Chapitre 1 – Partie 2 : Langage JavaScript – Généralités

1. Notions de base JavaScript

Il est possible de trouver un tutoriel complet sur les technologies web (HTML, CSS et JavaScript) sur : https://developer.mozilla.org/fr/docs/Web. Pour voir tous les détails du langage JavaScript avec des exemples et les descriptions détaillées de tous les concepts.

(Prière de consulter ce lien pour les détails omis dans ce cours.)

1.1. Emplacement du code JavaScript

En principe n'importe où dans un document HTML mais il existe des patterns connus:

```
<!doctype html>
<html>
<head>
< !-  Emplacement des bibliothèques JavaScript ->
….
</head>
<body>
 . . . le contenu de la page web est mis ici . . .
…..
< !-  Emplacement du code JavaScript ->
</body>
</html>
```

Il est conseillé de placer le code JavaScript juste avant la balise de fin du corps du document (</body>) pour permettre à la page de s'afficher plus vite.

Le code JavaScript peut :

- Soit se trouver dans le fichier HTML de la page web :

```
<script>
function surprise() {
alert("Salut!"); }
</script>
```

- Soit se trouver dans un fichier externe

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

Dans le fichier mycode.js

```
function surprise() {
alert("Salut!"); }
```

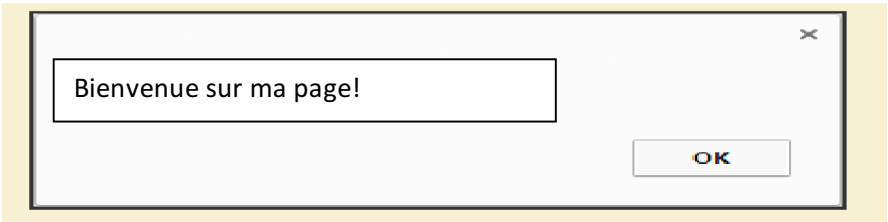
1.2. Dialogues en JavaScript

3 types :

- *alert() :* pour afficher un message

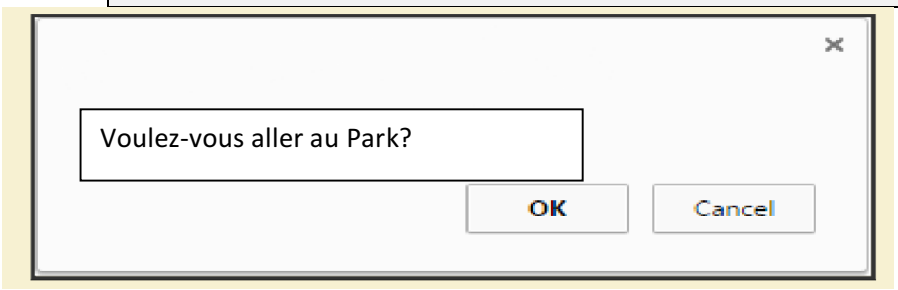
Ex : alert ("Bienvenue sur ma page! ")

Affichage :



- *confirm() :* pour afficher un message avec une décision

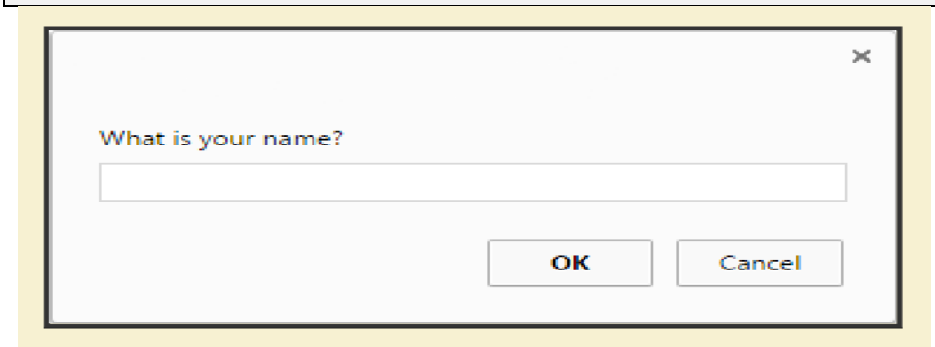
```
<!doctype html> <html>
<head>
<title>Exemple pour confirm()</title>
<script>
if (confirm("Voulez-vous aller au Park?"))
document.location.href  = "http://park.com";
</script>
</head>
</html>
```



- *prompt():*pour afficher un message et recevoir une données de l'utilisateur

Ex:

```
var user_name; // Création d'une variable
user_name = prompt("What is your name?");
```



Exemple1:

```
<!doctype html> <html>
<head>
<title>Exemple pour prompt()</title>
<script>
var user_name;
user_name=prompt("What is your name?");
document.write("Welcome to my page " + user_name + "!" );
</script>
</head>
</html>
```

1.3.Événements en JavaScript:

Un événement indique la survenue d'une chose importante qui doit être traitée. Par exemple: un clic sur quelque chose, un mouvement de la souris, appuyer sur une touche du clavier ou sur un bouton …

On peut prévoir d'exécuter un code lorsqu'un événement survient, c'est le traitement de l'événement.

Des exemples d'événements: ***onclick, onmouseover, onmouseout, onmouseup, onmousedown, keypressed*** etc.

Exemple : l'événement de chargement de la page web : ***onload***

```
<body onload="alert('Hello!')">
. . . le contenu de la page web est mis ici . . .
</body>
```

1.4.Fonctions en JavaScript:

Comme dans les autres langages de programmation, une fonction est un ensemble d'instructions exécutées lors de l'appel à la fonction. On peut avoir ou non des paramètres, retourner un résultat, faire des appels récursifs pour une fonction etc.

- En JavaScript, les fonctions jouent un rôle très important. Presque tout code JavaScript est écrit sous forme de fonctions.
- Une fonction qui retourne un résultat le fait à travers le mot-clé ***return***.
- Le mot clé ***return*** stoppe l'exécution de la fonction.
- Une fonction peut avoir une autre fonction comme paramètre, cet aspect est connu sous le nom de **callBack** function.

- Une fonction peut être déclarée de 2 manières en JS selon les exemples suivants :

```
// façon classique

function capitalize(str) {
return str.slice(1); }
```

```
//autre façon JS

var capitalize =
function(str){
return str.slice(1); }
```

Exemple2 :

```
<!doctype html>
<html>
<body onload="verifier_age()" style="position:absolute">
<h1>Ma premiere home page</h1>
<script>
var x;
function verifier_age(){
if (age_of_user() < 18)   alert("Allez sur une autre page SVP.");
}
function age_of_user(){
var age_text, age;
age_text=prompt("Quel est votre age?");
age=parseInt(age_text);
        return age;       }
</script>
</body>
</html>
```

1.5. Variables locales ou globales :

Les variables ne sont pas obligatoirement déclarées (par le mot-clé ***var***) en JavaScript mais il est conseillé de les déclarer dans leurs fonctions pour éviter certaines erreurs.

- Les variables déclarées à l'intérieur d'une fonction sont dites locales, elles ne peuvent être utilisées qu'à l'intérieur de cette fonction
- Les variables déclarées dans le programme principal sont dites globales et peuvent être utilisées à l'intérieur et à l'extérieur des fonctions.
- Si deux variables (l'une globale et l'autre locale) partagent le même nom, la priorité est pour la variable locale à l'intérieur de sa fonction. (Voir exercice en TD)

1.6. Objets en JavaScript :

- En JavaScript, tout est objet, y compris les types de base et les fonctions. Un objet est défini comme un ensemble de paires (attribut : valeur). Un objet peut avoir des propriétés (attributs) et peut avoir aussi des méthodes.

Exemple :

```
var personne = {
     nom: "Ali",
     age: 32,
     ville: "Mila"
};
```

- Les propriétés et les méthodes d'un objet sont accessibles en utilisant 2 notations possibles :

Notation point	Notation par crochets
Ex : `personne.nom`	Ex : `personne["nom"]`

La notation point nécessite que le nom de la propriété ne commence pas par un chiffre.

1.7 Éléments du langage JavaScript

a) Les fonctions sur les tableaux :

- *sort(); indexOf() ; slice() ; reverse() lastIndexOf() ; splice().*

b) Les itérateurs :

- *forEach() map()*

Exemples en TD

1.8 Affichages en JavaScript

Pour afficher un message en JavaScript il y-a 4 possibilités :

1) **Boites de dialogue** : (***alert, confirm, prompt***)
2) Écriture dans le document HTML par ***document.write(message)*** ce qui provoque l'effacement complet du contenu de la page et l'affichage de ***message*** uniquement.
3) Écriture dans un élément HTML en utilisant ***innerHTML(***possible avec ***le DOM)***
4) Écriture dans la console du navigateur en utilisant ***console.log(message)*** qui affichera le message dans la console et pas sur la page web.

1.9 Gestion des erreurs et debugging en JavaScript

Le langage JavaScript est un langage faiblement typé et considéré parmi les plus souples et les plus faciles à apprendre. Cependant, il comporte un défaut majeur qui est l'occurrence des erreurs.

- Dans le cas des erreurs syntaxiques, il est relativement facile de les corriger
- Pour ce qui est des erreurs logiques, elles sont souvent indétectables et c'est au programmeur de les trouver et de les corriger. Afin d'aider le programmeur dans cette tâche difficile, on dispose d'un debugger pour JavaScript disponible sur le navigateur (voir mise en œuvre en TP)
- On peut afficher des messages de vérification dans le code non pas à l'aide de boîtes de dialogues (***alert, confirm, prompt***) mais par ***console.log (message)*** qui affichera le message dans la console et pas sur la page web (pour faciliter la vérification par le programmeur)

2 Le DOM

DOM = Document Object Model.

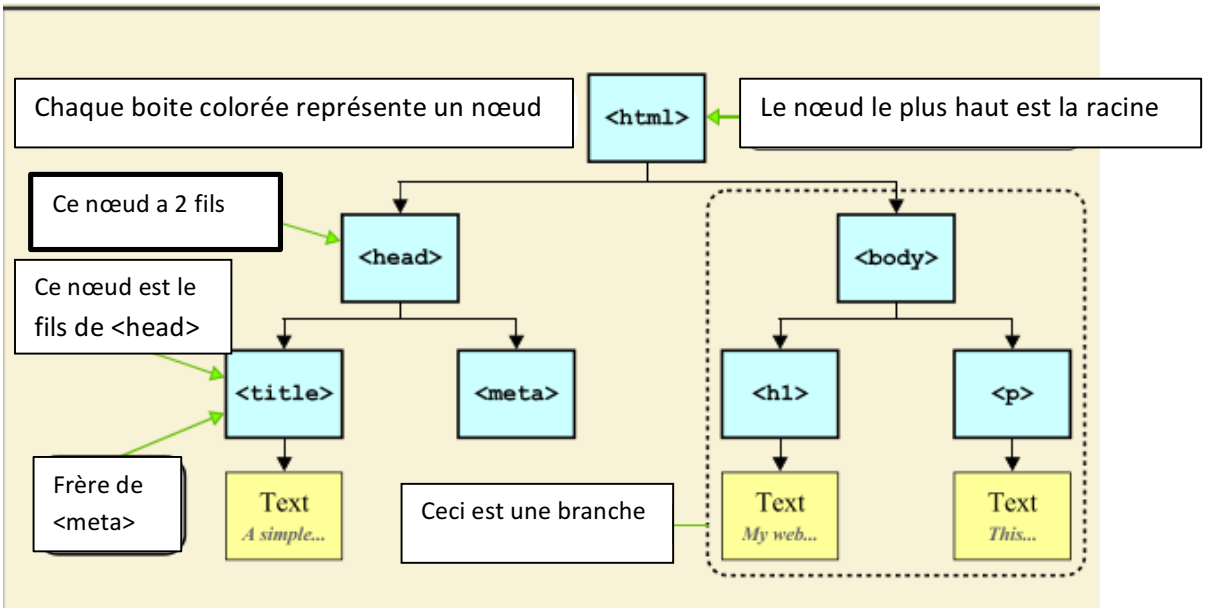
- Une page chargée dans un navigateur est mémorisée sous forme d'une structure arborescente appelée DOM.
- Le DOM représente une interface entre JavaScript et (HTML+CSS) car grâce à lui on peut les manipuler par des programmes JS.
- Le navigateur transforme chaque élément HTML sous forme d'objet JavaScript qu'on peut manipuler (modifier/supprimer etc)

Exemple de page :

```
<!DOCTYPE html>
<html>
  <head>
        <title>A Simple Web Page</title>
        <meta name="author" content="MyContent">
  </head>
  <body>
```

```
        <h1>My Web Page</h1>
        <p>This page is remarkable !</p>
    </body>
</html>
```

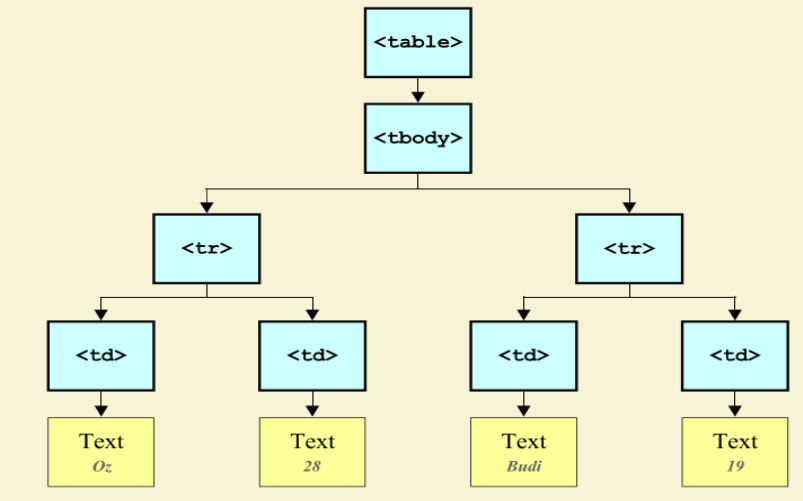
La structure DOM Associé :



Autre exemple de document html :

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
        <table>
            <tbody>
                <tr> <td>Oz</td> <td>28</td> </tr>
                <tr> <td>Budi</td> <td>19</td> </tr>
            </tbody>
        </table>
  </body>
</html>
```

Le DOM associé (à partir de l'élément table seulement)

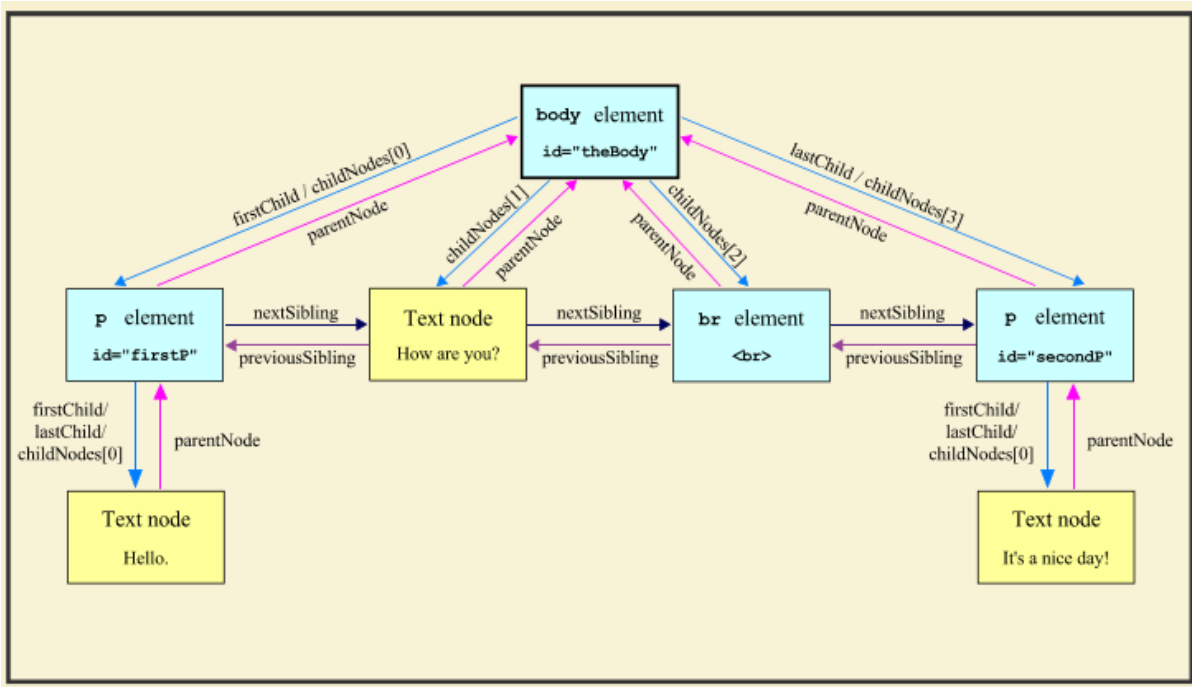


2.1 Relations entre nœuds

L'utilité du DOM tient au fait qu'il devient possible d'accéder aux différents nœuds représentant la structure d'une page web, grâce aux relations qui existent entre les nœuds et aux fonctions permettant d'y accéder. En particulier, on a les fonctions suivantes :

- Manipulation du nœud parent: ***parentNode***
- Manipulation des nœuds enfants : ***childNodes[ ], firstChild, lastChild***
- Manipulation des nœuds freres ***previousSibling, nextSibling***

Dans un DOM, la disposition des nœuds et de leur descendance permet d'utiliser ces fonctions comme schématisé :



Grâce à ces fonctions, on peut trouver le chemin vers un nœud, localiser un nœud ou un ensemble de nœuds afin de leur appliquer un certain traitement par exemple.

2.2 Sélection des nœuds

Il est possible grâce au DOM et aux fonctions précédentes d'ajouter, supprimer, copier ou changer n'importe quel nœud du DOM. Pour cela, on utilise des mécanismes d'accès et de sélection des nœuds à travers une des 3 méthodes suivantes :

- **Méthode 1 : Utilisation du chemin exact** : nécessite de connaître exactement ce chemin, il y a un risque d'erreur du fait que le DOM peut différer d'un navigateur à l'autre
- **Méthode 2 : Utilisation du type de l'élément** (le type de sa balise) :*getElementsByTagName( )* qui nécessite de connaître exactement le nom de la balise de l'élément recherché (h2 ou h3 ??), risque d'avoir plusieurs éléments qui ont ce nom.
- **Méthode 3 : Utilisation du nom de l'élément lui-même** : *getElementById( ).* Cette méthode est la plus facile si l'élément recherché porte un identificateur comme suit : <element_name id="truc"> . . . </element_name>

Exemple de document:

```
<body>
 <h2 style="color:black" id="pur_texte">
Cliquer sur un bouton pour changer la couleur
</h2>
<form>
<input onclick="change_color1()" type="button"
value="Change using method 1">
<input onclick="change_color2()" type="button"
value="Change using method 2">
<input onclick="change_color3()" type="button"
value="Change using method 3">
</form>
 </body>
```

Le DOM associé à ce document est comme suit :
(à finir)

Exemple3 : le code manipulant ce document : (à tester)

```
<!DOCTYPE html>
<html>
<head>
 <script>
function change_color1(){
document.childNodes[1].childNodes[2].childNodes[1].style.color="red";
}
function change_color2(){
document.getElementsByTagName("h2")[0].style.color="yellow";
}
function change_color3(){
document.getElementById("pur_texte").style.color="blue";
}
</script>
```

```
</head>
<body>
<!-- même contenu precedent-->
</body>
```

Sélecteurs dans le DOM

La racine du document HTML est un objet appelé ***document***, auquel on peut appliquer une série de méthodes pour sélectionner des éléments particuliers du document.

On verra plus particulièrement les méthodes suivantes :

- ***document.getElementById("id_element") :*** retourne l'élément ayant comme id="*id_element*"
- ***document.getElementsByClassName("nom_class"):*** retourne une liste d'éléments ayant comme classe la classe *nom_class*.
- ***document.getElementsByTagName("nom_tag"):*** retourne une liste d'éléments ayant comme balise <*nom_tag*>.
- ***document.querySelector("selecteur_CSS"):*** peut remplacer toutes les méthodes précédentes en spécifiant un nom d'id, de classe ou de tag en notation sélecteur CSS. Elle retourne le premier élément satisfaisant le sélecteur ***selecteur_CSS***
- ***document.querySelectorAll("selecteur_CSS") :*** fonctionne comme la méthode précédente mais donne un tableau de tous les éléments satisfaisant le sélecteur ***selecteur_CSS,*** accessible par des indices.

Exemple 33 : en utilisant le document HTML suivant :

```
<body>
 <h1>Hello</h1>
 <h1>Goodbye</h1>
 <ul>
   <li id="highlight">List Item 1</li>
   <li class="bolded">List Item 2</li>
   <li class="bolded">List Item 3</li>
 </ul>
</body>
```

- `var tag = document.getElementById("highlight")`

  est équivalent à: `tag = document.querySelector("#highlight")`

  donne l'élément: `<li id="highlight">List Item 1</li>`
- `var tags = document.getElementsByClassName("bolded")`

  est équivalent à: `tag = document.querySelectorAll(".bolded")`

  donne les 2 éléments : `<li class="bolded">List Item 2</li>` et `<li class="bolded">List Item 3</li>`
- `var tags = document.getElementsByTagName("li")`

  est équivalent à: `tag = document.querySelectorAll("li");`

  donne les 3 éléments : `<li id="highlight">List Item 1</li>`
  `<li class="bolded">List Item 2</li>`
  `<li class="bolded">List Item 3</li>`

Il est également possible d'accéder aux attributs d'un élément particulier et de les modifier.

Exemple :

```
the_node=getElementById("thisNode");
the_node.setAttribute("style", "color:red");
```

2.3 Événements associés au DOM

Permettent de rendre les pages interactives. Il existe beaucoup d'événements applicables sur les éléments du DOM : Click de bouton, passer sur un lien, copier-coller, appuyer sur retour etc.

Pour prendre en charge un événement, on doit:

- Sélectionner l'élément
- Lui placer un écouteur d'événement **(event listener)** qui doit exécuter un traitement en réponse à cet événement.

Syntaxe

Pour ajouter un listener à un élément, on utilise une méthode appelée ***addEventListener()***

element.addEventListener(type, functionToCall());

Exemples:

```
<button>Click Me</button>
<p>personne ne m'a clique encore</p>
<script>
//1 affichage d'un message à la console au click sur un bouton
var b = document.querySelector("button");
b.addEventListener("click", function() {
console.log("SOMEONE CLICKED THE BUTTON!"); });

//2  Affichage d'un message dans un paragraphe au click sur un bouton
var button = document.querySelector("button");
var paragraph = document.querySelector("p");
button.addEventListener("click", function() {
paragraph.textContent = "Quelqu'un a cliqué sur le bouton!"; });
</script>
```

2.3.1 Évènements associés à la souris (Mouse Events)

Les évènements de ce type les plus courants :

- onclick – quand l'utilisateur clique sur un objet
- onmousedown - quand l'utilisateur appuie sur le bouton pour descendre la souris
- onmouseup - quand l'utilisateur appuie sur le bouton pour monter la souris

Exemple 4:

```
<html>
<body>
<script>
function good_choice() { alert("Good choice!"); }
function bad_choice() { alert("I don't agree!"); }
</script>
```

```
<h1>Click on the best social network...</h1>
<img src="facebook_icon.png" onclick="bad_choice()">
<img src="google_plus_icon.png" onclick="bad_choice()">
<img src="twitter_icon.png" onclick="good_choice()">
</body>
</html>
```

2.3.2 Les événements temporels (Timers)

Les Timers sont très utiles pour le comportement dynamique des pages web.
Un timer est déclenché comme suit:

```
var the_timer;
the_timer=setTimeout(do_something(), 1000);
```

do_something() est une fonction qui sera exécutée après une seconde (1000 millisecondes)
Exemple 5 :

```
<html>
<head>
<script>
var wait_duration;
function set_things_up() {
wait_duration = prompt("How long do you want to sleep?");
var the_timer=setTimeout(show_wake_up_message(), wait_duration );
}
function show_wake_up_message() {
alert("WAKE UP! WAKE UP! WAKE UP!!");
}
</script>
</head>
<body onload="set_things_up()">
<h1>Alarm clock example</h1>
</body>
</html>
```

Si un timer a été déclenché par :

```
var the_timer = setTimeout(do_something, 1000);
```

Alors il peut être stoppé par :

```
clearTimeout(the_timer);
```

On peut vouloir exécuter une fonction périodiquement, pour cela, on utilise ***setInterval()*** comme suit :

```
var the_timer = setInterval(do_something, 2000);
```

do_somethig sera exécutée chaque 2 secondes.
Pour l'arrêter, on utilise: clearInterval(the_timer);
Exemple 6 :

```
<html>
<body>
<button onclick="start()">Lancer le décompte</button>
<div id="bip" class="display"></div>
```

```
<script>
var counter = 20;
var intervalId = null;
function action()
{
  clearInterval(intervalId);
  document.getElementById("bip").innerHTML = "TERMINE!";
}
function bip()
{
  document.getElementById("bip").innerHTML = counter +  " secondes
restantes";
  counter--;
}
function start()
{
  intervalId = setInterval(bip, 1000);
  setTimeout(action, counter * 1000);
}
</script>
</body>
</html>
```

2.4 Gestion des traitements d'événements

On peut ajouter un événement et une fonction de traitement de cet événement à un élément de la page HTML par :

```
<html>
<body id="theBody">
<script>
function do_something() { alert("Page has loaded") }
window.onload = do_something;
</script>
</body>
</html>
```

En ajoutant un écouteur d'événement à l'élément *window*, le code précédent est équivalent à :

```
<html>
<body>
<script>
function do_something() { alert("Page has loaded") }
window.addEventListener("load", do_something);
</script>
</body>
</html>
```

Pour supprimer un traitement d'événement :

```
var theBody = document.getElementById("theBody");
theBody.removeEventListener("load", do_something);
```

Exemple 7 :

```
<html>
<body>
<button id="btn0"); ">Click Me!</button><br>
<button id="btn1">Remove Listener</button>
<script>
function do_something() { alert('Clicked'); }
var btn0 = document.getElementById("btn0");
btn0.addEventListener("click", do_something);
var btn1 = document.getElementById("btn1");
btn1.addEventListener("click", function() {
btn0.removeEventListener("click", do_something);
});
</script>
</body>
</html>
```