JAVASCRIPT

2 – DOM – BOM – Boutons – Evénements – Formulaires - Animations

SOMMAIRE

Edition :novembre 2019

1 : DOM – DOCUMENT OBJECT MODELE

Installation des fichiers de tests

Dans le cours :

Les exemples de code sont présentés dans un chapitre en vert pomme !

Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript-02-01-DOM-Exemples.zip

Chargez ces fichiers et mettez-les :

Soit dans un JavaScript.

Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

1 - Présentation

Références

https://openclassrooms.com/courses/creez-des-pages-web-interactives-avec-javascript

https://openclassrooms.com/courses/creez-des-pages-web-interactives-avec-javascript/decouvrez-le-dom

http://www.w3schools.com/js/js_htmldom.asp

http://www.w3schools.com/jsref/dom_obj_document.asp

Arborescence d'une page HTML

Arbre

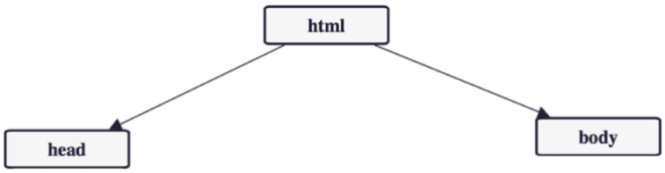
Les balises d'une page HTML forment une arborescence. On dit un **arbre**.

Un arbre est constitué de **nœuds**.

Tous les nœuds ont **0 ou n enfants et un seul parent**, sauf le premier nœud, **la racine**, qui **n'a pas de parent**.

Donc dans une page HTML, la racine c'est la balise <html>.

Elle a 2 enfants : <head> et <body> qui eux même peuvent avoir des enfants.



Arbre d'une page HTML

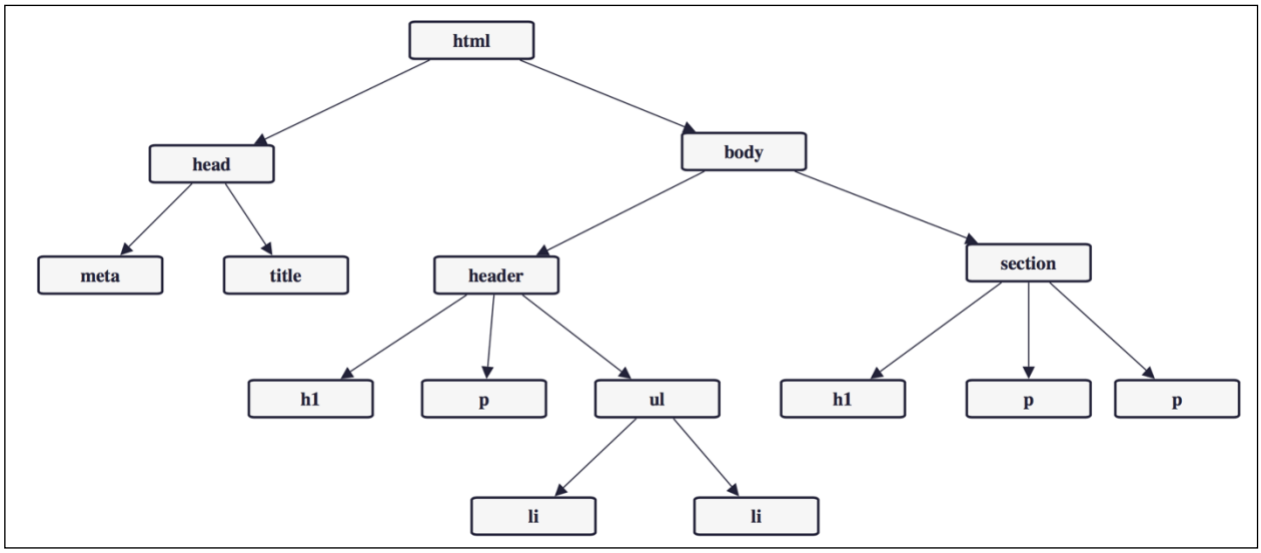
- ***Page HTML***

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

  <body>
     <header>
        <h1>Site</h1>
        <p>Mon paragraphe 1</p>
        <ul>
           <li>menu1</li>
           <li>menu2</li>
        </ul>
     </header>

     <section>
        <h1>Ma Section</h1>
        <p>Mon paragraphe 1</p>
        <p>Mon paragraphe 2</p>
     </section>
  </body>
</html>
```

➢ ***Arbre de la page HTML***



Structure du DOM

Présentation

DOM : Document Object Model.

Le DOM, c'est une bibliothèque, une API, proposant des classes avec leurs attributs et leurs méthodes permettent d'accéder à la structure d'une page HTML et permettent d'interagir avec elle.

Le DOM est standard pour tous les navigateurs (W3C en 1998).

Le DOM propose un objet appelé « document » comme variable globale pour tous les script JavaScript. L'objet « document » décrit la totalité de la page HTML sous la forme d'un arbre.

Chaque nœud de l'arbre correspond à une balise (ouvrante et fermante) ou au texte situé entre les balises ouvrantes et fermantes.

Chaque nœud de l'arbre correspond à un certain type d'objet avec ses attributs et ses méthodes permettant de le manipuler avec JavaScript.

Toute modification de l'objet « document » (ajout de balise, modification de contenu de balise, d'attributs de balise, etc.) est immédiatement traduite par une mise à jour de la page HTML dans le navigateur.

Les différents nœuds de l'arbre des balises

- ***Nœud « document »***

  C'est la racine de l'arbre. Elle au moins un enfant : la balise <html> mais aussi éventuellement des commentaires ou du texte brut.

  Ce nœud est une variable globale accessible directement dans le code JavaScript.

  On l'utilise quand on écrit : document.getElementById(« resultat »)

- ***Nœud ELEMENT***

  Toutes les balises ouvrantes et orphelines sont des nœuds (attributs compris) : Ce sont **les nœuds ELEMENT.**

  Ces nœuds peuvent avoir des enfants.

  Les balises fermantes ne sont pas enregistrées comme nœud.

- ***Nœud TEXTE***

  Le texte, entre deux balises est aussi un nœud. Ces nœuds ne peuvent pas avoir d'enfants. Ce sont **les nœuds TEXTE.**

  Attention, les espaces et les passages à la ligne entre deux balises forment un nœud TEXTE.

- ***Nœud COMMENT***

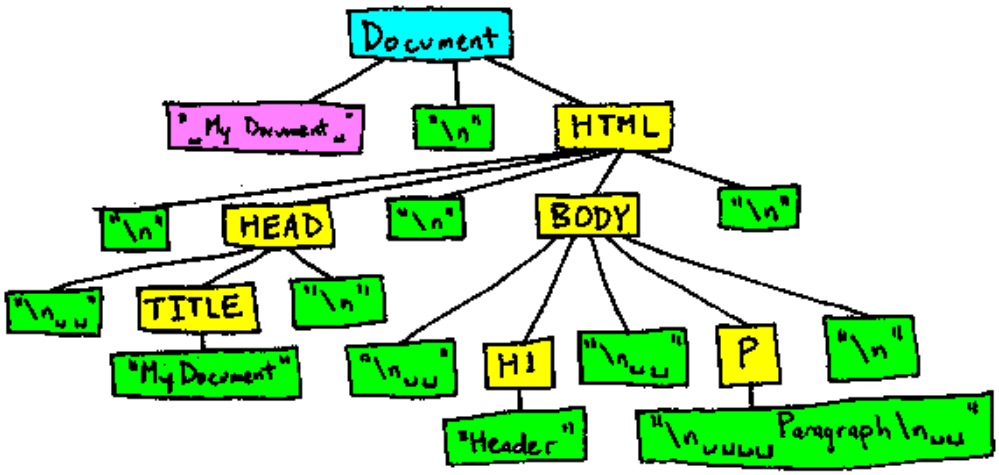
  Les commentaire HTML forment aussi un nœud. Ce sont **les nœuds COMMENT.**

Exemple

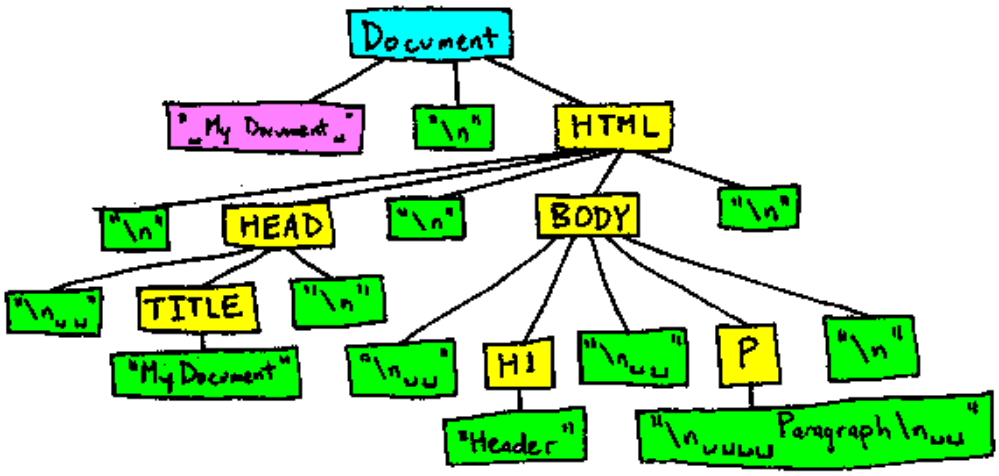
- ***Page HTML : les espaces sont matérialisés par des « . »***

```
<!--.My document.-->
<html>
<head>
..<title>My Document</title>
</head>
<body>
..<h1>Header</h1>
..<p>
....Paragraph
..</p>
</body>
</html>
```

- ***Arbre du DOM correspondant***



➢ ***Arbre du DOM correspondant***



➢ ***Commentaires***

- Le nœud document est en BLEU. Les nœuds ELEMENT sont en JAUNE.
- Les nœuds TEXT sont en VERT. Les nœuds COMMENT sont en ROSE.
- Entre le commentaire et <html>, on a un passage à la ligne (« \n »)
- Le premier enfant du <html> est un (« \n »), puis le <head>, puis un (« \n »), puis le <body> et encore un (« \n ») avant le </html>
- Pour n'avoir que 2 enfants dans le <body> (<h1> et <p>) , il faudrait tout mettre sur la même ligne, de <body> à </body>

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Whitespace_in_the_DOM

Exercice 1 : faire l'arbre du DOM du code HTML suivant :

- ***code HTML :***

```
<!DOCTYPE html>
<html lang="fr">
<head>
...<meta charset="utf-8">
...<title>Rectangle</title>
</head>

<body>
  <h1>Calculs sur le rectangle </h1>
  <!--.bouton dans un paragraphe.-->
  <p>Cliquez sur le bouton pour lancer la fonction :
  ...<button onclick="rectangle();">
  ......Lancer la fonction
  ...</button>
  </p>
  <p> Présentation de ma liste </p>
  <ul id="maliste">>
  ...<li> bla bla </li>
  ...<li> bli bli </li>
  </ul>

<script src="rectangle.js"></script>
</body>
</html>
```

Les espaces sont matérialisés par des « . »

2 - L'objet « document » et l'accès aux nœuds

L'objet « document » : la racine de l'arbre

Présentation

En JavaScript, **l'objet « document » est une variable globale** qui permet d'accéder à la racine de l'arbre (**la racine du DOM**).

Cet objet contient tous les éléments de la page web. Il donne accès à des méthodes et à des attributs

Attributs attachées

De nombreuses attributs sont accessibles à partir du document.

- **document.body** : Returns the <body> element
- **document.head** : Returns the <head> element
- **document.images** : Returns all <img> elements
- etc.

➢ ***Tous les attributs (en bas de la page) :***

http://www.w3schools.com/js/js_htmldom_document.asp

Méthode attachées

De nombreuses méthodes (fonctions) sont accessibles à partir du document.

- **document.write() :** ajouter un nœud et écrire à la fin du DOM.
- **document.getElementById(id)** : récupérer une balise par son id
- **document.getElementsByTagName(balise) :** récupérer les balises par leur nom
- etc.

➢ ***Toutes les méthodes par catégories :***

http://www.w3schools.com/js/js_htmldom_document.asp

➢ ***Toutes les méthodes et attributs par ordre alphabétique :***

http://www.w3schools.com/jsref/dom_obj_document.asp

Les objets « ELEMENT» : chaque nœud de l'arbre

Présentation

Un objet ELEMENT ou nœud correspond à **une balise HTML**.

Il a des **caractéristiques générales** qui sont celles de tous le nœud : **nom, contenu, parent, enfants, etc**.

Il aura des **caractéristiques particulières** qui seront les **attributs HTML de la balise** qui seront définis ou pas selon le contexte.

Attributs généraux attachés

Un objet ELEMENT ou nœud a plusieurs caractéristiques dont :

- **nodeName :** le nom du nœud. Il s'agit du nom de la balise ou #text pour du texte
  https://www.w3schools.com/jsref/prop_node_nodename.asp
- **nodeType** : qui peut prendre deux valeurs : Node.ELEMENT_NODE pour les balises et Node.TEXT_NODE pour le texte :
- **textContent :** le contenu texte (pas les balises) dans la balise :
  https://www.w3schools.com/jsref/prop_node_textcontent.asp
- **innerHTML :** contenu HTML complet de la balise avec les balises incluses.
  https://www.w3schools.com/jsref/prop_html_innerhtml.asp
- **childNodes** : collection ordonnée de nœuds (un genre de tableau) : les enfants dans l'ordre où ils apparaissent dans la page.
- **parentNode** : nœud parent
- **firstChild**, **lastChild**, etc. : autres attributs pour naviguer entre les objets du DOM.
  https://www.w3schools.com/jsref/prop_node_nodetype.asp

➢ ***Tous les attributs***

https://developer.mozilla.org/fr/docs/Web/API/Node.

Attributs particuliers attachés

- **id :** la valeur de l'attribut id
- **classList :** la liste des valeurs des attributs class
- **href :** la valeur de l'attribut href
- **src :** la valeur de l'attribut src

Les attributs sont définis uniquement s'il existe dans la page (cf. principes de programmation objet en JavaScript : on peut ajouter des attributs dynamiquement).

Méthode attachées

➢ ***Méthodes attachées***

De nombreuses méthodes (fonctions) sont accessibles à partir du noeud.

On retrouve en partie **celles qu'on avait sur l'objet document**.

Elles seront **détaillées dans les chapitres suivants**.

➢ ***Toutes les méthodes par ordre alphabétique :***

http://www.w3schools.com/jsref/dom_obj_all.asp

Accès aux nœuds

Méthodes d'accès aux nœuds (aux balises)

- ***Accès par id CSS : getElementById()***

```
noeud = document.getElementById("antiques"));
```

- ***Accès par balise (tag) : document.getElementsByTagName()***

```
lesNoeuds =document.getElementsByTagName("h2");//Tous les h2
```

- ***Accès par balise (tag) : element.getElementsByTagName()***

```
noeud = document.getElementById("nouvelles");
lesNoeuds = noeud.getElementsByTagName("li"); // Tous les li de
nouvelles
```

- ***Accès par classe CSS : getElementsByClassName()***

```
lesNoeuds = document.getElementsByClassName("merveilles");
```

- ***Accès par n'importe quel selecteur CSS : querySelectorAll()***

```
// Tous les paragraphes à l'intérieur d'un "#contenu" (id)
lesNoeuds = document.querySelectorAll("#contenu p");
```

- ***Accès au premier d'un selecteur CSS : querySelector()***

```
noeud = document.querySelector("p"); // un seul : le premier
```

- ***Accès direct***

```
noeud = document.body.childNodes[5];
```

Ici, on accède au 6ème enfant de <body>, les enfants pouvant être des balises, du texte ou des commentaires. C'est donc à éviter d'autant que les passages à la lignes sont des nœuds texte !

Bilan

- ***Un seul***

  Par identifiant : getElementById

  Le premier : querySelector

- ***Plusieurs***

  Par balise : getElementsByTagName

  Par classe : getElementsByClassName

  Par n'importe quel selecteur : querySelectorAll

Exercice 2 : donnez des instructions d'accès aux balises

Dans la page de l'exercice 1 (remise ci-dessous) :

- Donnez une ou plusieurs instructions permettant d'accéder à la balise h1
- Donnez une ou plusieurs instructions permettant d'accéder aux balises p
- Donnez une ou plusieurs instructions permettant d'accéder à la balise ul
- Donnez une ou plusieurs instructions permettant d'accéder aux balises li

code HTML :

```
<!DOCTYPE html>
<html lang="fr">
<head>
...<meta charset="utf-8">
...<title>Exercice 3</title>
</head>

<body>
  <h1>Calculs sur le rectangle </h1>
  <!--.bouton dans un paragraphe.-->
  <p>Cliquez sur le bouton pour lancer la fonction :
  ...<button onclick="rectangle();">
  ......Lancer la fonction
  ...</button>
  </p>
  <p> Présentation de ma liste </p>
  <ul id="maliste">>
  ...<li> bla bla </li>
  ...<li> bli bli </li>
  </ul>

<script src="script.js"></script>
</body>
</html>
```

Exemple 1 et Exercice 3 - DOM-Acces-aux-nœuds – Regardez le code JS

La page ci-dessous est associée à du code JavaScript affiché en console.

On peut regarder le code HTML et JS et les résultats en console.

Les codes des pages suivantes montrent les codes d'accès aux différents éléments de la page.

Les sept merveilles du monde

Connaissez-vous les merveilles du monde ?

Merveilles du monde antique

Cette liste nous vient de l'Antiquité.

- La pyramide de Khéops
- Les jardins suspendus de Babylone
- La statue de Zeus
- Le temple d'Artémis
- Le mausolée d'Halicarnasse
- Le Colosse de Rhodes
- Le phare d'Alexandrie

Nouvelles merveilles du monde

Il existe des centaines de merveilles référencées au patrimoine mondial par l'Unesco.

Une liste de 7 merveilles a été établie en 2009 à la suite d'un vote par Internet sur une iniative privée.

- La Grande Muraille de Chine
- Pétra
- Le Christ du Corcovado
- Machu Picchu
- Chichén Itzá
- Le Colisée
- Le Taj Mahal

Références

- Merveilles antiques : Wikipedia - Universalis
- Nouvelles merveilles : Wikipedia - Unesco

Accès par id CSS : getElementById()

On peut récupérer Le nœud, unique, pour un id donné : ici l'id « antiques»

```
console.log(document.getElementById("antiques"));
```

getElement : sans « s » ! avec un id, il doit n'y en avoir qu'un seul !

On peut alors afficher le innerHTML : le code HTML complet du nœud.

On peut aussi afficher le textCONTENT : le texte du nœud sans les balises.

```
console.log(document.getElementById("antiques").innerHTML);
console.log(document.getElementById("antiques").textContent);
```

Accès au premier d'un selecteur CSS : querySelector()

```
console.log(document.querySelector("p")); // un seul : le premier
```

Accès par balise (tag) : document.getElementsByTagName()

On peut récupérer tous les nœuds d'une même balise : tous les h2

```
var noeuds =document.getElementsByTagName("h2");//Tous les h2
```

On peut lister tous les noeuds

```
for (var i = 0; i < noeuds.length; i++) {
    console.log(noeuds[i]);
}
```

On peut accéder à chaque nœud par [ ] ou item()

```
console.log(noeuds [0]); // Affiche le premier titre h2
console.log(noeuds.item(0)); // Affiche le premier titre h2
console.log(noeuds.length); // Affiche 3 : il y a 3 h2
```

Accès par balise (tag) : element.getElementsByTagName()

On peut récupérer tous les nœuds enfants d'une même balise : tous les li d'un ul :

```
var noeud = document.getElementById("nouvelles");
var noeuds = noeud.getElementsByTagName("li"); // Tous les li de
nouvelles
```

On peut à nouveau lister tous les noeuds

```
for (var i = 0; i < noeuds.length; i++) {
    console.log(noeuds[i]);
}
```

Accès par classe CSS : getElementsByClassName()

On peut récupérer tous les nœuds d'une même « class » : ici la class « merveilles »

```
var merveillesElts =
document.getElementsByClassName("merveilles");
for (var i = 0; i < merveillesElts.length; i++) {
    console.log(merveillesElts[i]);
}
```

Accès par n'importe quel selecteur CSS : querySelectorAll()

On peut récupérer tous les nœuds pour un selecteur CSS donné : ici les « p »

```
// Tous les paragraphes
console.log(document.querySelectorAll("p").length); // Affiche 3
```

On peut préciser la recherche comme en CSS : ici les « p » dans #contenu

```
// Tous les paragraphes à l'intérieur d'un "#contenu" (id)
console.log(document.querySelectorAll("#contenu p").length); // 2
```

On peut récupérer tous les nœuds pour un selecteur CSS donné : ici les « .existe »

```
// Tous les ".existe" (class) équivent à getElementsByClassName
console.log(document.querySelectorAll(".existe").length); //
Affiche 8
```

Accès aux enfants d'un noeud

```
console.log(document.getElementById("antiques").childNodes)
```

On a tous les enfants de antique : le problème c'est que le passage à la ligne est un enfant !

```
console.log(document.getElementById("antiques").childNodes[0])
```

C'est le passage à la ligne

```
console.log(document.getElementById("antiques").childNodes[1])
```

C'est le premier li.

3 - Accéder aux informations des nœuds

Présentation

Une fois qu'on a récupéré un nœud (une balise), on peut accéder à ses informations, particulièrement :

- **le contenu de la balise** (ce qui se trouve entre l'ouvrante et la fermante)
- **la valeur d'un attribut**

Accès au contenu de la balise

innerHTML : tout le contenu HTML du nœud (avec les balises à l'intérieur du nœud)

```
console.log(document.getElementById("antiques").innerHTML);
```

Le innerHTML d'un <ul>, ce sont les <li>, les textes et les

textContent : tout le text sans les balises à l'intérieur du nœud

```
console.log(document.getElementById("antiques").textContent);
```

Le textContent d'un <ul>, ce sont uniquement les textes des

Accès aux valeurs des attributs

getAttribute()

ici on recupère la valeur d'un href

```
console.log(document.querySelector("a").getAttribute("href"));
```

.href, .id, … : accès direct aux attributs d'une balise

```
console.log(document.querySelector("a").href);
baliseA = document.querySelector("a")
console.log(baliseA.href);
```

hasAttribute : vérifier si un nœud a un attribut

```
if (document.querySelector("a").hasAttribute("target")) {
```

.classList : accès à toutes les « class » d'une balise

```
lesNoeuds = document.getElementById("antiques").classList;
```

contains() : vérifier si une classe est présente dans les classes d'un nœud

```
if
(document.getElementById("antiques").classList.contains("merveill
e")) {
```

Autres possibilités

Il y a d'autres possibilités d'accès.

Il faut chercher dans la documentation en cas de besoin :

http://www.w3schools.com/jsref/dom_obj_all.asp

http://www.w3schools.com/jsref/dom_obj_document.asp

https://developer.mozilla.org/fr/docs/Web/API/Node.

Exemple 2 et Exercice 4 - DOM-Acces-aux-infos-des-nœuds – Regardez le code JS

On repart de la page des 7 merveilles de l'exercice 1.

On peut regarder le code HTML et JS et les résultats en console.

Les codes ci-dessous montrent les codes d'accès au contenu et les codes d'accès aux attributs.

Accès au contenu HTML du nœud (avec les balises à l'intérieur du nœud) : innerHTML

```
// L'unique id antiques (qui est dans un ul)
console.log(document.getElementById("antiques"));

console.log(document.getElementById("antiques").innerHTML);
```

Accès au texte sans les balises à l'intérieur du nœud : textContent

```
// L'unique id antiques (qui est dans un ul)
console.log(document.getElementById("antiques"));

console.log(document.getElementById("antiques").textContent);
```

Accès à la la valeur d'un attribut

Pour récupérer la valeur d'un attribut, il faut sélectionner un nœud unique : querySelector ou getElementById

- ***getAttribute()***

```
console.log(document.querySelector("a").getAttribute("href"));
```

- ***accès direct : .href, .id,***

```
console.log(document.querySelector("a").href);
```

Modification de la la valeur d'un attribut

Exemple avec href :

```
document.querySelector("a").href = ”http://google.fr”;
```

Exemple avec id : ça crée l'attribut s'il n'existait pas.

```
document.querySelector("a").id = ”mon_a”;
```

Accès aux « class »

- ***Récupérer les classes dans un nœud : classList***

```
lesNoeuds = document.getElementById("antiques").classList;
```

On récupère la ou les class dans le nœud de l'id antiques

- ***Vérifier si une classe est présente dans les classes d'un nœud : contains()***

```
if
(document.getElementById("antiques").classList.contains("merveill
e")) {
```

Exercice 5 : accès aux nœuds, mise à jour de la page

Soit le code HTML suivant :

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="utf-8">
    <title>Langages</title>
</head>

<body>
    <h1>Quelques langages client</h1>
    <div id="client">
        <ul class="langages">
            <li id="html">HTML</li>
            <li id="css">CSS</li>
            <li id="javascript"><a
href="http://www.w3schools.com/jsref/dom_obj_all.asp"> JavaScript
</a></li>
            <li id="jquery">JQuery</li>
        </ul>
    </div>

    <h1>Quelques langages serveur</h1>
    <div id="serveur">
        <ul class="langages">
            <li id="java">Java</li>
            <li id="php">PHP</li>
            <li id="asp">C#</li>
            <li id="python">Python#</li>
            <li id="ruby">Ruby#</li>
        </ul>
    </div>
<script src="script.js"></script>
</body>
</html>
```

Ecrire un code JavaScript permettant de :

- Affichez le nœud d'id serveur
- Affichez le nœud d'id javascript
- Affichez les nœuds h1
- Affichez le innerHTML des class langages
- Affichez le texte des class langages
- Affichez le innerHTML des class langages seulement pour le client
- Affichez le href de l'id javascript
- Remplacez le href de l'id javascript par « bliaudet.free.fr » et affichez le nouvel href.

On écrira quelque chose commençant par :

```
console.log("Afficher le nœud d'id serveur");
etc.
```

Le code de la page HTML est avec les exemples, nommé « exercice5 ».

4 - Modifier la structure de la page

Modification des attributs

Modification du texte d'une balise

- ***innerHTML = …***

```
// ajouter un <li> à la liste : innerHTML += '...';
document.getElementById("langages").innerHTML += '<li
id="c">C</li>';

// vider le contenu HTML : innerHTML = "";
//document.getElementById("langages").innerHTML = "";
```

- ***textContent = …***

```
document.querySelector("h1").textContent += " de programmation";
```

Modification des attributs d'une balise

- ***setAttribute(key, value)***

  Via une méthode :

```
// ajout d'un id="titre"
document.querySelector("h1").setAttribute("id", "titre");
```

- ***nœud.key = value***

  Directement par l'attribut :

```
document.querySelector("h1").id="titre";
```

Modification du style d'une balise : attribut style

- ***nœud.style.attribut_de_style = value***

```
var noeud = document.querySelector("p");
noeud.style.color = "red";
noeud.style.margin = "50px";

noeud.style.fontFamily = "Arial";
noeud.style.backgroundColor = "black";
```

- ***Correspondance des nom d'attribut entre CSS – JavaScript***

  La correspondance entre CSS et JS est assez intuitive, mais pas toujours !

  En CSS on peut avoir des tirets : background-color.

  En JavaScript, la syntaxe est Camel Case : backgroundColor.

  https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference

Création de nouveaux nœuds

Création de nœud (ou élément)

- ***Création et définition du nouvel élément : createElement ()***

```
var noeud = document.createElement("li"); // <li></li>
noeud.id = "python"; // <li id="python"></li>
noeud.textContent = "Python"; // <li id="python">Python</li>
console.log('nodeName:'+noeud.nodeName + '  -id:' + noeud.id+'  -
textContent:', noeud.textContent);
```

Insertion de nœud (ou élément)

- ***Insertion d'un nouvel élément à la fin de la liste : appendChild ()***

On récupère un nœud, on ajoute le nouveau nœud à la fin de la liste de ses enfants.

```
document.getElementById("langages").appendChild(noeud);
```

- ***Insertion d'un nouvel élément avant un autre élément : insertBefore ()***

Syntaxe :

```
noeudParent.insertBefore (
      nœud à insérer,
      nœud avant lequel insérer
)
```

Exemple :

```
document.getElementById("langages").insertBefore(noeud,
    document.getElementById("php"));
```

Avec insertBefore, on pourra insérer en début de liste :

```
noeudParent.insertBefore (
      nœud à insérer, noeudParent.fistChild
)
```

Insertion de codeHTML : insertAdjacentHTML

- ***Syntaxe***

  On peut insérer du code HTML à une certaine position à partir d'un nœud.

```
noeudParent.insertAdjacentHTML(position, codeHTML) ;
```

- ***4 positions possibles, très pratiques :***
  - **beforebegin** : avant le nœud.
  - **afterbegin** : avant le premier enfant du nœud.
  - **beforeend** : après le dernier enfant du nœud.
  - **afterend** : après le nœud.

- ***Exemple***

  On peut aussi insérer directement du code HTML.

  Exemple :

```
document.getElementById('langages').insertAdjacentHTML(
      "afterBegin",
     '<li id="javascript">JavaScript</li>');
```

Remplacer un nœud (ou élément)

noeudParent.replaceChild(nouveauNoeud, ancienNoeud) ;

```
document.getElementById("langages").replaceChild(
    nouveauNoeud,
    document.getElementById("perl"));
```

Supprimer un nœud (ou élément)

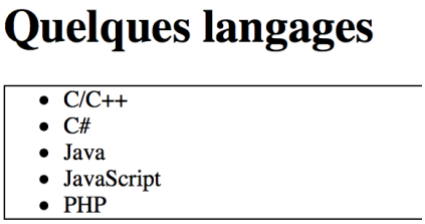
noeudParent.removeChild(noeudASupprimer) ;

```
document.getElementById("langages").removeChild(
    document.getElementById("bash"));
```

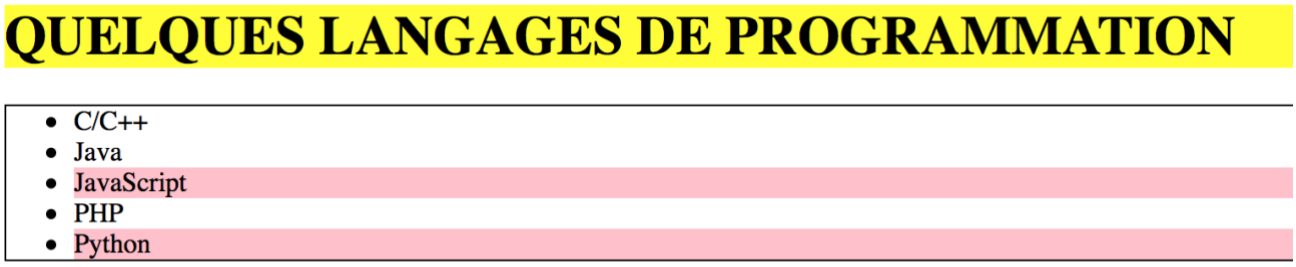
Exemple 3 – DOM - Modification des nœuds et du style

L'exemple propose le HTML sans JavaScript et le HTML avec JavaScript

- ***Sans JS***



- ***Avec JS***



- ***Bilan***

On a coloré le titre et complété le texte du titre.

On a supprimé le C#, ajouté le Python à la fin, coloré 2 langages.

Exemple 4 – DOM - Modification des nœuds et du style

L'exemple propose le HTML sans JavaScript et le HTML avec JavaScript

- ***Sans JS***

Quelques langages

- C/C++
- Java
- JavaScript
- PHP

- ***Avec JS***

Quelques langages

- HTML
- C/C++
- Java
- JavaScript
- PHP
- Python
- JQuery

- ***Bilan***

On a rajouté le HTML au début, le Python et le JQuery à la fin, le tout coloré.

Style HTML et style CSS – Exemple 5 -DOM-getComputedStyle

Principe

```
noeud.style
```

permet d' accéder au **style défini dans le HTML et le JavaScript**, mais pas dans le CSS.

```
getComputedStyle(noeud)
```

permet d'accéder au **style défini dans le HTML, le CSS et le JavaScript.**

On utilisera donc plutôt la deuxième écriture.

Exemple 5 : Accès au style de la page

On peut récupérer toutes les informations du style de la page.

Attention, c'est le style est celui défini dans la page et par le JavaScript.

Attention ! Le style défini par le CSS n'est pas pris en compte.

```
var noeuds = document.getElementsByTagName("p");
console.log(noeuds[0].style.color); // Affiche "red"
console.log(noeuds[1].style.color); // Affiche "green"
console.log(noeuds[2].style.color); // N'affiche rien car le
style est dans le CSS
```

Exemple 5 : Accès au style CSS : fonction getComputedStyle

La fonction renvoie un objet style correspondant au style « computé » d'un nœud.

Elle a ses propres attributs.

```
var style = getComputedStyle(document.getElementById("para"));
console.log(style.fontStyle); // Affiche "italic"
console.log(style.color); // Affiche bleue en RGB : 0.0.255
```

5 - Parcours complet du DOM

Parcours complet du DOM – Etape 1 – Exemples 6 et 7 – difficiles !

On peut écrire des fonctions qui permettent de parcourir entièrement le DOM et d'afficher son contenu dans la console.

Ces fonctions sont complexes à comprendre et à utiliser.

Elles sont présentées à la fin du chapitre.

Exemple 6 - DOM-html-head-body

Le code suivant permet d'afficher les enfants des balises HTML, HEAD et BODY.

On le teste avec un petit fichier HTML, puis un plus gros.

```
console.log("------------BALISE HTML-----------");
afficherInfos(document.documentElement);

console.log("------------BALISE HEAD-----------");
afficherInfos(document.head);

console.log("------------BALISE BODY-----------");
afficherInfos(document.body);

function afficherInfos(noeud){
    console.log('NodeName : '+noeud.nodeName);
    console.log('Les childNodes :')
    afficherLesNoeuds(noeud.childNodes);
}

function afficherLesNoeuds(noeuds){
    for (var i = 0; i < noeuds.length; i++) {
        console.log(noeuds[i].parentNode.nodeName+'['+i+'] : '
          +noeuds[i].nodeName);
    }
}
```

Exemple 7 -DOM-html-head-body-noeudIgnoralbe

- ***Principes***

  Cette version permet de ne pas afficher les nœuds sans texte.

  On trouve les fonctions utilisées -> ici

  On se dote de la fonction « noeudIgnorable » pour savoir si on peut ignorer le nœud.

  On utilise les fonctionnalités RegExp (expression régulière).

- ***Fonction noeudIgnorable***

```
function noeudIgnorable(noeud) {
  if ( noeud.nodeType == Node.COMMENT_NODE) return true;
  if ( (noeud.nodeType == Node.TEXT_NODE) && noeudVide(noeud) )
     return true;
  return false;}
```

  Node.COMMENT_NODE : commentaires

  Node. TEXT_NODE: texte. Si c'est du texte et que c'est vide.

- ***Fonction afficherNoeudsEnfants()***

  Il faut tester si le nœud n'est pas vide avant de l'afficher.

  On gère un 2ème compteur, j, pour que les indices affichés avancent de 1 en 1.

```
function afficherLesNoeuds(noeuds){
    for (var i=0, j=0; i < noeuds.length; i++) {
        if(!noeudIgnorable(noeuds[i])) {
            console.log(noeuds[i].parentNode.nodeName+'['+j+'] :'
              +noeuds[i].nodeName);
            j++;
        }
    }
}
```

Expression régulière : quelques explications -> ici

- ***Fonction noeudVide()***

```
function noeudVide(noeud) {
  if ( /[^\t\n\r ]/.test(noeud.data) ) return false;
  return true;}
```

  On vérifie si le « texte » est conforme à l'expression régulière /[^\t\n\r]/

  Méthode test() : https://www.w3schools.com/jsref/jsref_regexp_test.asp

- ***Principes de construction d'expressions régulières***

  https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/RegExp

  On met l'expression entre / /

  [^] : On a une correspondance pour tout ce qui n'est pas compris dans les crochets. Donc si on a que ce qui est entre les crochets, ça renvoie faux.

  \t : tabulation

  \n : passage à la ligne

  \r : retour chariot

Parcours complet du DOM – Etape 2 – Exemple 8 – difficile

On peut écrire un algorithme qui parcourt tous les enfants à partir d'un nœud.

C'est un algorithme récursif classique de parcours d'arbre.

Exemple 8 -DOM-tous les noeuds

```
// noeud    : noeud à afficher
// cpt      : compteur du numéro de l'enfant à afficher
// decalage : pour indenter l'affichage
function afficherArbre(noeud, cpt, decalage) {
  // on saute les noeuds texte pour alléger
    if(noeud.nodeType==Node.TEXT_NODE) return;
    console.log(decalage+noeud.parentNode.nodeName+'['+cpt+'] : '
       +noeud.nodeName);
    for (var i=0, j=0; i < noeud.childNodes.length; i++) {
        if(!is_ignorable(noeud.childNodes[i])) {
            afficherArbre(noeud.childNodes[i],j++,decalage+'  ');
        }
    }
}

afficherArbre(document.documentElement,0,'');
```

Exercice : afficher l'arbre de l'exmple 8 dans une page HTML

2 : BOM – BROWSER OBJECT MODELE

Installation des fichiers de tests

Dans le cours :

Les exemples de code sont présentés dans un chapitre en vert pomme !

Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript-02-02-BOM-Exemples.zip

Chargez ces fichiers et mettez-les :

Soit dans un JavaScript.

Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

BOM : Browser Oject Model

https://www.w3schools.com/js/js_window.asp

Exemple 1 - Introduction au BOM

- ***Code HTML***

```
<!-- myFunction est appelée au chargement et à chaque resize -->
<body onload="myFunction()" onresize="myFunction()">
  <h4 id="largeur"></h4>
  <h1>Test du DOM</h1>
  <h2>Affichage de la taille de la fenêtre en haut à droite</h2>

  <!-- pour afficher la taile de l'éran -->
  <script src="script.js"></script>
</body>
```

Dans la balise body : onload=myFunction et onresize=myFunction().

myFunction est appelée au chargement de la page (onload) et à chaque changement de taille (onresize).

C'est de la programmation événementielle.

- ***Code JS***

```
function myFunction() {
    //1 : Récupération de la taille de la fenêtre

    var largeur=window.innerWidth;

    //2 : En fonction de la taille, détermination du type d'écran

    var typeEcran='';
    if(largeur<768){
        typeEcran='smartphone - xs';
    }
    else if(largeur<992){
        typeEcran='tablette - sm';
    }
    else if(largeur<1200){
        typeEcran='ordinateur portable - md';
    }
    else {
        typeEcran='ecran fixe - lg';
    }

    //3 : Affichage du message dans la page HTML
    document.getElementById("largeur").innerHTML = typeEcran + "
: " + largeur;
}
```

1 : L'objet « window » permet d'accéder à des informations sur la fenêtre du navigateur.

2 : A partir de la largeur on définit le message à afficher

3 : La 3ème étape fait intervenir le DOM : on récupère la balise HTML d'id « largeur » et on modifie son contenu via l'attribut « innerHTML ».

Présentation

L'objet window

https://www.w3schools.com/jsref/obj_window.asp

https://developer.mozilla.org/fr/docs/Web/API/Window

Lorsqu'un code JavaScript s'exécute dans un navigateur, l'objet **window représente la fenêtre du navigateur**.

Il contient des attributs dont certains sont eux-mêmes des classes :

- **screen** : donne des informations sur la fenêtre
- **document** : correspond à l'objet document du DOM
- **navigator** : donne des informations sur le navigateur
- **location** : donne des informations sur le fichier

window n'est pas standard : le fonctionnement est variable selon les navigateurs.

L'attribut navigator

« navigator » un **attribut de l'objet window.**

Il contient les attributs appCodeName, appName et appVersion, etc. qui donnent des informations sur l'OS et le navigateur.

En console, on peut écrire :

```
window.navigator
// ou
window.navigator.appVersion
```

L'attribut location

« location» un **attribut de l'objet window.**

Il contient les attributs qui donnent des informations sur l'OS et le navigateur.

« **href** » est un attribut qui correspond au **fichier de la page**. Si on le change, on change la page.

En console, on peut écrire :

```
window.location
// ou
window.location.href
```

L'attribut screen

« screen» un **attribut de l'objet window.**

Il contient les attributs comme **height** et **width** qui donnent des informations sur l'écran (pas la fenêtre).

En console, on peut écrire :

```
window.screen
// ou
window.screen.width
```

L'objet document

C'est un **attribut de l'objet window.**

https://developer.mozilla.org/fr/docs/Web/API/Window/document

L'attribut document correspond à l'objet « document » du DOM.

On peut écrire :

```
doc=window.document;
doc.write("Bonjour !!!");
noeud=doc.getElementById("monId");
```

Exemple 2-BOM-Tests : exemples d'attributs et de methodes

https://developer.mozilla.org/fr/docs/Web/API/Window

https://www.w3schools.com/jsref/obj_window.asp

innerWidth

var largeur=window.innerWidth;

location.href

L'attribut location.href contient la page en cours. On peut la changer

```
<button onclick="window.location.href='http://bliaudet.free.fr'">
  Click Me
</button>
```

open()

La méthode open ouvre une nouvelle page dans un nouvel onglet.

```
<button onclick="window.open('http://bliaudet.free.fr');">
  Click Me
</button>
```

close()

On peut fermet une fenêtre ouverte avec open() avec la méthode close() :

https://www.w3schools.com/jsref/met_win_close.asp

alert()

alert() ou window.alert() sont équivalent : pas de confirmation

Exemple 3-BOM-propagation : confirmation de fermeture de la fenêtre

Principes de l'exemple

1-pageDeBase : propose 3 liens vers un site externes. Ce sont les même lien avec le même comportement.

2-pageDeBaseConfirmation : ajoute au cas précédent une demande de confirmation. C'est la même pour tous les liens.

3-pageDeBaseStopPropagation : bloque un lien, retire la demande de confirmation sur un autre.

2-pageDeBaseConfirmation – évènement beforeunload

On ajoute un événement beforeunload sur l'objet window :

```
// ajout d'un listener niveau window : le navigateur
window.addEventListener("beforeunload", function (e) {
  // console.log
  console.log("beforeunload");
  console.log(e);

  // on définit un message qui n'apparaît nul part !
  var message = "On est bien ici !";

  // on set returnValue :
  // ça provoque une demande de confirmation (standard)
  e.returnValue = message;

  // on return le message :
  // ça provoque une demande de conf. dans certains navigateur
  return message;
});
```

Plusieurs remarques :

- On affiche « e », l'événement, dans le console.log. On peut alors visualiser tous les attributs de e. C'est un objet de la classe BeforeUnloadEvent
- Le fonctionnement est un peu aléatoire ! C'est un événement du BOM : non standard

  En théorie, c'est la modification de l'attribut returnValue de l'objet Event qui suspend la fermeture de la page et provoque l'apparition d'une boîte de dialogue de confirmation affichant la valeur de cet attribut.

  Cependant, certains navigateurs se basent sur la valeur de retour de la fonction qui gère l'événement plutôt que sur l'attribut returnValue (situation 2015).

  On peut associer les deux techniques pour être poly-navigateurs.

  A noter que le returnValue n'est pas le message qui apparaît dans la fenêtre de confirmation (safari, firefox, chrome).

Propagation d'événement : e.stopPropagation

Principe de la propagation par défaut

Les événements se propagent depuis le nœud de prise en compte jusqu'à la racine (en remontant par les parents).

```
Si un bouton est dans un paragraphe
  Si le paragraphe est dans la page // toujours !
     Si le bouton, le paragraphe et la page ont un listener click
        Si on clic ALORS
           les trois fonctions des listener sont éxécutés :
              Listerner du bouton
              Listerner du bouton paragraphe
              Listerner de la page
```

Exemple 3-BOM-propagation : suite

On reprend l'exemple précédent.

3-pageDeBaseStopPropagation : bloque un lien, retire la demande de confirmation sur un autre.

Cet exemple va bloquer l'événement de niveau href et de niveau window.

3-pageDeBaseStopPropagation : blocage du lien

e.preventDefault(); bloque le comportement standard : le href

```
// Annulation du comportement standard, ici un href
document.getElementById("interdit1").addEventListener("click",
function (e) {
    alert("Le lien est bloqué");
    console.log("e.preventDefault()");
    console.log(e);
    e.preventDefault(); // blocage du lien
});
```

3-pageDeBaseStopPropagation : blocage de la demande de confirmation

e. stopPropagation(); bloque le comportement du nœud parent : ici la window.

Attention, ça ne marche pas avec tous les navigateurs (window, BOM, pas standard).

```
// Annulation du comportement du noeud parent : ici la window
document.getElementById("interdit2").addEventListener("click",
function (e) {
    alert("Le lien va être lancé sans vérification - ok Safari -
pas ok Firefox");
    console.log("e.stopPropagation()");
    console.log(e);
    e.stopPropagation(); // arrêt de la propagation
    // ne marche pas dans tous les navigateurs
    // ne marche pas dans Firefox
    // marche dans Safari
});
```

3 : BUTTON - EVENEMENT - FORMULAIRE

0 - Installation des fichiers de tests

Dans le cours :

Les exemples de code sont présentés dans un chapitre en vert pomme !

Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript_02_03_Bouton_Evenement_Formulaire

Chargez ces fichiers et mettez-les :

Soit dans un JavaScript.

Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

1 - Button HTML – onclick dans le HTML – Premiers événements

https://openclassrooms.com/courses/creez-des-pages-web-interactives-avec-javascript/reagissez-a-des-evenements-1

http://www.w3schools.com/tags/tag_button.asp

Principes

Le but de la gestion des événements est de faire réagir la page web aux actions de l'utilisateur, sans passer par le serveur.

C'est de la **programmation événementielle** : le programme (la page web) réagit à un événement déclenché par l'utilisateur : cliquer sur un bouton par exemple (button HTML).

On peut aussi réagir à n'importe quel événement : cliquer sur n'importe quelle balise, agrandir la fenêtre, taper au clavier, etc.

On va regarder le cas des boutons pour une première approche.

La balise <button > : Exemple 01 - Evt-Bouton 5 cas

Présentation

- ***Principes***

  La **balise button** sert à gérer un bouton, c'est-à-dire essentiellement **une action sur un clic**.

  - La balise **button** sert plutôt à **effectuer une tâche JS**. Elle peut aussi charge une autre page.
  - Un **a href** ou un **input submit** dans un formulaire servent à charger une autre page : donc potentiellement une tâche serveur (PHP, etc.).

- ***Syntaxe***

  La balise <button type=button onclick= « code JavaScript à définir »> réagit quand on clique dessus.

  C'est un événement « onclick » qui est dans le bouton.

- ***Exemple***

  https://www.w3schools.com/tags/tag_button.asp

```
<button type="button" onclick="alert('Hello world!')">
    Click Me!
</button>
```

- ***Remarque sur les boutons***

  Un « button » avec un onclick fait office de bouton déclencheur d'une action.

  « href » et « input » sont aussi des boutons déclencheur.

  http://www.w3schools.com/css/tryit.asp?filename=trycss_buttons_basic

- ***Toute balise peut être un bouton !***

  L'événement onclick="alert('Hello world!')" peut se mettre dans n'importe quelle balise.

  Dans l'exemple ci-dessus, on peut le mettre dans un « h2 » !

Variantes de syntaxe

- ***Présentation***

  On peut gérer l'événement de 5 façons :

  - Avec **du code JS** et par exemple une fonction JS prédéfinie : alert().
  - Avec **une fonction définie dans le onclick** et l'appel à cette fonction dans le onclick.
  - Avec **un appel à une fonction définie dans du code JS** : dans une balise script ou un fichier externe.
  - Avec **un onclick définie dans du code JS** : dans une balise script ou un fichier externe. Ensuite on va alors définir une fonction anonyme attachée à l'évenement.
  - Avec un **window.location.href** qui permet de faire un href : passer à une autre page. Le button remplace un <a href> classique.

Bonne pratique : séparez le HTML du JavaScript

La bonne pratique consiste à **éviter de mettre du JavaScript dans la page HTML**.

Mieux vaut que tout le code JS soit dans un script externe.

Pour faire ça, il faut créer les événements (onclick, etc.) avec du JavaScript et pas dans la page HTML. C'est ce qu'on va voir au chapitre suivant.

Exemple 02-Button-Image-changer-cacher

On peut facilement manipuler des images avec JS et les <button>

- ***HTML : button - this - .style.display - .src***

```
<h1>Changement d'images</h1>

<div>
  <img class="illustration" id="image1" src="images/img1.jpg"
height="200" />
  <img class="illustration" id="image2" src="images/img2.jpg"
height="200" onclick="this.style.display='none'"/>
</div>
<p><button onclick="changerImage();">changer la première
image</button></p>
<p>cliquer sur la deuxième image pour la faire disparaitre</p>
<p><button onclick="afficherImage();">Afficher l'image
cachée</button></p>

<script src="../js/cours.js"></script>
```

Ici on a 2 images et 2 boutons appelant 2 méthodes.

Et dans l'image2, on a directement un « onclick="this.style.display='none'" ».

Le « this » fait référence au nœud du DOM correspondant à la balise : ici une balise img.

Le display est un attribut CSS auquel on accède via le style. En le passant à 'none' on cache l'image (on la fait supprime de la page).

- ***JavaScript : style.display, src***

Une fonction très simple pour afficher l'image

```
function afficherImage () {
  console.log("Entrée dans afficherImage")
  document.getElementById('image2').style.display='';
}
```

Une fonction plus compliquée pour changer d'image

```
function changerImage () {
  image=document.getElementById('image1').src;
  image=document.getElementById('image1').src.substring(image.las
tIndexOf("/"));
  // On inverse la valeur de l'attribut src
  if(image=='/img1.jpg'){
     document.getElementById('image1').src = 'images/img2.jpg'
  }
  else{
     document.getElementById('image1').src = 'images/img1.jpg'
  }
}
```

Le problème est de récupérer le nom du fichier : le .src permet d'y accéder.

Mais c'est le nom avec le chemin complet : /usr/monprojet/vues/images/img1.jpg, par exemle.

La fonction lastIndexOf récupère la position de la dernière occurrence d'un caractère : ici le /

La fonction substring avec un seul entier récupère la fin de la chaine : /img1.jpg

A partir de là, on inverse le src.

Exemple03-Button-Image-double clic sans JS

Ici, on montre un usage le code JS directement dans le <button>

HTML : button et this

Pour chaque événement, onmouseover et onmouseout, on met plusieurs instructions séparées par des ; On choisit une image (src=) et un alt (alt=).

On accède à la balise en cours par un « this »/

```
<img
  id="image" src="images/img1.jpg" alt="leftovers" width="70"
  onmouseover="
      this.src='images/img2.jpg';this.width='150';this.alt='fargo'
  "
  onmouseout="

  this.src='images/img1.jpg';this.width='70';this.alt='leftovers'
  "
>
```

Ensuite, on a deux événements : click et dblclick sur un

Il modifie le contenu (innerHTML=) et l'image (src=) en récupérant la balise de l'image.

innerHTML permet de change tout le contenu de la balise, span en l'occurrence.

```
<li>
  L'image change quand on passe dessus avec la souris.<br>
</li>
<li
  onclick="
      this.innerHTML='click pour Fargo - dblclick pour Leftovers';
      document.getElementById('image').src='images/img2.jpg';
  "
  ondblclick="
      this.innerHTML='click pour Fargo - dblclick pour Leftovers';
      document.getElementById('image').src='images/img1.jpg';
  "
>
  click sur ce texte pour passer sur Fargo - dblclick pour The
Leftovers
</li>
```

Mot clé « this »

Le mot-clé this s'utilise en programmation objet pour faire référence à l'objet en cours.

Exemple 1 :

```
var eleve={
  nom:"toto",
  note:14,
  toString:function(){
     return "nom="+this.nom+" - note="+this.note;
     // this fait reference à élève
  },
}
```

Exemple 2 :

```
noeud=document.getSelector('li');
noeud.onmouseover=function(){
  this.style.backgroundColor='aqua';
  // this fait référence à noeud en tant qu'objet
}
```

Exemple 3 :

```
<img src="img/img.jpg" onclick="this.style.display='none'"/>
// this fait reference à img en tant qu'objet nœud dans le DOM
```

La fonction eval( )

La fonction eval prend une chaine de caractères en paramètre.
Elle l'évalue comme si c'était une expression.
Exemples
eval('2+2') vaut 4
eval('bouton1') vaut la balise button dont l'id est « bouton1 »

Il faut éviter d'utiliser cette fonction !
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/eval

2 - Les événements

Présentation

Principes

De très nombreux événements peuvent engendrer l'appel à une fonction JavaScript :

- onclick, ondblclick
- onkeypress
- onresize
- onload
- onscroll
- onmouseover, onmouseout,
- ect.

https://www.w3schools.com/jsref/dom_obj_event.asp

Notion de Listener : programmation événementielle

- ***Présentation***

  La gestion des événements relève de la **programmation événementielle**.

  Le navigateur est en attente d'un événement déclenché par l'utilisateur : cliquer sur un bouton par exemple.

  Quand le navigateur récupère un événement, il réagit.

- ***Listener***

  Chaque **élément en attente d'un événement** est rattaché à un « **écouteur d'événement** » : le **Listener**.

  Quand le navigateur lit un <button onclick="changerImage();"> il fait 2 choses :

  - il crée un Listener
  - il attache le Listener à la balise <button> du DOM.

  A partir de là, l'événement (le clic souris sur la balise) sera pris en compte.

- ***Technique***

  On va attacher des listener sur les balises avec la méthode **addEventListener**()

Création manuelle d'un listener - Exemple04-bouton_et_listener

Principes : addEventListener (« evenement », fonctionDeclenchée)

On peut attacher « manuellement » un listener à une balise.

L'intérêt est, entre autre, d'éviter de mettre du JavaScript dans son code HTML.

On va gérer « à la main » l'événement « onclick ».

Pour ça, on le supprime de la balise <button> puis on associe « à la main » un « Listener » de « click » sur le nœud bouton dans un programme JavaScript.

Fonction : addEventListener (« evenement », fonctionDeclenchée)

```
noeudBouton.addEventListener("click", buttonClic);
```

Avantages par rapport à l'attribution directe sur l'attribut onclick

On peut aussi écrire :

```
noeudBouton.onclick=nuttonClic();
```

Le « addEventListener » permet une gestion plus fine des événements (suppression, gestion de plusieurs événements, …).

https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener

supprimer un avénement : remove

On peut supprimer les événements à écouter.

Ce sont les mêmes paramètres qu'avec l'ajout.

Fonction : **removeEventListener (« evenement », fonctionDeclenchée)**

```
noeudBouton.removeEventListener("click", buttonClic);
```

Exemple04-bouton_et_listener

- ***Exemple : principes du code HTML***
  - Une balise <button> avec l'attribut « onclick » et la fonction buttonClic()

```
<button onclick=" buttonClic ()">button 1</button>
```

  - Une balise <button> sans attribut « onclick » : on gérera un listener.

```
<button id="bouton2">button 2 </button>
```

  - On peut mettre un listener (et aussi un événement) dans toutes les balises : ici dans <p>

```
<p id="bouton2"> button 3</p>
```

- ***Exemple : principes du code JavaScript***
  - On récupère la balise :

```
var bouton2 = document.getElementById("bouton2");
```

  - On accroche le listener à la balise. Il fait référence à une fonction : clic, sans parentèses

```
bouton2.addEventListener("click", buttonClic);
```

- ***Fonction anonyme***
  - On peut définir la fonction dans le addEventListener, et ne pas lui donner de nom :

```
var bouton3 = document.getElementById("bouton3");

bouton3.addEventListener("click", function () {
    alert("listener de fonction anonyme bien attaché");
    console.log("Clic anonyme");
});
```

Organisation des événements

Nom des événements

Le nom des événement dans les EventListener s'utilise sans « on » : **click** et pas **onclick**.

Tous les événements

Liste de tous les événements : http://www.w3schools.com/jsref/dom_obj_event.asp

Famille d'événement

Liste des familles d'événements : http://www.w3schools.com/jsref/dom_obj_event.asp

On peut signaler :

- **clavier :** appui ou relâchement d'une touche du clavier → Keyboard Events : **onkeypressed**, etc.
- **souris :** clic avec les différents boutons, appui ou relâchement d'un bouton de la souris, survol d'une zone avec la souris → Mouse Events : **onclick**, **ondblclick**, **onmouseover**, etc.
- **fenêtre :** chargement ou fermeture de la page, redimensionnement, défilement (scrolling) →Frame/Object Events : **onload**, **onresize**, **onbeforeunload**, **onscroll**, etc.
- **formulaire :** changement de cible de saisie (focus), envoi d'un formulaire → Form Events : **onfocus**, **onblur**, etc.

3 objets pour attacher des événements

- un noeud : n'importe quel nœud du DOM : n'importe quelle balise.
- document : c'est la racine du DOM
- window : c'est l'objet correspondant à la fenêtre du navigateur (ici). Fait partie du BOM : Browser Object Model, non standard (browser = navigateur).

Consulter un événement : paramètre 'event' de la fonction appelée : Exemple 05

Présentation

La fonction déclenchée par l'événement n'a pas de paramètre.

On toutefois peut en passer un : c'est l'événement déclencheur, quel que soit le nom donné.

C'est un objet de la classe Event : https://www.w3schools.com/jsref/obj_event.asp

En général, on l'appele « event ». Mais on pourrait mettre n'importe quel nom.

La classe Event propose plusieurs attributs. Notons particulièrement :

type : c'est le type d'événement, c'est-à-dire son nom

target : c'est la balise de l'événement

target.nodeName : à partir de target, on peut accéder au nom de la balise.

target.textContent : à partir de target, on peut accéder au contenu de la balise.

Exemple 05-consulter_evenement

```
var noeudBouton = document.getElementById("bouton");

noeudBouton.addEventListener("click", function (event) {
    console.log("Entrée dans la fonction du addEventListener !");
    console.log("Evénement : ");
    console.log(event);
    console.log("Evénement : "+event.type);
    console.log("target : ");
    console.log(event.target);
    console.log("Balise : "+event.target.nodeName);
    console.log("Contenu : "+event.target.textContent);
});
```

Exemple 06 - Exercice 1 Evénements du clavier : keypress, keyup, keydown

On affiche dans la console les lettres tapées au clavier

Code JavaScript

http://www.w3schools.com/jsref/dom_obj_event.asp

```
// Gestion de l'appui sur une touche du clavier produisant un
caractère
document.addEventListener("keypress", function (e) {
  console.log(e);
  console.log("Fonction anonyme: Evènement clavier : "
     + e.type
     + " / touche : " + String.fromCharCode(e.keyCode)
     + " / code de la touche : " + e.keyCode

   );

});
```

➢ ***Plusieurs remarques :***

- On affiche « e », l'événement, dans le console.log. On peut alors visualiser tous les attributs de e. C'est un objet de la classe KeyboardEvent.
- Lors d'un appui prolongé sur une touche, l'événement keydown est déclenché plusieurs fois.
- A noter que le keyCode d'un « e » minuscule sort en majuscule !
- Par contre, tout va bien avec les majuscules !

Exercice 1 : faites apparaître la saisie dans la page HTML

Reprenez de l'exemple précédent et mettez-le à jour pour qu'il permette de faire apparaître la saisie dans la page HTML

Exemple 07 : Événements de souris : mouseup & down, e.button, e.clientX & Y

On affiche dans la console log les caractéristiques du clic de souris

Code JavaScript

http://www.w3schools.com/jquery/jquery_ref_events.asp

```
// Renvoie le nom du bouton souris à partir de son code
function getBoutonSouris(code) {
    var bouton = "inconnu";
    switch (code) {
        case 0: bouton = "gauche"; break;
        case 1: bouton = "milieu"; break;
        case 2: bouton = "droit"; break;
    }
    return bouton;
}
// Affiche des informations sur un événement souris
function infosSouris(e) {
    console.log(e);
    console.log(
        "Evènement souris : " + e.type
        + " / bouton " + getBoutonSouris(e.button)
        + " / X : " + e.clientX + " / Y : " + e.clientY
    );
}
// Gestion du clic souris
document.addEventListener("mousedown", infosSouris); // enfoncer
document.addEventListener("mouseup", infosSouris); // relacher
document.addEventListener("click", infosSouris); // cliquer
```

➢ ***Plusieurs remarques :***

- On affiche « e », l'événement, dans le console.log. On peut alors visualiser tous les attributs de e. C'est un objet de la classe KeyboardEvent.
- Ordre de déclenchement des événements : down, up, click

3 - Manipuler des formulaires

Présentation

- JavaScript permet de manipuler le formulaire et ses données directement côté client, avant d'envoyer ces données vers un serveur externe.
- Ainsi, on peut avertir immédiatement l'utilisateur en cas de saisie erronée, ou bien lui proposer une liste de suggestions au fur et à mesure de sa frappe, et bien d'autres choses.
- Attention, malgré la vérification côté client, il faut toujours vérifier côté serveur : en effet, le JavaScript peut être bloqué côté client !
- On va regarder la structure d'un nœud formulaire dans le DOM.

Exemple 08-Formuaire-basique

Présentation

On a un formulaire avec 2 champs : Pseudo et Mot de passe.

On peut l'envoyer vers une autre page ou l'annuler.

Quand on saisit le Pseudo ou le mot de passe, certaines informations s'affichent pendant la saisie.

Si les informations ne sont pas valide, l'envoie sera bloqué.

On peut aussi annuler tout ce qui a été saisi.

Validation des données saisies

Le contrôle de validité a plusieurs étapes, éventuellement combinables.

Ces étapes sont repérables par des événements :

- « focus » : avant toute saisie, dans qu'on rentre dans le champ (le champ a le focus)
- « input » : au fur et à mesure de la saisie d'une donnée
- « blur » : à la fin de la saisie quand on sort du champ (le champ perd le focus)
- « submit » : quand valide le formulaire
- « reset » : quand annule le formulaire

On va regarder tous les cas.

Le formulaire : balise form

```
<form action="page.php" method="POST"
    onsubmit="return verifForm(this)"
    onreset="return resetForm(this)"
>
    <label for="pseudo">Pseudo : </label>
    <input type="text" name="pseudo" id="pseudo">
    <span id="aidePseudo"></span><br>

    <label for="passwd">Mot de passe : </label>
    <input type="password" name="passwd" id="passwd" oninput
="verifPasswd(this)" required>
    <span id="aidePasswd"></span><br>       ...

    <input type="submit" value="Envoyer">
    <input type="reset" value="Annuler">
</form>
```

Dans la balise <form> on met les classiques attributs action et method mais aussi des événements, ici :

- onsubmit : sur le bouton submit, on fera cette action
- on reset : sur le bouton reset, on fera cette action
- « this » correspond au nœud du DOM duquel la fonction est lancée : ici <form>

couple (label – input) + span

```
<label for="pseudo">Pseudo : </label>
<input type="text" name="pseudo" id="pseudo"
        onblur="verifPseudo(this)">
<span id="aidePseudo"></span><br>
```

Au classique couple <label> <input> on ajoute un qui va nous permettre de pouvoir ajouter des commentaires.

Dans la classique balise « input>, on ajoute un ou plusieurs événements, ici un « onblur », c'est-à-dire quand on perd le focus sur l'input.

Sur un onblur, on appelle la fonction verifPseudo(this).

La méthode du bouton submit : return false pour rester sur le formulaire

Le bouton « submit » déclenche l'action du formulaire.

Comment rester sur la page en cas d'erreur ?

Si la méthode de l'événement « onsubmit » retourne false, on reste sur la page, sinon, l'action du formulaire est appelée.

```
function verifForm(node) {
   …
   if(pseudoOk && passwdpOk)
      return true;
   else {
      alert("Veuillez remplir correctement tous les champs");
      return false;
   }
}
```

La méthode du bouton reset : return false pour rester sur le formulaire

Le bouton « reset » déclenche la fonction du formulaire.

Le reset met le « style » et le « textContent » à « » pour toutes les balises concernées.

On peut faire la liste des balises à partir de leur « id » en les mettant dans un tableau.

On pourrait aussi récupérer tous les « input » et tous les « span » du formulaire.

```
function resetForm() {
  let tabReset=["pseudo", "aidePseudo", "passwd", "aidePasswd" ];
  for(let key in tabReset){
    document.getElementById(tabReset[key]).style="";
    document.getElementById(tabReset[key]).textContent="";
  }
}
```

Mot-clé « this » : verifPseudo(this)

« this » correspond au nœud du DOM duquel la fonction est lancée.

Dans notre exemple, il s'agit de l'input.

A partir de ce nœud, on peut particulièrement récupérer l'attribut « value » : this.value.

On peut aussi accéder à l'attribut « style » pour chager le style.

- ***Exemple***

```
function verifPseudo(node) {
    if( node.value.length < 2 || node.value.length > 25 )
        node.style.backgroundColor = color;
}
```

- ***Correspondance des nom d'attribut entre CSS - JavaScript***

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference

ajout d'événements dans JavaScript : addEventListener – ajout de texte dans le HTML

- ***Principes de l'ajout d'événement***

  On peut ajouter des événements sur les balises <input> dans le fichier JavaScript.

  Un getElementById permet d'accéder un <input> particulier.

  Ensuite il reste à faire un addEventListener en précisant l'événement et en codant une fonction anonyme pour l'événement.

- ***Type d'événement***

  focus : quand on a le focus sur la balise

  blur : quand on perd le focus sur la balise

  input : à chaque saisie

  http://www.w3schools.com/jsref/dom_obj_event.asp

- ***Principes de l'ajout d'événement***

  On récupère la balise pour laquelle on va modifier le texte, par son id en général.

  Ici, c'est l'id « aidePseudo ».

  Et on modifie l'attribut textContent.

- ***Exemple***

```
// Affiche d'un message contextuel pour la saisie du pseudo
document.getElementById("pseudo").addEventListener("focus",
function () {
    document.getElementById("aidePseudo").textContent = "Entrez
votre pseudo";
});
```

Exemple 08-Formuaire-complexe

Expression régulière

Dans cet exemple, on trouve une expression régulière, c'est-à-dire une formule qui permet de vérifier un format complexe de chaine de caractères.

Par exemple : on veut une adresse mail : elle est constitué d'une première partie suivi d'un @ suivi d'une seconde partie suivi d'un . suivi de 2, 3 ou 4 caractères.

Ecriture de l'expression

```
var regex = /^[a-zA-Z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$/;
```

➢ ***Explications***

- ^[a-zA-Z0-9._-]+ => On commence (^), par 1 ou plusieurs (+) caractères alphanumériques (a-zA-Z). On peut aussi avoir des « . », des « - » et des « _ ».

  A noter que ça permet d'écrire : ._-_.
- @ => forcément 1 @
- [a-z0-9._-]{2,} => 2 ou plusieurs caractères alphanumériques avec des points, - et _

  On peut donc écrire : .@..
- \. => forcément 1 .
- [a-z]{2,4}$ => On termine ($) par 2 à 4 minuscules.

  On peut donc écrire .@...aa

➢ ***Principes de construction d'expressions régulières***

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/RegExp

On met l'expression entre / et /

Méthode test() : regex.test(string)

Avec la méthode test(), on vérifie si une string est conforme à la regex définie :

```
var regex = /^[a-zA-Z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$/;

if(!regex.test(node.value)){
      return false;
}
```

https://www.w3schools.com/jsref/jsref_regexp_test.asp

Exemple 09 - Formulaire-affichage

Dans cet exemple, on a un formulaire d'affichage avec aucun événement dessus.
On ajoute du JavaScript pour accéder aux informations.

1) On accède aux informations du formulaire
2) On ajoute un événement pour modifier la page
3) On accède aux information saisies avec des listener « change » sur les balises voulues
4) On accède aux informations saisies avec un listener « submit » sur la balise form

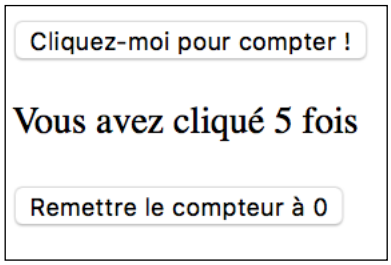
4 - Exercices

Exercice 2 : Compteur de clics

Objectif

Avec l'interface ci-dessous, on peut compter les clics et modifier l'affichage chaque fois qu'on clique sur « cliquez-moi pour compter ».

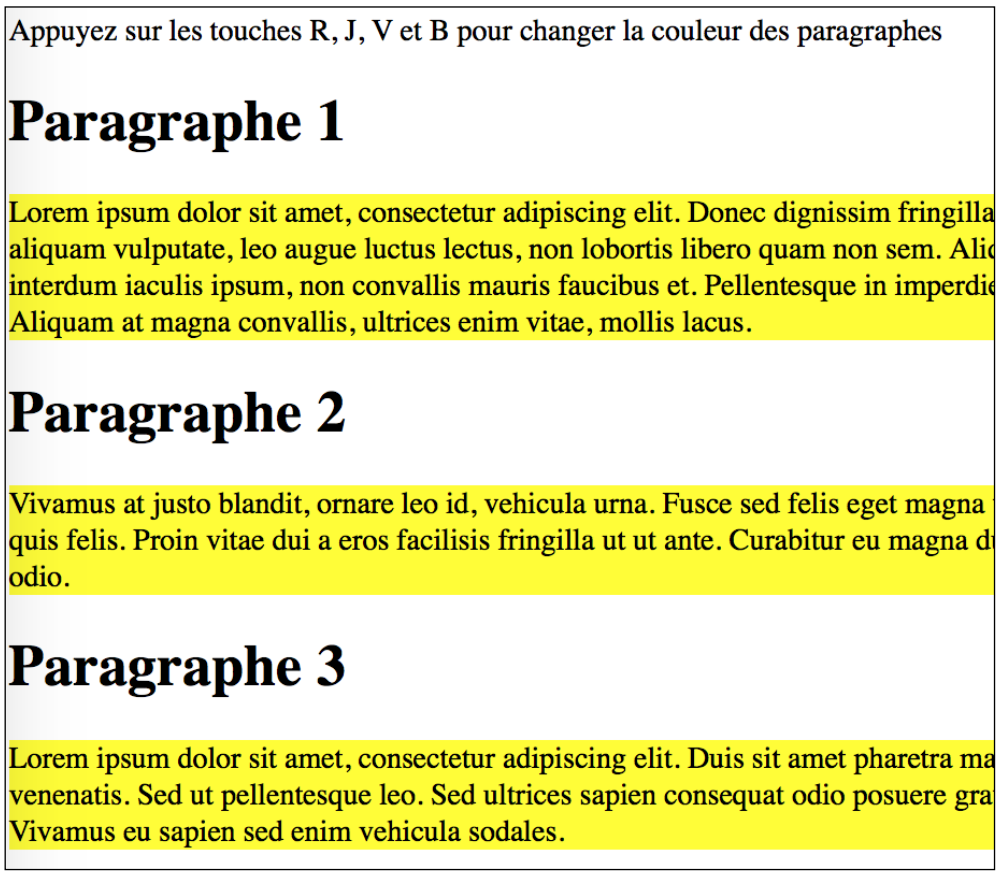
On peut remettre le compteur à zéro.



Exercice 3 : Changement de couleur au clavier

Objectif

En cliquant sur la touche « r », les paragraphes passent en rouge. Etc.



Exercice 4 : Todo list

Objectif

- ***Version 1 : créer une liste de « to do »***

  On peut simplement ajouter des choses à faire les unes en dessous des autres.

- ***Version 2 : ajouter la suppression***

  On peut vider toute la liste et aussi supprimer les éléments individuellement.

- ***Version 3 : ajouter la modification sur double-click***

  On peut modifier un élément de la liste par un doucle-clic.

  Ca ne change pas l'interface.

Technique

On peut utiliser 3 techniques différentes :

- innerHTML
- insertAdjacentHTML
- createElement.

On fera en sorte d'éviter de saisir une chaine vide. La méthode trim() supprime les espaces en début et fin d'une chaîne de caractères. Si on annule la saisie dans un prompt, il renvoie « null ».

Exercice 5

- ***LE FICHIER HTML est fourni***

  Le dossier « exo05-afficher-contenu-base » dans « JavaScript-02-02-Button-Evenements-Exemples.zip » contient le fichier HTML de l'exercice.

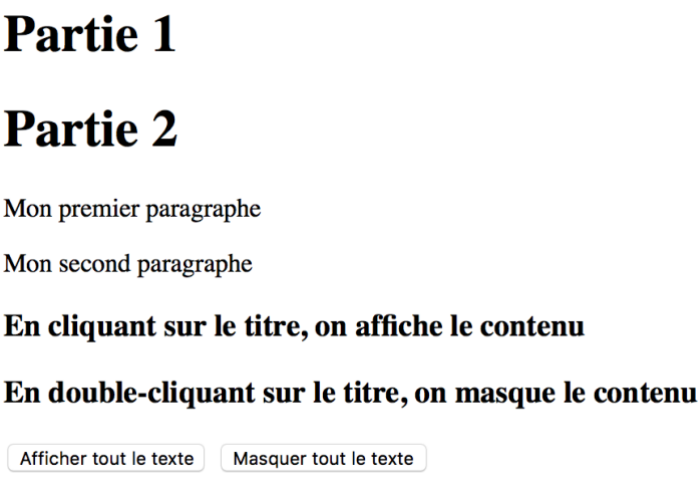
  Il reste à faire le JavaScript.

- ***Départ***

  Les boutons permettent d'afficher ou de masquer tout le contenu des parties.

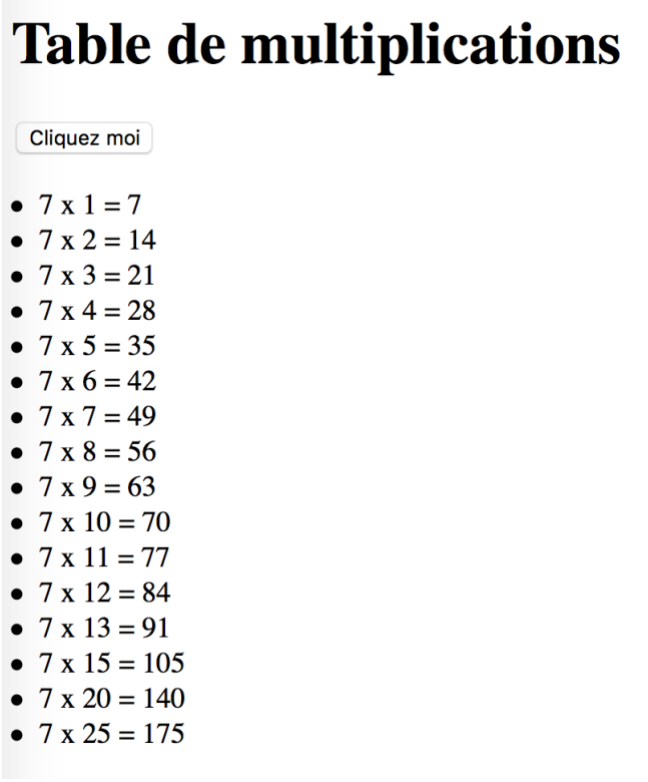
  Un clic ou double clic sur un titre permet d'afficher ou de masquer son contenu.

- ***Après un clic sur Partie 2***



Exercice 50 – Table de multiplication

1. On reprend l'exercice du premier cours : écrire une page HTML avec du JS qui permet d'obtenir le résultat suivant en cliquant sur le bouton :



On pourra saisir la valeur 7 ou bien n'importe quelle autre valeur.

On veut une version qui permette de surligner en jaune chaque ligne quand on passe dessus avec la souris et que ça revienne sans couleur quand la souris s'en va.

Principe de résolution :

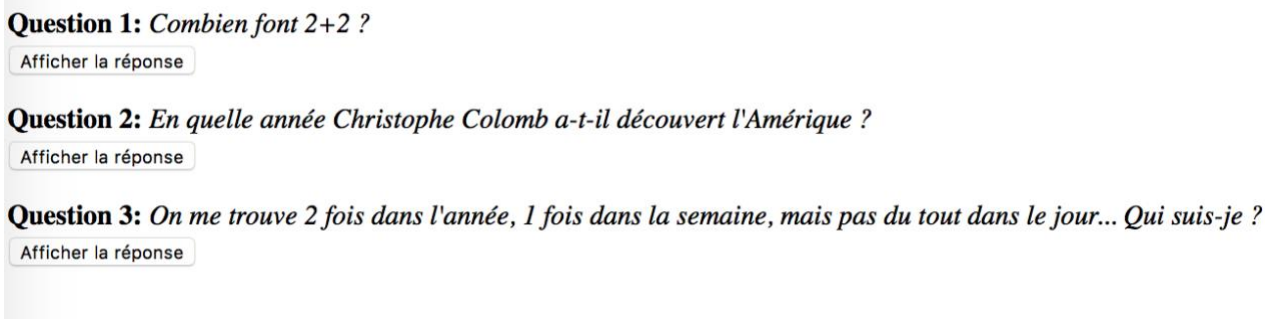
On commence par créer les <li> de la table de multiplication simple.

Ensuite on ajoute des événements sur ces

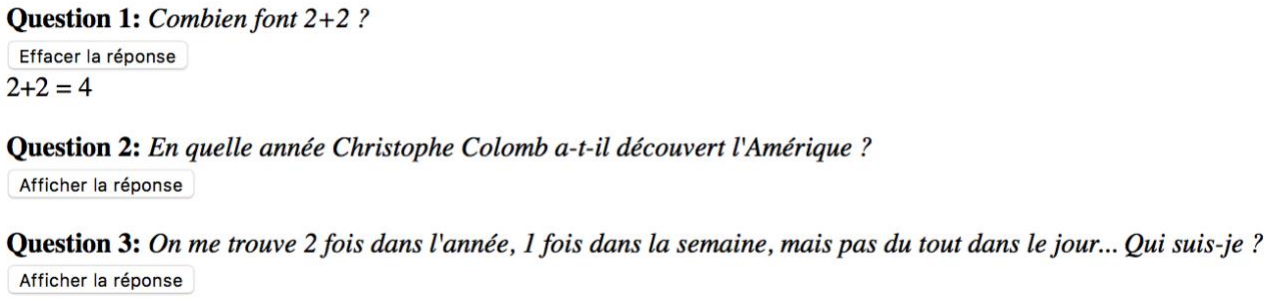
Exercice 6 : Quizz

Objectifs visuels

- ***Page d'accueil***



- ***Réponse affichée***



Principes techniques

La page HTML ne contient rien : uniquement un div avec un id et rien dedans.

Les questions/réponses seront rangées dans un tableau de structures :

```
var questions = [
   {
      enonce: "Combien font 2+2 ?",
      reponse: "2+2 = 4"
   },
etc
```

A partir de là, on ajoute les éléments dans le div.

Exercice 6 bis : QCM

Objectifs

Cette fois, on veut gérer un QCM de type : 1 réponse bonne et une seule parmi plusieurs choix.

On va utiliser des « radio-boutons » HTML.

On validera la totalité du QCM : ça donnera un nombre de bonnes réponses sur le nombre total.

On fera une version qui donne uniquement la note et une version qui donne la note et affiche en vert la bonne réponse et en rouge les mauvaises réponses données.

Comme pour l'exercice précédent, les données sont stockées dans un tableau JSON.

Exercice 7 : Liste de personnages

Objectifs visuels

- ***Page d'accueil***

- ***Page des résultats après le choix d'une maison dans la liste :***

L'objectif est d'obtenir l'équivalent d'un menu déroulant : balise <select> avec les maisons.

Quand on choisit une maison, on obtient la liste à points, balise <li> des personnages

Page HTML : menu déroulant et liste à points

- ***Menu***

Le menu est codé avec une balise <select> de menu déroulant. Dans la balise <select> on met des balises <option>. La première <option> est : Choisissez une maison. Cette option est affichée dans le HTML. Pour selectionner une option, il suffit de mettre l'attribut « selected » sans valeur dans la balise <option>. Les autres seront remplies dynamiquement à partir du JS.

http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_option_selected

- ***Liste à points***

Et les personnages peuvent apparaître dans des balises <li>. Au départ, il n'y a rien dedans : on aura un <ul> vide. Les personnages apparaîtront dynamiquement.

Le menu déroulant fonctionne avec un <select> et des <options> :

Page JS

- ***Première partie : création du menu déroulant***

On va remplir le menu déroulant et la liste des personnages dynamiquement.

Pour ça, on range toutes les informations des maisons une structure à 3 champs : code, nom, et personnages.

```
{
    code: "ST",
    nom: "Stark",
```

personnages : ["Eddard", "Catelyn", "Robb", "Sansa", "Arya", "Jon Snow"]

}

3 autres maisons :

("LA", "Lannister", ["Tywin", "Cersei", "Jaime", "Tyrion"]),

("BA","Baratheon", ["Robert", "Stannis", "Renly"])

("TA","Targaryen", ["Aerys", "Daenerys", "Viserys"])

On mettra les maisons dans un tableau appelé : lesMaisons[]

On crée le menu déroulant à l'aide de ce tableau.

- ***Deuxième partie : création de la liste à points***

  Il faut pouvoir récupérer la liste des personnages pour une maison sélectionnée.

  On va se doter d'une fonction function getPersonnages(codeMaison)

  En fonction du code maison, elle retourne la bonne liste de personnage.

  Elle utilise le tableau lesMaisons[] qui est une variable gloabale.

  Pour créer la liste à points, on ajoute un listener sur le <select> et sur un événement « change ».

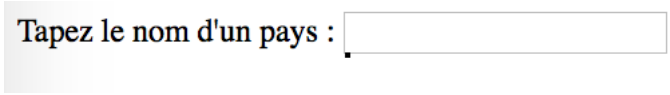
  Rappel de la liste des événements : https://www.w3schools.com/jsref/dom_obj_event.asp

Exercice 8 : Auto_complétion : difficile

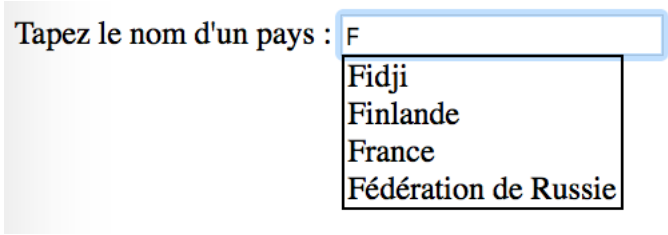
On saisit une valeur et une liste de valeurs possibles est proposée. La liste se réduit on fur et à mesure de la saisie. On peut sélectionner l'élément voulu dans la liste proposée.

- ***Objectifs visuels***

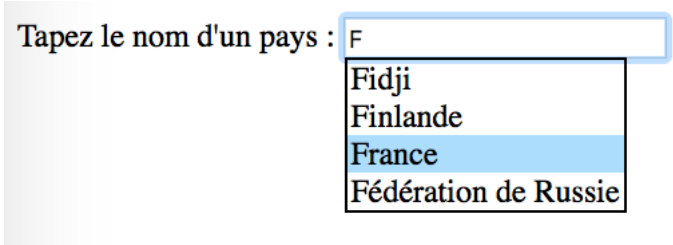
Page d'accueil



Saisie d'une première lettre



Sélection d'un pays



Le pays est sélectionné

L'objectif est de saisir un pays et qu'une liste soit fournie en filtrant à partir de ce qui a été saisi.

- ***Page HTML***

La page HTML est simple : elle contient un <label> et un <input>

Pour la liste des suggestions, on se dote d'un vide.

- ***Page JavaScript***

Il faut se doter d'un tableau de pays. On en fournit un avec les exemples : exo08-autocomplétion-tableauDePays

On se dote d'un nœud pour l'input et d'un nœud pour les suggestions.

On crée un évenement sur une saisie dans l'input : oninput

Pour positionner correctement les suggestions, on récupère l'offsetLeft de l'input et on le met dans le left des suggestions. Attention, il faut mettre : noeudInput.offsetLeft+'px' dans le left.

Ensuite, on parcourt le tableau des pays et pour chaque pays, on regarde s'il est égale à la saisie en cours. La fonction indexOf permet de faire ça : a.indexOf(b) dit la position de b dans a. Si la position vaut 0 c'est que a commence par b.

Si un pays correspond à la saisie en cours, on en fait un enfant du nœud des suggestions.

On va gérer 3 événement sur cet enfant :

Un onclick pour pouvoir le cliquer et passer la valeur cliquée dans l'input.

Un onmouseover pour le passer en grisé quand la souris passe dessus.

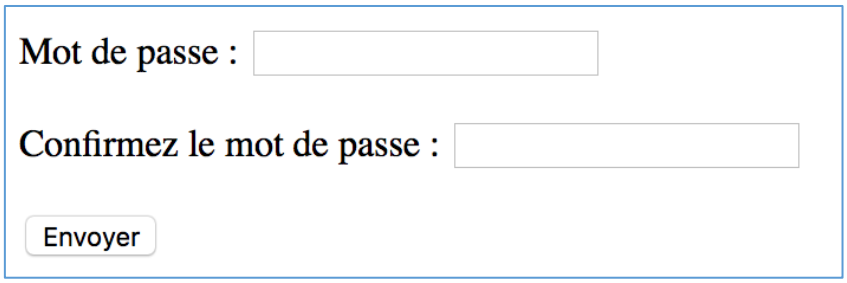
Un onmouseout pour supprimer le grisé quand la souris le quitte.

Enfin, sur le nœud des suggestions, on pourra gèrer un onmouseleave (presque la même chose qu'un onmouseout) pour supprimer les suggestions si la souris sort de la zone des suggestions.

Exercice 09 : Formulaire de mot de passe

Cahier des charges

- ***On veut pouvoir saisir un mot de passe et sa confirmation puis l'envoyer sur une autre page.***

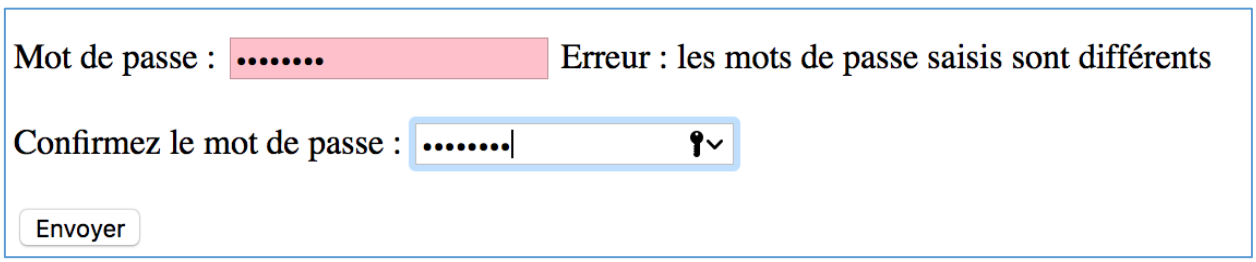


- ***Si le mot de passe est conforme, on affiche la page appelée : page.php :***

```
<?php
echo '<pre>' ; print_r($_POST); echo '</pre>'
echo "mot de pass correctement saisi";
?>
```

- ***Si le mot de passe n'est pas conforme, on affiche le bon message d'erreur***

Par exemple :



- ***Contraintes pour le mot de passe***

Il doit avoir au moins 6 caractères, au moins un caractère et au moins un chiffre.

- ***Variante***

Le mot de passe doit avoir au moins une minuscule, une majuscule, un chiffre. Il peut avoir un '-' mais aucun autre caractère non alphanumérique.

Exercice 10 : formulaire d'un projet PHP

Dans n'importe quel projet PHP (le projet « Artiste ») par exemple, mettez des contrôles sur un formulaire de saisie.

Exercice 11 – Jeu des allumettes (jeu de Marienbad)

Présentation du jeu

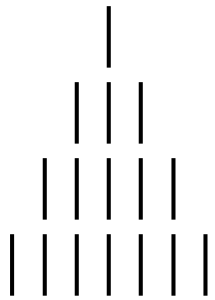
- ***Test du jeu***

  Par exemple :

  http://therese.eveilleau.pagesperso-orange.fr/pages/jeux_mat/textes/marienbad.htm

- ***Description du jeu***

  On a 4 lignes d'allumettes avec 1, 3, 5 et 7 allumettes.



  Le but du jeu est de ne pas ramasser la dernière allumette : qui prend la dernière perd.

  Le jeu se joue à 2. A son tour, chaque joueur peut prendre autant d'allumettes qu'il veut mais sur une seule ligne.

- ***Comment gagner à tous les coups ?***

  D'abord, il ne faut pas commencer.

  Ensuite, il faut laisser le jeu dans une configuration particulière expliquée maintenant.

  Chaque ligne contient un nombre d'allumettes : 1, 3, 5 et 7 au départ. Il faut traduire ce nombre en binaire.

  Situation de départ :

Nombre d'allumettes	Nombre d'allumettes en binaire		
1	0	0	1
3	0	1	1
5	1	0	1
7	1	1	1
Total en base 10 de chaque colonne binaire :	2	2	4

  Le joueur 1 qui commence est dans une configuration telle que le total en base 10 de chaque colonne binaire est un nombre pair. C'est cette configuration qui fait que le joueur va perdre, si l'adversaire ne fait pas d'erreur.

  Quand le joueur 1 va joueur, quoi qu'il fasse, il laissera un ou plusieurs totaux impairs.

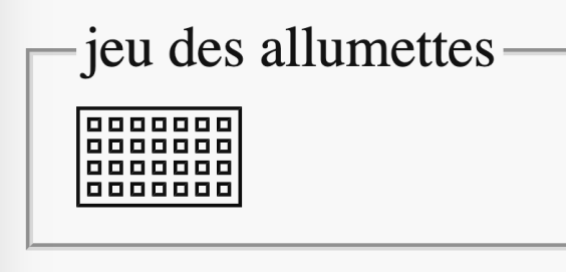
  Le joueur 2 devra jouer de telle sorte que chaque total soit à nouveau un nombre pair.

- ***Une exception***

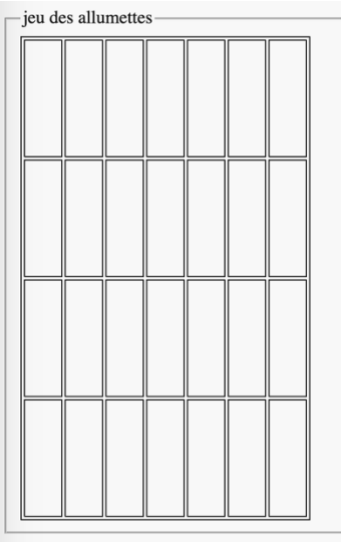
Il ne faut pas laisser 1 allumette sur les 4 lignes, ni 1 allumettes sur 2 lignes. Le total est pair. Mais celui qui joue va alors gagner !

Etapes de résolution

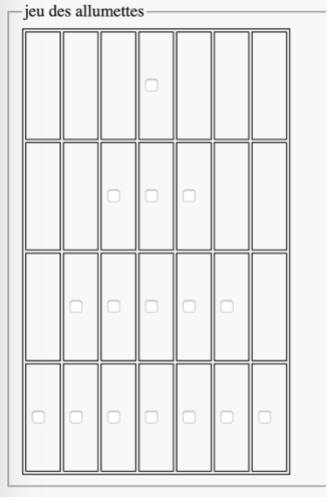
- ***Etape 1 : HTML : Affichage d'un tableau de 4 lignes et 7 colonnes, sans CSS***



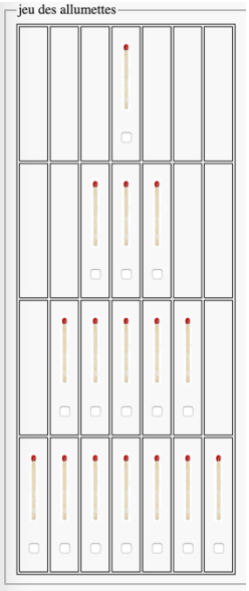
- ***Etape 2 : HTML : Affichage d'un tableau de 4 lignes et 7 colonnes, avec CSS minimal***



- ***Etape 3 : HTML : Ajout d'un input checkbox***



- ***Etape 4 : HTML : Ajout d'une image d'allumette***



- ***Etape 5 : JavaScript : afficher l'état des boutons en console***

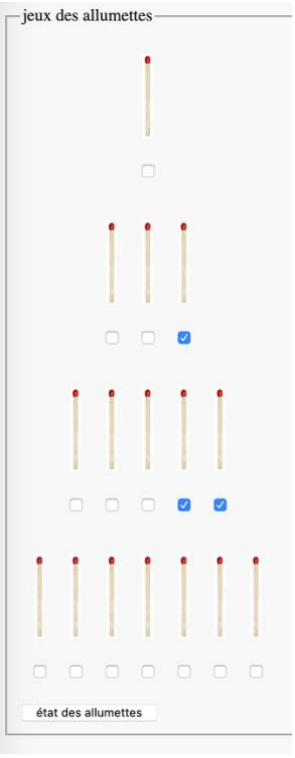
Pas de modification du HTML

JavaScript :

```
console.log("allumetteL1A1 : "+allumetteL1A1.checked);
```

allumetteL1A1, c'est la variable qui correspond à la balise. C'est un objet avec entre autre un attribut « checked ».

- ***Etape 6 : JavaScript : afficher l'état des boutons sur demande***



JavaScript :

```
document.getElementById('etat').onclick = function(){
```

- ***Etape 7 : JavaScript : effacer les allumettes : fonction « eval »***

Le bouton « état » devient un bouton « effacer ».

Pour effacer, on commencer par supprimer le « src » de l'image :

```
if(allumetteL1A1.checked){ allumetteL1A1img.src=""; }
```

On va aussi faire une deuxième version qui boucle sur toutes les allumettes.

On construit le nom de l'allumette dans une variable : par exemple, la variable « allumette » vaudra « allumetteL2A3 ».

Ensuite, on utilise la **fonction « eval » :**

```
if(eval(allumette).checked){
```

Cette fonction permet de faire comme si une chaine de caractères était un nom de variable.

- ***Etape 8 : on efface tout le TD en les numérotant***

HTML

```
                    <td id="tdL1A1">
                        <p><img src="allumette.png"></p>
                        <p><input type="checkbox"></p>
                    </td>
```

JavaScript

```
          baliseInput=baliseTD.getElementsByTagName("input")
          if(baliseInput[0].checked){
             baliseTD.style.display="none"
          }
```

- ***Etape 9 : on ajoute un RAZ***

Le RAZ permet de recommencer la partie.

- ***Etape 10 : on fait en sorte qu'à chaque nouveau chargement, on reparte avec un RAZ***

JavaScript

```
window.onload=function(){
  raz()
}
```

- ***Etape 11 : on évite de pouvoir cliquer sur deux lignes***

- ***Etape 12 : on affiche « gagné » ou « perdu » en bas de page***

4 : ANIMATION

Installation des fichiers de tests

Dans le cours :

Les exemples de code sont présentés dans un chapitre en vert pomme !

Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript-02-04-Animation-Exemples

Chargez ces fichiers et mettez-les :

Soit dans un JavaScript.

Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

L'animation des pages en JavaScript

https://openclassrooms.com/courses/creez-des-pages-web-interactives-avec-javascript/animez-vos-pages

Présentation

L'animation des pages est un vaste domaine !

Quelques fonctions permettent d'animer les éléments :

Les fonctions du « timers »

Ce sont des **fonctions très simples d'usage** pour :

- décaler dans le temps une action
- répéter périodiquement une action

https://developer.mozilla.org/fr/docs/Web/API/WindowTimers

https://www.w3schools.com/js/js_timing.asp

Il y en a 4 :

- setTimeout()
- clearTimeout()
- setInterval()
- clearInterval()

Elles sont accessibles à partir de l'objet window, ou directement.

Les fonctions AnimationFrame

https://developer.mozilla.org/fr/docs/Web/API/Window/requestAnimationFrame

Il y en a 2 :

- requestAnimationFrame
- cancelAnimationFrame

Elles sont accessibles à partir de l'objet window, ou directement.

Ce sont des fonctions plus complexes d'utilisation

setTimeout() : appeler une fonction après un délai

https://www.w3schools.com/jsref/met_win_settimeout.asp

Fonction d'animation très pratique, pour par exemple :

- découvrir une image, et qu'elle soit recouverte automatiquement après quelques secondes
- faire bouger les aiguilles d'une montre
- etc.

```
setTimeout(maFonction, delai)
```

Le délai est exprimé en millisecondes.

clearTimeout() : annuler la précédente

L'usage est plus rare.

Un exemple :

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_win_cleartimeout

setInterval() : appeler une fonction périodiquement

https://www.w3schools.com/jsref/met_win_setinterval.asp

Fonction d'animation très pratique, pour gérer des déplacements, ou un chronomètre.

```
var intervalId = setInterval(diminuerCompteur, 1000);
```

La fonction setInterval(maFonction, période) appelle « maFonction » tous les « période » millisecondes.

C'est une boucle sans fin.

clearInterval(): annuler la précédente

Fonction d'animation très pratique pour stopper une animation sans fin, ou un chronomètre.

Un exemple :

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_win_clearinterval

Exemple 1-Animation-fonction-timer

HTML

On met la valeur de départ du compteur dans un span avec un id

```
<span id="compteur">5</span>
```

Javascript

- ***Programme***

```
var intervalId = setInterval(diminuerCompteur, 1000);
```

- ***Fonction***

```
function diminuerCompteur() {
    compteurElt = document.getElementById("compteur");
    var compteur = Number(compteurElt.textContent); // récupère
5, de string à Number
    if (compteur > 1) { // on modifie le compteur affiché
        compteurElt.textContent = compteur - 1;
    } else { // on change tout l'affichage
        clearInterval(intervalId); // Annule l'exécution répétée

        // on récupère le noeud du titre pour modifier tout le
texte
        var titre = document.getElementById("titre");
        titre.textContent = "BOUMM !!!"; // on change le texte

        setTimeout(function () {  // on change encore le texte,
mais après 2 secondes
            titre.textContent = "Tout est cassé :(";
        }, 2000);
    }
}
// On appelle la fonction diminuerCompteur toutes les secondes
var intervalId = setInterval(diminuerCompteur, 1000);
```

Exemple 2 - Animation-horloge : à tester !

Horloge graphique simple à tester.

Exemple 3 - Animation-fonction-requestAnim-mini : requestAnimationFrame()

Présentation

C'est une fonction plus adaptée pour le graphisme.

https://developer.mozilla.org/fr/docs/Web/API/Window/requestAnimationFrame

https://software.intel.com/en-us/html5/hub/blogs/request-animation-frame-for-better-performance

La fonction requestAnimationFrame demande au navigateur d'exécuter « dès que possible » une fonction.

Le navigateur va optimiser la mise à jour de l'animation afin de la rendre fluide.

```
animationId = requestAnimationFrame(maFonction);
```

Principes

On fait une boucle récursive et on modifie la position

```
var bloc = document.getElementById("bloc"); // noeud du bloc

requestAnimationFrame(deplacerBloc);

function deplacerBloc() {
    // on met à jour le left avec l'offsetLeft
    bloc.style.left =(bloc.offsetLeft-8)+'px';

    requestAnimationFrame(deplacerBloc);
}
```

On récupère le nœud à faire bouger.

On appelle la fonction requestAnimationFrame avec une fonction en paramètre.

On définit la fonction :

- Elle modifie un élément de position du bloc (left, top, etc.) en partant de l'offset.
- On fait l'appel récursif.

REMARQUES :

- C'est la modification du style.left qui fait le mouvement (l'offsetLeft bouge tout seul). Avec -9, ça ne bouge plus. A -8, ça va très lentement.
- Le bloc est défini en position : absolute ou relative dans le CSS.

cancelAnimationFrame()

On peut arrêter la boucle infinie de l'animation avec un cancel :

```
idAnimation = requestAnimationFrame(deplacerBloc);

cancelAnimationFrame(idAnimation);
```

Comment choisir entre setInterval, requestAnimationFrame et CSS

- Si l'animation n'est **pas en temps réel** et doit simplement se produire à intervalles réguliers, utilisez **setInterval.**
- Si l'animation est **en temps réel** et que vous savez qu'elle peut être effectuée en **CSS**, adoptez cette technique : le CSS.
- Dans les autres cas, donc si l'animation est **en temps réel** et ne peut pas être effectuée en CSS, utilisez **requestAnimationFrame.**

L'animation des pages en CSS

Présentation

Le CSS est le plus performant mais ne permet pas de tout faire.

Cf. Cours CSS sur transitions et animations.

Exemple 4 - Animation-CSS

Structure du CSS

Le bloc à déplacer

```
#bloc {
    animation-name: deplacerBloc; /* Nom de l'animation */
    animation-duration: 6s; /* Durée de l'animation */
}
```

Les caractéristiques de l'animation : @keyframes avec le nom de l'animation de la balise

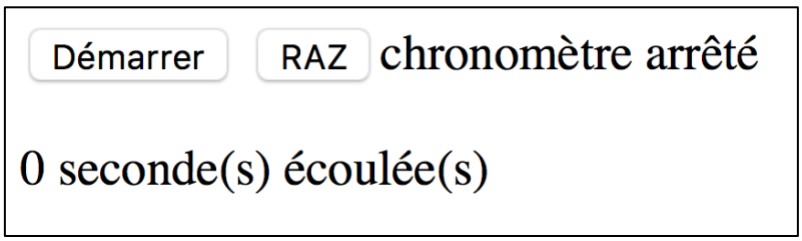
```
@keyframes deplacerBloc {
    from {
        left: 20px; /* Position initiale */
    }
    to {
        left: 100%;     /* Position finale */
    }
}
```

Exercices

Exercice 01 : Chronomètre

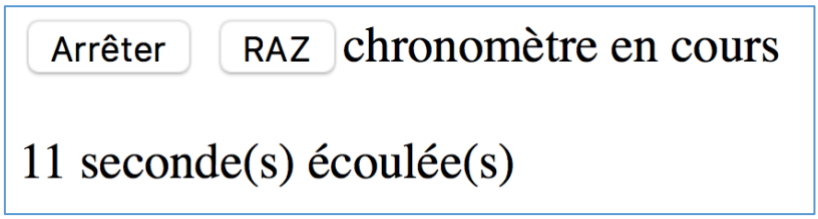
Cahier des charges

- ***On veut coder un chronomètre. L'interface est la suivante :***



- ***Bouton démarrer***

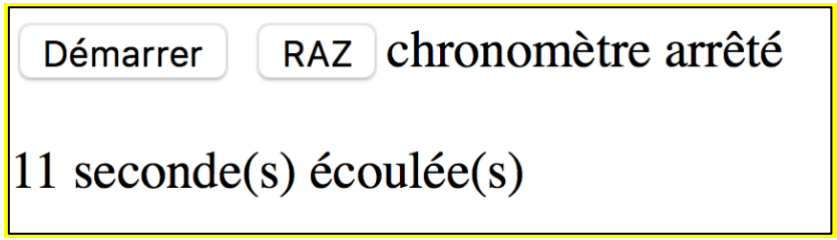
  Si on démarre, les secondes s'affichent. Le bouton « démarrer » est remplacer par « arrêter ».



- ***Bouton arrêter***

  Si on arrête, les secondes arrêtent de défiler. Le bouton passe à « démarrer ».

  On peut redémarrer et le compteur repart.



- ***Bouton RAZ***

  On ne peut utiliser RAZ que si le compteur est arrêté. Dans ce cas, le compteur passe à 0.

  Sinon, on a un message d'erreur.

Exercice 02 : Carte d'anniversaire

Source : OCR

http://exercices.openclassrooms.com/assessment/527?login=7078887&tk=d2693304a7a7999b58efb5cff0e6c0db&sbd=2016-02-01&sbdtk=2466d6bae51e373d89ac8e3f74213199

Réalisez une petite "carte" web amusante.

Construisez une page contenant un message d'anniversaire évolutif.

En haut de votre page, le message "Joyeux anniversaire", suivi de 3 petits paragraphes de vœux, qui se dévoilent un par un lorsque l'on clique sur le précédent.

Une fois que l'utilisateur a cliqué sur le dernier paragraphe, un gros effet sonore doit se faire entendre et l'image d'arrière-plan change.

On peut ensuite imaginer d'autres animations !

Exercice 03 : Mémory

- ***Cahier des charges***

  On affiche un carré d'images : 5 sur 5, 6 sur 6 etc. Ou un rectangle.

  Les images apparaissent en gris.

  On clique sur 1 elle se retourne. Sur une autre elle se retourne.

  Si ce sont les mêmes, on peut continuer. Sinon, les deux images se retournent et passent en gris.

  Et on recommence.

- ***BASE DE TRAVAIL***

  Le dossier « exo04-memory-base» contient une

  Le dossier « exo04-memory-base» dans « JavaScript-02-03-Formulaires-Animation-Exemples.zip » contient une première version du travail. On peut s'en inspirer pour avancer.