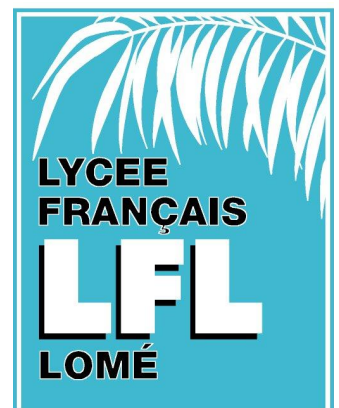


# Javascript

Travaux dirigés



# Sommaire :

<b>Introduction</b>	<b>3</b>
<b>Installation des logiciels requis</b>	<b>4</b>
<b>Les objets sous JavaScript</b>	<b>6</b>
L'objet window . . . . .	8
L'objet document . . . . .	12
L'objet style . . . . .	15
L'objet array . . . . .	16
L'objet string . . . . .	19
<b>Les événements (partie 1)</b>	<b>20</b>
<b>Les fonctions</b>	<b>22</b>
Généralités . . . . .	22
Variables locales et globales . . . . .	26
La méthode setInterval . . . . .	28
<b>les structures de contrôles</b>	<b>30</b>
Les structures conditionnelles . . . . .	31
les structures répétitives . . . . .	34
<b>Les événements (partie 2)</b>	<b>38</b>
Déclaration d'écouteurs . . . . .	38
L'objet event . . . . .	40
<b>Exemples commentés</b>	<b>42</b>
Introduction . . . . .	43
De simples boutons . . . . .	43

# I. Introduction :

Nous avons eu à rencontrer deux langages informatiques (*HTML et CSS*) mais ceux-ci n'étaient pas réellement des langages de programmation ; on parlera davantage de langages de descriptions :

- Le langage HTML décrit les éléments qui seront affichés par le navigateur.
- Le langage CSS affecte des propriétés de styles aux éléments HTML de notre page.

En aucun moment, nous avons programmé le comportement de notre ordinateur ou tout au moins du navigateur ; par contre, nous avons vu comment un navigateur recevait l'information et la mettait en forme pour l'afficher.



**JavaScript** est un langage de programmation côté client.

Cela signifie que le code s'exécute sur l'ordinateur du client.

Il existe deux types de langage de programmation Web : langage côté client ou côté serveur. Pour comprendre ces deux types de langages, on schématise la communication entre un client et un serveur de la manière suivante :

- ➔ Le client saisit une adresse dans son navigateur ou clique sur un lien hypertexte.
- ➔ Le serveur reçoit la demande et prépare la page à envoyer en réponse au client.
- ➔ Le client reçoit la page venant du serveur et l'affiche.

Voici décrit les deux types de langage de programmation Web :

- Les langages de programmation **côté serveur** : lorsque le serveur reçoit la demande d'une page par le client, des langages de programmation (*tels que PHP, ASP, Python...*) peuvent assembler les informations pour construire une page correspondant à l'attente du client : par exemple, une page renvoyée par le serveur peut dépendre des informations saisies précédemment par le client dans un formulaire, de l'heure de la journée (*clôture d'inscription*).
- Les langages de programmation **côté client** : le code du programme est inclus dans la page envoyée au client ; le code s'exécutera sur l'ordinateur du client au travers du navigateur. **JavaScript** est un langage de programmation Web côté client.

S'exécutant au travers du navigateur, ce code reçoit des informations sur les mouvements de la souris, sur l'utilisation du clavier...

L'intégration d'un script **JavaScript** dans une page Web se fait à l'aide de l'élément HTML **script**. Celui-ci peut être placé dans l'entête **head** de la page ou dans son corps **body**.

Il est possible d'écrire directement le code **JavaScript** directement dans l'élément **script** de

la manière suivante :

```
1 <script type="text/javascript">
2   ... Placez le code ici ...
3 </script>
```

L'attribut *type* permet d'indiquer la nature du code contenu dans l'élément **script** : on écrit *text/javascript* pour indiquer que le code est du texte représentant du **JavaScript** (*c'est la MIME JavaScript*). Cet attribut, bien recommandé par les standards du Web, est facultatif actuellement car **JavaScript** est le langage intégré par défaut dans les navigateurs (*Virtual Basic n'étant intégré que sur Internet Explorer*).

Il est également possible d'écrire le code dans un fichier externe et de rappeler celui-ci toujours à l'aide de l'élément **script** ; dans cette formation, on placera essentiellement le code directement dans l'élément **script**. Vous traverserez dans les manuels de cours, la manière pour placer vos codes dans un fichier externe (*manuel 3 - page 6*).

## II. Installation des logiciels requis :

Nous allons continuer la suite de logiciel dans cette partie de la formation, mais nous allons faire de légères modifications aux logiciels afin de nous apporter davantage de confort dans notre pratique de programmeur.

### Exercice 1

Emacs est un logiciel assez difficile à maîtriser mais entièrement paramétrable : au travers du fichier ".emacs" se trouvant à la racine de votre disque dur C:, on contrôle entièrement le comportement d'Emacs.

1. Ecrasez le fichier C:\.emacs par le fichier se trouvant dans le CD de la formation à l'emplacement :

```
e-javascript ~> .emacs
```



Ce fichier change la configuration d'Emacs.

Les modifications de ce fichier vont permettre d'afficher plus nettement les différentes composantes (*HTML, CSS, JavaScript, PHP*) d'un fichier HTML.

2. Copiez sur votre bureau le dossier "*f-javascriptExercice*" présent dans le CD de la formation.
3. a. A l'intérieur de ce dossier, éditez le fichier "*\_exerciceA.php*" avec le blocnote et observez le formatage de ce fichier.

b. Editez ce même fichier à l'aide d'Emacs, que remarquez-vous?

4. Dans cette page, repérez un élément `script` et, dans une nouvelle ligne, insérez le mot `abstract`. Que pouvez-vous du comportement d'emacs lorsque vous finissez de saisir ce mot?

5. Repérez une partie encadrée par les élément `<?php` et `?>` ; insérez-y, dans une nouvelle ligne, le mot `abstract`. Que remarquez-vous?



A l'ouverture d'un fichier, Emacs analyse son contenu et par un changement de couleurs, il permet de repérer plus facilement les parties suivantes :

⇒ Les **styles CSS** : elles sont définies par l'élément `style` ou par l'attribut `style` des différents éléments HTML de la page.

⇒ Le **code JavaScript** : il est contenu dans l'élément `script`.

⇒ Le **code JavaScript** : il est encadré par les balises `<?php` et `?>`.

De plus, à l'intérieur de chaque partie, Emacs va mettre en évidence des mot-clefs pour faciliter la programmation. Mais un mot-clef pour **JavaScript** ne l'est pas forcément pour PHP : le mot `abstract` ne représente rien pour PHP (*d'où la fonte grossière en rouge*) alors qu'il est reconnu en **JavaScript**.

## Exercice 2

Nous avons, dans la partie consacrée au HTML, utilisé "*HTML Validator*" qui permet, à FireFox, de donner des indications sur la validité et l'intégrité de notre code HTML.

FireFox possède la console d'erreurs disponible via la commande :

**Outils** ⇨ **Console d'erreurs**

qui permet de connaître les erreurs de codage sur le CSS et sur le **JavaScript**. Nous utiliserons également l'extension "*FireBug*" qui permet d'obtenir des renseignements précis sur notre page :

1. Nous allons installer "*FireBug*" :


a. Ouvrez FireFox et ouvrez le gestionnaire de module à l'aide de la commande :

*Outils ~> Module complémentaire ~> Extensions*

b. A partir du CD de la formation, faites un glisser/déposer du fichier :

*CD de la formation ~> e-Javascript ~> firebug-1.2.0.xpi*

c. Redémarrez FireFox pour finaliser et initialiser cette installation.

Au redémarrage, vous devez observer l'icône  suivant dans le coin inférieur droit de FireFox indiquant le lancement de FireBug.

2. Rendez-vous sur <http://google.fr> et effectuez un double clic sur l'icône de FireBug, la partie inférieure de FireFox affichera le débogueur FireBug. La (Figure 1) doit s'afficher :

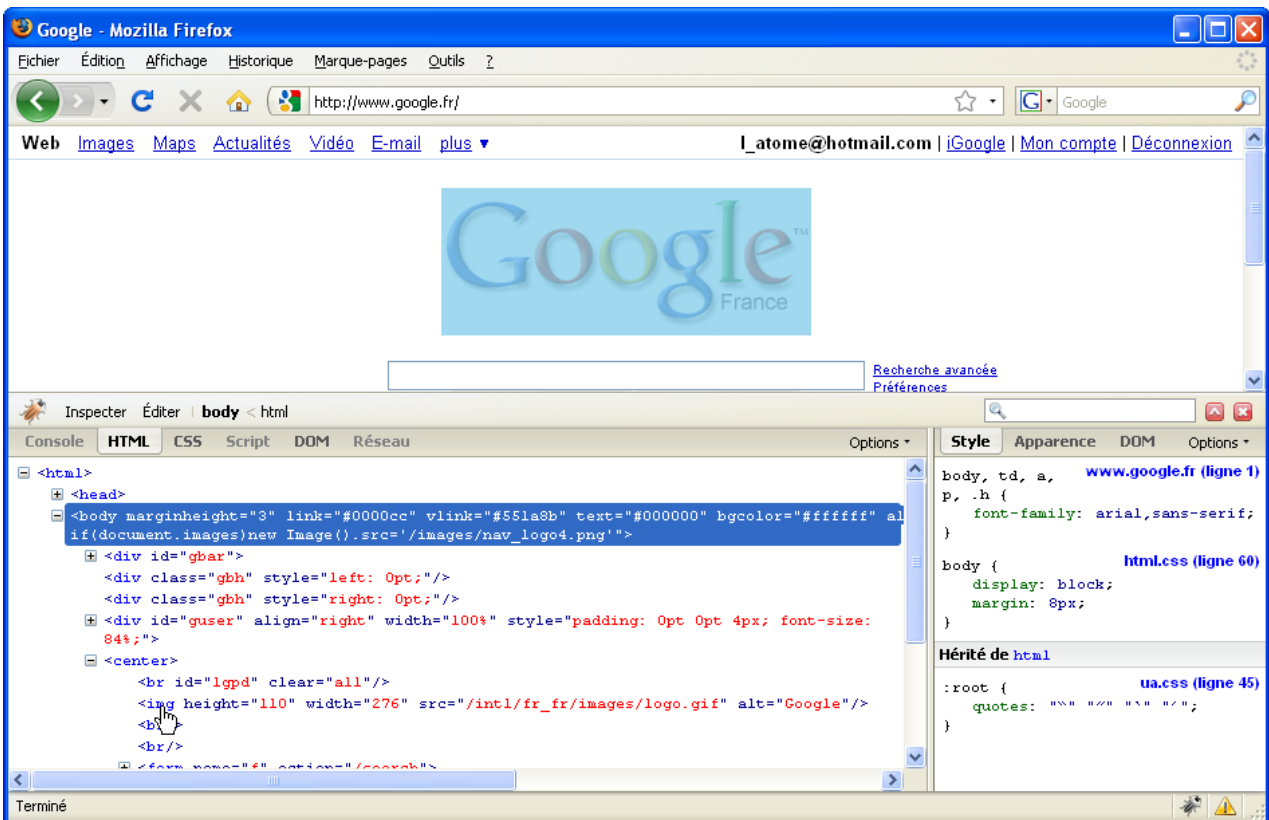


Figure 1: "L'affichage de FireBug dans FireFox"

Voici quelques explications simples sur l'utilisation de FireBug :



La partie inférieure gauche présente le code HTML de la page.

Le curseur est placé sur l'élément **img** représentant l'image de Google ; automatiquement l'image de l'affichage se surligne pour indiquer l'élément pointé.

La partie inférieure droite présente les propriétés CSS appliquées à l'élément pointé.

## III. Les objets sous JavaScript :

**JavaScript** est un langage de programmation orientée objet ; nous ne rentrerons pas dans le détail de ce concept au cours de cette formation, mais il faut savoir que la programmation consiste à utiliser des objets ; ces objets contenant eux-même des objets et des propriétés.

- La fenêtre du navigateur est représentée par l'objet **window** en **JavaScript**. A travers cet objet de **JavaScript**, on peut manipuler certaines caractéristiques de la fenêtre du navigateur telles que ses dimensions, son emplacement sur l'écran du client ...
- L'affichage principal du navigateur est un objet dénommé **document** ; c'est un sous-objet de **window**, on accède à cet objet à l'aide du code suivant :

```
window.document
```

Cet objet permettra en particulier de saisir tous les éléments HTML présents sur une page et de les modifier dynamiquement.

Le diagramme suivant (Figure 2) présente une petite partie des objets **JavaScript** et de leurs descendance

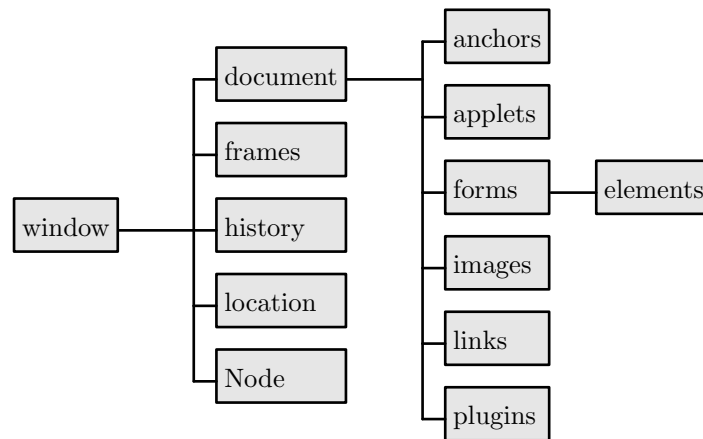


Figure 2: "Exemple de descendance d'objets JavaScript"

Voici quelques exemples d'utilisation des objets présentés ici :

- L'objet **images** est un tableau comprenant les images déjà chargées par le navigateur. Il permet aussi de pré-charger les images avant des les afficher. On accède à la première image de la page au travers du code :

```
window.document.images[0]
```

- L'objet **forms** est un tableau référençant les formulaires déjà chargés dans la page. On accède au premier formulaire de la page à l'aide du code suivant :

```
window.document.forms[0]
```

- ⇒ Ayant accéder à un formulaire, on accède aux différents contrôles de ce formulaire via l'objet **elements** :

```
window.document.forms[0].elements[2]
```

Le code précédent représente le troisième champ du première formulaire de la page affichée dans le navigateur (*s'il existe*)

Chaque objet **JavaScript** possède :

- ⇒ des **propriétés** caractérisants son état.

La propriété **defaultStatus** de l'objet **window** représente le texte affiché dans la barre

d'état de la fenêtre ; le code suivant aura pour effet (*si votre navigateur le permet*) de modifier la barre d'état du navigateur : `window.defaultStatus="Bonjour!";`

⇒ des **méthodes** qui sont des fonctions agissant sur l'objet même.

Ainsi, la méthode `write` de l'objet `document` permet d'écrire du texte à l'intérieur de cet objet ; c'est à dire dans la fenêtre d'affichage du navigateur

```
window.document.write("En plus");
```

Les mots “*En plus*” seront ajoutés à la page lors de l'exécution de ce script.

## A. L'objet `window` :

L'objet `window`, comme le montre le schéma présenté ci-dessus est l'objet, de départ : tout autre objet est nécessairement un sous-objet de celui-ci ; c'est l'*objet global* du langage **JavaScript**.

Principalement, il représente la fenêtre du navigateur. Au travers des exercices suivant, nous allons voir quelques unes de ses propriétés et méthodes qui nous permettrons de contrôler le navigateur au travers de cet objet :

### Exercice 3

Cet exercice présente quelques propriétés de l'objet `window` :

1. Nous allons utiliser le code suivant tout au long de cet exercice :

a. Créez une page Web vide sur le bureau. Pour cela, créez un nouveau document texte :

*menu contextuel* ~> *Nouveau* ► ~> *Document Texte*

Renommez ce fichier avec une extension “.html”.

b. Editez ce fichier avec Emacs, puis saisissez le code ci-dessous :

```
1 <html>
2   <head>
3     <title>Bonjour</title>
4   </head>
5   <body>
6     Yep,
7     <div id="boite" style="font-style: italic">
8       salut
9     </div>
10    Agnès
11    <script type="text/javascript">
12    </script>
13  </body>
14 </html>
```



c. Dans Emacs, actionnez la commande suivante :

`MMM ~> Parse Buffer`

Que s'est-il passé?

.....

.....

d. Appuyez sur “F1” pour afficher votre page dans FireFox et “F2” pour l'afficher dans Internet Explorer.



Lors de l'ouverture d'un fichier, Emacs scrute le document et y repère les parties représentant du CSS, du **JavaScript**, du PHP.

Si vous rajoutez manuellement une de ses parties, pour que Emacs la repère, il faut activer la commande précédente ou alors fermer et ré-ouvrir son document.

2. a. A l'intérieur de l'élément `script` de notre page, inscrivez, sur une nouvelle ligne, le code suivant :

```
window.innerWidth=100
```

b. Afficher la page dans un navigateur en appuyant sur la touche “F1”. Que remarquez-vous?

.....

.....

c. Appuyez sur la touche “F2” pour afficher votre page à l'aide d'Internet Explorer. Que remarquez-vous?

.....

.....



`innerHTML` est une propriété de l'objet `window`, ainsi on accède à cette propriété en écrivant `window.innerWidth`.

Cette propriété n'existe que pour FireFox. Elle contient la largeur courante de la fenêtre du navigateur. En lui affectant une nouvelle valeur à l'aide de l'opérateur “=”, on modifie la largeur de la fenêtre du navigateur.

Voici trois opérateurs fréquemment utilisés dans le cas de variable *numérique* :

- L'opérateur “=” permet d'affecter une valeur à une variable ; il est appelé **opérateur d'affectation** ; la variable se trouve à gauche et la valeur se situe à droite :

```
a=3
```

Le code ci-dessous affecte la valeur “3” à la variable `a`.

- L'opérateur “+=” permet d'augmenter la valeur d'une variable d'une certaine quan-

tité ; il est appelé **opérateur d'incrément** :

`a+=5` est équivalent à `a=a+5`.

Dans l'exemple ci-dessus, la variable `a` voit sa valeur augmentée de 5.

- L'opérateur "`--`" permet de réduire la valeur d'une variable ; il est appelé **opérateur de décrémentation** :

`a-=2` est équivalent à `a=a-2`.

L'exemple ci-dessus permet de réduire la valeur de la variable `a` de 2.

Pour en savoir plus sur les opérateurs et pour connaître les opérateurs n'opérant pas seulement sur les valeurs numériques, reportez-vous au premier manuel de cours à la page 26.

3. a. Modifiez la ligne précédente de l'élément `script` pour obtenir :

```
window.innerWidth+=100
```

- b. Visualisez la page avec Firefox. Puis, appuyez successivement sur la touche "`F5`" pour actualiser le contenu de la page. Que remarquez-vous?

4. a. Remplacez le contenu de l'élément `script` par le code suivant :

```
window.moveBy(10,5)
```

- b. Testez ce code avec Firefox et Internet Explorer. Vous utiliserez la "`F5`" pour recharger plusieurs fois la page et ainsi exécuter plusieurs fois le script. Que remarquez-vous?



`moveBy()` est une méthode de l'objet `window`.

Elle existe aussi bien sur Internet Explorer que sur Firefox ; elle permet de déplacer la fenêtre du navigateur.

Elle prend deux arguments qui sont des nombres entiers, sa syntaxe d'utilisation est :

```
window.moveBy(x,y)
```

Le déplacement se fera de  $x$  pixels horizontalement (*pour une valeur positive vers la droite et négative vers la gauche*) et de  $y$  pixels verticalement (*vers le bas pour une valeur positive et vers le haut pour une valeur négative*).

## Exercice 4

Il est assez difficile de détecter les erreurs dans un code **JavaScript** ; cet exercice nous montre une technique pour déceler le moment où un code **JavaScript** cesse de s'exécuter.

Dans cet exercice, nous allons utiliser de nouvelles méthodes de l'objet `window`, mais de nombreuses autres propriétés et méthodes sont accessibles pour cet objet `window` ; vous en trouverez une partie sur le troisième manuel - page 7.

1. a. Dans un fichier ".html", introduisez un élément `script` et saisissez à l'intérieur de celui-ci le code suivant :

```
    window.alert("Bonjour");
```

- b. Affichez la page dans un navigateur. Que se passe-t-il?

2. Modifiez le fichier pour obtenir le code suivant :

```
1 <script >
2   window.alert("1° Appel");
3   alert("2° Appel");
4 </script >
```

Affichez la page dans un navigateur, que pouvez-vous dire sur l'exécution de ces deux lignes?



Les règles et l'essentiel du langage **JavaScript** est défini par la norme ECMA-262 ; ce langage est également utilisé dans les logiciels Acrobat Reader, Photoshop et Flash (*sous le nom de `actionScript`*).

Dans son implémentation pour les navigateurs, l'objet `window` représente l'objet global du langage ; Il est possible de l'omettre : sa présence est implicite dans la seconde ligne du code précédent.

Les deux séquences suivantes sont équivalentes :

```
    alert("...") ; alert("window.alert(...)")
```

3. a. Modifier votre code pour obtenir :

```
1 <script type="text/javascript">
2   alert("premier appel");
3   window.move(10);
4   alert("second appel");
5 </script >
```

b. Ouvrez cette page dans un navigateur. Quelle partie du code ne s'est pas exécutée?

c. Ouvrez la console d'erreurs de FireFox à partir de la commande suivante :

**Outils** ~> **Console d'erreurs**

En actionnant cette commande, FireFox affiche une boîte à dialogue (Figure 3) où, en dernière position, apparaît l'erreur du code **JavaScript**.

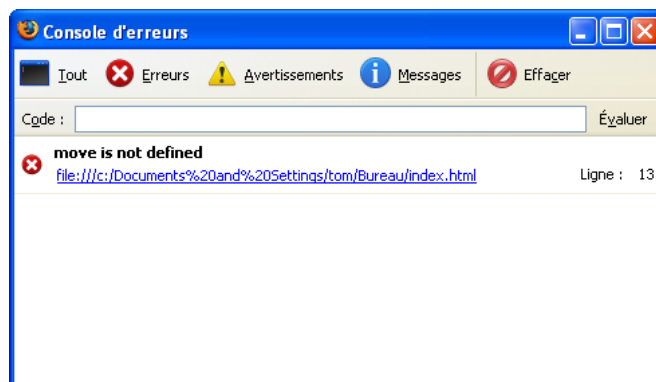


Figure 3: "Console d'erreurs de FireFox"



L'objet `window` ne possède pas de méthode dénommée "move()". Ainsi, la seconde ligne du code provoque une erreur et empêche l'exécution du reste du script : ceci permet de repérer l'erreur.

On utilise souvent cette méthode pour déboguer un programme **JavaScript**.

Les interpréteurs de **JavaScript** ont pour particularité d'arrêter l'exécution du code dès l'apparition d'une erreur.

Ainsi, lorsqu'un programme ne s'exécute pas de la manière attendue, on parseme son code de cette propriété pour repérer où l'exécution du script s'est interrompu.

La suite de l'exercice illustre cette situation.

## B. L'objet document :

L'objet `document` est un des objets les plus importants de **JavaScript** ; il représente la zone d'affichage du navigateur. C'est par lui qu'on accède à tous les éléments HTML de la page.

Nous allons parcourir quelques-unes de ses propriétés et méthodes, le reste est à découvrir dans le troisième manuel de cours, page 11

### Exercice 5

Cet exercice présente deux méthodes utiles pour modifier le texte représenté par le navigateur

1. La propriété `title` :

- a. Ouvrez une page “.html” à l’aide de Emacs, puis assurez-vous de vider tout élément `script` de la page et que celle-ci ne comporte aucun élément `title` dans l’entête de la page.
- b. Affichez la page dans Firefox. Quel est le titre de la page :

- c. Dans l’élément `script`, insérez le code suivant :

```
window.document.title="Bienvenu"
```

Puis, affichez la page dans Firefox et Internet Explorer. Que représente la propriété `title` de l’objet `document`?

- d. Changez le code en :

```
document.title="Au revoir";
```

L’exécution de la page entraîne-t-elle une différence?

- e. Compléter le code pour obtenir :

```
1 document.title="Bonjour";
2 alert("Changement de titre");
3 document.title="Au revoir";
```

## 2. La méthode `write()` :

- a. Dans une page “.html”, inscrivez le code suivant :

```
1 <script type="text/javascript">
2 document.write("<p>Bonjour,");
3 alert("1° interruption");
4 document.write(" à tous<br>");
5 alert("2° interruption");
6 document.write("Comment allez-vous?");
7 </script >
```

- b. Testez votre page dans un navigateur.



Dans cette exercice, nous commençons à voir comment le code **JavaScript** peut modifier le contenu d’une page Web. Les différentes méthodes `alert()` insérées dans le code permettent d’interrompre l’exécution du code afin d’observer pas à pas les changements effectués par **JavaScript**.

- La propriété `title` a pour valeur une chaîne de caractères représentant le titre de la page affiché dans la fenêtre du navigateur.

- La méthode `write()`, lors de son exécution, permet de rajouter du texte mais également du contenu à l'élément `body`.

## Exercice 6

**JavaScript** peut également modifier l'affichage d'une page Web en prenant le "contrôle" des éléments HTML qui la composent. Il existe plusieurs manières par laquelle **JavaScript** peut dialoguer avec les éléments HTML ; dans cet exercice, **JavaScript** va se servir de l'attribut `id` des éléments HTML. La méthode `getElementById()` de l'objet `document` permet de désigner un élément à l'aide de son attribut `id`.

Il est bon de rappeler que la valeur de l'attribut `id` d'un élément doit être unique dans une page Web ; la valeur de `id` permet de désigner un unique élément dans une page.

1. Editez le fichier "`_exerciceB.html`" à l'aide d'émacs, et repérez les deux éléments HTML ayant pour l'attribut `id` affecté respectivement des valeurs `monTitre` et `ici`.
2. Insérez le code suivant dans l'élément `script` :

```
1 <script type="text/javascript">
2 alert("1° interruption");
3 document.getElementById("monTitre").innerHTML="Au ↔
   revoir";
4 alert("2° interruption");
5 document.getElementById("ici").innerHTML="seconde";
6 </script >
```

3. Affichez la page dans un navigateur et regardez le contenu de votre page changer au fur et à mesure que le code **JavaScript** s'exécute.
4. a. Déplacez l'élément `script` dans l'entête de la page : c'est à dire à l'intérieur de l'élément `head`.  
b. Affichez à nouveau votre page dans Firefox. Qu'observez-vous? Quelles premières conclusions tirées vous de cette exécution?

- c. Pour comprendre ce qui se passe, ouvrez la console d'erreurs dans Firefox :

**Outils ↔ Console d'erreurs**

et rechargez votre page.



Intéressons nous à l'exécution du code :

```
document.getElementById("valId"
```

- ➔ La page courante contient un élément ayant *valId* pour valeur de l'attribut *id* ; le code précédent renvoie une référence à l'objet ciblé.
- ➔ La page courante ne contient aucun élément ayant *valId* pour valeur de l'attribut *id* ; le code renvoie l'objet `null` qui ne possède aucune propriété.

Revenons maintenant aux questions précédentes de la page et à l'exécution du code :

```
document.getElementById("monTitre").innerHTML="Au revoir";
```

- ➔ A la question 3., lors de l'exécution de ce code, il existe un élément ayant *monTitre* pour l'attribut *id* ; Ainsi, le code :

```
document.getElementById("monTitre")
```

renvoie la référence d'un élément HTML ; celui-ci possède la propriété `innerHTML` qui représente son contenu.

- ➔ A la question 4., à cause du déplacement du script, celui s'exécute alors qu'aucun élément de la page n'a l'attribut *id* fixé avec la valeur *monTitre* ; ainsi, le code ci-dessous renvoie l'objet `null` qui ne peut posséder de propriété :

```
document.getElementById("monTitre")
```

### C. L'objet `style` :

En **JavaScript** presque tous les éléments HTML possèdent le sous-objet `style` représentant l'ensemble des styles CSS rattachés à l'élément.

Ainsi, après avoir récupéré un objet à l'aide de la méthode `getElementById()`, on utilise la propriété `style` pour affecter dynamiquement des changements de style à l'élément :

1. Editez de nouveau le fichier "`_exerciceB.html`" ; effacer tous les éléments `script` de la page afin d'en placer un nouveau en fin de document à l'intérieur duquel vous saisirez le code suivant :

```
1 <script type="text/javascript">
2   var eltTitre=document.getElementById("monTitre");
3   var eltSpan=document.getElementById("ici");
4   alert("1° interruption");
5   eltTitre.style.fontSize="24pt";
6   alert("2° interruption");
7   eltTitre.style.margin="20px 100px";
8   eltTitre.style.textAlign="center";
9   alert("3° interruption");
10  eltSpan.style.textDecoration="overline underline";
11 </script >
```

2. Affichez la page dans un navigateur pour observer les changements de styles opérées au fur et à mesure de l'exécution du code **JavaScript**.



Voici quelques remarques sur l'exercice précédent :

- Dans le code précédent, on manipule les propriétés de styles des éléments ayant *monTitre* et *ici* à l'aide de la méthode `getElementById()` ; cette méthode renvoie une référence à l'objet pointé.  
Plutôt que d'effectuer plusieurs fois cet appel, ici, on a préféré stocker la référence de ces objets dans les variables `eltTitre` et `eltSpan`.
- Si vous déplacez l'élément `script` en début de page, aucune modification de style puisque l'exécution du code s'arrêtera à la ligne :  

```
eltTitre.style.fontSize="24pt";
```

Puisque la variable `eltTitre` contient l'objet `null` qui ne contient aucune propriété.
- Vous trouverez dans le troisième manuel de cours à la page 48 un tableau de correspondance entre les propriétés CSS et le nom de la propriété correspondant pour l'objet **JavaScript** `style`.

## D. L'objet `array` :

Les tableaux, en informatique, représentent une collection d'objet : il n'existe que pour contenir plusieurs objets, pas nécessairement de même nature, en son intérieur.

Ainsi, ils servent à stocker et à restituer des objets. Un tableau en **JavaScript** n'est pas limité par la taille : une fois créé, il est possible de rajouter ou d'enlever des objets à ce tableau.

En **JavaScript**, les éléments d'un tableau sont numérotés ; la numérotation des éléments commence à zéro :

1. Le premier élément d'un tableau a pour rang 0.
2. Le second élément d'un tableau a pour rang 1.

Les tableaux font partie du noyau de **JavaScript** ; une description complète des tableaux, ainsi que les propriétés et les méthodes de ces objets, est donné dans le second manuel de cours à la page 8.

## Exercice 7

1. a. Dans un fichier ".html", saisissez le code suivant :

```
1 <script >
2 var tab = new Array("Paul",5);
```



```
3 alert("Premier élément : "+tab[0]);
4 alert("Deuxième élément : "+tab[1]);
5 alert("Troisième élément : "+tab[2]);
6 </script>
```

b. Exécutez-le dans un navigateur



On crée la variable `tab` représentant un tableau à deux éléments à l'aide du code :

```
var tab = new Array("Paul",5);
```

Lors de l'exécution du code :

- on se rend compte que le mot "*Paul*" est le premier d'élément du tableau mais que son index est 0.
- L'appel à un index non-défini renvoie la valeur `undefined`

Pour adjoindre aux chaînes de caractères passées en argument à la méthode `alert`, on a utilisé l'opérateur `+` de concaténation. **JavaScript** effectue automatiquement le changement de type :

- Le nombre entier 5 est automatiquement changé en la chaîne de caractères "5".
- la valeur `undefined` est transformée en "*undefined*".

Lorsque l'opérateur `+` a un de ses arguments qui est une chaîne de caractère, il a pour fonction de concaténer les chaînes ; c'est à dire de les mettre bout à bout.

2. a. Insérer après la déclaration du tableau `tab` la ligne de code suivante :

```
tab.reverse();
```

b. Exécutez le code en affichant la page dans un navigateur. Que remarquez-vous?

3. a. Rajoutez à la suite de la ligne de code précédente le code :

```
tab.push("Bruno");
```

b. Quel est l'effet de cette commande?



Nous venons de rencontrer deux méthodes permettant

- `tab.reverse()` : cette méthode renverse l'ordre des éléments dans le tableau.
- Pour rajouter un élément en fin de tableau, on utilise la commande :  

```
tab.push(obj)
```

Reportez-vous au second manuel de **JavaScript**, pour connaître d'autres méthodes utilisables sur les tableaux.

4. a. Modifiez votre code en celui-ci :

```
1 <script >
2 var tab = new Array("Paul","Marie","Jacques");
3 alert("Nbr élément : "+tab.length);
4 tab[3]="Michel";
5 alert("Nbr élément : "+tab.length);
6 tab[10]="Arthur";
7 alert("Nbr élément : "+tab.length);
8 document.write(tab[9]);
9 </script >
```

b. Exécutez le code en affichant la page dans un navigateur. Que représente pour vous la propriété `length` d'un tableau?

.....  
.....



Nous venons de voir qu'il est possible d'affecter directement une valeur à un index du tableau avec un code de la forme :

Contrairement, la propriété `tab.length` d'un tableau est le plus grand index représentant un élément ne valant pas `undefined`. A la fin de l'exécution du code précédant, le tableau ne contient que cinq éléments ; les index 4 à 9 représentant a la valeur `undefined`.

### Exercice 8

Ici, nous allons utiliser l'objet `images` qui est un sous-objet de `document` pour modifier dynamiquement les images présentes dans notre page. Cet objet référence toutes les images présentes dans le document.

1. Editez le fichier "`_exerciceC.html`" et affichez-le dans un navigateur.

2. Saisissez le code suivant au contenu de l'élément `script` :

```
1 <script type="text/javascript">
2   alert("1° interruption");
3   document.images[0].src="";
4   alert("2° interruption");
```

```
5 document.images[0].src="guanajuato-mexique.jpg";
6 alert("3° interruption");
7 document.images[0].width="200";
8 alert("4° interruption");
9 document.images[0].height="100";
10 </script >
```

3. Affichez la page et expliquez l'effet des lignes 3, 5, 7 et 9 du code ci-dessous :

.....

.....

.....

.....

.....



`document.images` est un tableau contenant les objets **JavaScript** représentant toutes les images présentes dans le document lors de son appel.

Ainsi, `document.images[0]` représente la première image. Tout objet image en **JavaScript** possède les propriétés :

⇒ `img.src` représentant l'URL indiquant l'emplacement de l'image.

⇒ `img.width` et `img.height` représentant respectivement sa longueur et sa largeur exprimées en pixel.

## E. L'objet string :

Les chaînes de caractères sont très utilisées, ne serait-ce que pour afficher du texte dans le navigateur. Cette exercice va nous montrer comment manipuler les chaînes de caractères sous **JavaScript**.

### Exercice 9

1. Dans une page ".html", saisissez le code suivant :

```
1 <script type="text/javascript">
2 var ch="Aujourd'hui, il fait chaud";
3
4 alert("Nbr de caractères : "+ch.length);
5 alert("3ième caractères : "+ch.charAt(2));
6 alert("Recherche : "+ch.indexOf("jour"));
7 alert("Recherche : "+ch.indexOf("bonjour"));
```

```
8 alert("Extraction : "+ch.substr(16,4));
9 </script >
```

2. Exécutez le code pour observer la fonction de chacune des méthodes et propriétés ci-dessus.



- La propriété `ch.length` contient le nombre de caractères formant la chaîne de caractères.
- La méthode `ch.charAt(i)` renvoie le caractère de la chaîne `ch` d'index `i`. L'indexage d'une chaîne de caractère en **JavaScript** commence par le nombre zéro.
- La méthode `ch.indexOf(str)` cherche la chaîne de caractères `str` dans la chaîne `ch` :
  - ➔ Si `str` est contenue dans `ch` alors cette méthode renvoie l'index dans `ch` du premier caractère de `str`.
  - ➔ Si `str` n'est pas présent dans `ch`, alors elle renvoie `-1`.
- La méthode `ch.substr(i,n)` permet d'extraire de `ch` la sous-chaîne démarrant au caractère d'index `i` et ayant une longueur de `n`.

3. Remplacez le code par :

```
1 <script >
2 var a=prompt("Rentrez une phrase :");
3 document.write(a.toUpperCase());
4 </script >
```

4. Exécutez le code. Quelles sont les fonctions des méthodes `prompt()` et `toUpperCase()`

.....

.....

.....



La méthode `prompt`, appartenant à l'objet `window`, ouvre une boîte à dialogue possédant un champ de texte et permettant au client de saisir du texte.

Lors de la validation de cette boîte, la chaîne saisie par le client est retournée à la variable `var`.

La méthode `ch.toUpperCase()` permet de passer l'ensemble des caractères alphabétiques de la chaîne `ch` en majuscule.

## IV. Les événements (partie 1) :

**JavaScript** est un langage côté client ; s'exécutant sur la machine du client, il permet de rendre notre page dynamique (*DHTML - dynamic HTML*) en repérant les actions du client :

- Le passage de la souris au dessus d'un élément HTML ;
- l'utilisation du clavier...

On appelle ces actions des **événements** ; le **JavaScript** permet de lier les événements intervenant dans la page à l'exécution de code.

Dans ce chapitre, nous allons voir comment intercepter les événements et y associer l'exécution de code mais de manière très simple. L'utilisation des fonctions nous permettra d'utiliser plus agréablement les événements ; un second chapitre sera consacré aux événements (*voir page 38 de ce manuel*).

### Exercice 10

1. Editez le fichier “\_exerciceD.html” avec Emacs. Repérez la structure de la page et notamment les éléments **div** possédant l'attribut *id*.

2. Appuyez sur “F1” pour visualiser la page dans un navigateur.

3. a. Rajoutez dans la balise ouvrante de l'élément b1 l'attribut HTML suivant :

```
onmouseover="document.getElementById('b2').style.backgroundColor='red' "
```

b. Affichez la page et passez votre curseur sur les différents boutons de la page pour voir l'effet de votre code. Que remarquez-vous?

.....  
.....  
Utilisez la touche “F5” pour revenir, par moment, à l'affichage initial de la page.

c. Dans le code, changez `getElementBy('b2')` en `getElementBy('b1')` et exécutez de nouveau le code.



L'attribut HTML *onmouseover* permet de capturer l'événement :

“*Le curseur du client passe au dessus de l'élément*”

et d'exécuter le code **JavaScript** qu'il contient pour valeur ; ici, un changement de style d'un des deux boutons.

4. a. Rajoutez dans l'élément b1 l'attribut HTML suivant :

```
onmouseout="this.style.backgroundColor='';
```

Ici, la valeur de la propriété `backgroundColor` est une chaîne vide.

- b. Exécutez le code et faites passer votre curseur par dessus les différents éléments de la page. Que remarquez-vous?



- L'attribut HTML *onmouseout* permet de capturer l'événement :  
"Le curseur du client quitte l'élément"
- On remarquera l'absence de la méthode `getElementBy()` remplacée par le mot-clef `this` ; il permet de donner une référence directe à l'élément HTML qui lance l'exécution du script.
- Ici, la propriété `style` d'un élément HTML permet d'agir sur le style *interne* de l'élément ; en donnant une chaîne vide à la propriété de style :

`style.backgroundColor`

l'élément retrouve son style interne défini dans sa balise ouvrante ; dans le cas de notre élément, sa couleur de fond n'est pas définie par son style interne mais par la feuille interne de la page. Pour vous assurer que le style interne est rétabli en attribuant une chaîne vide à la propriété de **JavaScript**, vous pouvez rajouter l'attribut HTML suivant à notre élément HTML :

`style="background-color:green"`

5. a. Placez dans l'élément `body` de la page l'attribut suivant :

`onkeypress="alert('une touche a été enfoncée')"`

- b. Exécutez le code et appuyez sur une touche.



L'attribut HTML *onkeypress* permet de capturer l'événement :

"Une touche du clavier est enfoncée"

Cet attribut peut être rattaché à l'élément `body` ou aux éléments `input` et `textarea` de type champ de texte. Par propagation des événements (*phénomène un peu dur à appréhender - voir troisième manuel de cours à la page 41*), cet événement peut affecter presque tous les éléments HTML.

Nous verrons dans le second paragraphe sur les événements comment connaître quelle touche a été enfoncée.

## V. Les fonctions :

## A. Généralités :

Lors de la programmation, il arrive qu'une partie du code doit s'exécuter plusieurs fois. Il est alors plus facile de placer ce bout de code à un seul endroit et d'invoquer plusieurs fois son appel : on crée alors un sous-programme.

Une autre utilité des fonctions est de permettre de séparer le code HTML du code **JavaScript** ; dans le chapitre précédent, le code **JavaScript** était intégré directement dans la balise ouvrante de l'élément recevant l'événement ; il est plus agréable de placer le code **JavaScript** extérieurement au code HTML pour ne pas surcharger ce dernier.

Ces sous-programmes s'appellent actuellement des **procédures** ou des **fonctions** ; ce dernier se différencie du premier seulement par le fait qu'il renvoie une valeur mais la terminologie informatique peut varier d'un langage de programmation à l'autre.

**JavaScript** regroupe ces deux notions au travers du mot-clef **function**. Le mot-clef **return** permet à la fonction de retourner une valeur au programme l'ayant appelé entraînant la fin de l'exécution de cette fonction.

Une fonction peut également recevoir des paramètres qu'elle utilisera lors de son exécution ; une fonction peut également agir sur les variables extérieures. Il est également possible lors de l'appel à l'exécution d'une fonction de lui passer des paramètres, modifiant ainsi le comportement de la fonction. Nous verrons ainsi dans le prochain chapitre les différentes structures de contrôle permettant de contrôler le comportement d'une fonction suivant les paramètres reçus lors de son appel mais également suivant la valeur des variables extérieures.

L'utilisation des fonctions s'est imposé dans tous les langages de programmation, elles permettent :

- Une meilleure structuration du code ; pour **JavaScript**, en plaçant les fonctions dans l'entête **head**, on sépare davantage la présentation (*code HTML*) de la programmation (*code JavaScript*)
- La maintenance du code source est amélioré ; en regroupant les morceaux de codes redondants, on garde une meilleure lisibilité de l'ensemble du code ; lors du changement du programme, il n'est plus nécessaire de modifier le code à plusieurs endroits.
- Des fonctions ayant des usages généraux peuvent être placées dans un fichier externe et être utilisées par plusieurs pages Web.



On déclare une fonction à l'aide du mot-clef **function** suivi du nom de la fonction.

Voici un exemple d'utilisation :

```

1  function nomFonction (x,y){
2     ... Corps de la fonction ...
3  }
4
5  nomFonction(" abc ",7);

```

Le corps de la fonction est l'ensemble des instructions exécutées à chaque appel de la fonction.

x et y s'appellent les paramètres (ou *paramètres formels*), ils représentent les deux valeurs passées à la fonction lors de son appel ; la fonction agira sur ces valeurs dans son corps à l'aide des paramètres x et y les représentants respectivement.

La ligne 5 du code ci-dessus illustre un appel à cette fonction ; deux arguments (également appelés *paramètres effectifs*) sont passés à la fonction lors de son appel :

- Le paramètre formel x prend la valeur du premier argument : une chaîne de caractère.
- Le paramètre formel y prend la valeur du second argument : un nombre entier.

## Exercice 11

Ce premier exercice va nous permettre d'illustrer l'utilisation des fonctions. Vous remarquerez que d'écrire le code à exécuter hors de la balise ouvrante allège la lecture du code source.

1. Saisissez le code ci-dessous dans une page ".html" vide :

```

1  <script type="text/javascript">
2  function clique(x){
3     alert("Vous avez cliqué sur : "+x);
4  }
5  </script >
6
7  <div onclick="clique('1')">Un</div>
8  <div onclick="clique('2')">Deux</div>
9  <div onclick="clique('3')">Trois</div>

```

2. A la lecture de ce code, anticipez le comportement du navigateur :

a. A quel événement réagissent nos trois éléments **div** :

.....  
 .....

b. Quel est le nom de la fonction appelée? Quel est le nom de son paramètre formel?



.....  
.....  
c. Expliquez le rôle de l'opérateur "+" à la ligne 3.

.....  
.....  
d. Que se passe-t-il si on clique sur l'élément `div` contenant "Un"?

3. Affichez cette page dans un navigateur afin d'exécuter ce code et de vérifier vos affirmations.



Chaque élément `div` de cette page réagit avec l'événement `onclick` pour exécuter la fonction `clique`.

L'argument passé lors de l'apparition de l'événement est le numéro du `div` ; celui-ci s'affichera dans une boîte de dialogue au travers de la méthode `alert`.

4. a. Modifiez votre élément `script` de la manière suivante :

```
1 <script >
2   function clique(x){
3     alert("Le contenu de l'élément est : "+x.innerHTML);
4   }
5 </script >
6
7 <div onclick="clique(this)">Un</div >
8 <div onclick="clique(this)">Deux</div >
9 <div onclick="clique(this)">Trois</div >
```

Puis, exécutez ce code en affichant votre page dans un navigateur.

b. En cliquant sur chacun des éléments `div`, dites la particularité de chacune des boîtes à dialogue s'ouvrant :

.....  
.....  
c. Que représentent, d'après vous, le mot-clef `this` présent dans la valeur de chaque attribut `onclick`?



Le mot clef **this** représente l'élément HTML recevant l'événement *onclick* et lançant l'exécution de la fonction `clique()` ; une référence à cet élément est passé en argument au paramètre formel **x** à chaque appel de la fonction.

Ainsi, **x** représente l'objet sur lequel on a cliqué et la propriété `x.innerHTML` représente son contenu.

## B. Variables locales et globales :

Comme pour le corps du programme, on utilise les variables à l'intérieur d'une fonction pour effectuer des calculs, travailler sur des chaînes de caractères... Mais les fonctions en **JavaScript** peuvent également agir sur les variables extérieures ; il faudra alors bien faire la distinction entre :

- Les **variables locales** qui sont définies dans le corps d'une fonction dont la durée de vie est limité au temps d'exécution de la fonction.
- Les **variables globales** qui sont définies dans le corps même du code **JavaScript** et seront accessibles par toutes les autres fonctions ; les variables globales pourront notamment servir de communication entre les différentes fonctions.



Le langage **JavaScript** est un langage faiblement typé ; une variable peut successivement prendre des valeurs de types différents (*être un nombre puis une chaîne de caractères...*).

De même, certains oublis comme le point virgule en fin d'instructions peut être remplacé par un saut de ligne. Ce manque de rigueur entraîne une légèreté et une souplesse dans la programmation, mais ce manque de rigueur peut entraîner des erreurs parfois dures à déceler.

En **JavaScript**, on déclare proprement une variable à l'aide du mot-clef **var** mais celui-ci peut être omis ; attention alors à la confusion pouvant naître entre l'utilisation d'une variable qu'on imaginait locale mais qui est en fait globale. Voici trois instructions à bien comprendre :

⇒ `var a;` : ceci est une déclaration de la variable **a** ;

⇒ `var a=3;` : ce code représente la déclaration de la variable **a** suivie directement de l'affectation de la valeur ;

⇒ `a=3;` : ceci est l'affectation de la valeur 3 à la variable **a**.

Ainsi, le troisième code peut s'avérer incorrecte si l'affectation est effectuée avant la déclaration ; mais **JavaScript** va prendre la déclaration à son compte et va déclarer cette fonction dans le corps du code (*faisant de la variable **a** une variable globale*)

même si cette affectation, sans déclaration, s'effectue dans le corps d'une fonction.

Donc, pour qu'une variable soit locale à une fonction, il est nécessaire de la déclarer à l'aide du mot-clef `var`.

Considérons le code ci-dessous :

```
1 <script >
2  function nomFonction() {
3    b=3;
4    var a=2;
5  }
6 </script >
```

⇒ La variable `b` n'ayant pas été déclarée explicitement, `b` est une variable globale ;

⇒ La variable `a` est définie comme locale à la fonction.

Si une variable globale `a` existe et qu'on déclare explicitement une variable locale dénommée également `a` dans le corps d'une fonction, alors la variable utilisée est la variable locale.

Il est néanmoins possible d'accéder à la variable globale à l'aide du code :

```
window.a
```

Une variable globale est considérée, en **JavaScript**, comme une propriété de l'objet global `window`.

## Exercice 12

1. Etudiez le code ci-dessous avant de répondre aux questions suivantes :

```
1 <script type="text/javascript">
2  var a=3;
3  var b=5;
4
5  function test() {
6    var a=2;
7    document.write("a : "+a+"<br>b : "+b);
8    b=7;
9  }
10
11 document.write("a : "+a+"<br>b : "+b+"<p>");
12 test();
13 document.write("<p>a : "+a+"<br>b : "+b);
14 </script >
```

a. A la ligne 11, le script affiche dans la page les valeurs des variables a et b. Quelles sont-elles?

b. A la ligne 12, un appel à la fonction `test()` est effectuée. A la ligne 7, cette fonction utilise la méthode `write()` pour afficher les valeurs de deux variables. Quelles sont-elles?

c. Après exécution de la fonction `test`, un nouvel appel à `write` permet décrire les valeurs des variables a et b. Quelles seront-elles?

2. Saisissez ce code dans un fichier “.html” vide et vérifiez vos affirmations de la question précédente.

### C. La méthode `setInterval()` :

#### Exercice 13

La méthode `setInterval()` de l’objet `window` demande au navigateur d’exécuter du code ou une fonction à intervalles réguliers.

Nous allons utiliser cette fonction pour effectuer un compte à rebours.

1. a. A partir d’un fichier “.html” vierge, saisissez le code suivant :

```
1 <div style="background-color:yellow;width:100px;height↵
   :100px" id="compteur">20</div>
2
3 <script type="text/javascript">
4 var elt=document.getElementById("compteur");
5
6 function decompte(){
7   elt.innerHTML=elt.innerHTML-1;
8 }
9
10 boucle=setInterval("decompte()",500);
11 </script>
```

b. Affichez cet page dans un navigateur afin d’y exécuter ce code.

2. a. Que représente la variable `elt` après l’exécution de la ligne 4?

.....

b. Quel est le rôle du code suivant dans notre script :

```
elt.innerHTML=elt.innerHTML-1;
```

.....

c. Quel est l'intervalle d'exécution de la fonction `decompte()`?

.....



La méthode `setInterval(c,t)` lance l'exécution du code représenté par le paramètre `c` (*pouvant être une fonction à exécuter*) à un intervalle régulier de `t` millisecondes.

Cette méthode renvoie une valeur qui permet d'identifier le processus s'exécutant à intervalle régulier ; comme nous allons le voir dans la suite des questions, c'est cette valeur qui nous permettra d'interrompre l'exécution régulière ; au travers de la valeur `boucle`.

L'appel à la variable `codeelt`, à l'intérieure de la fonction `decompte()`, s'est effectué sans utilisation du mot-clef `var` ; c'est donc la variable `elt` globale qui est ciblée.

3. a. Ajoutez à votre code, dans l'élément `script`, la fonction suivante :

```
1 function arret(){
2   clearInterval(boucle);
3 }
```

b. Ajoutez à l'élément `div` de notre page l'attribut `onclick` avec pour valeur `"arret()"`.

c. Exécutez le script et remarquez l'arrêt du décompte lorsqu'on clique sur notre élément `"compteur"`.



On a stocké la valeur renvoyée par la méthode `setInterval()` dans la variable `boucle` ; cette valeur nous permet également de stopper l'exécution la répétition du code au travers de la commande `clearInterval()` qui est une méthode de l'objet `window`.

Il est possible d'imaginer qu'un autre click sur l'objet relance le compteur ; nous aurons alors besoin des structures de contrôle que vous étudierons dans le prochain chapitre.

4. a. Modifier la ligne 7 en :

```
elt.innerHTML=elt.innerHTML+1;
```

b. Exécutez le code ; que remarquez-vous?

.....

c. Donnez une explication :



La méthode `elt.innerHTML` renvoie une chaîne de caractère représentant le contenu de l'objet `elt`. Nous avons déjà vu que **JavaScript** est un langage peu typé et qu'il se charge de la conversion de type.

L'opérateur "+" permet d'additionner les nombres dans ce langage mais il permet également de concaténer les chaînes de caractères ; or, `elt.innerHTML` est un chaîne de caractères ; L'opérande gauche de + étant une chaîne de caractère, **JavaScript** transforme automatiquement l'opérande droit en chaîne de caractère expliquant le résultat obtenu.

Pour palier à ce problème, la méthode `parseInt()` permet, si cela est possible, de transformer une chaîne de caractère en un nombre entier. Voici, en quoi, on doit transformer le code précédent pour obtenir un compteur incrémentiel :

```
elt.innerHTML=parseInt(elt.innerHTML)+1;
```

L'opérateur "+" a ses deux opérands représentant des nombres ; leur somme sera calculée.

## VI. les structures de contrôles :

```
document.getElementById("tt")==null document.getElementById("tt")==undefined document.getElementById("tt")==null for(a in window)
```

Dans ce chapitre, nous allons étudier les structures de contrôles du langage **JavaScript** qui nous permettront :

- ➡ d'exécuter une partie du code mais pas une autre en fonction de condition remplie ou non ;
- ➡ de répéter plusieurs l'exécution d'une partie du code.

Nous allons passer en détails ces possibilités offertes par le langage **JavaScript**.



Presque toutes les structures de contrôles demandent des tests permettant l'exécution, la ré-exécution ou non de parties du code. De tels tests ne connaissent que deux valeurs : `true` ou `false`. Un objet ne possédant que ces deux statuts s'appellent un booléen.

Voici quelques tests les plus utiles :

- `a==b` renvoie `true` si après conversion de type, `a` et `b` représentent la même valeur.
- `a===b` renvoie `true` si les deux membres sont de même type et exactement de la

même valeur.

- `a!=b` renvoie `false` si les deux membres ont des valeurs distinctes même après conversion pour avoir le même type.
- `a<b`, valable pour des variables numériques, renvoie `true` si la valeur de `a` est inférieure à celle de `b`.

Voici un exemple un peu compliqué et peu usité de tels test ; mais qui peut permettre de déboguer efficacement votre code pour savoir si la variable `a` globale a été déclarée ou affectée :

- ➡ Si la variable `a` n'a pas été déclaré, l'exécution du code suivant provoquera une erreur et l'arrêt de **JavaScript** :

```
a===undefined
```

- ➡ Si la variable `a` n'a pas été déclaré, l'exécution du code suivant renverra la valeur `true` :

```
window.a===undefined
```

- ➡ Si la variable `a` a été déclaré mais non affectée, les deux codes suivant renvoient la valeur `true` :

```
window.a===undefined ; a===undefined
```

- ➡ Si la variable `a` a été déclaré et affectée, les deux codes suivant renvoient la valeur `false` :

```
window.a===undefined ; a===undefined
```

Attention, le test `undefined===nul` renvoie la valeur `true` : une variable déclarée et affectée de la valeur `null` n'est pas une variable non-affectée ; on voit ici la différence entre une variable non-affectée et une variable ayant la valeur `null`

## A. Les structures conditionnelles :

Elles permettent, suite à un test, d'exécuter une partie du code en fonction du résultat d'un test. Ainsi, on peut contrôler l'exécution de notre code en fonction de la valeur de certaines variables, de certains tests.



- La structure `if...then...else...` permet suivant la valeur d'un test d'exécuter une partie du code ou une autre.

Voici sa syntaxe d'utilisation :

```

1  if( condition){
2      instructions si la condition est vraie
3  }
4  else{
5      instructions si la condition est fausse
6  }

```

*condition* est généralement obtenue à partir d'un opérateur de comparaison (`==`, `!=`, `<`, `>= ...`) ; sa valeur est un booléen valant `true` ou `false`.

La partie `else{...}` est facultative.

- La structure de choix `switch(...){case :...}` permet de faciliter l'exécution d'une partie du code en fonction d'un paramètre quand celui-ci peut prendre une grande variété de valeurs :

```

1  switch( parametre){
2      case valeur1:
3          instructions si parametre est valeur1;
4          break;
5      case valeur2:
6          instructions si parametre est valeur2;
7          break;
8      default:
9          instructions si les tests ont échoués;
10 }

```

Le mot-clef `break` est obligatoire pour obliger le script de sortir d'une suite d'instructions lorsqu'un des tests a réussi.

Nous allons retrouver ces deux types de structures conditionnelles dans les exercices suivants :

### Exercice 14

1. Dans un fichier ".html" vide, recopiez le code ci-dessous :

```

1  <div style="text-align:center" id="titre" onclick="↔
    changeCouleur()">Ma présentation</div>
2
3  <script type="text/javascript">
4  var a=0;
5  var eltFond=document.getElementById("titre").style;
6

```



```
7 function changeCouleur () {
8     if (a==0) {
9         eltFond.backgroundColor="red";}
10    else {
11        eltFond.backgroundColor="yellow";}
12    }
13 </script >
```

Exécutez ce code, puis cliquez sur l'élément `div`. Que se passe-t-il? Justifiez le comportement de votre navigateur.

.....

.....

.....

2. a. Modifiez la ligne 4 du code en :

```
var a=1;
```

Puis, exécutez le code.

b. Modifiez cette même ligne en :

```
var a=2;
```

Après avoir exécuté votre code, comment justifiez-vous que votre script ait eu le même comportement :

.....

.....

.....

3. Nous allons maintenant modifier notre code pour qu'à la suite de clics successifs, notre `div` voit sa couleur de fond changée successivement.

a. Modifiez le test de la structure conditionnelle `if` en :

```
a%2==0
```

et rajoutez en fin du corps de la fonction `changeCouleur`, ligne 12, le code :

```
a++;
```

b. Exécutez le code et observez si le résultat attendu est obtenu.



Le code `a++` est équivalent à :

```
a=a+1
```

Il incrémente la valeur de `a` ; c'est à dire qu'il ajoute un à la valeur de `a`.

L'opérateur `%` permet de connaître le reste de la division euclidienne ; en particulier, lorsqu'on écrit `a%2`, on cherche à connaître le reste de la division euclidienne de la valeur de `a` par 2 : c'est à dire que la valeur de cette opération vaut successivement

0 ou 1 lorsqu'on incrémente la valeur de `a`.

Voici une remarque d'ordre plus générale sur les variables locales et globales :

- la variable `a` ne doit pas être déclarée à l'intérieur du corps de la fonction `changeCouleur()` avec le mot-clef `var`, car dans ce cas la variable `a` serait une variable local et l'instruction `a++` n'aurait pas d'influence d'un appel à l'autre de la fonction `changeCouleur()`.

4. Modifiez ce code pour obtenir trois changement successifs de couleur.

### Exercice 15

Nous allons utiliser la structure de choix `switch` pour afficher le nom du jour de la semaine en français :

1. a. Saisissez le code suivant dans une page `“.html”` vide.

```
1 <script type="text/javascript">
2   var hoy=new Date();
3
4  switch(hoy.getDay()){
5     case 0: x="Dimanche"; break;
6     case 1: x="Lundi"; break;
7     case 2: x="Mardi"; break;
8     case 3: x="Mercredi"; break;
9     case 4: x="Jeudi"; break;
10    case 5: x="Vendredi"; break;
11    case 6: x="Samedi"; break;
12  }
13
14  document.write(x);
15 </script >
```

- b. Exécutez et vérifiez le fonctionnement du script.
2. Les méthodes `hoy.getFullYear()`, `hoy.getDate()` et `hoy.getMonth()` renvoie respectivement, l'année, le jour du mois, et le numéro du mois (*de 0 à 12*).

Modifiez le script afin qu'il affiche la date actuelle sous la forme :

*“Vendredi 16 Avril 2010”*

## B. les structures répétitives :

Les structures répétitives permettent à **JavaScript** de répéter, sous certaines contraintes, plusieurs fois la même partie d'un code ; cette répétition peut être contrôlée de plusieurs

manières :

- on peut facilement exécuter un nombre de fois déterminé à l'avance une partie de code ;
- on peut exécuter une partie de code tant que certaines conditions de sortie ne sont pas atteintes ; on retrouve ce type de boucle dans les cas suivants :
  - ⇒ L'algorithme d'Euclide pour déterminer le PGCD de deux nombres entiers.
  - ⇒ Des algorithmes de tri pour ordonner les éléments d'un tableau.



- La boucle “pour ... jusqu'à ...” :

```
1 for(initialisation; test de fin; incrementation) {  
2   instructions à répéter...  
3 }
```

Cette boucle est utilisée pour répéter un nombre de fois déterminé les instructions contenues dans le corps de sa déclaration ; une variable est utilisée pour contrôler le nombre d'itérations à effectuer. Prenons l'exemple suivant :

```
for(i=0; i<5; i++){...}
```

- ⇒ La variable *i* contrôlant l'exécution du code est initialisée avec la valeur 0 ;
- ⇒ A chaque itération, la valeur de la variable est incrémentée comme nous l'indique le code :

```
i++;
```
- ⇒ le corps de cette boucle s'exécutera tant que la valeur de *i* restera strictement inférieure à 5 ; autrement dit, le corps de cette structure s'exécutera 5 fois (*de 0 à 4, il y a 5 nombres*) .

- La boucle “tant que” :

```
1 while(condition) {  
2   instructions à répéter...  
3 }
```

Cette boucle teste la condition et tant que celle-ci reste vraie, il exécute le bloc d'instructions concerné ; dans ce type d'instruction, les instructions modifient les paramètres (*variables et autres*) rentrant en compte dans la condition jusqu'à obtenir le résultat escompté (*attention aux boucles sans fin*)

- La boucle “jusqu'à que” :

```
1 do {  
2   instructions à répéter...  
3 } while(condition)
```

Cette boucle, à la différence de la structure précédente, commence à exécuter les instructions puis si la condition a toujours la valeur `true` alors il continue à exécuter le bloc d'instructions.

## Exercice 16

L'exercice suivant va nous permettre de créer un tableau dont la longueur va dépendre d'une variable ; cette variable est de type tableau et une structure répétitive s'adaptera facilement à la longueur de cette variable tableau.

1. Sans l'exécuter, saisissez le code suivant dans un fichier `“.html”` vide :

```
1 <script type="text/javascript">
2 var bouton=new Array("Math","Info","Contact");
3
4 document.write("<table width=100% border=1><tr>");
5 for(var i=0;i<bouton.length;i++){
6     document.write("<td width=5%><td width=20%>" + bouton[i] +
7         ");
8 }
9 document.write("<td width=5%>");
10 </script>
```

2. a. A partir de la ligne 3, donnez la valeur de la propriété `bouton.length` :

.....

- b. A la vue du code, donnez toutes les valeurs prises par la variable `i` lors de l'exécution de ce code.

.....

- c. Combien de fois sera exécuté le corps de cette boucle?

.....

3. Affichez cette page dans un navigateur.

Nous allons maintenant modifier notre script afin de lui rajouter quelques fonctionnalités supplémentaires.

4. Faites en sorte que la page affiche un quatrième bouton nommé *“Liens”*.

5. Ajoutez des gestionnaires d'événements `onmouseover` et `onmouseout` aux éléments `td` (représentant les boutons) afin qu'ils modifient leurs apparences (surtout leurs couleurs de fond) lors du passage de la souris indiquant ainsi une zone cliquable au client ; n'oubliez pas d'utiliser le mot-clef `this`.

6. Nous allons rendre ces boutons cliquables à l'aide du gestionnaire d'événement *onclick* :

a. Rajoutez la variable suivante :

```
var direct=new Array("math.html","./info/","contact.php",  
    "./liens/nouveau.html");
```

Elle indique les URL relatives suivies respectivement par chacun des boutons

b. Créez une fonction possédant un paramètre formel nommé *x* et dont le corps de la fonction est :

```
document.location.href=x;
```

c. Liez chaque élément `td`, représentant un bouton, à cette fonction à l'aide du gestionnaire d'événement *onclick* ; lors de son appel la fonction doit prendre comme argument `direct[i]`.

### Exercice 17

Nous allons nous intéresser à l'algorithme d'Euclide dont la représentation est donnée (Figure 4)

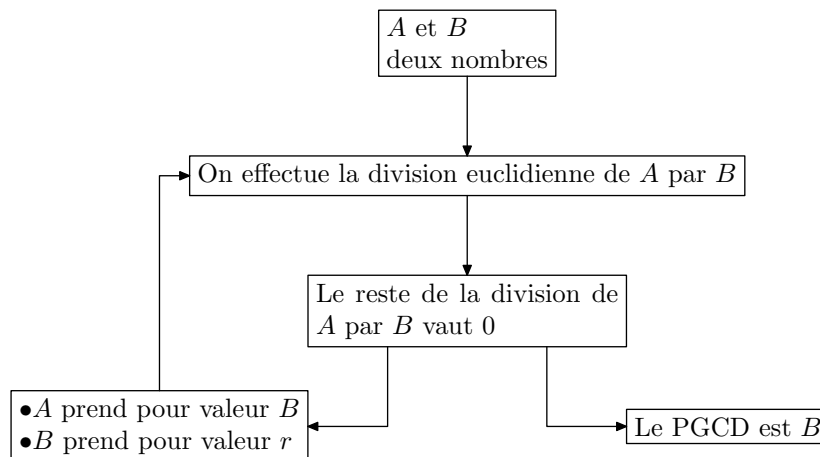


Figure 4: “représentation de l’algorithme d’Euclide”

Cet algorithme permet de calculer le PGCD de deux nombres permettant de donner la forme irréductible d’une fraction. Voici quelques indications qui vous permettront de construire ce programme :

- Le client est interrogé à l’aide de la méthode `prompt()` pour indiquer la valeur du numérateur et du dénominateur de la fraction.
- Rappelez-vous que l’opérateur “`\%`” permet d’obtenir le reste de deux entiers par la division euclidienne.
- On stockera le reste dans la variable `r` ; puis le code est exécuté en boucle à l’aide de la structure répétitive `while(...){...}` tant que le reste `r` ne vaut pas 0 (`r!=0`).
- On affichera la fraction sous forme irréductible, en utilisant le PGCD obtenue, à l’aide de l’une des deux méthodes `alert()` ou `document.write()`.

## VII. Les événements (partie 2) :

Le **JavaScript** a connu une longue évolution ; à travers son évolution, le HTML a connu également des changements, non pas le langage lui-même mais surtout par la manière dont le HTML est appréhendé par un navigateur ; pour cela reportez-vous au paragraphe sur le DOM dans le troisième manuel de cours page 23.

Ainsi, il existe trois manières de définir des écouteurs d'événement :

- Nous avons déjà vu comment intégrer les écouteurs d'événements dans la balise ouvrante des événements ;
- Dans ce paragraphe, nous allons voir que les éléments HTML vus par **JavaScript** possèdent les propriétés adéquates pour définir les écouteurs d'événements ;
- La troisième façon de faire est d'utiliser le modèle DOM de gestion, par les navigateurs, des objets d'une page Web ; cette technique est très compliquée à mettre en oeuvre, mais c'est la seule permettant notamment de permettre l'exécution de fonctions distinctes pour le même écouteur pour un même élément ; cette technique est essentielle lors de la programmation AJAX.

Nous allons, dans ce chapitre de la formation, étudier la seconde façon d'implémenter les écouteurs d'événement dans nos pages Web. Au cours de ce chapitre, nous allons aussi retrouver les vestiges de la guerre des standards qui a opposé Internet Explorer à Netscape pendant longtemps.



La programmation AJAX permet à des pages de recharger le contenu d'éléments HTML de sa page sans pour autant effectuer un rechargement complet de celle-ci ; ceci permet d'obtenir de la fluidité dans la navigation du client : “*yahoo mail*” en est un exemple.

Cette technique de programmation mélange le **JavaScript** et le PHP.

### A. Déclaration d'écouteurs :

#### Exercice 18

1. Saisissez le code ci-dessous dans un fichier “.html” vide :

```
1 <form action="traite.php">
2 Nom : <input type="text" name=nom><br>
3 <input type=submit value="Envoyer">
4 </form>
```

```

5
6 <script >
7 function change(){
8     alert("Vous avez changé la valeur du champ");
9 }
10
11 function touche(){
12     document.title+="e";
13 }
14
15 function valide(){
16     return false;
17 }
18 var champ=document.forms[0].elements["nom"];
19
20 champ.onchange=change;
21 champ.onkeypress=touche;
22 </script >

```

2. Après avoir affiché votre page dans un navigateur et avoir testé le code, répondez aux questions suivantes :

a. Que représente la variable `champ`?

.....

b. A quel moment la fonction `touche()` est-elle appelée? Quel est son effet?

.....

.....

c. A quel moment la fonction `change()` est-elle appelée? Quel est son effet?

.....

.....



Presque tous les éléments HTML possèdent les propriétés `onmouseover`, `onmouseout` donnant un accès direct à **JavaScript** aux écouteurs permettant ainsi de modifier dynamiquement ces écouteurs.

Pour utiliser ces propriétés prennent pour valeur le nom de la fonction qui sera activé lors de l'arrivée de l'événement.

Les éléments HTML de type "*champ de texte*" possède en plus les propriétés `onkeypress` et `onchange` correspondant respectivement aux événements suivants :

➡ "*une touche a été pressée lorsque l'élément avait le focus*"

⇒ “lorsque l’élément a perdu le focus, sa valeur a changé”

Voici quelques remarques supplémentaires pour mieux comprendre le script précédent :

- L’opérateur “+=” est un l’opérateur d’incrémentaion lorsqu’on a affaire à des valeurs numériques ; pour les chaînes de caractères, il facilite la concaténation des chaînes de caractères :

`a+="bonjour"` est équivalent à `a=a+"bonjour"` ;

- L’objet `form` est un tableau permettant d’accéder à tous les formulaires d’une page Web ; le script ci-dessus ne contient qu’un seul formulaire, on y accède via le code :

`document.forms[0]`

Chaque formulaire possède l’objet `elements`, également un tableau, qui permet d’accéder à tous les contrôles du formulaire ; on accède à un élément par son nom :

`document.forms[0].elements["nom"]`

3. Lorsque le client clique sur le bouton de soumission, les données sont transmises à la page `traite.php` pour traitement. Nous allons voir comment intercepter cette événement ; cela nous permet de vérifier l’intégrité des données saisies par le client avant l’envoi des données au serveur.

- a. Rajoutez la ligne de code suivant en fin du script :

`document.forms[0].onsubmit=valide;`

Quel est l’effet de cette ligne?

.....  
.....



L’écouteur `onsubmit` agit sur les éléments `form` ; cet événement intervient lors de la soumission du formulaire. Si la fonction appelée par l’écouteur renvoie la valeur `false`, alors le formulaire n’est pas envoyé au serveur. Seul la valeur `true` permet au formulaire d’être soumis.

- b. Modifiez la fonction `valide()` afin qu’elle accepte la soumission du formulaire seulement le champ texte à la valeur “*thomas*”.

## B. L’objet event :

Nous allons voir dans ce paragraphe qu’à chaque événement, un certain nombre d’informations sont transmises sur cet événement :

- on peut connaître la position du curseur lors d’un clic ;
- on peut connaître la touche du clavier qui a été pressée...

Ce paragraphe sera aussi l’occasion d’observer une grande différence entre le **JavaScript** de



Firefox (*le descendant de Netscape*) et celui d'Internet Explorer.

Pour commencer, nous allons étudier cet objet **sous Firefox uniquement** :

### Exercice 19

1. Récupérez le code précédent et effectuez les modifications suivantes :

- a. Avant l'élément `script`, rajoutez un élément `div` vide possédant l'attribut `id` ayant la valeur `"touch"`.
- b. Modifiez la fonction `touche` afin qu'elle accepte un paramètre formel nommé `e` ; le corps de cette fonction doit être :

```
1 touch=document.getElementById("touch");
2 touch.innerHTML=e.which+"<br>";
3 touch.innerHTML+=String.fromCharCode(e.which);
```

2. Testez le code dans Firefox et saisissez quelques lettres dans le champ texte. Que remarquez-vous?

.....  
.....  
.....



Lorsqu'un élément se déclenche et que l'exécution d'une fonction est lancée, Firefox transmet à la fonction en argument un objet **JavaScript** contenant des informations sur l'événement ; cet objet est de type **Event**.

Dans le cas de l'exemple ci-dessus, le paramètre formel `e` recevant cet objet contient le nombre repérant la touche sur le clavier ; un événement de souris transmettra par cette variable la position du curseur dans la page.

Plus précisément :

- ➔ La propriété `e.which` contient le code repérant la touche enfoncée.
- ➔ La méthode `String.fromCharCode()` permet de retrouver le caractère correspondant

### Exercice 20

Maintenant, la différence entre un événement créé par Internet Explorer ou par Firefox :

1. a. Remplacez le code de la fonction `touche` par :

```
1 function touche(e){
2     if(window.event===undefined){
```

```

3     document.title="Je suis qui?";
4 }
5 else{
6     document.title="A vous de trouver";
7 }
8 }

```

b. Affichez votre page dans Internet Explorer, puis dans Firefox pour observer la différence d'exécution de ce script.

c. Que pouvez-vous dire de l'objet `window.event`?

.....

.....

.....

2. a. Remplacez le code de la fonction `touche()` par :

```

1 function touche(e){
2     if(window.event===undefined)
3         keyNum=e.which;
4     else
5         keyNum=window.event.keyCode;
6     var touch=document.getElementById("touch");
7     touch.innerHTML=keyNum+"<br>";
8     touch.innerHTML+=String.fromCharCode(keyNum);
9 }

```

b. Exécuter ce code ; avez-vous remarqué des différences d'exécution?

.....



Alors que Firefox passe en argument à la fonction liée à un événement un objet de type `Event`, Internet Explorer le crée comme propriété de l'objet `window` ; ainsi, en testant l'existence de l'objet `window.event` lors du déclenchement d'un événement, on arrive à savoir si le navigateur utilisé est Firefox ou Internet Explorer.

Le code de la touche est contenu :

⇒ dans la propriété `which` pour Firefox ;

⇒ dans la propriété `keyCode` pour Internet Explorer.

On utilise la variable locale `keyNum` pour rétenir, dans les deux cas, le numéro de la touche afin que le reste de l'exécution du code soit commune aux deux navigateurs.

## VIII. Exemples commentés :

### A. Introduction :

Après dans le premier TD, avoir recopié du code et l'expliquer de bout en bout, ici ne vous sera représenté que des codes et leurs effets ... A vous de rechercher le code qui vous intéresse et comprendre tout son mécanisme.

Plus vous maîtriserez de concept et plus grande sera votre imagination... Ne vous contentez pas uniquement de recopier bêtement le code ... Analyser les codes qui vous plaisent.

### B. De simples boutons :

Nous allons utiliser **JavaScript**, pour créer des boutons :

- Le bouton doit changer de couleurs lorsque le curseur le survole.
- Il doit posséder une autre couleur lorsque le bouton est en position enfoncé.
- Au moment de relâcher, celui doit rediriger la page vers le fichier destination.

```
1 <head>
2 <style type='text/css '>
3   .bouton{color:blue;width:100px;background-color:#AAAAAA}
4 </style>
5 <script type='text/javascript '>
6   function passe(x){
7     x.style.backgroundColor="#EEEEEE";
8   }
9
10  function sort(x){
11    x.style.backgroundColor="#AAAAAA";
12  }
13
14  function appui(x){
15    x.style.backgroundColor="#AAFFEE";
16  }
17
18  function bouge(x,y){
19    x.style.backgroundColor="#AAAAAA";
20    document.location.href=y;
21  }
22 </script >
```

```

23 </head>
24 <body>
25 <table style='width:100%'>
26   <tr>
27     <td class=esp>&nbsp;
28     <td onmouseover='passe(this)' onmouseout='sort(this)' ←
        onmousedown='appui(this)' onmouseup='bouge(this,"←
        http://google.fr")' class=bouton>Google
29   <td class=esp>&nbsp;
30   <td onmouseover='passe(this)' onmouseout='sort(this)' ←
        onmousedown='appui(this)' onmouseup='bouge(this,"←
        http://yahoo.fr")' class=bouton>Yahoo
31   <td class=esp>&nbsp;
32   <td onmouseover='passe(this)' onmouseout='sort(this)' ←
        onmousedown='appui(this)' onmouseup='bouge(this,"←
        http://msn.fr")' class=bouton>MSN
33   <td class=esp>&nbsp;
34 </table>
35 </body>

```

Pour les utilisateurs avancés, voici comment on peut simplifier le code :

```

1 <head>
2 <style type='text/css '>
3   .bouton{color:blue;width:100px;background-color:#AAAAAA}
4 </style>
5 </head>
6 <body>
7 <table style='width:100%'>
8   <tr>
9     <td class=esp>&nbsp;
10    <td href="http://google.fr" class=bouton>Google
11    <td class=esp>&nbsp;
12    <td href="http://yahoo.fr" class=bouton>Yahoo
13    <td class=esp>&nbsp;
14    <td href="http://msn.fr" class=bouton>MSN
15    <td class=esp>&nbsp;
16 </table>
17
18 <script type='text/javascript '>

```

```
19 a=document.getElementsByTagName("td")
20 for(i=0;i<a.length;i++){
21     if(a[i].getAttribute("class")!="bouton")
22         continue;
23
24     a[i].onmouseover= function(){
25         this.style.backgroundColor="#EEEEEE";
26     }
27
28     a[i].onmouseout= function(){
29         this.style.backgroundColor="#AAAAAA";
30     }
31
32     a[i].onmousedown= function(){
33         this.style.backgroundColor="#AAFFEE";
34     }
35
36     a[i].onmouseup= function(){
37         this.style.backgroundColor="#AAAAAA";
38         document.location.href=this.getAttribute("href")
39     }
40 }
41 </script >
42 </body >
```