




# JavaScript

# JavaScript

Cette section va vous montrer comment utiliser le langage :

- JavaScript et HTML
- Objets, propriétés, fonctions et méthodes
- Définition d'un nouveau type d'objet
- Utilisation d'un objet par défaut
- Tour d'horizon du langage JavaScript



JavaScript est un langage objet interprété dont les instructions sont stockées (en clair) dans des pages HTML. Petit frère de Java, JavaScript en reprend les grandes lignes. Son accès est cependant plus simple pour les non-programmeurs. Après cette formation, les concepteurs de pages Web pourront donner vie à leurs œuvres en y intégrant du code JavaScript. Ainsi, ils pourront par exemple animer textes et graphiques et réagir aux actions de l'utilisateur telles que clic sur une zone, déplacement de la souris, etc..

JavaScript a été créé en 1995 par Brendan Eich pour Netscape sous le nom ECMAScript. Il est apparu en 1996 dans le navigateur Netscape.

Pour saisir du code JavaScript dans vos pages HTML, vous pouvez utiliser un simple éditeur de texte, comme le bloc-notes de Windows, ou mieux, un éditeur dédié à la saisie de code, comme Notepad++ par exemple.

Ce programme est librement téléchargeable sur <http://notepad-plus-plus.org/fr/>

L'intégration de code JavaScript dans une page HTML peut se faire de deux façons :

1. A l'aide d'une balise <script> </script>.
2. En mettant en place un gestionnaire d'événements.

Dans le premier cas, le code est exécuté pendant le chargement de la page. Dans le deuxième cas, il est en mesure de réagir à des événements utilisateur : clics, focus ou perte de focus sur un contrôle, appui sur le bouton **Submit** ou sur le bouton **Quit** dans un formulaire, etc.

## ***Définition de code JavaScript avec une balise `<script>`***

La syntaxe du balise `<script>` `</script>` est la suivante :

```
<script language = "JavaScript">  
    Lignes de code  
</script>
```

la balise `<script>` doit être intégré à la suite de la balise `<head>` :


```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8">  
    <script>  
        //Lignes de code JavaScript  
    </script>  
</head>  
<body>  
    <!-- Balises HTML -->  
<script></script>  
</body>  
</html>
```



Cet exemple simpliste vous montre où et comment implémenter vos premières instructions JavaScript.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  <p>Alors que celui-là provient de la section BODY du document HTML.</p>

  <script>
    document.write('Ce texte est écrit par une instruction JavaScript.');
```




Dans la diapositive précédente, nous avons utilisé la fonction JavaScript **document.write()** pour afficher du texte dans le DOM.

Vous auriez également pu utiliser :

- La fonction JavaScript `alert()` pour afficher une boîte de message :  
`alert('ceci est un message JavaScript');`
- Ou encore afficher du texte dans la console avec l'instruction suivante :  
`console.log('Texte affiché dans la console');`

Pour visualiser le texte dans la console, appuyez sur la touche **F12** du clavier et basculez sur l'onglet ou dans le panneau (selon le navigateur) **Console**.



Le code JavaScript peut être placé entre les balises `<head>` et `</head>` ou entre les balises `<body>` et `</body>`.

Si vous le souhaitez, vous pouvez stocker tout votre code JavaScript dans un fichier externe d'extension `.js`. Vous ferez référence à ce fichier avec les instructions HTML suivantes entre les balises `<head>` et `</head>` ou `<body>` et `</body>` :

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



# ***Une fonction dans une balise <script>***

Comme la plupart des langages, JavaScript permet de définir des fonctions. Ces fonctions doivent impérativement être placées entre les balises <head> et </head>. En effet, le code compris entre ces deux balises étant chargé en premier, cela garantit que les fonctions JavaScript auront été mémorisées avant que l'utilisateur ne tente une quelconque action qui provoquerait leur appel.

La syntaxe permettant de déclarer une fonction est proche de celle utilisée dans le langage C :

```
function nom([param1, ...paramN]) {  
    // Une ou plusieurs instructions  
}
```

Où :

- nom est le nom de la fonction ;
- param1 à paramN sont les éventuels paramètres passés à la fonction ;
- Instructions représente une ou plusieurs instructions exécutées par la fonction.

Lorsqu'une fonction a été définie, vous utiliserez un marqueur <script> </script> pour l'appeler :

```
<script>  
    nom( [param1, ...paramN] );  
</script>
```

- où nom est le nom de la fonction à appeler ;
- et param1 à paramN sont les éventuels paramètres passés à la fonction.



Exercice :  
Définissez la fonction **carre()** qui calcule et affiche le carré du nombre qui lui est passé en argument.

Solution :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  Test de la fonction <b>carre</b>:
  <script>
    function carre(x){
      document.write('Le carré de ' + x + ' est ' + x*x + '<br>');
    }
    carre(7);
    carre(12);
  </script>
</body>
</html>
```



# Portée des variables

Une variable déclarée au début du script, avant toutes fonctions, sera globale. Elle peut être utilisée n'importe où dans le script.

Une variable déclarée dans une fonction aura une portée limitée à cette seule fonction : elle n'est pas utilisable en dehors de la fonction. On parle alors de « **variable locale** ».



# *Mise en place d'un gestionnaire d'événements*

Dans le navigateur, certaines actions effectuées par l'utilisateur donnent lieu à des événements. Si vous mettez en place un gestionnaire d'événements, il sera automatiquement exécuté lorsque l'événement correspondant se présentera.

Pour définir un gestionnaire d'événements, vous utiliserez la syntaxe suivante :

```
<balise NomEvent = "code JavaScript">
```

- où balise est le nom d'une balise HTML ;
- NomEvent est le nom de l'événement déclenchant ;
- code JavaScript est une instruction JavaScript exécutée lorsque l'événement NomEvent se produit.

A titre d'exemple, l'instruction ci-après définit un bouton de commande, lui donne le nom Résultat et met en place un gestionnaire d'événements qui est sollicité lorsque l'utilisateur clique sur ce bouton.

```
<input  
  type = "button"  
  value = "Résultat"  
  onclick = "suivant(this.form) ; ">
```

Le code qui suit le déclencheur onclick correspond au nom d'une fonction. Vous pouvez spécifier au choix une instruction JavaScript ou le nom d'une fonction. Cette deuxième possibilité est bien plus intéressante, car elle donne le contrôle à une fonction qui contient autant d'instructions que vous le souhaitez.

Exercice :

Définissez un formulaire contenant une zone de texte et un bouton. Lorsque l'utilisateur clique sur le bouton, exécuter la fonction **carre()** qui affiche le carré du nombre tapé dans la zone de texte.

Indice :

Si une balise **<input>** de type **text** a pour attribut **name xyz**, JavaScript peut accéder en lecture et en écriture au contenu de cette balise avec l'instruction suivante :

```
form.xyz.value
```

Solution :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script language = "JavaScript">
    function carre(form) {
      form.resultat.value = form.resultat.value * form.resultat.value;
    }
  </script>
</head>
<body>
  <form>
    Test de la fonction <B>carre</B>:</P>
    <input type="text" name="resultat" size="25">
    <input type="button" value="Carré" onclick = "carre(this.form);">
  </form>
</body>
</html>
```



# ***Objets, propriétés, fonctions et méthodes***

Lorsque vous chargez un document dans le navigateur, quatre objets prédéfinis lui sont instantanément associés :

1. L'objet window représente la fenêtre affichée dans le navigateur.
2. L'objet location contient l'URL du document courant.
3. L'objet document donne accès aux propriétés du document : titre, couleur ou image d'arrière-plan, couleur par défaut du texte, etc.
4. L'objet history donne accès aux URL des pages précédemment visualisées.

Tous ces objets sont accessibles par une syntaxe hiérarchique "à point. Par exemple :

`objet1.objet2.objet3`

Ici, objet3 fait partie d'objet2, qui fait lui-même partie d'objet1.

Cette syntaxe à point se termine souvent par un élément qui n'est pas un objet, mais une propriété.



Exercice :

Définissez un formulaire composé de deux zones de texte, deux boutons radio et trois boutons de commande : Envoyer, Annuler et Majuscules. Un appui sur le bouton Majuscules transforme les caractères de l'adresse e-mail en majuscules.

The screenshot shows a web browser window with the address bar displaying 'E:\data\Mediaforma\Formation\Futures\JavaScript\code\formulaire.htm'. The browser window has a single tab titled 'Un premier formulaire'. The form itself is titled 'Entrez votre nom' and 'Entrez votre adresse e-mail'. The first text field contains 'Michel'. The second text field contains 'MM54@FREE.FR'. Below these fields are two radio buttons labeled 'OUI' and 'NON', with the 'OUI' button selected. At the bottom of the form are three buttons: 'Envoyer', 'Annuler', and 'Majuscules'. A red dashed arrow points from the 'Majuscules' button to the email input field.

Remarque :

Pour accéder à la valeur contenue dans une zone de texte en JavaScript, vous utiliserez la syntaxe suivante :

```
document.f.n.value
```

Où f est le nom (name) du formulaire et n le nom (name) de la zone de texte

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script language = "JavaScript">
    function Majusc(form) {
      document.Quest.Adresse.value = document.Quest.Adresse.value.toUpperCase();

      // Une autre syntaxe possible
      // form.Adresse.value = form.Adresse.value.toUpperCase();
    }
  </script>
  <title>Un premier formulaire</title>
</head>
<body>
  <form name="Quest" action="mailto:bertrand.oscar@Oracle.fr" method="post">
    Entrez votre nom <input type="text" name="Nom"><br>
    Entrez votre adresse e-mail <input type="text" name="Adresse"><br>
    Connaissez-vous Mediaforma ? OUI <input type="radio" name="Client" value="OUI" checked>
    NON <input type="radio" name="Client" value="non"><br>
    <input type="submit" value="Envoyer">
    <input type="reset" value="Annuler">
    <input type="button" value = "Majuscules" onclick="Majusc(this.form) ">
    <script>
  </form>
</body>
</html>
```

# Utilisation d'un objet par défaut

JavaScript implémente plusieurs types d'objets. Lorsqu'un programme utilise fréquemment des propriétés et/ou des méthodes relatives à un même type d'objet, il est possible de simplifier la syntaxe en utilisant l'instruction with. Cette dernière définit un objet par défaut à l'intérieur de ses délimiteurs. Il n'est alors plus nécessaire de le spécifier comme préfixe des propriétés et méthodes.

Exemple :

Les instruction ci-après utilisent plusieurs méthodes relatives à l'objet Math :

```
document.write("Le sinus de " + x + " est " + Math.sin(x) + "<br>");  
document.write("Le cosinus de " + x + " est " + Math.cos(x) + "<br>");  
document.write("La tangente de " + x + " est " + Math.tan(x) + "<br>");  
document.write("L'arc sinus de " + x + " est " + Math.asin(x) + "<br>");  
document.write("L'arc cosinus de " + x + " est " + Math.acos(x) + "<br>");  
document.write("L'arc tangente de " + x + " est " + Math.atan(x) + "<br>");
```

Essayez ce code



Pour soulager l'écriture, il est possible d'utiliser une instruction with :

```
with (Math) {  
    document.write("Le sinus de " + x + " est " + sin(x) + "<br>");  
    document.write("Le cosinus de " + x + " est " + cos(x) + "<br>");  
    document.write("La tangente de " + x + " est " + tan(x) + "<br>");  
    document.write("L'arc sinus de " + x + " est " + asin(x) + "<br>");  
    document.write("L'arc cosinus de " + x + " est " + acos(x) + "<br>");  
    document.write("L'arc tangente de " + x + " est " + atan(x) + "<br>");  
}
```

Essayez ce code

Comme vous le voyez, il n'est plus nécessaire de préciser l'objet Math devant les méthodes, puisqu'il est utilisé par défaut sur toute l'étendue de l'instruction with.

Les fonctions mathématiques `sin()`, `cos()`, `tan()`, etc. demandent des arguments en radians.  
Si vous voulez travailler avec des degrés, vous devez effectuer une conversion degrés -> radians.

Exercice :

Trouvez comment effectuer cette conversion et affichez le sinus de 90°.

Solution

```
<script>
  function radians(degrees) {
    return degrees * Math.PI / 180;
  };

  function degrees(radians) {
    return radians * 180 / Math.PI;
  };

  var x=radians(90);
  document.write('Le sinus de 90° est ' + Math.sin(x) + ' .<br>');
</script>
```





# ***Tour d'horizon du langage JavaScript***

Dans les diapositives qui suivent, vous allez faire connaissance avec les instructions du langage JavaScript et avec les divers objets qui peuvent être accédés par son intermédiaire.

# ***Variables et types de données***

JavaScript est en mesure de travailler avec des nombres réels ou entiers.

Les entiers peuvent être exprimés en décimal (base 10), en octal (base 8) ou en hexadécimal (base 16).

Un nombre octal commence toujours par un zéro, et un nombre hexadécimal par 0x ou 0X. Les éventuelles lettres d'un nombre hexadécimal peuvent indifféremment être exprimées en minuscules ou en majuscules.

Les nombres réels sont composés d'une partie entière suivie d'un point, d'une partie décimale, de la lettre e (ou E) et d'un exposant, éventuellement précédé d'un signe + ou -. Voici quelques exemples de nombres réels :

1.4142135

-32E-7

.5e12

JavaScript peut également travailler avec des chaînes de caractères délimitées par des guillemets ou des apostrophes.

Les affectations de chaînes ci-après sont correctes :

```
var x = "essai";
```

```
x = 'essai';
```

```
x = "essai\concluant";
```

Si nécessaire, vous pouvez inclure un ou plusieurs des caractères de contrôle suivants dans une valeur chaîne :

Code de contrôle	Effet
\b	Backspace
\f	Form feed
\n	Line feed
\r	Carriage return
\t	Tabulation



Il est également possible d'utiliser toutes les balises HTML qui affectent le style des caractères. Par exemple, l'affectation suivante est correcte :

```
var texte = "essai <font size='5' color='#FF0000'>concluant</font>";  
document.write(texte);
```



JavaScript peut également manipuler :

- des booléens qui prennent la valeur **true** ou **false**.
- des tableaux monodimensionnels, accessibles par leur indice.

Par exemple, **memo[0]** désigne le premier élément du tableau et **memo[12]** le treizième élément de ce même tableau.



Pour définir une variable, il n'est pas nécessaire de préciser son type. Utilisez simplement le mot réservé var, indiquez le nom de la variable et affectez-lui une valeur.

Exemples :

```
var petit = 10;  
var grand = 100;  
var titre = 'Types de données';  
var choix = true;
```

#### Attention

Tout comme le C++, le langage JavaScript tient compte des majuscules et des minuscules. Ainsi, par exemple, les variables `choix` et `Choix` ne seront pas, *a priori*, égales. De même, n'essayez pas d'affecter la valeur `True` à une variable booléenne. Une erreur serait générée à l'exécution. En effet, JavaScript connaît la valeur littérale true mais pas la valeur True.

Lorsqu'une valeur typée a été affectée à une variable, il n'est pas interdit de lui affecter par la suite un autre type de valeur. Par exemple, le code ci-après est tout à fait correct :

```
var ma_variable = 'valeur texte';  
...  
ma_variable = 123.456e12;  
...  
ma_variable = true;
```

Le signe + peut être utilisé pour concaténer deux valeurs de type différent. Par exemple, l'instruction suivante est tout à fait licite :

```
ma_variable = 500 + 'kilomètres';
```

# *Conversions de types*

JavaScript est doté de trois fonctions de conversion chaîne/numérique :

- eval() : évaluation et conversion numérique d'une chaîne.
- parseInt() : conversion d'une chaîne en utilisant une base de numération donnée.
- parseFloat() : conversion d'une chaîne en un nombre réel.

## Fonction eval()

```
Résultat = eval(Expression);
```

- Expression est une chaîne de caractères contenant un nombre ou une formule dont le résultat est un nombre ;
- et Résultat est la valeur numérique retournée par la fonction.

Exemples :

```
var a = 5;
```

```
eval("a*2"); // retourne la valeur numérique 10
```

```
eval("123.45"); // retourne la valeur numérique 123.45
```



## Fonction parseInt()

Résultat = parseInt(Chaîne[,Base])

- Chaîne est la chaîne à convertir ;
- Base est la base de numération de la chaîne ;
- Résultat est le résultat numérique de la conversion.

Exemples :

parseInt('FF',16); retourne la valeur numérique 255

parseInt('12',8); retourne la valeur numérique 10

parseInt('essai',10); retourne la valeur NaN

## Fonction parseFloat()

Résultat = parseFloat(Chaîne)

- Chaîne est la chaîne à convertir ;
- Résultat est le résultat réel de la conversion.

Exemples :

`parseFloat('427.35E-1');` retourne la valeur numérique 42.735

`parseFloat('X12');` retourne la valeur NaN

Exercice :

Créez un formulaire contenant deux zones de texte et un bouton.

L'utilisateur pourra entrer un calcul quelconque dans la première zone de texte. Au clic sur le bouton, le résultat du calcul sera affiché dans la deuxième zone de texte.

Solution

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Calculs dans un formulaire</title>
</head>
<body>
  <form name="f">
    <input type="text" name="saisie">
    <input type="button" value="Calculer" onclick="calcul();">
    <input type="text" name="resultat">
  </form>
  <script>
    function calcul() {
      document.f.resultat.value = eval(document.f.saisie.value);
    }
  </script>
</body>
</html>
```



# ***Opérateurs et expressions***

JavaScript utilise la plupart des opérateurs des autres langages de programmation. Pour faciliter leur énumération, nous les répartirons en six grands groupes. Ainsi, nous distinguerons les opérateurs :

- D'affectation.
- Arithmétiques.
- Relationnels et logiques.
- Dédiés aux chaînes de caractères.
- Dédiés aux opérations binaires (qui agissent au niveau des bits).

# Opérateurs d'affectation

Outre l'opérateur d'affectation " = ", JavaScript dispose de six autres opérateurs qui allègent l'écriture :

Opérateur	Fonction	Exemple	Equivalent à
+ =	Addition de l'opérateur de droite à l'opérateur de gauche et affectation à l'opérateur de gauche	a + = b	a = a + b
- =	Soustraction de l'opérateur de droite à l'opérateur de gauche et affectation à l'opérateur de gauche	a - = b	a = a - b
* =	Multiplication de l'opérateur de droite par l'opérateur de gauche et affectation à l'opérateur de gauche	a * = b	a = a * b
/ =	Division de l'opérateur de gauche par l'opérateur de droite et affectation à l'opérateur de gauche	a / = b	a = a / b
% =	Reste de la division entière de l'opérateur de gauche par l'opérateur de droite et affectation à l'opérateur de gauche	a % = b	a = a % b
? :	Affectation conditionnelle (condition ternaire)	a = (c < d) ? e : f	if (c < d) a = e else a = f

Exercice :

Définissez un formulaire contenant deux zones de saisie, un bouton et un textarea. Lorsque l'utilisateur clique sur le bouton, appliquez les opérateurs  $+=$ ,  $*=$ ,  $/=$  et  $\%=$  aux valeurs numériques et affichez les résultats dans le textarea.

**Test des opérateurs élémentaires**

---

**Entrez vos données**

Premier nombre réel R1 =

Deuxième nombre réel R2 =

**Résultats**

```
Après l'opération r1 += r2, r1 a pour valeur 36.620000000000004
Après l'opération r1 *= r2, r1 a pour valeur 301.5
Après l'opération r1 /= r2, r1 a pour valeur 0.5182421227197347
Après l'opération r1 %= r2, r1 a pour valeur 12.5
```

Voici le code HTML à utiliser :

```
<form name="Demo">
  <table>
    <tr align="center">
      <td colspan="2">
        <h1>Test des opérateurs d'affectation</h1>
        <hr>
      </td>
    </tr>
    <tr align = "center">
      <td>
        <b>Entrez vos données </b>
        <p>Premier nombre réel R1 = <input type ="text" name="Reel1" size="10"></p>
        <p>Deuxième nombre réel R2 = <input type="text" name="Reel2" size="10"></p>
      </td>
      <td>
        <input type="button" value="Calcul" onclick="Calcul(this.form) ">
      </td>
    </tr>
    <tr align = "center">
      <td colspan="2">
        <b>Résultats</b>
        <textarea name = Result rows = 4 cols = 65></textarea>
      </td>
    </tr>
  </table>
</form>
```

Ecrivez le code JavaScript nécessaire pour arriver au résultat demandé.



Solution :

```
<script language="JavaScript">
  function Calcul(form) {
    // Récupération des données entrées
    var r1 = parseFloat(form.Reel1.value);
    var r2 = parseFloat(form.Reel2.value);

    // Calculs
    var r = "Après l'opération r1 + = r2, r1 a pour valeur "+eval(r1+r2);
    var ChRes = r+"\r\n";
    r = "Après l'opération r1 - = r2, r1 a pour valeur "+eval(r1-r2);
    r = r1+"-"+r2+" = "+ eval("r1-r2");
    var r = "Après l'opération r1 * = r2, r1 a pour valeur "+eval(r1*r2);
    ChRes = ChRes+r+"\r\n";
    var r = "Après l'opération r1 / = r2, r1 a pour valeur "+eval(r1/r2);
    ChRes = ChRes+r+"\r\n";
    var r = "Après l'opération r1 % = r2, r1 a pour valeur "+eval(r1%r2);
    ChRes = ChRes+r+"\r\n";

    // Affichage des résultats
    form.Result.value = ChRes;
  }
</script>
```

# Opérateurs arithmétiques

Ces opérateurs effectuent des opérations arithmétiques sur leurs opérandes.

Opérateur	Fonction	Exemple
=	affectation	a = 65
+	addition	c = a + b
-	soustraction	d = -a
*	multiplication	c = c * a
/	division	c = a / b
%	modulo (reste de la division entière)	c = a % b

Les opérateurs sont exécutés selon une hiérarchie bien définie ; du plus prioritaire vers le moins prioritaire :

- %
- de changement de signe
- /
- \*
- 
- +
- =



Il est parfois nécessaire d'utiliser des parenthèses pour forcer l'ordre d'exécution ou pour clarifier une expression de calcul.

A titre d'exemple, l'expression :

$$z = a + b / c + 4;$$

N'est pas du tout équivalente à l'expression :

$$z = (a + b) / c + 4;$$

Exercice :

Définissez le code JavaScript nécessaire pour obtenir le résultat suivant lorsque l'utilisateur appuie sur le bouton Calcul :

The screenshot shows a web browser window with the address bar displaying `E:\data\Mediaforma\Formation\Futures\JavaScript\cd\JavaScript\Arithm.htm`. The page title is "Test des opérateurs arithmétiques". The interface is divided into two main sections: "Entrez vos données" and "Résultats".

**Entrez vos données**

Entier 1 :

Entier 2 :

Réel 1 :

Réel 2 :

**Résultats**

13+42=55  
13-42=-29  
13\*42=546  
13/42=0.30952380952380953  
13%42=13  
1.56+4.7=6.26  
1.56-4.7=-3.14  
1.56\*4.7=7.332000000000001  
1.56/4.7=0.331914893617021  
25



Voici le code HTML à utiliser :

```
<form name="demo">
  <table>
    <tr align="center">
      <td colspan="3">
        <h1>Test des opérateurs arithmétiques</h1>
        <hr>
      </td>
    </tr>
    <tr align="center">
      <td>
        <b>entrez vos données</b><br>
        <p>entier 1 : <input type="text" name="Entier1" size="10"></p>
        <p>entier 2 : <input type="text" name="Entier2" size="10"></p>
        <p>réal 1 : <input type="text" name="Reel1" size="10"></p>
        <p>réal 2 : <input type="text" name="Reel2" size="10"></p>
      </td>
      <td>
        <input type="button" value="calcul" onclick="calcul(this.form);">
      </td>
      <td>
        <p><b>résultats</b></p>
        <textarea name="Result" rows="10" cols="26"></textarea>
      </td>
    </tr>
  </table>
</form>
```

Solution :

```
<script language = "JavaScript">
    function calcul(form) {
        var e1 = parseInt(form.Entier1.value);
        var e2 = parseInt(form.Entier2.value);
        var r1 = parseFloat(form.Reel1.value);
        var r2 = parseFloat(form.Reel2.value);

        // Calculs sur les nombres entiers
        var r = e1+" "+e2+" = "+ eval("e1+e2");
        var ChRes = r+"\r\n";
        r = e1+"-"+e2+" = "+ eval("e1-e2");
        ChRes = ChRes+r+"\r\n";
        r = e1+"*"+e2+" = "+ eval("e1*e2");
        ChRes = ChRes+r+"\r\n";
        r = e1+"/"+e2+" = "+ eval("e1/e2");
        ChRes = ChRes+r+"\r\n";
        r = e1+"%" +e2+" = "+ eval("e1%e2");
        ChRes = ChRes+r+"\r\n";

        // Calculs sur les nombres réels
        r = r1+" "+r2+" = "+ eval("r1+r2");
        var ChRes = ChRes+r+"\r\n";
        r = r1+"-"+r2+" = "+ eval("r1-r2");
        ChRes = ChRes+r+"\r\n";
        r = r1+"*"+r2+" = "+ eval("r1*r2");
        ChRes = ChRes+r+"\r\n";
        r = r1+"/"+r2+" = "+ eval("r1/r2");
        ChRes = ChRes+r+"\r\n";
        r = r1+"%" +r2+" = "+ eval("r1%r2");
        ChRes = ChRes+r+"\r\n";

        // Affichage des résultats
        form.Result.value = ChRes;
    }
</script>
```

## Opérateurs relationnels et logiques

Ces opérateurs effectuent des comparaisons sur leurs opérandes qui peuvent être numériques, chaînes ou logiques. Le résultat renvoyé est 0 si la condition n'est pas vérifiée. Il est différent de 0 dans le cas contraire.

Opérateur	Fonction	Exemple
!	NON logique	!(a == b)
<	Inférieur à	a < b
>	Supérieur à	a > b
<=	Inférieur ou égal à	a <= b
>=	Supérieur ou égal à	a >= b
!=	Différent de	a != b
==	Egal à	a == b
&&	ET logique	(a == b) && (c > d)
	OU logique	(a == b)    (c > d)

## Attention

Une des erreurs les plus fréquentes en JavaScript consiste à utiliser l'opérateur `≡` à la place de l'opérateur `==` dans une comparaison logique.

Par exemple, l'instruction :

```
if (couleur = 3)
```

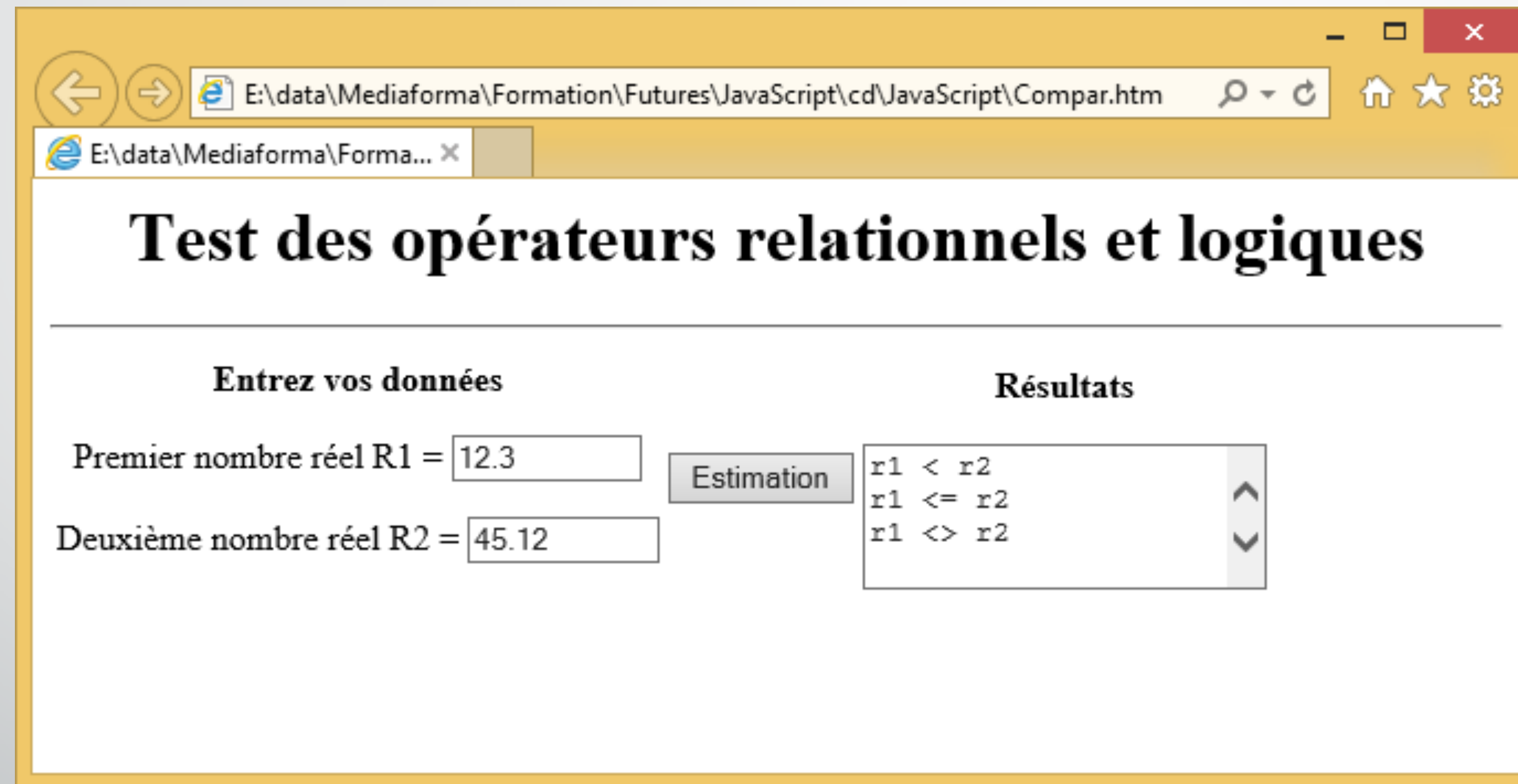
Ne compare pas le contenu de la variable couleur avec la valeur 3. Elle affecte au contraire la valeur 3 à la variable couleur ! L'instruction permettant d'effectuer la comparaison est :

```
if (couleur == 3)
```



Exercice :

Définissez le code JavaScript nécessaire pour obtenir le résultat suivant lorsque l'utilisateur clique sur le bouton **Estimation** :



The screenshot shows a web browser window with a yellow border. The address bar displays the file path `E:\data\Mediaforma\Formation\Futures\JavaScript\cd\JavaScript\Compar.htm`. The page title is "Test des opérateurs relationnels et logiques". Below the title, there are two columns: "Entrez vos données" and "Résultats". In the "Entrez vos données" column, there are two input fields: "Premier nombre réel R1 =" with the value "12.3" and "Deuxième nombre réel R2 =" with the value "45.12". A button labeled "Estimation" is positioned between the input fields and the results column. The "Résultats" column contains a text area with the following output: `r1 < r2`, `r1 <= r2`, and `r1 <> r2`. The text area has a vertical scrollbar on the right side.

Entrez vos données	Résultats
Premier nombre réel R1 = <input type="text" value="12.3"/>	<div>Estimation</div> <div><code>r1 &lt; r2</code> <code>r1 &lt;= r2</code> <code>r1 &lt;&gt; r2</code></div>
Deuxième nombre réel R2 = <input type="text" value="45.12"/>	

Voici le code HTML :

```
<form name="demo">
  <table>
    <tr align="center">
      <td colspan="3">
        <h1>Test des opérateurs relationnels et logiques</h1>
        <hr>
      </td>
    </tr>
    <tr align="center">
      <td>
        <b>Entrez vos données </b></p>
        Premier nombre réel r1 = <input type="text" name="Reel1" size="10"><br>
        Deuxième nombre réel r2 = <input type="text" name="Reel2" size="10"><br>
      </td>
      <td>
        <input type="button" value="Estimation" onclick="Estimation(this.form);">
      </td>
      <td>
        <b>Résultats</b><br>
        <textarea name="Result" rows="4" cols="20"></textarea>
      </td>
    </tr>
  </table>
</form>
```

Solution :

```
<script language="JavaScript">
function Estimation(form) {
    // Récupération des données entrées
    var r1 = parseFloat(form.Reel1.value);
    var r2 = parseFloat(form.Reel2.value);
    // Estimation des données
    var r = "";
    var ChRes = "";
    if (r1 < r2){
        r = "r1 < r2";
        ChRes = ChRes+r+"\r\n";
    }
    if (r1> r2){
        r = "r1> r2";
        ChRes = ChRes+r+"\r\n";
    }
    if (r1 <= r2){
        r = "r1 <= r2";
        ChRes = ChRes+r+"\r\n";
    }
    if (r1>= r2){
        r = "r1>= r2";
        ChRes = ChRes+r+"\r\n";
    }
    if (r1 == r2){
        r = "r1 = r2";
        ChRes = ChRes+r+"\r\n";
    }
    if (r1 != r2){
        r = "r1 <> r2";
        ChRes = ChRes+r+"\r\n";
    }

    // Affichage du résultat
    form.Result.value = ChRes;
}
</script>
```

## Opérateurs dédiés aux chaînes de caractères

En plus des opérateurs de comparaison dont nous venons de parler, vous pouvez utiliser les opérateurs `+` et `+=` pour (respectivement) concaténer deux chaînes et stocker une chaîne à la suite d'une autre.

Exemples :

L'instruction ci-après :

```
a = "gauche " + "droite";
```

stocke la valeur "gauche droite" dans la variable a.

Supposons maintenant que la variable `a` contienne la valeur "gauche ". En utilisant l'instruction suivante, la variable `a` contiendra la valeur "gauche droite" :

```
a += "droite";
```

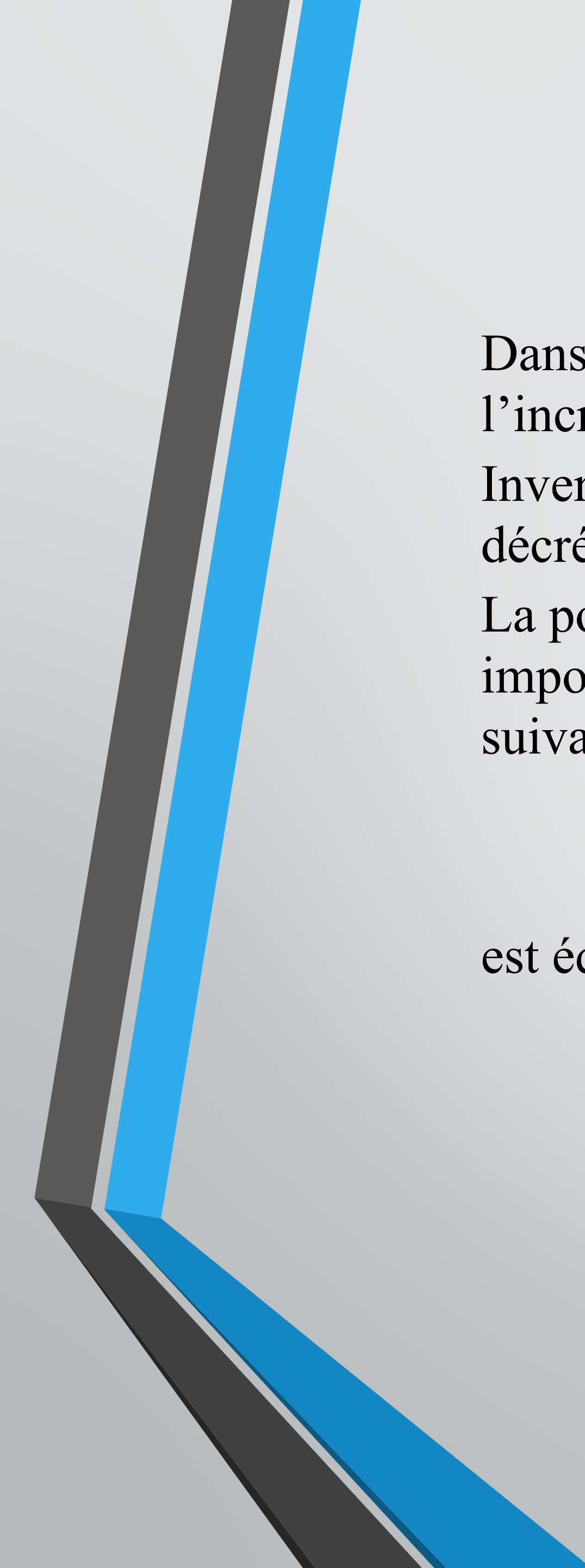


## Opérateurs d'incrémentation

Les opérateurs d'incrémentation (++) et de décrémentation (--) sont utilisés, respectivement pour augmenter et pour diminuer de 1 la valeur stockée dans une variable.

Selon la position de l'opérateur, on parle de post incrémentation, de pré incrémentation, de post décrémentation et de pré décrémentation :

Opérateur	Fonction	Exemple
++	post incrémentation	a++
++	pré incrémentation	++a
--	post décrémentation	a--
--	pré décrémentation	--a



Dans une expression comportant un opérateur de pré incrémentation ou de pré décrémentation, l'incrément (la décrémentation) se déroule avant l'évaluation de l'expression.

Inversement, dans une expression comportant un opérateur de post incrémentation ou de post décrémentation, l'incrément (la décrémentation) se déroule après l'évaluation de l'expression.

La position (post ou pré) des opérateurs d'incrément et de décrémentation n'a aucune importance dans une expression ne comportant qu'un facteur. Par exemple, les expressions suivantes sont la plupart du temps équivalentes :

```
a++;
```

est équivalent à :

```
++a;
```

Cependant, testez le code suivant :

```
var i = 5;  
var j = 5;  
var k = i++;  
var l = ++j;  
document.write(i + ' ' + j + ' ' + k + ' ' + l);
```

Qu'allez-vous obtenir ?

i et j valent 6.

Par contre, k vaut 5 et l vaut 6 !

Exercice :

Ecrivez du code HTML/JavaScript qui met en évidence l'importance de la post/pré incrémentation en appliquant les opérations  $E1 * ++E2$  et  $E1 * E2++$  à deux données saisies par l'utilisateur :

The screenshot shows a web browser window with the address bar displaying `E:\data\Mediaforma\Formation\Futures\JavaScript\cd\JavaScript\IncrDecr.htm`. The page title is "Test des opérateurs d'incrémentation / décrémentation". Below the title, there is a section labeled "Entrez vos données" with two input fields: "Premier nombre entier E1 =" with the value 12, and "Deuxième nombre entier E2 =" with the value 15. A button labeled "Inc/Déc" is positioned to the right of the second input field. Below the input fields, there is a section labeled "Résultats" containing a text area with the following text: "Soit  $E3 = E1 * ++E2$  avec  $E1=12$  et  $E2=16$   
Après le calcul, on obtient  $E2 = 16$  et  $E3 = 192$   
Soit  $E3 = E1 * E2++$  avec  $E1=12$  et  $E2=16$   
Après le calcul, on obtient  $E2 = 16$  et  $E3 = 180$ ".



Voici une variante du code :

```
<script language = "JavaScript">
  function IncDec(form) {
    // Récupération des données entrées
    var e1 = parseInt(form.Entier1.value)
    var e2 = parseInt(form.Entier2.value)

    // Sauvegarde de ces données
    var s1 = e1
    var s2 = e2

    // Premier calcul
    var e3 = e1*++e2
    var r = "Soit E3 = E1 * ++E2 avec E1 = "+e1+" et E2 = "+e2+"\r\n"
    r = r+"  Après le calcul, on obtient E2 = "+e2+" et E3 = "+e3+"\r\n\r\n"
    var ChRes = r

    //Restitution des valeurs et deuxième calcul
    e1 = s1
    e2 = s2
    e3 = e1*e2++
    r = "Soit E3 = E1 * E2++ avec E1 = "+e1+" et E2 = "+e2+"\r\n"
    r = r+"  Après le calcul, on obtient E2 = "+e2+" et E3 = "+e3
    ChRes = ChRes+r

    // Affichage du résultat
    form.Result.value = ChRes
  }
</script>
```

Suite :

```
<form name="demo">
  <table>
    <tr align="center">
      <td colspan="2">
        <h1>Test des opérateurs d'incrémentation/décrémentation</h1>
        <hr>
      </td>
    </tr>
    <tr align="center">
      <td>
        <b>Entrez vos données </b>
        <p>Premier nombre entier e1 = <input type="text" name="Entier1" size="10"></p>
        <p>Deuxième nombre entier e2 = <input type="text" name="Entier2" size="10"></p>
      </td>
      <td>
        <input type="button" value="Inc/Déc" onclick="IncDec(this.form);">
      </td>
    </tr>
    <tr align="center">
      <td colspan="2">
        <b>Résultats</b><br>
        <textarea name="Result" rows="5" cols="60"></textarea>
      </td>
    </tr>
  </table>
</form>
```

# Opérations intervenant au niveau des bits

JavaScript possède plusieurs opérateurs qui effectuent des calculs au niveau binaire, tels que OU logique ou décalage d'un ou de plusieurs bits. Le tableau ci-après résume ces opérateurs.

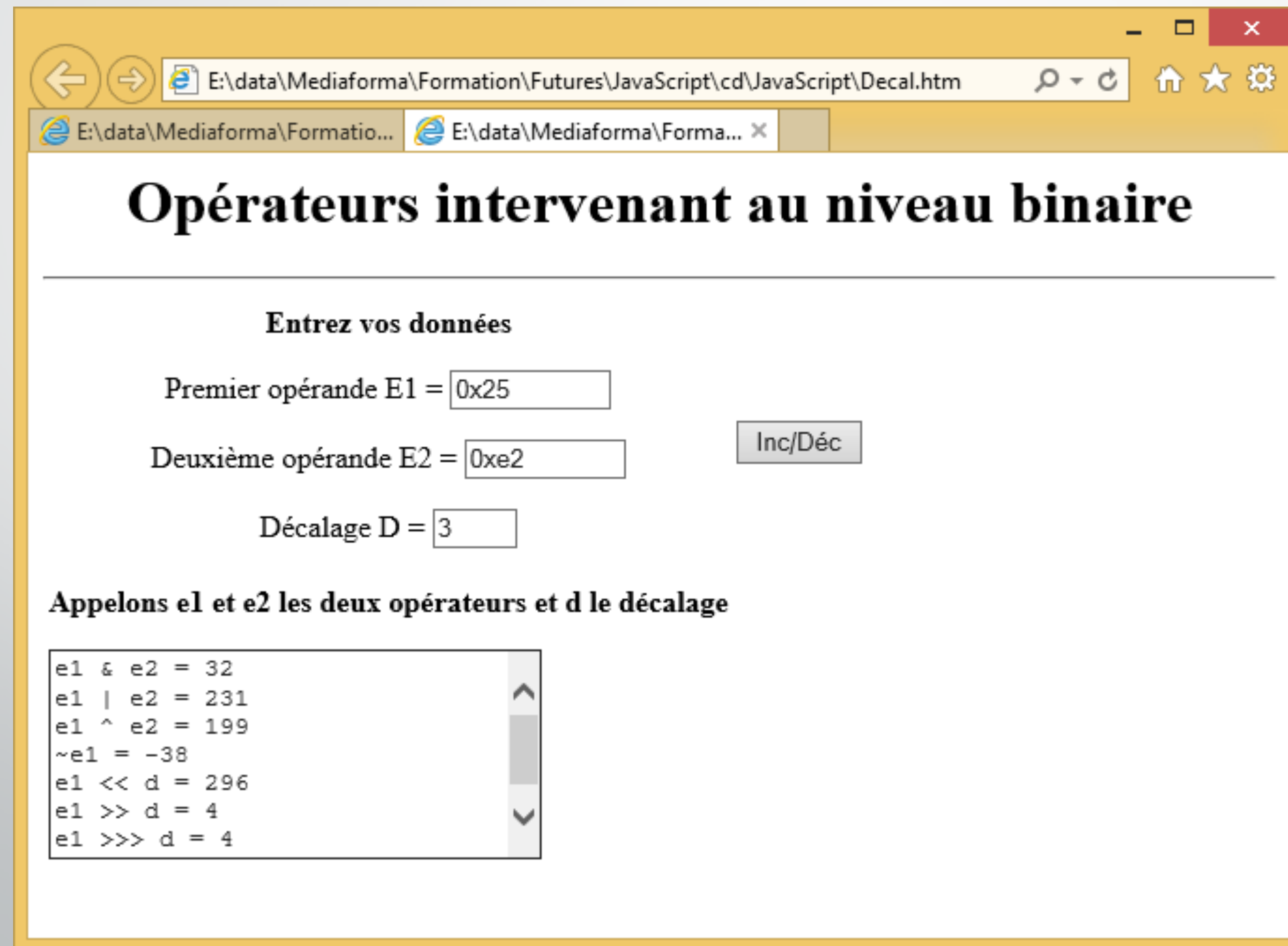
Opérateur	Fonction	Exemple
&	ET logique	a = 0xfa & 0x35
	OU logique	a = 0xfa   0x12
^	OU EXCLUSIF logique	a ^ = 0x32
~	NON logique	a = ~b
>>	Décalage à droite avec signe	a>>= c
>>>	Décalage à droite sans signe	a>>>= c
<<	Décalage à gauche	a = b << c

## Attention

Les opérateurs &, | et ~ ne doivent pas être confondus avec leurs homologues &&, || et ^. Les premiers effectuent des opérations binaires sur leurs opérandes, alors que les seconds effectuent des comparaisons sur leurs opérandes.

Exercice :

Définissez le code HTML et JavaScript nécessaire pour tester les opérateurs binaires. Vous pourriez réaliser le formulaire suivant :



The screenshot shows a web browser window with a yellow title bar. The address bar displays the file path `E:\data\Mediaforma\Formation\Futures\JavaScript\cd\JavaScript\Decal.htm`. The browser has two tabs open, both with the same path. The main content area has a white background and a yellow border. At the top, the title **Opérateurs intervenant au niveau binaire** is centered. Below it is a horizontal line. The form section is titled **Entrez vos données**. It contains three input fields: 'Premier opérande E1 =' with the value `0x25`, 'Deuxième opérande E2 =' with the value `0xe2`, and 'Décalage D =' with the value `3`. To the right of the second input field is a button labeled 'Inc/Déc'. Below the inputs is the instruction **Appelons e1 et e2 les deux opérandeurs et d le décalage**. At the bottom, there is a text area with a scrollbar containing the following JavaScript code:

```
e1 & e2 = 32
e1 | e2 = 231
e1 ^ e2 = 199
~e1 = -38
e1 << d = 296
e1 >> d = 4
e1 >>> d = 4
```



Voici le code HTML

```
<form name="demo">
  <table>
    <tr align="center">
      <td colspan="2">
        <h1>Test des opérateurs d'affectation</h1>
        <hr>
      </td>
    </tr>
    <tr align="center">
      <td>
        <b>Entrez vos données</b><br>
        Premier opérande e1 = <input type="text" name="Entier1" size="10"><br>
        Deuxième opérande e2=<input type="text" name="Entier2" size="10"><br>
        Décalage d = <input type="text" name="Decal" size="3"><p>
      </td>
      <td>
        <input type="button" value="Décalage" onclick="Decalage(this.form);">
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <b>Appelons e1 et e2 les deux opérateurs et d le décalage</b><p>
        <textarea name="Result" rows="7" cols="30"></textarea>
      </td>
    </tr>
  </table>
</form>
```

## Et le code JavaScript

```
<script language = "JavaScript">
  function Decalage(form)
  {
    // Récupération des données entrées
    var e1 = parseInt(form.Entier1.value);
    var e2 = parseInt(form.Entier2.value);
    var d = parseInt(form.Decal.value);

    // Application des opérandes
    r = "e1 & e2 = "+eval(e1&e2)+"\r\n";
    r = r+ "e1 | e2 = "+eval(e1 | e2) +"\r\n";
    r = r+ "e1 ^ e2 = "+eval(e1 ^ e2) +"\r\n";
    r = r+"~e1 = "+eval(~e1) +"\r\n";
    e3 = e1<<d;
    r = r+"e1 << d = "+e3+"\r\n";
    e3 = e1>> d ;
    r = r+"e1>> d = "+e3+"\r\n";
    e3 = e1>>> d ;
    r = r+"e1>>> d = "+e3+"\r\n";

    // Affichage du résultat
    form.Result.value = r;
  }
</script>
```

# L'instruction de contrôle if... else [.. else]

Dans une fonction JavaScript ou entre les marqueurs <script> et </script>, les instructions s'exécutent séquentiellement. Il peut s'agir d'instructions provenant du langage de programmation lui-même ou d'appels de fonctions. Dans le second cas, les instructions appelées peuvent comporter une ou plusieurs instructions simples et/ou appels de fonctions.

Il est souvent nécessaire d'exécuter une portion de code, à la condition qu'une expression logique soit vérifiée. Pour cela, il faut utiliser l'instruction if else dont voici la syntaxe :

```
if (condition) {  
    instruction 1;  
    ...  
    instruction N;  
}  
else {  
    Instruction 1;  
    ...  
    Instruction P;  
}
```

- condition est une condition logique. Elle sera vérifiée si son évaluation est différente de 0. Elle ne sera pas vérifiée si son évaluation est égale à 0. Par exemple, la condition if (3) est toujours vérifiée, et la condition if (0) n'est jamais vérifiée.
- instruction 1 à instruction N sont les instructions exécutées dans le cas où la condition logique est vérifiée.
- instruction 1 à instruction P sont les instructions exécutées dans le cas contraire.



Exercice :

Ecrivez du code HTML/JavaScript pour :

- Demander à l'utilisateur d'entrer une lettre au clavier.
- Afficher un des trois messages suivants en fonction du caractère entré :

Lettre majuscule → Vous avez entré une lettre majuscule.

Lettre minuscule → Vous avez entré une lettre minuscule.

Autre caractère → Le caractère entré n'est pas une lettre.



## Solution :

```
<script language="JavaScript">
function test_casse(form){
    // Récupération de la lettre entrée
    var L = form.Lettre.value;

    // Estimation des données
    if (L>= "A" && L <= "Z") {
        Res = "Vous avez entré une lettre majuscule";
    }
    else{
        if (L>= "a" && L <= "z") {
            Res = "Vous avez entré une lettre minuscule";
        }
        else {
            Res = "Le caractère entré n'est pas une lettre";
        }
    }

    // Affichage des résultats
    form.Result.value = Res;
}
</script>
```

```
<form name = "demo">
    <table>
        <tr align = center>
            <td colspan="2">
                <h1>Test de la casse d'une lettre</h1>
            </td>
            <hr>
        </tr>
        <tr align="center">
            <td>
                <p>Entrez une lettre : <input type="text" name="Lettre" size="1"><br>
            </td>
            <td>
                <input type="button" value="test" onclick="test_casse(this.form);">
            </td>
        </tr>
        <tr align="center">
            <td colspan="2">
                <p><input type="text" name="Result" size="40"></p>
            </td>
        </tr>
    </table>
</form>
```

# ***Instructions répétitives***

Dans un programme, il est souvent nécessaire de faire exécuter plusieurs fois un même bloc d'instructions. Plutôt que de réécrire plusieurs fois ce bloc, il est plus judicieux d'utiliser une instruction de répétition. JavaScript dispose de deux instructions de ce type : for et while. Nous allons examiner leur fonctionnement dans les diapositives suivantes.

## L'instruction for

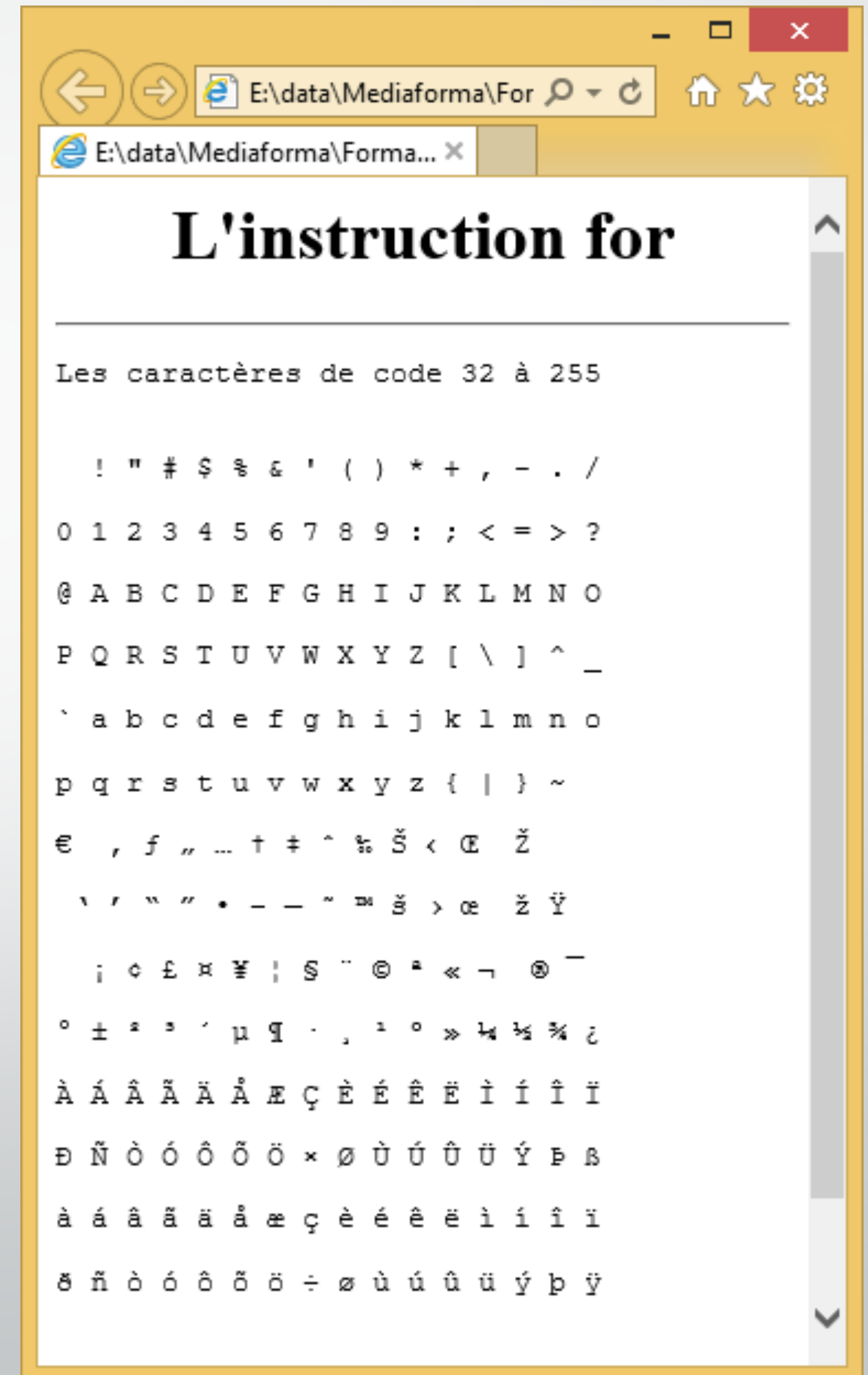
L'instruction for exécute de façon répétitive un bloc d'instructions tant qu'une condition est vérifiée. Cette condition est basée sur la valeur d'une variable incrémentée par l'instruction. Voici la syntaxe de l'instruction for :

```
for (expression 1; expression 2; expression 3) {  
    instruction 1;  
    ...  
    instruction N;  
}
```

- expression 1 initialise la variable compteur utilisée dans la boucle ;
- expression 2 définit la condition qui doit être vérifiée pour que la boucle se poursuive ;
- expression 3 indique comment faire varier la variable compteur utilisée dans expression 2 ;
- instruction 1 à instruction N sont les instructions ou appels de fonctions exécutés tant que l'expression 2 est vérifiée.

Exercice :

Utilisez une boucle for pour afficher les caractères de code ASCII 32 à 255, comme dans la copie d'écran ci-contre :





Solution :

```
<script language="JavaScript">
  document.write("<h1>L'instruction for </h1><hr>");
  document.write("Les caractères de code 32 à 255 \n\n");
  document.write("<pre>");
  for (var i=32; i<256; i++){
    if ((i % 16) == 0){
      document.write("<br>");
    }
    document.write("&#" + i + "; ");
  }
  document.write("</pre>");
</script>
```

Remarque :

La balise `<pre></pre>` permet d'afficher des caractères de largeur fixe, et donc d'aligner verticalement les caractères affichés sur l'écran.

## Un cas particulier de l'instruction for

L'instruction for ... in accède séquentiellement aux propriétés d'un objet. Sa syntaxe est la suivante :

```
for (var in objet) {  
    instruction 1;  
    ...  
    instruction N;  
}
```

- var est la variable compteur utilisée pour décrire l'objet ;
- objet est un objet quelconque dont on désire connaître les propriétés ;
- instruction 1 à instruction N sont les instructions ou appels de fonctions exécutés pour chaque propriété de l'objet.

Exemple : Passage en revue des cellules d'un tableau

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  <script language="JavaScript">
    var t = ["Paris", "Londres", "Amsterdam",
"Madrid"];
    for (var i in t){
      document.write(t[i] + '<br>');
    }
  </script>
</body>
</html>
```



## L'instruction while

L'instruction while exécute de façon répétitive un bloc d'instructions tant qu'une condition est vérifiée. Le test de la condition est fait avant l'exécution de la boucle. Voici la syntaxe de l'instruction while :

```
while (expression) {  
    instruction 1;  
    ...  
    instruction N;  
}
```

où expression est une condition qui doit être vérifiée pour que les instructions de la boucle s'exécutent et instruction 1 à instruction N les instructions ou appels de fonctions exécutés tant que l'expression est vérifiée.



## Réaliser une boucle infinie

Pour définir une boucle infinie (qui boucle sans fin), vous utiliserez l'une des deux instructions suivantes :

```
for ( ; ; ){  
    instructions;  
}
```

Ou :

```
while (1){  
    instructions;  
}
```

## L'instruction continue

Il est parfois nécessaire d'interdire l'exécution d'une boucle lorsqu'une condition logique est satisfaite. Pour cela, vous utiliserez l'instruction **continue**, en tête du bloc :

Dans une instruction for :

```
for (expression 1; expression 2; expression 3) {  
    if (condition) {  
        continue;  
    }  
    instruction 1;  
    ...  
    instruction N;  
}
```

Dans une instruction while :

```
while (expression) {  
    if (condition) {  
        continue;  
    }  
    instruction 1;  
    ...  
    instruction N;  
}
```



Exemple :

Le code de la diapositive suivante calcule le résultat de la fonction  $y = 1/(x - 3)$  pour des valeurs comprises entre 2 et 4, par pas de 0.1. Il est nécessaire d'exclure la valeur 3 du calcul qui donne un résultat indéterminé. Une instruction **continue** s'en charge.

Saisissez et testez le code suivant :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script language = "JavaScript">
function calcul() {
    document.write("<PRE>");
    var x;
    document.write("<B>Résultats :</B>\n\n");
    for (x = 2; x<= 4; x = x+0.1)
    {
        if (((x-3)<0.01) && ((x-3)>-0.01))
            continue; // Interdit l'exécution de la boucle pour x = 3
        document.write("x = " + Math.round(x*10)/10,"\\t   y = " + Math.round(1/(x-3)*100)/100,"\\n");
    }
    document.write("</PRE>");
}
</script>
</head>
<body>
<h1>Test de l'instruction continue</h1>
<hr>
<form>
Appuyez sur le bouton Calcul pour calculer les valeurs de la fonction  $y = 1/(x - 3)$  pour x compris entre
2 et 4
<input type="button" name="Calcul" value="Calcul" onclick="calcul();">
</form>
</body>
</html>
```



## L'instruction break

Il est parfois nécessaire d'arrêter l'exécution de la boucle lorsqu'une condition logique est satisfaite. Pour cela, utilisez l'instruction break en tête du bloc :

Dans une instruction for :

```
for (expression 1; expression 2; expression 3) {  
    if (condition) {  
        break;  
    }  
    instruction 1;  
    ...  
    instruction N;  
}
```

Dans une instruction while :

```
while (expression) {  
    if (condition) {  
        break;  
    }  
    instruction 1;  
    ...  
    instruction N;  
}
```

# ***Des commentaires dans le texte***

Quel que soit le langage utilisé, il est toujours intéressant d'insérer des commentaires dans un programme. Cela facilite sa maintenance et précise les points épineux.

Comme nous l'avons déjà vu, la balise suivante permet d'insérer des commentaires dans un document HTML.

```
<!-- Ceci est un commentaire -->
```

Si les commentaires doivent être placés entre les balises <script> et </script>, vous procéderez différemment :

- Pour insérer une ligne de commentaire, faites-la commencer par un double slash.
- Pour insérer plusieurs lignes de commentaire consécutives, encadrez-les par les caractères /\* et \*/.

# ***L'instruction return***

Une fonction JavaScript est en mesure de renvoyer une valeur calculée ou issue d'une variable. Pour cela, il suffit d'utiliser l'instruction return à l'intérieur de la fonction.

Exercice :

Définissez un formulaire contenant deux zones de texte et un bouton.

L'utilisateur entre une valeur hexadécimale dans la première zone de texte et appuie sur le bouton. La valeur décimale correspondante est alors affichée dans la deuxième zone de texte.



**Test de l'instruction return**

Entrez la valeur hexa à convertir, puis appuyez sur le bouton

13 -> 19

Pour effectuer une conversion hexa -> decimal, vous utiliserez la fonction `parseInt` :

```
parseInt (St, 16) ;
```

## Solution :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<h1>Test de l'instruction return</h1>
<hr>
<form>
  Entrez la valeur hexa à convertir, puis appuyez sur le bouton
  <input type="text" name="Hex" size="4">
  <input type="button" value="->" onclick="Conversion(this.form);">
  <input type="text" name="Dec" size="4">
</form>
<script language = "JavaScript">
  function HexDec(St){
    return parseInt(St,16);
  }
  function Conversion(form){
    var valeur = HexDec(form.Hex.value);
    form.Dec.value = valeur;
  }
</script>
</body>
</html>
```



# Définition d'un tableau en JavaScript Objet

La définition d'un tableau peut se faire de façon traditionnelle ou en instanciant la classe **Array**.

De façon traditionnelle :

```
var t = [];
```

En objet :

```
var t = new Array;
```

Pour définir les éléments d'un tableau lors de sa création.

De façon traditionnelle :

```
var t = ['chien', 'chat', 'cochon'];
```

En objet :

```
var t = new Array('chien', 'chat', 'cochon');
```

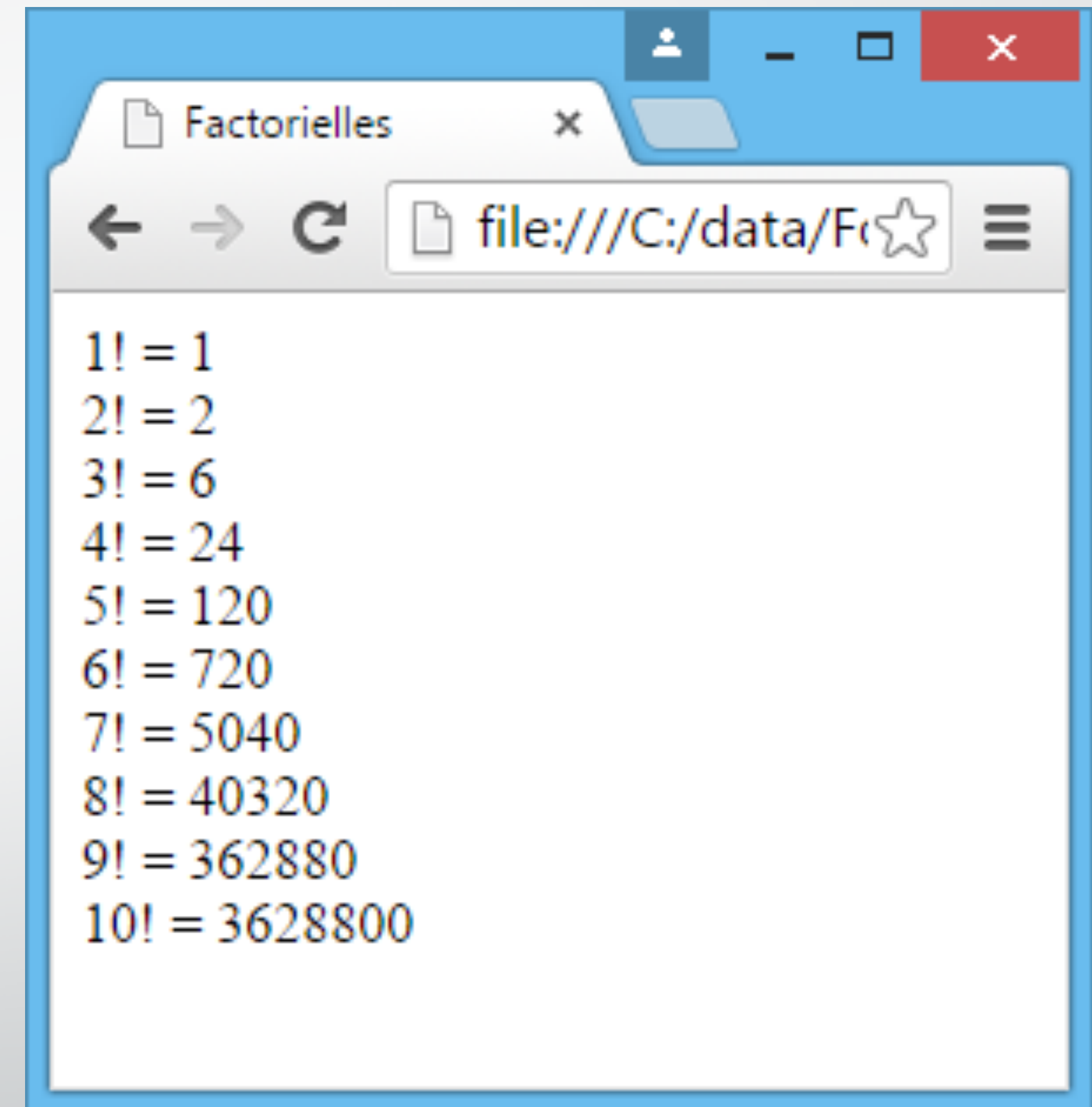


## Exercice

Définissez le code HTML et JavaScript pour calculer les factorielles de 1! à 10!.  
Stockez-les dans un tableau et affichez le contenu de ce tableau.

# Solution

```
<html>
<head>
  <meta charset="utf-8">
  <title>Factorielles</title>
</head>
<body>
  <script>
    var t = new Array;
    t[1]=1;
    for (var i=2; i<=10; i++)
      t[i] = t[i-1]*i;
    for (i=1; i<=10; i++)
      document.write(i + '!' = ' + t[i] + '<br>');
  </script>
</body>
</html>
```



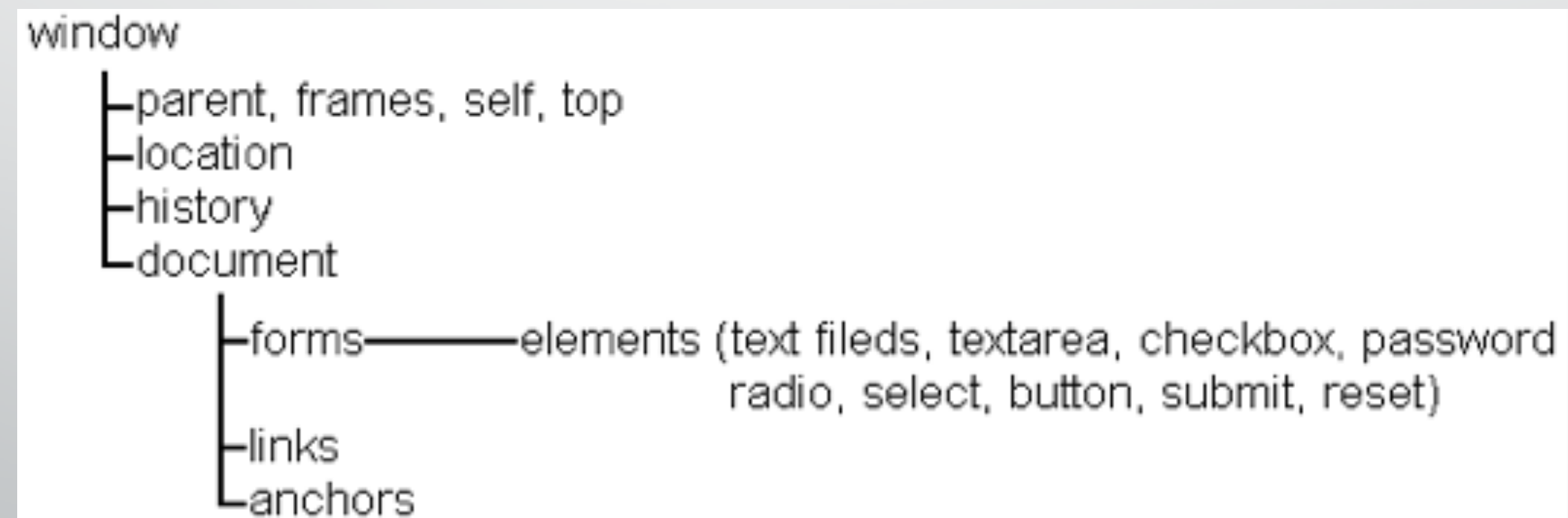
# Les objets standards de JavaScript

JavaScript est en mesure de manipuler divers types d'objets. Nous allons les passer en revue dans les pages qui suivent.

## Les objets liés au navigateur

Lorsqu'une page est chargée dans le navigateur, plusieurs objets sont automatiquement créés :

- **window** : la fenêtre affichée dans le navigateur.
- **location** : l'URL courante et ses propriétés.
- **history** : les URL déjà visitées.
- **document** : donne accès aux propriétés du document courant (titre, couleur d'arrière-plan, etc.).





Supposons qu'une page HTML nommée hexdec.htm soit disponible à cette URL : <http://www.mediaforma.com/hexdec.htm>. Voici son code :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Conversion Hexa->Décimal</TITLE>
  <script language = "JavaScript">
    function HexDec(St) {
      return parseInt(St,16);
    }
    function Conversion(form) {
      valeur = HexDec(form.Hex.value);
      form.Dec.value = valeur;
    }
  </script>
</head>
<body>
  <h1>Conversion hexa -> decimal</h1>
  <hr>
  <form name = "Donnees">
    Entrez la valeur hexa à convertir, puis appuyez sur le bouton
    <input type = "text" name = "Hex" size = 4>
    <input type = "button" name = "Action" value = "->" onclick = "Conversion(this.form)">
    <input type = "text" name = "Dec" size = 4>
  </form>
</body>
</html>
```

Les objets standard location, history et document sont automatiquement créés lors du chargement de ce document. Voici quelques-unes de leurs propriétés :

```
location.href = "http://www.google.fr"
```

```
document.title = "Conversion Hexa->Décimal"
```

```
document.body.style.color = #000000
```

```
document.body.style.backgroundColor = #ffffff
```

```
history.length = 5 (par exemple)
```

```
document.Donnees.Hex.name = "Hex"
```

```
document.Donnees.Hex.value = "3F" (par exemple)
```

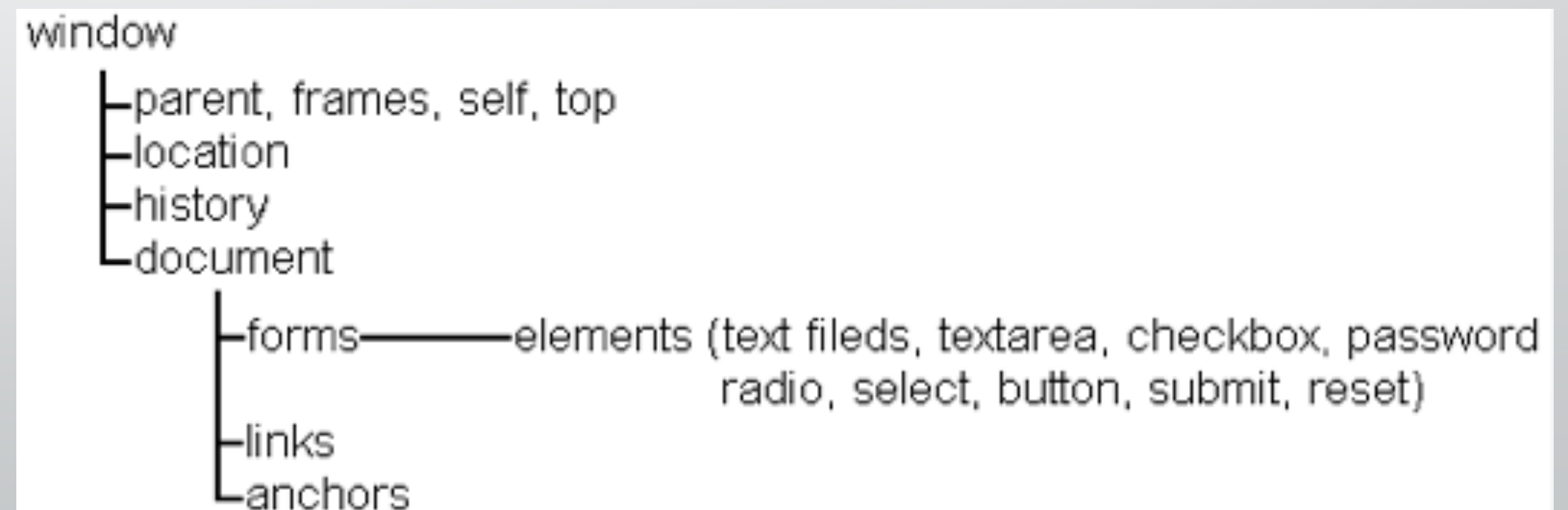
```
document.Donnees.Action.value = "->"
```


```
document.Donnees.Dec.value = 63 (par exemple)
```

Comme vous le voyez, l'accès aux objets se fait à l'aide d'instructions "à point". Ces instructions respectent la hiérarchie de la figure, en omettant toutefois l'objet window. Ainsi, par exemple, l'entité document.Donnees.Action.value représente la propriété value du contrôle Action. Ce contrôle fait partie du formulaire Données, qui est lui-même un sous-ensemble de l'objet document.

Cet exemple contient un formulaire qui a été nommé à l'aide du marqueur suivant :

```
<form name = "Donnees">  
</form>
```





Il est ainsi possible d'accéder aux éléments du formulaire à l'aide de l'entité document.Donnees. Lorsque le formulaire n'a aucun nom, il est cependant représenté par le tableau forms[]. Le premier formulaire rencontré dans la page est appelé forms[0], le deuxième forms[1], et ainsi de suite.

Pour accéder à la propriété value du contrôle Action, vous pouvez donc utiliser indifféremment les deux entités suivantes (l'indice 0 laisse supposer que le formulaire est le premier de la page) :

```
document.Donnees.Action.value
```

```
document.forms[0].Action.value
```



# Les autres objets

Nous allons maintenant nous intéresser aux autres objets accessibles au langage JavaScript. Dans la mesure du possible, nous définirons un ou plusieurs exemples pour faciliter leur compréhension.

## *L'objet anchor*

Les anchors permettent de définir des signets qui pointent sur les destinations à accéder ou des liens hypertexte vers des signets. Reportez-vous au paragraphe intitulé "Liens hypertexte" dans la première partie de cet ouvrage pour plus d'informations sur ces marqueurs.

La syntaxe HTML permettant de définir un anchor est la suivante :

```
<a [href = "#Signet"] name = "Nom" [target = "Nom de la fenêtre"]>
  Texte
</a>
```

- Signet est le nom du signet à atteindre.
- Nom est le nom donné au signet.
- Nom de la fenêtre est le nom de la fenêtre dans laquelle est défini le lien. Ce paramètre n'a un sens que dans le cas où le marqueur est utilisé pour définir un signet.
- Texte est le texte utilisé pour définir le signet ou le lien hypertexte.

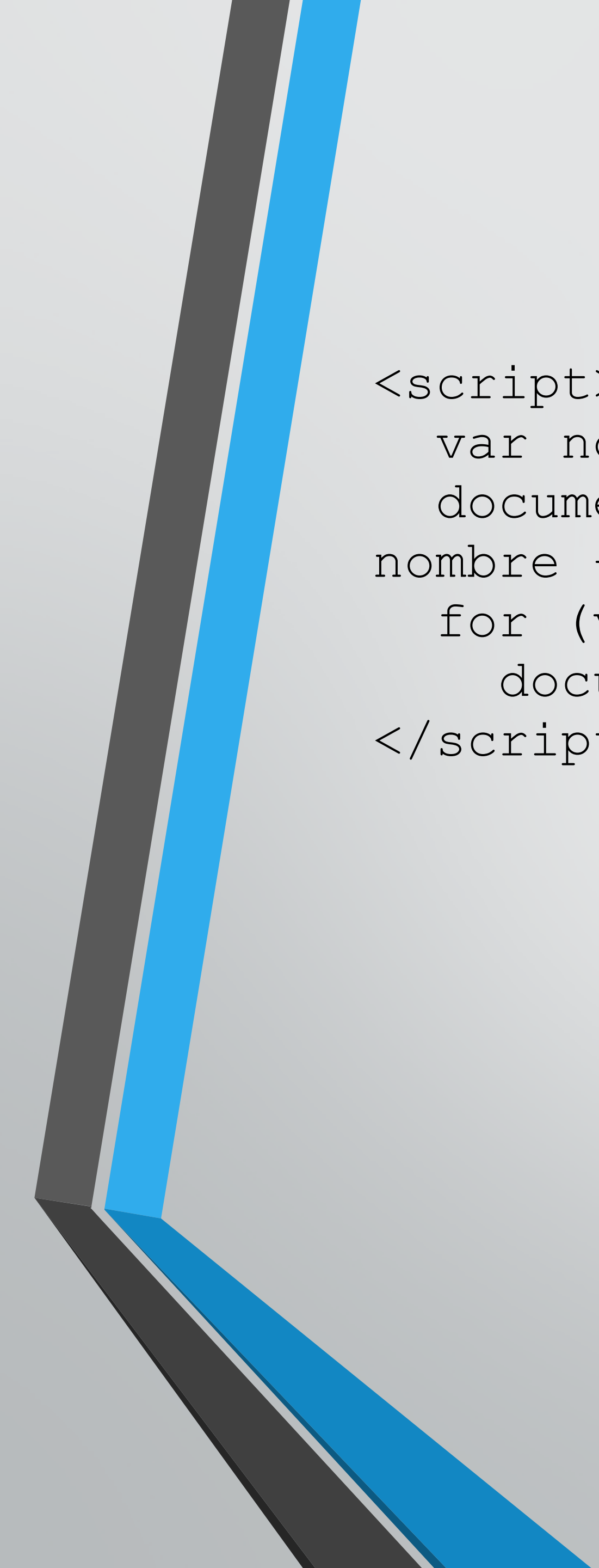
Lorsqu'un document contient des marqueurs , il est possible d'utiliser la propriété document.anchors.length pour en connaître le nombre.

Lorsqu'un marqueur est utilisé pour définir un lien hypertexte, il est référencé dans les tableaux anchors[] et links[].

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <a name="Aligner"><H1>Alignement</H1></a>
    <p align="LEFT">Ce texte est aligné à gauche</p>
    <p align="CENTER">Ce texte est centré</p>
    <p align="RIGHT">Ce texte est aligné à droite</p>

    <a name="Attrib"><H1>Attributs de caractères</H1></a>
    Le mot <B>gras</B> est en gras, le mot <I>italique</I> en italique.
  </body>
</html>
```

Définissez le code nécessaire pour afficher le nombre d'ancres et les ancres



```
<script>
  var nombre=document.anchors.length;
  document.write('Nombre de balises &lt;a&gt; dans le document : ' +
nombre + '<br>');
  for (var i=0; i<=nombre; i++)
    document.write(document.anchors[i].innerHTML + '<br>');
</script>
```

## *L'objet button*

Les objets button correspondent aux boutons de commande placés dans les formulaires HTML.

La syntaxe permettant de définir un bouton est la suivante :

```
<input type="button" name="Nom" value="Valeur" [ONCLICK="Traitement"]>
```

- Nom est le nom du bouton ;
- Valeur est le texte affiché sur le bouton ;
- Traitement est une éventuelle instruction ou fonction de traitement JavaScript activée lorsque le bouton est cliqué.



Deux propriétés sont accessibles en lecture et en écriture au code JavaScript : name et value. En d'autres termes, le nom et la légende du bouton. Pour ce faire, vous utiliserez les syntaxes ci-après :

```
document.NomFormulaire.NomBouton.name
```

```
document.NomFormulaire.elements[N].name
```

```
document.NomFormulaire.NomBouton.value
```

```
document.NomFormulaire.elements[N].value
```

- NomFormulaire est le nom du formulaire donné dans le marqueur <form> ;
- NomBouton est le nom du bouton donné dans le marqueur <input> ;
- N est le numéro d'ordre du bouton dans le formulaire.

## ***L'objet checkbox***

Les objets checkbox correspondent aux cases à cocher placées dans les formulaires HTML. La syntaxe permettant de définir un objet checkbox est la suivante :

```
<input type="checkbox" name="NOM" value="Valeur"  
      [checked] [onclick="Traitement"]>
```

Légende de la case à cocher

- Nom est le nom de la case à cocher ;
- Valeur est la valeur renvoyée au serveur lorsque la case est cochée ;
- [checked] (s'il est précisé) coche la case par défaut ;
- Traitement est une éventuelle instruction ou fonction de traitement JavaScript activée lorsque l'utilisateur clique sur la case.

Quatre propriétés sont accessibles au code JavaScript : checked, defaultChecked, name et value. Elles correspondent à l'état du bouton radio, à l'état par défaut, au nom et à la valeur du bouton radio.

Pour accéder à ces propriétés, vous utiliserez les syntaxes ci-après :

```
document.NomFormulaire.NomBouton.checked  
document.NomFormulaire.NomBouton.defaultChecked  
document.NomFormulaire.NomBouton.name  
document.NomFormulaire.NomBouton.value
```

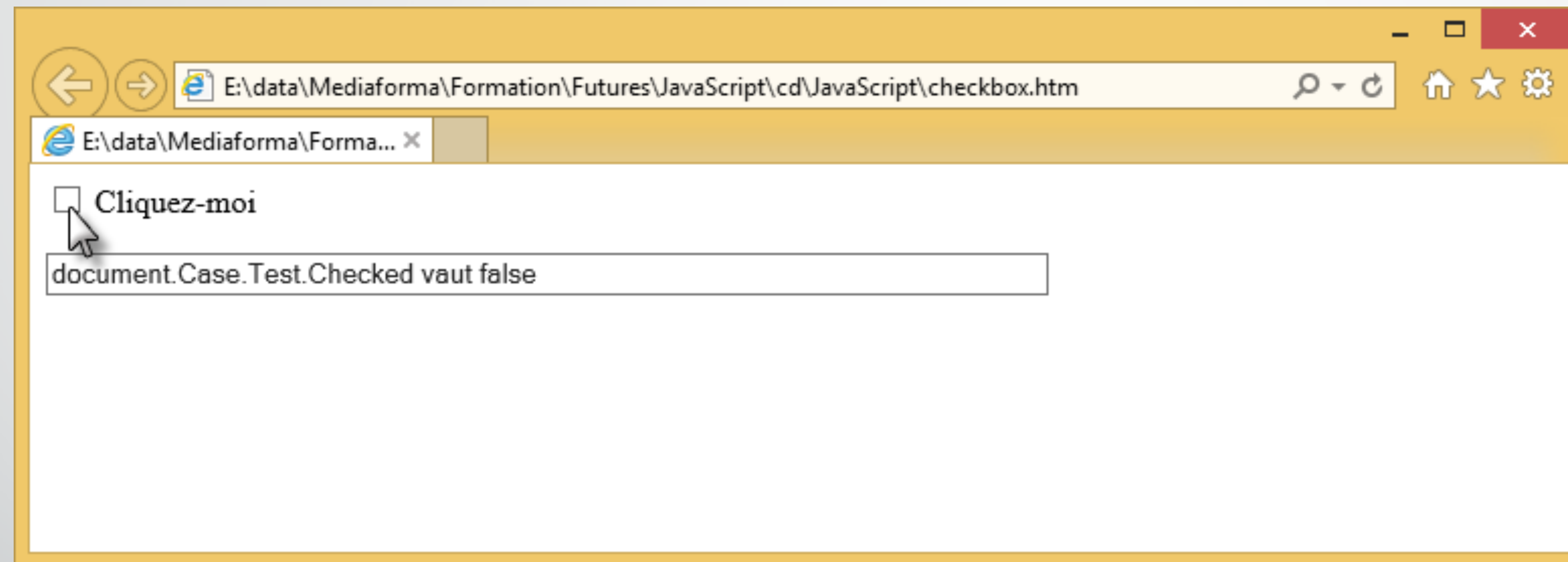
ou encore :

```
document.NomFormulaire.elements[N].checked  
document.NomFormulaire.elements[N].defaultChecked  
document.NomFormulaire.elements[N].name  
document.NomFormulaire.elements[N].value
```

- NomFormulaire est le nom du formulaire tel qu'il apparaît dans le marqueur <form> ;
- NomBouton est le nom du bouton tel qu'il apparaît dans le marqueur <input> ;
- N est le numéro d'ordre du bouton dans le formulaire. La méthode click() est associée aux objets checkbox. Reportez-vous à l'Annexe 2 pour savoir comment l'utiliser.

## Exercice

Ecrivez le code HTML et JavaScript nécessaire pour afficher la valeur de la propriété Checked d'une case à cocher dans une zone de texte :





## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <form name="Case">
      <input type="checkbox" value="Case" name="Test" onclick="cli(this.form)">
      Cliquez-moi
      <input type="text" name="resultat" size="80">
    </form>
    <script>
      function cli(form){
        form.resultat.value = "document.Case.Test.checked vaut "+document.Case.Test.checked;
      }
    </script>
  </body>
</html>
```

## ***L'objet document***

L'objet document donne accès aux propriétés du document courant. Le code HTML permettant de définir un document est placé entre les marqueurs <body> et </body> :

```
<body>
```

```
    Corps du document
```

```
</body>
```

Le marqueur <body> peut comporter de nombreux paramètres :

```
<body  
  [background] = Image ou URL,  
  [bgcolor] = "#Couleur arrière-plan"  
  [text] = "#Couleur du texte"  
  [link] = "#Couleur des liens"  
  [alink] = "#Couleur du lien actif"  
  [vlink] = "#Couleur des liens visités"  
  [onLoad = "Instruction"]  
  [onUnload = "Instruction"]>
```

Tous les paramètres sont facultatifs.

- background définit le nom de l'image à afficher sur l'arrière-plan ou son URL ;
- bgcolor définit la couleur uniforme de l'arrière-plan ;
- text, link, alink et vlink définissent respectivement la couleur du texte, des liens, du lien actif et des liens déjà visités ;
- OnLoad et onUnload font référence à une instruction ou à une fonction JavaScript exécutée (respectivement) au chargement de la page et au passage à la page suivante.



Pour accéder à un objet document, vous utiliserez l'une des deux syntaxes suivantes :

`document.propriété`

ou

`document.méthode (paramètres)`

- propriété est une propriété du document (alinkColor, anchors, bgColor, cookie, fgColor, forms, lastModified, linkColor, links, location, referrer, title ou vlink) ;
- méthode est une méthode applicable à l'objet document (clear, close, open, write ou writeln) ;
- paramètres correspond aux éventuels paramètres passés à la méthode.



Affichez un texte H1 de couleur cyan. Définissez un formulaire comportant deux zones de texte et deux boutons. Définissez le code HTML et JavaScript nécessaire pour :

- 1) afficher la couleur hexadécimale du texte dans la première zone de texte lorsque l'utilisateur clique sur le premier bouton
- 2) Modifier la couleur d'arrière-plan de la page en utilisant le code couleur de la deuxième zone de texte lorsque l'utilisateur clique sur le deuxième bouton.



## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre du document</title>
    <script language = "JavaScript">
      function Lit_Couleur_Texte(form) {
        form.Texte.value = document.fgColor;
      }
      function Change_Couleur_Fond(form) {
        document.bgColor = form.Fond.value;
      }
    </script>
  </head>
  <body text = "cyan">
    <h1>Les propriétés fgColor et bgColor</h1>
    <hr>
    <form name="leform">
      <input type="text" name="Texte" SIZE="25">
      <input type="button" value="Lit couleur texte" onclick="Lit_Couleur_Texte(this.form) ">
      <input type="text" name="Fond" SIZE="25">
      <input type="button" value="Modifie couleur fond" onclick="Change_Couleur_Fond(this.form) ">
    </form>
  </body>
</html>
```

## ***Le tableau `elements[]`***

Les contrôles qui composent un formulaire peuvent être référencés à l'aide du tableau **`elements[]`**.

Supposons qu'un formulaire nommé Test comporte :

- Une zone de texte de nom Saisie.
- Une case à cocher de nom Option.
- Un bouton de commande de nom Action.

Ces contrôles sont accessibles à l'aide des instructions suivantes :

```
Test.Saisie.value
```

```
Test.Option.value
```

```
Test.Action.value
```


Mais également par l'intermédiaire du tableau `elements[]` :

```
Test.elements[0].value
```

```
Test.elements[1].value
```

```
Test.elements[2].value
```

Ce deuxième jeu d'instructions suppose que le code HTML a défini, dans l'ordre, les contrôles Saisie, Option puis Action.



En utilisant la propriété length, vous pouvez connaître le nombre d'entrées dans le tableau elements[], c'est-à-dire le nombre de contrôles définis dans le formulaire.

Pour reprendre l'exemple précédent, l'expression :

```
Test.elements.length
```

renvoie la valeur 3. Cela signifie que trois contrôles ont été définis dans le formulaire.

### Exercice

En partant de l'exercice précédent, ajoutez un bouton de commande dans le formulaire et affichez la valeur des quatre contrôles contenus dans le formulaire.



# Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre du document</title>
    <script language = "JavaScript">
      function Lit_Couleur_Texte(form){
        form.Texte.value = document.fgColor
      }
      function Change_Couleur_Fond(form){
        document.bgColor = form.Fond.value
      }
      function Affiche_Contrôles(form){
        var nbControls = form.elements.length;
        document.write("<pre>Il y a ", nbControls, " contrôles dans ce formulaire.<p>");
        document.write("Voici leurs valeurs :<br>");
        for (i=0; i<nbControls; i++) {
          document.write("Contrôle n° " + i + " : " + form.elements[i].value + "<br>");
        }
      }
    </script>
  </head>

  <body text = "cyan">
    <h1>Les propriétés fgColor et bgColor</h1>
    <hr>
    <form name="leform">
      <input type="text" name="Texte" size="25">
      <input type="button" value="Lit couleur texte" onclick="Lit_Couleur_Texte(this.form);">
      <input type="text" name="Fond" size="25">
      <input type="button" value="Modifie couleur fond" onclick="Change_Couleur_Fond(this.form);">
      <input type="button" value="Valeur des contrôles" onclick="Affiche_Contrôles(this.form);">
    </form>
  </body>
</html>
```



## *L'objet form*

Les formulaires sont utilisés pour afficher des contrôles évolués sur une page HTML (zones de texte, boutons radio, cases à cocher, etc.) afin que l'utilisateur puisse entrer des données. Ces dernières seront communiquées au serveur par un appui sur le bouton de commande submit.

La syntaxe permettant de définir un objet form est la suivante :

```
<form  
  name = "Nom"  
  target = "Fenêtre"  
  action = Traitement  
  method = {get|post}  
  enctype = "Type d'encodage"  
  onSubmit = "Fonction">  
</form>
```

- name est le nom du formulaire ;
- target est la fenêtre dans laquelle doit être affiché le résultat renvoyé par le serveur (il peut s'agir d'une fenêtre ou d'un cadre existant) ;
- action est le nom d'un programme de traitement, par exemple /cgi-bin/Trait.pl ou encore l'adresse de votre boîte aux lettres, précédée de mailto. Par exemple <mailto:pierre.durand@Bertold.fr> ;
- method définit la méthode à utiliser pour prendre en charge les données du formulaire. La méthode get() étant limitée quant à la quantité de données transmissibles, la méthode post() est presque toujours utilisée ;
- enctype est le type d'encodage utilisé pour transmettre les données ;
- onSubmit spécifie le nom de la fonction JavaScript activée lors d'un appui sur le bouton Submit.



Pour accéder à un formulaire en JavaScript, vous utiliserez l'une des deux syntaxes suivantes :

```
Nom_formulaire.propriété  
forms[index].propriété
```

où Nom\_formulaire est la valeur affectée au paramètre NAME dans le marqueur <FORM>, et propriété l'une des propriétés suivantes :

- action : correspond au paramètre ACTION.
- elements[] : tableau qui fait référence aux contrôles du formulaire.
- encoding : correspond au paramètre ENCTYPE.
- length : nombre de contrôles dans le formulaire.
- method : correspond au paramètre METHOD.
- target : correspond au paramètre TARGET.
- index est le numéro du formulaire à atteindre.

Dans la deuxième syntaxe, le terme "propriété" ne peut prendre que la valeur length.



## ***L'objet hidden***

Bien qu'ils fassent partie d'un formulaire, les objets hidden n'apparaissent pas sur l'écran. Ils sont utilisés pour envoyer le nom et la valeur d'une variable au serveur sans en informer l'utilisateur final.

La syntaxe permettant de définir un objet hidden est la suivante :

```
<input  
  type = "hidden"  
  name = "Nom"  
  value = "Texte">
```

où Nom est le nom de l'objet, et Texte sa valeur initiale.

Pour accéder à un objet hidden en JavaScript, vous utiliserez l'une des deux syntaxes suivantes :

```
Nom.Propriété  
NomFormulaire.elements[index].Propriété
```

- Nom est le nom de l'objet ;
- Propriété est la propriété à atteindre (name ou value) ;
- NomFormulaire est le nom du formulaire où se trouve l'objet hidden (variable name dans le marqueur <form>) ;
- index est le numéro d'ordre de l'objet hidden dans le formulaire.

## *L'objet history*

L'objet history contient les URL des divers sites visités.

Pour connaître le nombre d'entrées dans l'objet history, vous utiliserez l'expression history.length.

Pour vous déplacer dans l'objet history, vous utiliserez la méthode back() (vers l'arrière), forward() (vers l'avant) ou go() (saut chiffré) dans une expression du type :

```
history.méthode ([paramètre]) ;
```

La syntaxe la plus simple est la suivante :

```
history.back() ;
```

Elle affiche la page précédemment visitée dans la fenêtre active.

## Exercice :

Entraînez-vous à manipuler l'objet history.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre du document</title>
  </head>

  <body>
    <h1>L'objet history</h1>
    <hr>
    <a href="test.htm">test</a>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre du document</title>
  </head>

  <body>
    <h1>L'objet history</h1>
    <hr>
    <a href="javascript:history.back()">Page Précédente</a>
    <form>
      <input type="button" value="Précédent"
        onclick="javascript:history.back()">
    </form>
  </body>
</html>
```



## *L'objet link*

Les objets link définissent un lien hypertexte ou hypermédia. Lorsque l'utilisateur clique sur un tel objet, le document HTML référence est chargé.



La définition d'un objet link se fait à l'aide du marqueur <a> </a> :

```
<a  
  href = "URL"  
  name = "Nom du lien"  
  target = "Nom de la fenêtre"  
  onClick = "fonction"  
  onMouseOver = "fonction">  
  Texte  
</a>
```

- url est l'adresse du lien à atteindre ;
- nom définit un signet sur le texte spécifié entre les marqueurs <a> et </a> ;
- target indique le nom de la fenêtre dans laquelle doit s'afficher le document lié ;
- onClick déclenche une instruction ou une fonction JavaScript lorsque l'utilisateur clique sur le lien ;
- onMouseOver déclenche une instruction ou une fonction JavaScript lorsque le pointeur de la souris se trouve au-dessus du lien.



Pour accéder à un objet link, vous utiliserez le tableau links[] comme suit :

```
document.links[index]
```

où index est le numéro d'ordre de l'objet link sur la page.

Vous pouvez connaître le nombre de liens contenus dans la page en utilisant la propriété length de l'objet link :

```
document.links.length
```

Les objets link possèdent de nombreuses propriétés accessibles à travers le tableau links[] :

- hash : fait référence au nom du lien (NAME).
- host : nom de l'URL visée (une partie de HREF).
- hostname : nom et domaine de l'URL visée (une partie de HREF).
- href : URL visée (HREF).
- pathname : path de l'URL (une partie de HREF).
- port : port de communication utilisé par le serveur.
- protocol : début de l'URL (une partie de HREF).
- search : l'éventuelle question posée dans l'URL (une partie de HREF).
- target : nom de la fenêtre (TARGET).

### Exercice

Utilisez les propriétés de l'objet link pour décortiquer et afficher les différentes parties de l'URL suivante :

<https://www.google.fr/search?site=&source=hp&q=java+javascript&oq=java+javascript>

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre du document</title>
  </head>
  <body>
    <h1>Les propriétés d'un objet link</h1>
    <hr>
    <a href = "https://www.google.fr/search?site=&source=hp&q=java+javascript&oq=java+javascript">
      Google.fr sur les termes java et javascript</a>

    <script language = "JavaScript">
      document.write("<p>Le lien utilisé est <b>https://www.google.fr/search?site=&source=hp&q=java+javascript&oq=java+javascript</b>");

      document.write("<P>document.links[0].hash = ",document.links[0].hash);
      document.write("<BR>document.links[0].host = ",document.links[0].host);
      document.write("<BR>document.links[0].hostname = ",document.links[0].hostname);
      document.write("<BR>document.links[0].href = ",document.links[0].href);
      document.write("<BR>document.links[0].pathname = ",document.links[0].pathname);
      document.write("<BR>document.links[0].port = ",document.links[0].port);
      document.write("<BR>document.links[0].protocol = ",document.links[0].protocol);
      document.write("<BR>document.links[0].search = ",document.links[0].search);
    </script>
  </body>
</html>
```



## ***L'objet location***

L'objet location fait référence à l'URL courante. Il est défini implicitement par le navigateur. Pour y accéder, vous utiliserez la syntaxe suivante :

`location.Propriété`

où Propriété est l'une des propriétés suivantes :

- protocol : début de l'URL ;
- hostname : nom et domaine de l'URL visé ;
- port : port de communication utilisé par le serveur ;
- href : URL visé ;
- pathname : path de l'URL ;
- search : l'éventuelle question posée dans l'URL ;
- hash : fait référence au nom du lien.

## ***L'objet Math***

L'objet Math donne accès à huit constantes accessibles sous la forme d'une propriété de l'objet Math :

Intitulé	Constante
E	Constante d'Euler (environ 2,718)
LN2	Logarithme népérien de 2 (environ 0,693)
LN10	Logarithme népérien de 10 (environ 2,3026)
LOG2E	Logarithme à base 2 de E (environ 1,442)
LOG10E	Logarithme à base 10 de E (environ 0,434)
PI	Constante PI (environ 3,14159265)
SQRT1_2	Racine carrée de $\frac{1}{2}$ (environ 0,7071)
SQRT2	Racine carrée de 2 (environ 1,4142)

Pour accéder à la constante LOG10E, vous utiliserez l'instruction

`Math.LOG10E`

ou encore le bloc suivant :

```
with (Math) {  
  LOG10E...  
}
```

Cette deuxième version facilite l'écriture si vous devez manipuler de nombreuses constantes et méthodes

L'objet Math donne également accès à dix-sept méthodes :

Méthode	Signification
abs()	Valeur absolue
acos()	Arc cosinus
asin()	Arc sinus
atan()	Arc tangente
ceil()	Entier supérieur ou égal à la valeur spécifiée
cos()	Cosinus
exp()	Exponentielle
floor()	Entier inférieur ou égal à la valeur spécifiée
log()	Logarithme décimal
max()	Valeur maximale
min()	Valeur minimale
pow()	Mise à la puissance
random()	Nombre aléatoire
round()	Arrondi
sin()	Sinus
sqrt()	Racine carrée
tan()	Tangente

## ***L'objet navigator***

Cet objet renvoie des informations sur le type et la version du navigateur utilisé. Il est défini implicitement par le navigateur. Pour y accéder, vous utiliserez la syntaxe suivante :

`navigator.Propriété`

où Propriété est l'une des propriétés suivantes :

- appName : Nom de code du navigateur ;
- appName : Nom du navigateur ;
- appVersion : Version du navigateur ;
- userAgent : Information "userAgent".

Exercice

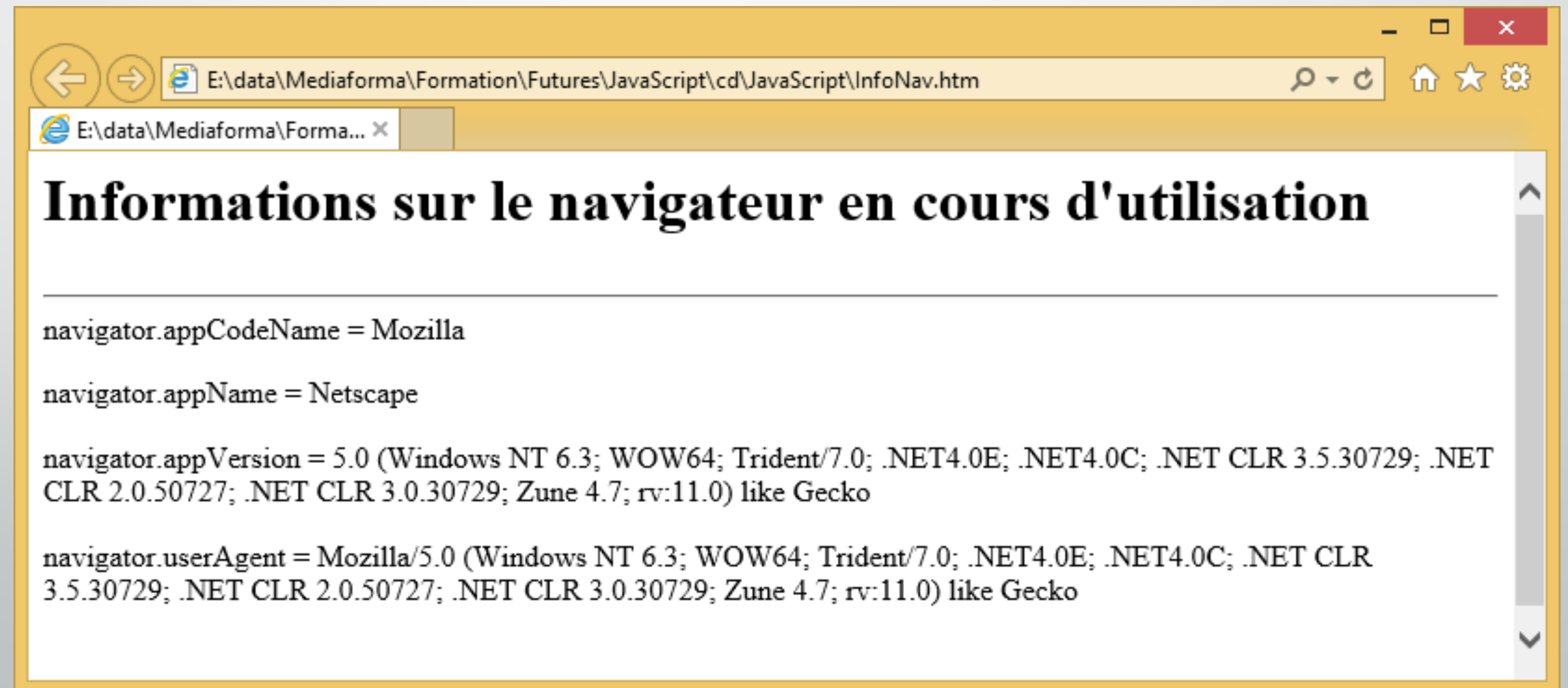
Ecrivez le code nécessaire pour afficher les informations sur votre navigateur Web.



## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Informations sur le navigateur en cours d'utilisation</P></h1>
    <hr>
    <script language = JavaScript>
      document.write("navigator.appCodeName = ",navigator.appCodeName,"<br>");
      document.write("navigator.appName = ",navigator.appName,"<br>");
      document.write("navigator.appVersion = ",navigator.appVersion,"<br>");
      document.write("navigator.userAgent = ",navigator.userAgent,"<br>");
    </script>
  </body>
</html>
```

Un exemple  
d'exécution dans  
IE11 :



## ***L'objet password***

Les objets password sont des champs texte. Ils sont utilisés dans un formulaire lorsque l'information tapée ne doit pas apparaître sur l'écran. Chaque caractère est remplacé par un astérisque.

La syntaxe permettant de définir un objet password est la suivante :

```
<input  
  type = "password"  
  name = "nom"  
  value = "Valeur"  
  size = Taille>
```

- Nom est le nom du champ ;
- Valeur est la valeur par défaut de l'objet password ;
- Taille représente le nombre de caractères qui peuvent être entrés sans provoquer de scrolling.

Les propriétés defaultValue, name et value permettent de connaître respectivement la valeur par défaut (paramètres value), le nom (paramètres name) et la valeur courante de l'objet.



Pour accéder à ces propriétés, vous utiliserez l'une des deux syntaxes suivantes :

`Nom.Propriété`

ou

`Nom_du_formulaire.elements[index].Propriété`

- Nom est le nom de l'objet password ;
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet password ;
- index est le numéro d'ordre de l'objet password dans le formulaire ;
- Propriété est la propriété à accéder.



Vous pouvez également utiliser les méthodes suivantes :

- blur(), pour enlever le focus de l'objet ;
- focus(), pour donner le focus à l'objet ;
- select(), pour sélectionner le contenu de l'objet.

Ces trois méthodes sont accessibles à l'aide des syntaxes suivantes :

`Nom.Méthode()`

ou

`Nom_du_formulaire.elements[index].Méthode()`

- Nom est le nom de l'objet password ;
- Méthode est le nom de la méthode à utiliser (blur, focus ou select) ;
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet password ;
- index est le numéro d'ordre de l'objet password dans le formulaire.



Exercice :

Définissez un formulaire contenant deux zones de texte et donnez le focus à la deuxième

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Focus</h1>
    <hr>
    <form name="leForm">
      <input type="text" name="premier">
      <input type="text" name="deuxieme">
    </form>
    <script language = JavaScript>
      document.forms['leForm'].deuxieme.focus();
    </script>
  </body>
</html>
```

## *L'objet reset*

Dans un formulaire, le bouton reset permet d'affecter les valeurs par défaut à chacun des contrôles.

La syntaxe permettant de définir un bouton reset est la suivante :

```
<INPUT  
  TYPE = "reset"  
  NAME = "Nom"  
  VALUE = "Libellé"  
  [onClick = "Traitement"]>
```

- Nom est le nom du bouton ;
- Libellé est le texte qui apparaît sur le bouton ;
- Traitement est la procédure de traitement JavaScript exécutée lorsque l'utilisateur clique sur le bouton.

Les propriétés name et value donnent accès aux paramètres NAME et VALUE du marqueur INPUT. Pour y accéder, vous pouvez utiliser l'une des deux syntaxes suivantes :

Nom.Propriété

ou

Nom\_du\_formulaire.elements[index].Propriété

- Nom est le nom de l'objet reset ;
- Propriété est la propriété à accéder (name ou value) ;
- Nom\_du\_formulaire est le nom du formulaire qui contient l'objet reset ;
- index est le numéro d'ordre de l'objet reset dans le formulaire.

## *L'objet select*

Les objets select permettent de définir des listes déroulantes et des zones de liste.

Pour définir un objet select, vous utiliserez la syntaxe suivante :

```
<select  
  name = Nom  
  [size = Hauteur]  
  [multiple]  
  [onBlur = "Traitement"]  
  [onChange = "Traitement"]  
  [onFocus = "Traitement"]>  
  <option value = "valeur1" [selected]>  
  <option value = "valeur2" [selected]>  
  ...  
  <option value = "valeurN" [selected]>  
</select>
```

Le paramètre name définit le nom de l'objet.

Le paramètre optionnel size définit la hauteur du champ. Une hauteur égale à 1 définit une liste déroulante. Une valeur supérieure définit une zone de liste.

Lorsque le paramètre multiple est spécifié, il autorise la sélection multiple dans l'objet.

Les paramètres onBlur, onChange et onFocus permettent (respectivement) de lancer une fonction lorsque l'objet select perd le focus, change de valeur et prend le focus.

Enfin, les marqueurs <option> définissent les entrées dans la liste déroulante ou la zone de liste. Le paramètre value indique la valeur retournée au serveur lorsqu'une option particulière est sélectionnée.



Pour accéder aux propriétés d'un objet select, vous utiliserez l'une des syntaxes suivantes :

`Nom.Propriété`

`Nom_du_formulaire.elements[Index].Propriété`

`Nom.options[I].Propriété`

`Nom_du_formulaire.elements[Index].options[i].Propriété`

- Nom est le nom de l'objet select ;
- Propriété est la propriété à accéder ;
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet select (paramètre NAME dans le marqueur <FORM>) ;
- Index est le numéro d'ordre de l'objet dans le formulaire ;
- i est le numéro d'ordre de l'entrée à tester dans la liste déroulante/zone de liste.

Nom.Propriété

Nom\_du\_formulaire.elements[Index].Propriété

Si vous utilisez l'une des deux premières syntaxes, vous aurez accès aux propriétés suivantes :

- length : Nombre d'options dans l'objet select ;
- name : Paramètre NAME de l'objet select ;
- options : Tableau contenant les options de l'objet select ;
- selectedIndex : numéro d'ordre de l'option sélectionnée dans l'objet select. Dans le cas d'une sélection multiple, selectedIndex contient le numéro de la première sélection.

`Nom.options[I].Propriété`

`Nom_du_formulaire.elements[Index].options[I].Propriété`

Si vous utilisez l'une des deux dernières syntaxes, vous aurez accès aux propriétés suivantes :

- defaultSelected : Option sélectionnée par défaut (paramètre selected) ;
- index : Numéro d'ordre de l'option sélectionnée ;
- length : Nombre d'options dans l'objet select ;
- name : Paramètre name de l'objet select ;
- selected : Permet de sélectionner une option.
- selectedIndex : Numéro d'ordre de l'option sélectionnée dans l'objet select. Dans le cas d'une sélection multiple, selectedIndex contient le numéro de la première sélection.
- text : Valeur texte affichée dans un marqueur <option> ;
- value : Valeur du paramètre value.

Supposons qu'un élément select de nom Decision soit défini dans un formulaire de nom Utilisateur :

```
<form name = "Utilisateur">  
  <select name = "Decision">  
    <option selected>Oui</option>  
    <option>Non</option>  
    <option>Peut-être</option>  
  </select>  
</form>
```

Pour connaître le nombre d'options dans l'objet Decision, vous utiliserez l'une des deux instructions suivantes :

```
document.Utilisateur.Decision.length
```

ou

```
document.Utilisateur.elements[0].length
```

La deuxième syntaxe suppose que Decision est le premier objet défini dans le formulaire.

Pour connaître la valeur sélectionnée par défaut dans l'objet Decision, vous utiliserez l'instruction suivante :

```
Utilisateur.elements[0].defaultSelected
```

Enfin, pour connaître la deuxième option possible dans l'objet Decision, vous utiliserez l'instruction suivante :

```
Decision.options[1].text
```



Vous pouvez également utiliser les méthodes blur() et focus() :  
blur() permet d'enlever le focus et focus() donne le focus à l'objet select.

Ces méthodes sont accessibles à l'aide des syntaxes suivantes :

`Nom.Méthode()`

ou

`Nom_du_formulaire.elements[index].Méthode()`

- Nom est le nom de l'objet select ;
- Méthode est le nom de la méthode à utiliser (blur ou focus) ;
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet select ;
- index est le numéro d'ordre de l'objet select dans le formulaire.

## ***L'objet submit***

Le bouton submit permet d'envoyer le contenu des objets d'un formulaire à l'ordinateur distant.

La syntaxe permettant de définir un bouton submit est la suivante :

```
<input  
  type = "submit"  
  name = "Nom"  
  value = "Libellé"  
  [onClick = "Traitement"]>
```

- Nom est le nom du bouton ;
- Libellé est le texte qui apparaît sur le bouton ;
- Traitement est la procédure de traitement JavaScript exécutée lorsque l'utilisateur clique sur le bouton.

Les propriétés name et value donnent accès aux paramètres name et value du marqueur input. Pour y accéder, vous pouvez utiliser l'une des deux syntaxes suivantes :

`Nom.Propriété`

ou

`Nom_du_formulaire.elements[index].Propriété`

- Nom est le nom de l'objet submit ;
- Propriété est la propriété à accéder (name ou value) ;
- Nom\_du\_formulaire est le nom du formulaire qui contient l'objet submit ;
- index est le numéro d'ordre de l'objet submit dans le formulaire.

## ***L'objet text***

Les objets text sont utilisés pour définir des zones de texte dans un formulaire. Pour définir un objet text, vous utiliserez un marqueur <input> du type suivant :

```
<input
  type = "text"
  name = "Nom"
  value = "Valeur"
  size = "Taille1"
  [onBlur = "Traitement1"]
  [onChange = "Traitement2]"
  [onFocus = "Traitement3]"
  [onSelect = "Traitement4"]>
```

- name est le nom de l'objet text ;
- value est la valeur par défaut initialement affichée dans l'objet texte ;
- size est la taille en caractère de l'objet ;
- les TraitementN sont les fonctions JavaScript associées aux événements suivants :
- onBlur() : Perte de focus.
- onChange() : Changement de valeur.
- onFocus() : Prise de focus.
- onSelect() : Sélection du texte.



Pour accéder aux propriétés d'un objet text, vous utiliserez l'une des deux syntaxes suivantes :

`Nom.Propriété`

ou

`Nom_du_formulaire.elements[Index].Propriété`

- Nom est le nom de l'objet text ;
- Propriété est la propriété à accéder,
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet text (paramètre name dans le marqueur <form>) ;
- index est le numéro d'ordre de l'objet text dans le formulaire.

Les propriétés d'un objet text sont :

- defaultValue : valeur par défaut (paramètre value) ;
- name : nom de l'objet (paramètre name) ;
- value : valeur courante de l'objet.

# ***L'objet textarea***

Les objets textarea sont utilisés pour définir des zones de texte multiligne dans un formulaire. Pour définir un objet textarea, vous utiliserez un marqueur <input> du type suivant :

```
<textarea
  type = "textarea"
  name = "Nom"
  rows = "NbLignes"
  cols = "NbColonnes"
  [onBlur = "Traitement1"]
  [onChange = "Traitement2]"
  [onFocus = "Traitement3]"
  [onSelect = "Traitement4"]>
  Texte
</textarea>
```

- name est le nom de l'objet ;
- rows et cols définissent le nombre de lignes et de colonnes de l'objet ;
- Texte est le texte à afficher par défaut dans la zone de texte ;
- les TraitementN sont les fonctions JavaScript associées aux événements suivants :
- onBlur : Perte de focus.
- onChange : Changement de valeur.
- onFocus : Prise de focus.
- onSelect : Sélection du texte.

Pour accéder aux propriétés d'un objet textarea, vous utiliserez l'une des deux syntaxes suivantes :

`Nom.Propriété`

ou

`Nom_du_formulaire.elements[Index].Propriété`

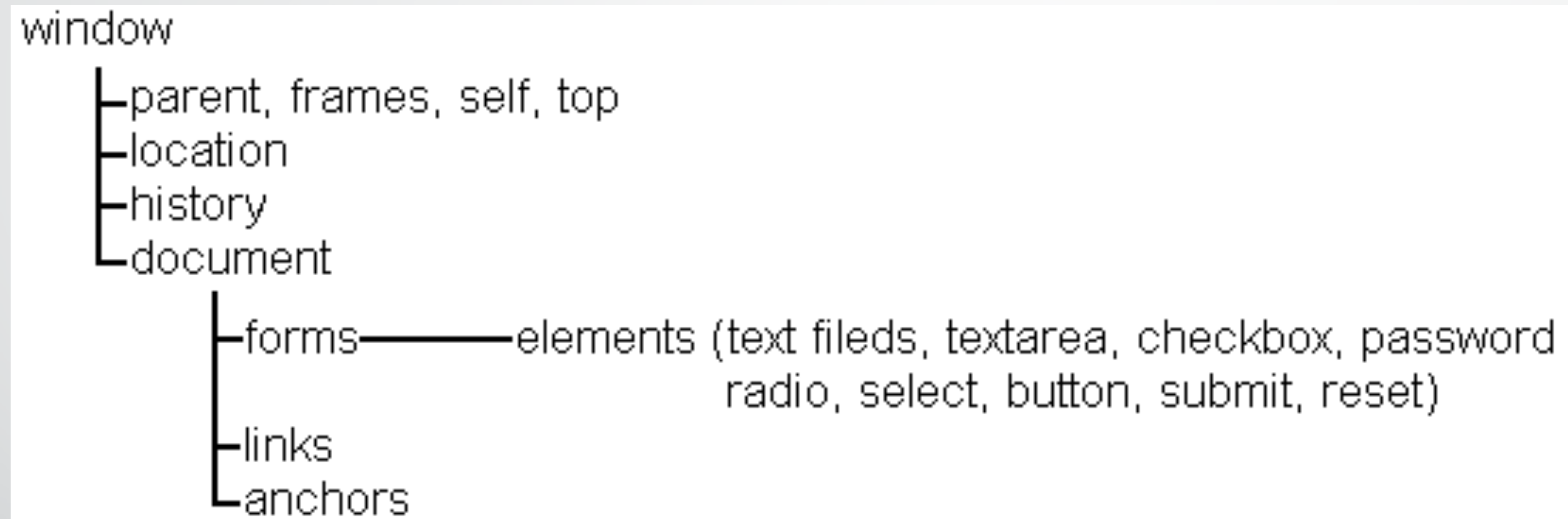
- Nom est le nom de l'objet textarea ;
- Propriété est la propriété à accéder,
- Nom\_du\_formulaire est le nom du formulaire dans lequel se trouve l'objet textarea (paramètre name dans le marqueur <form>) ;
- index est le numéro d'ordre de l'objet textarea dans le formulaire.

Les propriétés d'un objet textarea sont :

- name : nom de l'objet (paramètre name).
- value : valeur courante de l'objet.

## *L'objet window*

L'objet window se trouve au sommet de la hiérarchie :





Vous vous en doutez certainement, les propriétés relatives aux objets window sont très nombreuses. Pour y accéder, vous utiliserez l'une des syntaxes suivantes :

`window.Propriété`

`self.Propriété`

`top.Propriété`

`parent.Propriété`

`variable.Propriété`

- où window et `self` font référence à la fenêtre courante ;
- top est la fenêtre hiérarchiquement supérieure (dans le cas où plusieurs fenêtres sont affichées) ;
- parent fait référence à un document dans lequel ont été définies plusieurs cadres ;
- variable représente un nom de variable utilisé pour définir une nouvelle fenêtre.

Voici la liste des diverses propriétés accessibles :

Propriété	Fonction
defaultStatus	Message affiché par défaut dans la barre d'état
length	Nombre de cadres dans la fenêtre parent
name	Nom de la fenêtre, tel qu'il est défini dans le deuxième paramètre de la méthode open()
parent	Nom du document dans lequel ont été définies plusieurs cadres
self	Nom de la fenêtre courante
status	Message affiché dans la barre d'état
top	Nom de la fenêtre hiérarchiquement supérieure (dans le cas où plusieurs fenêtres sont affichées)
window	Nom de la fenêtre courante

Pour manipuler les objets window, vous utiliserez les méthodes suivantes :

Méthode	Fonction
alert()	Affichage d'une boîte de message contenant un message texte et un bouton OK
close()	Fermeture de la fenêtre spécifiée
confirm()	Affiche une boîte de dialogue de confirmation contenant un message texte, un bouton Annuler et un bouton OK
open()	Ouverture d'une fenêtre
prompt()	Saisie d'une donnée dans une boîte de dialogue contenant un bouton Annuler et un bouton OK
setTimeout()	Exécution d'une expression après un délai paramétrable
clearTimeout()	Suppression d'un délai initialisé avec setTimeout

## Exercice

Entraînez-vous à :

1. Afficher une boîte de message avec la méthode `alert()`
2. Afficher une boîte de confirmation avec la méthode `confirm()`

```
<script language="JavaScript">  
  alert('Ceci est une boîte de message affichée par JavaScript');  
</script>
```

```
<script language="JavaScript">  
function Confirmation(){  
  if (confirm("Etes-vous sûr de vouloir fermer le navigateur ?"))  
    window.close();  
}  
</script>  
...  
<input type="button" value="Fermer" onClick="Confirmation();">
```



# Ouverture d'une fenêtre - La méthode open()

Pour ouvrir une nouvelle fenêtre, vous utiliserez la méthode open() :

```
Nom = window.open("URL", "NomFenetre", "Caractéristiques")
```

- Nom est le nom de la nouvelle fenêtre ;
- URL est l'URL de la nouvelle fenêtre ;
- NomFenetre est le nom qui sera utilisé dans le paramètre target d'un marqueur <form> ou <a> ;
- Caractéristiques fixe une ou plusieurs caractéristiques de la fenêtre :
- **⚠ Elles ne sont pas toutes compatibles avec tous les navigateurs (voir sur MDN)**

Paramètre	Fonction
toolbar=[yes no]	Affiche/cache la barre d'outils dans la fenêtre
location=[yes no]	Affiche/cache la barre location dans la fenêtre
directories=[yes no]	Affiche/cache la barre contenant les boutons What's new, What's cool et Handbook
status=[yes no]	Affiche/cache la barre d'état
menubar=[yes no]	Affiche/cache la barre de menus
scrollbars=[yes no]	Affiche/cache les barres de défilement
resizable=[yes no]	Permet/interdit à l'utilisateur de redimensionner la fenêtre
width=Largeur	Définit la largeur de la fenêtre en pixels
height=Hauteur	Définit la hauteur de la fenêtre en pixels

## Exercice

Ouvrez une fenêtre de 400x400 pixels sur le site [www.google.fr](http://www.google.fr) suite au clic sur un bouton



## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ouverture d'une autre fenêtre</title>
  </head>
  <body>
    Cliquez sur ce bouton pour ouvrir une nouvelle fenêtre sur Google :
    <button onclick="window.open('http://www.google.fr','google','scrollbars=yes,
      status=yes,width=400,height=400')">Google</button>
  </body>
</html>
```



## Exercice

Définissez un document contenant deux boutons.

Le premier bouton ouvre une fenêtre secondaire de 200x100 pixels et y écrit le texte suivant :  
"Ceci est la fenêtre secondaire".

Le deuxième bouton ferme la fenêtre secondaire.

## Solution

```
<!DOCTYPE html>
<html>
<head>
</head>
<script>
  var maFenetre;
  function Ouvrir() {
    maFenetre = window.open('', 'maFenetre', 'width=200, height=100');
    maFenetre.document.write('<p>Ceci est la fenêtre secondaire</p>');
  }
  function Fermer() {
    maFenetre.close();
  }
</script>

<body>
  <button onclick="Ouvrir()">Ouvrir</button>
  <button onclick="Fermer()">Fermer</button>
</body>
</html>
```

## *Saisie d'une donnée - La méthode prompt()*

La méthode prompt() affiche une boîte de dialogue dans laquelle l'utilisateur peut saisir une donnée numérique ou texte. Une valeur par défaut peut être affichée. Un appui sur le bouton Cancel renvoie la valeur null et un appui sur le bouton OK renvoie la valeur entrée.

La syntaxe de la méthode prompt() est la suivante :

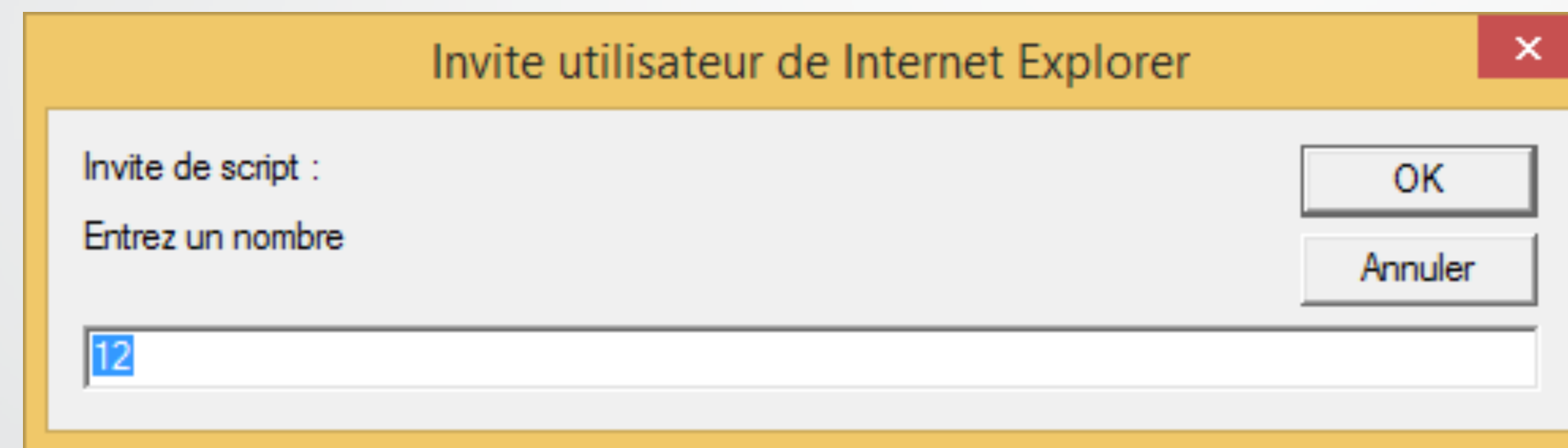
```
Retour=prompt ("Texte" [, Défaut] );
```

- Texte est le texte affiché dans la boîte de dialogue ;
- Défaut est la valeur par défaut ;
- Retour est la valeur retournée après appui sur le bouton OK ou Cancel.



## Exercice

Ecrivez le code nécessaire pour afficher la boîte de saisie suivante à l'ouverture de la page :



Invite utilisateur de Internet Explorer

Invite de script :  
Entrez un nombre

12

OK  
Annuler

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Boîte de saisie</title>
  </head>
  <body>
    <script>
      var valeur = prompt("Entrez un nombre",12);
      document.write("Vous avez entré la valeur ", valeur);
    </script>
  </body>
</html>
```

## ***La classe Date***

Les objets issus de la classe Date permettent de travailler avec les dates (jours, mois, années) et les heures (heures, minutes, secondes).

Pour définir un objet Date, vous pouvez utiliser l'une des quatre syntaxes suivantes :

```
var d = new Date();  
d = new Date("mois, jour, année heures:minutes:secondes");  
d = new Date(année, jour, mois);  
d = new Date(année, jour, mois, heures, minutes, secondes);
```

où d représente le nom d'un nouvel objet ou d'une propriété d'un objet existant, et mois, jour, année, heures, minutes et secondes les composantes de l'objet Date. Dans la deuxième syntaxe, ils sont exprimés sous une forme chaîne. Dans la troisième et la quatrième syntaxe, ils sont exprimés sous une forme entière.



De nombreuses méthodes sont attachées aux objets Date :

`getDate()`, `getDay()`, `getMinutes()`, `getMonth()`, `getSeconds()`, `getTime()`,  
`getTimeZoneoffset()`, `getYear()`, `getFullYear()`, `parse()`, `setDate()`, `setHours()`,  
`setMinutes()`, `setMonth()`, `setSeconds()`, `setTime()`, `setYear()`, `toGMTString()`,  
`toLocaleString()`, `UTC()`.

Exercice

Affichez la date et l'heure système.



## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Date et heure système</title>
    <script language="JavaScript">
      function WhatDayIsIt() {
        Aujourd'hui = new Date();
        Jour = Aujourd'hui.getDate();
        Mois = Aujourd'hui.getMonth()+1;
        Annee = Aujourd'hui.getFullYear();
        document.write("Aujourd'hui, nous sommes le " + Jour + " / " + Mois + " / " + Annee + ".<br>");
      }

      function WhatTimeIsIt() {
        Maintenant = new Date ();
        Heures = Maintenant.getHours();
        Minutes = Maintenant.getMinutes();
        Secondes = Maintenant.getSeconds();
        document.write ("Il est "+ Heures + " heures " + Minutes + " min " + Secondes + " sec.");
      }
    </script>
  </head>
  <body>
    <script>
      WhatDayIsIt();
      WhatTimeIsIt();
    </script>
  </body>
</html>
```

Une horloge JavaScript qui s'affiche dans un `<input>` de type text.

Ici, on utilise la fonction **setInterval()** pour exécuter la fonction **affiche()** à intervalles réguliers de **1000 ms**.

La fonction **commence()** autorise l'affichage de l'horloge et la fonction **arrete()** l'interdit.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Date</title>
  <script>
    var startStop = false;
    function affiche() {
      if (startStop) {
        var d = new Date();
        f.hms.value=d.toLocaleString();
      }
    }
    function commence(){
      startStop = true;
    }
    function arrete(){
      startStop = false;
    }
  </script>
</head>
<body>
  <form name="f">
    <input type="text" name="hms">
    <input type="button" value="Start" onclick="commence();">
    <input type="button" value="Stop" onclick="arrete();">
  </form>
  <script>
    setInterval(affiche, 1000);
  </script>
</body>
</html>
```

## ***La classe String***

Les chaînes de caractères sont des objets de classe String. Lorsque vous utilisez une instruction du type ci-après, vous définissez un objet String de nom "chaîne" :

```
chaîne = "Créer ses pages Web - Comment faire"
```

Pour connaître la longueur d'un objet String, vous utiliserez la propriété length.  
En reprenant l'exemple précédent, l'instruction :

```
chaîne.length
```

renvoie la valeur 35, car la chaîne comporte 35 caractères.

De nombreuses méthodes sont associées aux chaînes de caractères. Le tableau ci-après en dresse la liste :

Méthode	Utilisation
<code>anchor()</code>	Définition d'un signet
<code>big()</code>	Caractères de grande taille (identique au marqueur <BIG>)
<code>blink()</code>	Affecte l'attribut clignotant au texte (identique au marqueur <BLINK>)
<code>bold()</code>	Affecte l'attribut gras au texte (identique au marqueur <B>)
<code>charAt()</code>	Renvoie le caractère qui se trouve à la position indiquée
<code>fixed()</code>	Police non proportionnelle (identique au marqueur <TT>)
<code>fontcolor()</code>	Définit la couleur des caractères (identique au paramètre COLOR du marqueur <FONT>)
<code>fontsize()</code>	Définit la taille des caractères (identique au marqueur <FONTSIZE>)
<code>indexOf()</code>	Première position d'une sous-chaîne dans une chaîne
<code>italics()</code>	Affecte l'attribut italique au texte
<code>lastIndexOf()</code>	Dernière position d'une sous-chaîne dans une chaîne
<code>link()</code>	Définit un lien hypertexte
<code>small()</code>	Caractères de petite taille (identique au marqueur <SMALL>)
<code>strike()</code>	Affecte l'attribut barré au texte (identique au marqueur <STRIKE>)
<code>sub()</code>	Caractères en indice (identique au marqueur <SUB>)
<code>substring()</code>	Extraction d'une sous-chaîne
<code>sup()</code>	Caractères en exposant (identique au marqueur <SUP>)
<code>toLowerCase()</code>	Conversion en minuscules
<code>toUpperCase()</code>	Conversion en majuscules



### Exemple 1 - Caractères de grande taille et de petite taille : méthodes big() et small().

Ces instructions affichent le texte de la variable chaine en caractères de grande taille, puis de petite taille :

```
chaine = "Créer ses pages Web - Comment faire";  
document.write(chaine.big());  
document.write(chaine.small());
```

### Exemple 2 - Attributs gras, italique et barré : méthodes blink(), bold(), italics() et strike().

Ces instructions affectent les attributs gras et italique aux mots "Créer", et "ses pages".

```
ch1 = "Créer ";  
ch2 = "ses pages";  
document.write(ch1.bold());  
document.write(ch2.italics());
```

### Exemple 3 - Extraction de caractères : méthode charAt().

Ces instructions extraient les caractères en position 3 et 11 dans la chaîne ch :

```
ch = "Créer ses pages Web - Comment faire<br>";  
document.write(ch, "<br>");  
document.write("Le caractère en position 3 est un ", ch.charAt(3), "<br>");  
document.write("et le caractère en position 11 est un ", ch.charAt(11));
```

### **Exemple 4 - Caractères non proportionnels : méthode fixed().**

Ces instructions affichent le texte de la chaîne ch en utilisant une police conventionnelle, puis une police non proportionnelle :

```
ch = "Créer ses pages Web - Comment faire<BR>";  
document.write(ch);  
document.write(ch.fixed());
```

### **Exemple 5 - Modification de la taille des caractères : méthode fontSize().**

Ces instructions JavaScript affichent un texte en taille 4 et en taille 7 :

```
ch = "Créer ses pages Web - Comment faire<BR>";  
document.write(ch.fontSize(4));  
document.write(ch.fontSize(7));
```

### **Exemple 6 - Position d'une sous-chaîne dans une chaîne : méthodes indexOf() et lastIndexOf().**

Ces instructions affichent la position du premier et du dernier "e" dans la chaîne "Créer ses pages Web - Comment faire" :

```
ch = "Créer ses pages Web - Comment faire<BR>";  
document.write(ch);  
document.write("Le premier e occupe la position ",ch.indexOf("e"),"<BR>");  
document.write("Le dernier e occupe la position ",ch.lastIndexOf("e"));
```

## Exemple 7 - Définition d'un lien hypertexte : méthode link().

Les instructions JavaScript ci-après définissent deux liens hypertextes :

```
ch1 = "Recherche Google";  
ch2 = "Recherche Bing";  
URL1 = "http://www.google.fr";  
URL2 = "http://www.bing.com";  
document.write("Sélectionnez l'un de ces sites :<BR>");  
document.write(ch1.link(URL1), "<BR>");  
document.write(ch2.link(URL2), "<BR>");
```

## Exemple 8 - Caractères en indice et en exposant : méthodes sub() et sup().

Ces instructions JavaScript affichent une formule mathématique qui utilise un indice et un exposant :

```
ch1 = "(x)"  
ch2 = "2"  
document.write("f", ch1.sub(), " = a x", ch2.sup(), " + b x + c");
```

## Exemple 9 - Extraction d'une sous-chaîne : méthode substring().

Les instructions suivantes affichent les caractères "ivr" qui occupent les positions 4 à 7 dans la chaîne ch :

```
ch = "Le livre d'or Java<BR>"  
document.write("ch.substring(4,7) = ", ch.substring(4,7));
```

Remarquez que le premier caractère a pour index 0 (et non 1).



## Exemple 10 - Caractères majuscules et minuscules : méthodes toLowerCase() et toUpperCase().


Ces instructions affichent la chaîne originale, sa conversion en minuscules, puis sa conversion en majuscules :

```
ch = "Créer ses pages Web - Comment faire<BR>"
document.write(ch);
document.write(ch.toLowerCase());
document.write(ch.toUpperCase());
```

Voici le résultat obtenu :

```
Créer ses pages Web - Comment faire
créer ses pages Web - comment faire
CREER SES PAGES WEB - COMMENT FAIRE
```





# Validation de données en HTML et en JavaScript

# Validation de données

Lorsqu'un champ de saisie doit obligatoirement être rempli, il suffit de lui affecter l'attribut required. Dans ce formulaire, le champ de saisie Nom est obligatoire. Si l'utilisateur clique sur le bouton Submit sans l'avoir renseigné, un message d'erreur est généré et le formulaire n'est pas envoyé.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Champ de saisie obligatoire</title>
  </head>
  <body>
    <form id="validation">
      <label>Nom</label><input type="text" name="Nom" required><br><br><br><br>
      <input type="submit" value="Valider"></p>
    </form>
  </body>
</html>
```

Code062.htm

Essayez ce code

# Validation de données

Le champ d'action des attributs required est bien plus étendu, puisqu'il permet également de valider des données complexes (telles que des adresses e-mail ou des URL). À titre d'exemple, le formulaire ci-après teste la validité des champs email et url. Lorsqu'on clique sur le bouton Submit, un message d'erreur s'affiche si un de ces deux champs n'est pas conforme.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Validation e-mail et url</title>
  </head>
  <body>
    <form id="validation">
      <label>Nom</label> <input type="text" name="Nom"><br>
      <label>e-mail</label> <input type="email" name="email" required><br>
      <label>Adresse URL</label> <input type="url" name="url" required><br>
      <label>Commentaire</label> <textarea name="comment"></textarea><br>
      <input type="submit" value="Valider"></p>
    </form>
  </body>
</html>
```

Code063.htm

Essayez ce code

# Validation de données

Pour faciliter la reconnaissance des champs obligatoires, il est possible d'utiliser quelques lignes de CSS. Par exemple, cette ligne affecte un arrière-plan de couleur jaune aux champs <input> dont l'attribut required est spécifié :

```
input:required { background:yellow }
```



# Validation de données

JavaScript permet de contrôler la validité des champs d'un formulaire lors du clic sur le bouton Submit. Le formulaire n'est pas communiqué au serveur si les données ne sont pas valides.

Examinez et testez ce code :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Texte</title>
    <script>
      function validation() {
        var leNom = document.form1.nom.value;
        if (leNom == "") {
          alert("Vous devez remplir le champ Nom avant d'envoyer le formulaire");
          return false; // Le formulaire n'est pas validé
        }else{
          return true; // Le formulaire est validé
        }
      }
    </script>
  </head>
  <body>
    <form name="form1" action="traitement.php" method="post" onsubmit="return validation();">
      Nom <input type="text" name="nom">
      <input type="submit" value="Envoyer">
    </form>
  </body>
</html>
```

## Accès aux éléments du DOM avec la fonction `document.getElementById()`

Les éléments insérés dans le DOM peuvent avoir un attribut `id` (identifiant) qui les identifie de façon unique. Ces éléments peuvent alors être accédés en lecture et en écriture en JavaScript avec la fonction `document.getElementById()`.

Exemple :

```
<div id="laDiv"></div>
<script>
  document.getElementById('laDiv').innerHTML = 'test';
</script>
```

Supposons qu'un élément d'id unElement soit défini :

```
<span id="unElement">Un simple texte</span>
```

Si vous définissez cette variable :

```
var el = document.getElementById('unElement');
```

Vous pouvez :

- connaître le contenu de cet élément avec la propriété innerHTML :  
`document.write(el.innerHTML);`
- modifier le contenu de cet élément avec la propriété innerHTML :  
`el.innerHTML = 'Un autre contenu';`
- connaître ou modifier la couleur de cet élément avec la propriété style.color :  
`el.style.color='red';`
- connaître ou modifier la couleur de cet élément avec la propriété style.backgroundColor :  
`el.style.backgroundColor='yellow';`
- cacher ou afficher l'élément avec la propriété display :  
`el.style.display='none';`  
`el.style.display='inline';`

Et beaucoup d'autres choses encore...

Affichage ou dissimulation d'un <div> en fonction de boutons radio Mr et Mme :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Mr Mme</title>
  <script>
    function mSelectionne() {
      var leDiv = document.getElementById('d');
      leDiv.style.display = 'none';
    }
    function mmeSelectionnee() {
      var leDiv = document.getElementById('d');
      leDiv.style.display = 'block';
    }
  </script>
</head>
<body>
  <form name="f">
    <input type="radio" value="M" name="MMme" onclick="mSelectionne();"> Mr
    <input type="radio" value="Mme" name="MMme" checked onclick="mmeSelectionnee();"> Mme<br>
    <div id="d">
      <label>Nom de jeune fille</label>
      <input type="text" name="njf">
    </div>
  </form>
</body>
</html>
```





## Exercice

Dans un formulaire contenant plusieurs zones de texte, mettez en place les instructions nécessaires pour colorer l'arrière-plan de l'élément qui a le focus (c'est-à-dire celui dans lequel se fait la saisie).

# Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Coloration sur focus en JS</title>
    <script>
      function vert(el) {
        el.style.backgroundColor = 'green';
      }
      function blanc(el) {
        el.style.backgroundColor = 'white';
      }
    </script>
  </head>
  <body>
    <form>
      <input type="text" onfocus="vert(this);" onblur="blanc(this);">
      <input type="text" onfocus="vert(this);" onblur="blanc(this);">
      <input type="text" onfocus="vert(this);" onblur="blanc(this);">
      <input type="text" onfocus="vert(this);" onblur="blanc(this);">
      <input type="text" onfocus="vert(this);" onblur="blanc(this);">
    </form>
  </body>
</html>
```

## Exercice :

Définissez une page composée de deux liens et d'une balise <div> qui contient du texte. Lorsque l'utilisateur clique sur le premier lien, déplacez le <div> à 100 pixels du bord gauche. Lorsqu'il clique sur le deuxième lien, déplacez le <div> à 500 pixels du bord gauche.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>getElementById() dans un formulaire</title>
    <script>
      function gauche(){document.getElementById('laDiv').style.left = '100px';}
      function droite(){document.getElementById('laDiv').style.left = '500px';}
    </script>
    <style>
      div {
        background-color: yellow;border: dotted 1px #666;
        width: 150px;height: 150px;left: 200px;display: block;position: absolute;
      }
    </style>
  </head>
  <body>
    <a href="javascript:gauche();">Vers la gauche</a><br>
    <a href="javascript:droite();">Vers la droite</a><br>
    <div id="laDiv">Un peu de texte</div>
  </body>
</html>
```

Vous pouvez également modifier un attribut d'une balise HTML avec la fonction **setAttribute()**.

Exercice :

En partant de ce code, ajoutez un bouton de commande qui modifie l'image affichée.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Test de setAttribute()</title>
  </head>
  <body>
    
  </body>
</html>
```



## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Test de frames</title>
    <script>
      function changeImage() {
        var im = document.getElementById('image');
        im.setAttribute('src', 'chat2.jpg');
      }
    </script>
  </head>
  <body>
    
    <button onclick="changeImage()">Changer l'image</button>
  </body>
</html>
```

Vous pouvez également accéder à plusieurs éléments avec la méthode **getElementsByTagName()**. Voici un exemple :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>getElementsByTagName</title>
</head>
<body>
  <p>un texte</p>
  <p>un autre texte</p>
  <p>un dernier texte</p>
  <script>
    var el = document.getElementsByTagName('p');
    for (var i=0; i<el.length; i++){
      el[i].innerHTML='Un nouveau texte';
    }
  </script>
</body>
</html>
```

Enfin, vous pouvez accéder à plusieurs éléments via leur classes avec la fonction **getElementsByClassName()**. Voici un exemple :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>getElementsByClassName</title>
</head>
<body>
  <p class="rouge">un texte</p>
  <p>un autre texte</p>
  <p class="rouge">un dernier texte</p>
  <script>
    var el = document.getElementsByClassName('rouge');
    for (var i=0; i<el.length; i++){
      el[i].style.color='red';
    }
  </script>
</body>
</html>
```

# Version moderne du gestionnaire d'événements

Si vous utilisez un navigateur récent, vous pouvez mettre en place des gestionnaires d'événements en utilisant une syntaxe JavaScript plus moderne. Pour cela, vous passerez par la fonction `addEventListener()` :

```
élément.addEventListener(type, listener);
```

Où :

- **élément** est l'élément sur lequel s'applique le gestionnaire ;
- **type** représente le type de l'événement à capturer ;
- **listener** est la fonction JavaScript à exécuter quand l'événement se produit.



Essayez ce code :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'événement moderne</title>
  </head>
  <body>
    <button id="bouton">Cliquez ici</button>
    <script>
      var element = document.getElementById('bouton');
      var maFonction = function() {
        alert('Vous avez cliqué');
      };
      element.addEventListener('click', maFonction);
    </script>
  </body>
</html>
```



Pour supprimer un gestionnaire d'événements, vous utiliserez la fonction `removeEventListener()` :

```
élément.removeEventListener(type, listener);
```

Où :

- **élément** est l'élément sur lequel s'applique le gestionnaire ;
- **type** représente le type de l'événement à capturer ;
- **listener** est le gestionnaire d'événements rattaché à cet événement.

Essayez ce  
code :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'événement moderne</title>
  </head>
  <body>
    <button id="bouton">Cliquez ici</button>
    <button onclick="ajoute();">Ajouter le gestionnaire</button>
    <button onclick="supprime();">Supprimer le gestionnaire</button>
    <script>
      var element = document.getElementById('bouton');
      var maFonction = function() {
        alert('Vous avez cliqué');
      };
      function ajoute() {
        element.addEventListener('click', maFonction);
      }
      function supprime() {
        element.removeEventListener('click', maFonction);
      }
    </script>
  </body>
</html>
```

Comment auriez-vous fait pour ajouter ou supprimer un gestionnaire d'événements sur le premier bouton lorsque l'utilisateur clique sur les deux autres boutons ? Et cela, sans utiliser `addEventListener()` ni `removeEventListener()`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'événement moderne</title>
  </head>
  <body>
    <button id="bouton">Cliquez ici</button>
    <button onclick="ajoute();">Ajouter le gestionnaire</button>
    <button onclick="supprime();">Supprimer le gestionnaire</button>
  </body>
</html>
```



Solution :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'événement moderne</title>
  </head>
  <body>
    <button id="bouton">Cliquez ici</button>
    <button onclick="ajoute();">Ajouter le gestionnaire</button>
    <button onclick="supprime();">Supprimer le gestionnaire</button>
    <script>
      var element = document.getElementById('bouton');

      function ajoute() {
        element.onclick = function() {
          alert('Vous avez cliqué');
        };
      }
      function supprime() {
        element.onclick = function() {};
      }
    </script>
  </body>
</html>
```

# Gestion de clic unique

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Clic unique</title>
    <script>
      function traitement() {
        var leBouton = document.getElementById('b');
        leBouton.removeEventListener('click', traitement);
        leBouton.setAttribute('disabled', true);
        alert('Merci d\'avoir cliqué !');
      }
    </script>
  </head>
  <body>
    <button id="b">Cliquez ici</button>
    <script>
      var leBouton = document.getElementById('b');
      leBouton.addEventListener('click', traitement);
    </script>
  </body>
</html>
```

# Cookies

Un cookie est un petit fichier texte stocké côté client par le navigateur.

Il contient plusieurs données :

1. Une paire nom / valeur qui contient les données du cookie.
2. Une date d'expiration au-delà de laquelle il n'est plus valide.
3. Un domaine et une arborescence qui indiquent quel répertoire de quel serveur y aura accès.

Les cookies peuvent être créés, lus et supprimés par JavaScript.  
Ils sont accessibles à travers la propriété **document.cookie**.

Voici un exemple de création de cookie :

```
document.cookie = 'cookie1=testcookie; expires=Sat, 27 Jun 2015  
00:00:00 UTC; path=/';
```

Cette chaîne comporte :

1. Une paire nom / valeur : 'cookie1=testcookie'
2. Un point-virgule et un espace
3. La date d'expiration : expires=Sat, 27 Jun 2015 00:00:00 UTC
4. Un point-virgule et un espace
5. Le domaine : path=/'

Si vous définissez un deuxième cookie :

```
document.cookie = 'cookie2=testcookie2; expires=Sat, 20 Jun 2015  
08:00:00 UTC; path=/';
```

Le premier cookie n'est pas effacé.





Pour lire les cookies, vous utiliserez cette syntaxe :

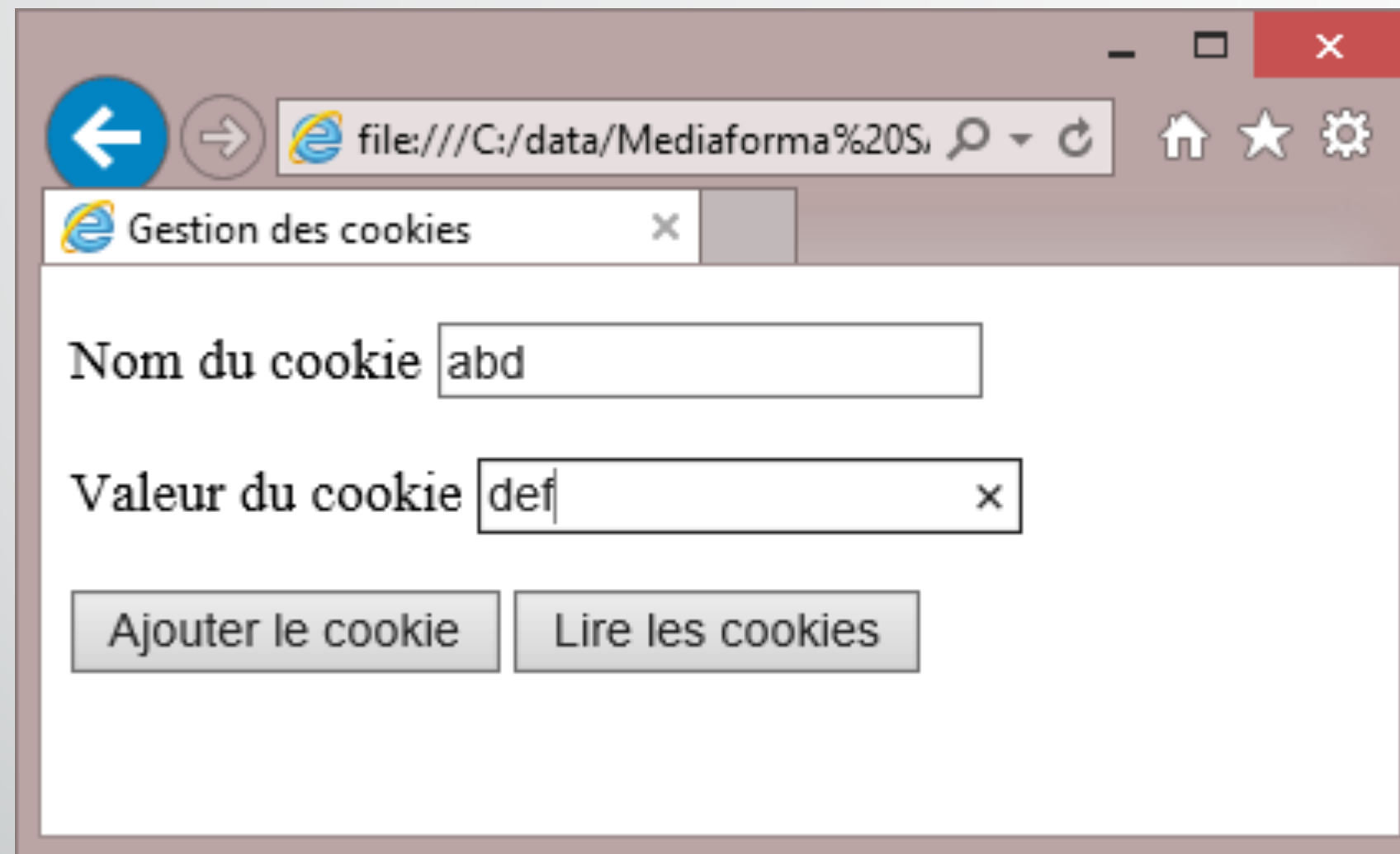
```
var x = document.cookie;
```

Après l'exécution de cette instruction, x contiendra la liste des paires/valeurs de tous les cookies stockés par le navigateur, séparés entre eux par des ";"

**Exercice :**

Définissez deux cookies et affichez les cookies du navigateur dans un <textarea>

Voici ce que vous devez obtenir :



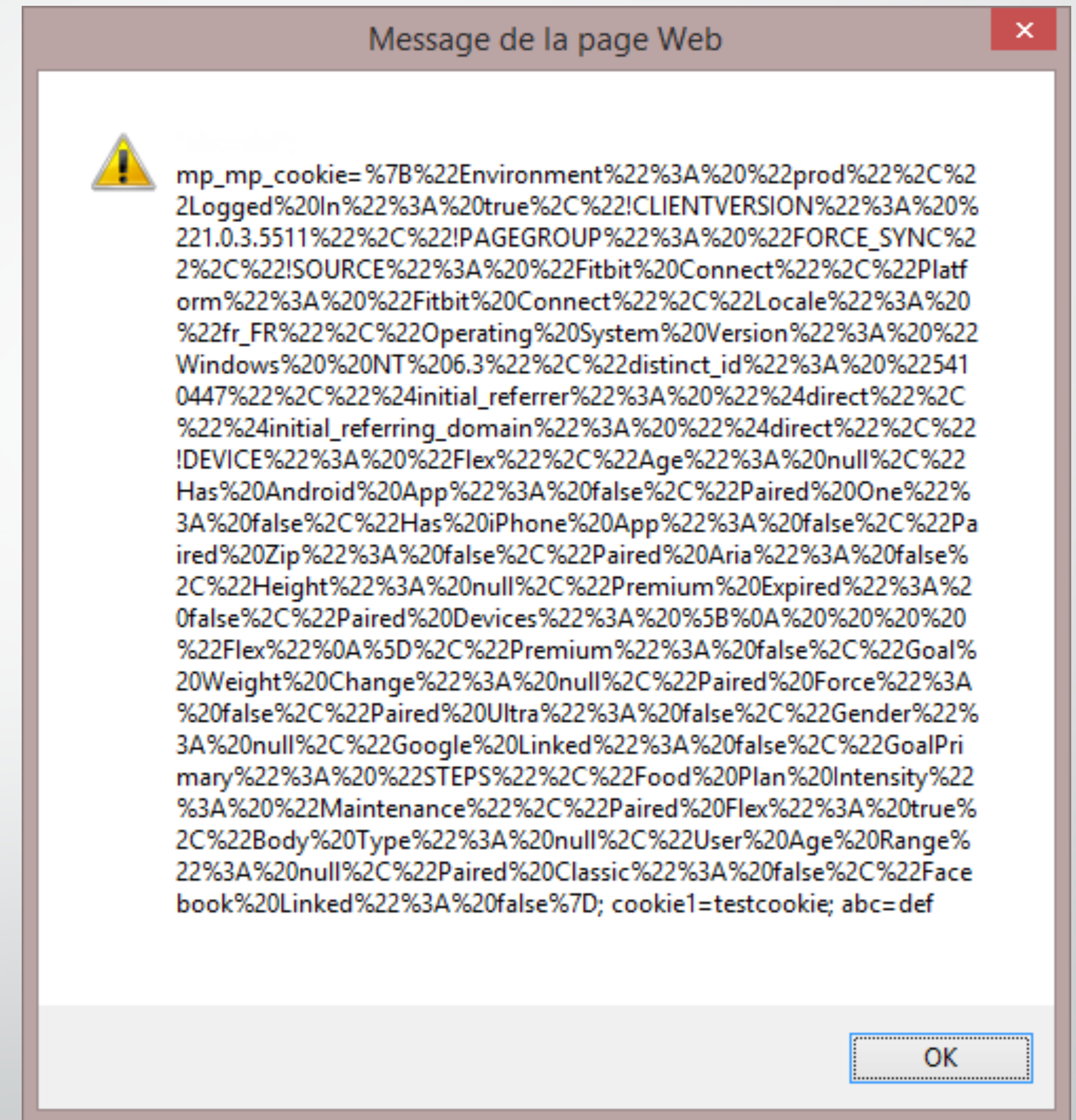
file:///C:/data/Mediaforma%20S...

Gestion des cookies


Nom du cookie

Valeur du cookie

Ajouter le cookie Lire les cookies



Message de la page Web

 mp\_mp\_cookie=%7B%22Environment%22%3A%20%22prod%22%2C%22Logged%20In%22%3A%20true%2C%22!CLIENTVERSION%22%3A%20%221.0.3.5511%22%2C%22!PAGEGROUP%22%3A%20%22FORCE\_SYNC%22%2C%22!SOURCE%22%3A%20%22Fitbit%20Connect%22%2C%22Platform%22%3A%20%22Fitbit%20Connect%22%2C%22Locale%22%3A%20%22fr\_FR%22%2C%22Operating%20System%20Version%22%3A%20%22Windows%20%20NT%206.3%22%2C%22distinct\_id%22%3A%20%225410447%22%2C%22%24initial\_referrer%22%3A%20%22%24direct%22%2C%22%24initial\_referring\_domain%22%3A%20%22%24direct%22%2C%22!DEVICE%22%3A%20%22Flex%22%2C%22Age%22%3A%20null%2C%22Has%20Android%20App%22%3A%20false%2C%22Paired%20One%22%3A%20false%2C%22Has%20iPhone%20App%22%3A%20false%2C%22Paired%20Zip%22%3A%20false%2C%22Paired%20Aria%22%3A%20false%2C%22Height%22%3A%20null%2C%22Premium%20Expired%22%3A%20false%2C%22Paired%20Devices%22%3A%20%5B%0A%20%20%20%20%20%22Flex%22%3A%20%5D%2C%22Premium%22%3A%20false%2C%22Goal%20Weight%20Change%22%3A%20null%2C%22Paired%20Force%22%3A%20false%2C%22Paired%20Ultra%22%3A%20false%2C%22Gender%22%3A%20null%2C%22Google%20Linked%22%3A%20false%2C%22GoalPrimary%22%3A%20%22STEPS%22%2C%22Food%20Plan%20Intensity%22%3A%20%22Maintenance%22%2C%22Paired%20Flex%22%3A%20true%2C%22Body%20Type%22%3A%20null%2C%22User%20Age%20Range%22%3A%20null%2C%22Paired%20Classic%22%3A%20false%2C%22Facebook%20Linked%22%3A%20false%7D; cookie1=testcookie; abc=def

OK

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion des cookies</title>
    <script>
      function ajouter(form) {
        var nom = form.nom.value;
        var valeur = form.valeur.value;
        var leCookie = nom + '=' + valeur + '; expires=Sat, 27 Jun 2015 00:00:00 UTC; path=/';
        document.cookie = leCookie;
      }
      function lire() {
        alert(document.cookie);
      }
    </script>
  </head>
  <body>
    <form>
      <p><label>Nom du cookie</label> <input type="text" name="nom"></p>
      <p><label>Valeur du cookie</label> <input type="text" name="valeur"></p>
      <button onclick="ajouter(this.form);">Ajouter le cookie</button>
      <button onclick="lire();">Lire les cookies</button><br>
    </form>
  </body>
</html>
```

Pour modifier un cookie, vous utiliserez la même technique que pour le créer.

Par exemple, supposons que vous ayez défini ce cookie :

```
document.cookie = 'cookie1=testcookie; expires=Sat, 27  
Jun 2015 00:00:00 UTC; path=/';
```

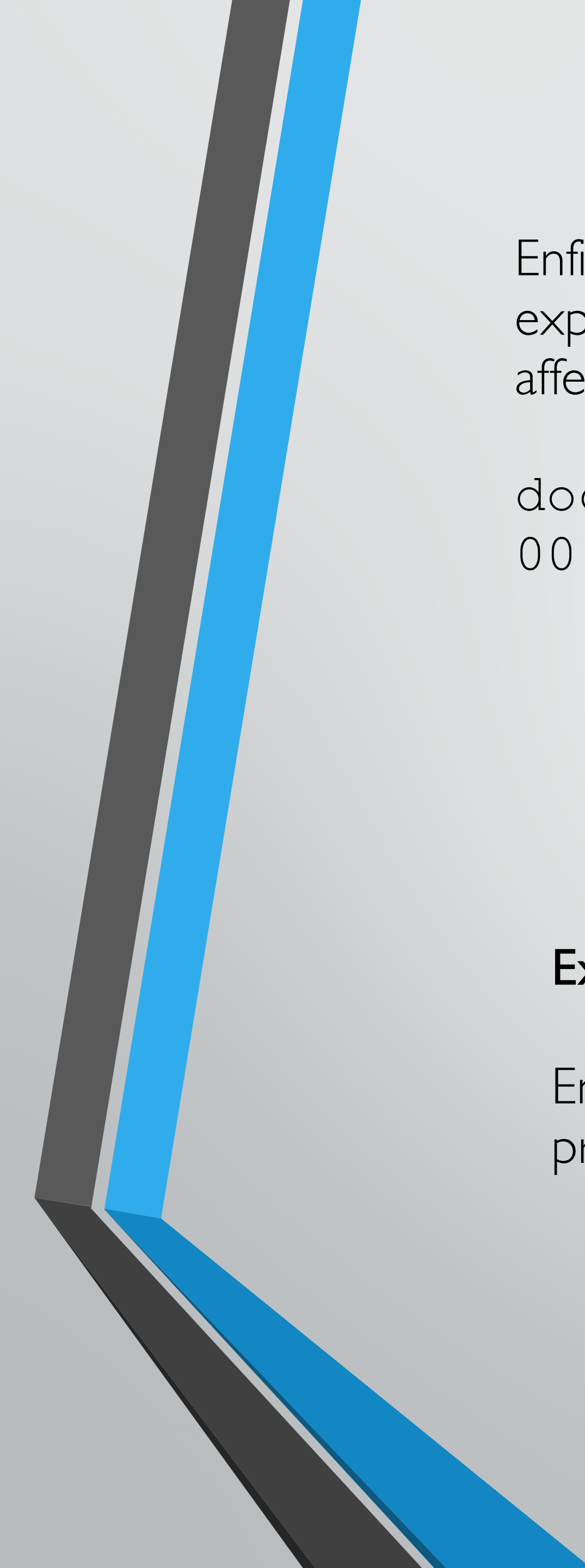
Et que vous vouliez changer :

- La valeur **testcookie** en **nouveauTestCookie**
- La date d'expiration au **Dimanche 28 Juin**

Vous écrirez quelque chose comme ceci :

```
document.cookie = 'cookie1=nouveauTestCookie;  
expires=Sun, 28 Jun 2015 00:00:00 UTC; path=/';
```





Enfin, pour supprimer un cookie, affectez une date antérieure à aujourd'hui au paramètre expires. Vous pouvez ou non affecter une valeur au cookie. Ici par exemple, nous ne lui affectons aucune valeur :

```
document.cookie = 'cookie1=; expires=Sat, 16 May 2015  
00:00:00 UTC; path=/';
```

## Exercice

Entrenez-vous à modifier et à supprimer des cookies en complétant le programme précédent.

# Stockage local – Local Storage

Avec HTML5, le stockage de données en local est désormais un jeu d'enfant. D'ici à prédire la disparition des cookies, il n'y a qu'un pas...

Voyons ce que HTML5 propose. En fait, le code à utiliser est du JavaScript. Il repose sur l'utilisation des fonctions associées à l'objet localStorage :

Fonction	Signification
<u>localStorage.getItem('nom')</u>	Retourne la valeur correspondant au nom spécifié dans l'argument
<u>localStorage.setItem('nom', 'valeur')</u>	Stocke la valeur spécifiée sous le nom spécifié dans l'argument
<u>localStorage.key(position)</u>	Retourne le nom de l'élément stocké à la position spécifiée (l'indice 0 correspond à la première position de sauvegarde)
<u>localStorage.length</u>	Retourne le nombre de données mémorisées
<u>localStorage.clear()</u>	Efface toutes les données sauvegardées
<u>localStorage.removeItem('nom')</u>	Efface la donnée sauvegardée sous le nom spécifié dans l'argument

# Exercice

Définissez un formulaire comparable à la copie d'écran ci-contre et donnez vie aux quatre boutons en utilisant les fonctions liées au stockage local.

The screenshot shows a web browser window titled "Stockage local HTML5". The address bar displays the file path: `file:///C:/Data/Pearson/HTML%205/code/storage.htm?name=a&data=`. The page content is as follows:

## Stockage local HTML5

**Entrez la donnée à stocker**

Nom

Donnée 

Une autre donnée texte  
stockée sur  
plusieurs lignes

**Données stockées**

Nom	Valeur
deuxième	Une autre donnée texte stockée sur plusieurs lignes
premier	Cette donnée est stockée sous le nom "premier"

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Stockage local HTML5</title>

    <style>
      div
      {
        border-width: 1px;
        border-style: dotted;
      }
    </style>

    <script>
      function Affiche()
      {
        var key = "";
        var NomValeur = "<tr><th>Nom</th><th>Valeur</th></tr>\n";
        var i=0;
        for (i=0; i<=localStorage.length-1; i++)
        {
          key = localStorage.key(i);
          NomValeur += "<tr><td>"
            +key+"</td>\n<td>"
            +localStorage.getItem(key)
            +"</td></tr>\n";
        }
        document.getElementById('NomValeur').innerHTML = NomValeur;
      }

      function Enregistre()
      {
        var nom = document.forms.editor.name.value;
        var valeur = document.forms.editor.data.value;
        localStorage.setItem(nom, valeur);
        Affiche();
      }

      function Lit()
      {
        var nom = document.forms.editor.name.value;
        document.forms.editor.data.value = localStorage.getItem(nom);
        Affiche();
      }
    </script>
  </head>
  <body>
    <div id="div" style="border: 1px dotted black; padding: 10px; width: fit-content; margin: auto;">
      <h1>Stockage local HTML5</h1>
      <div id="divForm" style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 80%;">
        <div id="divDonnees" style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">
          <h2>Entrez la donnée à stocker</h2>
          <p><label>Nom <input type="text" name="name"></label></p>
          <p><label>Donnée <input type="text" name="data"></label></p>
          <p><input type="button" value="Lit" onclick="Lit();" />
            <input type="button" value="Enregistre" onclick="Enregistre();" />
            <input type="button" value="Efface" onclick="Efface();" />
            <input type="button" value="Efface tout" onclick="EffaceTout();" />
          </div>
          <div id="divValeurs" style="border: 1px solid black; padding: 5px; margin-top: 5px;">
            <h2>Données stockées</h2>
            <table id="NomValeur" border="1"></table>
          </div>
        </div>
      </div>
    </body>
</html>
```

Testez ce code  
dans Google Chrome

```
function Efface()
{
  var nom = document.forms.editor.name.value;
  document.forms.editor.data.value = localStorage.removeItem(nom);
  Affiche();
}

function EffaceTout()
{
  localStorage.clear();
  Affiche();
}
</script>
</head>

<body onload="Affiche()">
  <h1>Stockage local HTML5</h1>
  <div id="div" style="border: 1px dotted black; padding: 10px; width: fit-content; margin: auto;">
    <div id="divForm" style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 80%;">
      <div id="divDonnees" style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">
        <h2>Entrez la donnée à stocker</h2>
        <p><label>Nom <input type="text" name="name"></label></p>
        <p><label>Donnée <input type="text" name="data"></label></p>
        <p><input type="button" value="Lit" onclick="Lit();" />
          <input type="button" value="Enregistre" onclick="Enregistre();" />
          <input type="button" value="Efface" onclick="Efface();" />
          <input type="button" value="Efface tout" onclick="EffaceTout();" />
        </div>
        <div id="divValeurs" style="border: 1px solid black; padding: 5px; margin-top: 5px;">
          <h2>Données stockées</h2>
          <table id="NomValeur" border="1"></table>
        </div>
      </div>
    </div>
  </body>
</html>
```



# Stockage local – Session Storage

Vous savez maintenant utiliser l'objet **localStorage**. Une alternative s'offre à vous avec l'objet **sessionStorage**.

Les propriétés et méthodes de l'objet **sessionStorage** sont les mêmes que celles de l'objet **localStorage**. Ce qui change, c'est la durée de vie des données stockées :

- avec localStorage, le stockage est permanent, jusqu'à ce que les données soient effacées avec la méthode `clear()` ou `removeItem()`.
- avec sessionStorage, le stockage dure le temps de la session de navigation, c'est-à-dire tant que la page est ouverte. Les données stockées sont automatiquement supprimées à la fermeture de la page.

# Gestion événementielle

Avec HTML5, de nombreux nouveaux types d'événements font leur apparition. Ils concernent la fenêtre, le clavier, la souris, les formulaires et les médias. Nous allons les découvrir au fil de ce chapitre.

Dans cette section :

- Bases de la gestion événementielle
- Événements liés à la fenêtre
- Événements liés au clavier
- Événements liés à la souris
- Événements liés aux formulaires
- Événements liés aux médias

# Bases de la gestion événementielle

Pour capturer un événement, il suffit d'insérer l'attribut correspondant dans l'élément cible et de préciser le nom de la procédure JavaScript à exécuter, en lui passant zéro, un ou plusieurs arguments. Lorsque l'événement se produit, le code JavaScript correspondant est exécuté.

Dans le code de la diapositive suivante, trois événements sont capturés :

- chargement du document ;
- clic sur le bouton 1 ;
- clic sur le bouton 2.

Ils exécutent respectivement les fonctions JavaScript load(), bouton() avec un argument égal à 1, et bouton() avec un argument égal à 2.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion événementielle</title>
    <script>
      function load()
      {
        document.getElementById("status").innerHTML = "L'événement 'load' a été généré.";
      }
      function bouton(b)
      {
        document.getElementById("status").innerHTML = "Le bouton " +b + " a été cliqué.";
      }
    </script>
  </head>
  <body onload="load()" onunload="unload();">
    <p>Status: <span id="status">En attente.</span></p>
    <button onClick="bouton(1)">Bouton 1</button>
    <button onClick="bouton(2)">Bouton 2</button>
  </body>
</html>
```



# Événements liés à la fenêtre

Mis à part onblur, onfocus et onload, qui étaient déjà présents dans HTML 4.x, tous les événements de ce tableau sont nouveaux.

Attribut	Exécution du script
<u>onafterprint</u>	Après l'impression
<u>onbeforeprint</u>	Avant l'impression
<u>onbeforeunload</u>	Avant le chargement du document
<u>onblur</u>	Lorsque la fenêtre perd le focus
<u>onerror</u>	Lorsqu'une erreur est détectée
<u>onfocus</u>	Lorsque le focus est donné à la fenêtre
<u>onhaschange</u>	Lorsque le contenu du document est modifié
<u>onload</u>	Lorsque le document est chargé
<u>onmessage</u>	Quand le message est généré
<u>onoffline</u>	Au passage de l'état en ligne à l'état déconnecté
<u>ononline</u>	Au passage de l'état déconnecté à l'état en ligne
<u>onpagehide</u>	Lorsque la fenêtre devient invisible
<u>onpageshow</u>	Lorsque la fenêtre devient visible
<u>onpopstate</u>	Lorsque l'historique de la page change
<u>onredo</u>	Après l'exécution d'un "redo"
<u>onresize</u>	Au redimensionnement de la fenêtre
<u>onstorage</u>	Au chargement d'un document
<u>onundo</u>	À l'exécution d'un "undo"
<u>onunload</u>	Lorsque l'utilisateur change de document

# Événements liés à la fenêtre

Code078.htm

```
<!DOCTYPE html>
<html>
  <head>
    <title>Événements liés à la fenêtre</title>
    <script>
      function unload()
      {
        document.getElementById("status").innerHTML = "L'événement 'unload' a été généré";
      }
      function load()
      {
        document.getElementById("status").innerHTML = "L'événement 'load' a été généré";
      }
    </script>
  </head>
  <body onload="load()" onunload="unload();" >
    <p>Status: <span id="status">En attente de l'événement 'unload'</span></p>
  </body>
</html>
```

# Événements liés au clavier

Tous les événements clavier étaient déjà disponibles dans HTML 4.x.

Attribut	Exécution du script
<u>onkeydown</u>	Lorsqu'une touche est pressée
<u>onkeypress</u>	Lorsqu'une touche est pressée puis relâchée
<u>onkeyup</u>	Lorsqu'une touche est relâchée

Le code de la diapositive suivante affiche un élément span et une zone de texte <input type="text">. Un caractère tapé dans la zone de texte est récupéré *via* la fonction événementielle faitecho, déclenchée sur l'événement onkeypress. Le code de la touche utilisée est récupéré (ev.keyCode), converti en une chaîne (String.fromCharCode()) affichée dans l'élément span (document.getElementById("echo").innerHTML).

Code079.htm

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion événementielle</title>
    <script>
      function faitecho(ev)
      {
        document.getElementById("echo").innerHTML =
document.getElementById("echo").innerHTML + String.fromCharCode(ev.keyCode);
      }
    </script>
  </head>
  <body>
    <p>echo: <span id="echo"></span></p>
    <input type="text" id="saisie" onkeypress="faitecho(event);">
  </body>
</html>
```



# Evénements liés à la souris

Attribut	Exécution du script
<u><a href="#">onclick</a></u>	Au clic du bouton gauche
<u><a href="#">ondblclick</a></u>	Au double-clic du bouton gauche
<u><a href="#">ondrag</a></u>	Lorsqu'un élément est déplacé par la technique du glisser-déposer
<u><a href="#">ondragend</a></u>	Lorsqu'un élément a fini d'être déplacé par la technique du glisser-déposer
<u><a href="#">ondragenter</a></u>	Lorsqu'un élément a été déplacé sur une destination valide
<u><a href="#">ondragleave</a></u>	Lorsqu'un élément est déplacé depuis un emplacement valide
<u><a href="#">ondragover</a></u>	Lorsqu'un élément est en cours de déplacement vers une destination valide
<u><a href="#">ondragstart</a></u>	Au début du glisser-déposer
<u><a href="#">ondrop</a></u>	Au dépôt de l'élément sur la destination
<u><a href="#">onmousedown</a></u>	Lorsque le bouton de la souris est enfoncé
<u><a href="#">onmousemove</a></u>	Lorsque le pointeur se déplace
<u><a href="#">onmouseout</a></u>	Lorsque le pointeur se déplace en dehors d'un élément
<u><a href="#">onmouseover</a></u>	Lorsque le pointeur est déplacé sur un élément
<u><a href="#">onmouseup</a></u>	Au relâchement du bouton de la souris
<u><a href="#">onmousewheel</a></u>	Au déplacement de la roulette de la souris
<u><a href="#">onscroll</a></u>	Lorsque la barre de défilement de l'élément est utilisée

Ce code capture les événements souris liés à un élément img et les affiche dans un élément span.

Code080.htm

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion événementielle</title>
    <script>
      function traitement(param)
      {
        document.getElementById("activite").innerHTML = param;
      }
    </script>
  </head>
  <body>
    
    <p>Activité : <span id="activite"></span></p>
  </body>
</html>
```

## Exercice

Définissez un document HTML qui contient une image et un `<span>`. Lorsque l'utilisateur déplace la souris sur l'image, affichez ses coordonnées dans le `<span>`.

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Coordonnées de la souris</title>
    <script>
      function coord(e) {
        var y = e.clientY - 8;
        var x = e.clientX - 8;
        document.getElementById('xy').innerHTML = "x = " + x + ", y = " + y;
      };
    </script>
  </head>
  <body>
    <br/>
    <span id="xy"></span>
  </body>
</html>
```





## Exercice

Définissez un document HTML qui contient deux `<div>` dont le premier est éditable (attribut **`contenteditable="true"`**).

Capturez les caractères tapés dans le premier `<div>` et affichez-les en majuscules dans le deuxième.

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Transformation des touches frappées</title>
    <script>
      function capture(e) {
        var car = e.charCode;
        document.getElementById('span2').innerHTML = String.fromCharCode(car).toUpperCase();
      };
    </script>
  </head>
  <body>
    <div id="span1" style="width: 200px; height: 20px; border: 1px solid black; background-color: yellow;" onkeypress="capture(event);" contenteditable="true"></div><br>
    <div id="span2" style="width: 200px; height: 20px; border: 1px solid black; background-color: green;"> </div>
  </body>
</html>
```

# Événements liés aux formulaires

Ces événements sont déclenchés par les actions effectuées par l'utilisateur à l'intérieur d'un formulaire. Cinq d'entre eux sont une nouveauté du langage HTML5 : oncontextmenu, onformchange, onforminput, oninput et oninvalid.

Attribut	Exécution du script
<u>onblur</u>	Lorsqu'un élément perd le focus
<u>onchange</u>	Lorsque la valeur/le contenu d'un élément change
<u>oncontextmenu</u>	Lorsqu'un menu contextuel est déroulé
<u>onfocus</u>	Lorsqu'un élément reçoit le focus
<u>onformchange</u>	Lorsque le contenu du formulaire change
<u>onforminput</u>	Lorsque l'utilisateur entre des données dans le formulaire
<u>oninput</u>	Lorsqu'un élément reçoit des données entrées par l'utilisateur
<u>oninvalid</u>	Lorsqu'un élément n'est pas valide
<u>onselect</u>	Lorsqu'un élément est sélectionné
<u>onsubmit</u>	Lorsque le formulaire est soumis (généralement au clic sur le bouton Submit)

Ce code capture les actions effectuées sur une zone de texte et un bouton Submit et les affiche dans un élément span.

Code08I.htm

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion événementielle</title>
    <script>
      function traitement(param)
      {
        document.getElementById("activite").innerHTML = param;
      }
    </script>
  </head>
  <body>
    <form name="MonFormulaire" method="post">
      <label>Quel est le meilleur système d'exploitation selon vous ?</label>
      <input name="texte"
        placeholder="Entrez votre réponse ici"
        onforminput="traitement('Saisie de données');"
        onformchange="traitement('Le contenu du formulaire change');"
        onfocus="traitement('La zone de texte a le focus');"
        onblur="traitement('La zone de texte a perdu le focus');"
      >
      <input type="submit" value="Envoyer" onsubmit="traitement('Le bouton Submit a été pressé');">
    </form>
    <p>Activité : <span id="activite"></span></p>
  </body>
</html>
```



# Événements liés aux médias

Ces événements s'appliquent aux éléments video, image et audio. Mis à part l'événement onabort, ils sont tous apparus dans HTML5.

Attribut	Exécution du script
<u>onabort</u>	Sur l'événement "abort"
<u>oncanplay</u>	Lorsque le média peut commencer à être lu (il peut être amené à s'arrêter si le buffer de lecture devient vide)
<u>oncanplaythrough</u>	Lorsque le média peut être lu sans interruption jusqu'à la fin
<u>ondurationchange</u>	Lorsque la longueur du média change
<u>onemptied</u>	Lorsque le média n'est plus accessible à la suite d'un problème de réseau ou d'une erreur de chargement par exemple
<u>onended</u>	Lorsque le média a été entièrement joué
<u>onerror</u>	Lorsqu'une erreur survient pendant le chargement du média
<u>onloadeddata</u>	Lorsque les données du média ont été chargées
<u>onloadedmetadata</u>	Lorsque la durée et les autres caractéristiques du média ont été lues
<u>onloadstart</u>	Lorsque le navigateur commence à charger les données du média
<u>onpause</u>	Lorsque le média est mis en pause
<u>onplay</u>	Lorsque le média est mis en lecture
<u>onplaying</u>	Lorsque le média a commencé à être joué
<u>onprogress</u>	Lorsque l'élément est en cours de récupération des données pour le média
<u>onratechange</u>	Lorsque la vitesse de lecture change
<u>onreadystatechange</u>	Lorsque l'état (prêt/pas prêt) du média change
<u>onseeked</u>	Lorsque la recherche a pris fin
<u>onseeking</u>	Pendant la recherche (attribut <u>seeking=true</u> )
<u>onstalled</u>	Lorsqu'une erreur est rencontrée lors de la récupération des données
<u>onsuspend</u>	Lorsque la récupération des données est arrêtée avant la fin
<u>ontimeupdate</u>	Lorsque la position de lecture change
<u>onvolumechange</u>	Lorsque le volume du média est modifié
<u>onwaiting</u>	Lorsque le média n'est plus en mode de lecture mais que l'utilisateur peut demander une relecture

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Gestion événementielle</title>
    <script>
      function etat(param)
      {
        document.getElementById("activite").innerHTML = param;
      }
    </script>
  </head>
  <body>
    <video id="video" src="https://www.youtube.com/watch?v=zas5BE2FvBc"
      controls="controls"
      onabort="etat('onabort');"
      oncanplay="etat('oncanplay');"
      oncanplaythrough="etat('oncanplaythrough');"
      ondurationchange="etat('ondurationchange');"
      onemptied="etat('onemptied');"
      onended="etat('onended');"
      onerror="etat('onerror');"
      onloadeddata="etat('onloadeddata');"
      onloadedmetadata="etat('onloadedmetadata');"
      onloadstart="etat('onloadstart');"
      onpause="etat('onpause');"
      onplay="etat('onplay');"
      onplaying="etat('onplaying');"
      onprogress="etat('onprogress');"
      onratechange="etat('onratechange');"
      onreadystatechange="etat('onreadystatechange');"
      onseeked="etat('onseeked');"
      onseeking="etat('onseeking');"
      onstalled="etat('onstalled');"
      onsuspend="etat('onsuspend');"
      ontimeupdate="etat('ontimeupdate');"
      onvolumechange="etat('onvolumechange');"
      onwaiting="etat('onwaiting');"
    >
  </video>
  <p>Activité : <span id="activite"></span></p>
</body>
</html>

```

Code082.htm

Cet exemple affiche une vidéo issue du site Mediaforma ([www.mediaforma.com](http://www.mediaforma.com)) dans un élément video. Tous les événements liés à cet élément sont capturés et affichés dans un élément span, en dessous de l'élément video

# Événements liés à l'orientation

Les ordinateurs récents, les téléphones mobiles et les tablettes sont équipés de capteurs qui fournissent des informations sur l'orientation, le mouvement et l'accélération de ces appareils. Certains navigateurs web donnent accès à ces informations.

Pour savoir si le navigateur peut détecter le changement d'orientation, vous utiliserez la fonction **window.DeviceOrientationEvent**.

Si cette fonction retourne true, le navigateur peut détecter le changement d'orientation. Si elle retourne false, le navigateur n'est pas en mesure de détecter l'orientation de l'appareil :

```
if (window.DeviceOrientationEvent) alert('DeviceOrientation supporté');  
else alert('DeviceOrientation non supporté');
```



# Événements liés à l'orientation

Si l'orientation de l'appareil peut être détectée, définissez un gestionnaire d'événements pour l'événement **orientationchange** :

```
window.addEventListener('orientationchange', function(event) {}, false);
```



# Événements liés à l'orientation

Ces quelques lignes de code affichent l'orientation du matériel dans un élément span :

```
<!DOCTYPE html>
<html>
  <head>
    <title>device Orientation</title>
    <script>
      if (window.DeviceOrientationEvent) {
        alert('DeviceOrientation supporté');
        window.addEventListener('orientationchange',
          function(event) {
            document.getElementById('status').innerHTML='orientation : ' +
              window.screen.orientation.angle + ' degrés';
          }, false);
      }
    </script>
  </head>
  <body>
    <span id="status">Modifiez l'orientation de votre device</span>
  </body>
</html>
```

# Événements liés à l'orientation

Supposons maintenant que vous vouliez afficher l'inclinaison d'un appareil. Commencez par tester la valeur retournée par `window.DeviceMotionEvent` : `true` indique que le navigateur est compatible avec cette fonctionnalité, `false` indique qu'il ne l'est pas :

```
if (window.DeviceMotionEvent)
    alert('DeviceMotion supporté');
Else
    alert('DeviceMotion non supporté');
```

Si l'inclinaison de l'appareil peut être détectée, définissez un gestionnaire d'événements pour l'événement `devicemotion` :

```
window.addEventListener('devicemotion', function(event){}, false);
```

# Événements liés à l'orientation

Ces quelques lignes de code suivantes affichent l'inclinaison du matériel selon les axes X,Y,Z dans un élément span :

```
<!DOCTYPE html>
<html>
  <head>
    <title>device Motion</title>
    <script>
      if (window.DeviceMotionEvent) {
        window.addEventListener('devicemotion', function(event) {
          var x = event.accelerationIncludingGravity.x;
          var y = event.accelerationIncludingGravity.y;
          var z = event.accelerationIncludingGravity.z;
          document.getElementById('status').innerHTML =
            '<ul><li>X : ' + x + '</li><li>Y : ' + y +
            '</li><li>Z : ' + z + '</li></ul>';
        }, false);
      }
    </script>
  </head>
  <body>
    <span id="status">Modifiez l'inclinaison de votre device</span>
  </body>
</html>
```

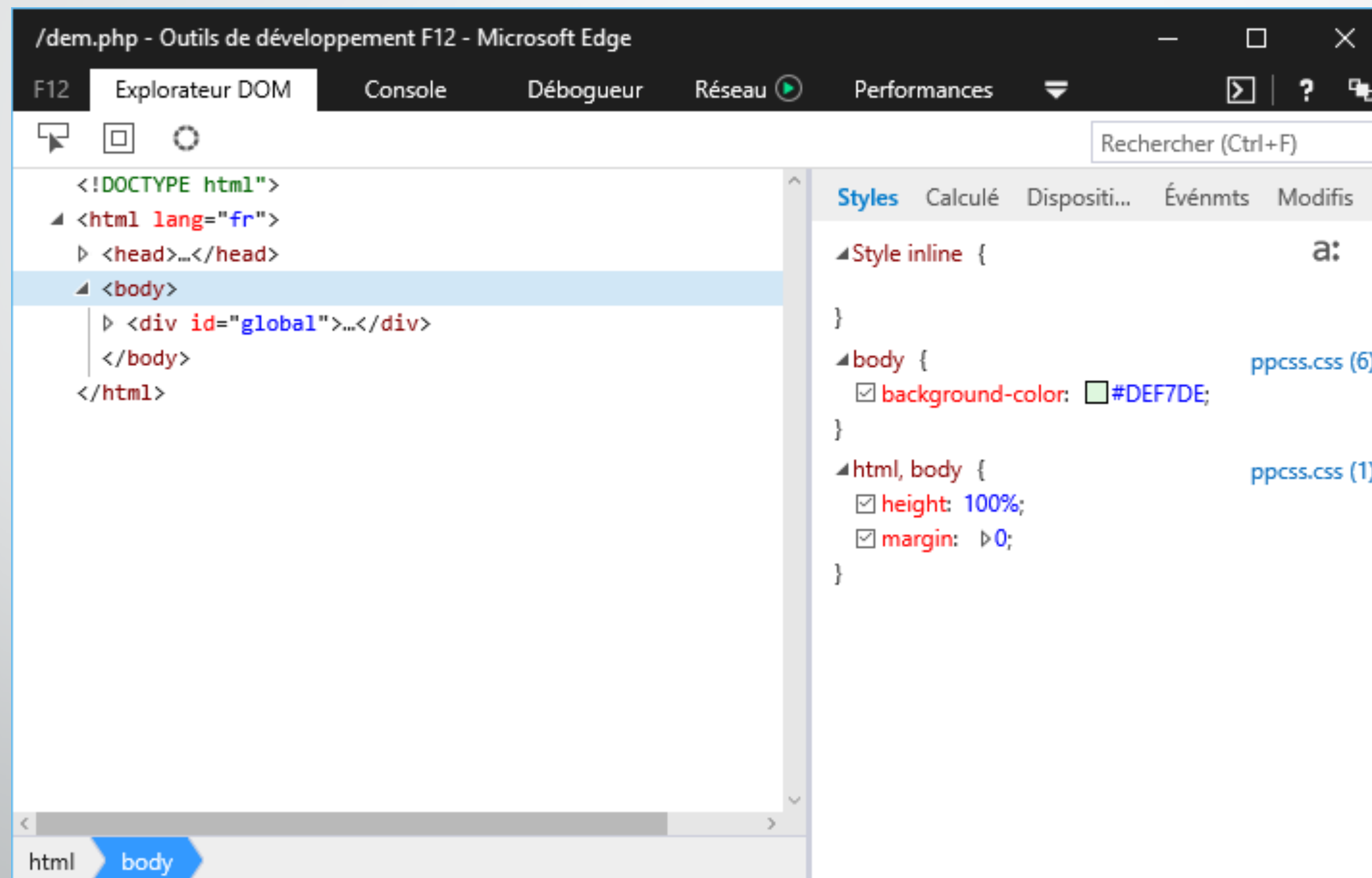


# Débogage dans la console du navigateur

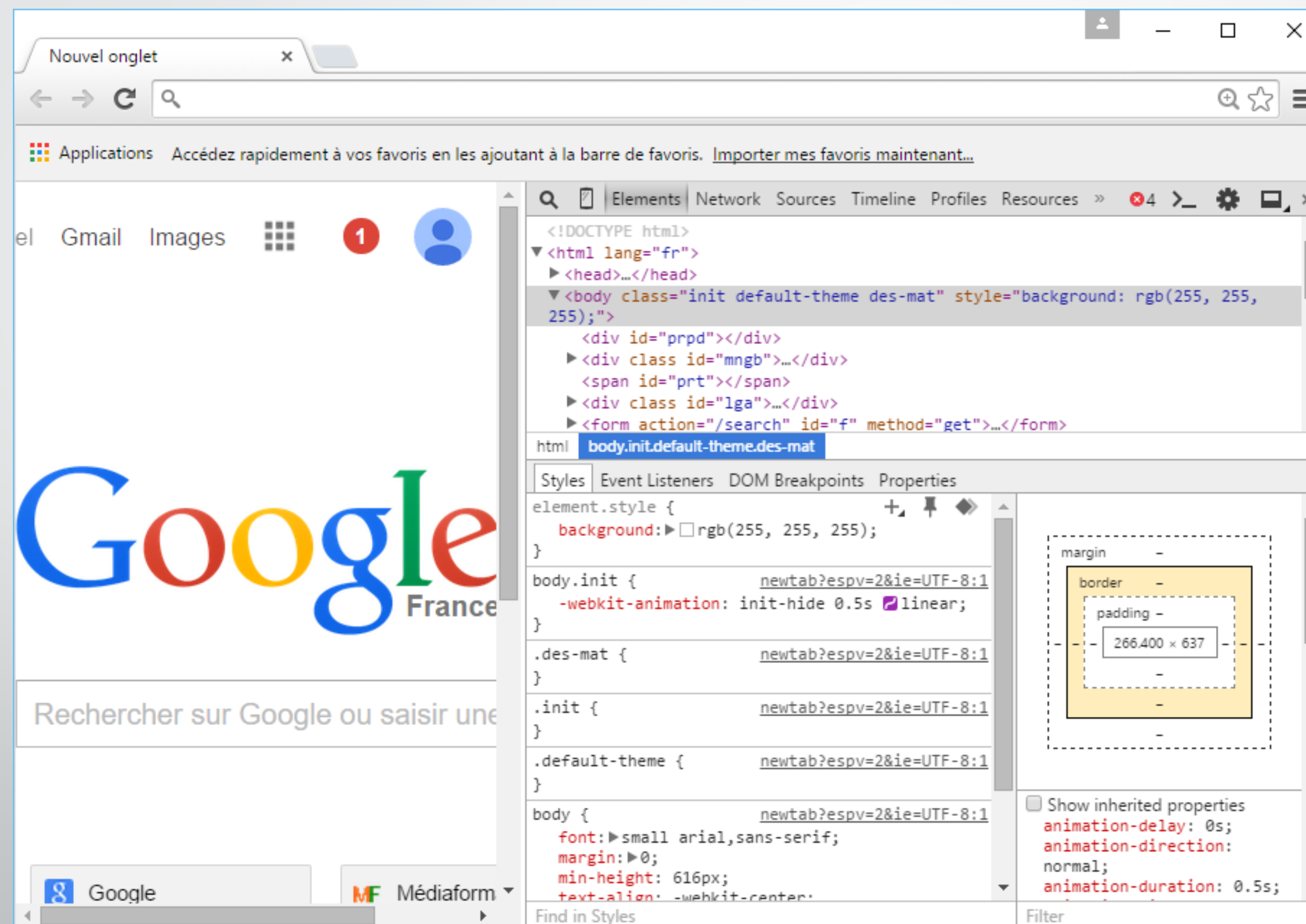



# Débogage dans le navigateur

Tous les navigateurs Web actuels intègrent un débogueur accessible avec la touche de fonction **F12**. Ici, par exemple dans Microsoft Edge :



Ou encore dans Google Chrome :





Quel que soit le navigateur utilisé, les possibilités sont identiques. La différence se fait au niveau de l'interface. Les possibilités sont les suivantes :

- Console pour voir les erreurs de code
- Surlignage du DOM en pointant une instruction HTML et inversement
- Modification live du code CSS des éléments

Vous avez eu l'occasion de tester toutes ces possibilités tout au long de cette formation.



# Introduction à la POO



# Introduction à la POO

La programmation orientée objet repose sur plusieurs notions fondamentales :

- Les objets
- Les classes
- Les instances
- Les membres de classe et d'instance
- L'encapsulation
- L'héritage
- Le polymorphisme

# Objets

Pour faire un parallèle avec le monde réel, les objets sont omniprésents : humains, livres, plantes, etc.

En POO, un objet est caractérisé par une **identité**, des **états** et des **comportements**.

- **L'identité** permet d'identifier l'objet sans ambiguïté
- **Les états** sont stockés dans des propriétés (équivalentes à des variables)
- **Les comportements** sont implémentés à l'aide de méthodes (procédures et fonctions)

A titre d'exemple, voici comment pourrait être modélisé un objet "être humain" :

- **Identité** : nom ou numéro
- **Etats** : taille, poids, couleur des yeux, etc.
- **Comportements** : respirer, marcher, parler, etc.

# Classes

Le monde réel regroupe des objets du même type. Il est pratique de concevoir une maquette (un moule) d'un objet et de produire les objets à partir de cette maquette.

En POO, une maquette se nomme une **classe**. Une classe est donc un modèle de la structure statique (variables d'instance) et du comportement dynamique (les méthodes) des objets associés à cette classe





# Instances

Les objets associés à une classe se nomment des **instances**. Une instance est un objet, occurrence d'une classe, qui possède la structure définie par la classe et sur lequel les opérations définies dans la classe peuvent être appliquées.

*C'est un gâteau issu du moule.*





# Héritage

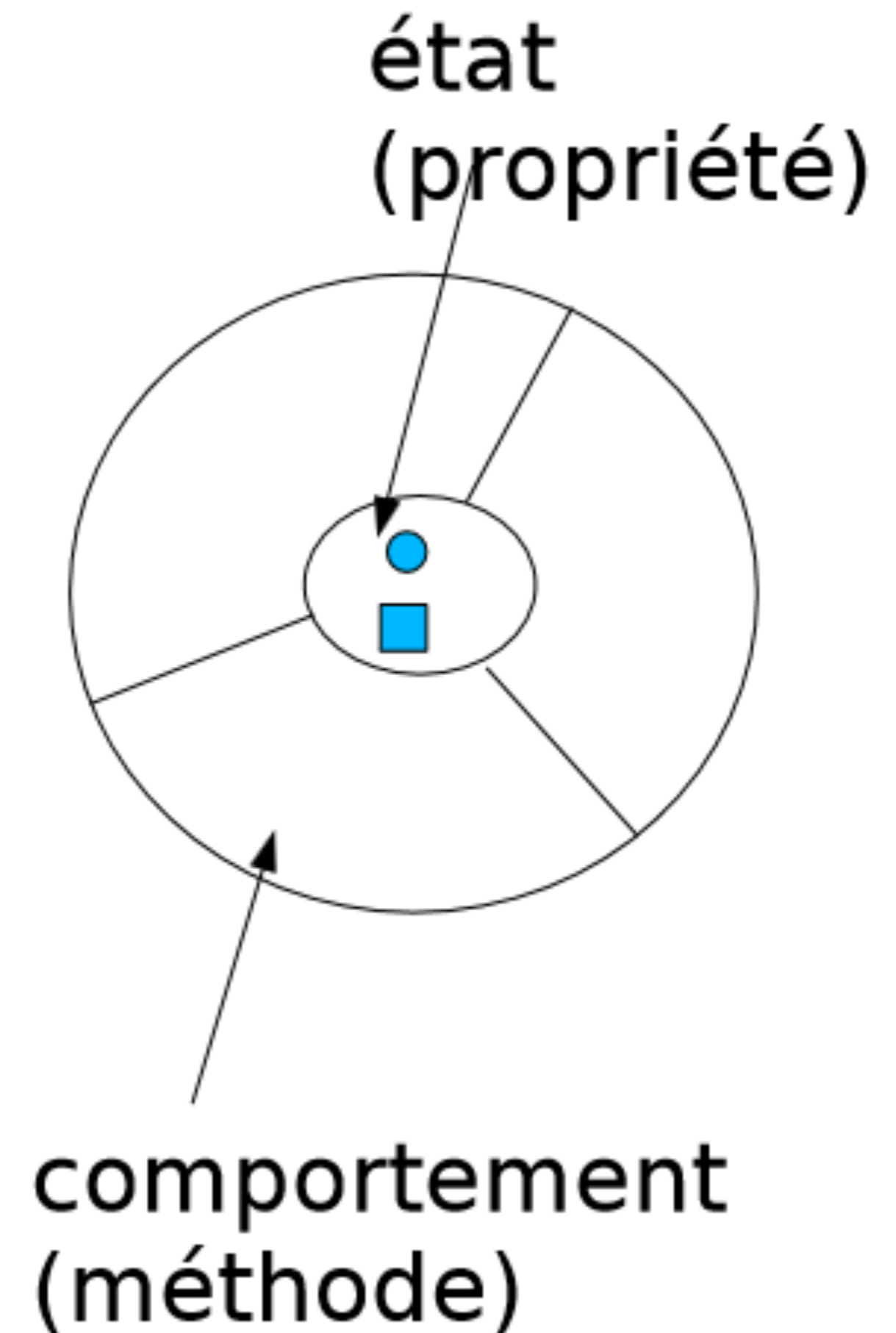
Il est possible de créer un nouveau type d'objet en prenant comme modèle un type objet déjà existant. Le nouvel objet possèdera (héritera) les mêmes champs et les mêmes méthodes que son modèle, avec toutefois la possibilité d'ajouter des nouveaux champs, d'ajouter des nouvelles méthodes ou de redéfinir les méthodes existantes. On dira que le nouvel objet est un objet **dérivé** ou un **descendant** de l'ancien.

Cette notion d'héritage permet de ne pas avoir à réécrire des méthodes déjà écrites. D'autre part les corrections ou modifications du code pourront être réalisées plus rapidement et de façon plus sûre.

# Encapsulation

En programmation objet, le concept d'encapsulation consiste à protéger l'information contenue dans un objet (ses propriétés et ses données) et de ne proposer que des méthodes spécifiques à la manipulation de cet objet.

Ainsi, les propriétés de l'objet ne seront accessibles et modifiables que par les méthodes de l'objet. Le programmeur ne pourra donc pas modifier directement les propriétés d'un objet et risquer de le mettre en péril.



# Polymorphisme

La notion de polymorphisme permet à une méthode commune à plusieurs objets descendants d'un même 'ancêtre' de fonctionner différemment selon l'objet qui l'appelle. De telles méthodes sont dites **virtuelles**.

Un exemple : Supposons que les classes **Rectangle** et **Cercle** héritent de la classe **ObjetGraphique**. On peut définir une instance **Rect1** d'un **Rectangle** comme un **ObjetGraphique**. Mais aussi, une instance **Cercle1** d'un **Cercle** comme un **ObjetGraphique**. Et par la suite, utiliser des méthodes **ObjetGraphique** sur les instances **Rect1** et **Cercle1**.

```
class ObjetGraphique {  
    methode1 () {  
    }  
    methode2 () {  
    }  
    ...  
}  
class Rectangle extends ObjetGraphique {  
    methode3 () {  
    }  
}  
class Cercle extends ObjetGraphique {  
    methode4 () {  
    }  
}
```



# ***JavaScript Objet en ES5***



# Extension des classes prédéfinies

Pour étendre une classe, vous passerez par son prototype.

Par exemple, pour ajouter la fonction **even()** à la classe **Number**, vous utiliserez la syntaxe suivante :

```
Number.prototype.even = function() {  
    return this.valueOf() % 2 === 0;  
}
```

Et pour utiliser la fonction **even()**, vous ferez ceci :

```
var a = 10;  
alert(a.even()); // true car 10 est pair  
a = 11;  
alert(a.even()); //false car 11 est impair
```

# Les objets par défaut de JavaScript

JavaScript peut manipuler :

- des variables non typées;
- des objets.

Plusieurs objets prédéfinis sont disponibles :

Number, Boolean, Date, Math, String **et** Array

Chacun possède des méthodes dédiées. Pour en savoir plus sur les méthodes disponibles, connectez-vous sur le site <http://www.w3schools.com/js/default.asp> et cliquez sur :

- JS Number Methods
- JS Booleans
- JS Date Methods
- JS Math
- JS String Methods
- JS Array Methods

dans le volet gauche.

Voici un extrait des méthodes de l'objet **Number** :

## Global Methods

JavaScript global functions can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

## Number Methods

JavaScript number methods are methods that can be used on numbers:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length
valueOf()	Returns a number as a number

Définition de la méthode toCapitalize() dans la classe String qui retourne la chaîne passée en mettant en majuscules la première lettre de chaque mot.

```
String.prototype.toCapitalize = function() {  
    var s = this.valueOf();  
    var mots = s.split(' ');  
    var resultat = '';  
    for (var i in mots) {  
        resultat += mots[i].slice(0,1).toUpperCase() +  
            mots[i].slice(1) + ' ';  
    }  
    return resultat.slice(-1);  
}
```

```
<script>  
    var s = 'Il fait beau aujourd\'hui';  
    alert(s.toCapitalize());  
</script>
```



# ***Définition d'un objet***

JavaScript implémente de nombreux types d'objets à caractère général. En utilisant une syntaxe particulière, il est possible de définir de nouveaux objets parfaitement adaptés à chaque situation.

Pour créer un nouvel objet, vous pouvez :

1) instancier l'objet **Object** :

```
var objet1 = new Object();  
//Description de l'objet
```

2) utiliser la notation JSON :

```
var objet2 = { // Description de l'objet};
```

# Création d'un nouvel objet en instanciant la classe Object()

Supposons qu'un commerçant souhaite créer un objet **reference** pour mémoriser le nom, le prix, la quantité en stock et la remise possible sur un des produits de son catalogue.

Pour cela, il définira l'objet **reference** qui possède les propriétés **nom**, **pu**, **quantite** et **remise**.

Supposons maintenant qu'il réapprovisionne de temps en temps le stock de ce produit à raison de 10 produits à chaque fois. Il pourra définir la méthode `ajoute10()` pour réapprovisionner son stock.

Voici le code à utiliser :

```
var reference = new Object();

reference.nom = 'HD';
reference.pu = 2.2;
reference.quantite = 10;
reference.remise = 5;

reference.ajoute10 = function() {
    reference.quantite = reference.quantite + 10;
}

reference.ajoute10();
console.log(reference.quantite);
```

La première instruction crée l'objet **reference**.

Les quatre instructions suivantes définissent les propriétés de l'objet **reference**.

Le bloc d'instructions suivant définit la méthode **ajoute10()** de l'objet **reference** et ajoute 10 à la propriété **quantite** de cet objet.

Le bloc d'instructions suivant appelle la fonction **ajoute10()** et affiche la valeur de la propriété **quantite** de l'objet **reference**.



Supposons maintenant que le commerçant veuille référencer plusieurs produits.

Le plus simple consiste à créer une fonction qui définit les propriétés et la méthode **ajoute10()** du produit, et de créer autant d'objets que nécessaire en utilisant l'opérateur **new**.



Voici le code à utiliser :

```
function Reference(nom, pu, quantite, remise) {  
    this.nom = nom;  
    this.pu = pu;  
    this.quantite = quantite;  
    this.remise = remise;  
    this.ajoute10 = function() {  
        this.quantite = this.quantite + 10;  
    }  
}
```

```
var E54V67 = new Reference("HD", 2.2, 10, 5);  
E54V67.ajoute10();  
console.log(E54V67.quantite);
```

# Définition d'un objet en JSON

Nous allons créer un objet **chat** de couleur noire et de nom Tosca :

```
var chat = {couleur: 'noir', nom: 'tosca'};
```

Si nécessaire, vous pouvez ajouter des propriétés à l'objet **chat** :

```
chat.age = 12;
```

Vous pouvez modifier une propriété existante :

```
chat.couleur = 'beige';
```

Voire même supprimer une propriété :

```
delete chat.age;
```

L'opérateur **in** permet de tester si un objet possède ou non une certaine propriété. Par exemple pour tester si la propriété **age** fait partie de l'objet **chat**, on utilisera cette syntaxe :

```
if ('age' in chat)
```


# Définition d'un objet en JSON

Il est possible d'ajouter des méthodes dans l'objet **chat** :

```
var chat = { couleur: 'noir',  
             nom: 'tosca',  
             miaule: function() {  
                 console.log('miaou');  
             }  
};
```

Pour faire miauler le chat, il suffit d'écrire :

```
chat.miaule();
```



# Jours 5 et 6

## jQuery



# jQuery et AJAX

Dans cette section :

- Pourquoi utiliser jQuery
- Installation et accès à jQuery (local et CDN)
- Modifier un élément du DOM
- Sélectionner des éléments dans le DOM
- Gestion événementielle
- AJAX - Echanges avec un serveur
- Le concept d'échanges asynchrones avec http
- Les fonctions `load()`, `$.get()`, `$.post()`, `$.getJSON()`, `$.ajax()`

# Introduction à jQuery

jQuery est une bibliothèque JavaScript libre qui simplifie la manipulation du code JavaScript côté client. Dans cette formation, vous allez apprendre à utiliser jQuery pour établir des échanges avec un serveur Web via AJAX.

# Introduction à jQuery

## Pourquoi utiliser jQuery ?

Les intérêts de jQuery sont multiples. Voici les principaux :

- Dynamiser les pages Web afin d'améliorer l'IHM (*Interface Home Machine*) et rendre la navigation aussi agréable et naturelle que possible.
- Uniformiser le rendu dans les différents navigateurs disponibles sur le marché.
- Diminuer les temps de développement en utilisant les fonctionnalités avancées de jQuery
- Faciliter la maintenance du code (le nombre d'instructions est bien plus limité qu'en JavaScript)
- Manipuler des ensembles de données, rendant souvent inutile l'utilisation de boucles
- Chaîner des méthodes pour effectuer plusieurs manipulations en une seule instruction

# Introduction à jQuery

## Installation et accès à jQuery

Pour "installer" jQuery, il suffit de faire référence à ce fichier. Vous pouvez pour cela :

1. Placer le fichier **jquery.js** à l'endroit où sont hébergés les autres fichiers qui constituent votre site Web.
2. Faire appel à un **CDN** (*Content Delivery Network*)

Dans un Intranet d'entreprise, le mieux est de placer jquery.js sur le serveur de l'entreprise.

Sur un site Web, le mieux est de faire appel à un CDN

En développement, le mieux est de placer le fichier jquery.js sur l'ordinateur du développeur



# Introduction à jQuery

Dans cette formation, nous utiliserons jQuery à partir d'un CDN

Vous n'avez rien à installer sur l'ordinateur local : il vous suffit de faire référence à la bibliothèque jQuery sur le CDN. Par exemple, en utilisant l'une des deux URL suivantes :

<http://code.jquery.com/jquery.min.js>

<http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js>

# Introduction à jQuery

La documentation sur jQuery

Toute la documentation sur jQuery se trouve sur <http://api.jquery.com/>

# Introduction à jQuery

## Où placer le code jQuery ?

Le code jQuery doit être placé entre les balises `<script>` et `</script>`, quelque part dans le corps du document, c'est-à-dire entre les balises `<body>` et `</body>`.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  <script src="jquery.min.js"></script>
  <script>
    <!-- Le code jQuery sera écrit ici -->
  </script>
</body>
</html>
```

# Vocabulaire jQuery

## Fonction jQuery

C'est le point d'entrée de la bibliothèque jQuery. Vous pouvez utiliser au choix l'instruction **jQuery()** ou son alias **\$()**. Dans ce cours, nous utiliserons systématiquement l'alias pour limiter l'écriture.



# Méthodes jQuery


La bibliothèque jQuery est constituée d'un ensemble de blocs de code autonomes appelés **méthodes**. Ce qui fait la puissance de cette bibliothèque, c'est avant tout la grande diversité des méthodes proposées. Pour exécuter une méthode jQuery, il suffit de préciser son nom à la suite d'un sélecteur en le séparant de ce dernier par un point :

**`$(sélecteur).méthode(paramètres);`**



## Objet jQuery

On appelle « objet jQuery » l'entité retournée par la fonction jQuery, c'est-à-dire par `$()`. Cet objet représente un ensemble de zéro, un ou plusieurs éléments issus du DOM.



Pour faire référence à la bibliothèque jQuery, vous utiliserez une balise `<script>` dans le `<head>` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Le titre du document</title>
    <script src="http://code.jquery.com/jquery.min.js"></script>
  </head>
  <body>
    <!-- Une ou plusieurs balises HTML pour définir le contenu du document -->
    <script>
      // Code jQuery
    </script>
  </body>
</html>
```

# Attendre la disponibilité du DOM

Le langage jQuery est utilisé pour manipuler (en lecture et en écriture) le DOM, c'est-à-dire l'arborescence du document. Il est important d'attendre la fin de la définition de l'arbre DOM avant de commencer à le manipuler. Sans quoi, des erreurs imprévisibles pourraient se produire...

Voici le code à utiliser :

```
$(function() {  
    // Le DOM a été entièrement défini  
    // Des instructions jQuery peuvent donc être exécutées  
});
```



# Syntaxe générale d'une instruction jQuery

Toutes les instructions jQuery sont construites sur le modèle suivant :

```
$ ( 'requête' ) .méthode (paramètre1, ... paramètreN) ;
```

La première partie requête le DOM. Il en résulte un ensemble de 0, 1 ou plusieurs éléments. la méthode jQuery est alors appliquée à cet ensemble en utilisant les paramètres entre parenthèses.

# Callbacks

Une fonction de rappel (ou **callback**) est exécutée lorsqu'une autre fonction a fini de s'exécuter. En jQuery, les fonctions de rappel sont essentiellement utilisées dans les animations et dans les appels AJAX.

Par exemple :


```
$('#element').slideUp(1000, function(){  
    // Une ou plusieurs instructions exécutées lorsque  
    // la méthode slideUp() a fini de s'exécuter  
});
```

# L'objet jQuery

## Chaînage des méthodes

En jQuery, il est possible (et souvent très efficace) de chaîner plusieurs méthodes. Il suffit pour cela de séparer les méthodes par un ".", comme par exemple dans :

```
$('#element').css('background','yellow').css('color','blue');
```



Vous en savez maintenant assez sur jQuery pour envoyer des requêtes AJAX.  
Mais avant de passer à la pratique, vous allez devoir installer un serveur local.



## WAMP Server

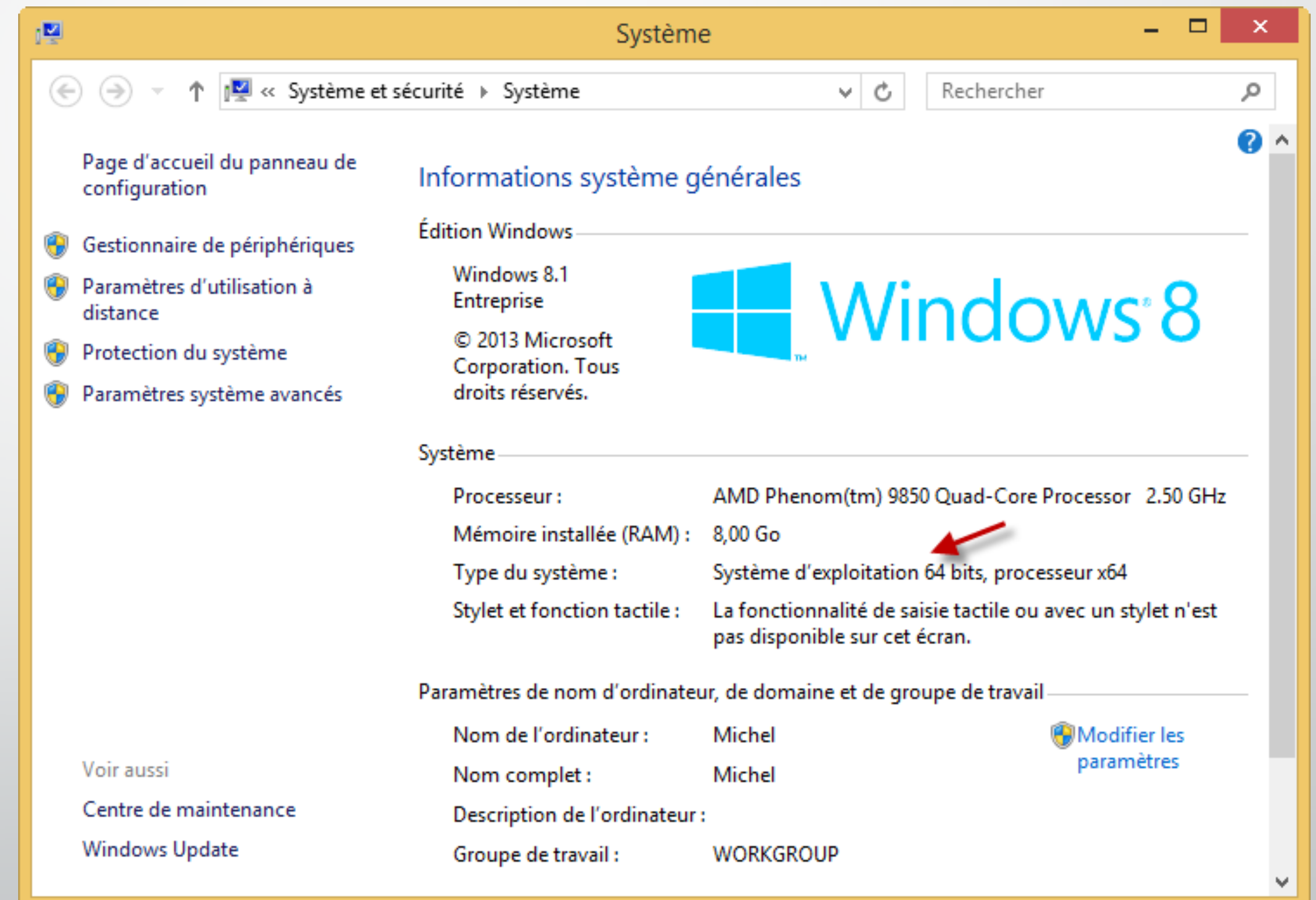
Pour exécuter du code AJAX localement, vous devez au préalable installer :

- 1) **Un serveur Web Apache.** C'est lui qui sera chargé de délivrer les pages Web aux visiteurs. Par défaut, Apache ne peut gérer que des sites statiques, constitués de pages HTML. Pour programmer en PHP, vous devez installer le support des langages PHP et MySQL dans ce serveur.
- 2) **Le plugin PHP pour Apache.** Ainsi, Apache sera en mesure de traiter des pages écrites en PHP.
- 3) **Le logiciel de gestion de bases de données MySQL.** Vous pourrez ainsi créer des bases de données et les interroger pour créer vos pages Web dynamiquement.

Plusieurs paquetages incluant Apache, PHP et MySQL sont disponibles. Par exemple :

- WAMP Server sous Windows ;
- MAMP sous Mac OS X ;
- XAMPP sous Linux.

Avant d'installer un paquetage, vérifiez le type de votre système d'exploitation : **32 ou 64 bits**. Pour cela, appuyez simultanément sur les touches *Windows* et *Pause*. La fenêtre **Système** s'affiche. L'information apparaît en face du libellé **Taille du système** :







Rendez-vous sur la page <http://sourceforge.net/projects/wampserver/> et installez le logiciel WAMP Server, en version 32 ou 64 bits selon le type de votre système d'exploitation. Pour cela, sélectionnez l'onglet **Files**, cliquez sur **Wamp Server 2**, sur la dernière version du logiciel (2.5 alors que j'écris ces lignes), puis sur la version 32 ou 64 bits en fonction de votre système.

A titre d'information, la version 32 bits est moins "lourde" que la version 64 bits. Il est donc facile de la reconnaître même si son nom n'est pas totalement affiché sur l'écran. Vous pouvez aussi pointer les liens proposés pour afficher le nom complet du fichier correspondant. Si vous voyez "32b" dans le nom du fichier, il s'agit d'une version 32 bits. Si vous voyez "64b", il s'agit d'une version 64 bits :

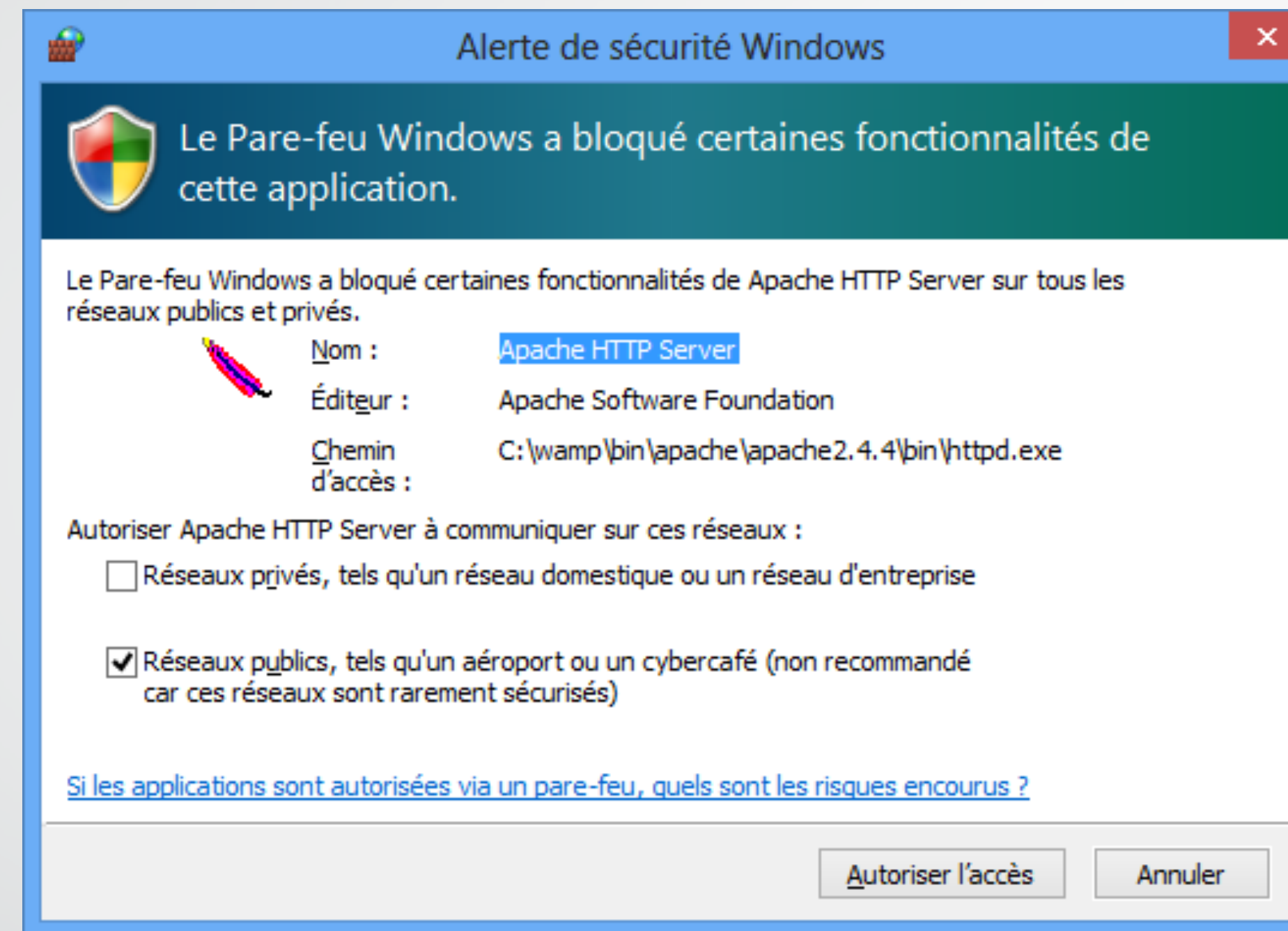
Looking for the latest version? [Download wampserver2.5-Apache-2.4.9-Mysql-5.6.17-php5.5.12-32b.exe \(39.9 MB\)](#)

[Home](#) / [WampServer 2](#) / Wampserver 2.5

Name ↕	Modified ↕	Size ↕	Downloads / Week ↕		
↑ <a href="#">Parent folder</a>					
<a href="#">wampserver2.5-Apache-2.4.9-Mysql...</a>	2014-05-01	39.9 MB	34 035		
<a href="#">wampserver2.5-Apache-2.4.9-Mysql...</a>	2014-05-01	43.5 MB	38 737		
Totals: 2 Items		83.4 MB	72 772		



Installez Wamp Server en conservant les options par défaut. A la fin de l'installation, le pare-feu de Windows se manifeste :



Cliquez sur **Autoriser l'accès** pour autoriser Apache à communiquer sur votre réseau.

Acceptez toutes les options par défaut jusqu'à la fin de l'installation.

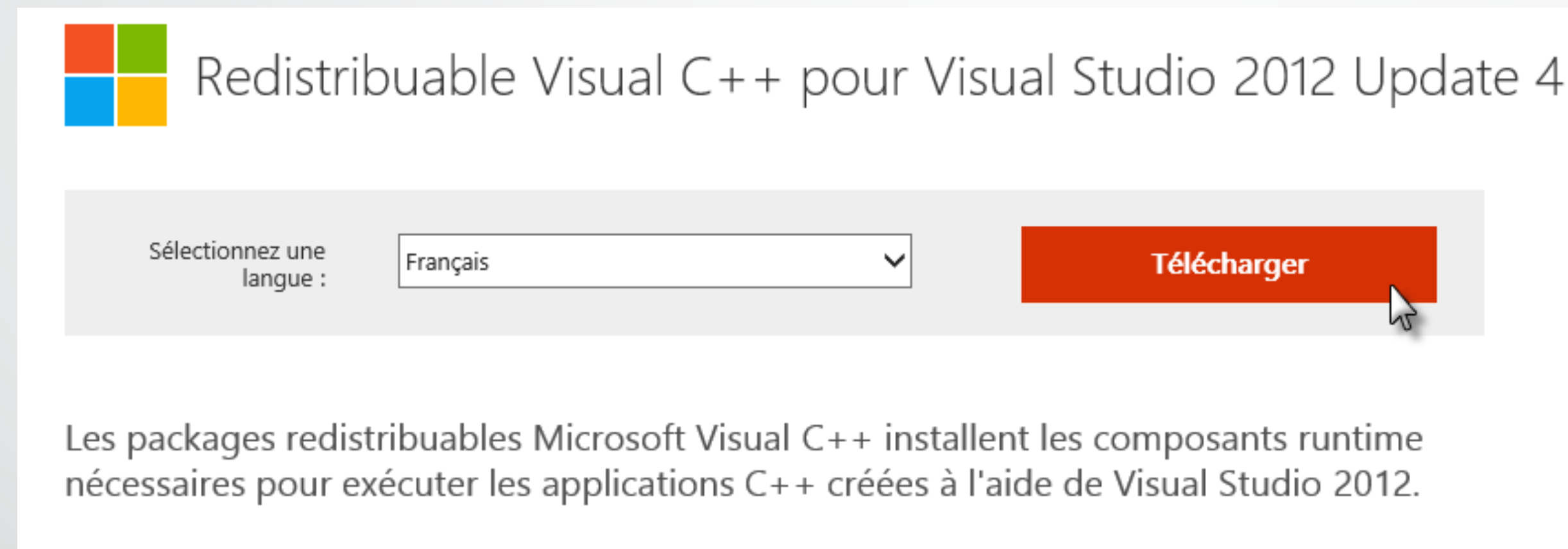


## Erreur d'installation

En cas d'erreur 0xc000007b sur la DLL msvcr100.dll lors de l'installation de WAMP Server, téléchargez et installez le logiciel Visual C++ Redistributable for Visual Studio 2012. Pour cela, rendez-vous sur la page suivante :

<http://www.microsoft.com/en-us/download/details.aspx?id=30679>

Choisissez **Français** dans la liste déroulante et cliquez sur **Télécharger** :



Choisissez la version **32 bits** ou **64 bits** en fonction de votre système d'exploitation et installez-la sur votre ordinateur :

Choisissez le téléchargement souhaité

<input type="checkbox"/> Nom du fichier	Taille
<input type="checkbox"/> VSU4\vcredist_arm.exe	1.4 MB
<input type="checkbox"/> VSU4\vcredist_x64.exe	6.9 MB
<input type="checkbox"/> VSU4\vcredist_x86.exe	6.3 MB

Désinstallez WAMP Server, redémarrez l'ordinateur et réinstallez-le.

### *WAMP Server s'est-il bien installé ?*

Une icône représentant WAMP Server devrait se trouver dans la Zone de notification. Si ce n'est pas le cas, tapez *wamp* dans le menu Démarrer (Windows XP/7) ou dans la page d'accueil (Windows 8) et cliquez sur **Start WampServer**. Quelques secondes plus tard, une icône représentant Wamp Server est disponible dans la Zone de notification :



Si l'icône de WAMP Server reste orange dans la zone de notification, il se peut que le service Apache ou MySQL ne fonctionne plus. Dans ce cas :

- cliquez sur l'icône de **WAMP Server**, puis sur **Apache, Service et Installer le service**.
- cliquez sur l'icône de **WAMP Server**, puis sur **MySQL, Service et Installer le service**.



Il se peut aussi que le **port 80** soit utilisé par un autre service que Apache.

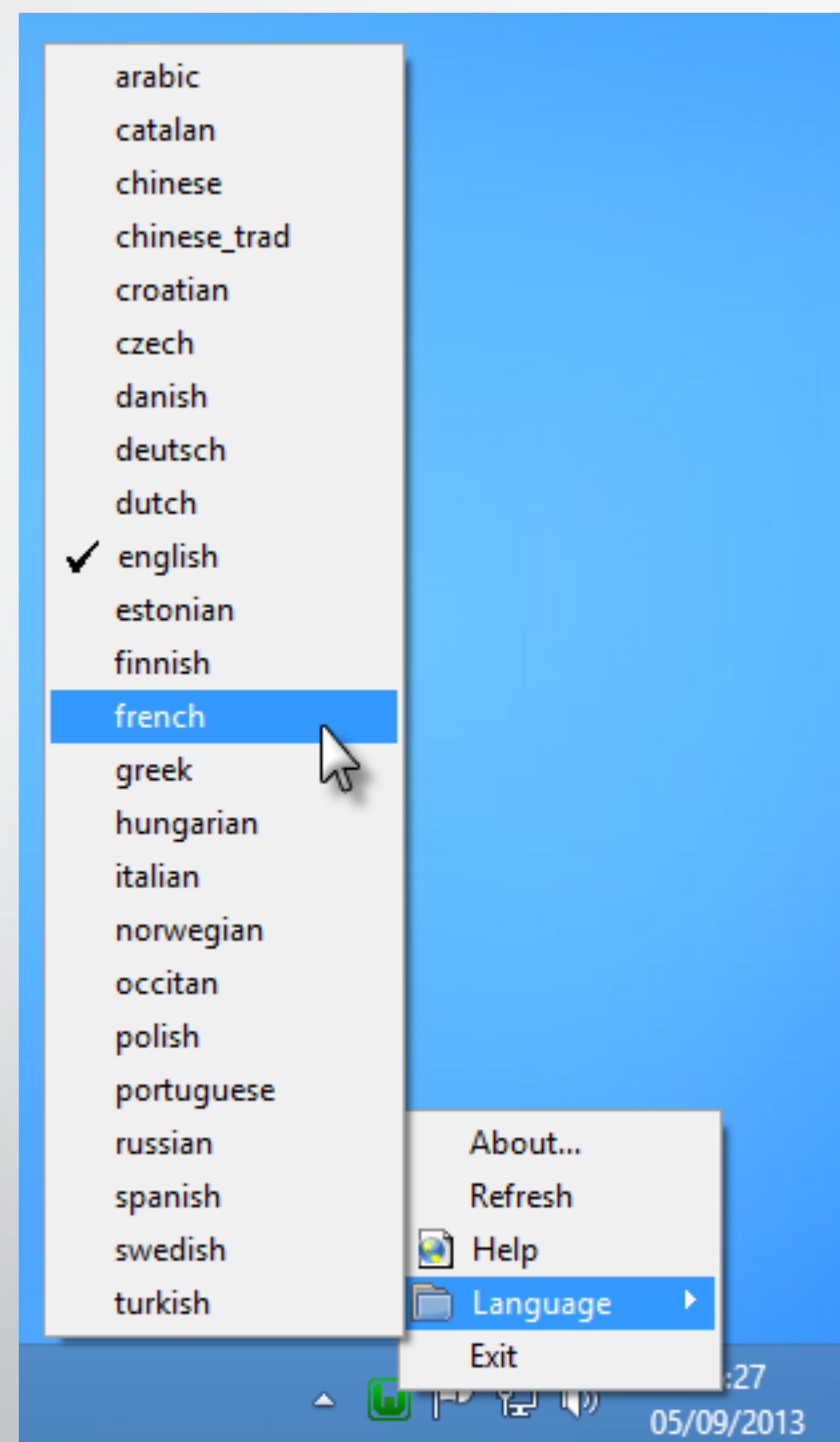
Cliquez sur l'icône de **WAMP Server**, puis sur **Apache**, **Service** et **Tester le port 80**.

Si le port 80 n'est pas utilisé par Apache, :

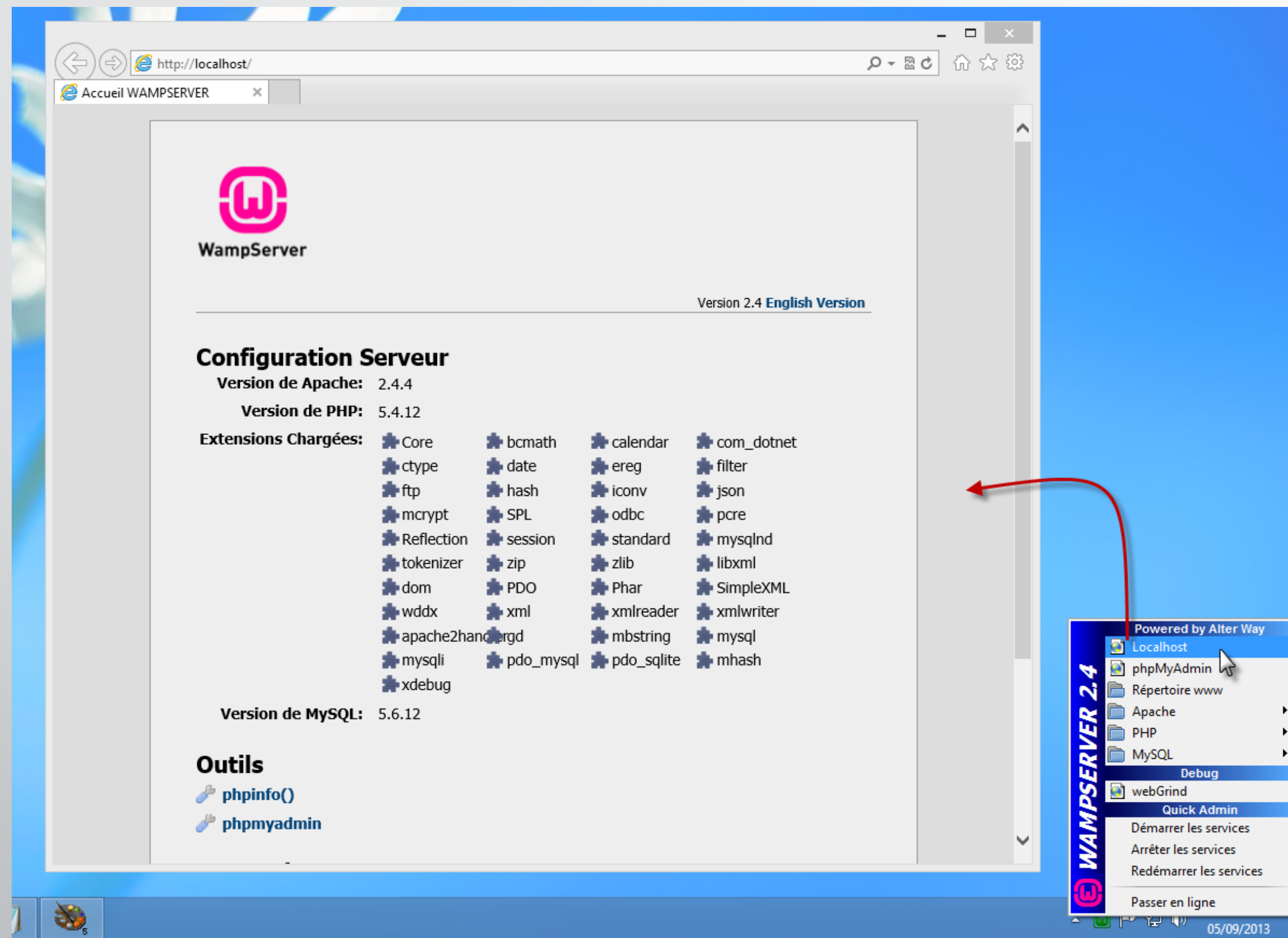
- cliquez sur l'icône de **WAMP Server**, sur **Apache** puis sur **httpd.conf**.
- recherchez le port **80** dans ce fichier et remplacez-le par **81**.
- fermez puis redémarrez **Wamp Server** pour prendre en compte la nouvelle configuration.



Par défaut, WAMP s'installe en anglais. Pour lui faire parler la langue de Molière, cliquez du bouton droit sur son icône, pointez **Language** et cliquez sur **French**.




Pour savoir si WAMP s'est bien installé, cliquez sur son icône et choisissez **Localhost** dans le menu. Au bout de quelques instants, la page d'accueil de WAMP Server s'affiche dans votre navigateur par défaut. Apache est donc opérationnel :



Si cette page refuse de s'afficher, relancez Apache. Pour cela, cliquez sur l'icône de **WAMP Server**, puis sur **Redémarrer les services**.

Si cela n'a toujours aucun effet, désinstallez puis réinstallez WAMP Server.



Ca y est, vous allez (enfin) passer à la pratique et lancer vos premières requêtes AJAX en jQuery !

Attention : tous les codes développés dans la suite de la formation ne fonctionnent que sur un serveur

# AJAX

## Charger un fichier

Pour mettre à jour un élément sur une page Web en utilisant des données stockées sur le serveur, le plus simple consiste à utiliser la méthode jQuery `load()` :

```
load('URL de l'élément', function() {  
    //une ou plusieurs instructions exécutées après le chargement des données  
});
```

La fonction callback est facultative. Si elle est présente, les instructions qui la composent seront exécutées lorsque le fichier aura été entièrement rapatrié par la méthode `load()`.

A titre d'exemple, le document de la diapo suivante (`load.htm`) contient deux boutons et un élément `div`. Le premier bouton va être utilisé pour afficher un texte dans l'élément `div` et le deuxième pour afficher une image dans ce même élément.



# AJAX

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Chargement AJAX avec load()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <button id="charge-texte">Charger le texte</button>
    <button id="charge-image">Charger l'image</button><br><br>
    <div id="zone"></div>
    <script>
      $(function() {
        $('#charge-texte').on('click', function(){
          $('#zone').load('texte.htm');
        });
        $('#charge-image').on('click', function(){
          $('#zone').load('image.htm');
        });
      });
    </script>
  </body>
</html>
```

Dans cet exemple, aucune fonction callback n'étant spécifiée dans les paramètres de la méthode **load()**, cette dernière se contente de charger les fichiers HTML correspondants et de les afficher dans l'élément **div**.

81.htm

# AJAX

Voici le code du fichier **texte.htm** :

```
<p>
  <font size="3"><i>Lorem <b>ipsum</b> dolor sit amet, consectetur adipiscing
elit. Sed non isus. Lectus tortor, dignissim sit amet, adipiscing nec,
ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa,
varius a, semper congue, euismod non, mi. Proin porttitor, orci nec
nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet
erat. Duis semper.</i></font>
</p>
```

Et voici le code du fichier **image.htm** :

```
<style type="text/css">
  p { text-align: center; }
</style>
<p></p>
```

# AJAX

## Charger une partie d'un fichier

En ajoutant un deuxième paramètre à la méthode `load()`, il est possible de limiter le chargement de données en utilisant un sélecteur jQuery :

```
load('URL sélecteur', function() {  
    //une ou plusieurs instructions exécutées après le chargement des données  
});
```

Où **URL** est l'URL de l'élément à charger et **sélecteur** est un sélecteur jQuery sans le signe `$` et sans les parenthèses.



✂ En partant du code de l'exemple précédent (**load.htm**), insérer trois boutons dans le corps du document et associez-leur un gestionnaire d'événement qui charge les parties **#p1**, **#p2** et **#p3** de ce document (**texte2.htm**) :

```
<p id="p1">
<font size="3"><i>Lorem <b>ipsum</b> dolor sit amet, consectetur adipiscing elit. Sed non
risus. Lectus tortor, dignissim sit amet, adipiscing nec,
ultricies sed, dolor. Cras elementum ultrices diam. Maecenas
ligula massa, varius a, semper congue, euismod non, mi. Proin
porttitor, orci nec nonummy molestie, enim est eleifend mi, non
fermentum diam nisl sit amet erat. Duis semper.</i></font>
</p>
```

```
<p id="p2">
Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vitae
dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas
sit aspernatur aut odit aut fugit, sed quia consequuntur magni
dolores eos qui ratione voluptatem sequi nesciunt.
</p>
```

```
<p id="p3">
Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet,
consectetur, adipisci velit, sed quia non numquam eius modi tempora
incidunt ut labore et dolore magnam aliquam quaerat voluptatem.
Ut enim ad minima veniam, quis nostrum exercitationem ullam
corporis suscipit laboriosam, nisi ut aliquid ex ea commodi
consequatur? Quis autem vel eum iure reprehenderit qui in ea
voluptate velit esse quam nihil molestiae consequatur, vel
illum qui dolorem eum fugiat quo voluptas nulla pariatur?
</p>
```

texte2.htm



# AJAX

## Solution

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Chargement AJAX avec load()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <button id="charge-texte1">Charger le texte #p1</button>
    <button id="charge-texte2">Charger le texte #p2</button>
    <button id="charge-texte3">Charger le texte #p3</button>
    <div id="zone"></div>
    <script>
      $(function() {
        $('#charge-texte1').on('click', function(){
          $('#zone').load('texte2.htm #p1');
        });
        $('#charge-texte2').on('click', function(){
          $('#zone').load('texte2.htm #p2');
        });
        $('#charge-texte3').on('click', function(){
          $('#zone').load('texte2.htm #p3');
        });
      });
    </script>
  </body>
</html>
```

82.htm

# AJAX

## Requête GET

La méthode `load()` n'est pas la seule à pouvoir récupérer des données *via* AJAX. Vous pouvez également utiliser :

- la fonction jQuery `$.get()` pour obtenir des données envoyées par le serveur en utilisant une requête **HTTP GET**.
- la fonction jQuery `$.post()` pour obtenir des données envoyées par le serveur en utilisant une requête **HTTP POST**.

Vous utiliserez la fonction `$.get()` si les données envoyées au serveur sont de petite taille. Vous utiliserez la fonction `$.post()` si les données envoyées au serveur sont de grande taille ou contiennent des informations confidentielles (des mots de passe par exemple).

# AJAX

Voici la syntaxe de la fonction **\$.get()** :

```
$.get(URL, function() {  
    // Une ou plusieurs instructions exécutées lorsque les données ont été  
    rapatriées  
});
```

A titre d'exemple, nous allons exécuter un fichier PHP afin d'extraire les données qui y sont stockées. Ces données correspondent aux trois lois de la robotique d'Isaac Asimov. L'URL passée sera du type suivant :

```
donnees.php?l=1 // pour obtenir la première loi  
donnees.php?l=2 // pour obtenir la deuxième loi  
donnees.php?l=3 // pour obtenir la troisième loi
```

# AJAX

## Voici le code du fichier PHP :

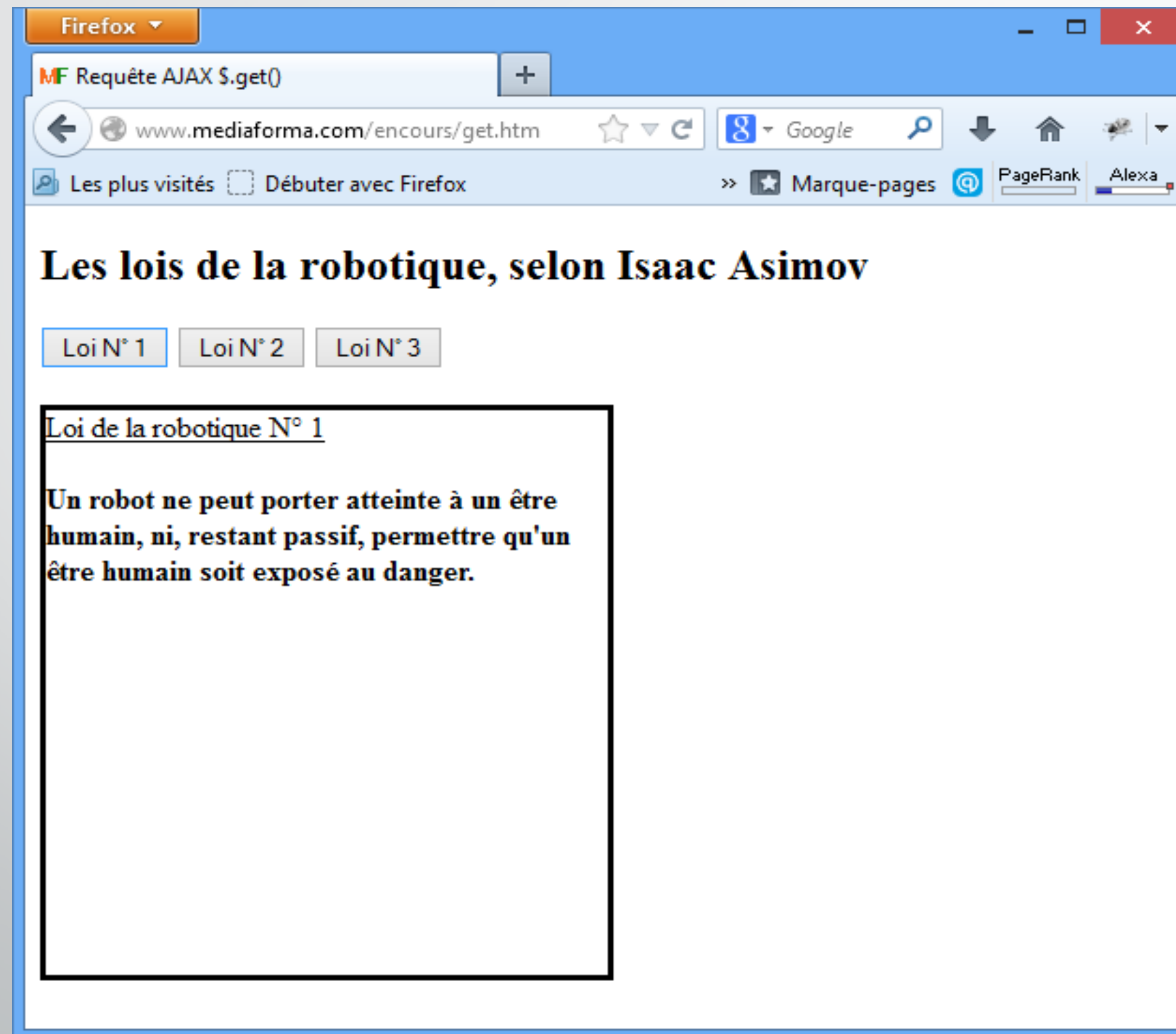
```
<?php
    $loi = array("Un robot ne peut porter atteinte à un être
humain, ni, restant passif, permettre qu'un être humain
soit exposé au danger.",
                "Un robot doit obéir aux ordres que lui
donne un être humain, sauf si de tels ordres entrent en
conflit avec la Première loi.",
                "Un robot doit protéger son existence tant
que cette protection n'entre pas en conflit avec la
Première ou la Deuxième loi.");
    $l=$_GET["l"];
    echo "<u>Loi de la robotique N° ".$l."</u><br><br>";
    echo "<b>".$loi[$l-1]."</b>";
?>
```

donnees.php



# AJAX

Voici le résultat obtenu lorsque l'utilisateur clique sur le premier bouton :



Et voici le code HTML5/jQuery utilisé :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Requête AJAX $.get()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <h2>Les lois de la robotique, selon Isaac Asimov</h2>
    <button id="loi1">Loi N° 1</button>
    <button id="loi2">Loi N° 2</button>
    <button id="loi3">Loi N° 3</button><br>
    <div id="zone"></div>
```

83.htm

Suite...

83.htm

```
<script>
    $(function() {
        $('#loi1').on('click', function() {
            $.get(' http://lem.korp.free.fr/jquery/donnees.php?l=1',function(data) {
                $('#zone').html(data);
            });
        });
        $('#loi2').on('click', function() {
            $.get('http://lem.korp.free.fr/jquery/donnees.php?l=2',function(data) {
                $('#zone').html(data);
            });
        });
        $('#loi3').on('click', function() {
            $.get('http://lem.korp.free.fr/jquery/donnees.php?l=3',function(data) {
                $('#zone').html(data);
            });
        });
    });
</script>
</body>
</html>
```

# AJAX

## Requête POST

Dans l'exemple précédent, les paramètres passés apparaissaient dans l'URL. Pour garder les paramètres secrets ou pour passer des données de taille importante, vous utiliserez une requête **POST**.

Voici la syntaxe de la fonction **\$.post()** :

```
$.post(URL, {donnée1: 'valeur1', donnée2: 'valeur2',...}, function() {  
    // Une ou plusieurs instructions exécutées lorsque les données ont été  
    rapatriées  
});
```

Ici, les données sont passées dans le deuxième paramètre de la fonction **\$.post()**.



# AJAX

Nous allons modifier le code du programme précédent pour accéder aux données *via* une requête **POST**.

L'URL interrogée aura pour nom **donneesPost.php**. Une seule donnée nommée "l" sera communiquée. Elle aura pour valeur 1, 2 ou 3 selon la loi à afficher.

Le programme PHP est légèrement différent :

```
<?php
    $loi = array("Un robot ne peut porter atteinte à un être humain, ni, restant
passif, permettre qu'un être humain soit exposé au danger.",
                "Un robot doit obéir aux ordres que lui donne un être humain, sauf
si de tels ordres entrent en conflit avec la Première loi.",
                "Un robot doit protéger son existence tant que cette protection
n'entre pas en conflit avec la Première ou la Deuxième loi.");
    $l=$_POST["l"];
    echo "<u>Loi de la robotique N° ".$l."</u><br><br>";
    echo "<b>".$loi[$l-1]."</b>";
?>
```

donneesPost.php

# Le code HTML5/jQuery est également légèrement différent :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Requête AJAX $.post()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <h2>Les lois de la robotique, selon Isaac Asimov</h2>
    <button id="loi1">Loi N° 1</button>
    <button id="loi2">Loi N° 2</button>
    <button id="loi3">Loi N° 3</button><br><br>
    <div id="zone"></div>
```

Suite ...

```
<script>
  $(function() {
    $('#loi1').on('click', function(){
      $.post(' http://lem.korp.free.fr/jquery/donneesPost.php',{l:'1'},function(data){
        $('#zone').html(data);
      });
    });
    $('#loi2').on('click', function(){
      $.post(' http://lem.korp.free.fr/jquery/donneesPost.php',{l:'2'},function(data){
        $('#zone').html(data);
      });
    });
    $('#loi3').on('click', function(){
      $.post(' http://lem.korp.free.fr/jquery/donneesPost.php',{l:'3'},function(data){
        $('#zone').html(data);
      });
    });
  });
</script>
</body>
</html>
```

# AJAX

## Charger des données JSON

Les fichiers au format **JSON** (*JavaScript Object Notation*) permettent de représenter des informations structurées.

Par exemple :

```
{
  "Employés": [
    { "Prénom": "John" , "Nom": "Doe" },
    { "Prénom": "Anna" , "Nom": "Smith" },
    { "Prénom": "Peter" , "Nom": "Jones" }
  ]
}
```

Ce qui équivaut à :

```
<Employés Prénom="John" Nom="Doe">
<Employés Prénom="Anna" Nom="Smith">
<Employés Prénom="Peter" Nom="Jones">
```



# AJAX

Pour accéder à des données **JSON**, vous utiliserez la fonction **\$.getJSON()** :

```
$.getJSON(URL, function (données) {  
    // Une ou plusieurs instructions pour traiter les données lues  
});
```

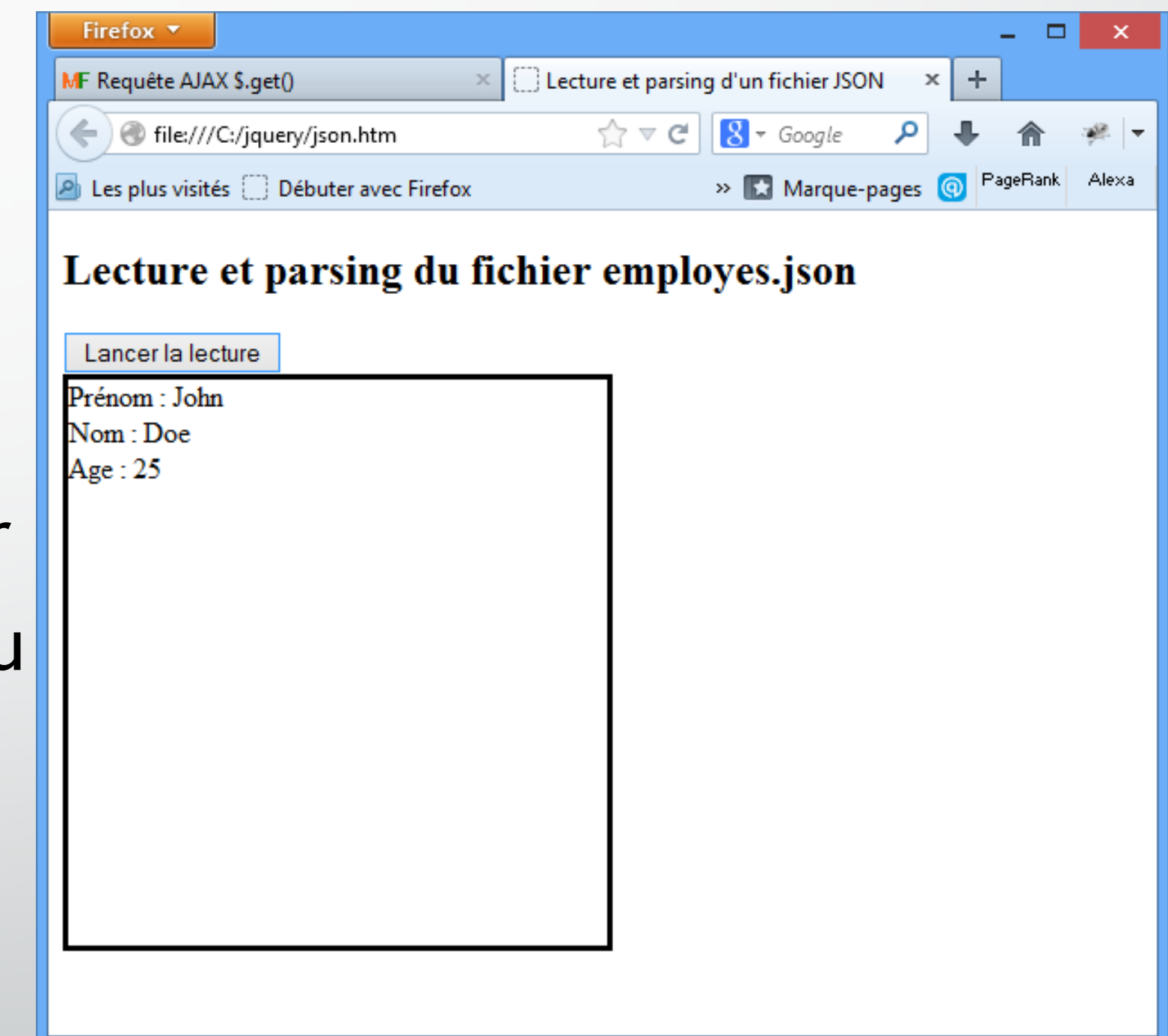
# AJAX

Saisissez ces données dans NotePad++ et sauvegardez-les dans le fichier **employees.json** :

```
{  
  "Prenom": "John",  
  "Nom": "Doe",  
  "Age": "25"  
}
```

employees.json

Nous allons accéder à ce fichier en jQuery et exploiter les données qui le composent. Voici le résultat attendu



# AJAX

Et voici le fichier HTML5/jQuery utilisé :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Lecture et parsing d'un fichier JSON</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
```

85.htm

Suite ...

```
<body>
  <script src="jquery.js"></script>
  <h2>Lecture et parsing du fichier employees.json</h2>
  <button id="lecture">Lancer la lecture</button>
  <div id="zone"></div>
  <script>
    $(function() {
      $('#lecture').on('click', function(){
        $.getJSON('employees.json',function(data){
          $('#zone').append('Prénom : ' + data.Prenom + '<br>');
          $('#zone').append('Nom : ' + data.Nom + '<br>');
          $('#zone').append('Age : ' + data.Age + '<br>');
        });
      });
    });
  </script>
</body>
</html>
```



# AJAX

L'extraction (aussi appelée **parsing**) des données se fait dans la fonction callback :

```
$.getJSON('employees.json', function(data) {  
    $('#zone').append('Prénom : ' + data.Prenom + '<br>');  
    $('#zone').append('Nom : ' + data.Nom + '<br>');  
    $('#zone').append('Age : ' + data.Age + '<br>');  
});
```

La structure du fichier JSON pris en exemple est ultra simple. Pour accéder à ses différentes composantes, on utilise le paramètre de la fonction callback, suffixé par le nom des différents champs à accéder :

```
data.Prenom  
data.Nom  
data.Age
```

## ✂ Exercice :

Créez le fichier **employees2.json** :

```
[  
  { "Prenom": "John" , "Nom": "Doe", "Age": "25" },  
  { "Prenom": "Anna" , "Nom": "Smith", "Age": "34" },  
  { "Prenom": "Peter" , "Nom": "Jones", "Age": "17" }  
]
```

employees2.json

Modifiez le code jQuery pour afficher toutes les données de ce fichier JSON. Pour cela, vous parcourrez les données retournées par la fonction **\$.getJSON()** en utilisant la fonction **\$.each()** dont voici la syntaxe :

```
$.each(données à examiner, function(index, données extraites){  
  // Instructions pour examiner les données extraites  
});
```

## ✂ Solution :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Lecture et parsing d'un fichier JSON</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
```

Suite ...

```
<body>
  <script src="jquery.js"></script>
  <h2>Lecture et parsing du fichier employees.json</h2>
  <button id="lecture">Lancer la lecture</button>
  <div id="zone"></div>
  <script>
    $(function() {
      $('#lecture').on('click', function(){
        $.getJSON('employees2.json',function(data){
          $.each(data,function(index,d){
            $('#zone').append('Prénom : ' + d.Prenom + '<br>');
            $('#zone').append('Nom : ' + d.Nom + '<br>');
            $('#zone').append('Age : ' + d.Age + '<br><br>');
          });
        });
      });
    });
  </script>
</body>
</html>
```



# AJAX

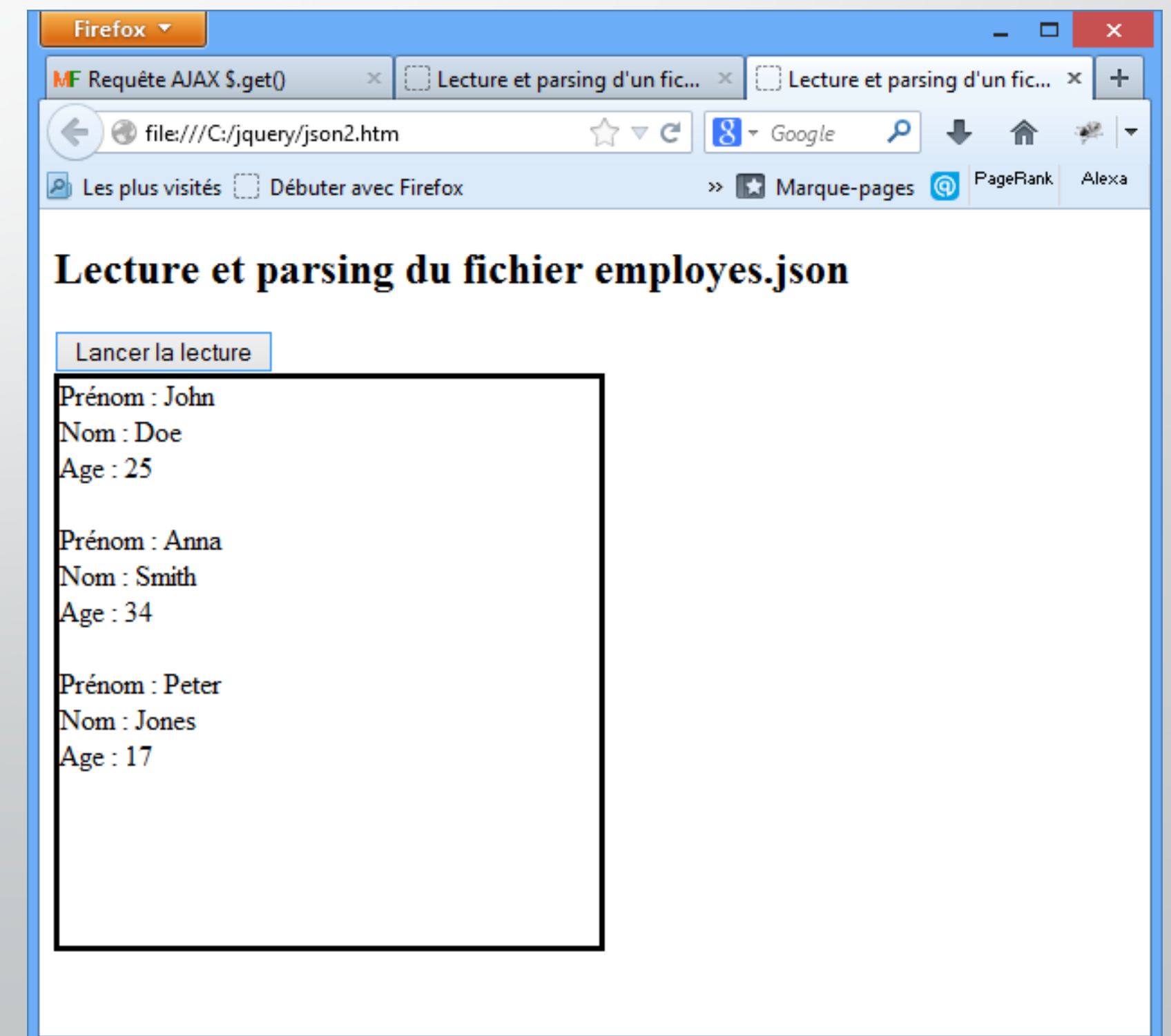
La fonction callback de `$.getJSON()` obtient la totalité des données du fichier JSON. La boucle `$.each()` porte sur les données extraites par `$.getJSON()`. Sa fonction callback va s'intéresser individuellement à chaque enregistrements, ici représentés par la lettre "d" :

```
$.getJSON('employees2.json',function(data){  
    $.each(data,function(index,d){
```

Pour extraire les données d'un enregistrement, il suffit de passer par "d" et de le suffixer par le nom des champs concernés : **Prenom**, **Nom** et **Age** :

```
$('#zone').append('Prénom : ' + d.Prenom + '<br>');  
$('#zone').append('Nom : ' + d.Nom + '<br>');  
$('#zone').append('Age : ' + d.Age + '<br><br>');
```

Voici le résultat :



# AJAX - Lecture de données XML

Voici un fichier XML nommé **data.xml** :

```
<sites>
  <site id="1">
    <title>Google</title>
    <url>http://www.google.fr</url>
  </site>
  <site id="2">
    <title>Microsoft</title>
    <url>https://www.microsoft.com/fr-fr/default.aspx</url>
  </site>
  <site id="3">
    <title>Mediaforma</title>
    <url>http://www.mediaforma.com</url>
  </site>
</sites>
```

Nous allons extraire les données de ce fichier et les afficher dans une balise `<div>`.  
Pour cela, nous utiliserons :

- La fonction `find()` pour trouver un nœud dans la structure ;
- La fonction `each()` pour boucler sur les nœuds `<site>`.

Voici le code utilisé :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Lecture et parsing XML</title>
    <style>
      #zone {
        width: 500px;
        height: 300px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
```

```
<body>
  <script src="jquery.js"></script>
  <h2>Lecture et parsing du fichier data.xml</h2>
  <button id="lecture">Lancer la lecture</button>
  <div id="zone"></div>
  <script>
    $(function() {
      $('#lecture').on('click', function(){
        $.get('data.xml', function(donnees) {
          $(donnees).find('site').each(function(){
            var id = $(this).attr('id');
            var title = $(this).find('title').text();
            var url = $(this).find('url').text();
            $('#zone').append(id + ' : <a href="' +
url + '"'>' + title + '</a><br>');
          });
        });
      });
    });
  </script>
</body>
</html>
```

Testez ce code

# AJAX

## La fonction \$.ajax()

La fonction `$.ajax()` permet d'émettre des requêtes AJAX. Elle admet de très nombreux paramètres. Pour avoir une description exhaustive de cette fonction, reportez-vous à la documentation en ligne : <http://api.jquery.com/jQuery.ajax/>.



# AJAX

Dans cette formation, nous utiliserons l'une des syntaxes de la fonction `$.ajax()` :

```
$.ajax(options);
```

Où options peut contenir les éléments suivants :

- **type** : type de la requête, **GET** ou **POST** (**GET** par défaut)
- **url** : adresse à laquelle la requête doit être envoyée
- **data** : données à envoyer au serveur
- **dataType** : type des données qui doivent être retournées par le serveur : **xml**, **html**, **script**, **json**, **text**
- **success** : fonction à appeler si la requête aboutit
- **error** : fonction à appeler si la requête n'aboutit pas
- **timeout** : délai maximum (en millisecondes) pour que la requête soit exécutée. Si ce délai est dépassé, la fonction spécifiée dans le paramètre **error** sera exécutée

A titre d'exemple, nous allons réécrire les programmes **get.htm** et **post.htm** pour utiliser la fonction `$.ajax()` à la place des fonctions `$.get()` et `$.post()`.

# Requête GET *via* la fonction \$.ajax()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Requête get $.ajax()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <h2>Les lois de la robotique, selon Isaac Asimov</h2>
    <button id="loi1">Loi N° 1</button>
    <button id="loi2">Loi N° 2</button>
    <button id="loi3">Loi N° 3</button><br><br>
    <div id="zone"></div>
```

88.htm

Suite ...

```
<script>
$(function() {
    $('#loi1').on('click', function() {
        $.ajax({
            type: 'GET',
            url: 'donnees.php?l=1',
            timeout: 3000,
            success: function(data) {
                $('#zone').html(data);
            },
            error: function() {
                $('#zone').html('Cette requête AJAX n\'a pas abouti');
            }
        });
    });
    $('#loi2').on('click', function() {
        $.ajax({
            type: 'GET',
            url: 'donnees.php?l=2',
            timeout: 3000,
            success: function(data) {
                $('#zone').html(data);
            },
            error: function() {
                $('#zone').html('Cette requête AJAX n\'a pas abouti');
            }
        });
    });
});
```

88.htm

Suite ...

88.htm

```
$( '#loi3' ).on( 'click', function() {  
    $.ajax({  
        type: 'GET',  
        url: 'donnees.php?l=3',  
        timeout: 3000,  
        success: function(data) {  
            $( '#zone' ).html( data );  
        },  
        error: function() {  
            $( '#zone' ).html( 'Cette requête AJAX n\'a pas abouti' );  
        }  
    });  
});  
</script>  
</body>  
</html>
```



# AJAX

Le code est bien plus "verbeux" qu'avec la fonction `$.get()`, mais il est aussi très simple à comprendre, facile à maintenir et plus complet. Ici, nous avons introduit un message d'erreur si la requête n'aboutit pas :

```
error: function() {  
    $('#zone').html('Cette requête AJAX n\'a pas abouti');  
}
```

Ce code est accessible ici : <http://www.mediaforma.com/encours/getAjax.htm>

# Requête POST *via* la fonction \$.ajax()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Requête post $.ajax()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
```

```
<body>
  <script src="jquery.js"></script>
  <h2>Les lois de la robotique, selon Isaac Asimov</h2>
  <button id="loi1">Loi N° 1</button>
  <button id="loi2">Loi N° 2</button>
  <button id="loi3">Loi N° 3</button><br><br>
  <div id="zone"></div>
  <script>
    $(function() {
      $('#loi1').on('click', function() {
        $.ajax({
          type: 'POST',
          url: 'donneesPost.php',
          data: {l:'1'},
          timeout: 3000,
          success: function(data) {
            $('#zone').html(data);
          },
          error: function() {
            $('#zone').html('Cette requête AJAX n\'a pas abouti');
          }
        });
      });
    });
  </script>
```

```
$('#loi2').on('click', function() {
    $.ajax({
        type: 'POST',
        url: 'donneesPost.php',
        data: {l:'2'},
        timeout: 3000,
        success: function(data) {
            $('#zone').html(data);
        },
        error: function() {
            $('#zone').html('Cette requête AJAX n\'a pas abouti');
        }
    });
});
```



```
$( '#loi3' ).on( 'click', function() {  
    $.ajax({  
        type: 'POST',  
        url: 'donneesPost.php',  
        data: {l: '3'},  
        timeout: 3000,  
        success: function( data ) {  
            $( '#zone' ).html( data );  
        },  
        error: function() {  
            $( '#zone' ).html( 'Cette requête AJAX n\'a pas abouti' );  
        }  
    });  
});  
</script>  
</body>  
</html>
```

Ce code est accessible ici :

<http://www.mediaforma.com/encours/postAjax.htm>

89.htm

# Lecture de données dans une base de données

Pour être en mesure de lire des données dans une base de données serveur en utilisant une requête AJAX, vous devez utiliser du code côté serveur. Ici par exemple, nous utiliserons du code PHP.

Voici les étapes que nous allons suivre :

- 1) PHP : Création d'une base de données et d'une table puis injection des données dans la table.
- 2) PHP : Création d'un programme d'interrogation de la base de données.
- 3) jQuery : Interrogation du programme créé dans l'étape 2 pour afficher les données de la table en AJAX avec (par exemple) une requête GET.

## Etape I – Création de la base, de la table et des données

```
<?php
// Création de la base de données
try {
    $base = new PDO('mysql:host=localhost', 'root', '');
}
catch(exception $e) {
    die('Erreur ' . $e->getMessage());
}
$base->exec("CREATE DATABASE basephp DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci");
$base = null;

// Création de la table
try {
    $base = new PDO('mysql:host=localhost; dbname=basephp', 'root', '');
}
catch(exception $e) {
    die('Erreur ' . $e->getMessage());
}
```

```
$base->exec("CREATE TABLE tablephp(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id), prenom
varchar(50), nom varchar(50), compteurvisite smallint, dernierevisite timestamp)");

// Ajout de données dans la table
$base->exec("INSERT INTO tablephp(prenom, nom, compteurvisite, dernierevisite)
VALUE('Pierre', 'Dubur', 34, NOW())");
$base->exec("INSERT INTO tablephp(prenom, nom, compteurvisite, dernierevisite)
VALUE('Chantal', 'Garnier', 128, NOW())");
$base->exec("INSERT INTO tablephp(prenom, nom, compteurvisite, dernierevisite)
VALUE('Jean', 'Dupont', 2, NOW())");
$base->exec("INSERT INTO tablephp(prenom, nom, compteurvisite, dernierevisite)
VALUE('Belle', 'Vercor', 45, NOW())");

$retour = $base->query('SELECT * FROM tablephp');
echo "<table border>";
while ($data = $retour->fetch()){
    echo "<tr><td>".$data['prenom']."</td>";
    echo "<td>".$data['nom']."</td>";
    echo "<td>".$data['compteurvisite']."</td>";
    echo "<td>".$data['dernierevisite']."</td></tr>";
}
echo "</table>";
$base = null;
?>
```

Co code est accessible sur <http://www.mediaforma.com/orsys/creation-donnees.php>  
Placez-le dans le dossier www de Wamp Server



## Etape 2 – Lecture des données dans la table

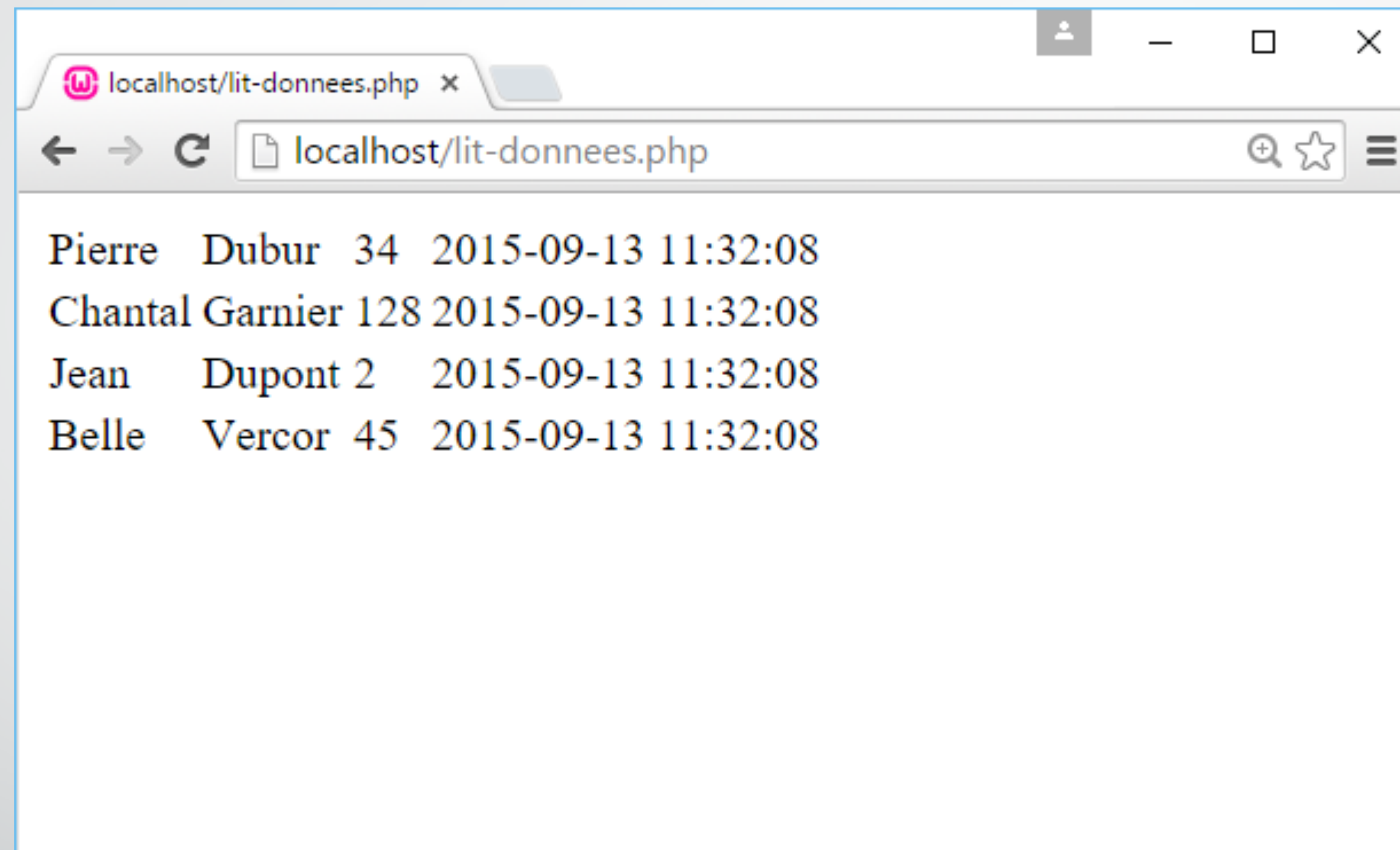
```
<?php
try {
    $base = new PDO('mysql:host=localhost; dbname=basephp', 'root', '');
}
catch(exception $e) {
    die('Erreur '.$e->getMessage());
}
$retour = $base->query('SELECT * FROM tablephp');
echo "<table border>";
while ($data = $retour->fetch()) {
    echo "<tr><td>".$data['prenom']."</td>";
    echo "<td>".$data['nom']."</td>";
    echo "<td>".$data['compteurvisite']."</td>";
    echo "<td>".$data['dernierevisite']."</td></tr>";
}
echo "</table>";
$base = null;
?>
```

Co code est accessible sur <http://www.mediaforma.com/orsys/lit-donnees.php>  
Placez-le dans le dossier www de Wamp Server

## Etape 3 – Obtention des données de la table en jQuery/AJAX

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Requête AJAX $.get()</title>
    <style>
      #zone {
        width: 300px;
        height: 315px;
        border-style: solid;
        border-width: 3px;
        border-color: black;
      }
    </style>
  </head>
  <body>
    <script src="jquery.js"></script>
    <h2>Récupération de données dans une base de données</h2>
    <button id="lecture">Lecture des données</button>
    <div id="zone"></div>
    <script>
      $(function() {
        $('#lecture').on('click', function(){
          $.get('lit-donnees.php',function(data){
            $('#zone').html(data);
          });
        });
      });
    </script>
  </body>
</html>
```

Voici le résultat :



Pierre	Dubur	34	2015-09-13 11:32:08
Chantal	Garnier	128	2015-09-13 11:32:08
Jean	Dupont	2	2015-09-13 11:32:08
Belle	Vercor	45	2015-09-13 11:32:08



La formation est terminée