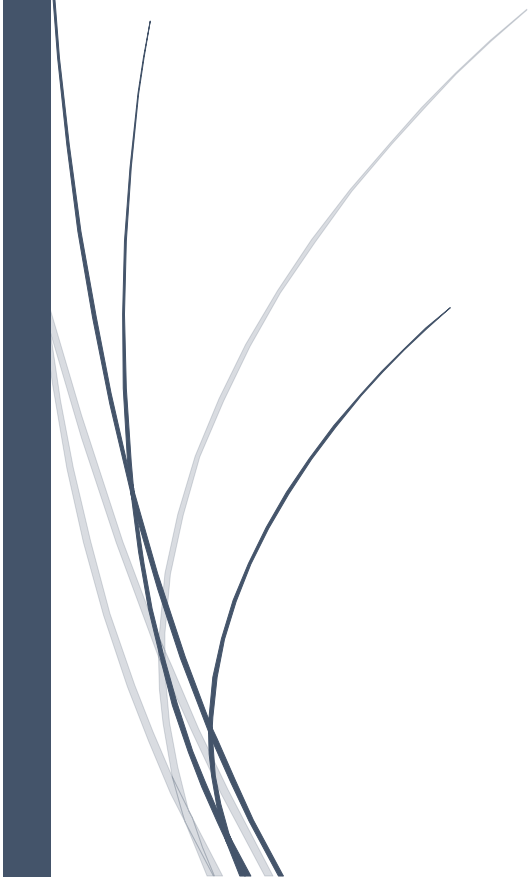
16/09/2020

JAVASCRIPT

Exercices de mise en application

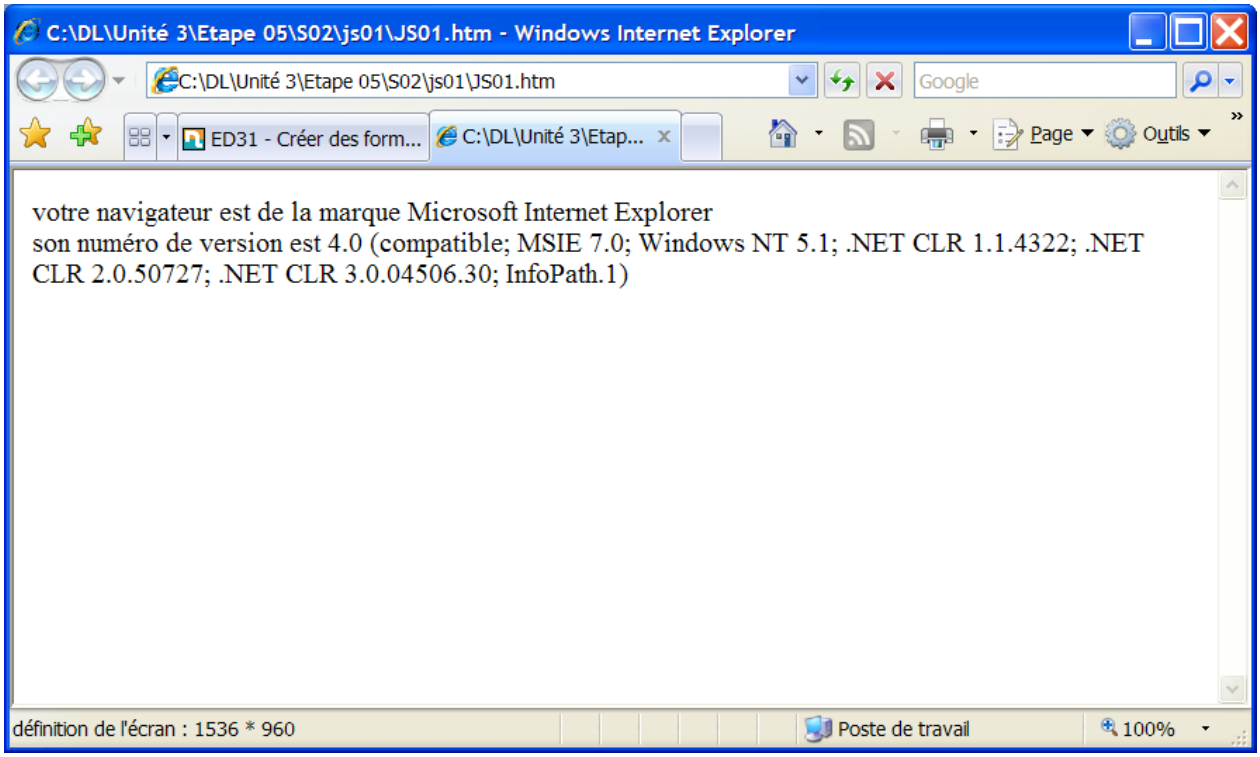


Rudy Lesur

Table des matières

I AFFICHAGE DU NAVIGATEUR

Affichez dans une page HTML, le nom de votre explorateur, son numéro de version, ainsi que les caractéristiques de définition de l'écran dans la zone status, comme suit :



Points abordés :

La méthode write() de l'objet document.
L'objet navigator et ses propriétés (appName, appVersion….)
L'objet window et sa propriété status.
L'objet screen et ses propriétés (width, height).

I.1 UN PEU DE THEORIE

Window est l'objet JavaScript de plus haut niveau dans la hiérarchie d'une page HTML. C'est lui qui possède le plus de propriétés, de sous-objets et de méthodes. Il est le seul à autoriser une écriture simplifiée de ses propriétés et méthodes. Par exemple JS identifie le sous-objet *document* de *window* aussi bien avec l'écriture complète *window.document* qu'avec *document*. Cette particularité a un avantage sur la rapidité d'écriture.

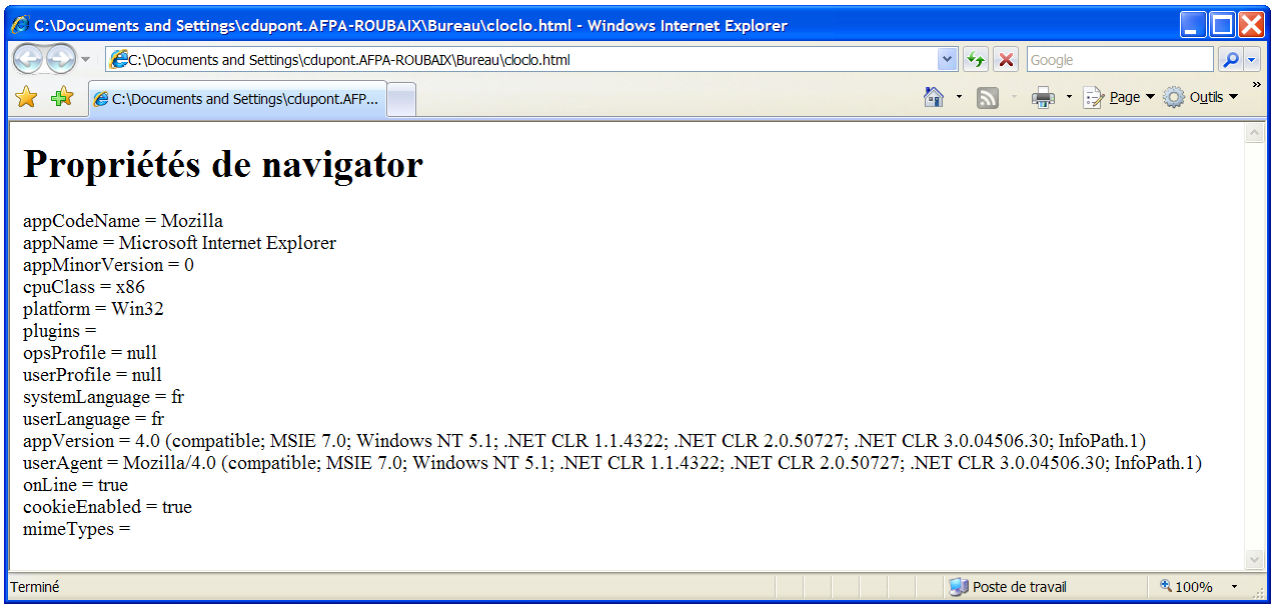
L'objet navigator

Il contient de nombreuses informations sur le navigateur en cours d'utilisation.
Il existe des différences de comportement entre navigateurs, il est donc difficile de repérer simplement l'application utilisée par le navigateur.

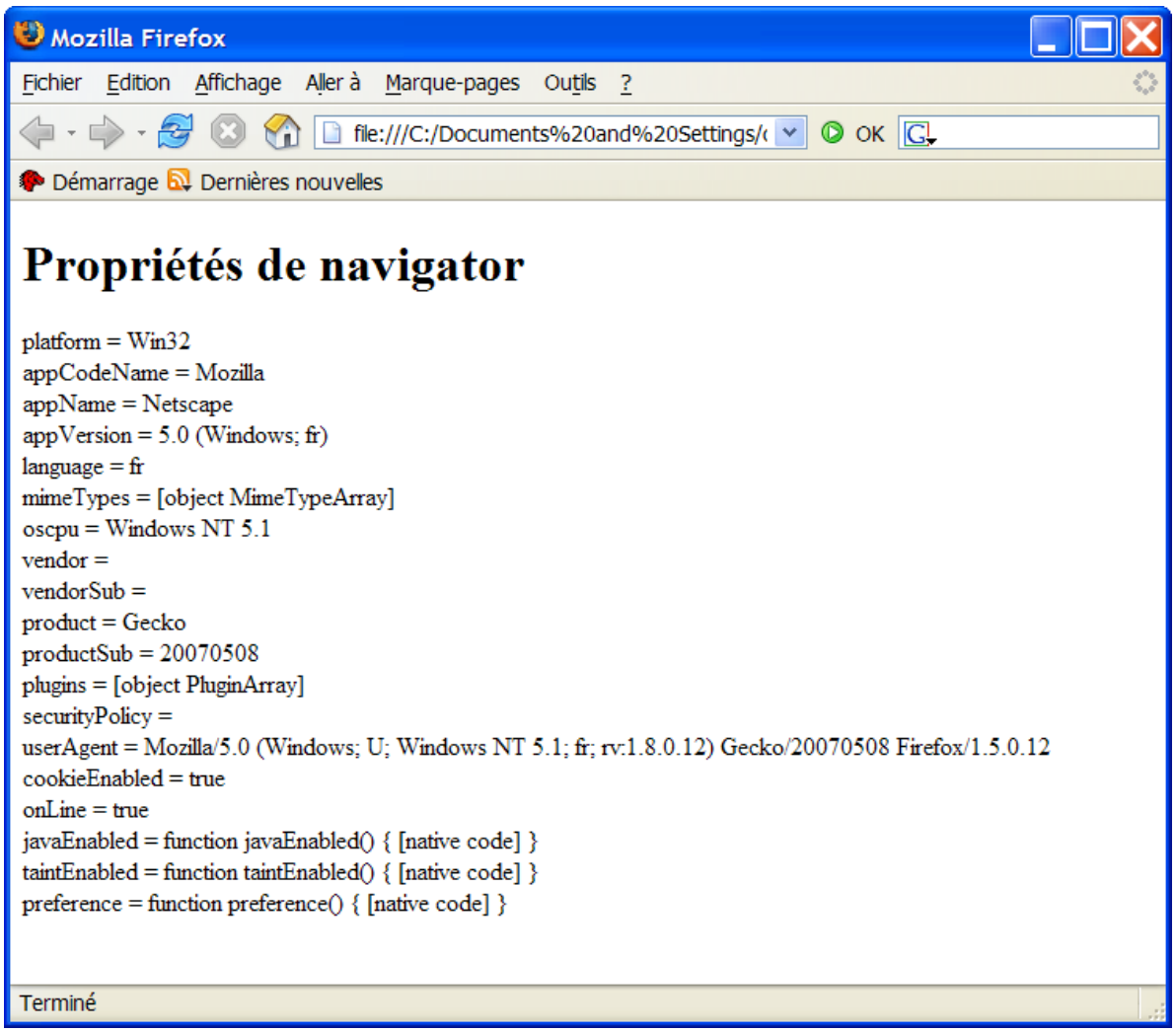
A l'aide de la boucle *for…in*, nous pouvons extraire toutes les propriétés de cet objet et leurs valeurs dans 2 navigateurs principaux :

```
<body>
  <p><h1> Propriétés de navigator</h1></p>
<script type="text/javascript">
  for (i in window.navigator) {
        document.write(i+" = "+window.navigator[i]+"<br/>");
        }
  </script>
  </body>
```

Internet Explorer :



Firefox :



7 propriétés sont compatibles.
userAgent contient le plus d'informations. C'est cette propriété qui nous permettra au mieux d'identifier un navigateur.
Le nom de code de l'application donné par ***appCodeName*** est identique pour les 2 navigateurs !
La version du navigateur donnée par ***appVersion*** n'est pas sous la forme d'un nombre mais sous une chaine de caractères complexe.
La propriété ***language*** n'est pas reconnue par IE qui utilise 2 propriétés ***systemLanguage*** et ***userLanguage*** pour la remplacer. Il est possible de gérer un site multilingue avec ces propriétés et d'afficher directement la bonne page au visiteur en fonction de la langue de son navigateur.
platform indique le type de système d'exploitation. Elle peut être utile pour proposer des téléchargements adaptés.
cookieEnabled vaut true si les cookies sont activés.

L'objet screen

Le sous-objet *screen* lié à *window* possède des propriétés sur la résolution de l'écran du visiteur.

Propriétés :

screen.height et screen.width donnent le nombre de pixels affichés à l'écran en hauteur et en largeur.

screen.availWidth et screen.availHeight donnent le nombre de pixels disponibles pour les applications.

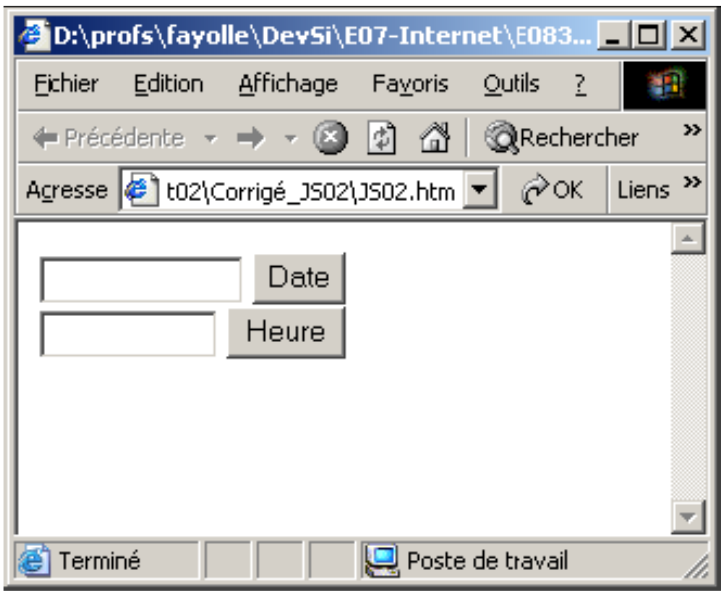
screen.colorDepth qui indique la profondeur de couleurs en bits (nombre de couleurs distinctes affichables)

math.pow(2,screen.colorDepth) donne le nombre de couleurs.

Cet objet permet donc de connaître la résolution de l'internaute. Grâce à ces informations, nous pouvons déterminer l'équipement de nos visiteurs et optimiser les informations à l'écran pour limiter au maximum les ascenseurs horizontaux.

II MANIPULATION DE LA DATE

Affichez la page HTML suivante.



Lorsque l'on clique sur le bouton date, la date système s'affiche dans le champ correspondant.
Lorsque l'on clique sur le bouton heure, l'heure système s'affiche dans le champ correspondant.

Points abordés :

L'objet Date et ses méthodes : getDate(), getMonth(), getHours()…….
La création de fonctions JavaScript.
Gestion d'événements : onClick.

II.1 UN PEU DE THEORIE

La programmation événementielle

JavaScript est un langage événementiel, c'est-à-dire que son exécution peut être déclenchée par des événements qui surviennent sur la page. Ils consistent en un changement de l'état de la page. Il en existe plusieurs.
Parmi les plus courants nous trouvons :

- Le déplacement de la souris
- Un clic, sur un bouton ou un lien
- Le chargement d'une page
- Le survol de la souris au-dessus d'un élément HTML
- La modification d'une zone de saisie
- Un appui sur une touche du clavier

La plupart des événements sont déclenchés par une action de l'utilisateur à partir des périphériques de saisie connectés à son ordinateur (clavier, souris).
La programmation événementielle est idéale pour la gestion de l'interactivité mais elle impose des contraintes importantes au développeur. Le visiteur est complètement libre dans sa navigation et rien ne l'oblige à suivre l'ordre de saisie prévu. Il faut donc s'assurer que la fonctionnalité d'une page est bien remplie : le visiteur a-t-il bien respecté l'ordre logique, a-t-il oublié des étapes ou inversé des étapes ?

Pour indiquer au navigateur l'action à réaliser en fonction de l'événement, il faut utiliser les attributs de type *onEvenement* des balises HTML.
Par exemple, pour détecter un clic sur un lien :

```
<a href="#" onclick="alert('Bonjour') ">Cliquez sur le lien</a>
```

L'attribut *onclick* contient le code JavaScript à exécuter. Il est possible de définir plusieurs instructions pour le même événement en utilisant le séparateur d'instructions JavaScript à savoir le point-virgule.
Pour des raisons de lisibilité, il est préférable de regrouper les instructions au sein d'une fonction. Cela permettra de récupérer la fonction pour un autre événement et d'éviter des erreurs dans l'alternance des guillemets et des simples quotes.

```
<a href="#" onclick="message('Bonjour') ">Cliquez sur le lien</a>

<script type=''text/javascript''>
        function message(texte) {
                alert(texte) ;
        }
</script>
```

Les différents événements :

- onAbort : quand l'utilisateur appuie sur Echap
- onBlur : quand un objet perd le focus
- onChange : quand le curseur quitte la change et que son contenu a été modifié
- onClick
- onContextMenu : avant l'apparition d'un menu contextuel avec clic droit
- onDblClick
- onDragDrop : détecte un glisser-déposer
- onError : permet de capturer les erreurs JavaScript qui surviennent
- onFocus : survient quand un élément reçoit le focus
- onKeyDown : se produit quand l'utilisateur appuie sur une touche du clavier
- onKeyPress : se déclenche quand l'utilisateur relâche une touche du clavier sur laquelle il a appuyé
- onKeyUp : se déclenche au moment où la touche est relâchée
- onLoad : est déclenché quand une page est entièrement chargée
- onMouseDown : déclenché au moment où le visiteur appuie sur l'un des boutons de la souris (gauche, droit ou les deux en même temps)
- onMouseMove : déclenché en permanence pendant les mouvements de la souris
- onMouseOut : déclenché quand le curseur de la souris quitte un objet
- onMouseOver : déclenché quand le curseur de la souris quitte un objet
- onMouseUp : déclenché au moment où un bouton de la souris est relâché
- onReset : avec la balise <form>. Ce déclenche quand le formulaire est réinitialisé
- onSelect : quand l'utilisateur sélectionne une zone de texte
- onSubmit : avec la balise <form>. Déclenché à la soumission du formulaire
- onUnload : quand la page courante est remplacée par une autre ou à la fermeture du navigateur.

Les fonctions

Les fonctions sont un moyen de coder une seule fois une fonctionnalité et de réutiliser ce code en cas de besoin. Les fonctions sont à la base du développement durable. L'idée est de se constituer un ensemble de fonctions et de les réutiliser dans chaque contexte où elles sont nécessaires.
JS nous permet d'écrire nos propres fonctions personnalisées.
La syntaxe est la suivante :

```
function nomDeLaFonction(param1, param2, …, paramN) {
    // instructions
}
```

Une fonction attend des paramètres. L'ensemble du code source composant le groupe d'instructions est appelé le corps de la fonction.

Pour exécuter une fonction dans un script, il suffit d'écrire son nom suivi, entre parenthèses, des différents paramètres nécessaires à son fonctionnement. Une fois définie, une fonction peut être appelée autant de fois que nécessaire.
Une fonction peut ne pas avoir de paramètre. Même dans ce cas, les parenthèses sont obligatoires, dans la définition de la fonction et dans son appel.

Une fonction ne retourne pas toujours de valeur. L'instruction *return* n'est pas obligatoire.

Conseil : il est toujours préférable de définir une fonction pour regrouper le code source présent à plusieurs endroits. La maintenance en est facilitée : vous ne modifiez qu'une seule fois le code. De plus, vous pourrez réutiliser votre fonction dans d'autres pages ou dans d'autres projets.

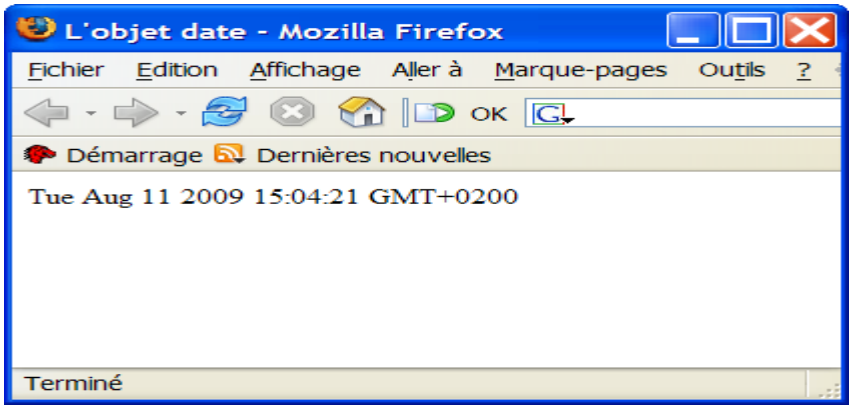
L'objet DATE

L'objet *Date* est une structure de données qui contient à la fois une date et une heure précises.
L'année est exprimée sur chiffres.
Le mois est exprimé par un nombre compris entre 0 et 11. L'indice 0 correspond à janvier et 11 à décembre.
Le jour du mois varie de 1 à 31.
Le jour de la semaine est exprimé par un nombre compris entre 0 et 6. L'indice 0 correspond au dimanche et 6 au samedi.
L'heure varie de 0 à 23, les minutes et les secondes de 0 à 59.
Les millisecondes varient de 0 à 999.
Le fuseau horaire correspond au fuseau horaire défini dans les paramètres de l'horloge du poste client.

La création d'un objet *Date* est réalisée avec le constructeur *Date()* :
var dtJour = new Date() ;
Après exécution, *dtJour* est un objet *Date* contenant la date et l'heure de l'instant de création.
Le code suivant :

```
<body>
    <script type="text/javascript">
        var dtJour = new Date();
        document.write(dtJour);
    </script>
</body>
```

Affichera comme résultat :



Ce format d'affichage par défaut n'est pas très compréhensible.

L'objet *Date* ne possède pas de propriété. L'accès aux diverses informations se fait par une série de méthodes. Toutes ses méthodes portent le préfixe *get* dans leur nom pour montrer qu'elles retournent un résultat.

getDate()	Retourne le numéro du jour du mois (entre 1 et 31)
getDay()	Retourne le numéro du jour de la semaine (entre 0 et 6)
getFullYear()	Retourne l'année complète sur 4 chiffres. Remplace getYear() non prévu pour le passage à l'an 2000.
getHours()	Retourne l'heure de l'objet (entre 0 et 23)
getMilliseconds()	Retourne le nombre de millisecondes de l'objet (de 0 à 999)
getMinutes()	Retourne les minutes (de 0 à 59)
getMonth()	Retourne l'indice du mois (de 0 à 11)
getSeconds()	Retourne les secondes (de 0 à 59)
getTime()	Retourne le nombre de millisecondes qui séparent la date sur laquelle s'applique la méthode et le 1er janvier 1970. Utile pour déterminer l'écart entre deux dates.
getTimezoneOffset()	Retourne le nombre de minutes séparant l'heure locale et le fuseau horaire de référence.
getYear()	Retourne l'année mais il est conseillé d'utiliser getFullYear()

III MANIPULATION D'IMAGES

Affichez la page HTML suivante.

Lorsque l'on clique sur l'ampoule, l'image change (l'ampoule s'allume si elle est éteinte, s'éteint si elle est allumée).
Les images sont disponibles sous forme de fichiers GIF.

Points abordés :

Gestion des événements : onClick
Ecriture d'une fonction
Utilisation de variables
Tests conditionnels : if

III.1 UN PEU DE THEORIE *LES VARIABLES*

Tous les langages de programmation s'appuient sur des variables pour stocker et manipuler des informations.
Une variable est une zone mémoire contenant une donnée. Les variables ont des noms nous permettant de les manipuler.
En JS, une variable est définie avec le mot clé *var*.
Nous pouvons déclaration une variable avec ou sans initialisation.

Ex : var compteur = 0 ;
var text1 ;
var text2, text3 = ''bonjour'' ;

JS n'est pas très strict : nous pouvons utiliser une variable sans qu'elle ait été déclarée. Cela s'appelle une déclaration implicite.

Le nom d'une variable peut être composé des 26 lettres de l'alphabet en majuscules et en minuscules, des 10 chiffres et du caractère underscore _ (touche 8). Le caractère d'espacement est interdit et un nom ne peut pas commencer par un chiffre.

JS distingue la casse c'est-à-dire les majuscules et les minuscules.

JS est pauvrement typé. Une variable peut contenir indifféremment tous les types de données au sein du même script.
JS possède une instruction qui nous permet de connaître le type de la donnée contenue dans une variable : *typeof*.
Le fait de déclarer une variable sans lui affecter de valeur initiale rend la variable indéfinie. Elle contient la valeur *undefined* et a le type *undefined*.

Le type **number** : concerne toutes les variables contenant un nombre, entier, à virgule, positif ou négatif, petit ou grand.
Le type **string** : concerne toutes les variables contenant une chaine de cractères. Une chaine de caractères est délimitée par les guillemets (‘’)
Le type **boolean** : peut contenir 2 valeurs : « vrai » ou « faux » soit *true* ou *false*.

Les tests conditionnels

Un programme est une succession d'instructions exécutées les unes à la suite des autres. Les instructions conditionnelles orientent l'exécution vers une partie du programme selon le résultat des tests.

Exemple :

```
if (prixTotal > 50)
        {
                prixTransport = 10 ;
        }
else
        {
                prixTransport = 20 ;
        }
```

L'expression du test s'écrit obligatoirement entre parenthèses.
Le ELSE n'est pas obligatoire.

Les opérateurs de tests :

==	égal
!=	différent
>	strictement supérieur
<	strictement inférieur
>=	supérieur ou égal
<=	inférieur ou égal

Les combinaisons de tests

il est possible de combiner des tests ensemble pour créer des conditions plus complexes. Le test logique ET est symbolisé par &&. Le test logique OU est symbolisé par ||.

Les tests multiples

Pour comparer une variable à plusieurs valeurs, nous pouvons utiliser l'instruction **switch**.

Exemple :

```
var nbTour = 2 ;
switch (nbTour) {
        case (1) :    document.write(« Premier tour ») ;
                      break ;
        case (2) :    document.write(« deuxième tour ») ;
                      break ;
        default :     document.write("je ne sais pas");
}
```

nbTour est comparée aux différentes valeurs. Chaque comparaison de valeur est définie par un *case* suivi de la valeur à comparer entre parenthèses. Si le test d'égalité entre cette valeur et la variable à comparer du switch est vérifié, le groupe d'instructions suivant

les : est exécuté. L'instruction **break** marque la fin de ce groupe et arrête toutes les comparaisons définies dans le switch. L'exécution du script reprend immédiatement après l'accolade fermante de l'instruction switch.
Si aucun des tests n'est vérifié, les instructions qui suivent **default** sont exécutées. *default* n'est pas obligatoire. S'il n'est pas présent et si aucun des tests d'égalité n'est vérifié, aucune instruction du groupe *switch* n'est exécutée.
Il est impossible de donner plus d'une valeur de comparaison à chaque *case.*

L'opérateur ternaire

Il existe pour les tests d'égalité une écriture rapide en une seule ligne qui retourne une valeur si le test est vérifié et une autre valeur sinon.

Syntaxe :

test ? valeur si test vrai : valeur si test faux

exemple :

```
var txtAccueil = heure < 18 ? ''Bonjour'' : ''Bonsoir'' ;
```

cet opérateur permet d'affecter une valeur en évitant l'écriture d'un test conditionnel *if else*.

IV LES CONTROLES DE SAISIE

IV.1 CREATION D'UNE EXPRESSION REGULIERE

Les expressions régulières permettent de définir, en quelque sorte, un modèle auquel on pourra comparer les champs que l'on veut contrôler.
Elles sont utilisées principalement pour effectuer des contrôles de validation (dans notre cas), et pour effectuer des recherches et remplacement des textes ou partie de texte.
Associées à JavaScript, elles simplifient considérablement la programmation des contrôles coté client, et peuvent être utilisées notamment pour valider les formats des données saisies (N° téléphone, N° carte bleue, e-mails, etc.…).
Les expressions régulières sont créées de la manière suivante (manière la plus simple, sachant qu'il existe d'autres méthodes et d'autres paramètres) :

```
var nomExpression = /chaine/;
```

Le nom de l'expression régulière créée sera le nom utilisé lors des contrôles de validité.
La chaine va représenter le format de l'expression que l'on veut obtenir. Cette chaine est composée d'un ensemble de caractères (représentant les valeurs voulues ou plages de valeurs), et d'autres caractères représentant des opérateurs entre valeur, des positionnements dans la chaine, des nombres d'occurrence, des caractères spéciaux, etc.….

IV.2 ECRITURE D'UNE EXPRESSION REGULIERE

Les caractères et autres symboles spéciaux représentent des caractères non-imprimables comme des sauts de lignes (\n), des tabulations (\t) ou des options particulières ou encore des plages de lettres ou de chiffres, etc.

		Exp	Résultat
^	Début de chaîne	/^ab/	Recherche si la chaîne commence par ab
$	Fin de chaîne	/ab$/	Recherche si la chaîne finit par ab
.	Un caractère quelconque	/…/	Recherche si la chaîne a au moins 3 caractères
\r	désigne une nouvelle ligne (caractère «CR»)		
\n	désigne un retour en début de ligne (caractère «LF»)		
\t	désigne une tabulation horizontale		
\v	désigne une tabulation verticale		
\d	caractère numérique		Teste si le caractère est numérique
\D	caractère non numérique		Teste si le caractère n'est pas num
\r	désigne une nouvelle ligne (caractère «CR»)		

\n	désigne un retour en début de ligne (caractère «LF»)		
\t	désigne une tabulation horizontale		
\v	désigne une tabulation verticale		
\d	caractère numérique		Teste si le caractère est numérique
\D	caractère non numérique		Teste si le caractère est non numérique
\s	caractère blanc (espace, tabulation,...)		Teste si le caractère est blanc
\S	caractère non blanc		Teste si le caractère n'est pas numérique
\w	caractère alphanumérique (lettre ou chiffre)		Teste si le caractère est alphanumérique
\W	caractère non alphanumérique		Teste si le caractère n'est pas alphanumérique
[abc**]**	caractère a, b ou c		Teste si la chaine ne contient que les caractères a, b ou c
[^abc**]**	caractère autre que a, b ou c		Teste si la chaine ne contient que des caractères autres que a, b ou c
[a-h**]**	tout caractère entre a et h inclus		Teste si la chaine est composée de caractères entre a et h
[^a-h**]**	tout caractère non entre a et h inclus		Teste si la chaine ne contient pas de caractères entre a et h
*****	Le caractère précédent peut exister de zéro à plusieurs fois	/is*/	mie, m**is**, m**iss**ion
+	Le caractère précédent doit être trouvé de une à plusieurs fois	/is+/	Ma**is**on, m**iss**ion
?	Le caractère précédent est optionnel	/is?/	m**i**e, m**is**, m**is**sion
{n}	Le caractère précédent doit être trouvé un nombre *n* fois	/o{2}/	z**oo**
{n,m}	Le caractère précédent doit être trouvé au moins *n* fois et au plus *m* fois	/1{2,4}/	2**11**, 13**1111**
{n, }	Le caractère précédent doit être trouvé au moins *n* fois	/1{2,}/g	2**11**, 13**1111**, **111111**
ch1**|**ch2	présence soit de ch1 soit de ch2	/mois|jour|semaine/	..la durée de 7 **jour**s soit une **semaine**

Attention : pour tester un caractère qui est un méta caractère du langage (par exemple **^** ou **.**), il faut le faire précéder d'un \ (caractère d'échappement).

Exemples d'expression régulière

Pour tester si un champ est non vide :

```
var nonvideExp = /./;
```

Pour tester si un champ est composé de 3 chiffres :

```
var codeExp = /^\d{3}/;
```

Pour tester si un code de plusieurs lettres+2chiffres :

```
var code2Exp = /^[a-z]+[0-9]{2}/;
```

Pour tester un montant (avec ou sans virgule, en dollars) :

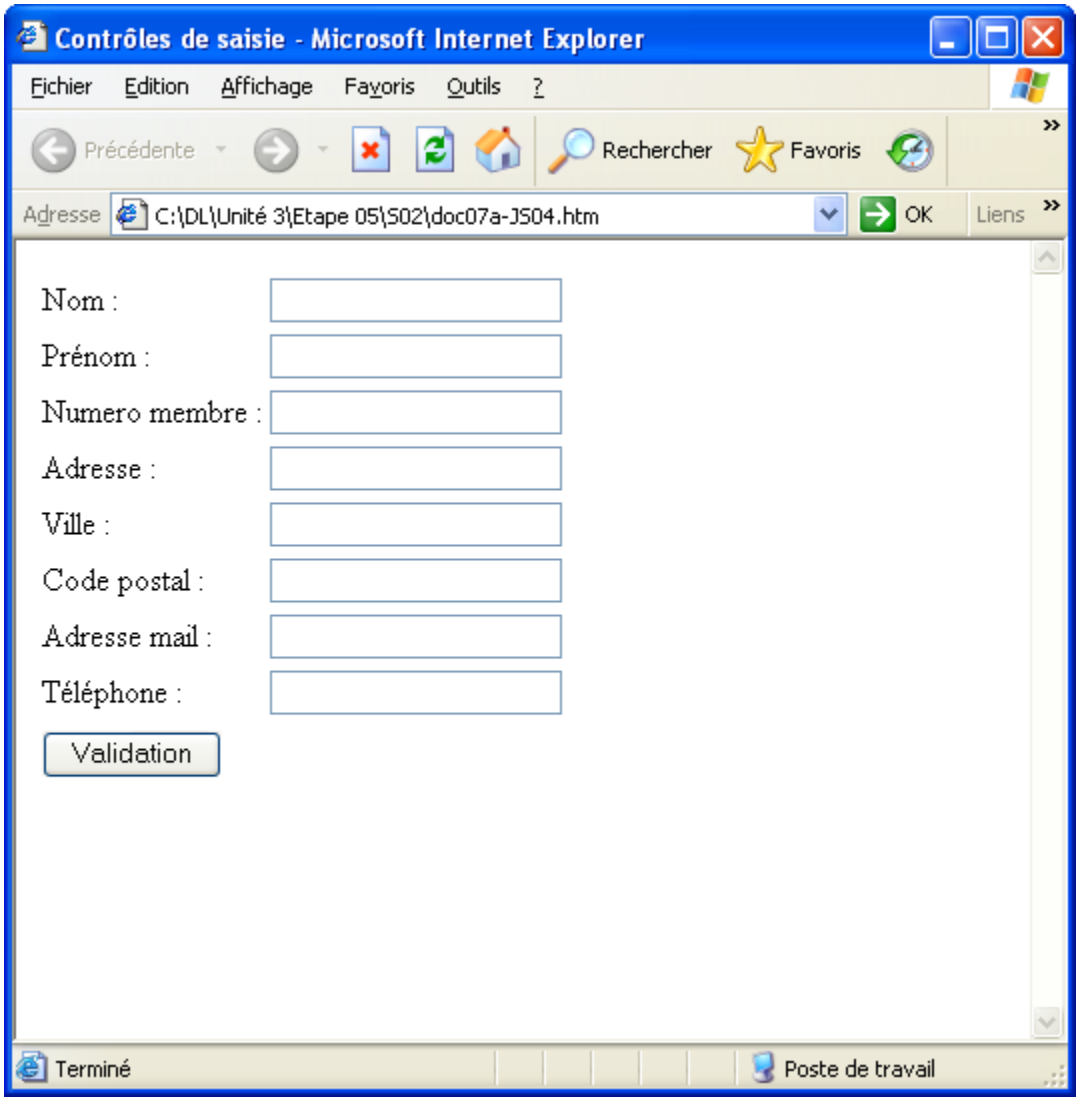
```
var montantExp = /^\d*$|^\d*\.\d{2}$/;
```

IV.3 TEST D'UNE CHAINE EN UTILISANT UNE EXPRESSION REGULIERE

Pour tester si une chaîne, après avoir défini l'expression régulière (nommée expReg), il suffit d'appeler la méthode test comme suit :
expReg**.test(**chaine**).**
Ceci renvoie un Booléen à true si une correspondance est trouvée dans chaîne avec l'expression expReg, et renvoie false sinon.
Il sera judicieux, dans les pages HTML, d'inclure ces contrôles dans des fonctions JavaScript.
D'autres méthodes sont aussi disponibles, qui permettent de remplacer des chaînes, de récupérer des chaînes, etc.…

IV.4 EXERCICES D'APPLICATION

Cet exercice va nous permettre d'utiliser une des grandes fonctionnalités de JavaScript, qui est extrêmement importantes et largement utilisée, à savoir l'utilisation de JavaScript pour effectuer des contrôles locaux (sur le client) : contrôles de saisie dans un formulaire. Pour cela, plusieurs manières de faire sont envisageables : on va utiliser ici les expressions régulières qui sont très intéressantes. Suite à ces contrôles locaux, si tout est OK, le formulaire sera envoyé, et une page html suivante sera affichée.
Dans le corrigé qui vous est proposé, les contrôles sont effectués au niveau du formulaire dans son ensemble. Il est bien entendu possible d'avoir des contrôles au niveau d'un champ (événement onblur). De même, on vous propose plusieurs manières d'imposer une saisie numérique (événement onkeypress, évènement onblur..).
Ecrire la page HTML suivante :



Effectuer en utilisant les expressions régulières, les contrôles suivants :

- Le nom est obligatoire
- Le prénom est obligatoire
- Le numéro de membre est composé de 8 chiffres
- L'adresse est obligatoire
- Le code postal est composé de 5 chiffres
- L'e-mail est sous la forme nom1@nom2.nom3 dans lesquels nom1 commence par une lettre minuscule, et est suivi d'une série de lettres minuscules, chiffres, underscore, tiret ou point. nom2 est une série de lettres minuscules, chiffres, underscore, tiret ou point, et nom3 est composé de 2 ou 3 lettres minuscules.
- Le téléphone est composé de 10 chiffres.

Si les contrôles sont bons, vous soumettrez le formulaire en affichant une page suivante.

Un peu plus de théorie
Le return devant un appel de fonction

L'événement **onSubmit** associé à l'objet *Form* permet de spécifier l'exécution sur le poste client (donc par le navigateur) d'une fonction JavaScript. Cette fonction s'exécute au moment où l'on clique sur le bouton *submit*.
Cette fonction peut renvoyer un booléen (true ou false) qui permettra de bloquer l'exécution du script CGI spécifié par l'attribut action tant que le booléen n'a pas la valeur *true*.

Exemple :
<form method=''post'' name=''form1'' action=''form2.htm'' onSubmit=''return verif()''>

Si nous ne mettons pas *return* devant l'appel de la function verif(), même si notre fonction a détecté des erreurs, le formulaire form2.htm s'affichera.
En mettant *return*, tant que la fonction retournera *false,* l'affichage de la feuille form2.htm ne s'effectuera pas.
En cas d'erreur et pour rester sur le formulaire, il faut donc faire précéder l'appel de la fonction de *return* et dans l'écriture de la fonction renvoyer *false* si des erreurs sont détectées.
Si tout est correct, la fonction renvoie *true* et dans ce cas la feuille form2.htm s'affichera.

La méthode alert()

Un message d'information en JavaScript est affiché par la méthode **alert()** de l'objet window. La syntaxe est :

window.alert(txtMessage) ;

L'objet window permet le raccourci dans la notation et l'écriture est souvent de la forme :

alert(txtMessage) ;

La méthode alert() ne retourne pas de valeur. Elle affiche une boite de dialogue avec une icône d'alerte (généralement le point d'exclamation), le message contenu dans la chaine de caractères *txtMessage* et un bouton OK. Tant que le visiteur ne clique pas sur le bouton OK, l'accès à la fenêtre est impossible (cette fenêtre est dite modale), il est donc contraint de lire le message.
Ce type de boite de dialogue est très utilisé pour informer (alerter) le visiteur sur un point important comme une valeur saisie incohérente ou une action impossible.
N'abusez pas de ce type de message cela peut devenir lassant et faire fuir les visiteurs.

Chaque navigateur a son propre style de boite de dialogue mais tous utilisent le même principe avec l'icône, le texte et le bouton.

Pour écrire le message d'alerte sur plusieurs lignes, il faut utiliser le caractère de saut de ligne \n.

Le message d'alerte est adapté pour afficher des données au développeur pendant la phase de test.

Exemple sous IE :

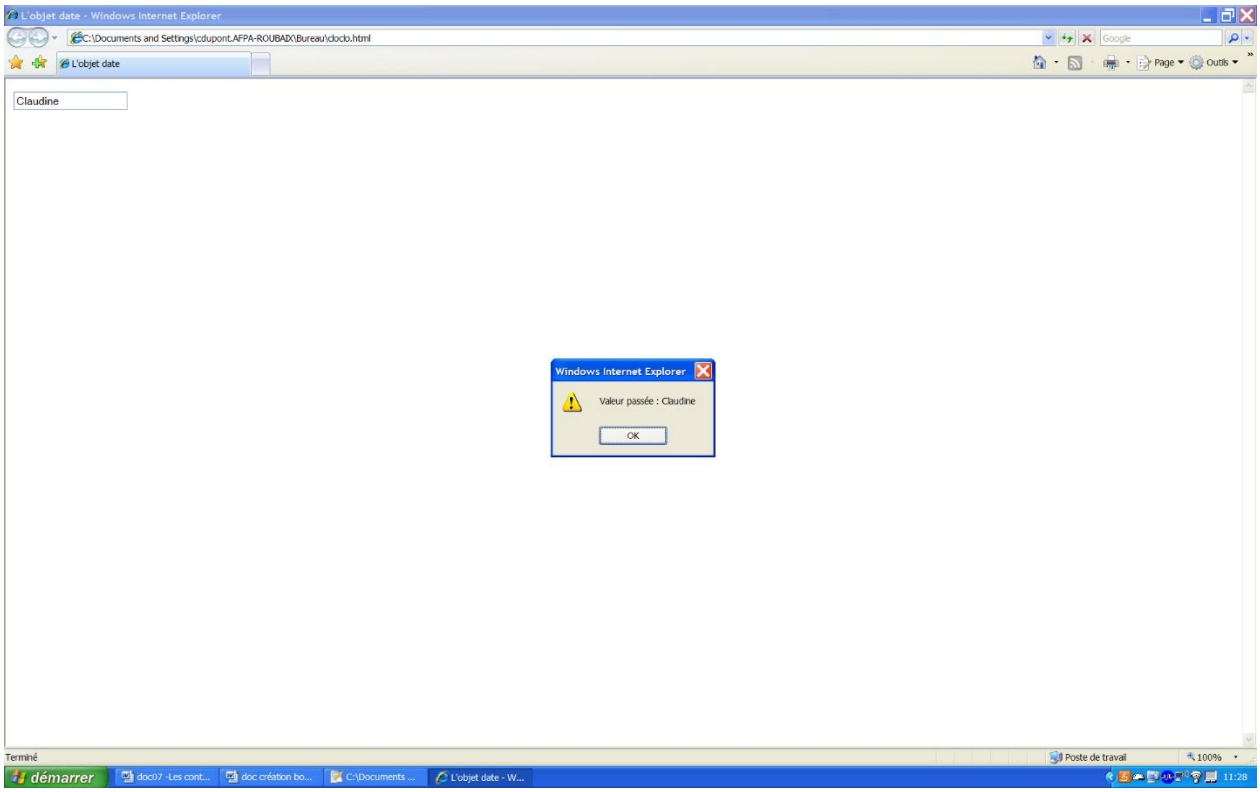
L'objet this

Le mot-clé *this* désigne l'objet en cours de manipulation. Lors de l'appel d'une fonction, il nous permet de passer la valeur d'un attribut du contrôle à partir duquel nous avons appelé la fonction.

Exemple :

```
<html>
  <head>
    <title>L'objet date</title>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
        <script type="text/javascript">
        function verifNom(nom) {
                alert("Valeur passée : " + nom);
                }
        </script>
  </head>
  <body>
  <input type="text" name="txtNom" value="" onBlur="verifNom(this.value)" />
  </body>
</html>
```

this.value passe la valeur de l'attribut *value* du champ input *txtNom*.



La méthode focus()

Cette méthode est utilisée pour donner le focus au champ c'est-à-dire placer le curseur dans la zone et autoriser la saisie via le clavier.
Cette méthode est utile pour signaler une erreur sur un champ de saisie et donner directement la possibilité au visiteur de corriger les informations enregistrées.

Le curseur se place derrière le dernier caractère de la zone. Le visiteur doit donc prendre sa souris pour sélectionner la zone à corriger et retaper la bonne information.
Avec la méthode **select()** vous pouvez donner le focus au champ et sélectionner toute la saisie. Le texte passe en surbrillance. Une saisie du visiteur sur un texte sélectionné efface directement le contenu initial du champ (très confortable pour l'utilisateur).

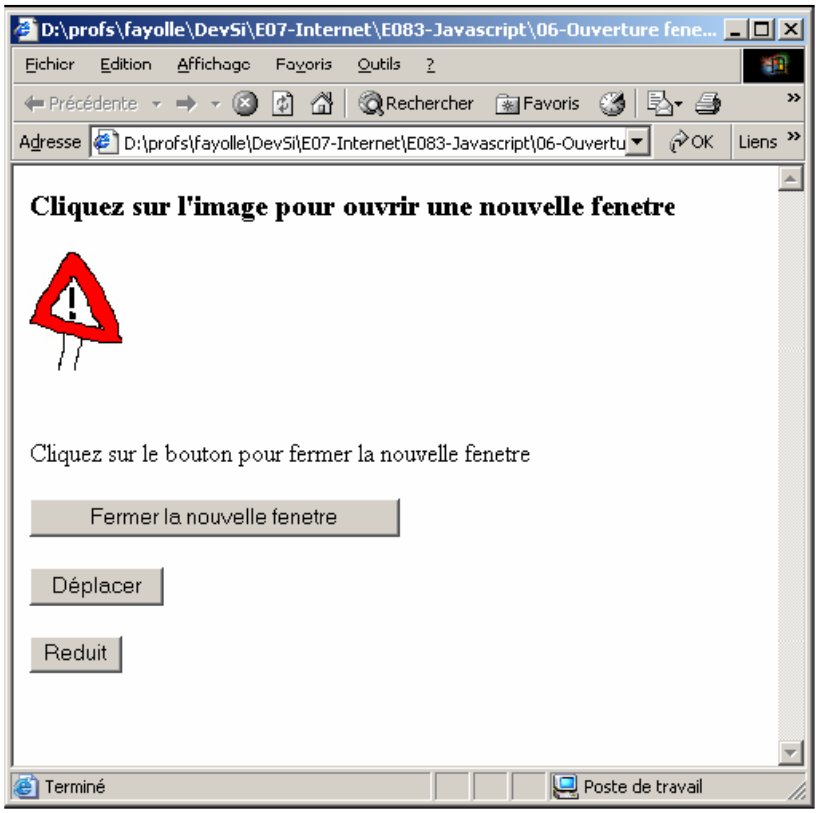
Expression régulière adresse mail :

nom1 *doit commencer par une lettre minuscule et est suivi d'une série de lettres minuscules, chiffres, undescore, tiret ou point.*
commencer par une lettre minuscule : [a-z]
suivi d'une série de lettres minuscules : [a-z]+ si on considère qu'il faut au moins un caractère après le premier
chiffres, undescore, tiret ou point : [a-z0-9_\-\.]+
devant le - et le . il faut mettre un \. Pour le _ ce n'est pas nécessaire.
donc pour nom1 : [a-z][a-z0-9_\-\.]+
derrière nom1 : @ => [a-z][a-z0-9_\-\.]+@
nom2 *est une série de lettres minuscules, chiffres, undescore, tiret ou point* : idem à 2ème partie de nom1 soit [a-z0-9_\-\.]+
nom1@nom2 donne [a-z][a-z0-9_\-\.]+@[a-z0-9_\-\.]+
derrière nom2 il faut un point : \.
nom1@nom2. donne [a-z][a-z0-9_\-\.]+@[a-z0-9_\-\.]+\.
nom3 *est composé de 2 ou 3 lettre minuscules* : lettres minuscules
[a-z], 2 ou 3 => **[a-z]{2,3}**
il ne faut rien d'autre devant => faire précéder l'expression par ^. On ne veut rien d'autre derrière, terminer l'expression par $.
l'expression finale est donc :
/^[a-z][a-z0-9_\-\.]+@[a-z0-9_\-\.]+\.[a-z]{2,3}$/

V GESTION DES FENETRES

V.1 OUVERTURE DES FENETRES

Nous allons ouvrir une nouvelle fenêtre en JavaScript et faire interagir ces deux fenêtres. Développer la page HTML suivante :



Lorsque l'on clique sur l'image, la fenêtre suivante va s'ouvrir :

Depuis la fenêtre fille on aura deux possibilités :

- fermer la fenêtre mère (opener.close())
- fermer la fenêtre fille (self.close()).

Depuis la fenêtre mère, on peut :

- Ouvrir un fenêtre fille (nouvelle_fenetre.open(……))
- fermer cette fenêtre fille nouvelle_fenetre.close())
- déplacer cette fenêtre fille (nouvelle_fenetre.moveBy(….))
- réduire cette fenêtre fille (nouvelle_fenetre. resizeTo(….)).

Un peu de théorie

Le mot **pop-up** peut se traduire par fenêtre « surgissante ». Il s'agit le plus souvent d'une fenêtre de taille réduite qui s'ouvre soit automatiquement, soit suite à une action déclenchée par l'utilisateur. L'objectif est d'afficher des informations qui viennent en complément de la page principale du site. Les pop-ups sont adaptés pour afficher des agrandissements de vignettes de photo ou de produit, le détail d'un produit, une aide contextuelle, etc.

L'ouverture d'un pop-up en JS

Elle se fait par la méthode **open()** de *window*. Elle retourne un nouvel objet de type *window* contenant les propriétés du pop-up.

var winPopup = window.open(txtUrl, txtName, txtOption) ;

L'objet winPopup est de type *window*. Il permet de communiquer avec le pop-up qui vient de s'ouvrir. Les 3 paramètres sont des chaines de caractères :

- txtUrl :
  adresse URL de la page à charger dans le pop-up
- txtName :
  nom du pop-up. Dès lors que le pop-up a un nom défini et qu'il est ouvert, toute nouvelle ouverture provoque son remplacement par un nouveau pop-up. Cela évite d'envahir l'écran avec de multiples fenêtres. Il n'est pas obligatoire mais s'il est omis, le pop-up sera toujours ouvert dans une nouvelle fenêtre.
- txtOption :
  liste d'options d'ouverture et d'affichage du pop-up (barre d'outils, de menus, de statut, ascenseur…). Il n'est pas obligatoire. En cas d'omission, la fenêtre sera ouverte sous l'apparence d'une fenêtre classique. Elle est sous la forme d'une chaine de caractères c'est-à-dire commence par '' et finit par ''.
  La chaine est composée de couples nomPropriete=valeur, chaque couple étant séparé par une virgule.

Exemple de création d'un pop-up à dimensions fixes, positionné en haut à gauche de l'écran :
var winPopup=window.open(''popup.html'','''','top=10, left=10, width=200, height=200'');

Liste des options disponibles :

Option	Signification	Valeur attendue
directories	Affiche ou non la barre de liens	Yes ou no
menubar	Affiche ou non la barre de menus	Yes ou no
status	Affiche ou non la barre de statut	Yes ou no
location	Affiche ou non la barre d'adresse	Yes ou no
scrollbars	Affiche ou non des ascenseurs	Yes, no ou auto
resizable	Autorise ou non le redimensionnement de la fenêtre par l'utilisateur	Yes ou no
height	Hauteur en pixels	Nombre entier
width	Largeur en pixels	Nombre entier
left	Position horizontal en pixels sur l'écran par rapport au bord gauche	Nombre entier
top	Position verticale en pixels sur l'écran par rapport au bord supérieur	Nombre entier
fullscreen	Affiche ou non en plein écran	Yes ou no

Pour placer un pop-up au premier plan, ajoutez le code suivant : winPopup.focus() ;

La communication entre fenêtres

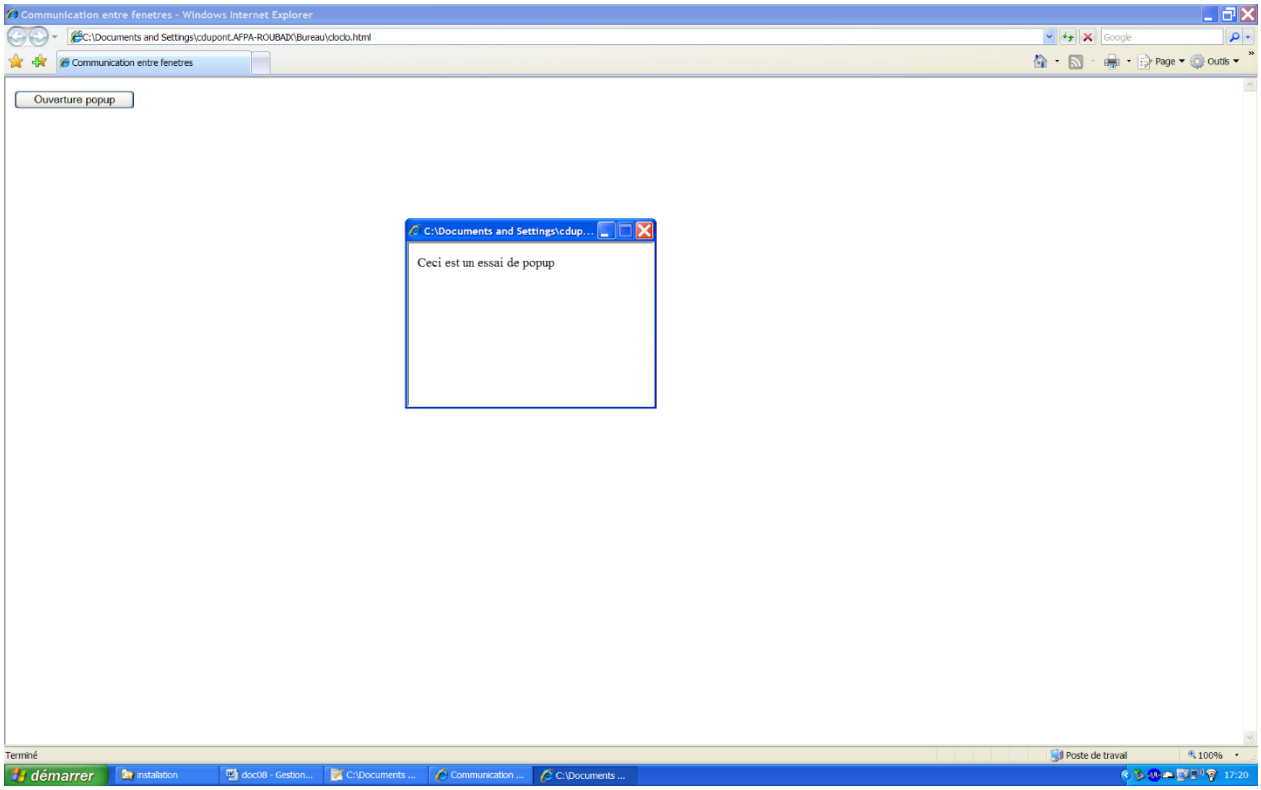
Grâce à l'objet de type window retourner par l'appel à *open()*, il est possible de communiquer avec le pop-up depuis la fenêtre appelante.
Pour communiquer avec la fenêtre appelante depuis le pop-up, il faut utiliser l'objet **opener** lié à l'objet window du pop-up.

Script de la fenêtre principale qui ouvre le pop-up :

```
<html>
 <head>
  <title>Communication entre fenetres</title>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
      <script type="text/javascript">
      function openPopup() {
            var      winPopup=window.open("popup.html","wPopup","width=300,
                  height=200, menubar=no, scrollbars=no");
            winPopup.focus();  // pour mettre le popup au premier plan
            }
      </script>
 </head>
 <body>
 <input type="button" value="Ouverture popup" onClick="openPopup()" /><br /><br />
```

```
 </body>
</html>
```

Quand nous cliquons sur le bouton voilà ce qui s'affiche :



Script de la fenêtre principale qui ferme le popup :

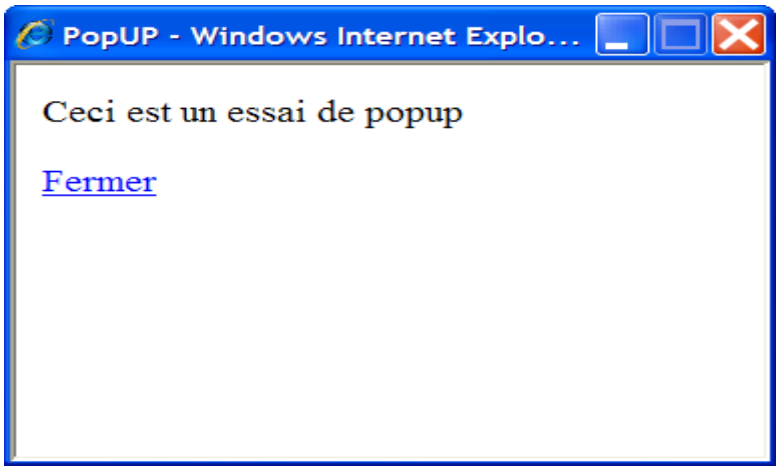
```
<html>
 <head>
  <title>Communication entre fenetres</title>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
       <script type="text/javascript">
       function openPopup() {
            winPopup=window.open("popup.html","wPopup","width=300,    height=200,
menubar=no, scrollbars=no");
            winPopup.focus();            // pour mettre le popup au premier plan
       }
       function closePopup() {
            // je teste pour savoir si le popup existe
            if(winPopup) {
                 // je teste si le popup est encore ouvert
                 if(!winPopup.closed) {
                      winPopup.close();
                 }
            }
       }
```

```
        </script>
  </head>
  <body>
<input type="button"  value="Ouverture popup"
        onClick="javascript:openPopup()" /><br /><br />
  <a href="javascript:closePopup()">Fermer le popup</a>
  </body>
</html>
```

La communication avec le pop-up se fait via l'objet *winPopup* (nom que j'ai donné à la création du pop-up) ouvert par la méthode *open()* dans la fonction openPopup().
Avant de fermer, il faut tester si l'objet existe sinon l'appel de la fonction close déclencherait une erreur, idem si le pop-up a déjà été fermé.

Fermeture du pop-up à partir du pop-up :

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>PopUP</TITLE>
</HEAD>
<BODY>
Ceci est un essai de popup<br /><br />
<a href="javascript:window.close()">Fermer</a>
</BODY>
</HTML>
```



Passage d'une information du pop-up à la page principale :

Code de la page principale :

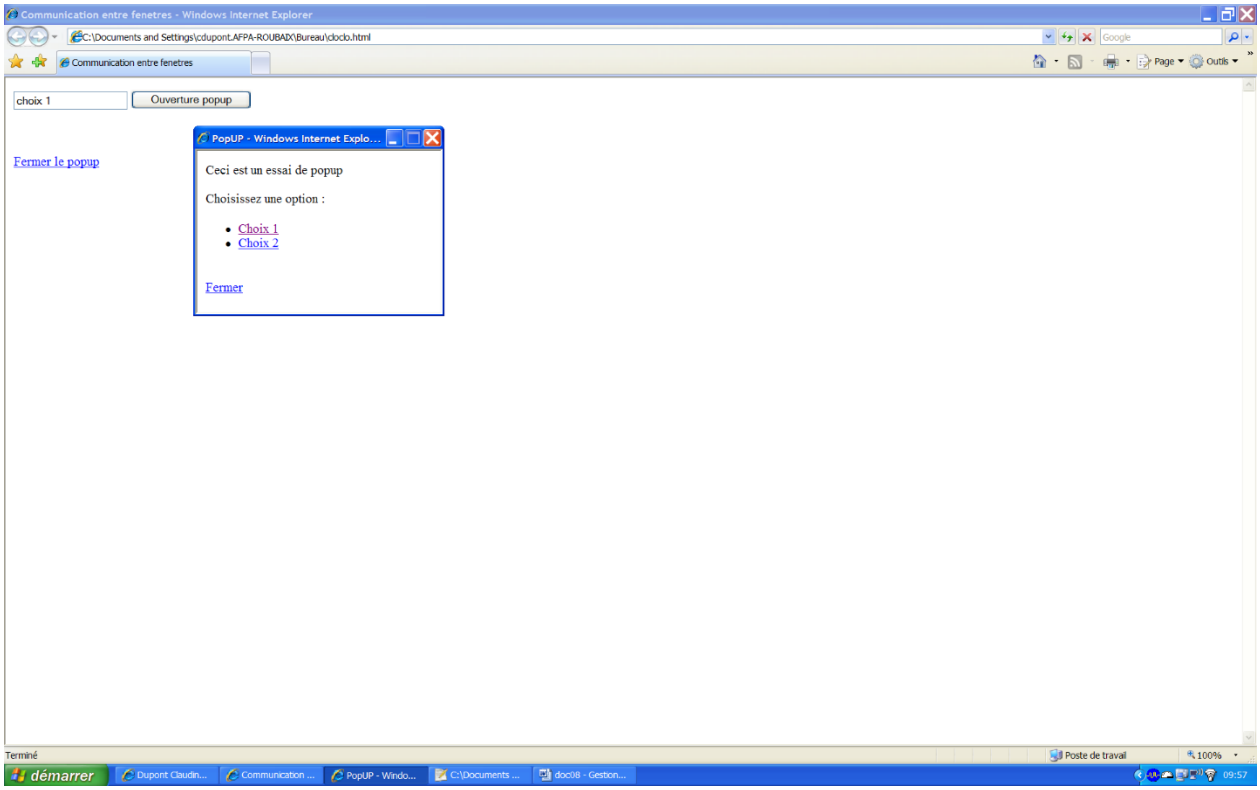
```
<body>
  <form name="form1">
  <input type="text" name="choix" value="" />
```

```
  <input type="button"  value="Ouverture popup" onClick="javascript:openPopup()" />
  </form>
  <br /><br />
  <a href="javascript:closePopup()">Fermer le popup</a>
  </body>
```

Code du pop-up :

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>PopUP</TITLE>
<script type="text/javascript">
        function message(text) {
// vérifier que la page appelante existe toujours et qu'elle n'est pas fermée
                if (window.opener) {
// verifier que la page n'a pas été remplacée
                        if(window.opener.document.form1) {
                                window.opener.document.form1.choix.value=text;
                        }
                }
                else
                        {alert("la fenetre principale n'est plus active");}
        }
</script>
</HEAD>
<BODY>
Ceci est un essai de popup<br /><br />
Choisissez une option :
<ul>
<li><a href="javascript:message('choix 1')">Choix 1</a></li>
<li><a href="javascript:message('choix 2')">Choix 2</a></li>
</ul>
<br />
<a href="javascript:window.close()">Fermer</a>
</BODY>
</HTML>
```

Dans le pop-up, il existe 2 liens pour choisir entre 2 messages. Ce choix s'affiche dans la zone de texte de la fenêtre principale.
Pour éviter les erreurs JS, il est indispensable de s'assurer que la page appelante (fenêtre principale) existe toujours et qu'elle n'a pas été fermée par le visiteur ou que la page n'a pas été remplacée par une autre.
La page appelante est accessible par **window.opener**.
Pour vérifier que la page n'a pas été remplacée, il faut vérifier que la zone du document destinée à recevoir le choix est bien présente sur la page appelante.



Fermeture automatique des pop-ups

Les pop-ups doivent toujours être vus comme une aide à la navigation. Une fermeture aisée et automatique sera appréciée et évite de surcharger l'écran du visiteur.

Exemple de code à mettre dans le pop-up :

```
function closeAuto(){
                window.close();
        }
setTimeout("closeAuto()",5*1000);        // fermeture dans 5 secondes
```

La fin des pop-ups

Les pop-ups sont souvent associés à des affichages publicitaires. Certains les nomment « popubs ». Leur réputation est mauvaise. Les éditeurs de navigateurs ont pratiquement tous intégré un système par défaut de blocage des pop-ups. Il faut donc toujours prévoir une solution pour les utilisateurs qui ne parviennent pas à les ouvrir. Le script de détection d'ouverture permet d'exécuter une action en cas de blocage.
Il faut absolument éviter leur emploi pour des traitements importants comme des inscriptions, des ajouts de produits dans un panier de commande, des paiements, etc…

Méthodes moveBy() et resizeTo()

La méthode **moveBy()** permet de déplacer une fenêtre par rapport à sa position courante.
moveBy(deplacementLeft, deplacementTop) ;

La fenêtre est déplacée d'un nombre *deplacementLeft* de pixels vers la gauche et *deplacementTop* de pixels vers le bas.
Les paramètres peuvent être négatifs : le déplacement s'opère alors vers la droite ou vers le haut.

Il existe 2 méthodes permettant de modifier les dimensions d'une fenêtre.
La méthode **resizeTo()** redimensionne la fenêtre selon des valeurs fixées.

resizeTo(pixelLargeur, pixelHauteur) ;

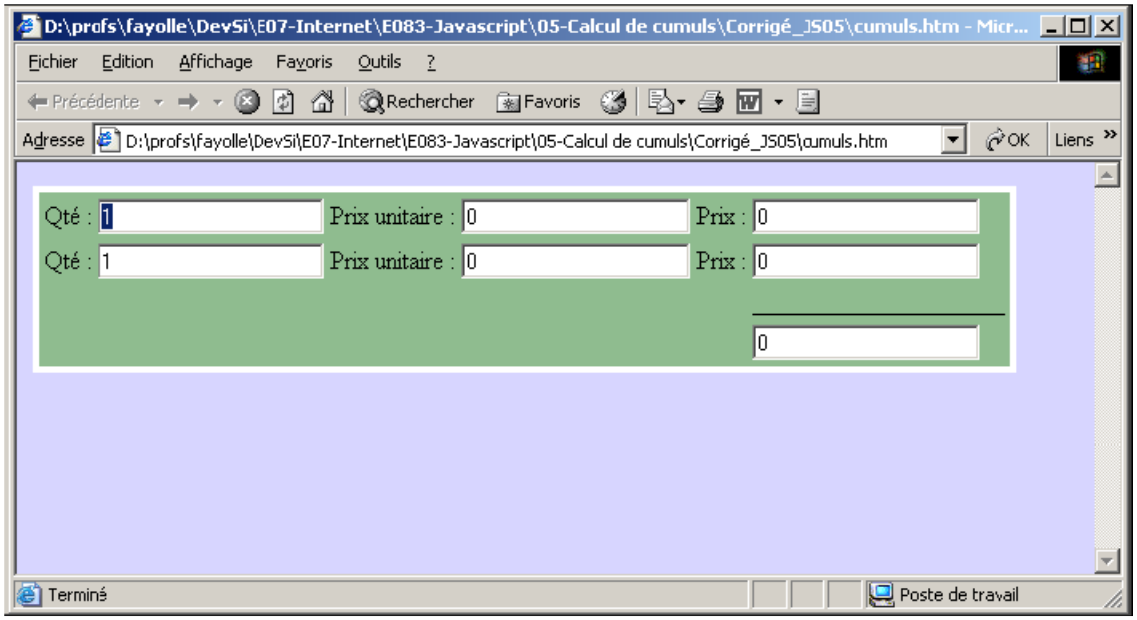
La fenêtre prend comme largeur la valeur en pixels de *pixelLargeur* et comme hauteur la valeur en pixels de *pixelHauteur*.

La méthode **resizeBy()** permet de modifier la taille par rapport à la taille courante de la fenêtre.
resizeBy(pixelLargeur, pixelHauteur) ;

La fenêtre est agrandie de *pixelLargeur* pixels en largeur et de *pixelHauteur* pixels en hauteur. Les paramètres peuvent être négatifs dans le cas d'une diminution de taille. La position du coin supérieur gauche n'est pas modifiée par un redimensionnement.
Il n'existe pas de méthode pour agrandir directement une fenêtre de sorte qu'elle occupe la totalité de l'écran.

VI CALCULS DYNAMIQUES

Ecrire la page HTML suivante :



Modifier dynamiquement (en utilisant JavaScript) les sous-totaux et le total général lorsque l'on change un prix unitaire ou une quantité.
Contrôler que les quantités et les prix unitaires saisis sont numériques (les décimales sont autorisées…).

Points abordés :

La fonction parseFloat()
La propriété length
La méthode substring() de la classe String
L'objet event
Le codage ASCII

VI.1 UN PEU DE THEORIE

Les conversions

JS n'est pas un langage fortement typé mais quand nous gérons des opérations mathématiques, il est nécessaire de s'assurer que la variable à manipuler est bien de type numérique.
La valeur que nous récupérons dans la propriété *value* d'un *input* de type *text* est une chaine de caractères.
Avant tout calcul mathématique, il faut donc convertir le contenu de cette propriété en type numérique.
Il existe 2 fonctions principales pour convertir automatiquement une chaine de caractères en nombre :

parseFloat() qui convertit en nombre décimal
parseInt() qui convertit en nombre entier.

```
montant = parsefloat(txtNombre) ;
```

Les fonctions *parseInt()* et *parseFloat()* ont un comportement particulier avec les chaines de caractères qui commencent par un nombre.
La conversion se fait caractère par caractère tant que le caractère suivant peut compléter le nombre initial. Dès qu'un caractère invalide est rencontré, la conversion s'arrête et retourne la conversion de la première partie.
Quand la chaine de caractères n'est pas convertible en nombre, le résultat de la conversion vaut *NaN* (abréviation de Not A Number) qui est une constante de type *number*.
La fonction *isNaN()* permet de détecter la valeur *NaN* d'une conversion. Elle retourne *true* si le paramètre n'est pas un nombre ou une chaine au format nombre. Cette fonction vérifie si le paramètre est bien un nombre, elle analyse la chaine dans son intégralité. La chaine « 12,95 » n'est pas un nombre car le séparateur décimal est le point.

L'objet String (chaine de caractères)

C'est grâce aux méthodes et aux propriétés de l'objet String que les manipulations sur les chaines de caractères sont possibles en JS.

La longueur d'une chaine

La longueur d'une chaine de caractères est donnée par la propriété *length*. Elle permet de parcourir un à un avec une boucle *for* les caractères de la chaine.
La longueur d'une chaine vide est 0.

var longueur = text1.length.

L'accès à un caractère

Pour accéder à un caractère d'une chaine, nous utiliserons la méthode **charAt()** qui nécessite en paramètre le numéro du caractère à retourner.
Attention, comme pour les tableaux, le premier élément est toujours à la position 0. le dernier élément a donc comme indice *chaine.length – 1*.
Cette méthode permet uniquement de lire un caractère. Il est impossible de modifier une chaine avec cette méthode.

Le codage des caractères

Les caractères d'une chaine sont stockés en mémoire sous forme d'un code numérique. JS permet de manipuler le codage des caractères.
La méthode **charCodeAt()** renvoie la valeur numérique décimale du code correspond à un caractère d'une chaine. Elle attend en paramètre l'indice du caractère à analyser. La table de correspondance de caractères est la table ASCII.

La méthode inverse **fromCharCode()** retourne un caractère à partir de son code. Elle attend en paramètre une liste de codes de caractère. Elle retourne la chaine composée de tous les caractères.

Exemple :
String.fromCharCode(66,111,110,106,111,117,114) ; renvoie la chaine « Bonjour »

L'extraction d'une sous-chaine

Pour extraire un morceau de chaine de caractères, la méthode **substring()** est idéale. Elle attend 2 paramètres : la position du premier caractère et la positon du dernier caractère de la chaine à extraire.
chaine.substring(debut, fin)
Attention, le caractère à la positon *fin* n'st pas inclus dans la chaîne extraite.
Le premier caractère d'une chaine a comme position 0.
Il est possible de ne passer qu'un seul paramètre à la méthode. Dans ce cas, la chaine retournée est celle qui commence au paramètre et qui se termine à la fin de la chaine.

La méthode **substr()** est comparable à **substring()**. La seule différence est que le second paramètre attendu est le nombre de caractères à extraire et non la positon de fin de la sous-chaine.

La recherche dans une chaine

Pour faciliter la recherche de caractères dans une chaine et éviter le recours systématique à une boucle afin de parcourir un à un les caractères, JS met à notre disposition 2 méthodes : **indexOf()** et **lastIndexOf()**.

lastIndexOf() retourne la position de la dernière sous-chaine correspond à une chaine recherchée. Si la sous-chaine n'est pas présente dans la chaine, elle retourne -1.

indexOf() retourne la position du prochain caractère correspond à une chaine recherchée. Si la recherche n'est pas satisfaisante, la méthode retourne -1.

La gestion des majuscules et minuscules

JS permet de forcer l'écriture d'une chaine en majuscules ou en minuscules, quelle que soit l'écriture initiale. La distinction entre minuscules et majuscules s'appelle la casse.
toLowerCase() transforme la chaine en minuscules, **toUpperCase()** en majuscules.

Les événements

Nous avons déjà vu les événements classiques associés aux balises HTML : onFocus, onClick,…
Il existe un moyen plus global de les gérer en utilisant l'objet **event**.
Attention, malheureusement cet objet n'est pas manipulé de la même façon dans les navigateurs.

L'objet **event** est créé automatiquement par le navigateur. Il est passé en paramètre de la fonction gestionnaire d'événement que le développeur écrit soit de manière automatique avec IE soit par programmation.

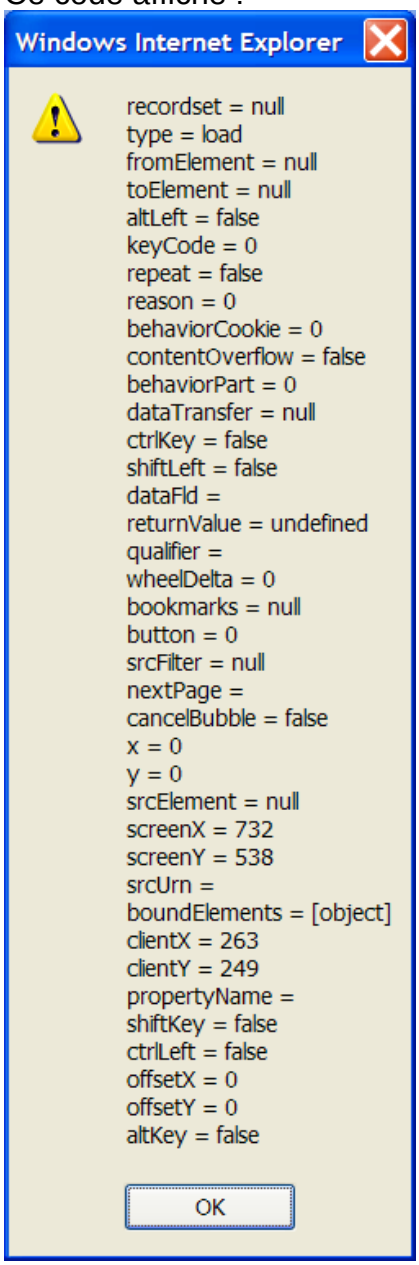
Regardons ce que ce gestionnaire d'événements me renvoie sous IE et Firefox.
Ecrivons un petit script JS qui s'exécute chargement d'une page et regardons toutes les propriétés et les valeurs que l'objet *event* généré automatiquement contient.

Version IE :

```
<html>
  <head>
    <title>Objet event</title>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
        <script type="text/javascript">
                function traiterEv() {
                        var txt="";
                        for (i in event)
                        {
                                txt += i + " = " + event[i] + "\n";
                        }
                alert (txt);
```

```
            }
        </script>
  </head>
  <body onLoad="traiterEv()">
  </body>
</html>
```

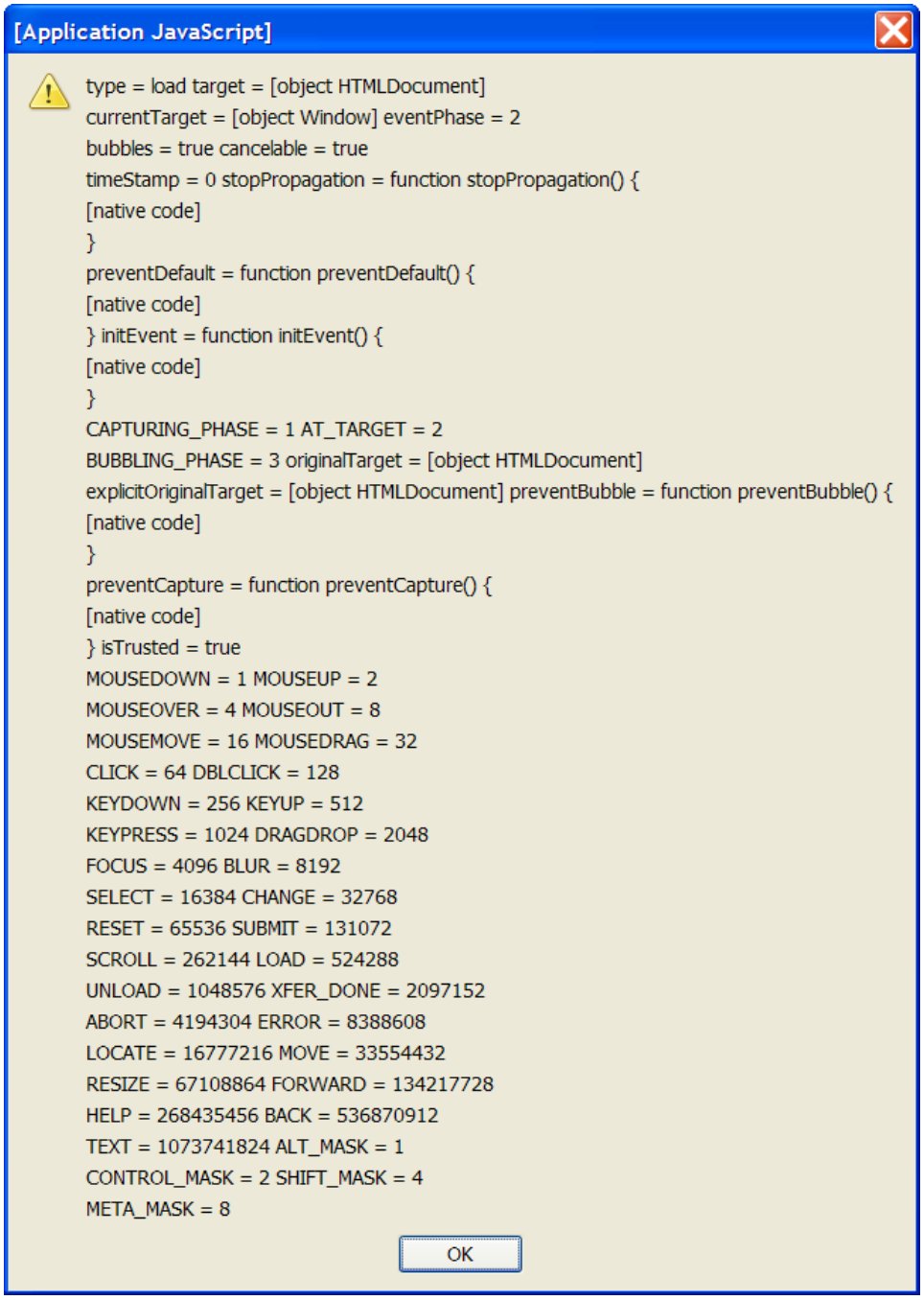
Ce code affiche :



Version Firefox :

```
<html>
  <head>
    <title>Objet event</title>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
        <script type="text/javascript">
function aff(e)
{
        var txt = "";
        for (i in e)
        {
                txt += i + " = " + e[i] + "\n";
        }
        alert (txt);
}
window.captureEvents(Event.LOAD);
window.onload = aff;
</script>
  </head>
  <body>   </body>
</html>
```

Ce code affiche :



Certaines propriétés sont différentes.
La propriété *keyCode* contient le code ASCII de la touche enfoncée pendant l'événement (IE).
La propriété *charCode* contient le code ASCII de la touche enfoncée pendant l'événement (Firefox).

Table ASCII

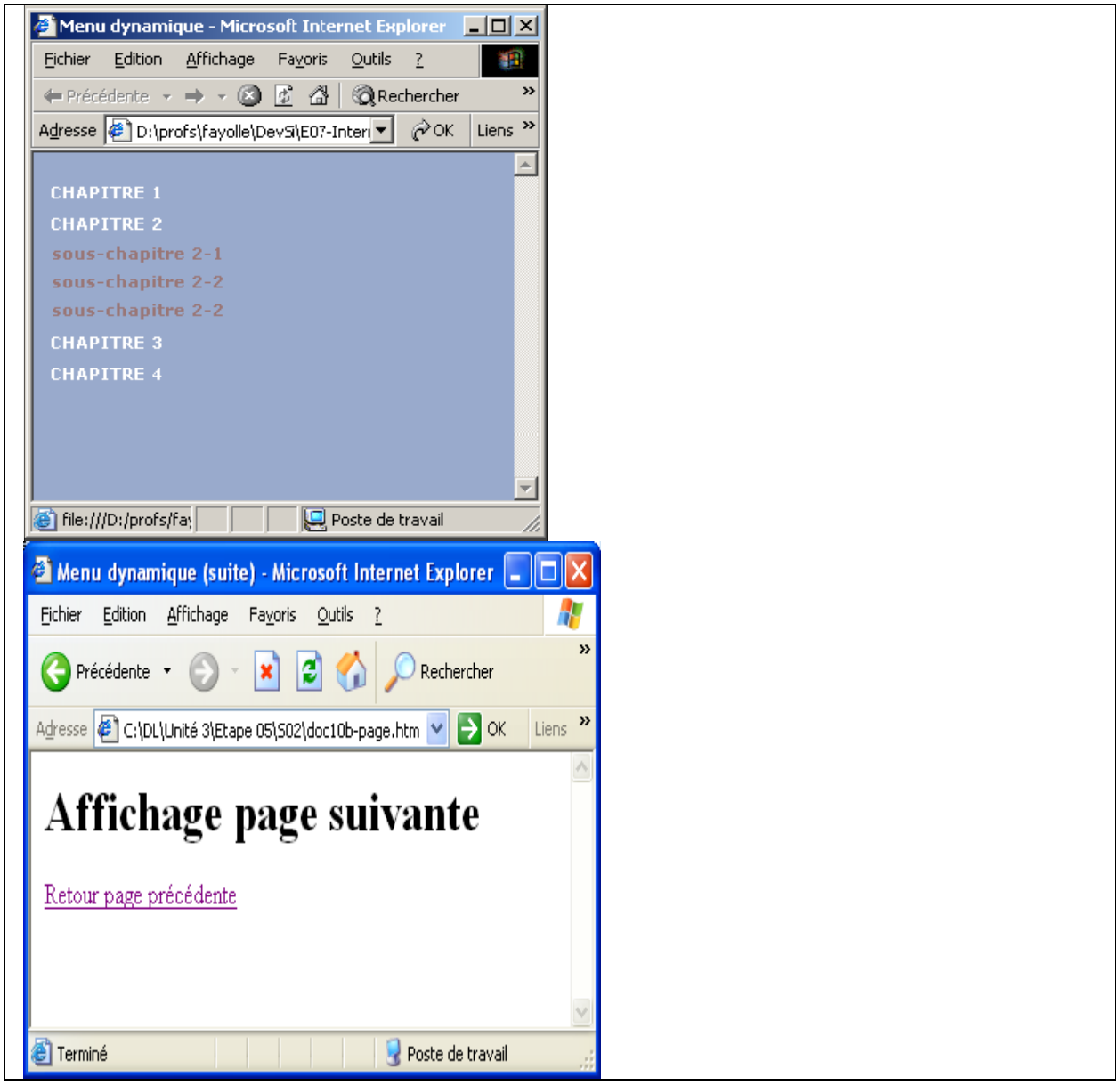
Décimal	Octal	Hex	Binaire	Caractère	
000	000	00	00000000	NUL	(Null char.)
001	001	01	00000001	SOH	(Start of Header)
002	002	02	00000010	STX	(Start of Text)
003	003	03	00000011	ETX	(End of Text)
004	004	04	00000100	EOT	(End of Transmission)
005	005	05	00000101	ENQ	(Enquiry)
006	006	06	00000110	ACK	(Acknowledgment)
007	007	07	00000111	BEL	(Bell)
008	010	08	00001000	BS	(Backspace)
009	011	09	00001001	HT	(Horizontal Tab)
010	012	0A	00001010	LF	(Line Feed)
011	013	0B	00001011	VT	(Vertical Tab)
012	014	0C	00001100	FF	(Form Feed)
013	015	0D	00001101	CR	(Carriage Return)
014	016	0E	00001110	SO	(Shift Out)
015	017	0F	00001111	SI	(Shift In)
016	020	10	00010000	DLE	(Data Link Escape)
017	021	11	00010001	DC1	(XON)(Device Control 1)
018	022	12	00010010	DC2	(Device Control 2)
019	023	13	00010011	DC3	(XOFF)(Device Control 3)
020	024	14	00010100	DC4	(Device Control 4)
021	025	15	00010101	NAK	(Negative Acknowledgement)
022	026	16	00010110	SYN	(Synchronous Idle)
023	027	17	00010111	ETB	(End of Trans. Block)
024	030	18	00011000	CAN	(Cancel)
025	031	19	00011001	EM	(End of Medium)
026	032	1A	00011010	SUB	(Substitute)
027	033	1B	00011011	ESC	(Escape)
028	034	1C	00011100	FS	(File Separator)
029	035	1D	00011101	GS	(Group Separator)
030	036	1E	00011110	RS	(Request to Send)(Record Separator)
031	037	1F	00011111	US	(Unit Separator)
032	040	20	00100000	SP	(Space)
033	041	21	00100001	!	(exclamation mark)
034	042	22	00100010	"	(double quote)
035	043	23	00100011	#	(number sign)
036	044	24	00100100	$	(dollar sign)
037	045	25	00100101	%	(percent)
038	046	26	00100110	&	(ampersand)
039	047	27	00100111	'	(single quote)
040	050	28	00101000	(	(left opening parenthesis)
041	051	29	00101001	)	(right closing parenthesis)
042	052	2A	00101010	*	(asterisk)
043	053	2B	00101011	+	(plus)
044	054	2C	00101100	,	(comma)
045	055	2D	00101101	-	(minus or dash)
046	056	2E	00101110	.	(dot)
047	057	2F	00101111	/	(forward slash)
048	060	30	00110000	0	
049	061	31	00110001	1	
050	062	32	00110010	2	
051	063	33	00110011	3	
052	064	34	00110100	4	

053	065	35	00110101	5	
054	066	36	00110110	6	
055	067	37	00110111	7	
056	070	38	00111000	8	
057	071	39	00111001	9	
058	072	3A	00111010	:	(colon)
059	073	3B	00111011	;	(semi-colon)
060	074	3C	00111100	<	(less than sign)
061	075	3D	00111101	=	(equal sign)
062	076	3E	00111110	>	(greater than sign)
063	077	3F	00111111	?	(question mark)
064	100	40	01000000	@	(AT symbol)
065	101	41	01000001	A	
066	102	42	01000010	B	
067	103	43	01000011	C	
068	104	44	01000100	D	
069	105	45	01000101	E	
070	106	46	01000110	F	
071	107	47	01000111	G	
072	110	48	01001000	H	
073	111	49	01001001	I	
074	112	4A	01001010	J	
075	113	4B	01001011	K	
076	114	4C	01001100	L	
077	115	4D	01001101	M	
078	116	4E	01001110	N	
079	117	4F	01001111	O	
080	120	50	01010000	P	
081	121	51	01010001	Q	
082	122	52	01010010	R	
083	123	53	01010011	S	
084	124	54	01010100	T	
085	125	55	01010101	U	
086	126	56	01010110	V	
087	127	57	01010111	W	
088	130	58	01011000	X	
089	131	59	01011001	Y	
090	132	5A	01011010	Z	
091	133	5B	01011011	[	(left opening bracket)
092	134	5C	01011100	\	(back slash)
093	135	5D	01011101	]	(right closing bracket)
094	136	5E	01011110	^	(caret cirumflex)
095	137	5F	01011111	_	(underscore)
096	140	60	01100000	`	
097	141	61	01100001	a	
098	142	62	01100010	b	
099	143	63	01100011	c	
100	144	64	01100100	d	
101	145	65	01100101	e	
102	146	66	01100110	f	
103	147	67	01100111	g	
104	150	68	01101000	h	
105	151	69	01101001	i	
106	152	6A	01101010	j	
107	153	6B	01101011	k	
108	154	6C	01101100	l	
109	155	6D	01101101	m	
110	156	6E	01101110	n	

111	157	6F	01101111	o	
112	160	70	01110000	p	
113	161	71	01110001	q	
114	162	72	01110010	r	
115	163	73	01110011	s	
116	164	74	01110100	t	
117	165	75	01110101	u	
118	166	76	01110110	v	
119	167	77	01110111	w	
120	170	78	01111000	x	
121	171	79	01111001	y	
122	172	7A	01111010	z	
123	173	7B	01111011	{	(left opening brace)
124	174	7C	01111100	|	(vertical bar)
125	175	7D	01111101	}	(right closing brace)
126	176	7E	01111110	~	(tilde)
127	177	7F	01111111	DEL	(delete)

VII LES MENUS DYNAMIQUES (MENUS DEROULANTS)

Elaboration de menus dynamiques. Ecrire la page HTML suivante.
Lorsque le curseur survole une ligne chapitre (ici CHAPITRE2), le sous menu correspondant s'affiche. Chaque ligne du sous-menu est constituée de lien qui débranche vers une page suivante.



Cet affichage (ou ce masquage des sous-menus) s'effectue en utilisant des propriétés des styles. Pour chaque menu, on va inclure les sous-menus dans une balise DIV (nommée menu1 pour la première ligne de menu par exemple). On pourra alors cacher l'ensemble du contenu de la balise (donc tout le détail du menu1 par exemple) en affectant comme style à cette balise {display='none'} et on pourra à l'inverse l'afficher en affectant le style {display= 'block'}.

VII.1 UN PEU DE THEORIE

L'accès aux propriétés de style

Deux propriétés gèrent la visibilité d'un élément sur le document :

visibility vaut *visible* ou *hidden* selon que l'élément est visible ou non.

display vaut *none* si l'élément est caché et vaut *block* ou *inline* ou même reste vide si l'élément est visible.

Ces 2 propriétés peuvent sembler équivalentes mais il existe une différence importante. Un élément ayant la propriété *visibility* fixée à *hidden* ne sera pas affiché mais il occupera malgré tout la place qui lui est nécessaire dans le document pour l'affichage de son contenu. Si *display* vaut *none* l'élément est caché et la place qu'il occuperait est entièrement libérée.

La propriété style

Pour toutes les propriétés de style CSS, il faut accèder à la propriété de style de l'élément en passant par la propriété objet **style**.
Par exemple, la visibilité est accessible avec :
nomElement.style.visibility
Si vous oubliez *style* dans la notation pointée, le script ne fonctionnera pas et générera une erreur.

Les événements onmouseover et onmouseout

L'événement **onmouseover** se déclenche quand la souris entre sur la zone définie par la balise.
A l'inverse, l'événement **onmouseout** se déclenche quand la souris quitte la zone de la balise.
Ces 2 événements permettent de réaliser des effets graphiques sur les pages.

VIII LES COOKIES

VIII.1 PRESENTATION

Les cookies sont utilisés par la plupart des sites Web.
Ils représentent la seule solution pour stocker de manière permanente des informations sur le poste du visiteur. Ces informations sont réutilisables lors des visites suivantes sur le site.
HTTP est un protocole 'sans état' c'est-à-dire qui ne permet pas de conserver des informations d'une transaction à une autre. Les cookies sont donc un des moyens de pallier à ce manque.

Voici quelques exemples d'utilisation :

- ✓ Cookie enregistrant un identifiant unique pour une reconnaissance automatique du visiteur et de son compte client
- ✓ Cookie enregistrant une liste de paramètres de préférence de navigation pour personnaliser la page présentée (langue, rubriques préférées…)
- ✓ Cookie enregistrant la date de la dernière visite

Les cookies sont aussi utilisés par les responsables de sites pour optimiser la rentabilité de leurs pages (ex : cookie enregistrant l'origine d'un visiteur pour déterminer les partenariats les plus efficaces).

Les cookies sont enregistrés sur le disque dur du poste client, dans un fichier dédié dans un format texte. Le stockage est différent selon les navigateurs mais la manipulation et les fonctionnalités sont identiques.

Sous Mozilla v1.4, on les trouve sur le chemin :

C:\Documents and Settings\<user>\Application Data\Mozilla\Firefox\Profiles\<nomAleatoire>.default\Cookies.txt

où USER est à remplacer par le nom de l'utilisateur du compte, et où directory.slt est une chaîne alphanumérique aléatoire qui se termine par .slt.

Sous Internet Explorer, on trouve les cookies sous le répertoire :

C:\Documents and Settings\USER\Cookies\

où USER est le nom de l'utilisateur.

JavaScript ne permet pas d'accéder directement au fichier du disque dur. Il sera nécessaire de passer par l'objet cookie du navigateur qui se chargera ensuite des opérations de lecture et écriture.

Un cookie est défini par un ensemble de données :

- Un nom qui permet d'identifier le cookie
- Une valeur associée au nom
- Une date d'expiration au-delà de laquelle l'ensemble de ces données sera supprimé définitivement
- Un domaine pour lequel le cookie est valide
- Un chemin qui définit le répertoire de la page pour lequel le cookie est lisible
- Un indicateur de sécurité qui rend le cookie inaccessible depuis des pages non sécurisées (qui n'utilisent pas le protocole HTTPS)

Le navigateur peut enregistrer plusieurs cookies par site dans la limite d'environ 3 Ko de données par domaine (ce qui correspond environ entre 15 et 20 cookies).

Les cookies ont une mauvaise réputation auprès des internautes. Les avantages apportés dans le confort de navigation et le niveau de sécurité élevé imposés par les navigateurs devraient rassurer les utilisateurs.
Les cookies sont entièrement gérés par le navigateur et configurés pas l'utilisateur. Firefox propose une interface détaillée pour la configuration et l'autorisation des cookies : menu Options du menu Outils, rubrique Vie privée, onglet Cookies.

La principale limite des cookies vient de leur caractère volatile. Il est impossible de s'assurer qu'un visiteur les acceptera. Il est également impossible de s'assurer qu'ils ne seront pas effacés volontairement ou automatiquement par des logiciels d'antivirus ou par les pare-feux.
Pour enregistrer plus de valeurs dans cette zone mémoire, il faut éviter de multiplier les noms de cookie. Pour chaque cookie, il faut enregistrer des données annexes (date expiration, domaine, chemin…). Il est donc préférable d'enregistrer une chaine de caractères complète dans un seul cookie en utilisant un traitement sur les chaines pour isoler les informations.

Par exemple :

Pour stocker le pseudo et la date de dernière connexion du visiteur, il est préférable de créer un cookie « infos » contenant « LePseudo ;27/07/2009 » plutôt que les informations séparées dans un cookie « pseudo » et un autre appelé « DerConnexion ».

Au niveau sécurité, votre site ne pourra pas lire les cookies posés par les autres sites consultés.
Le seul risque éventuel est le profilage des internautes. Les sites Web des sociétés multinationales sont consultés par des millions de personnes. En créant un réseau de sites Web, ces sociétés consolident entre elles les diverses données de consultation et d'achat. Ce profilage existe depuis des années avec les cartes de fidélité. Cela leur permet de proposer des offres et des messages publicitaires ciblés et plus efficaces.

VIII.2 MANIPULATION DES COOKIES AVEC JAVASCRIPT

JavaScript ne propose pas de méthodes natives pour la gestion des cookies. Il est donc nécessaire de redéfinir les fonctions de lecture et d'écriture à chaque fois que nous utilisons des cookies dans une page. Ces fonctions sont toujours identiques mais elles alourdissent les pages.

Ecrire un cookie

La fonction setCookie() enregistre un cookie sur le poste du navigateur.

function setcookie(name, value, expires, path, domain, secure)

Les 2 premiers paramètres (**name** et **value**) sont obligatoires.

Le paramètre **name** est le nom du cookie. Il faut éviter les caractères spéciaux.

Le paramètre **value** est la valeur du cookie. Pour gérer les caractères spéciaux dans cette valeur, on enregistre le cookie avec la valeur passée dans la fonction **escape()**. Cette fonction transforme les caractères spéciaux en codes qui seront retranscrits à la lecture par la fonction inverse **unescape()**.

Le paramètre **expires** est un objet Date. Il n'est pas obligatoire. S'il n'est pas défini, le cookie sera détruit à la fin de la session (intervalle de temps entre l'accès à un site et la fermeture du navigateur).

La paramètre **path** correspond au chemin du cookie. Pour autoriser le cookie sur tous les répertoires du site, il faut lui affecter la valeur « / ». S'il n'est pas défini, le cookie est configuré avec le chemin de la page.

Le paramètre **secure** vaut *true* si le cookie est réservé aux connexions sécurisées par HTTPS.

Code possible de la fonction setCookie :
Pour chacune des valeurs non obligatoires, on peut utiliser les opérateurs ternaires.

```
    function setCookie(name, value, expires, path, domain, secure)
{
            document.cookie = name + "=" + escape(value) +
                    ((expires==undefined) ? "" : ("; expires=" + expires.toGMTString() ))
    +
                    ((path==undefined) ? "" : ("; path=" + path)) +
                    ((domain==undefined) ? "" : ("; domain=" + domain)) +
                    ((secure==true) ? "; secure" : "");
}
```

Pour la date :

- Récupération de la date système : var expires = new Date() ;
- Date du jour + 1 mois : expires.setTime(expires.getTime() + 30*24*60*60*1000) ;
- Date du jour + 1 an : expires.setTime(expires.getTime() + 365*24*60*60*1000) ; La fonction setTime(n) permet d'assigner la date. n est un entier représentant le nombre de millisecondes depuis le 1er janvier 1970. La fonction getTime() permet de récupérer le nombre de millisecondes depuis le 1er janvier 1970.
- Nous pouvons assigner directement la date : var expires = new Date(2009,06,27) ; pour le 27 juillet 2009. le mois commence à 0.

Lire un cookie

La fonction getCookie() retourne la valeur du cookie à partir de son nom. Le principe est de manipuler la chaine des cookies contenue dans **doument.cookie**.

Code possible de la fonction getCookie :

```
function getCookie(name)
{
	if (document.cookie.length==0) { return null ;}
		// la chaine document.cookie ne contient aucun caractère
		// si la chaine n'est pas vide, elle contient des cookies sous la forme
		//	Nom1=Valeur1 ; Nom2=Valeur2 ;…. ; NomN=ValeurN
		//	chaque nom/valeur est séparé par un point-virgule et un espace
	var cookies = document.cookie.split('; ');
		// La fonction split() permet de scinder une chaîne de caractère et de retourner les résultats dans un tableau, grâce à une chaîne définie comme séparateur.
	// boucle pour lire tous les cookies
	for (var i=0 ; i<cookies.length ; i++)
		{
			var infos = cookies[i].split('=');
				// on isole le nom de la partie valeur
			if infos[0]==name {return unescape(infos[1]);}
		}
	return null;		// aucune correspondence n'est trouvée
}
```

Remarque : le contenu de document.cookie est différent de son affectation. Lors de l'affectation, toutes les propriétés sont enregistrées. Lors de la lecture seuls les couples nom/valeur sont retournés.

Si vous utilisez régulièrement les cookies, conservez ces 2 fonctions dans un fichier cookie.js et appelez ce fichier dans vos pages avec la syntaxe :

```
<script type= "text/javascript" src="cookie.js"></script>
```

Effacer un cookie

Il n'existe pas de fonction particulière pour effacer un cookie. Il suffit en fait de le définir avec une date d'expiration passée.

Exemple :
On a défini un cookie de nom : user et de valeur stagiaire avec une date d'expiration dans un an.
Pour effacer ce cookie, il suffit de redéfinir le cookie avec une date égale à la date du jour – 1.

```
var expiresDel = new Date() ;
expiresDel.setTime(expiresDel.getTime()-1) ;
 setcookie("user","",expiresDel);
```

S'assurer que les cookies sont activés

La propriété **cookieEnabled** de l'objet **navigator** vaut *true* si les cookies sont activés.
Cette propriété n'est pas fiable. Pour vérifier qu'un visiteur accepte les cookies est de tenter d'en créer un puis de le lire.

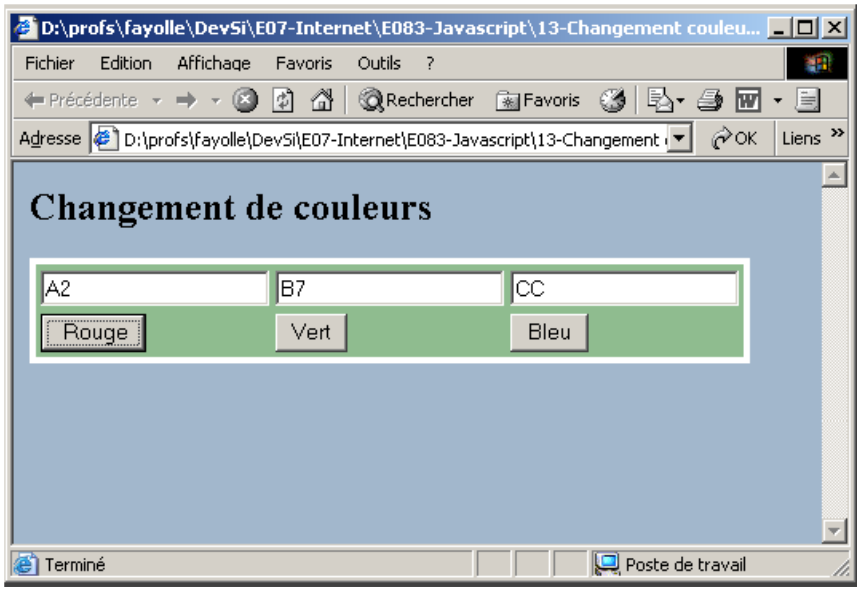
Remarque : un cookie stocké par JavaScript est accessible sur le serveur avec un langage tel que PHP. Un cookie écrit par un programme serveur est accessible par un code JavaScript. Cette possibilité permet de communiquer directement des informations entre PHP et JavaScript.

VIII.3 EXERCICE D'APPLICATION

Réalisez une page web qui demande à l'internaute son nom et son prénom lors de sa première connexion. Ces données seront enregistrées dans un cookie, et seront exploitées lors d'une connexion ultérieure (on affichera bonjour M.xxx).

IX GESTION DES COULEURS

Ecrire la page HTML suivante.



Cette page permet de modifier dynamiquement la couleur de fond de la page. Lorsque l'on clique sur les boutons rouge, vert ou bleu, la couleur de fond devient rouge, verte ou bleue et le contenu de chaque zone contient soit FF ou 00 selon le cas (code hexadécimal de la couleur). De plus, on peut saisir les codes RGB des couleurs. Une fois la saisie effectuée dans une des 3 zones, lorsque le focus quitte la zone, la couleur de fond prend la nouvelle valeur.

Points abordés :

Evénement onblur
Les fonctions
La propriété **bgcolor :** cette propriété permet de récupérer et de modifier la couleur de fond de la page HTML.