Les objets de base

Les objets seront abordés par ordre alphabétique dans ce chapitre.

1. Objet Array

Dans la mesure où cet objet a été étudié dans le cadre du chapitre Tableaux, nous ne revenons pas ici sur cet objet.

2. Objet Date

La gestion des dates constitue une préoccupation fréquente dans les développements Web.

À titre indicatif, voici les principales méthodes rattachées à cet objet :

- `getDate()`
- `getDay()`
- `getFullYear()`
- `getHours()`
- `getMilliseconds()`
- `getMinutes()`
- `getMonth()`
- `getSeconds()`
- `getTime()`
- `getTimezoneOffset()`
- `getYear()`
- `setDate()`
- `setFullYear()`
- `setHours()`
- `setMilliseconds()`
- `setMinutes()`
- `setMonth()`
- `setSeconds()`
- `setTime()`
- `setYear()`
- `toGMTString()`
- `toLocaleString()`

La grande majorité des méthodes annoncées ci-avant va être étudiée au travers d'un premier exemple nommé DATE_01.htm.

Exemple 1

Le code source de l'exemple illustratif est fourni entièrement ci-après (code intégré dans la section HTML <body> du script).

Débutons par la méthode getDate() qui donne le numéro de jour dans le mois pour un objet de type Date. Dans notre cas la date analysée est la date du jour. Une date nommée dateDuJour est instanciée et a pour valeur la date du jour. La fonction getDate() appliquée à l'objet dateDuJour donne finalement le numéro du jour dans le mois. Le résultat est affiché pour contrôle.

```
/* Méthode getDate -> retourne le n° du jour
dans le mois de la date système */
var dateDuJour = new Date();
var jourMois = dateDuJour.getDate();
document.write("N° du jour de la date système : " + jourMois);
```

La méthode getDay retourne le numéro du jour dans la semaine pour un objet de type Date. Le traitement porte une fois de plus sur la date du jour. La méthode getDay() fournit un résultat allant de 0 à 6. Un tableau de correspondance pourrait donner le nom du jour en clair le cas échéant (ceci n'a pas été prévu dans notre exemple).

```
/* Méthode getDay -> retourne le n° du jour dans la semaine
(n° de 0 à 6) de la date système */
var dateDuJour = new Date();
var jourSemaine = dateDuJour.getDay();
document.write("<br />N° du jour dans la semaine (n° de 0 à 6)
de la date système : " + jourSemaine);
```

Le numéro du mois (compris entre 0 et 11) est obtenu à partir d'une date par la méthode getMonth(). Une fois de plus le traitement porte toujours sur la date du jour.

```
/* Méthode getMonth -> retourne le n° du mois (n° de 0 à 11)
de la date système */
var dateDuJour = new Date();
var mois = dateDuJour.getMonth();
document.write("<br />N° du mois de la date système : " + (mois + 1));
```

Le numéro de l'année (sur quatre positions) est obtenu à partir d'une date par la méthode getFullYear(). Dans notre exemple le traitement porte sur la date du jour.

```
/* Méthode getFullYear -> retourne le n° de l'année
(sur 4 positions) de la date système */
var dateDuJour = new Date();
var annee = dateDuJour.getFullYear();
document.write("<br />N° de l'année de la date système : " + annee);
```

Pour récupérer l'heure (compris entre 0 et 23) à partir d'un objet Date, il sera fait recours à la méthode getHours ().

```
/* Méthode getHours -> retourne l'heure (n° de 0 à 23) de la
```

```
date système */
var dateDuJour = new Date();
var heure = dateDuJour.getHours();
document.write("<br />N° de l'heure de la date système : " + heure);
```

Pour les minutes, la méthode se nomme `getMinutes()`.

```
/* Méthode getMinutes -> retourne le n° de la minute (n° de 0 à 59)
de la date système */
var dateDuJour = new Date();
var minute = dateDuJour.getMinutes();
document.write("<br />N° de la minute de la date système : " + minute);
```

et enfin pour les secondes, la méthode est `getSeconds()`.

```
/* Méthode getSeconds -> retourne le n° de la seconde (n° de 0 à 59) */
var dateDuJour = new Date();
var seconde = dateDuJour.getSeconds();
document.write("<br />N° de la seconde de la date système : " + seconde);
```

Passons maintenant aux méthodes permettant d'affecter des valeurs (setters).

Par exemple, il est possible d'annoncer la date correspondant au 15 du mois pour la date en cours comme ceci :

```
/* Méthode setDate -> définit un n° de jour dans le mois par rapport
à la date système (le 15 du mois dans notre cas) */
var dateDuJour = new Date();
dateDuJour.setDate(15);
document.write("<br />Le 15 du mois correspondant à la date système : " + dateDuJour);
```

Avec la séquence de code suivante, vous pouvez effectuer un "retour arrière" d'un mois par rapport à une date de référence (ici la date du jour).

```
/* Méthode setMonth -> définit la date un mois en arrière
par rapport à la date système */
var dateDuJour = new Date();
dateDuJour.setMonth(dateDuJour.getMonth()-1);
document.write("<br />Un mois en arrière par rapport à la date
système : " + dateDuJour);
```

Le même traitement est aussi possible pour un "retour arrière" d'un an :

```
/* Méthode setFullYear -> définit la date un an en arrière par rapport à
la date système */
var dateDuJour = new Date();
dateDuJour.setFullYear(dateDuJour.getFullYear()-1);
```

```
document.write("<br />Un an en arrière par rapport à la date système : "
+ dateDuJour);
```

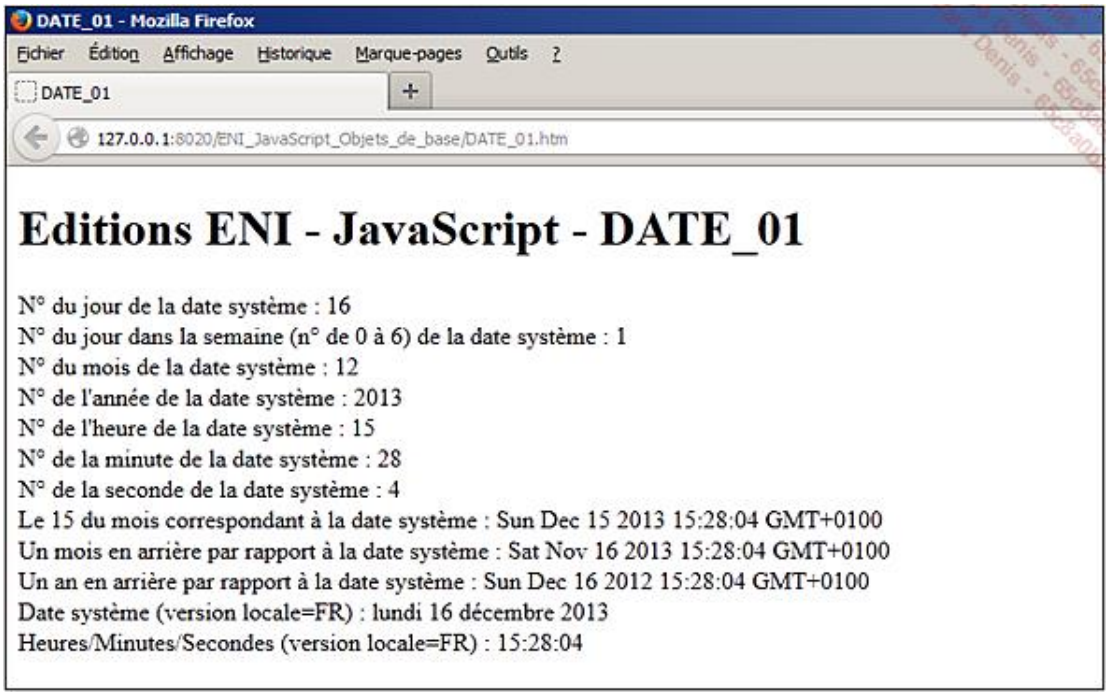
Il nous reste à voir comment convertir en version française la date système. Nous aurons recours à la méthode `toLocaleDateString()` :

```
/* Méthode toLocaleDateString -> convertit la date système
en chaîne date en version locale=FR */
var dateDuJour = new Date();
var dateLocale = dateDuJour.toLocaleDateString();
document.write("<br />Date système (version locale=FR) : " + dateLocale);
```

Un traitement identique peut être appliqué pour obtenir les heures, les minutes et les secondes dans un formalisme français :

```
/* Méthode toLocaleTimeString -> convertit la date système
en chaîne heures/minutes/secondes en version locale=FR */
var dateDuJour = new Date();
var heureLocale = dateDuJour.toLocaleTimeString();
document.write("<br />Heures/Minutes/Secondes (version locale=FR) : " +
heureLocale);
```

Ce script donne l'affichage suivant lors de son exécution (le lundi 26 août 2013) :



Exemple 2

Dans ce deuxième exemple, toujours lié à la problématique de date, nous allons afficher l'heure en cours dans un formalisme hh:mm:ss.

Le code placé dans la section HTML <body> est le suivant :

```
<!-- Début script JavaScript -->
<script type="text/JavaScript">

    /* Affichage du nom du script */
    alert("DATE_02");

    /* Appel de la fonction afficherHeureCourante */
    var monHeure = setInterval (
           function()
           {
               afficherHeureCourante()
           }, 1000
    );

</script>

<!-- Élément span d'affichage du timer -->
Heure en cours : <span id="heureEnCours"></span>
```

La méthode `setInterval` a pour premier paramètre une fonction qui elle-même appelle une fonction de nom `afficherHeureCourante` (étudiée par la suite) et une durée exprimée en millisecondes (1000 = 1 seconde) correspondant au rythme d'actualisation de l'affichage de l'heure système au format hh:mm:ss.

Un `span` identifié par `heureEnCours` est aussi prévu pour recevoir l'affichage généré par la fonction `afficherHeureCourante`.

Passons au code de la fonction `afficherHeureCourante` placée dans la section HTML <head>.

```
/* Fonction afficherHeureCourante */
function afficherHeureCourante()
{
    /* Récupération de l'heure système */
    var maReferenceTemps = new Date();

    /* Conversion de l'heure système en format local (hh:mm:ss) */
    var hhmmss = maReferenceTemps.toLocaleTimeString();

    /* Affichage dans le span heureEnCours */
    document.getElementById("heureEnCours").innerHTML = hhmmss;
}
```

L'heure système (date système) est récupérée par instanciation d'un objet de type `Date` dans la variable `maReferenceTemps`.

Une conversion de la partie heures/minutes/secondes de cette date est ensuite assurée avec le formalisme local (hh:mm:ss).

Le script de la fonction se termine par le report de cette information dans la division évoquée précédemment.

L'affichage à l'écran se présente comme suit :

Exemple 3

Étudions un dernier exemple lié à la gestion du temps, mais n'utilisant pas explicitement la notion d'objet, dans lequel nous allons programmer un chronomètre (timer) simpliste.

Le code positionné dans la section HTML <body> est le suivant :

```
<!-- Formulaire avec boutons de démarrage et d'arrêt du timer -->
<div>
    <input
          type="button"
          value="Démarrage du timer"
          onclick="demarrerTimer()"
    />
    <input
          type="button"
          value="Arrêt du timer"
          onclick="arreterTimer()"
    />
</div>

<!-- Division d'affichage du timer -->
<br /><br />
<div id="affichageTimer">Timer</div>
```

Le code débute par un formulaire à deux boutons déclenchant pour le premier une fonction de démarrage d'un timer et pour le second l'arrêt de ce même timer.

Ensuite une division d'affichage du timer est prévue.

Passons à l'étude des fonctions positionnées dans la section HTML <head>.

La fonction `demarrerTimer` a pour code :

```
/* Fonction demarrerTimer */
function demarrerTimer()
{
```

```
    /* Valeur de départ du timer */
    valeurTimer = 11;

    /* Appel de la fonction d'actualisation du timer */
    actualiserTimer();
}
```

Cette fonction fixe la valeur de départ du timer (11 dans notre cas) et appelle la fonction d'actualisation du timer (décrémentation d'une unité toutes les secondes).

La fonction `actualiserTimer` contient ce code :

```
/* Fonction actualiserTimer */
function actualiserTimer()
{
    if (valeurTimer > 0)
    {
        /* Décrémentation d'une unité (seconde) */
        valeurTimer = valeurTimer - 1;
        /* Affichage de la valeur actualisée du timer */
        document.getElementById("affichageTimer")
        .innerHTML = valeurTimer;
        /* Actualisation par appel de la fonction actualiserTimer */
        if(valeurTimer > 0)
        {
            /* Actualisation toutes les secondes
            (1000 millisecondes) */
            monTimer = setTimeout("actualiserTimer()", 1000);
        }
    }
}
```

La fonction `actualiserTimer` s'appelle elle-même (notion de récursivité en programmation) toutes les secondes par l'intermédiaire de la méthode `setTimeout("actualiserTimer()", 1000)`.

L'actualisation sera interrompue quand le décompte aura atteint la valeur zéro.

Un affichage de la valeur actuelle du timer est effectué par :

```
/* Affichage de la valeur actualisée du timer */
document.getElementById("affichageTimer").innerHTML = valeurTimer;
```

Enfin, le timer peut aussi être interrompu via un clic sur un bouton dédié du formulaire déclenchant la fonction `arreterTimer`.

Le code de la fonction `arreterTimer` est le suivant :

```
/* Fonction arreterTimer*/
function arreterTimer()
{
```

```
    clearTimeout(monTimer);
}
```

La méthode `clearTimeout()` à laquelle le nom du timer à stopper a été passé (`monTimer`) provoque l'arrêt du timer.

L'affichage obtenu à l'écran est :

3. Objet Math

Une fois de plus pour cet objet, il n'est pas prévu ici de faire le tour complet des méthodes disponibles. Vous pourrez consulter un référentiel comme celui du Mozilla Developper Network pour obtenir la liste exhaustive de celles-ci à l'adresse suivante : https://developer.mozilla.org/fr/docs/JavaScript/Reference/Global_Objects/Math

L'objet `Math` est, comme `Array` et `Date`, un objet natif. Les méthodes associées servent à manipuler des éléments numériques.

Il faut noter que `Math` n'est pas un constructeur, donc écrire `monNombre = new Math()` serait une erreur.

Une mise en œuvre des principales méthodes est faite dans les lignes qui suivent.

Débutons par la méthode `abs` retournant la valeur absolue d'un nombre passé en paramètre.

```
/* Valeur absolue de -13 */
document.write("Valeur absolue de -13 : " + Math.abs(-13));
```

La méthode `ceil` fournit la valeur entière immédiatement supérieure à un nombre passé en paramètre.

```
/* Valeur plafond de 12.5 */
document.write("<br />Valeur plafond (entier immédiatement supérieur)
de 12.5 : " + Math.ceil(12.5));
```

La méthode `floor` retourne au contraire la valeur entière immédiatement inférieure à un nombre passé en paramètre.

```
/* Valeur plancher de 12.5 */
document.write("<br />Valeur plancher (entier immédiatement inférieur)
de 12.5 : " + Math.floor(12.5));
```

La méthode `max` est particulièrement intéressante car elle renvoie le maximum non pas uniquement de deux nombres mais d'une série de nombres. Elle évite donc une multitude de tests en cascade.

```
/* Valeur maximale entre 2, 5, -3, 12 */
document.write("<br />Valeur maximale entre 2, 5, -3, 12 : " +
Math.max(2, 5, -3, 12));
```

La méthode `min` est le pendant de la méthode `max` mais cette fois-ci pour la détermination de la valeur minimale.

```
/* Valeur minimale entre 2, 5, -3, 12 */
document.write("<br />Valeur minimale entre 2, 5, -3, 12 : "
+ Math.min(2, 5, -3, 12));
```

La méthode `pow` sert à l'élévation à la puissance (exponentiation). Elle requiert deux paramètres.

```
/* Valeur 5 à l'exposant 2 */
document.write("<br />Valeur 5 à l'exposant 2 : " + Math.pow(5, 2));
```

Il est souvent utile de réaliser des tirages aléatoires de nombres. La méthode `random` produit un nombre compris entre 0 et 1 de manière aléatoire. Il est ensuite aisé de retraiter cette valeur pour obtenir par exemple un tirage de dé à 6 faces (faces numérotées de 1 à 6) dans un jeu.

```
/* Valeur aléatoire (comprise entre 0 et 1) */
document.write("<br />Valeur aléatoire (comprise entre 0 et 1) : " +
Math.random());
```

La méthode `round` est aussi très utile, elle permet d'arrondir un nombre à l'entier le plus proche.

```
/* Valeur arrondie de 12.4 */
document.write("<br />Valeur arrondie de 12.4 : " + Math.round(12.4));

/* Valeur arrondie de 12.6 */
document.write("<br />Valeur arrondie de 12.6 : " + Math.round(12.6));
```

La méthode `sqrt` est également fréquemment employée, elle sert à obtenir la racine carrée d'un nombre passé en paramètre.

```
/* Valeur racine carrée de 25 */
document.write("<br />Racine carrée de 25 : " + Math.sqrt(25));
```

Terminons avec les méthodes trigonométriques.

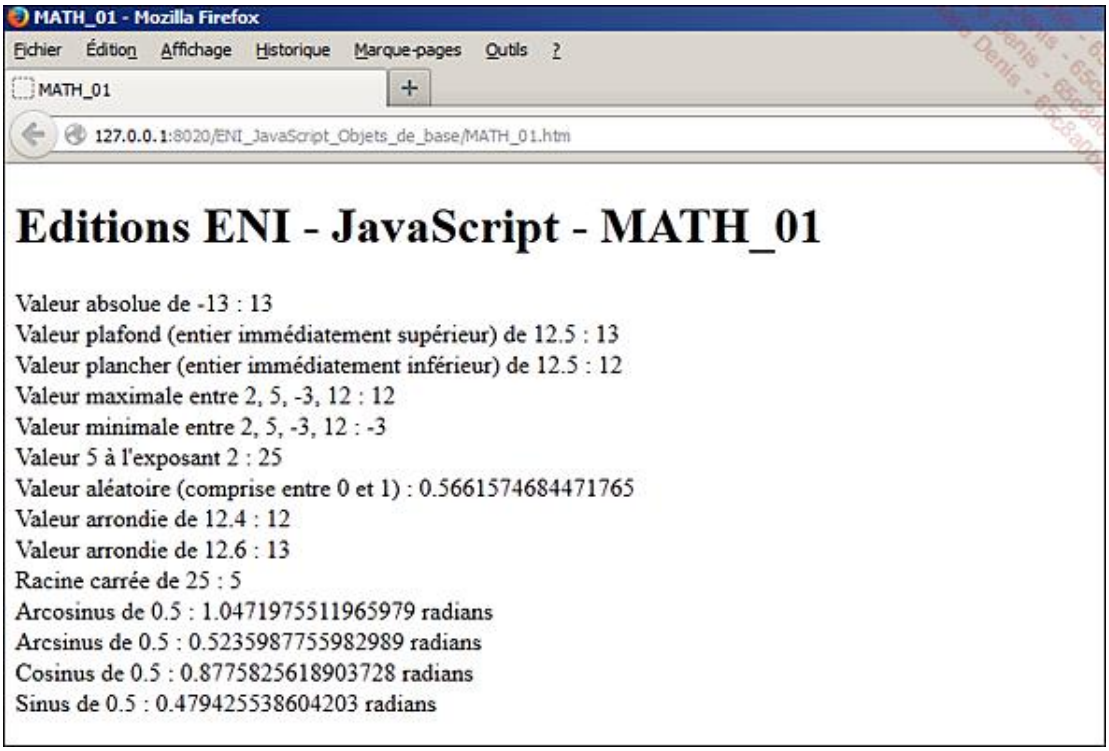
```
/* Valeur arcosinus de 0.5 (exprimée en radians) */
document.write("<br />Arcosinus de 0.5 : "
                + Math.acos(0.5)
                + " radians");

/* Valeur arcsinus de 0.5 (exprimée en radians) */
document.write("<br />Arcsinus de 0.5 : "
                + Math.asin(0.5)
                + " radians");

/* Valeur cosinus de 0.5 (exprimée en radians) */
document.write("<br />Cosinus de 0.5 : "
                + Math.cos(0.5)
                + " radians");

/* Valeur sinus de 0.5 (exprimée en radians) */
document.write("<br />Sinus de 0.5 : "
                + Math.sin(0.5)
                + " radians");
```

Nous obtenons ceci à l'exécution :



4. Objet window

Faisons dans cette section le tour rapide (non exhaustif) des méthodes et propriétés associées à l'objet DOM `window`.

L'objet window représente la fenêtre du navigateur.

Notez que dans les navigateurs proposant le multifenêtrage (onglets), comme Firefox par exemple, chaque onglet contient un objet window.

Illustrons l'utilisation des différentes méthodes par cinq exemples distincts.

Exemple 1

Dans ce premier exemple, l'ensemble du code est une fois de plus totalement intégré dans la section HTML <body>.

L'objectif du code est simple, ouvrir une fenêtre pop-up à partir de l'onglet courant. Il est à noter que le traitement ne s'exécute correctement que si le navigateur autorise la création de fenêtres de type "pop-up". Cette possibilité est souvent interdite (option du navigateur) afin d'éviter l'apparition de fenêtres publicitaires intempestives.

Le code de la méthode open est très simple :

```
/* Ouverture de la fenêtre */
/* Paramètres (dans l'ordre de mise en place) :
- URL (optionnel) : Indique l'adresse URL de la page à ouvrir. Si aucune
  URL n'est spécifiée, une nouvelle fenêtre avec about: blank est ouverte
- Nom de la fenêtre (optionnel) : Spécifie l'attribut cible ou le nom de
  la fenêtre.
  Les valeurs suivantes sont supportées :
    _blank : URL est chargée dans une nouvelle fenêtre (valeur par défaut)
    _parent : URL est chargée dans le cadre parent
    _self : URL remplace la page courante
    _top : URL remplace les jeux de cadres qui peuvent être chargés
    Nom : Le nom de la fenêtre
- Spécifications détaillées de la fenêtre :
    * height=pixels : Hauteur de la fenêtre (100 pixels minimum)
    * left=pixels : Position de la fenêtre par rapport à la gauche de l'écran
    * location=yes|no|1|0 : Présence ou non d'une barre d'adresse
    * menubar=yes|no|1|0 : Présence ou non d'une barre de menus
    * resizable=yes|no|1|0 : Possibilité ou non de redimensionner la fenêtre
    * scrollbars=yes|no|1|0 : Présence ou non de barres de défilement
    * status=yes|no|1|0 : Présence ou non d'une barre de statut
    * titlebar=yes|no|1|0 : Présence ou non d'une barre de titre
    * toolbar=yes|no|1|0 : Présence ou non d'une barre d'outils
    * width=pixels : Largeur de la fenêtre en pixels
*/
maFenetre = window.open("", "FENETRE_01", "width=200, height=100,
location=yes, titlebar=yes");
```

Les nombreux commentaires placés dans le code source indiquent précisément le rôle des différents paramètres. Dans notre exemple, il n'est pas prévu d'URL (la fenêtre sera donc a priori sans contenu). La fenêtre est ici nommée (FENETRE_01). Enfin, ses dimensions (largeur et hauteur exprimées en pixels) sont indiquées et la fenêtre apparaîtra avec une barre d'adresse (location) et une barre de titre (titlebar).

Il est possible, notamment en cas d'absence d'URL, de programmer un contenu pour la nouvelle fenêtre comme suit :

```
/* Écriture d'un message dans la fenêtre */
maFenetre.document.write("Fenêtre créée en <b>JavaScript</b>");
```

Il reste ensuite à se positionner sur cette fenêtre par la méthode `focus` :

```
/* Focus sur la fenêtre */
maFenetre.focus();
```

L'exécution du script donne l'affichage suivant :

Exemple 2

Dans ce deuxième exemple, outre l'ouverture (création) d'une nouvelle fenêtre, voyons la fermeture. Le code est réparti entre la section HTML `<body>` (formulaire à deux boutons pour créer et fermer la fenêtre) et la section HTML `<head>`.

Commençons par le formulaire :

```
<!-- Formulaire -->
<form>
   Ouverture d'une fenêtre avec passage de l'URL en paramètre<br />
   <input
           type="button"
           value="Ouverture d'une fenêtre yahoo.fr"
           onclick="ouvrirFenetre('http://www.yahoo.fr')"
   />
   <br /><br /><br />
   Fermeture de la fenêtre<br />
   <input
           type="button"
           value="Fermeture de la fenêtre"
           onclick="fermerFenetre()"
```

```
   />
</form>
```

Le premier `input` de type `button` appelle la fonction JavaScript nommée `ouvrirFenetre` en lui passant l'URL d'un site à afficher dans cette fenêtre (`http://www.yahoo.fr`).

Le second `input` appelle une fonction, nommée `fermerFenetre`, dont le rôle est bien sûr de fermer la fenêtre.

Passons au code source de la fonction `ouvrirFenetre`.

```
/* Fonction ouvrirFenetre */
function ouvrirFenetre(URL)
{
    maFenetre = window.open(
           URL, "FENETRE_02",
           "width=400,
           height=200,
           location=1,
           titlebar=yes,
           resizable=yes"
   );
}
```

Ce code est sans surprise, l'URL passée en paramètre est positionnée en première position dans les parenthèses. Viennent ensuite le nom de la fenêtre (`FENETRE_02`) et des indications techniques :

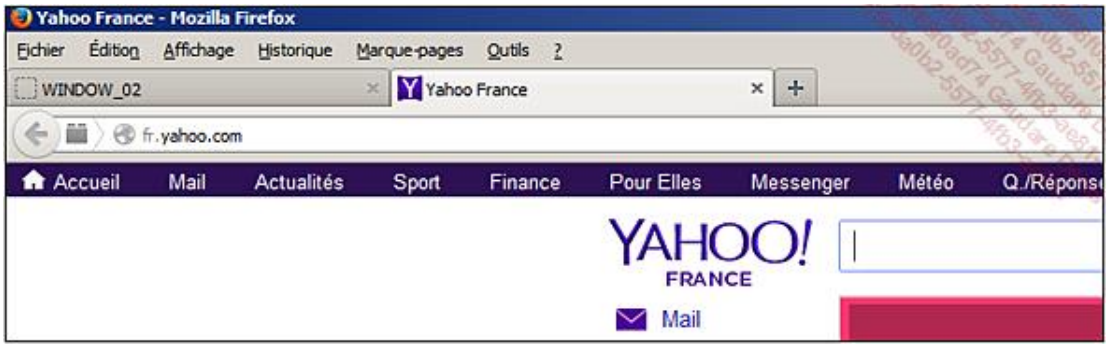
- `width` : largeur de la fenêtre exprimée en pixels,
- `height` : hauteur de la fenêtre exprimée en pixels,
- `location` : présence d'une barre d'adresse (1 équivaut à yes),
- `resizable` : possibilité pour l'utilisateur de redimensionner la fenêtre.

Le code de la fonction `fermerFenetre` est le suivant :

```
/* Fonction fermerFenetre */
function fermerFenetre()
{
    maFenetre.close();
}
```

La méthode `close` ferme naturellement la fenêtre. Nous obtenons ceci à l'exécution :

Un clic sur le bouton intitulé **Ouverture d'une fenêtre yahoo.fr** donne ceci :



Exemple 3

Le comportement de ce script va être assez proche de celui de l'exemple précédent.

Un formulaire avec un unique bouton déclenchant la création est prévu, la fermeture de la fenêtre étant déclenchée 10 secondes après la création.

Le code du formulaire est le suivant :

```
<!-- Formulaire -->
<form>
   Ouverture d'une fenêtre avec passage de l'URL en paramètre<br />
   <input
          type="button"
          value="Ouverture d'une fenêtre yahoo.fr (pendant 10 secondes)"
          onclick="ouvrirFenetre('http://www.yahoo.fr')"
   />
</form>
```

Le code de la méthode `ouvrirFenetre` appelle la fonction `fermerFenetre` après une temporisation de 10 secondes.

```
/* Fonction ouvrirFenetre */
function ouvrirFenetre(URL)
{
    /* Ouverture de la fenêtre */
    maFenetre = window.open(
           URL,
           "FENETRE",
           "width=400,
           height=200,
           location=1,
           titlebar=yes,
           resizable=yes");

    /* Temporisation de la fermeture de la fenêtre
    (10 secondes d'affichage) */
    timeout = setTimeout("fermerFenetre()", 10000);
}
```

La méthode `setTimeout` utilise deux paramètres :

- le nom de la méthode à déclencher, la méthode `fermerFenetre` dans notre cas,
- le délai avant le déclenchement, 10000 millisecondes, c'est-à-dire 10 secondes.

À l'exécution du script, l'affichage est :

Une fois un clic effectué sur le bouton une fenêtre Yahoo s'ouvre et reste affichée 10 secondes.

Exemple 4

Dans cet exemple, nous allons voir comment détecter la résolution graphique de l'écran de l'utilisateur. En fonction de la résolution repérée, il sera possible d'adapter la mise en forme des contenus Web.

La séquence de code est très simple :

```
/* Détermination de la résolution (largeur et hauteur) */
var largeur = window.screen.availWidth;
var hauteur = window.screen.availHeight;
var resolution = "Largeur (pixels) : " + largeur + "<br />" + "Hauteur
```

```
(pixels) : " + hauteur;
document.write("Résolution : "+ resolution);
```

La propriété `availWidth` donne la résolution en largeur (exprimée en pixels) et la propriété `availHeight` la résolution en hauteur. L'affichage obtenu est :

Exemple 5

Dans ce cinquième exemple, nous allons mettre en œuvre deux scripts `WINDOW_05.htm` et `WINDOW_06.htm` qui vont s'appeler mutuellement. Cet exemple sera l'occasion d'étudier un certain nombre de méthodes et de propriétés propres aux objets de type `window`.

Dans le script `WINDOW_05.htm`, vous trouverez un formulaire dans lequel un bouton d'appel du deuxième script est placé :

```
<!-- Accès à la page WINDOW_06.htm -->
<form post="WINDOW_06.htm">
   <input
           type="button"
           value="Accès à la page WINDOW_06.htm"
           onclick="accederWINDOW_06()"
   />
</form>
```

Le bouton active une fonction nommée `accederWINDOW_06` qui se trouve dans la section HTML `<head>` du script `WINDOW_05.htm`. Voici son code source :

```
/* Fonction accederWINDOW_06 */
function accederWINDOW_06()
{
    /* Accès à la page WINDOW_06.htm */
    maFenetre = window.open("WINDOW_06.htm", "_self");
}
```

Le code est sans surprise, il s'agit d'un appel du script `WINDOW_06.htm` qui va s'afficher dans le même onglet (paramètre `_self`).

L'exécution du script donne ceci à l'affichage :

Passons au décryptage du fichier `WINDOW_06.htm`. Dans la section HTML `<body>`, le code intégré est :

```
<!-- Début script JavaScript -->
<script type="text/JavaScript">

    /*
    Informations sur l'URL de la page en cours
    /*


/* Propriété host */
    document.write("<br /> Propriété location.host : "
    + window.location.host);

    /* Propriété hostname */
    document.write("<br /> Propriété location.hostname : "
    + window.location.hostname);

    /* Propriété href */
    document.write("<br /> Propriété location.href : "
    + window.location.href);

    /* Propriété pathname */
    document.write("<br /> Propriété location.pathname : "
    + window.location.pathname);

    /* Propriété port */
    document.write("<br /> Propriété location.port : "
    + window.location.port);

</script>
```

L'affichage des valeurs de propriétés donne dans notre cas :

Rappelons que la plupart des scripts JavaScript de ce livre ont été testés sur un serveur Web local (Apache). L'installation de ce serveur a été réalisée par l'intermédiaire du package EasyPHP (http://www.easyphp.org).

La propriété host est constituée de l'adresse du serveur (adresse locale `127.0.0.1` dans notre cas) suivie du port de communication du serveur (`8020`).

Un bouton de retour à la page précédente (`WINDOW_05.htm`) est aussi positionné dans cette section HTML `<body>` :

```
<!-- Retour à la page précédente -->
<div>
    <input type="button" value="Retour à la page précédente"
    onclick="revenirPagePrecedente()" />
</div>
```

Ce bouton appelle une fonction nommée `revenirPagePrecedente` dont le code est :

```
/* Fonction revenirPagePrecedente */
function revenirPagePrecedente()
{
    /* Retour à la page précédente */
    window.history.back()
}
```

La méthode `window.history.back()` provoque le retour au script se trouvant immédiatement avant la page en cours dans l'historique de navigation, c'est-à-dire dans notre cas `WINDOW_05.htm`.

Le code JavaScript étudié au travers de cet exemple est extrêmement intéressant notamment quand un contrôle d'accès entre les pages d'un site doit être mis en place.

5. Objet navigator

Cet objet contient les propriétés de votre navigateur. Le langage JavaScript a la réputation d'être sensible

(comportements différents) au navigateur voire à la version du navigateur dans lequel il s'exécute. Vous trouverez dans les propriétés de l'objet `navigator` le nécessaire pour résoudre cette problématique.

Dans l'exemple présenté ci-après, les principales propriétés vont être testées.

Dans la section HTML `<body>`, vous trouverez essentiellement un bouton d'appel d'une fonction listant les propriétés ainsi qu'une division d'affichage :

```
<!-- Appel de la fonction d'affichage des caractéristiques du navigateur -->
<div>
        <input
                type="button"
                value="Caractéristiques du navigateur"
                onclick="testerNavigateur()"
        />
</div>

<!-- Division d'affichage des caractéristiques du navigateur -->
<br /><br />
<div id="affichageInfosNavigateur"></div>
```

La fonction `testerNavigateur` a le code suivant :

```
/* Fonction testerNavigateur */
function testerNavigateur()
{
    var caracteristiquesNavigateur = "";
    caracteristiquesNavigateur =  "Code du navigateur : "
        + navigator.appCodeName + "<br />";
    caracteristiquesNavigateur += "Nom du navigateur : "
        + navigator.appName + "<br />";
    caracteristiquesNavigateur += "Version : "
        + navigator.appVersion + "<br />";
    caracteristiquesNavigateur += "Support des cookies : "
        + navigator.cookieEnabled + "</p>";
    caracteristiquesNavigateur += "Système d'exploitation : "
        + navigator.platform + "<br />";
    caracteristiquesNavigateur += "User-agent header: "
        + navigator.userAgent + "<br />";
    caracteristiquesNavigateur += "User-agent language: "
        + navigator.systemLanguage + "<br />";

    /* Affichage des caractéristiques du navigateur */
    document.getElementById("affichageInfosNavigateur")
    .innerHTML = caracteristiquesNavigateur;

}
```

Les propriétés `appCodeName`, `appName` et `appVersion` sont particulièrement intéressantes en matière de détection de navigateur.

La propriété `cookieEnabled` est aussi très fréquemment utilisée pour savoir si le navigateur supporte ou non les cookies. En cas de non-prise en compte des cookies, il faudra mettre en œuvre d'autres solutions de stockage (local ou distant). La notion de cookie sera étudiée en détail dans le chapitre Gestion des cookies en JavaScript.

La propriété `userAgent` est aussi très exploitée car elle contient, sous la forme d'une chaîne de caractères, toutes les informations nécessaires pour une détection fine du navigateur sur lequel le script s'exécute.

À titre indicatif, sur ma propre configuration l'exécution donne ceci :

6. Objet String

Comme dans la majorité des langages de programmation, le traitement des chaînes de caractères est extrêmement important. JavaScript propose pour cela un jeu complet de méthodes. Nous allons étudier les principales méthodes au travers d'un exemple commenté.

Ce code exposé ci-après est placé dans la section HTML `<body>`.

Commençons par définir deux variables texte :

```
/* Définition de variables de type String */
var prenom = new String("Christian");
var nom = "VIGOUROUX";
document.write("Prénom : " + prenom);
document.write("<br />Nom : " + nom);
```

Pour la variable `prenom`, il a été fait recours explicitement à un constructeur. Les deux variables sont ensuite affichées pour contrôle.

Déterminons ensuite l'initiale du prénom :

```
/* Initiale du prénom */
```

```
document.write("<br />Initiale du prénom : " + prenom.charAt(0));
```

La méthode `charAt()` avec en paramètre 0 permet cette opération aisément. Vous noterez que les caractères d'une chaîne sont numérotés à partir de zéro.

Déterminons ensuite la position occupée dans la table ASCII par l'initiale du nom :

```
/* Caractère Unicode (position dans table ASCII) de l'initiale du nom */
document.write("<br />Caractère Unicode (position dans table ASCII) de
l'initiale du nom : " + nom.charCodeAt(0));
```

La méthode utilisée est `charCodeAt()` avec toujours en paramètre la position de la lettre dans la chaîne. Dans notre cas la réponse sera 86 car la lettre "V" est en position 86 dans la table ASCII.

Depuis les années 60, le code ASCII (*American Standard Code for Information Interchange*) est devenu le standard en matière de codage et d'échange entre ordinateurs. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

Voyons maintenant comment concaténer (juxtaposer ou accoler) deux chaînes de caractères :

```
/* Concaténation du prénom et du nom */
var identite = prenom.concat(" ");
identite = identite.concat(nom);
document.write("<br />Prénom & Nom : " + identite);
```

La concaténation est un peu surprenante en JavaScript, la méthode `concat()` est appliquée à la chaîne de base à laquelle on ajoute une extension (une constante ou une autre variable).

Étudions ensuite les multiples techniques de repérage ou d'extraction d'une sous-chaîne dans une chaîne de caractères. L'exposé ne sera pas complet car JavaScript comme la majorité des langages de programmation supporte également les expressions régulières, concept qui ne sera pas étudié dans ce livre.

Commençons par le repérage de la première apparition d'une lettre (à partir de la gauche) dans une chaîne de caractères :

```
/* Position de la première lettre "U" dans le nom (attention la
numérotation débute à 0) */
var position_premier_u = nom.indexOf("U") + 1;
document.write("<br />Le premier 'U' du nom est en position : " +
position_premier_u);
```

La méthode utilisée est `indexOf()` avec en paramètre la lettre recherchée ("U" dans notre cas). Vous aurez noté dans l'exemple que la valeur affichée est incrémentée de 1, n'oubliez pas que dans les chaînes de caractères la numérotation des lettres débute à zéro.

Il est aussi possible de repérer la dernière occurrence d'une lettre (recherche à partir de la droite) par l'intermédiaire de la méthode `lastIndexOf()` :

```
/* Position de la dernière lettre "U" dans le nom (attention la
numérotation débute à 0) */
var position_dernier_u = nom.lastIndexOf("U") + 1;
document.write("<br />Le dernier 'U' du nom est en position : " +
position_dernier_u);
```

Pour les méthodes `indexOf()` et `lastIndexOf()`, si aucune occurrence n'est trouvée la réponse est `0`.

Passons au remplacement d'une sous-chaîne par une autre dans une chaîne de caractères :

```
/* Remplacement d'une chaîne de caractères */
var origineNom = nom.replace("ROUX", "REUX");
document.write("<br />Origine du nom : " + origineNom);
```

Dans cet exemple, la sous-chaîne "ROUX" est remplacée par "REUX" par l'intermédiaire de la méthode `replace()`.

Il est aussi fréquent que l'on ait à extraire une sous-chaîne à partir d'une chaîne de caractères. Plusieurs méthodes sont disponibles pour cela.

Commençons par la méthode `slice()` :

```
/* Extraction d'une sous-chaîne par la méthode slice(position_début,
position_fin - 1) */
var surnom = nom.slice(0, 4);
document.write("<br />Surnom (méthode slice) : " + surnom);
```

La méthode `slice()` assure ici l'extraction du premier caractère (numéroté zéro) jusqu'au cinquième non compris. Cela peut paraître surprenant donc, la réponse dans notre cas sera "VIGO".

La méthode `substr()` est aussi utilisable :

```
/* Extraction d'une sous-chaîne par la méthode substr(position_début,
longueur) */
surnom = nom.substr(0, 4);
document.write("<br />Surnom (méthode substr) : " + surnom);
```

Avec la méthode `substr()` le deuxième paramètre a une signification différente, il s'agit de la longueur de la sous-chaîne extraite. La réponse sera également "VIGO".

Enfin la troisième possibilité est l'emploi de la méthode `substring()` :

```
/* Extraction d'une sous-chaîne par la méthode substring(position_début,
position_fin - 1) */
surnom = nom.substring(0, 4);
document.write("<br />Surnom (méthode substring) : " + surnom);
```

La seule différence entre `slice()` et `substring()` est que le deuxième paramètre est facultatif dans le cas de

la méthode `substring()`. En l'absence du second paramètre de la méthode `substring()`, l'extraction se fait jusqu'au dernier caractère de la chaîne explorée.

La méthode `split()` assure la mise en tableau des différents mots constituant une chaîne de caractères :

```
/* Mise d'une phrase en tableau */
var auteur = "Christian VIGOUROUX est le rédacteur de ce livre";
var tableauAuteur = auteur.split(" ");
document.write("<br />Le nom de l'auteur de ce livre : " +
tableauAuteur[1]);
```

Dans le message de contrôle, le nom de l'auteur (VIGOUROUX) est reproduit, le nom ayant été stocké à la position 1 dans le tableau `tableauAuteur` (n'oubliez pas la numérotation des éléments des tableaux à partir de zéro).

Voyons aussi comment assurer les conversions de chaînes de caractères en majuscules (méthode `toUpperCase()`) et en minuscules (méthode `toLowerCase()`) :

```
/* Conversion en majuscules */
document.write("<br />Le prénom en majuscules : " +
prenom.toUpperCase());

/* Conversion en minuscules */
document.write("<br />Le nom en minuscules : " + nom.toLowerCase());
```

Enfin, terminons par la suppression des espaces non significatifs en début et en fin de chaîne de caractères :

```
/* Suppression des espaces non significatifs en début et fin de chaîne de
caractères */
var identite = "                    Christian VIGOUROUX
";
document.write("<br />Identité avec espaces non significatifs : " + "[" +
identite + "]");
document.write("<br />Identité sans espaces non significatifs : " + "[" +
identite.trim() + "]");
```

Les crochets ([]) placés dans les deux méthodes `document.write` ont pour objectif de bien matérialiser la suppression des espaces non significatifs (dans le second cas) assurée par la méthode `trim()`.

L'affichage obtenu est le suivant :