JavaScript

M2 G2M, Univ. Paris 8
par Isis TRUCK

*Inspiré de diverses sources comme
<https://developer.mozilla.org/>
<http://www.commentcamarche.net/>
<http://www.toutjavascript.com>*

<http://www.thesitewizard.com/javascripts/change-style-sheets.shtml>

JavaScript : motivations

- On a vu dans le TP 1 (exercice 9) les limitations d'un langage de type HTML
- Il faut bien distinguer langage de programmation et langage de balisage (très restrictif!)
- Javascript est un **langage de script** (à mi-chemin entre les deux)
- Il n'a aucune utilité en tant que langage autonome mais est conçu pour **s'intégrer facilement** au sein d'un navigateur Web par exemple
- Lorsqu'utilisé au sein d'un environnement hôte, JavaScript peut être relié aux objets de cet environnement et ainsi fournir un **moyen de contrôler ces objets** par le biais de la programmation.
- Il est interprété sur le **poste du client**

Intérêts (parmi d'autres)

- L'exécution de code (JavaScript en particulier) sur le poste client peut:
 - Améliorer l'interactivité (**temps de réponse plus court** car inutile d'aller interroger le serveur)
 - **Améliorer les débits** sur le réseau (**éviter des envois contenant des erreurs**, et donc de surcharger inutilement le réseau, par exemple: test d'un formulaire avant l'envoi)

JavaScript de base

- Le JavaScript de base (le « noyau ») contient un ensemble d'objets de base (Array, Date et Math) ainsi qu'un ensemble d'éléments de base pour tout langage de programmation : **opérateurs, structures de contrôle** et autres instructions.
- il peut être *étendu* pour être applicable à n'importe quel domaine en lui ajoutant des objets supplémentaires. Exemples :
 - Le *JavaScript client* étend le langage de base en lui ajoutant des objets destinés à **contrôler un navigateur** et à manipuler le DOM. Par exemple, les extensions client permettent aux applications
 - d'ajouter des éléments à un formulaire HTML,
 - de répondre aux actions de l'utilisateur tels que les clics de souris
 - de gérer l'entrée des formulaires et la navigation entre les pages, etc.
 - Le *JavaScript serveur* étend le langage de base en fournissant des objets relatifs au **bon fonctionnement de JavaScript du côté du serveur**. Par exemple, les extensions serveur permettent aux applications de manipuler des fichiers présents sur le serveur.

API JavaScript

- Il y en a énormément ! Par exemple,
 - jQuery, voir TP 2
 - WebGL, génération et affichage d'images 2D et 3D dynamiques, fondé sur OpenGL
 - GreenSock Animation Platform (GSAP) : suite d'outils JS pour faire des animations HTML5 complexes et rapides
 - AngularJS : librairie permettant d'étendre HTML avec de nouveaux attributs. Suit la logique MVC (Modèle-Vue-Contrôleur). Utilise les *directives*, *expressions*, *filters*, *modules* et *controllers*. (voir par exemple ici : https://www.w3schools.com/angular/angular_intro.asp)
 - etc.

Interaction avec HTML (XHTML, HTML5...)

- Méthodes pour insérer du code JavaScript dans une page type HTML :
 - Utilisation de la balise `<script>` : code dans la page HTML
 - Déclaration des fonctions dans l'entête (`<head>`)
 - Appel de fonction ou exécution d'une commande JavaScript (dans `<body>`)
 - Balise `<script>` : insertion d'un fichier JavaScript **externe**
 - Déclaration du fichier .js dans l'entête (`<head>`)
 - Appel de fonction ou exécution d'une commande JavaScript (dans `<body>`)
 - Utilisation dans une URL :
 - Une URL peut être une exécution de fonction JavaScript (entre `<a>...</a>` ou `<form>...</form>`)
 - Utilisation de nouveaux attributs de balise pour la gestion d'événements utilisateur
 - `<balise onEvenement="Action_Javascript_ou_Fonction();">`
 - Exemples d'événements: `onClick`, `onMouseOver`...

Méthode 1 : JavaScript dans le HTML

- Ecriture de JavaScript au sein d'une page :
- Syntaxe:
 - Exple : Mettre une alerte, dans le head ou dans le body :

```
<script type="text/javascript">
```

```
  // <![CDATA[
```

```
    alert("Houston, we've got a problem");
```

```
  //]]>
```

```
</script>
```

```
<noscript>Votre navigateur ne supporte pas le javascript</noscript>
```

- *NB 1* : Ici, alert() n'a pas besoin d'être déclarée : c'est une fonction (méthode) de JavaScript, pas une fonction créée par l'utilisateur.
- *NB 2* : les éléments situés dans l'en-tête se comportent comme des déclarations, ils ne s'exécutent pas directement.

Les éléments situés dans le corps s'exécutent au fur et à mesure du chargement de la page

Méthode 1 : JavaScript dans le HTML (2)

- Exple 2: appeler une fonction créée par l'utilisateur :

```
<head>
```

```
    <script type="text/javascript">
```

```
        function maFonction(){ ...;}
```

```
    </script>
```

```
</head>
```

```
<body>
```

```
    <a href="javascript:maFonction();">appeler la  
    fonction</a>
```

```
</body>
```

Méthode 2 : insertion d'un fichier externe à la page

- Déclaration d'un fichier JavaScript externe à la page

=> mettre la balise `<script>` dans **<head>** :

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

```
<noscript>Votre navigateur ne supporte pas le
  javascript</noscript>
```

=> appeler la fonction dans **<body>** :

```
<a href="javascript:maFonction();">appeler la fonction</a>
```

Ou bien :

```
<p>passez la souris<a
  href="" onMouseOver="maFonction();">ici</a></p>
```

Éléments de langage

- Variables
 - = lieux de mémorisation des chiffres et des caractères.
- L'affectation permet d'attribuer une valeur à une variable
 - Syntaxe : `target = source`
 - Affectation conditionnelle: `var = (condition) ?
expr_alors : expr_sinon`
 - Pour assigner une chaîne de caractères, on la place entre guillemets doubles ou simples
 - Exemples :
 - `a = 3`
 - `b = "hello world"`
 - `c = 'hello from Paris 8'`
 - `a = b`
 - `a = 3*4*3+2`

Éléments de langage (2)

- Attention à bien distinguer l'affectation (=) de la comparaison (==)
- Il existe des notations abrégées comme:
 - $x+=y$ (équivalent à $x=x+y$)
 - $x--$ (équivalent à $x=x-1$ décrémentation)
- Commentaires : // commentaire sur 1 ligne
/* commentaires sur
plusieurs lignes */
- Séparateur d'instructions : ';' ou retour chariot

Types de variables ?

- **pas obligatoire de déclarer le type** de variables que l'on utilise en JavaScript, contrairement à des langages évolués comme C, C++ ou Java pour lesquels il faut préciser s'il s'agit d'entier (`int`), de nombre à virgule flottante (`float`), de caractères (`char`), etc.
 - => il s'agit d'un langage à **typage faible**
- **Portée** des variables : locale (uniquement dans le script ou la fonction) ou globale (en tout point du script)
 - lorsqu'une variable est déclarée sans le mot clé `var`, (de façon **implicite**), elle est accessible de partout dans le script (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de **variable globale**
 - La portée d'une variable déclarée **de façon explicite** (précédée du mot-clé `var`), dépend de l'endroit où elle est déclarée.
 - Une variable déclarée au début du script, avant toute fonction, sera globale. Elle peut être utilisée n'importe où dans le script
 - Une variable déclarée explicitement dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire qu'elle est inutilisable ailleurs. On parle alors de **variable locale**.

Types de variables (2)

- JavaScript n'autorise la manipulation que de 4 types de données :
 - des nombres : entiers ou à virgule
 - des chaînes de caractères (`string`) : une suite de caractères
 - des booléens (variables à deux états permettant de vérifier une condition) :
 - `false`: résultat est faux
 - `true`: résultat est vrai
 - des variables de type `null` : un mot caractéristique pour indiquer qu'il n'y a pas de données

Nom des variables

- nom d'une variable peut être aussi long que souhaité, mais il doit :
 - commencer par une lettre (majuscule ou minuscule) ou un "_"
 - comporter des lettres, des chiffres et les caractères _ et & (pas d'espaces !)
 - Il ne peut pas être un des noms suivants (noms réservés) :
 - abstract
 - boolean break byte
 - case catch char class const continue
 - debugger default delete do double
 - else export extends
 - false final finally float for function
 - goto
 - if, implements, import, in, infinity, instanceof, int, interface
 - label, long
 - native, new, null
 - package, private, protected, public
 - return
 - short, static, super, switch, synchronized
 - this, throw, throws, transient, true, try, typeof
 - var, void, volatile
 - while, with
 - (ainsi que tous les noms des objets Javascript)

Opérateurs (1)

Opérateurs logiques	
&&	Opérateur ET : implique que toutes les conditions soient réalisées
	Opérateur OU : implique qu'au moins une des conditions soit réalisée
!	Opérateur de négation : implique que la condition ne soit pas réalisée
Opérateurs de calcul	
+	Addition de deux valeurs numériques
-	Soustraction de deux valeurs numériques
*	Multiplication de deux valeurs numériques
/	Division de deux valeurs numériques
=	Affectation d'une valeur à une variable
%	Reste de la division entière entre deux valeurs numériques 9 % 3 vaut 0, 7 % 2 vaut 1, 7 % 8 vaut 7 ...
Opérateurs d'incrémentatation	
++	Incréméte de 1 la valeur de la variable
--	Décréméte de 1 la valeur de la variable

Opérateurs (2)

Opérateurs d'affectation	
+=	Ajoute à la variable de gauche la variable de droite. Stocke la nouvelle valeur dans la variable de gauche
-=	Soustrait à la variable de gauche la variable de droite. Stocke la nouvelle valeur dans la variable de gauche
*=	Multiplie la variable de gauche par la variable de droite. Stocke la nouvelle valeur dans la variable de gauche
/=	Divise la variable de gauche par la variable de droite. Stocke la nouvelle valeur dans la variable de gauche
%=	Calcule le modulo entre deux valeurs numériques. Stocke la nouvelle valeur dans la variable de gauche

Opérateurs (3)

Opérateurs de comparaison	
==	opérateur d'égalité : retourne 1 si les éléments comparés sont égaux, 0 sinon
!=	opérateur de différence : retourne 1 si les éléments comparés sont différents, 0 sinon
<	opérateur d'infériorité stricte : retourne True si la comparaison est correcte, False sinon
<=	opérateur d'infériorité : retourne True si la comparaison est correcte, False sinon
>	opérateur de supériorité stricte : retourne True si la comparaison est correcte, False sinon
>=	opérateur de supériorité : retourne True si la comparaison est correcte, False sinon
===	opérateur d'identité : compare la valeur et le type des variables, retourne 1 si les éléments comparés sont les mêmes, 0 sinon
!==	opérateur de différence : compare la valeur et le type des variables, retourne 1 si les éléments comparés sont différents, 0 sinon

Exemples d'expressions

- arithmétique

```
(3+4) * (56.7 / 89)
```

- chaîne de caractères

```
"parce" + " " + "que"
```

- logique

```
temp == 37
```

```
h2o = (temp<100) ? "eau" : "vapeur";
```

```
h2o = (temp>0) ? ((temp<100) ? "eau" : "vapeur") : "glace";
```

Exemples

- La date :

```
<html>
  <head>
    <title>Page dynamique</title>
  </head>
  <body>
    <script type = "text/javascript">
      //<![CDATA[
      date = new Date();
      document.writeln("Nous sommes le ", date);
      //]]>
    </script>
  </body>
</html>
```

Structures de contrôle

- Instructions de branchement

- if else :

- if (condition) {instructions;} [else {instructions;}]

- switch case :

- switch (variable) {

- case 'valeur 1':

- code à exécuter si "variable == 'valeur 1'";

- break;

- case 'valeur 2':

- code à exécuter si "variable == 'valeur 2'";

- break;

- case 'valeur 9':

- code à exécuter si "variable == 'valeur 1'";

- break;

- default:

- code à exécuter si tous les autres cas ont échoué;

- break;

- }

Structures de contrôle

- Boucles

- for :

- `for (i=1;i<n;i++) {instructions;}`
 - `for (p in objet) {instructions;}`

- while :

- `while (condition) {instructions;}`

- do while :

- `do {instructions;} while (condition)`

- Sortie d'une boucle : `break;`

- Itération suivante d'une boucle (pour "skip" une itération) : `continue;`

Fonctions

- Définition :

```
function nomFonction(param1, ..., paramN) {  
    // code JavaScript  
    return expression ;  
}
```

- Appel :

```
nomVariable = nomFonction(exp1, ..., expN);
```

- Les fonctions JavaScript ont une propriété spéciale appelée `arguments`, qui contient un tableau (`Array`) des paramètres pris en entrée. En utilisant la propriété `length` d'un `Array`, il est possible d'**itérer** sur chaque paramètre au travers du tableau.
- => Cela permet de développer des fonctions qui changent en fonction du nombre de paramètres.

Exemple de fonction

```
function somme() {  
    var argv = somme.arguments;  
    var argc = somme.arguments.length;  
    var result = 0;  
    for (var i = 0 ; i < argc ; i++) {  
        result += argv[i];  
    }  
    return result;  
}
```

Que retourne `somme(1,2,3)` ?
et `somme(2)` ?

Fonctions: passage des paramètres (1)

- Les variables sont **passées par valeurs**, les objets **par référence**. Les `Array` sont des objets
- **Cas d'une variable simple**
 - La fonction `test()` ajoute une chaîne à une chaîne donnée en paramètre.

```
function test(z)
{
    z = z + "suffixe";
}
```

```
var x = "démo";
test(x);
document.write(x);
```

=> Résultat : démo

- On peut vérifier que la variable `x` n'est pas modifiée hors du contexte de la fonction.
- La même chose se vérifierait avec un nombre.

Fonctions: passage des paramètres (2)

- **Cas d'un objet String**

```
var x = new String("démo");  
test(x);  
document.write(x);
```

=> Résultat : démo

- Les objets simples comme String, Boolean, Number sont aussi passés par valeur.

Fonctions: passage des paramètres (3)

- **Cas d'un objet Array**

- La fonction `testa` ajoute un élément à un tableau donné en paramètre.

```
function testa(a)
{
    a.push("ajouté");
}
```

```
x = new Array("un", "deux", "trois");
testa(x);
document.write(x);
```

=> Résultat : un , deux , trois , ajouté

- On voit qu'un objet `Array` est passé par référence, puisque l'élément ajouté dans le corps de la fonction reste inclus dans le tableau après l'appel de la fonction.

Fonctions: passage des paramètres (4)

- Cela vaut-il aussi pour un tableau déclaré comme un littéral?

```
var y = [1, 2, 3];  
testa(y);  
document.write(y);
```

Résultat : 1 , 2 , 3 , ajouté

=> oui, cela vaut aussi

Fonctions: passage des paramètres (5)

- En conclusion, si l'on veut passer une variable simple par référence, il faut **l'inclure dans un tableau** :

```
x = "essai";  
x = [x];  
testa(x);  
x = x.join(' ');  
document.write(x);
```

- **Résultat : essai ajouté**
- La fonction `join(String separ)` transforme le tableau en chaîne de caractères (les éléments étant séparés par `separ`) après l'appel de la fonction.
- Cela alourdit le code mais, reste le moyen le plus simple de passer une chaîne de caractères par référence.

Fonctions: passage des paramètres (6)

- **Autre possibilité (plus compliquée) pour passer une chaîne par référence**

```
function testb(a)
{
  var x = a[0];
  x += "suffixe";
  a[0] = x;
}
```

```
x = "essai";
x = [x];
testb(x);
x = x[0];
document.write(x);
```

Résultat : essaisuffixe

Les objets prédéfinis

- Objets du noyau

Nom de l'objet	Description
Array	L'objet Array permet de créer des tableaux. Il possède de nombreuses méthodes permettant d'ajouter, de supprimer ou d'extraire des éléments d'un tableau ainsi que de les trier.
Boolean	L'objet Boolean permet de créer des valeurs booléennes, c'est-à-dire des éléments possédant deux états : <i>vrai</i> et <i>faux</i> .
Date	L'objet Date permet de créer des dates et des durées. Il propose des méthodes permettant de les manipuler.
Function	L'objet Function permet de définir des fonctions personnalisées.
Math	L'objet Math permet de manipuler des fonctions mathématiques, comme par exemple les fonctions trigonométriques ou encore la fonction générant un nombre aléatoire, etc.
Number	L'objet Number est un objet permettant de faire des opérations de base sur les nombres.
RegExp	L'objet RegExp permet de créer des expressions régulières, c'est-à-dire des éléments permettant de faire des opérations avancées sur les chaînes de caractères.
String	L'objet String propose une grande variété de méthodes permettant de manipuler des chaînes de caractères.

Les objets prédéfinis (2)

- Objets du navigateur
 - `navigator` : qui contient des informations sur le navigateur de celui qui visite la page
 - `window` : c'est l'objet où s'affiche la page, il contient donc des propriétés concernant la fenêtre elle-même mais aussi tous les objets-enfants contenus dans celle-ci
 - `location` : contient des informations relatives à l'adresse de la page à l'écran
 - `history` : c'est l'historique, c'est-à-dire la liste de liens qui ont été visités précédemment
 - `document` : il contient les propriétés sur le contenu du document (couleur d'arrière plan, titre, ...)

Objet window

`window` est l'objet principal dans le modèle objet JavaScript

- Propriétés
 - closed** (Indicateur de fermeture de fenêtre)
 - defaultStatus** (Texte par défaut de la barre de statut)
 - name** (Nom de la fenêtre)
 - opener** (Fenêtre ouvrante)
 - screenLeft** (Position horizontale de la fenêtre navigateur.)
 - screenTop** (Position verticale de la fenêtre navigateur.)
 - status** (Texte de la barre de statut)
- Méthodes
 - alert()** (Affiche un message d'information)
 - blur()** (Enlève le focus à la fenêtre)
 - clearInterval()** (Arrête l'exécution d'un traitement à intervalle régulier)
 - clearTimeout()** (Suspend une minuterie)
 - close()** (Ferme la fenêtre)
 - confirm()** (Permet le choix entre Oui et Non)
 - focus()** (Donne le focus à la fenêtre)
 - home()** (Ouvre la page de démarrage du navigateur)
 - moveBy()** (Déplace la fenêtre)
 - moveTo()** (Positionne la fenêtre)

Objet window (2)

- Méthodes de `window` (suite)
 - `open()`** (Ouvre un popup)
 - `print()`** (Imprime la page en cours)
 - `prompt()`** (Ouvre une boîte de saisie)
 - `resizeBy()`** (Agrandit ou réduit la taille d'une fenêtre)
 - `resizeTo()`** (Redimensionne une fenêtre)
 - `scrollBy()`** (Déroule les ascenseurs)
 - `scrollTo()`** (Positionne les ascenseurs)
 - `setInterval()`** (Lance un traitement à intervalle régulier)
 - `setTimeout()`** (Déclenche une minuterie)
 - `stop()`** (Arrête le chargement de la page)
- Sous-objets
 - `document`** (Objet document visible sur la page)
 - `external`** (Objet propre à Internet Explorer)
 - `frames`** (Tableaux des frames)
 - `history`** (Objet permettant de naviguer dans l'historique)
 - `location`** (Informations sur l'adresse de la page)
 - `parent`** (Fenêtre mère du jeu de frames)
 - `screen`** (Information sur l'écran)
 - `self`** (Fenêtre en cours)
 - `top`** (Fenêtre initiale du jeu de frames)

Objet document

- L'objet `document` est probablement l'un des objets les plus importants du modèle objet JavaScript. Il permet d'accéder à tous les éléments affichés sur la page, de contrôler les saisies, de modifier l'apparence et le contenu.
- Propriétés (non exhaustif)
 - bgColor** (Couleur de fond de page)
 - cookie** (Chaîne contenant les cookies du domaine)
 - fgColor** (Couleur du texte)
 - fileSize** (Taille de la page en octets)
 - lastModified** (Date de dernière modification)
 - location** (Url de la page)
 - referrer** (Adresse de la page d'origine)
 - title** (Titre de la page)
- Méthodes (non exhaustif)
 - getElementById()** (Retourne un objet HTML à partir de son id)
 - getElementsByName()** (Retourne un tableau d'objets HTML à partir de son nom)
 - getElementsByTagName()** (Retourne un tableau d'objets à partir de leur balise)
 - write()** (Ecrit du texte dans le document)
- Sous-objets
 - forms** (Tableau des formulaires du document)
 - images** (Tableau des images de la page)

Lire / écrire

- `prompt ( )`
 - Ouvre une boîte de dialogue avec une zone saisie et 2 boutons (OK et Annuler),
 - retourne l'information lue
- `confirm ( )`
 - Ouvre une boîte de dialogue avec 2 boutons OK et Annuler,
 - retourne un booléen
- `alert ( )`
 - Permet d'écrire un message dans une fenêtre

Lire / écrire : exemple

```
<html>
  <head>
    <title>Utilisation de prompt() et d>alert()</title>
  </head>
  <body>
    <script type = "text/javascript">
      //
      annee = <b>prompt</b>('En quelle année sommes-nous ? ', '2018');
      alert('Vous avez répondu : ' + annee);
      //]]&gt;
    &lt;/script&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="855 899 883 922" data-label="Page-Footer"><p>36</p></div>
```

Écrire avec document.write()

```
<html>
  <head>
    <title>Utilisation de document.write</title>
  </head>
  <body>
    <script type = "text/javascript">
      //<![CDATA[
      document.write('Hello world !');
      //]]>
    </script>
  </body>
</html>
```

Ouvrir une nouvelle fenêtre

```
<script type = "text/javascript">
  //<![CDATA[
    fenetre = open('', '', 'height=50, width=300,
      status=yes');
    fenetre.document.write('<html>');
    fenetre.document.write('<head>');
    fenetre.document.write('<title>' + 'Titre fenetre'
      + '</title>');
    fenetre.document.write('</head>');
    fenetre.document.write('<body>');
    fenetre.document.write('Texte dans la fenetre');
    fenetre.document.write('</body>');
    fenetre.document.write('</html>');
  //]]>
</script>
```

Encadrer les items (avec jQuery)

- Encadrer les items d'une liste au passage de la souris :

- // code HTML

```
<ul id="liste">  
  <li>texte1</li>  
  <li>texte2</li>  
  <li>texte3</li>  
</ul>
```

- // code css

```
.li_hover { border:1px solid black; }
```

- // code JavaScript

```
$("#liste li").hover(  
    function() {  
        $(this).addClass('li_hover');  
    },  
    function() {  
        $(this).removeClass('li_hover');  
    }  
);
```

JavaScript et CSS

- En reprenant l'exemple de Vermeer (cf. cours 1, page 28), dans un fichier JS externe, on pourra ainsi modifier la largeur du <div> via la syntaxe suivante :

```
document.getElementById("peinture").style.width =  
    "100px";
```

(On accède à l'élément dont l'id est `peinture` et on change sa largeur (à 100 pixels))

- De la même façon, pour changer dynamiquement le style, il faut marier JavaScript et CSS en, par exemple : (voir le fichier à l'adresse : <http://isis.truck.free.fr/Site/ens/fichiers/TP2/chgmtStyleJS.html>)
 - créant 2 balises `link` successives dans le HTML contenant l'attribut `rel` (avec 2 valeurs distinctes : `alt` et `stylesheet`, par exemple)
 - affectant à la variable `lien` tous les nœuds nommés `link` : `var lien = document.getElementsByTagName("link") ;`
 - testant `lien[i].rel.indexOf("stylesheet") != -1`
 - affectant `lien[i].disabled` à `true` ou `false`, selon les cas

Formulaires et vérification (1)

- On utilise la balise HTML `form` et on fait des tests sur les variables récupérées via JS
- Exemple 1 : Changer dynamiquement le contenu d'un champ texte :

```
<form>
  Nom du personnage : <br /> <input type="text" name="le-nom" id="nomP" value="Mickey" />
  <input type="submit" onclick="switch_name();return false;" name="chg" value="Changer !" />
</form>
```

- La `form` en elle-même n'est pas visible.
- La largeur par défaut d'un champ `text` est 20 caractères.
- L'appel à l'événement JS se fait grâce à `onclick="switch_name();" ; "`
- Le script JS contient :

```
function switch_name() {
    nom=document.getElementById("nomP");
    if (nom.value=="Mickey")
        nom.value="Donald";
    else
        nom.value="Mickey";
}
```

Formulaires et vérification (2)

- On utilise la balise HTML `form` et on fait des tests sur les variables récupérées en mode GET ou POST **via du PHP** (voir cours suivants)

- Exemple 2 :

```
<form>
    Mot de passe: <input type="password" name="pwd">
</form>
```

- Exemple 3 :

```
<form name="input" action="form_action.php" method="get">
    /* voir le cours sur PHP */
    Nom de login: <input type="text" name="user">
    <input type="submit" value="Valider">
</form>
```

- Il y a également des boutons radio, etc. Voir :

http://www.w3schools.com/html/html_forms.asp

Formulaires et vérification (3)

- *name* et *id* dans une *form*
 - l'attribut *name* dans une *form* est intéressant mais n'est pas valide en XHTML. De ce fait, **on préférera utiliser un *id*** (qui, lui, est valide en XHTML) **pour identifier la *form***
 - Par ailleurs, l'attribut *name* sur les **éléments** d'une *form* sont valides et nécessaires pour la soumission au code côté serveur. Et si l'on associe un élément *label* (grâce à un attribut *for*) à un élément de la *form*, cet élément associé devra utiliser un *id*:

```
<label for="nomF">Nom de famille : </label>
<input type="text" name="nomF" id="nomF" />
```
 - Ainsi, **les éléments d'une *form*** vont typiquement **inclure un attribut *name*** et également **un attribut *id*** quand c'est nécessaire.

Conseils (1)

- document.write
 - Le comportement de cette fonction est problématique. Lors du chargement d'une page HTML cette fonction **ajoute** la chaîne passée en paramètre au contenu. Une fois la page chargée, cette fonction **remplace totalement** le HTML de la page par la chaîne en paramètre.
 - On a vu plus haut que le chargement d'une page HTML est séquentiel, c'est un exercice périlleux pour le navigateur que de rajouter du contenu à une page en train de charger. Si c'est périlleux, les bugs ne sont pas loin. Pour éviter les comportements « étranges » il ne faut pas utiliser **document.write()**
 - L'alternative est d'utiliser les standards et plus précisément le DOM. Si on veut rajouter du contenu à une page HTML le plus simple est de placer un élément vide à l'endroit voulu et de le remplir une fois la page chargée. Si l'on applique les recommandations sur la place de <script> , cela donne :

Pour remplacer document.write

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Une page web</title>
</head>
<body>
  <h1>Titre de la page</h1>
  <p>Aujourd'hui, j'ai</p>
  <p id="dynamique"></p>
  <p>et toutes mes dents</p>
  <script type="text/javascript">
    document.getElementById("dynamique").innerHTML = "8 ans";
  </script>
</body>
</html>
```

Conseils (2)

- Les scripts javascript doivent être placés dans un **fichier externe** à la page HTML.
- Habituellement on place la balise d'insertion des fichiers dans le `<head>` .
- Mais, pour une question de performances on peut préférer la placer à la fin de la page, juste avant `</body>`
- À noter que les deux méthodes sont valides du point de vue de la spécification HTML et XHTML.