JavaScript et du CSS pour détecter la prise en charge de fonctionnalités individuelles, au lieu d'employer du code côté serveur.

Fonctionnalités JavaScript

Geolocalisation: pris en charge

Evenements tactiles: non pris en charge

Fonctionnalités HTML5

SVG: pris en charge

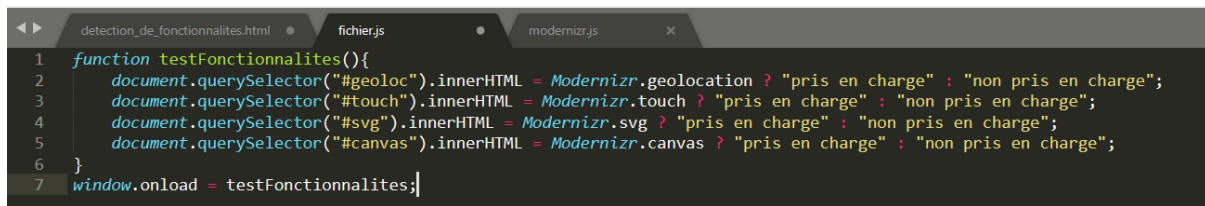
Canvas: pris en charge

Fonctionnalités CSS3

animations CSS prises en charge

Image 14 : *Page web detection_de_fonctionnalites.html*

Dans un nouveau fichier.js (javascript), on écrit le code suivant :



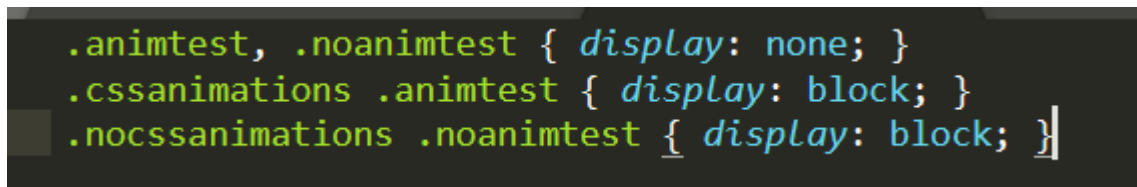
```
1 function testFonctionnalites(){
2   document.querySelector("#geoloc").innerHTML = Modernizr.geolocation ? "pris en charge" : "non pris en charge";
3   document.querySelector("#touch").innerHTML = Modernizr.touch ? "pris en charge" : "non pris en charge";
4   document.querySelector("#svg").innerHTML = Modernizr.svg ? "pris en charge" : "non pris en charge";
5   document.querySelector("#canvas").innerHTML = Modernizr.canvas ? "pris en charge" : "non pris en charge";
6 }
7 window.onload = testFonctionnalites;
```

Image 15 : Fichier.js

La fonction testFonctionnalites, teste les fonctionnalités, comme son nom l'indique. Pour chaque élément, la fonction va aller voir dans le fichier HTML comment sont « geolocation / touch / svg / canvas ». En fonction de leur état, ils seront pris ou en charge ou non.

Les fonctionnalités qui sont testés sont la géolocalisation, les événements tactiles, le svg (coordonnées, structures et dimensions des objets vectoriels) et le canvas (contour des images).

Nous ajoutons ces lignes de code dans le fichier.css.



```
.animtest, .noanimtest { display: none; }
.cssanimations .animtest { display: block; }
.nocssanimations .noanimtest { display: block; }
```

Image 16 : Fichier.css

Ces lignes de codes traitent de l'affichage de certains éléments du html. Les éléments qui ont pour classe animtest et noanimtest ne seront pas affichés. Puis les éléments d'après seront affichés en bloc.

Ce genre de code permet de gérer l'affichage selon la situation, et ainsi, de parfaire l'affichage de sa page Web.

EXERCICE 4

On récupère le fichier stockage_local.html :

```

<!DOCTYPE html>
<html>
  <head>
    <title>API Web Storage du W3C</title>
    <meta charset="UTF-8"/>
    <meta name="Viewport" content="width=device-width" />
  </head>
  <body>
    <h1>Utilisation du DOM localStorage</h1>
    <div>
      L'<a href="http://www.w3.org/TR/webstorage/">API Web Storage
      du W3C</a>
      fournit 2 nouvelles manières pour stocker des informations
      côté client -
      les objets <code>sessionStorage</code> et <code>localStorage</
      code>. Cette démo illustre
      comment utiliser la fonctionnalité localStorage pour
      enregistrer des informations dans le navigateur
      sans avoir à utiliser les cookies.</div>
    <h2>Exemple de formulaire</h2>
    <p>Essayez de saisir des informations dans le formulaire, puis
    fermez et rouvrez la page.</p>
    <div>
      <form action="" method="post" id="infoform">
        <fieldset name="LocalStorage" id="ls">
          <legend>Informations personnelles</legend>
          <p><label id="fnLabel" for="firstName">Prénom: </label>
          <input id="firstName" name="firstName" /></p>
          <p><label id="lnLabel" for="lastName">Nom: </label>
          <input id="lastName" name="lastName" /></p>
          <p><label id="pcLabel" for="postCode">Code postal: </label>
          <input id="postCode" name="postCode" /></p>
          <input type="button" value="Enregistrer" onclick="
            storeLocalContent(
              document.querySelector('#firstName').value,
              document.querySelector('#lastName').value,
              document.querySelector('#postCode').value
            )">
          <input type="button" value="Effacer" onclick="
            clearLocalContent()">
        </fieldset>
      </form>
    </div>
  </body>
</html>

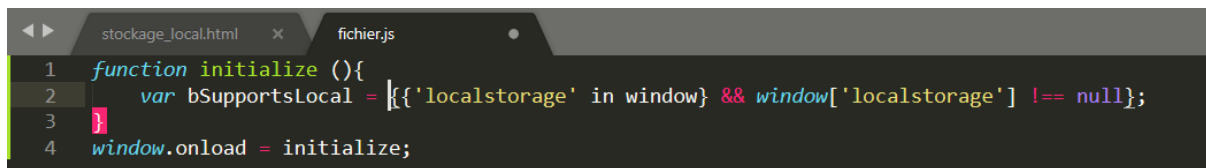
```

Image 17 : stockage_local.html

Ce programme commence par du texte, dont un lien URL vers 'API Web Storage du W3C', puis est une succession, de textes, titres, étiquettes, et champs. Enfin, nous avons 2

fonctions javascript, storeLocalContent et clearLocalContent, qui servent respectivement à enregistrer et effacer les informations rentrées dans les champs.

Nous créons un fichier javascript, dans lequel nous entrons les morceaux de codes suivants :

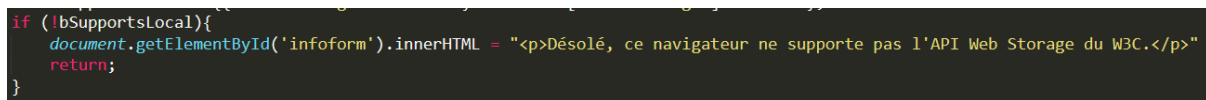


```
1 function initialize(){
2     var bSupportsLocal = [{ 'localStorage' in window } && window['localStorage'] !== null];
3 }
4 window.onload = initialize;
```

Image 18 : fichier javascript

Ces quelques lignes de code servent à initialiser les données. Ainsi, on s'assure que la variable bSupportsLocal est non nulle, et que l'initialisation a bien été réalisée.

On ajoute les lignes de codes suivantes :



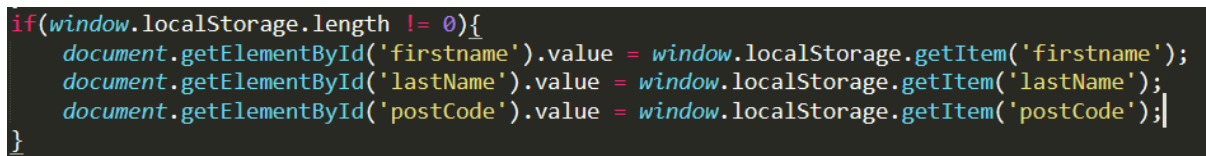
```
if (!bSupportsLocal){
    document.getElementById('infoform').innerHTML = "<p>Désolé, ce navigateur ne supporte pas l'API Web Storage du W3C.</p>";
    return;
}
```

Image 19 : Ajout condition dans la boucle d'initialisation

Cette condition est là au cas où il y aurait une erreur pendant l'initialisation. Dans ce cas-là, la fonction renverra un message d'erreur, « Désolé, ... ».

Ces lignes là (ci-dessous) servent à prendre ce qu'il y a dans les champs, et à le stocker sur un document. Ainsi, le prénom, nom, et code postale seront stockés, à condition qu'il y ait quelque chose d'écrit, et donc que la longueur de la chaîne de caractère soit différente de 0.

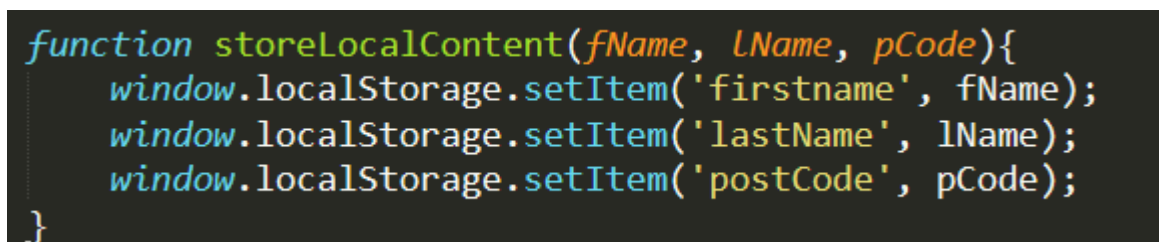
Autrement dit, c'est un Getters.



```
if(window.localStorage.length != 0){
    document.getElementById('firstname').value = window.localStorage.getItem('firstname');
    document.getElementById('lastName').value = window.localStorage.getItem('lastName');
    document.getElementById('postCode').value = window.localStorage.getItem('postCode');
}
```

Image 20 : Stockage des valeurs

Et nous avons donc nos Setters, qui sont appelés lors du clic sur le bouton « Enregistrer ». Ceux-ci prennent en paramètres les valeurs rentrés dans les champs pour les réécrire.



```
function storeLocalContent(fName, lName, pCode){
    window.localStorage.setItem('firstname', fName);
    window.localStorage.setItem('lastName', lName);
    window.localStorage.setItem('postCode', pCode);
}
```

Image 21 : Setters des données

On écrit le corps de la fonction `clearLocalContent()`, qui va remettre toutes les variables comme à l'origine, c'est-à-dire, rien.

```
function clearLocalContent() {  
    window.localStorage.setItem('firstname', "");  
    window.localStorage.setItem('lastName', "");  
    window.localStorage.setItem('postCode', "");  
}
```

Image 22 : Fonction `clearLocalContent`

On teste le résultat avec un navigateur, on entre des données, on va sur une autre page Web. Et lorsque l'on revient, nos informations sont toujours là.

EXERCICE 5

On commence par récupérer notre fichier `geopositionnement.html`, qui à première vue, affiche les différentes informations liées à la géolocalisation, tel que la longitude, latitude, etc.

On ajoute dans la balise 'head', le lien vers la librairie Modernizr, et la balise nécessaire.

```
<!DOCTYPE html>  
<head>  
    <title>Exemple géopositionnement</title>  
    <meta name="viewport" content="width=device-width" />  
    <meta charset="UTF-8"/>  
    <html class="no-js">  
    <script src="Modernizr.js"></script>  
</head>  
<body>
```

Image 23 : Ajout du nécessaire pour l'utilisation de Modernizr

On crée un fichier javascript, `fichier.js`, et nous ajoutons le lien dans la balise head.

On place les lignes données dans le fichier Javascript, la fonction `GetLocation` sera appelée à la toute fin du programme.

On ajoute les lignes suivantes :

```
if(Modernizr.geolocation){  
    navigator.geolocation.getCurrentPosition(geoSuccess, geoError);  
}
```

Image 24 : Getters de position

Ces lignes de codes renverront la position souhaitée (fonction geoSuccess), et le cas échéant, une erreur (fonction geoError).

La fonction GeoSuccess, renvoie les informations telles que la longitude, altitude, etc. Ces informations seront affichées sur la page Web.

La fonction GeoError renvoie un message d'erreur selon le type d'erreur. Il y a 3 erreurs possibles, dues à la confidentialité, le temps de latence, ou encore la capacité à déterminer la position.

Je lance le programme, et la première chose qui apparaît est un pop-up me demandant si je veux autoriser le fichier à connaître ces informations :



Image 25 : Autorisation

J'accepte.

Exemple géopositionnement

Cet exemple illustre comment utiliser la fonction de géopositionnement du terminal mobile.

Données de position:

Longitude: 5.062656

Latitude: 47.3202688

Précision: 4221

Altitude:

Précision altitude: undefined

Cap:

Vitesse:

Distance de l'ESIREM:

Image 26 : Géopositionnement

On a donc accès à différentes informations, comme la longitude, la latitude, ou encore la précision (4221 chiffres après la virgule). Cependant, l'altitude n'a pas pu être trouvée, donc la précision altitude non plus. Il en va de même pour le Cap et la Vitesse.

La localisation de l'Esirem étant inconnue du programme, il est impossible de calculer la distance.

On entre la fonction calculDistance, et la fonction DegreesEnRadians.

On lance le programme et on obtient :

Distance de l'ESIREM: NaN

CONCLUSION

Ces exercices nous ont permis de nous familiariser avec différents langages informatiques, tels que le HTML5, le CSS3, mais aussi le Javascript. Nous avons pu mettre en place nos premières pages Web, requêtes, gérer l'affichage, mais aussi stocker des informations. Ces connaissances nous seront utiles et nécessaires dans la gestion et la création de sites Web.