Initiation au Java Script

Plan

Introduction

A l'origine le Web n'était qu'une sorte de lieu de stockage. Il est aujourd'hui bien plus que cela : on s'y montre, s'y informe, s'y instruit et s'y amuse. Les outils du Web ont évolué en conséquence. Le HTML simple chaîne de balisage a été complété par différents langages de programmation.

Java Script est un langage de programmation très simple et constitue une excellente initiation à la programmation web. La création de programmes Java Script requiert très peu de connaissances.

Il est facile d'utiliser Java Script pour améliorer les pages Web crées en HTML. Les programmes en Java Script peuvent être composées d'une seule ligne. Cependant il est possible d'utiliser Java Script pour créer des applications complexes.

Partie I : Les bases de Java Script

I. Qu'est ce que Java Script.

Initialement, le Web ne permettait que la diffusion de documents textes ; la première version du HTML ne prévoyait pas la possibilité d'inclure des images dans une page. Désormais, un site Web peut comprendre toutes sortes de choses : images, sons, animations, vidéo, et autres éléments. Les langages de script pour le Web tels que Java Script sont un moyen simple et efficace de rendre interactive une page web.

1. Scripts et programmes

Un script en Java Script peut être constitué d'une seule ligne ; il peut également être une application complète. Dans tous les cas, le script s'exécutera dans un navigateur.

Certains langages de programmation sont compilés ou traduits en code machine pour être exécutés. Java Script, pour sa part est un langage interprété : le navigateur exécute chacune des lignes du programme au fur et à mesure.

L'avantage des langages interprétés est que l'écriture ou la modification d'un script est très facile. Il est aussi simple de changer le contenu d'un script Java Script que celui d'un document HTML.

2. Présentation de Java Script

Java Script a été développé par la société Netscape, Java Script est le premier langage de script à avoir été développé pour le Web ; il est aujourd'hui le plus répandu. Initialement Java Script s'appelait LiveScript ; il fut introduit avec Netscape que 2.0 en 1995.

3. Intégrer un script Java Script à une page Web

Pour ajouter un script à une page Web, vous ferez appel à la balise <script>. La balise <script> demande au navigateur d'interpréter la suite du document comme un script, et la balise </script> indique que la suite du document doit à nouveau être interprétée comme du HTML. Les instructions Java Script doivent toujours être placées à l'intérieur des balises <script>.

Avec la balise <script>, nous pouvons ajouter au document Web un petit script comme le montre l'exemple ci-dessous :

```
<HTML>
<head>
<TITLE> ma première page Java Script </TITLE>
</head>
<body>
<h1> ma première page Java Script </h1>
<p> bienvenue sur notre page web. Malheureusement elle en cours de construction, sa dernière modification remonte à la date suivante  : </p>
<script language="Java Script1.3">
document.write(document.lastModified);
</script>
</body>
</HTML>
```

L'instruction *document.write* (dont nous parlerons un peu plus bas) affiche un texte directement dans la page Web ; dans ce cas ce texte est la date de la dernière modification du document.

Dans cet exemple, le script a été placé dans le corps du document HTML. Un script peut être placé à quatre endroits différents :

- dans le corps de la page : dans ce cas, le résultat du script s'affiche au sein du document HTML lorsque la page est chargée.

- Dans l'entête de la page, à l'intérieur des balises Head : lorsqu'un script est placé dans l'entête, il n'est pas exécuté immédiatement, mais d'autres scripts peuvent s'y référer. L'entête est souvent employée pour les fonctions.
- A intérieur d'une balise HTML : on appelle cela un gestionnaire d'événements ; il permet au script d'interagir avec des éléments HTML.
- Dans un fichier séparé : vous pouvez stocker des scripts dans des fichiers comportant l'extension .js ; pour vous y référer vous indiquerez le nom de fichier dans la balise <script>.

4. Alternatives à Java Script

Java Script n'est pas le seul langage utilisé sur le Web, et dans certains cas, d'autres langages seront plus adaptés à la tache que vous souhaitez effectuer.

- VbScript : celui là est la réponse de Microsoft à Java Script. Tout comme Java Script, VbScript est un langage de scripts simple, et les scripts VbScript peuvent être inclus directement dans HTML. Le défaut principal du VbScript est qu'il n'est pris en charge que par Internet Explorer. Java Script pour sa part est pris en charge dans une large mesure, à la fois par Internet explorer et Netscape Navigator.
- CGI : ce n'est pas vraiment un langage, mais plutôt une spécification permettant à des programmes de s'exécuter sur des serveurs web. Les programmes CGI peuvent être écrits en différents langages : Perl, C, VB …., la principale différence entre les CGI et Java Script est qu'un programme CGI s'exécutera sur un serveur, alors que les applications Java Script sont exécutés au niveau du client. L'inconvénient majeur du CGI est que, dans la mesure où les informations font un aller/retour entre le client et le serveur, le temps de réponse peut être très long. En revanche les programmes CGI permettent d'effectuer des taches impossibles à réaliser avec Java Script : lire, écrire des informations sur un serveur.

II. Stockage et utilisation des valeurs

1. Les variables

Les variables sont des conteneurs nommés qui peuvent stocker des données : un nombre, une chaîne de caractères ou un objet. Pour le choix des noms des variables il faut prendre les mêmes précautions déjà vues avec C.

2. Affecter des valeurs à des variables

Pour affecter une valeur à une variable c'est le même principe qu'en langage C. (j'espère que vous avez une bonne mémoire);

Exemples d'affectation :

lignes = 40;

lignes = lignes -30;

exemple d'incrémentation / décrémentation:

lignes = lignes +1; lignes = lignes-1;

lignes++; lignes--;

3. Types de données en Java Script

Dans certains langages informatiques, il est nécessaire de déclarer le type de donnée qui sera stockée dans une variable : un nombre ou une chaîne de caractères ... . En Java Script, il n'est pas nécessaire de spécifier de type de données, sauf dans certaines situations exceptionnelles. Cependant il est important de savoir quels types de données les variables Java Script peuvent stocker :

- les nombres : 25 ; 3.14 …. Java Script permet l'utilisation à la fois des entiers et des nombres en virgule flottante.
- Les valeurs logiques ou booléennes : ce sont les deux valeurs true et false.
- Les chaînes de caractères : "ceci est un test" ; les chaînes sont constituées d'une série de caractères (au sens strict les chaînes sont des objets ; nous en reparlerons plus tard)
- La valeur nulle, représentée par le mot clé null il s'agit de la valeur d'une variable non définie.

Java Script enregistre le type de donnée stocké dans chacune des variables, mais ne vous empêche pas de changer de type de variable en cours de script,

Exp :

Total = 31 ;

Rien ne vous empêche d'affecter à la variable total une chaîne de caractère :

Total = "ceci est un test";

4. Conversion de données d'un type à un autre

Java Script convertit automatiquement les données d'un type à un autre chaque fois que c'est possible.

Il arrive qu'une variable contienne une chaîne, et qu'on souhaite convertir cette chaîne en valeur numérique ; on aura alors recours à l'une des fonctions suivantes :

parseInt() : qui convertit une chaîne en nombre entier.

parseFloat() : qui convertit une chaîne en nombre en virgule flottante.

Exp :

```
chaîne = "30 arbres coupés";
Nombre= parseInt(chaîne);
```

Apres l'exécution de ces deux instructions la variable nombre contiendra la valeur 30.

5. Variables locales et globales

Avant de parler des variables globales et locales nous allons voir de quelle façon on définit une fonction;

Les fonctions sont des ensembles d'instructions Java Script qui peuvent être exécutées comme une seule unité.

Exp : Définition d'une fonction :

```
function bonjour([Arg1],[arg2],…)
{
[utilisation des arguments]
alert("bonjour tout le monde");
}
```

Les fonctions peuvent également renvoyer des valeurs, qu'on peut exploiter dans d'autres places du script.

```
function Somme(a,b)
{
 c=a+b;
```

```
return c;
}
```

Généralement on place les définitions de fonctions dans la section <Head>du document. Dans la mesure où le contenu de la section <Head> est exécuté en premier, on s'assure ainsi que les fonctions sont définies avant d'être utilisées.

Exp:

```
<HTML>
<head>
<title> ma premiere page Java Script </title>
<script language="Java Script1.3">
function bonjour(arg)
{
alert("bonjour tout le monde" + arg);
}
function Somme(a,b)
{
 var  c=a+b;     // remarquer l'utilisation du mot clé var
return c;
}
</script>
</Head>
<body>
<h1> ma première page Java Script </h1>
<p> bienvenue sur notre page web. Malheureusement elle en cours de construction, sa dernière modification remonte à la date suivant  : </p>
<script language="Java Script1.3">
bonjour("ESTO");          /*  ceci est un commentaire : ici on fait appel à la
                              fonction qu'on a définit dans le <Head> */
var c=somme(1,2);   //  remarquer la deuxième utilisation du mot clé var
</script>
</body>
</HTML>
```

➔ Dans certains langages de programmation, il est nécessaire de déclarer une variable avant de l'employer. Java Script comprend le mot clé var, qui peut être

utilisé pour déclarer une variable. Cependant dans de nombreux cas vous pouvez omettre ce mot clé ; la variable sera automatiquement déclarée dés que vous lui affecterez une valeur.

La portée d'une variable est la partie du script dans laquelle elle peut être utilisée. Il existe deux types de variables :

- les variables globales : qui peuvent être utilisées dans tout le script, et même dans d'autres scripts du même document HTML.
- Les variables locales : qui ne peuvent être employées qu'à intérieur de la fonction où elles ont été crées.

Pour créer une variable globale, il suffit de la déclarer dans le script principal à l'extérieur de toute fonction. Vous pouvez utiliser le mot clé var pour la déclarer :

var étudiants = 20 ; ⇔ étudiants = 20;

Note : avant de prendre l'habitude d'omettre le mot clé var, assurer vous de savoir exactement quand il est obligatoire. En fait, il est conseillé d'avoir toujours recours au mot clé var ; il permet d'éviter des erreurs et rend votre script plus lisible sans jamais causer de problèmes.

Une variable locale fait partie d'une fonction donnée ; toute variable déclarée ou utilisée pour la première fois dans une fonction est une variable locale. Ainsi, les variables faisant partie de la liste des paramètres d'une fonction sont des variables locales.

Pour être sûr de créer une variable locale à l'intérieur d'une fonction, utilisez le mot clé var. vous forcez ainsi Java Script à créer une variable locale, même s'il existe une variable globale qui porte le même nom (l'exemple ci-dessus explique très bien cette notion de variable locale et globale).

Partie II : Les commandes essentielles

I. Chaînes et tableaux

1. L'objet String

les chaînes permettent de stocker des ensembles de caractères, Java Script stocke les chaînes en tant qu'objets *String*, le nom en lui même n'est pas très important, mais ce sont les techniques de manipulation des chaînes de caractères qui font appel aux méthodes de l'objet String qui présentent un intérêt majeur.

Exp : les deux instructions suivantes créent une chaîne de caractère de deux manières différentes :

```
test  ="'ceci est un test";
test = new String("ceci est un test");
```

Pour ce qui est de l'affectation ce sont les mêmes règles qu'on a vu qui s'appliquent, rien de spécial à ajouter.

Exp : affectation de valeurs aux chaînes

```
<HTML>
<Head>
<title> les chaînes en Java Script </title>
</Head>
<body>
<h1> les chaînes en Java Script </h1>
<script language="Java Script1.3">
test_1 = "ceci est un test";
test_2 = " ne vous inquiétez pas";
chaine=test_1 + test_2;          // le signe + entre deux chaînes réalise la concaténation
document.write(test_1);
document.write("<p>"+test_2 + "</p>");
document.write("<p>"+chaine + "</p>");
</script>
</body>
</HTML>
```

Comme l'avoir déjà évoqué, ce sont les méthodes de l'objet String qui représentent un intérêt ; voici quelques unes :

- test = "ceci est un test";

➔ document.write(test.length); /* la propriété length de l'objet String renvoie la longueur de la chaîne en question*/

➔ test = test.toLowerCase(); /* la méthode toLowerCase() retourne une chaîne en minuscules */

➔ test = test.toUpperCase(); /* la méthode toUpperCase() retourne une chaîne en majuscules */

➔ document.write(test.substring(3,6)); : la méthode substring() renvoie une chaîne extraite d'une autre chaîne en fonction des deux valeurs d'indice que vous indiquerez comme paramètres. D'après l'exemple ci-dessus nous aurons les caractères 4,5 et 6 !! pourquoi ?

et bien pour les raisons suivantes :

- le premier caractère de la chaîne a une valeur d'indice de 0. de ce fait, le quatrième caractère à l'indice de 3.
- Le second paramètre est exclusif, autrement dit, si le second numéro est 6, les caractères de la chaîne sont pris en compte jusqu'à celui dont la valeur de l'indice est 5, ie : le sixième caractère

Exp:

Alpha = abcdefghijklmnopqrstuvwxyz;

Alpha.substring(0,4) ➔➔ abcd

Alpha.substring(10,12) ➔➔ kl

Alpha.substring(6,7) ➔➔ g

Alpha.substring(4,4) ➔➔ renvoie la valeur null : une chaîne vide.

En particulier :

Alpha.charAt(0) ➔➔ a / la propriété charAt() réalise l'extraction Alpha.charAt(12) ➔➔ m d'un seul caractère, c'est un cas particulier Alpha.charAt(25) ➔➔ z de substring */*

➔ position = test.indexOf("ceci"); : on peut rechercher une chaîne de caractères au sein d'une autre chaîne et ceci à l'aide de la propriété indexOf, qui dans ce cas commence la recherche depuis le début de la chaîne mère.

➔ Position = test.indexOf("ceci",3); : le deuxième argument pour préciser d'où commencer la recherche.

La valeur renvoyée est stockée dans la variable position (elle contient la valeur d'indice qui correspond au premier numéro d'indice de substring).

➔ position = test.lastIndexOf("ceci",[valeur]); : de même que indexOf, mais celle là commence depuis la fin de la chaîne.

2. Les tableaux numériques

Un tableau est une série d'éléments de données qu'il est possible de manipuler comme une seule unité. Les tableaux peuvent contenir des chaînes; des nombres, des objets ou d'autres types de données.

- Créer un tableau

Contrairement à la plupart des autres types de variables Java Script, les tableaux doivent être déclarés avant de pouvoir être utilisés.

Exp :

Scores = new Array(30);

Pour affecter des valeurs au tableau, utilisez des crochets et des numéros d'indice.

Exp : tableau numérique

Scores[0]=39;

Scores[24]=2;

Scores[12]=3;

Scores[11]=53;

Tout comme les chaînes, les tableaux ont une propriété *length.* Celle-ci correspond au nombre d'éléments dont est constitué le tableau.

Exp :

document.write(scores.length);

- Accéder aux éléments d'un tableau

Pour lire le contenu d'un tableau, utilisez la même syntaxe que lors de l'affectation des valeurs. Ainsi, l'instruction qui suit affiche les valeurs des quatre éléments définis ci-dessus du tableau scores :

Affichage_scores="Scores : " + Scores[0] + " " + Scores[24] + " " + Scores[12];

document.write(affichage_scores);

Exp : tableau de chaînes

Noms = new Array(30); // c'est le même type de déclaration que celui ci-dessus

Pour l'affectation c'est la même chose, sauf qu'ici on affectera au tableau des chaînes de caractères.

Noms[0]="amine";

Noms[15]="Mohamed";

Noms[14]="ghizlane";

Noms[25]="narjiss";

Dans cet exemple le tableau contient des chaînes, donc en fait chaque emplacement du tableau est une chaine de caractères, donc on peut lui appliquer toutes les propriétés de l'objet String.

Document.write(Noms[14].substring(2,4));

- Scinder une chaîne de caractère

Java Script comprend une méthode nommée *split* qui divise une chaine en plusieurs parties.

Exp :

Chaine_a_diviser = " pourquoi on aime la vie";

Parties_de_la_chaine = chaine_a_diviser.split(" ");

La méthode *split* divise la chaine au niveau de chacun de caractères d'espacement. Les parties de chaînes résultantes sont stockées dans le tableau Parties_de_la_chaine. Donc :

Parties_de_la_chaine[0]="pourquoi";

Parties_de_la_chaine[1]="on";

Parties_de_la_chaine[2]="aime";

Parties_de_la_chaine[3]="la";

Parties_de_la_chaine[4]="vie";

Java Script propose une autre méthode qui réalise l'inverse, c'est la méthode *join*. Donc :

Chaine_produite= Parties_de_la_chaine.join(" ");

document.write(chaine_produite);

Note : il faut faire attention, ce sont les paramètres des deux méthodes split et join qui précisent avec quel caractère on va réaliser soit la division ou bien la jointure.

- Tri de tableau

Pour trier le contenu d'un tableau, nous utiliserons la méthode *sort*. Cette méthode renvoie une version triée de votre tableau du petit vers le plus grand dans le cas d'un tableau numérique et par ordre alphabétique dans le cas d'un tableau de chaînes.

Exp:

```
Noms=Noms.sort();
```

II. Test et comparaison de valeurs

Pour ce qui est des tests, c'est pratiquement le même principe qu'en langage C, c'est avec l'instruction *if, [else]* qu'on réalise un test avec Java Script.

Exp :

```
a=1;
if(a==2) document.write("la valeur de a est : 2");
else document.write("Erreur !!");
```

bien sûr, quand on a plusieurs instructions il faut ajouter des accolades {}.

Les opérateurs avec lesquels on réalise des tests sont comme suit :

==	:	*égale*
!=	:	*différent de*
<	:	*inférieur à*
>	:	*supérieur à*
<=	:	*inférieur ou égale*
>=	:	*supérieur ou égale*

➔ Conditions et opérateurs logiques : il arrive des fois qu'on doit tester une variable avec plusieurs variables à la fois, Java Script propose pour ce type de tache des opérateurs *logiques ou booléens :*

||	:	*le OU logique;*
&&	:	*le ET logique;*

Exp :

```
if(a==1 || a==2) alert("c correct");
if(a<5 &&  a!=4) alert("a est inférieur ou égale  3");
```

➔ conditions multiples avec *switch* : il vous arrivera couramment d'avoir recours à plusieurs instructions pour tester différents conditions sur la même variable.

Exp :

```
if(a==1) document.write("a=1");
if(a==21) document.write("a=21");
if(a==3) document.write("a=3");
if(a==4) document.write("a=4");
```

Il faut savoir que les blocs des *if* peuvent être très longs, dans ce cas vaut mieux utiliser l'instruction *switch*.

Exp :

```
switch(a)
{
case 1 : document.write("a=1");
        break;
case 2 : document.write("a=1");
        break;
case 3 : document.write("a=1");
        break;
case 4 : document.write("a=1");
        break;
default : document.write("c'est une chaine de caractère");
}
```

C'est le même principe qu'en langage C; les instructions *break* servent à quitter le bloc switch ; autrement dit continuer après l'accolade fermante.

III. Les boucles

1. Les boucles for ; while ; do-while.

Pour ces boucles c'est le même principe qu'en langage C qui s'applique ici avec Java Script.

Exp :

```
➔ for(i=0;i<10;i++)
{
instructions …….
```

```
}
```

➔ *for(i=0,j=100;i<12,j>46;i++,j--)*

```
{
instructions .. .. ..
}
i=0;
```

➔ *while(i<150)*

```
{
instructions ....
i++;
}
i=0;
```

➔ *do*

```
{
 instructions ....
 i++;
}while(i<150);
```

Comme vous le savez tous, la seule différence qui existe entre while et do-while c'est que la deuxième s'exécute au moins une fois.

De n'importe quelle boucle on peut sortir à l'aide de l'instruction *break*.

- Les boucles for…in

  Ce type de boucles n'est pas aussi polyvalente comme les autres types de boucles vues juste ci-dessus, mais elle permet d'effectuer des opérations sur les propriétés d'un objet.

  Exp :

```
for(i in navigator)
{
 document.write("la propriété est : "+i);
 document.write("la valeur est :"+ navigator[i]);
}
```

Partie III. Fonctions avancées de Java Script

I. les objets prédéfinis de Java Script

Cette troisième partie traitera essentiellement deux fonctions de Java Script : les objets et les gestionnaires d'événements.

Nous allons commencer par le fonctionnement de base des objets en Java Script ainsi que des détails de l'utilisation des objets *Math* et *Date*.

1. Qu'est ce qu'un objet

Les objets permettent de combiner différents types de données (propriétés) et de fonctions permettant d'agir sur ces données (méthodes) en un seul élément facile à manipuler.

2. Créer un objet

Pour chaque objet, on utilise une fonction particulière nommée *constructeur*, qui permet de créer un nouvel objet. Ainsi, Java Script dispose d'une fonction prédéfinie, *String*, qui permet de créer un objet *String*. Ce qui fait que vous pouvez créer une variable de chaîne de la manière suivante :

Nom = new String("ESTO");

Le mot clé *new* demande à Java Script de créer un nouvel objet – ou en terme technique, une nouvelle *instance* de l'objet *String*. Cette instance particulière aura la valeur de *ESTO* et sera stockée dans la variable *Nom*.

3. Propriétés et valeurs des objets

Chaque objet peut avoir une ou plusieurs propriétés ou attributs. La propriété d'un objet est l'équivalent d'une variable qui serait contenue dans un objet. Il est possible d'affecter une valeur à toutes les propriétés d'un objet ; il peut s'agir de n'importe quel type de données, comme pour les variables.

Nous avons déjà fait appel à diverses propriétés, par exemple la propriété *length* des chaînes et des tableaux. Pour vous référer à une propriété indiquez le nom de l'objet, suivi d'un point et du nom de la propriété. Par exemple : *Nom.length* : la longueur de la chaîne *Nom*.

4. Méthodes

Les méthodes sont des fonctions stockées en tant que propriétés d'objets. On a déjà utilisé des méthodes, telles que : *toUpperCase(), toLowerCase(),….* Et on se réfère aux méthodes des objets de la même manière que les propriétés vues ci-dessus, Exp : *Nom.toUpperCase().*

Il se peut qu'une méthode d'un objet renvoie une valeur, dans ce cas on procède de la même manière que les autres fonctions,

Exp : *résultat=Math.round(nombre);*

5. Le mot clé *with*

Le mot clé *with* permet de simplifier l'ecriture des programmes en Java Script – ou du moins la quantité de code à saisir. Le mot cle *with* permet de specifier un objet : il est suivi d'un bloc d'instructions placees entre accolades. Pour chacune des instructions du bloc, les proprietes mentionnees sans que l'objet correspondant soit indiqué se réfèrent à l'objet indiqué après *with*.

Exp :

```
prénom = "Foulan";
with(prénom)
{
document.write("<p>la longueur est : " + length);
ch=toUpperCase();
document.write("<p>la chaîne est " + ch);
}
```

dans ce type d'exemple, l'intérêt de *with* n'est pas évident mais lorsqu'on cherche à accéder à un même objet tout au long d'une longue procédure, *with* peut faire gagner du temps.

6. L'objet Math

Math est un objet prédéfini de Java Script qui comprend de nombreuses constantes et fonctions. Les proprietes de l'objet Math sont des constantes mathématiques, ses méthodes des fonctions mathématiques.

- Arrondis : l'objet Math comprend trois méthodes très utiles qui permettent d'arrondir des valeurs décimales de différentes manières :

  ➔ *Math.ceil()* : arrondit un nombre à l'entier supérieur le plus proche.

  ➔ *Math.floor()* : arrondit un nombre en entier.

  ➔ *Math.round()* : arrondit un nombre à l'entier le plus proche.

- Générer des nombres aléatoires : L'une des méthodes les plus employées est la méthode *random* qui génère un nombre aléatoire entre 0 et 1. en général on cherche un nombre aléatoire entre 1 et x, pour cela on peut écrire une fonction spécifique.

Exp : générer un nombre aléatoire entre 1 et nb

```
Function rand(nb)
{
  x=Math.random();
  x=x*nb;
  y=Math.floor(x) + 1;
  return y;
}
```

7. Les dates

Date est un objet prédéfini de Java Script qui permet de manipuler facilement dates et heures.

- Créer un objet Date : pour créer un objet Date utilisez le mot cle new. Vous pouvez specifier la date à stocker dans l'objet lors de sa création. Exp :

  Anniversaire = new Date();

  Anniversaire = new Date("September 20 2002 08:00:00");

  Anniversaire = new Date(9, 20, 2002);

  Si vous ne précisez aucun paramètre, comme dans le premier exemple,c'est la date courante qui est stockée dans l'objet;

  Exp : Calcul de temps restant pour la coupe du monde 2010

```
<html>
<head>
<title>Temps restant pour la coupe du monde 2010</title>
</head>
<body>
<h1>Temps restant pour la coupe du monde 2010</h1>
<script language="Java Script1.3">
```

```
date_coupe = new Date("June 17 2010 00:00:00");
date_courante = new Date();
diff = date_coupe - date_courante; /* temps en millisecondes (c'est
                                   a vous de faire le
                                   changement) */
document.write(" ce qui reste de temps est :  " + diff);
</script>
</body>
</html>
```

- Modifier un objet Date : les méthodes *set* permettent définir les valeurs des objets Date.

  ➔ *setDate()* : définit le jour du mois ;

  ➔ *setMonth()* : définit le mois, les mois sont numérotés de 0 à 11 ;

  ➔ *setFullYear()* : définit l'année en quatre chiffres ;

  ➔ *setHours(), setMinutes() et setSecondes()* définit l'heure ;

  Exp :

  Anniversaire= new Date(); // Anniversaire contient la date courante

  Anniversaire.setDate(0); // les jours sont numérotés de 0 à 30

  Anniversaire.setMonth(1); // les mois sont numérotés de 0 à 11

  Anniverasaire.setFullYear(2003);

- Lire les valeurs des objets Date : les méthodes *get* permettent de lire les valeurs des objets Date. Le recours à ces méthodes est obligatoire dans le mesure où les valeurs de ces objets ne sont pas disponibles en tant que proprietes.

  ➔ *x=Anniversaire.getDate()* : permet de lire le jour du mois ; // x contient la valeur indiquant le jour du mois de la date Anniversaire.

  ➔ *x=Anniversaire.getMonth()* : permet de lire le mois ;

  ➔ *x=Anniversaire.getFullYear()* : permet de lire l'année, avec quatre chiffres ;

  ➔ *getHours(), getMinutes() et getSecondes()* permettent de lire les heures, les minutes et les secondes.

- Fuseaux horaires : Java Script comprend plusieurs fonctions permettant de gérer les fuseaux horaires et les heures locales.

  ➔ *getTimeZoneOffset()* : renvoie le décalage du fuseau horaire local par rapport à l'heure GMT. En l'occurrence, le fuseau horaire local est celui du navigateur de l'utilisateur.

  ➔ *getGMTString()* : convertit en texte la valeur de l'objet Date en fonction de l'heure de GMT.

  ➔ *toLocalString()* : convertit en texte la valeur de l'objet Date en fonction de l'heure locale.

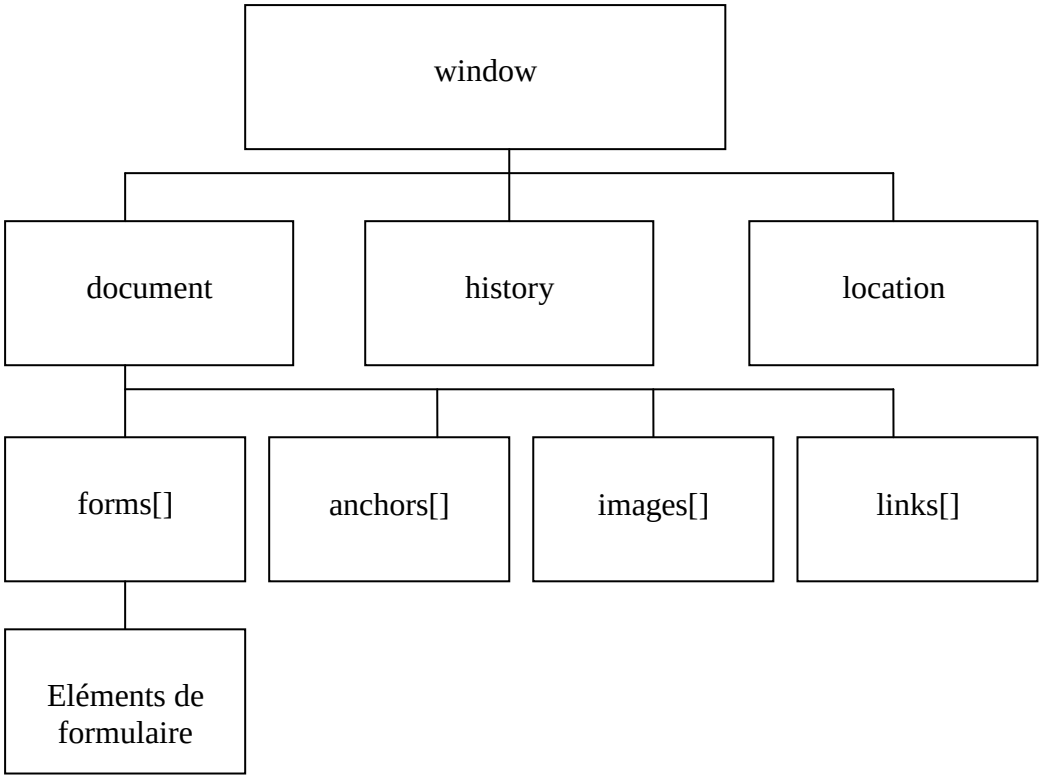
II. Les objets du navigateur

Les objets que nous utiliserons le plus fréquemment sont les objets de navigateur, qui permettent aux scripts de manipuler des pages Web, des fenêtres et des documents.

1. Le fonctionnement des objets de navigateur

L'un des avantages de Java Script est qu'il permet de manipuler directement le navigateur Web. Vous pouvez utiliser un script Java Script pour charger une nouvelle page dans le navigateur, manipuler des parties de la fenêtre et même ouvrir de nouvelles fenêtres.

Pour effectuer ces manipulations, Java Script fait appel à des objets de navigateur. Chacun de ces objets représente une partie d'une page Web. Les objets du navigateur ont des propriétés qui décrivent la page Web ou le document et des méthodes qui permettent de les modifier.



D'après le schéma ci-dessus, on peut déduire que les objets de navigateur sont structurés en une hiérarchie d'objets parents et filles. Lorsqu'on se réfère à un objet, on utilise l'objet parent suivi du ou des noms des objets filles, les objets étant séparés par des points. L'objet *window* se situe au sommet de la hiérarchie des objets de navigateur.

2. Manipulation des documents Web

L'objet *document* représente un document Web. Les documents Web sont affichés dans la fenêtre du navigateur ; il n'est donc pas surprenant que les objets *document* soient des objets filles de *window*.

Dans la mesure où l'objet *window* représente toujours la fenêtre en cours (celle qui contient le script), vous pouvez utiliser *window.document* pour vous référer au document en cours. Vous pouvez également vous référer uniquement à *document* : vous désignez ainsi automatiquement le document de la fenêtre en cours. Si plusieurs fenêtres ou cadres sont ouverts, il existera plusieurs objets *window* et un objet *document* pour chacun d'eux.

- Informations sur le document : différentes propriétés d'un objet *document* comprennent informations générales sur le document courant :

➔ La propriété *URL* indique l'URL du document, sous la forme d'une chaîne de texte. Vous ne pouvez modifier cette propriété.

➔ La propriété *title* représente le titre de page, tel qu'il est défini dans la balise HTML <title>

➔ La propriété *referrer* est l'URL de la page affichée ans le navigateur avant la page actuelle ; il s'agit généralement d'une page avec un lien vers la page en cours.

➔ La propriété *lastModified* est la date de dernière modification de la page. Cette information est envoyée par le serveur en même temps que la page.

➔ Vous connaissez déjà la méthode *write.*

- Liens : l'objet link est un objet fille de document. Un document peut contenir plusieurs objets link, chacun d'eux contenant les informations quant à l'URL correspondante.

  ➔ Vous pouvez accéder à des objets link à l'aide du tableau links[]. Chaque membre de ce tableau est l'un des objets link de la page courante. Une propriété du tableau document.links.length indique le nombre de liens dans la page.

  ➔ Chaque objet link (ou membre du tableau links) dispose d'une liste de propriétés définissant l'URL.

- Les ancres : ce sont des positions à l'intérieur d'un document html auxquelles vous donnez un nom; pour définir une ancre, on utilise la balise <a>, donc : *<a NAME="ancre1">* pour lui donner un nom et pour créer ensuite un lien vers cette ancre, on utilise toujours la même balise : *<a href="#ancre1">.* Les objets anchor sont également des objets filles de l'objet document. Chaque objet anchor représente une ancre du document en cours, ie : une position dans le document à laquelle on peut accéder directement. De même que pour les liens, il est possible de manipuler les ancres à l'aide d'un tableau anchors. Chaque élément de ce tableau est un objet anchor. La propriété document.anchors.length indique le nombre d'éléments du tableau anchors.

3. Accès à l'historique du navigateur

L'objet history est une propriété fille de l'objet window, il contient des informations sur les URL qui ont été visitées avant et après la page courante, ainsi que des méthodes pour y accéder :

L'objet history dispose de quatre propriétés :

- *history.length* : qui indique la longueur de l'historique, ie : le nombre de pages différentes visitées par l'utilisateur.
- *history.current* : qui contient la valeur courante de l'historique, autrement dit l'URL de la page que l'utilisateur est en train de visualiser.
- *history.next* : cette propriété ne contient pas de valeur que si l'utilisateur à déjà cliqué sur Précédente.
- *history.previous* : qui est la valeur de l'élément précèdent de l'historique, ie : la page où sera envoyé l'utilisateur s'il clique sur Précédente.

L'objet history dispose de trois méthodes :

- *history.go* : qui ouvre une URL de l'historique. Pour l'utiliser, indiquez un nombre positif ou négatif comme paramètre ; Exp : *history.go(-2)* : équivaut à cliquer deux fois sur Précédente.
- *history.back()* : qui affiche la page précédente.
- *history.forward()* : qui affiche la page suivante.

4. L'objet location

L'objet location est lui même fille de l'objet window. Il stocke des informations concernant l'URL de la page affichée dans la fenêtre et permet de charger de nouvelles pages. Ainsi, pour charger une URL dans la page courante, on utilisera ce qui suit :

window.location.href = www.caramail.com

➔➔➔ *Note : Même si la propriété document.URL, dont il a été question plus haut, contient la même URL que la propriété window.location.href , vous ne pouvez pas l'utiliser pour charger une nouvelle page; utilisez la deuxième, c'est mieux !!!!*

L'objet location dispose de deux méthodes :

➔ *location.reload* : qui permet de recharger le document courant et qui équivaut à un clic sur actualiser.

➔ *location.replace* : qui remplace la page courante par une nouvelle page. Cette méthode est similaire à l'emploi de location.href pour charger une nouvelle page, à la différence qu'avec location.replace, l'historique du navigateur n'est pas modifié. Autrement dit, l'utilisateur ne pourra pas revenir à la page precedemment affichée à l'aide de Précédente.

III. Création d'objets personnalisés

A ce niveau, nous allons voir comment créer des objets personnalisés !!!!

1. Définir un objet

La première étape de la création d'un objet consiste à lui donner un nom et à nommer ses propriétés. Puis on doit créer une fonction *(ce qu'on avait appelé : Constructeur)* pour pouvoir instancier notre objet.

Exp :

```
function personne(nom,prenom,pro,dom)
{
this.nom     = nom;
this.prenom = prenom;
this.telpro    = pro;
this.teldom = dom;
}
```

Le constructeur est une fonction simple acceptant des paramètres pour initialiser le nouvel objet, puis les affectant aux propriétés correspondantes. La fonction s'appelant *personne*, l'objet lui aussi portera le nom de personne.

Donc : *P = new personne("zhong","ming","557744","556699");*

Note 1 : notez l'utilisation du mot clé this, il permet de se référer à l'objet courant ➔ ie : celui crée par la fonction.

Note 2 : il faut très bien remarquer que la définition du constructeur ne contient pas de méthodes.

2. Définir une méthode pour un objet

On va créer à ce moment une méthode pour manipuler l'objet *personne*. On va créer une méthode qui sert à afficher les propriétés de l'objet *personne*.

Exp :

```
function Afficher()
{
 ligne1 = "<b> Nom : </b>" + this.nom + "<br>";
 ligne2 = "<b> Prenom : </b>" + this.prenom + "<br>";
 ligne3 = "<b> <pre> Le téléphone professionnel : </b>" + this.telpro + "<br>";
 ligne4 = "<b> <pre> Le téléphone de domicile : </b>" + this.teldom + "<br>";
 document.write(ligne1,ligne2,ligne3,ligne4);
}
```

A ce niveau, la fonction d'affichage n'est pas encore reconnue par l'objet *personne*, il faut qu'elle en fasse partie de la définition de l'objet *personne*.

```
function personne(par_nom,par_prenom,par_pro, par_dom)
{
this.nom      = par_nom;
this.prenom = par_prenom;
this.telpro    = par_pro;
this.teldom = par_dom;
this.Afficher = Afficher;
}
```

3. Création d'instances d'objet

Pour créer une instance de l'objet défini, on utilise comme on déjà vu le mot clé *new*. Donc :

```
M = new personne("zhong","ming","557744","556699");
S = new personne("pepoloni","stephano","557744","556699");
```

On peut créer un instance d'objet vide, puis lui affecter des valeurs, comme suit :

```
A = new personne();
A.nom ="maziane";
A.prenom="Abdel";
....
```

Pour afficher les propriétés de ces objets, on peut utiliser la méthode Afficher de chaque objet, Donc :

➔ *M.Afficher();*

➔ *S.Afficher();*

➔ *A.Afficher();*

4. Personnalisation d'objets prédéfinis

Il est possible d'étendre les fonctions des objets prédéfinis de Java Script. Pour ajouter des propriétés ou des méthodes à un objet *String*, on utilise le mot clé *prototype* qui permet de modifier la définition d'un objet à l'extérieur de son constructeur.

Exp : on ajoutera une méthode titre à l'objet String qui convertit une chaîne de caractères en titre html.

```
<HTML>
<Head>
<title> Personnalisation d'objets prédéfinis </title>
<script language="Java Script1.3">
function ajout_titre(niveau)
{
balise = "H" + niveau;
debut = "<" + balise + ">";
fin    = "</" + balise + ">";
contenu = this.toString();
resultat  = debut + contenu + fin;
return resultat;
}
function length_nouvelle(c)
{
tab_temp_1 = new Array();
tab_temp_1=this.split("");
x=tab_temp_1.length
for(i=0;i<tab_temp_1.length;i++)
{
if(tab_temp_1[i] == c) x=x-1;
}
return x;
}
</script>
```

```
</Head>
<body>
<h1> Personnalisation d'objets prédéfinis  </h1>
<script language="Java Script1.3">
String.prototype.titre = ajout_titre;
String.prototype.nv_length = length_nouvelle;
texte = "je suis un titre";
document.write("<p>" + texte.titre(1));
document.write("<p>" + texte.titre(2));
document.write("<p>" + texte.titre(3));
/*******************************/
tab=texte.split("");
document.write("<p>" + texte.length);
document.write("<p>" + tab.length + "<p>");
document.write("<p>" + texte.nv_length("s") + "<p>");
</script>
</body>
</HTML>
```

IV. Gestionnaires d'événements

Les scripts qui font appel à des gestionnaires d'événements interagissent directement avec l'utilisateur au lieu de s'exécuter de manière ordonnée.

1. Qu'est ce qu'un gestionnaire d'événement

Les programmes en Java Script ne s'exécutent pas obligatoirement d'une manière ordonnée : ils peuvent aussi détecter des événements et y réagir. Ces événements peuvent être un clic de souris ou un déplacement de pointeur, la fin de chargement d'une image ou d'une page Web….

2. Créer un gestionnaire d'événement

Pour définir un gestionnaire d'événement, on l'ajoute simplement en tant qu'attribut à la balise HTML correspondante.

Exp :

```
<A href=www.caramail.com onMouseOver="window.alert('vous êtes passé au dessus du lien menant vers caramail ');"> Cliquer ici </A>
/* une boite de dialogue s'affiche lorsque la souris passe au-dessus du lien */
```

vous pouvez utiliser directement des instructions Java Script en tant que gestionnaires d'événements, mais on peut aussi faire appel à une fonction.

<A href=www.caramail.com ***onMouseOver="action();"*** *> Cliquer ici </A>*

/ il faut définir au préalable la fonction action() */*

3. Utilisation de l'objet event

L'objet *event* est un objet spécial envoyé à un gestionnaire d'événement chaque fois qu'un événement se produit. Les propriétés de l'objet *event* fournissent des informations supplémentaires sur l'événement qui s'est produit.

- *type* : c'est le type d'événement qui s'est produit ; Exp : mouseover.
- *target* : c'est l'objet cible de l'événement ; Exp : lien, bouton, ….
- *which* : est une valeur numérique indiquant quel bouton de souris a été cliqué (pour les événements de souris) ou quelle touche a été enfoncée (pour les événements de clavier).
- *modifiers* : est la liste des touches de modification (Alt, Ctrl, Maj…) enfoncées lors d'un événement de souris ou de clavier.
- *pageX et pageY* : sont les positions X et Y de la souris au moment où l'événement s'est produit.
- ….

4. Emploi des événements de souris

Java Script pourra détecter les déplacements du pointeur de souris et le fait que l'un ou l'autre de ses boutons soit cliqué, enfoncé ou relâché.

- *onMouseOver* : ce gestionnaire est appelé lorsque le pointeur de la souris se trouve au dessus d'un lien, d'une image ou d'un autre objet.
- *onMouseOut* : celui la fonctionne de manière opposée au premier → il est appelé lorsque le pointeur de la souris quitte l'objet.
- *onClick* : celui la est appelé lorsqu'on a cliqué une seule fois sur un des boutons de la souris.
- *onMouseDown* : lorsque l'utilisateur enfonce le bouton de la souris.
- *onMouseUp* : lorsque l'utilisateur relâche le bouton de la souris.

Exp :

➔ *<A href=www.caramail.com* ***onClick="window.alert('vous avez cliqué sur lien menant vers caramail ');"*** *> Cliquer ici </A>*

➔ *<A href=www.caramail.com* ***onClick="return(window.comfirm('vous avez cliqué sur lien menant vers caramail , voulez vous sortir de cette page ?'));"*** *> Cliquer ici </A>*

- pour savoir lequel des boutons de la souris a été enfoncé, vous pouvez utiliser la propriété *which* de l'objet *event*. Cette propriété contient la valeur 1 si c'est le bouton gauche et 3 si c'est le bouton droit (bien sur dans le cas où un des événements de la souris est appelé).

5. L'événement *onload*

Cet événement se produit lorsque le téléchargement de la page en cours est fini avec toutes ses images. Pour définir ce gestionnaire, on l'appelle dans la balise *body*.

Exp :

<body onload="alert('chargement de page terminé');">

Partie IV : Manipulation d'éléments de page Web

I. Fenêtres et cadres

A ce niveau, nous allons voir en détail comment gérer différentes fenêtres et différents cadres (*frames*) à l'aide de Java Script.

1. L'objet *window*

Nous avons vu que l'objet window est au sommet de la hiérarchie des objets. Les objets history, document et location sont tous des objets filles de l'objet window.

L'objet window se réfère toujours à la fenêtre courante ie : celle qui contient le script. Plusieurs fenêtres peuvent être ouvertes simultanément, et on utilisera leurs noms respectifs pour vous référer à chacune d'elles.

- Créer une nouvelle fenêtre

L'une des utilisations les plus intéressantes de l'objet window est la création de nouvelles fenêtres. On peut ainsi afficher un nouveau document sans effacer l'ancien.

Pour créer une nouvelle fenêtre, on utilise la méthode *window.open().* Voici une instruction type pour permettre l'ouverture d'une nouvelle fenêtre :

Objetfenetre = window.open("URL","Nom de la fenêtre","Liste des caractéristiques");

➔ *Objetfenetre* : cette variable permet de stocker le nouvel objet *window*. Pour accéder aux propriétés et aux méthodes de cet objet, on utilisera le nom de cette variable.

➔ *URL* : en l'occurrence celle du document qui sera affiché dans la nouvelle fenêtre. Si ce paramètre n'est pas mentionné, aucune page ne sera affichée.

➔ *Nom de la fenêtre* : ce paramètre indique le nom de la fenêtre. Ce nom sera affecté à la propriété *name* de l'objet *window*.

➔ Le troisième paramètre est une liste de caractéristiques optionnelles, séparées par des virgules. Il permet de personnaliser la nouvelle fenêtre en choisissant d'y inclure la barre d'outils, la ligne d'état et d'autres éléments.

Détails : les caractéristiques du troisième paramètre de la méthode window.open() sont, entre autres, *width* et *height* pour déterminer la taille de la fenêtre ; ainsi qu'une série de caractéristiques pouvant être activées (1 ou yes) ou désactivées (0 ou no) : *toolbar, directories, status, menubar, scrollbars* et *resizable*. Si vous omettez l'une de ces caractéristiques, c'est la valeur par défaut qui sera utilisée.

Exp :

```
Petitefenetre=window.open("","petite","width=100,height=120,toolbar=0,status=0
")
```

- Ouverture et fermeture de fenêtres

Java Script vous permet également de fermer une fenêtre que vous avez ouverte. La méthode *window.close()* permet de fermer les fenêtres du navigateur. On utilisera cette méthode pour fermer les fenêtres que nous avons crée.

Exp :

```
Petitefenetre.close();     // cette instruction permet de fermer la fenêtre
Petitefenetre
```

L'exemple ci-dessous est un exemple plus complet : il présente un document HTML permettant d'ouvrir une nouvelle fenêtre en cliquant sur un bouton.

```
<HTML>
<Head>
<title> Ouverture d'une nouvelle fenêtre </title>
</Head>
<body>
<h1> Ouverture d'une nouvelle fenêtre </h1>
```

```
<hr>
utilisez les boutons ci-dessous pour tester la création et la fermeture de fenêtres à
l'aide de Java Script
<hr>
<FORM name="winform">
<input type="button" value="Ouvrir une nouvelle fenêtre"
onClick="nouvellefenetre=window.open('','NouvelleFenetre','toolbar=no,
status=no,width=200,height=200');">
<p>
<input type="button" value="Fermer la nouvelle fenêtre"
onClick="nouvellefenetre.close();">
<p>
<input type="button" value="Fermer la fenêtre principale"
onClick="window.close();">
</form>
<hr>
</body>
</HTML>
```

- Insertion de pauses dans l'exécution d'un script

Parfois on cherche à faire en sorte qu'un script ne fasse rien, et ce pour une période déterminée. Java Script possède pour cela une fonction prédéfinie. La méthode *window.setTimeout* permet de spécifier un délai et une commande, laquelle s'exécutera une fois le délai écoulé.

Note : contrairement à ce que peut laisser penser le titre de cette section, les pauses laissent le navigateur exécuter les actions en cours. L'instruction spécifiée dans la méthode setTimeout ne sera exécutée qu'une fois le délai spécifié écoulé, mais d'autres choses peuvent se produire entre-temps (par exemple : un événement activant un gestionnaire d'événement).

La méthode setTimeout comprend deux paramètres ; le premier est une instruction (un groupe d'instructions) en Java Script entre guillemets. Le second est un délai en *ms.*

Exp :

```
Id = window.setTimeout("alert(' Les 20 secondes se sont écoulés')",20000);
```

Une variable (ici : Id) permet de stocker un identificateur ; vous pouvez ainsi créer plusieurs pauses, chacune ayant un identificateur particulier. Vous pouvez ensuite interrompre la pause à l'aide de la méthode *clearTimeout()* en spécifiant son identificateur :

Exp :

Window.clearTimeout(Id);

Vous pouvez l'utiliser et la tester dans le TP N°1 dans l'exo du message défilant.

2. Les cadres et Java Script

La plupart des navigateurs actuels prennent en charge les cadres ou *frames*, qui permettent de diviser la fenêtre du navigateur en plusieurs volets. Chaque volet peut afficher un document HTML séparé.

- Objets Java Script permettant de manipuler les cadres

Lorsqu'une fenêtre contient plusieurs cadres, chacun d'eux est représenté en Java Script par un objet *frame*. Cet objet est équivalent à un objet window, à cela prêt qu'il sert de manipuler les cadres. Le nom de l'objet *frame* est celui qu'on lui donne avec l'attribut *name* à l'intérieur de la balise <*FRAME*>.

Exp :

```
<HTML>
<Head>
<title> Personnalisation d'objets prédéfinis </title>
</Head>
<FRAMESET ROWS="*,*" COLS="*,*">
<FRAME NAME="hautgauche" SRC="e.html">
<FRAME NAME="hautdroite" SRC="f.html">
<FRAME NAME="basgauche"  SRC="g.html">
<FRAME NAME="basdroite"  SRC="h.html">
</FRAMESET>
<BODY>  ////////////////////////// remarquer !!!!!
</body>
</HTML>
```

Note : la declaration des cadres se fait en dehors de toute balise html.

- Le tableau *frames*

Au lieu de vous référer aux cadres d'un document par leurs noms, utilisez le tableau *frames*. Ce tableau stocke les informations concernant chacun des cadres d'un document. Le numéro d'indice des cadres commence à 0 et avec la première balise <frame> du document frameset.

Exp :

* *parent.frames[0]* : c'est le cadre hautgauche.

* *parent.frames[1]* : c'est le cadre hautdroite.

…

II. Les formulaires

Java Script permet et rend les formulaires HTML plus interactifs, de valider les informations saisies par l'utilisateur et d'afficher les données en fonction d'autres données.

1. Les formulaires HTML

Les formulaires font partie des fonctions les plus importantes de HTML. Comme on le verra Java Script permet d'ajouter diverses fonctions très utiles. On va commencer par créer un formulaire HTML.

- Définir un formulaire

Les formulaires HTML commencent par la balise *<FORM>*. Cette balise qui indique donc le début d'un formulaire, autorise l'insertion d'éléments de formulaire. La balise *<FORM>* comprend trois paramètres :

➔ *NAME* : qui est le nom du formulaire. Vous n'étes pas obligé d'attribuer de nom à vos formulaires, mais on les manipulera ainsi plus facilement à l'aide de Java Script.

➔ *METHOD* : qui peut avoir les valeurs GET et POST ; il s'agit des deux méthodes d'envoi des réponses au serveur.

➔ *ACTION* : qui est le script CGI auquel les réponses seront envoyées. Vous pouvez également utiliser l'action *mailto:* pour envoyer les réponses du formulaire à une adresse électronique.

Exp :

<FORM NAME="commandes" METHOD="GET" ACTION="commandes.cgi">

L'exemple ci-dessus de la balise <FORM> est un formulaire nommé commandes. Ce formulaire utilise la méthode GET et envoie les données

recueillies à un script CGI nommé commandes.cgi qui se trouve dans le même répertoire que la page web.

Dans le cas où c'est Java Script qui traite toutes les données du formulaire, vous pouvez vous passer des attributs METHOD et ACTION, utilisez une balise *<FORM>* pour indiquer seulement le nom du formulaire :

<FORM NAME="formulaire">

La balise *<FORM>* est suivie de différentes balises représentant les éléments de formulaire. Il peut, par exemple, s'agir de champs de texte, de boutons ou de cases à cocher. Nous allons voir ci-dessous comment Java Script affecte des objets à chacun des éléments d'un formulaire.

2. Utilisation de l'objet form

Chaque formulaire d'une page Web est représenté en Java Script par un objet *form*, dont le nom est identique à l'attribut NAME de la balise <FORM> qui a contribué à sa création.

On peut aussi utiliser le tableau *forms* pour se référer aux formulaires d'une page. Ce tableau regroupe tous les formulaires de la page, le premier recevant l'indice 0. Si le premier formulaire du document s'appelle *formulaire1*, on a le choix de se référer à ce formulaire comme suit :

** document.formulaire1*

** document.forms[0]*

3. Propriétés de l'objet form

Chaque objet *form* dispose d'une liste d'éléments et de propriétés, qui correspondent pour la plupart des attributs de la balise *<FORM>* correspondante. On peut cependant les définir à partir de Java Script. En voici les principaux :

- *action* : correspond à l'attribut ACTION du formulaire ; soit le programme auquel les réponses du formulaire seront soumises.
- *length* : c'est le nombre d'éléments du formulaire. Cette propriété ne peut être modifiée.
- *method* : c'est la méthode employée pour envoyer le formulaire, GET ou POST.
- *target* : c'est la fenêtre dans laquelle le résultat du formulaire (traité par un script CGI) s'affichera. En principe, on utilise la fenêtre principale, et c'est le formulaire lui même qui est remplacé.

- Envoyer et réinitialiser des formulaires

L'objet *form* dispose de deux méthodes, *submit* et *reset*. On peut utiliser ces méthodes pour envoyer les données ou réinitialiser le contenu du formulaire sans que l'utilisateur ait besoin de cliquer sur un bouton. Ces méthodes pourront vous être utiles dans le cas où l'utilisateur clique sur une image,un lien ou effectue une autre action qui, en principe, ne permet pas d'envoyer les réponses.

Note : si vous utiliser la méthode submit pour envoyer les réponses par courrier électronique, le navigateur demandera à l'utilisateur de confirmer l'envoi des données. Il n'existe aucun moyen d'envoyer ainsi les données sans l'accord de l'utilisateur.

- Détecter les événements de formulaire

L'objet *form* dispose de deux gestionnaires d'événements, *onSubmit* et *onReset*. On peut spécifier un ensemble d'instructions Java Script ou un appel de fonction pour ces événements en les plaçant dans la balise *<FORM>* qui définit le formulaire.

4. Eléments de formulaire et Java Script

La propriété la plus importante de l'objet *form* est le tableau *éléments*, qui contient un objet pour chacun des éléments du formulaire. On peut se référer à un élément par son nom ou son numéro d'indice dans le tableau. Ainsi, les expressions ci-dessous se réfèrent toutes les deux au premier élément du formulaire commandes, le champ de texte nom1 :

** document.commandes.elements[0]*

** document.commandes.nom1*

Et si on suppose que c'est notre premier formulaire; alors on peut utiliser l'expression suivante :

** document.forms[0].éléments[0]*

** document.forms[0].nom1*

Note : généralement on utilise les noms des formulaires et des éléments ; c'est la manière la plus simple de s'y référer.

Exp :

➔ *document.forms.length* : cette expression renvoie le nombre de formulaires du document.

➔ *document.forms[0].elements.length* : cette expression renvoie le nombre d'éléments du premier formulaire du document.

- *Champs de texte* : l'élément de formulaire le plus couramment employé est le champ de texte. Celui-ci servira à demander à l'utilisateur tout genre d'information. Java Script permet d'afficher automatiquement du texte dans un champ de texte.

Exp :

<INPUT TYPE="TEXT" NAME="texte1" VALUE="bonjour" SIZE="30">

On définit ainsi un champ de texte nommé *texte1*. le champ a la valeur par défaut *"bonjour"* et permet la saisie d'un maximum de 30 caractères. Java Script de réfère à ce champ comme un objet *text* ayant le nom *texte1*.

Les champs de texte sont les plus faciles à manipuler à l'aide de Java Script. Chaque objet *text* dispose des propriétés suivantes :

➔ *name* : qui est le nom du champ. Il est aussi utilisé comme nom de l'objet.

➔ *defaultvalue* : qui est la valeur par défaut du champ ; elle correspond à l'attribut *VALUE*. Il s'agit d'une propriété en lecture seule.

➔ *value* : qui est la valeur courante. Initialement, elle est identique à la valeur par défaut, mais elle peut être modifiée par l'utilisateur ou par une fonction Java Script.

➔➔➔ En général, la manipulation de champs de texte consiste à utiliser l'attribut value pour lire la valeur saisie par l'utilisateur ou pour modifier son contenu. Ainsi, l'instruction ci-dessous donne la valeur "champ de texte" au champ de texte type_element du formulaire formulaire_1 :

Exp : *document.formulaire_1.type_element.value = "champ de texte";*

- Zone de texte : les zones de texte sont définies à l'aide d'une balise spécifique, *<TEXTAREA>*, et sont représentées par l'objet *textarea*. Les zones de textes se distinguent des champs de texte puisqu'ils permettent à l'utilisateur de saisir plus d'une ligne de données.

Exp :

<TEXTAREA NAME="texte2" ROWS="2" COLS="70">

Cela est le contenu de la balise TEXTAREA

</TEXTAREA>

L'exemple ci-dessus définit une zone de texte nommée *texte2* d'une taille de 2 lignes sur 70 colonnes. En Java Script, cette zone de texte sera représentée par un objet zone de texte nommé *texte2*, qui sera fille de l'objet form.

Le texte situé entre la balise de début et la balise de fin est utilisée comme valeur initiale de la zone de texte ; on peut y insérer des sauts de ligne.

➔ Manipulation du texte de formulaires : les objets text et textarea disposent de plusieurs méthodes :

→ *focus()* : pour activer le champ correspondant. De ce fait, le pointeur vient se placer dans ce champ et celui-ci devient le champ courant.

→ *blur()* : produit l'effet inverse ; il "désactive" le champ.

→ *select()* : sélectionne le texte du champ, cette méthode sélectionne toujours tout le texte ; il n'existe aucun moyen de sélectionner une partie du texte.

On peut également se servir des gestionnaires d'événement pour détecter le changement de valeur d'un champ. Les objets text et textarea prennent en charge les gestionnaires d'événements suivants :

→ *onFocus* : se produit lorsque le champ de texte est activé (lorsque l'utilisateur clique dessus par la souris par exemple).

→ *onBlur* : se produit lorsque le champ de texte est désactivé.

→ *onChange* : se produit lorsque l'utilisateur modifie du texte, puis "sort" du champ en passant à un autre.

→ *onSelect* : se produit lorsque l'utilisateur sélectionne une partie ou tout le texte d'un champ. Si le texte a été sélectionné par la méthode *select()* cet événement n'est pas déclenché.

Exp :

```
<TEXTAREA name="texte" rows="2" cols="70"
 onFocus="window.alert('coucouccccc');" onBlur="window.alert('salut');">
coucou
</textarea>
```

<script language="Java Script1.3">

texte.select();

texte.blur();

</script>

- Boutons

Les boutons sont un autre type d'éléments de formulaire. Ils utilisent la balise *<INPUT>* et peuvent être de trois types :

➔ *type="SUBMIT",* un bouton qui permet d'envoyer le contenu du formulaire à un script CGI.

➔ *type="RESET",* un bouton qui permet de réinitialiser le contenu du formulaire.

➔ *type="BUTTON",* un bouton générique. Il n'effectue aucune action de lui même, mais vous pouvez lui en attribuer une à l'aide d'un gestionnaire d'événements Java Script.

Exp :

<INPUT TYPE="SUBMIT" NAME="envoi" VALUE="Cliquer Ici">

pour détecter le clic d'un utilisateur sur un bouton d'envoi ou de réinitialisation, utilisez les gestionnaires d'événements onSubmit et onReset dont il a été question plus haut. Pour les boutons génériques, utilisez un gestionnaire d'événement onClick.

- Cases à cocher

Une case à cocher est un élément de formulaire qui se présente sous la forme d'un petit carré. En cliquant sur une case à cocher, on l'active ou on la désactive, ce qui permet de proposer à l'utilisateur des choix oui/non.

Exp :

<INPUT TYPE="CHECKBOX" NAME="coche" VALUE="Oui" CHECKED>

De même que dans les cas précédents, NAME permet de donner un nom à l'élément de formulaire. L'attribut *VALUE* affecte une signification à la case à cocher ; c'est la valeur qui est renvoyée si la case est cochée. La valeur par défaut est "*on*". L'attribut CHECKED peut être utilisé pour que la case soit cochée par défaut.

Les cases à cocher sont simples : elles ne comportent que deux états. Cependant, elles disposent de quatre propriétés en Java Script :

➔ *name* : qui est le nom de la case à cocher et **aussi le nom de l'objet**.

➔ *value* : qui est la valeur "true" de la case à cocher, généralement *on*. Cette valeur est utilisée par le serveur pour indiquer que la case est cochée. En Java Script utilisez plutôt la propriété CHECKED.

➔ *checked* : qui est la valeur courante de la case à cocher. Il s'agit d'une valeur booléenne : true si elle est cochée, false si elle ne l'est pas.

Exp :

Document.commandes.coche.checked = true;

La case à cocher dispose d'un seul événement : onClick, qui se produit lorsque l'utilisateur clique sur la case à cocher. Cet événement se déclenche quelque soit l'état initial de la case à cocher ; par conséquent on doit faire appel à la propriété *checked* pour savoir ce qui s'est passé.