Introduction à JavaScript

INTRODUCTION À JAVASCRIPT

1. Le langage JavaScript

- Langage de programmation qui permet d'accéder aux éléments d'un document XHTML.
- Noté directement dans le document XHTML ou dans un fichier séparé.
- Ne remplace pas le XHTML, complète le XHTML.
  - Si on pense que le XHTML concerne le contenu d'un document et le CSS concerne l'apparence d'un document on peut dire que le Javascript concerne le comportement d'un document.
  - Mais le document doit rester fonctionnel sans le Javascript (7 à 10 % des usagers n'acceptent pas de Javascript dans les documents qu'ils accèdent.)
- Le navigateur interprète les instructions Javascript et les exécute:
  - Quand le document est chargé
  - Quand un événement particulier se produit
- C'est de la programmation "coté client" parce que c'est l'application client qui exécute le programme. Par opposition à de la programmation "coté serveur" où c'est le serveur qui exécute le programme.

- Développé par Netscape, version la plus répandue (autres versions: Jscript de Microsoft)
- Il existe d'autres langages de script (VBScript) mais JavaScript est le plus utilisé et le plus supporté.
- Standards: Standard ECMA-262 ECMAScript Language Specification (voir http://www.ecma-international.org/publications/standards/Ecma-262.htm)
- Différent de Java: interprété plutôt que compilé, moins complexe que Java mais les 2 langages sont orientés objet (bien que JavaScript est plus un langage qui utilise l'objet qu'un langage purement orienté objet).
  - Un **objet** est une entité possédant des **propriété**s (caractéristiques pouvant être lues ou modifiées) et des **méthodes** (opérateurs pouvant agir sur l'objet)
- Relativement sécuritaire: ne peut pas lire ni écrire sur le disque dur, ne peut pas propager de virus.
- Utilisé principalement pour:
  - Valider (coté client) les données entrées dans un formulaire
  - Introduire du "dynamisme" dans un document (DHTML) (forme d'animation)
  - Fournir des fonctionnalités coté-client.

1.1. Notion de programmation

Un langage de programmation permet à l'humain de communiquer dans un mode compréhensible pour un ordinateur.

L'humain utilise ce langage pour décrire le traitement que l'ordinateur doit faire sur des données. La description du traitement correspond à un algorithme et les données correspondent à de variables.

Le traitement des données s'exprime par une série d'instructions parmi lesquelles on a les instructions conditionnelles (quand un traitement se fait seulement si certaines conditions sont satisfaites) et les instructions itératives (quand un traitement est répété).

2. Fonctionnement

- Le navigateur agit comme une machine Javascript: il lit et exécute le code JavaScript.
- Si il y a des erreurs dans le code JavaScript, le navigateur le détecte et nous en informe.
- Il est possible de faire en sorte que le navigateur n'interprète pas le JavaScript:
  - Firefox: boîte de dialogue Options…onglet Contenu: case à cocher: activer le JavaScript.

2.1. Traitement des événements

- Le navigateur a des réactions prédéfinies à différents événements.
  - Il charge un document dans la fenêtre quand on clique sur la source d'un hyperlien.
  - Il envoie le formulaire au serveur quand on clique sur le bouton Submit.
- Avec JavaScript il est possible de modifier ces comportements par défaut.
  - À chaque événement correspond un gestionnaire d'événement (EventHandler) :
  - On associe du code JavaScript à un gestionnaire d'événement en particulier.
  - Le code JavaScript sera alors exécuté quand l'événement se produit.

- Le nom d'un gestionnaire d'événement commence par `"on"` et est suivi du nom de l'événement concerné: `onload, onclick,...`
- Une façon d'associer du code JavaScript à un gestionnaire d'événement est d'utiliser un attribut du nom du gestionnaire d'événement dans une balise HTML
  - La valeur de l'attribut est le code javaScript
  - Par exemple `onload="...code javaScript..."`

3. Exemple: ouverture d'un hyperlien dans une fenêtre distincte

Voici le code JavaScript qui permet d'ouvrir la destination d'un hyperlien dans une nouvelle fenêtre de navigateur:

```
<p>L'hyperlien suivant s'ouvre dans une nouvelle fenêtre <a
href="http://www.uqtr.ca/~helene/INF1001" onclick="window.open(this.href); return
false;" >Site du cours</a> </p>
```

Voir http://linux04.uqtr.ca/~helene/Javascript/exOuvrirFenetre.xhtml

Le code JavaScript ici est: `onclick="window.open(this.href); return false;"`

- `onclick` est un attribut de l'élément `a`.
- La valeur de l'attribut `onclick` est le code qui sera exécuté quand l'événement correspondant à l'attribut se produit. Ici l'événement est le fait de cliquer sur l'élément.
- `window.open(this.href); return false;` est du code JavaScript. Ce code est formé de 2 **instructions**. Chaque instruction est terminée par un `;`.

- L'objet **`window`**
  - Objet prédéfini en JavaScript.
  - Désigne la fenêtre dans lequel est le document est affiché
  - L'objet le plus élevé dans la hiérarchie des objets de JavaScript
  - Quelques propriétés de l'objet `window`:
    - `name`: le nom de la fenêtre
    - `status` : le contenu de la barre d'état
  - Quelques méthodes de l'objet `window`:
    - `alert()`:ouvre une boîte de dialogue pour informer
    - `confirm()`:ouvre une boîte de dialogue pour demander une confirmation
    - `open()`: ouvre une nouvelle fenêtre
  - L'objet `window` contient des sous-objets dont:
    - L'objet `document`: le document affiché dans la fenêtre
  - Voir http://fr.selfhtml.org/javascript/objets/window.htm
  - Pour avoir accès à une méthode d'un objet on utilise la syntaxe `<nom de l'objet>.<nom de la méthode>(argument)`. On dit qu'on fait **appel** à une méthode.
    - Quand une méthode a besoin d'informations pour faire son action, cette information est fournie entre parenthèses après le nom de la méthode. Ce sont les **arguments** de la méthode.

    - Par exemple l'instruction `window.open(this.href)` va faire un appel à la méthode `open` de l'objet `window` en passant comme argument la valeur `this.href`.

- L'objet **`document`**
  - Objet prédéfini en JavaScript.
  - Représente le document HTML
  - Voir http://fr.selfhtml.org/javascript/objets/document.htm
  - Quelques propriétés de l'objet `document`
    - `lastModified:` date de la dernière modification du document
    - `title:` contenu de l'élément `title` du document
    - `url:` url du document
  - Quelques sous-objets de l'objet `document`
    - L'objet `forms[]`: tableau contenant tous les objets `form` (pour formulaire) du document.
      - Ce tableau sera utilisé pour accéder aux données d'un formulaire dans le cas où on voudrait valider ces données.
    - L'objet `images[]`: tableau contenant tous les objets `image` du document
      - Ce tableau sera utilisé pour introduire des effets de "rollover" sur des images
  - Méthodes de l'objet `document`
    - `open()`: ouvrir un document
    - `close()`: fermer un document

- L'objet **`this`**
  - Désigne l'objet courant.
  - Ici `this` désigne l'objet correspondant à l'élément `a` pour lequel on définit l'attribut `onclick`.
  - `this.href` désigne la valeur de l'attribut `href` de l'objet courant. Comme l'objet courant est l'élément a, la propriété désigne la valeur de l'attribut `href` de l'élément `a`. Dans le code JavaScript `window.open(this.href)` l'argument `this.href` désigne donc l'url du document qui devra être affiché dans la fenêtre qui sera ouverte par la méthode `open`.
- L'instruction `return false;` :
  - Fait en sorte que le traitement par défaut (dans notre cas charger l'hyperlien destination dans la fenêtre courante) ne s'exécute pas.
- Instructions séparées par ; (optionnel si une seule instruction par ligne)

3.1. Langage de script par défaut

Plusieurs navigateurs considèrent que le langage de script par défaut est JavaScript. Il est plus prudent d'indiquer explicitement le langage de script:

```
<meta http-equiv="Content-Script-Type" content="application/javascript" />
```

4. Exemple: confirmation de l'envoi d'un formulaire

```
<form action="http://www.uqtr.ca/~helene/cgi-bin/traitePost.pl" method="post"
onsubmit="return confirm('Vous voulez vraiment envoyer le formulaire?');">
```

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

- La méthode `confirm()` appartient à l'objet `window`. Elle prend en argument une chaîne de caractères
  - La méthode `confirm()` ouvre une boîte de dialogue affichant
    - le texte qui lui est passé comme argument
    - un bouton Ok
    - un bouton Annuler
  - Si l'usager clique sur le bouton Ok , la méthode retourne `true`
  - Si l'usager clique sur le bouton Annuler , la méthode retourne `false`
- Remarquez l'utilisation de l'apostrophe pour délimiter la chaîne de caractère (le guillemet étant déjà utilisé pour délimiter la valeur de l'attribut `onSubmit`).
- Remarquez l'instruction `return`. Elle permet de propager à l'attribut `onSubmit` la valeur `false` ou `true` retournée par la boîte de dialogue affichée par la méthode `confirm()`.
  - Ainsi si l'usager a cliqué sur le bouton Ok le formulaire sera envoyé (parce que `onSubmit` aura la valeur `true`) et si l'usager a cliqué sur le bouton Annuler le formulaire ne sera pas envoyé.
- Le mot `return` est un **mot réservé** pour JavaScript.. Un mot réservé est un mot qui a un sens particulier pour JavaScript. Vous trouverez la liste des mots réservés de JavaScript à: http://fr.selfhtml.org/javascript/langage/reserve.htm

- Voir http://linux04.uqtr.ca/~helene/Javascript/exConfirmFormulaire.xhtml

5. Exemple: validation d'une entrée d'un formulaire

L'événement `blur` se produit lorsque l'élément perd le focus. Par exemple quand l'usager clique sur un autre élément ou passe à un autre élément avec la touche tab. En programmant le gestionnaire pour cet événement on peut valider une entrée dès que l'usager quitte le champ.

```
<p><label for="nom">Nom:</label>
<input onblur="ValideNom(this.value)"  name="nom" id="nom" type="text" size="50" />
</p>
```

- L'objet `this` représente l'élément input
- La propriété `value` est la chaîne de caractères entrée dans ce champ.
  - La fonction `ValideNom()` va simplement vérifier que cette valeur n'est pas la chaîne vide.
- C'est une fonction définie par l'usager.
- La définition d'une fonction comprend 4 éléments:
  - Le mot réservé function
  - Le nom de la fonction
  - Les arguments, entre parenthèses
  - Le code , entre accolades.
- La fonction `ValideNom()` est définie dans l'entête du document


```
<script  type="text/javascript">
function ValideNom(valeur){
        if (valeur=="") {
                alert("Vous devez entrer un nom.");
                document.forms[0].elements[0].focus();
        }
}
</script>
```

Voir http://linux04.uqtr.ca/~helene/Javascript/exValidationChamp.xhtml

- La méthode `alert()` appartient à l'objet `window`.
- Elle prend en argument une chaîne de caractères.
- La différence entre la méthode `alert()` et la méthode `confirm()` est que la boîte de dialogue affichée avec la méthode `alert()` ne contient que la chaîne fournie en argument et un bouton OK tandis que la boîte de dialogue affichée par la méthode `confirm()` contient en plus un bouton pour annuler.
- L'instruction conditionnelle : `if (valeur=="") {...}`
  - Le traitement entre `{}` est exécuté seulement si la condition entre parenthèses `(valeur=="")` est vérifiée
- L'objet `forms` est un tableau indexé à partir de 0. Il permet d'accéder à tous les formulaires d'un document. Dans notre cas, il n'y a qu'un formulaire d'où `document.forms[0]`
- Une fois qu'on a accédé à un formulaire on peut accéder à ces éléments au moyen du sous-objet `elements[]`:
  - Le tableau `elements[]`est indexé à partie de 0. Puisque l'élément à valider est le premier élément du formulaire on y accède par `elements[0]`
- La méthode `focus()` d'un élément donne le focus à l'élément concerné
- On peut également accéder à un formulaire au moyen de l'identificateur du formulaire. Si on avait donné la valeur `unForm` à l'attribut `id` de l'élément `form` on aurait pu écrire `document.forms['unForm']`
  - Ça fonctionne également pour les éléments du formulaire: `document.forms['unForm'].elements['nom']` désigne l'élément avec `id="nom"` du formulaire avec `id="unForm"`
  - Voir http://linux04.uqtr.ca/~helene/Javascript/exValidationChampV2.xhtml

6. Exemple: validation d'un formulaire

6.1. Définition du formulaire:

```
<form id="crLecture" method="post"
    action=http://www.uqtr.ca/CGI/uqlib/WebMail
    onsubmit="return Valide()">

<p>
<input type="hidden" name="_TO" value="helene.desaulniers@uqtr.ca"/>
<input type="hidden" name="_SUBJECT" value="testWebMail"/>
```

```
<input type="hidden" name="_URL_FIN"
    value="http://linux04.uqtr.ca/~helene/Formulaire/finFormulaire.html"/>

<label for="nom">Votre nom: </label>
<input type="text" name="nom" id="nom" size="50" maxlength="50"/>
</p>

<p>En <strong>une phrase</strong>, énoncez le point qui vous a particulièrement
intéressé dans votre lecture:</p>
<p><input type="text" name="desc1" id="desc1" size="50" maxlength="50" value=""/></p>

<p>Décrivez ce point et pourquoi ça vous a intéressé (maximum 750 caractères --
environ 10 lignes):</p>

<p><textarea name="expl1" id="expl1" rows="5"  cols="70"></textarea></p>
<p class="droite">
<input type="button" value="Nombre de caractères"
onclick="afficheNbCar(this.form.elements['expl1'], this.form.elements['nbCar1'])"/>

<input readonly="readonly" type="text" name="nbCar1" id="nbCar1" size="3"
maxlength="3" value="0"/></p>
<p><input type ="submit" value="Envoyer"/> </p>

</form>
```

- Voir http://linux04.uqtr.ca/~helene/Formulaire/CRLecture.xhtml
- Le gestionnaire d'événement `onsubmit` est associé au formulaire
  - La valeur de `onsubmit` est la fonction qui sera exécutée au moment où le bouton de soumission est cliqué. Ici la fonction `Valide()`.
- Le gestionnaire d'événements `onsubmit` n'exécute pas son traitement par défaut (transmettre le formulaire) dans le cas où il reçoit la valeur `"false"`.
- Le formulaire a l'identificateur `crLecture`. Ce nom sera utilisé pour accéder au formulaire dans le code JavaScript.
- Les trois premiers contrôles du formulaire sont des contrôles cachés qui sont en fait des informations nécessaires à WebMail
- Les contrôles du formulaire qu'on veut manipuler avec le code JavaScript ont un identificateur:`nom, desc1, expl1` et `nbCar1`.
  - Le texte que l'usager a entré dans la zone de texte pour son nom est donc accessible par `document.forms['crLecture'].elements['nom'].value`

6.2. Le code JavaScript

- Avant d'envoyer le formulaire on veut vérifier que:
  - Le nom est entré
  - La description ne dépasse pas 750 caractères.

```
function Valide(){
  if(document.forms[0].elements['nom'].value==""){
    alert("Vous n'avez pas donné votre nom");
    document.forms[0].elements['nom'].focus();
    return false;
  }

  var long1 = document.forms[0].elements['expl1'].value.length;
  if (long1 >750){
    alert("Votre description est trop longue. Elle contient "+ long1 + " caractères.");
```

```
    document.forms[0].elements['expl1'].focus();
    return false
  }
}
```

- Dans les exemples précédents, les données qui ont été manipulées par le code JavaScript étaient des objets et des propriétés déjà existantes. Si on a besoin de données supplémentaires, on doit les déclarer pour réserver l'espace mémoire nécessaire. En terme de programmation on parle d'une **variable** et on utilise le mot réservé `var`:

```
var long1;
```

- La ligne de code qui commence par mot le mot réservé `var` est une **déclaration**. Ici on déclare la variable `long1`.
- Le nom d'une variable est appelé un **identificateur.** Ce nom est choisi par le programmeur et doit respecter certaines règles:
  - formé d'une suite de caractères parmi les lettres, les chiffres, le caractère de soulignement `(_)`, le signe dollar `($)`
  - ne commence pas par un chiffre
  - n'est pas un mot réservé
- Il est conseillé de choisir un identificateur qui reflète le sens de la variable déclarée.
- On peut déclarer plus d'une variable dans une même déclaration. Il suffit de séparer les identificateurs par une virgule `(,)`
- On peut déclarer et initialiser une variable dans la même instruction:

```
  var long1 = document.forms[0].elements['expl1'].value.length;
```

- Pour avoir accès aux contrôles du formulaire avec JavaScript, on part de l'objet `document`.
  - Chaque formulaire d'un document est accessible par son index: `document.forms[0]` ou par son identificateur: `document.forms['crLecture'].`
  - Chaque contrôle du formulaire est accessible par son nom: `document.forms[0].elements['nom']`
  - La valeur d'un contrôle est accessible par l'attribut `value` du contrôle:
    - `if (document.forms[0].elements['nom'].value== "")`
- La fonction `Valide()` prend fin dès qu'elle rencontre une instruction `return (return false).` Si la fonction se termine par la fin du code elle retourne vrai par défaut.

Pour faire le calcul du nombre de caractères dans le champ des explications, on a simplement un bouton:

```
<input type="button" value="Nombre de caractères"
onclick="afficheNbCar(this.form.elements['expl1'], this.form.elements['nbCar1'])"/>
```

La fonction `afficheNbCar()` associée à ce bouton reçoit 2 arguments: `this.form.elements['expl1'] et this.form.elements['nbCar1'].` Ce sont les 2 contrôles qui seront utilises par la function `afficheNbCar().`

- L'objet `this` désigne ici le bouton qu'on est à définir.
- La propriété `form` du bouton désigne le formulaire contenant le bouton. Cette propriété nous permet en quelque sorte de remonter dans la hiérarchie des objets du formulaire.
- À partir du formulaire on accède à tous les contrôles avec l'objet `elements`

Voici le code de la fonction `afficheNbCar()`:

```
function afficheNbCar(leChampTexte,leChampNombre){
      leChampNombre.value=leChampTexte.value.length;
}
```

Voir aussi: http://openweb.eu.org/articles/validation_formulaire/

7. Élément noscript

Qu'est-ce qui arrive à une page contenant du JavaScript dans le cas où le Javascript est désactivé? Habituellement rien, simplement que le code JavaScript n'est pas exécuté.

Cependant il est possible de détecter le fait que le lecteur a désactivé le JavaScript. On pourrait alors souhaiter informer le lecteur que la page contient des fonctionnalités additionnelles accessibles en activant le JavaScript.

Ça se fait au moyen de l'élément noscript.

```
<body>
<noscript>
<p>Cette page contient des fonctionnalités accessibles en activant
JavaScript</p>
</noscript>

<h1>Formulaire de compte-rendu de lecture pour le cours INF1001</h1>
```

JavaScript et XHTML

En XHTML le code JavaScript est analysé par l'agent utilisateur. Ce qui fait que un caractère comme < devrait plutôt être codé comme <.

7.1. Exemple: valider l'âge

La fonction suivante permet de valider un formulaire en s'assurant que l'usager est majeur

```
function Valide() {
            if(document.forms['formAge'].elements['age'].value < 18) {
                  alert("Vous n'êtes pas majeur");
                  return false;
            }
      }
```

- Voir http://linux04.uqtr.ca/~helene/Javascript/exValideAge.xhtml

Le navigateur détecte une erreur de syntaxe d XML.

Solution 1: Utiliser les entités de caractères de xml

```
<script type="text/javascript">
   function Valide() {
         if(document.forms['formAge'].elements['age'].value &lt; 18) {
               alert("Vous n'êtes pas majeur");
               return false;
         }
   }
</script>
```

Mais en utilisant les entités de caractères on introduit un problème de lisibilité parce qu'on fait intervenir des notions de xhtml à l'intérieur du JavaScript..

- Voir http://linux04.uqtr.ca/~helene/Javascript/exValideAgeSoln1.xhtml

Solution 2: Utiliser les sections CDATA de xml

Une section CDATA permet d'introduire du texte qui ne sera pas traité par l'analyseur xml. La syntaxe est:

```
<![CDATA[
    Texte qui ne sera pas analysé.
  ]]>
```

```
<script type="text/javascript">
<![CDATA[
   function Valide() {
         if(document.forms['formAge'].elements['age'].value < 18) {
               alert("Vous n'êtes pas majeur");
               return false;
         }
   }
]]>
</script>
```

- Voir http://linux04.uqtr.ca/~helene/Javascript/exValideAgeSoln2.xhtml

Solution 3: utiliser un fichier externe pour le code JavaScript

- Fichier texte
- Extension js
- Déclaré dans l'entête du document HTML avec l'attribut `src`:

```
<script type="application/javascript" src="valide.js"></script>
```

et le fichier JavaScript `valide.js`:

```
function Valide() {
```

```
	if(document.forms['formAge'].elements['age'].value < 18) {
		alert("Vous n'êtes pas majeur");
		return false;
	}
}
```

- Voir http://linux04.uqtr.ca/~helene/Javascript/exValideAgeSoln3.xhtml

Avantage:

- Facilite la gestion du code
- Rapidité de chargement quand plusieurs documents html partagent du code. Le navigateur va charger le document javaScript dans la cache et utiliser la cache chaque fois qu'il y a une requête pour ce document.

8. Module scripting

Elements	Attributes	Minimal Content Model
noscript	Common [p.32]	(Heading | List | Block)+
script	charset (Charset [p.25]), defer ("defer"), id (ID [p.24]), src (URI [p.28]), type* (ContentType [p.25])	PCDATA

De plus l'élément `script` est ajouté dans le modèle de contenu de l'élément `head`

9. Traitement des erreurs

- Avec Netscape, Mozilla et Firefox:
  - Entrer `javascript:` dans la barre d'adresse pour avoir l'affichage des erreurs
  - Tools>Web Development> JavaScript Console

10. Syntaxe

10.1. Variables

- Désigne un espace mémoire utilisé pour mémoriser une donnée utilisée par JavaScript,
- Accédée par son nom
  - Ne commence pas par un chiffre
  - contient des chiffres, des lettres, le caractère de soulignement `(_)` ou le signe dollar `($)`
  - maximum 32 caractères,
  - différent d'un mot réservé (voir http://fr.selfhtml.org/javascript/langage/reserve.htm)
- Locale
  - variable déclarée dans une fonction par le mot réservé `var` et valide uniquement dans cette fonction
- Globales: valide dans tout le code JavaScript.
- Type:
  - Numériques: entier ou nombre à point flottant
  - Chaînes de caractères: délimité par " ou ‘
  - Le type de la variable est déterminé par son contenu. (Il existe aussi des variables de type booléen: valeur `true` ou `false`)

- Null utilisé pour déterminer si une variable a reçu une valeur ou non

10.2. Opérateurs

- Opérateurs arithmétiques: `+`, `-`, `*`, `/`, `%` (modulo)
  - Notez le double usage du symbole "+": addition et concaténation
  - Si les 2 opérandes sont numériques, le + correspond à l'addition.
  - Dès qu'une opérande est une chaîne de caractères, + correspond à l'opérateur de concaténation +
  - Voir : http://linux04.uqtr.ca/~helene/Javascript/doubleUsage.xhtml
- Opérateurs de comparaison: `<`,`<=`, `>`, `>=`,`= =`,`!=`
- Opérateurs logiques
  - `&&` (et)
  - `||` (ou)
  - `!` (non)

10.3. Fonction

- Bloc d'instructions identifié par un nom et exécutant une tâche spécifique
- Définition de la fonction
  - Mot réservé `function`
  - Nom de la fonction
  - Paramètres entre parenthèses
  - Corps de la fonction entre accolades
    - Variables locales
    - Instructions
    - Mot réservé `return` pour une fonction qui retourne un résultat
- Utilisation de la fonction
  - Appel de la fonction avec
    - son nom
    - les arguments entre parenthèses

10.4. Entrée de données

- Par un formulaire
- Par une boîte de dialogue : `Prompt`
- Boîte de dialogue `Confirm` pour demander une confirmation

10.5. Sortie

- Par un formulaire
- En écrivant dans un document HTML
- Par une boîte de dialogue Alert
- Dans un fichier texte nommé cookie

10.6. Instruction conditionnelle

- `if` (expression booléenne){instructions;…; }
- `if` (expression booléenne) { instructions;…; } `else` { instructions;…; }

10.7. Boucle

- For
- Voir http://fr.selfhtml.org/javascript/langage/affichage/police.htm
- Boucle While
  - Syntaxe:

```
while(expression booléenne) {
      instruction;
      .
      .
      instruction
}
```

- http://fr.selfhtml.org/javascript/langage/boucles.htm