Introduction à Javascript

Code: js-intro

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/js-intro/js-intro.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/js-intro.pdf**

Auteurs et version

- Daniel K. Schneider - Patrick Jermann - Vivian Synteta - Olivier Clavel

- Version: 1.6 (modifié le 4/11/10 par DKS)

Prérequis:

- Connaître (savoir lire) le langage HTML

  Module technique précédent: html-intro

  Module technique précédent: html-forms

Suites possibles:

Module technique suppl.: js-dom (Javascript avec DOM)

Module technique suppl.: php-intro, ... php-dom (Dom avec PHP)

Objectifs:

1. Connaître les origines et les principes de JavaScript
2. Appliquer quelques exemples simples d'utilisation:
   - Obtenir des informations sur le browser
   - Vérification de formulaires
   - Quiz en ligne
3. Attention: trop peu de DOM !
   - Ce document suit surtout le modèle de JavaScript 1.2/1.3 (NS 4x / IE4) et qui marche toujours dans les navigateurs modernes
   - cf. js-dom (Javascript avec DOM). Un jour il faudrait intégrer ces deux modules ...

1. Table de matières détaillée

2. JavaScript (client-side), introduction

2.1 Origine

- JavaScript a été développé par Netscape (d'abord sous le nom LiveScript)
- Il existe une version Microsoft (appelée parfois JScript)
- Le nom "JavaScript" reflète un certain voisinage syntactique avec JAVA (et il a été choisi pour des raisons de marketing)

2.2 Utilisation principale de JavaScript

Formulaires interactives

- Applications interactives avec des formulaires (par ex. tests et quiz)
- Vérification de formulaires traitées "server-side"

Pages interactives (DHTML)

- Pages HTML plus riches (par ex. "highlighting", menus, etc.)
- Appplications HTML, SVG, VRML, etc. (avec Ecmascript)
- Animations

Gestion de contenus

- Génération de pages HTML selon le profil de l'utilisateur
- Gestion de plugins, versions de Java etc.

2.3 Versions

- JS 1.0: Netscape 2 et plus
- JS 1.1: Netscape 3 et plus
- JS 1.2: Netscape 4 et plus, IE 5 (avec qq déficiences)
- JS 1.3: Netscape 4.06 et 4.5 et plus - à éviter
- JS 1.5: Netscape 6 / Mozilla (assez compatible ECMA Script + DOM)
- ECMAScript + DOM: Mozilla/Firefox/IE6. Le langage Javascript standardisé (sans objets) + les "bindings" des spécifications W3C DOM (éléments d'une page).
- JS 1.6: Firefox 1.5
- Note: Il faut éviter d'utiliser des extensions propriétaires (portabilité)

2.4 Ressources et outils de développement

- On conseille l'utilisation d'un éditeur de programmation
- Pour tester des fonctions JavaScript simples: console Javascript de Firefox
  taper l'URL: `JavaScript:`
  ou encore avec le menu: Tools->JavaScript Console
  - ensuite dans la petite fenètre "input" du haut vous pouvez entrer du code

```
alert("salut")
```

- pour debugger, ouvrir toujours la console Javascript !!!!
  - dans Mozilla/Firefox, c.f. ci-dessus
  - dans IE: Outils->Options Internet->Avancé (cocher "Afficher notification)
  - dans Firefox: il existe plusieurs AddOns comme "DomInspector", "WebDeveloppper", "InspectThis"

Pages ressources:

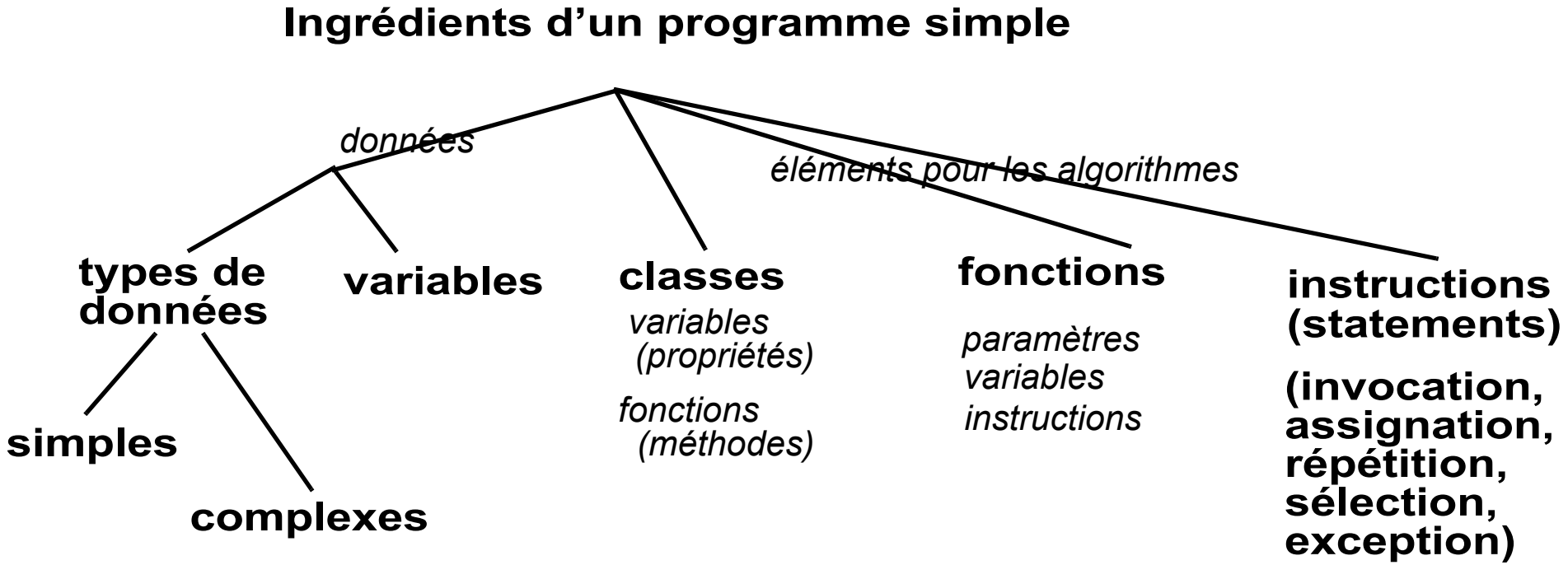
- **http://edutechwiki.unige.ch/en/JavaScript_links**

3. Caractéristiques du langage JavaScript

3.1 Caractéristiques clefs:

- Syntaxe: ressemble à Java, C, Php etc.
- Intégration avec HTML
- Langage basé objets (pas orienté objets)
- Des objets qui représenent le navigateur, la page web ou son contenu font partie d'une implémentation JavaScript et permettent la manipulation de contenus "Web" (notamment des formulaires et liens/images). Il existe 3 versions principales de ces objets
  - IE 3,4 + NS 3
  - NS 4.x
  - NS 6/7 + IE 5.5/6 + Mozilla
    (avec sous-variantes, +/- compatibles Document Object Model (DOM)
  - Firefox / IE 7
    (comme ci-dessus, mais plus DOM compatible)

3.2 La notion de programme



La notion de programme (suite)

Un ***programme*** est composé d'une série ***d'instructions***, parfois regroupées en ***fonctions***, qui compose un ***algorithme*** permettant de traiter des ***données***.

Les ***données*** sont manipulées par le biais de ***variables.***

Les ***fonctions*** peuvent recevoir des ***paramètres*** (données), effectuent des calculs ou des actions en utilisant (parfois) des ***variables*** internes. Les fonctions peuvent retourner une valeur à la fin de leur exécution. Un fonction est déclenché par une invocation, c-a-d. une instruction ailleurs dans le programme y compris la gestion d'un événement qui survient (l'utilisateur clique sur un bouton par exemple)

L'algorithme contient une ou plusieurs ***structures de contrôle*** qui permettent au programmeur de décider les calculs et les actions à entreprendre. Ces structures permettent de sélectionner une action en fonction de la valeur d'une variable, de répéter une action de multiples fois...

3.3 Les variables

Une variable est une zone de mémoire informatique contenant une valeur pouvant varier au cours de l'exécution d'un programme.

- Il s'agit d'un "conteneur" auquel on donne un nom (e.g. **`score`**).
- La valeur contenue peut être de différents types

```
entier (integer) : 134
réel (float) : 345.38743
chaîne de caractère (string) : "réponse 3 : Lausanne"
```

- Pour donner une valeur à la variable, on fait une ***assignation***.

```
score = 34;
```

On lit de droite à gauche "mettre la valeur 34 dans la variable score" (non pas "score égale 34", ***ce n'est pas un opérateur d'égalité***)

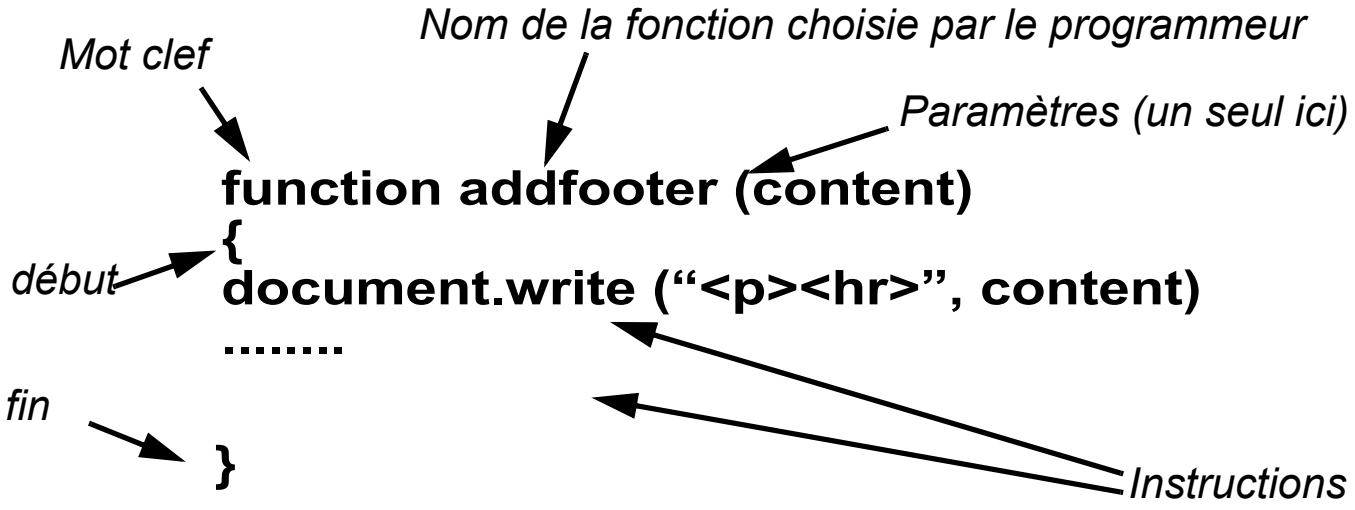
```
feedback = "Vous avez gagné !";
```

- On peut réutiliser la valeur de la variable à tout moment
  write(feedback); (écrit le contenu de la variable feedback)
- Avec Javascript, il n'est pas obligatoire (mais fortement conseillé) de déclarer et d'initialiser les variables. Cela permet de s'assurer que chaque variable est bien définie et contient une valeur par défaut avant d'effectuer des opérations avec. Vous gagnez du temps pour trouver les erreurs !!

```
var score = 0;
```

3.4 Les fonctions

Ensemble d'instructions identifié par un nom unique permettant d'effectuer une action, un calcul...



- Une fonction peut accepter des paramètres (données)
- Une fois qu'elle est définie, on peut l'appeler depuis n'importe quel endroit du programme, y compris depuis une autre fonction.

```
addfooter("Fait à TECFA");
```

- Si la fonction retourne une valeur on peut la stocker dans une variable par exemple

```
score = calcule(reponse1, reponse2);
```

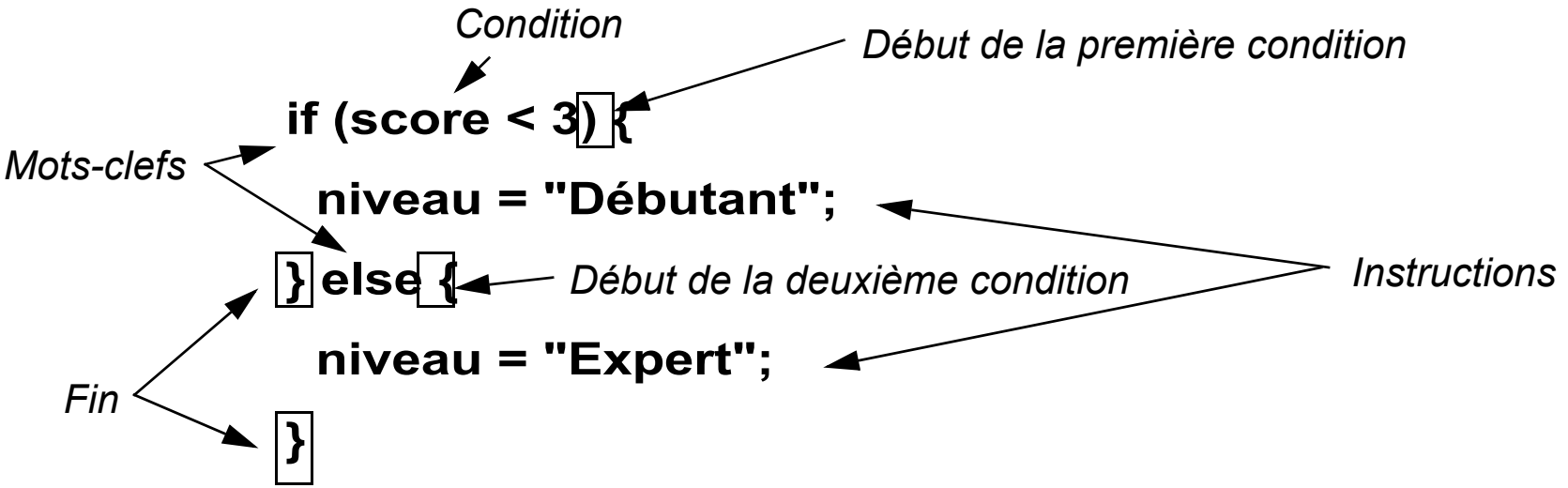
- Les variables et paramètres définis dans une fonction seront locaux à la fonction (on ne les "voit" pas dans le reste du programme).

3.5 Structures de contrôle

Les structures de contrôle permettent de contrôler l'exécution d'un programme en faisant des tests sur les valeurs d'une variable. Il existe plusieurs structures de contrôle. Seuls 2 sont présentées ici en quelque détail.

A. La sélection : if... else

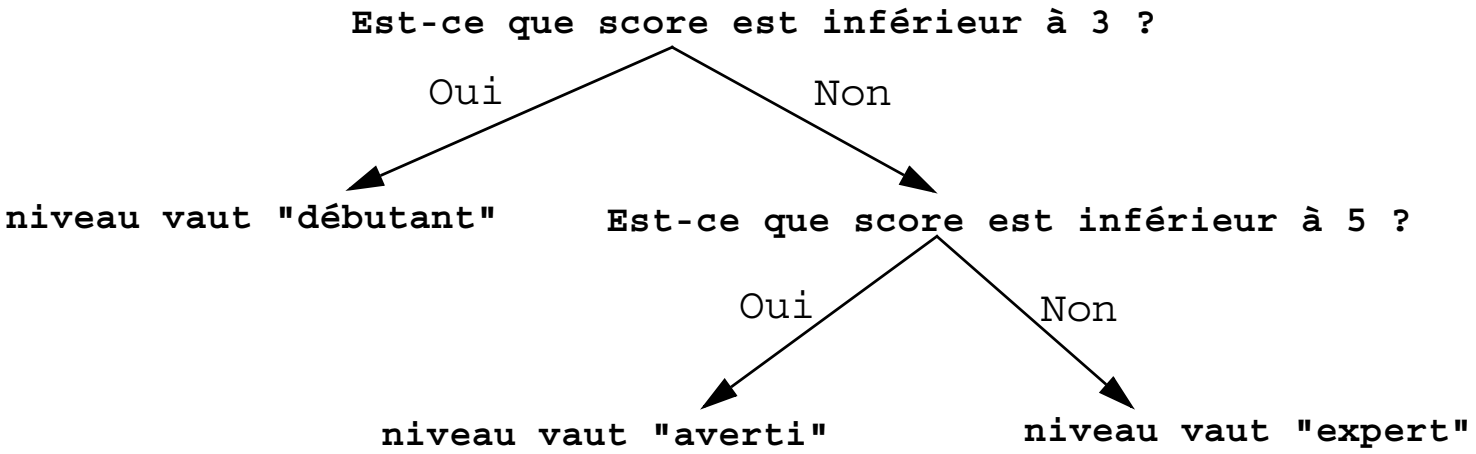
On peut résumer en quelques mots cette structure à "si une condition est vraie alors faire ceci sinon faire cela".



- On peut imbriquer plusieurs sélections : "si la condition est vraie alors faire ceci sinon, si cette autre condition est vraie alors faire ceci, sinon faire cela";

```
if (score < 3) {
    niveau = "débutant";
} else {
    if (score < 5) {
        niveau = "averti";
    } else {
        niveau = "expert";
    }
}
```

Voici l'arbre de décision associé à l'exemple ci-dessus.



B. La boucle incrémentale: for...

```
for (<expression de début>; <condition>; <expression de fin>) {
   expression 1;
   expression 2;
   ...
}
```

- L'***expression de début*** est exécutée quand on rentre dans la boucle pour la première fois.
- la ***condition*** est évaluée à chaque passage dans la boucle. Si elle est vraie, on exécute les expressions de la boucle. Si elle est fausse, on sort de la boucle.
- L'***expression de fin*** est exécutée quand toutes les expressions de la boucle on été exécutées (avant le passage suivant dans la boucle).

Exemple : calcul d'un produit factoriel (n! = 1 * 2 *... * n-2 * n-1 * n)

```
var n = 12;
var prodfact = 1;
for (i=1; i<=n; i++) {
    prodfact = prodfact * i;
}
document.write("le produit factoriel de" + n + " vaut " + prodfact); )
```

C. While

```
while (condition) {
     statements
   }
```

D. break (dans while, for, switch): sortir de la boucle

E. switch

```
switch (expression){
      case label :
         statement;
         break;
      case label :
         statement;
         break;
      ...
      default : statement;
   }
```

3.6 Les objets

(sujet à option pour les débutants ...)

- JavaScript est un langage “basé objets”:
  - le programmeur peut “utiliser” des objets “JavaScript” pré-définis
  - il peut définir des objets simples (sans héritage)
  - tout élément conceptuel utilisé, affiché etc. à l’intérieur d’une page est représenté par un objet qui s’insère dans une structure de type DOM
  - d’autres objets existent (par exemple informations sur le navigateur)
- un objet est une structure informatique qui:
  - possède des propriétés (variables contenant de l’information);
  - possède des méthodes (procédures permettant de manipuler les informations contenues dans l’objet);
  - (parfois) possède des “event handlers” associés.
- Les objets préexistants (DOM) sont organisés hiérarchiquement.
  - Il existe les des objets introduits par Netscape (repris par Microsoft) et qui représentent le navigateur et la fenêtre.
  - Le DOM (Document Object model) du W3C represente la structure informatique d’une page HTML selon une logique plus rigoureuse que les anciens modèles de NS/MS et qu’il ne faut plus utiliser.
  - Méthodes (fonctions) ou propriétés (variables attachées aux objets) vous permettent de traiter une page HTML, créer des boites de dialogues, fabriquer des nouvelles pages, faire des pages interactives, etc.

A. Instantiation d'objets et utilisation de propriétés et méthodes

2 méthodes d'instantiation:

new Object();

objet = {};

- expression équivalentes qui créent un objet

```
var objet = new Object();
var objet = {};
```

Lire la valeur d'une propriété

objet.propertyName

obj["nom_propriété"];

- expressions équivalentes pour lire la valeur d'une propriété

```
var name = obj.name;
var name = obj["name"]
```

Assigner une valeur à une propriété

objet.propertyName = ...

obj["nom_propriété"] = ...

- expressions équivalentes pour lire la valeur d'une propriété

```
obj.name = "James_Bond";
obj["name"] = "James";
```

Invocation d'une méthode

objet.nom_méthode(arguments...)

- une méthode d'un objet est invoquée selon une syntaxe "habituelle"

B. Création d'objets

- JavaScript vous permet aussi de créer vos propres objets.
- Il existe au moins 2 façons de faire, ici on utiliser la méthode qui consiste à définir simplement une fonction "constructrice".
- Exemple qui crée "une voiture"

  url: **http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide:Creating_New_Objects**

Exemple simple:

```
function car(make, model, year) {
   this.make = make;
   this.model = model;
   this.year = year;
}
```

- Cette fonction "car" crée implicitement un objet
- chaque ligne this.xxx assigne un paramètre passé lors de la création à une variable interne (propriété)

```
var ta_voiture = new car("Toyota", "Corolla", 2005);
var ma_voiture = new car("Mazda", "Miata", 1990);
```

Objets dans objets

- Dans cet exemple, on définit d'abord un 2ème objet

```
function person(name, age, sex) {
   this.name = name;
   this.age = age;
   this.sex = sex;
}
```

- on crée une instance

```
rand = new person("Rand McKinnon", 33, "M");
```

- on modifie la fonction de construction pour le type d'objet "car"

```
function car(make, model, year, owner) {
   this.make = make;
   this.model = model;
   this.year = year;
   this.owner = owner;
}
```

- Lorsqu'on crée un instance de car, on peut maintenant lui indiquer son propriétaire.

```
bagnole = new car("Eagle", "Talon TSi", 1993, rand);
```

- Pour connaitre le propriétaire de bagnole:

```
bagnole.owner.name
```

Autrement dit: On peut descendre une hiérarchie d'objets en enchaînant: v.x.y.z

Définition de méthodes

object.methodname = function_name

- associe un nom de fonction à une méthode

Voici la suite de l'exemple:

- Définition d'une fonction

```
function displayCar() {
   var result = "A Beautiful " + this.year + " " + this.make
      + " " + this.model;
   pretty_print(result);
}
```

- Définition de cette fonction comme méthode pour le type d'objet "car".

```
function car(make, model, year, owner) {
   this.make = make;
   this.model = model;
   this.year = year;
   this.owner = owner;
   this.displayCar = displayCar;
}
```

- Exemple d'utilisation

```
bagnole.displayCar()
```

4. Insertion de code JavaScript dans une page

4.1 HTML: Plusieurs possibilités

A. Insertion dans un fichier *.html

- Utilisation de la balise “script”

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

- on insère normalement toutes les fonctions et initialisations dans le “head” de la page HTML (cela assure que les procédures soient connues, avant d’être appelées).

Exemple 4-1: Bonjour avec Javascript

url: **Pour voir: http://tecfa.unige.ch/guides/js/ex-intro/hello-world1.html**

- programme sayhello() qui permettra d’afficher une petite fenêtre avec le texte: "Bonjour cher lecteur !!!"

```
<HEAD>
<TITLE>Hello World avec JavaScript (18-Nov-1997)</TITLE>

<script language="JavaScript" type="text/javascript">
  // ICI on definit une fonction JavaScript
  function sayhello() {
    alert("Bonjour cher lecteur !!! ")
  }
</script>
</HEAD>
<BODY>
```

B. Autre possibilités:

- Le code réside dans fichier externe:

```
<script type="text/javascript" language="Javascript" src="buttons.js">
</script>
```

- Certains tags HTML peuvent contenir des attributs avec des valeurs en JavaScript (à suivre)

C. Gestion de browsers qui ne comprennent pas JavaScript

Cacher un script:

```
<SCRIPT>
<!-- Begin to hide script contents from old browsers.

..... expressions JavaScript...

// End the hiding here. -->
</SCRIPT>
```

Afficher un contenu alternatif pour ces browsers: <noscript>

```
<NOSCRIPT>

<B>This page uses JavaScript, so you need to get Netscape Navigator 2.0
or later!
<BR>
<A HREF="http://home.netscape.com/comprod/mirror/index.html">
<IMG SRC="NSNow.gif"></A>
If you are using Navigator 2.0 or later, and you see this message,
you should enable JavaScript by on the Advanced page of the
Preferences dialog box.

</NOSCRIPT>
```

D. Gérer différentes versions de JavaScript:

- Il existe des recommendations sur tous les sites "Webmaster" et/ou Javascript

E. Commentaires:

- Toute ligne qui commence par un // est un commentaire:

```
<script language=JavaScript>
  // CECI EST UN COMMENTAIRE
  // faites des commentaires pour documenter votre code !
```

4.2 Javascript avec XHtml

- XHTML est un langage XML avec toutes les contraintes qui en découlent, il est notamment interdit d'insérer des formalismes autres que XML.

A. Utilisation d'un fichier externe

- Même principe que pour HTML (mais utiliser des minuscules !!)

```
<script type="text/javascript" language="Javascript" src="bla.js">
</script>
```

Note: Ne minimsez pas cette balise, sinon IE 6/7 ne va pas afficher la page.

B. Utilisation de sections "CDATA"

- Les "CDATA" sont des "character data" non interprétés, donc légales.
- Normalement, tout code JS devrait être dans une section CDATA. Ceci n'est pas nécessaire pour le pseudo XHTML, mais pour du XHTML servi en tant que XHTML (XML).

```
<script language="JavaScript">
<![CDATA[
   alert ("salut, je suis un alert depuis une page XHTML");
]]> </script>
```

C. Formulaires "Html" avec XHtml

- Attention: en XML chaque attribut doit avoir une valeur !!
  - au lieu de HTML: **`<input type="radio" name="choice" value="1" checked>`**
  - utiliser en XHTML: **`checked="checked"`**

4.3 Utilisation de code JavaScript

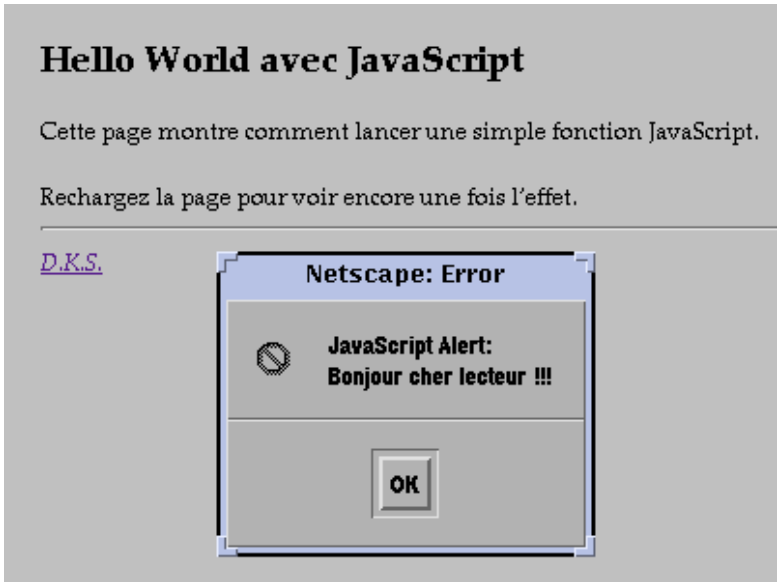
☞ une fonction JavaScript est déclenchée par 2 moyens:

A. Appel de fonction dans le <body> d'une page HTML

- Pour déclencher la fonction définie à la page 20,
  insérez le code suivant quelque part dans votre page:

```
<script>
 // ICI on apelle la fonction
sayhello()
</script>
```

url: **Pour voir:** http://tecfa.unige.ch/guides/js/ex-intro/hello-world1.html



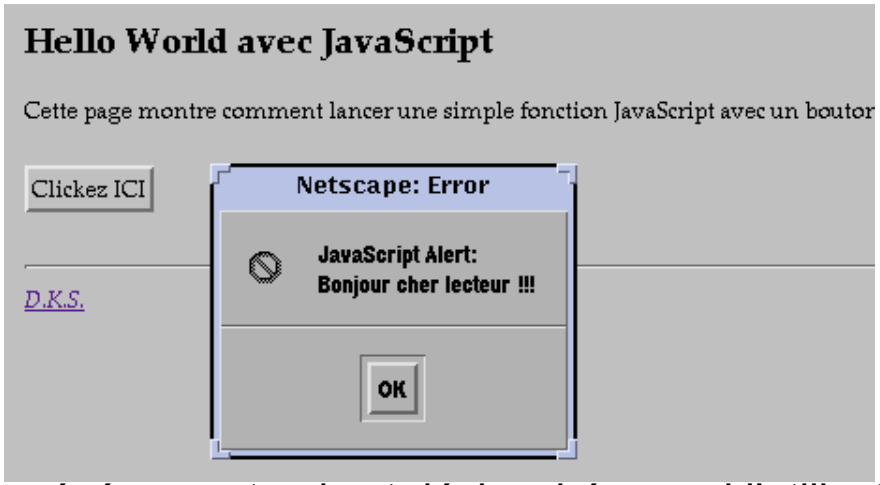
B. Appel de fonction par un “événement”

la notion d’événement en bref:

- un événement est produit par un “geste” de l’utilisateur
- par exemple: bouger la souris, cliquer sur un bouton, remplir un champs etc.
- JavaScript pré-définit un certain nombre d’événements que l’on peut exploiter

Exemple 4-2: Hello avec un événement Javascript

```
<FORM METHOD="post">
<INPUT TYPE="button" VALUE="Clickez ICI" onClick="sayhello()">
</form>
```



- **“onClick”** est un événement qui est déclenché quand l’utilisateur clique sur le bouton.
- Il existe plusieurs méthodes pour associer des événements à une fonction qui les gère (event handlers). Ci-dessus on utiliser la méthode "HTML inline", la plus simple.
- Une fois l’événement déclenché, la fonction “sayhello” est exécutée.

url: **Pour voir: http://tecfa.unige.ch/guides/js/ex-intro/hello-world2.html**

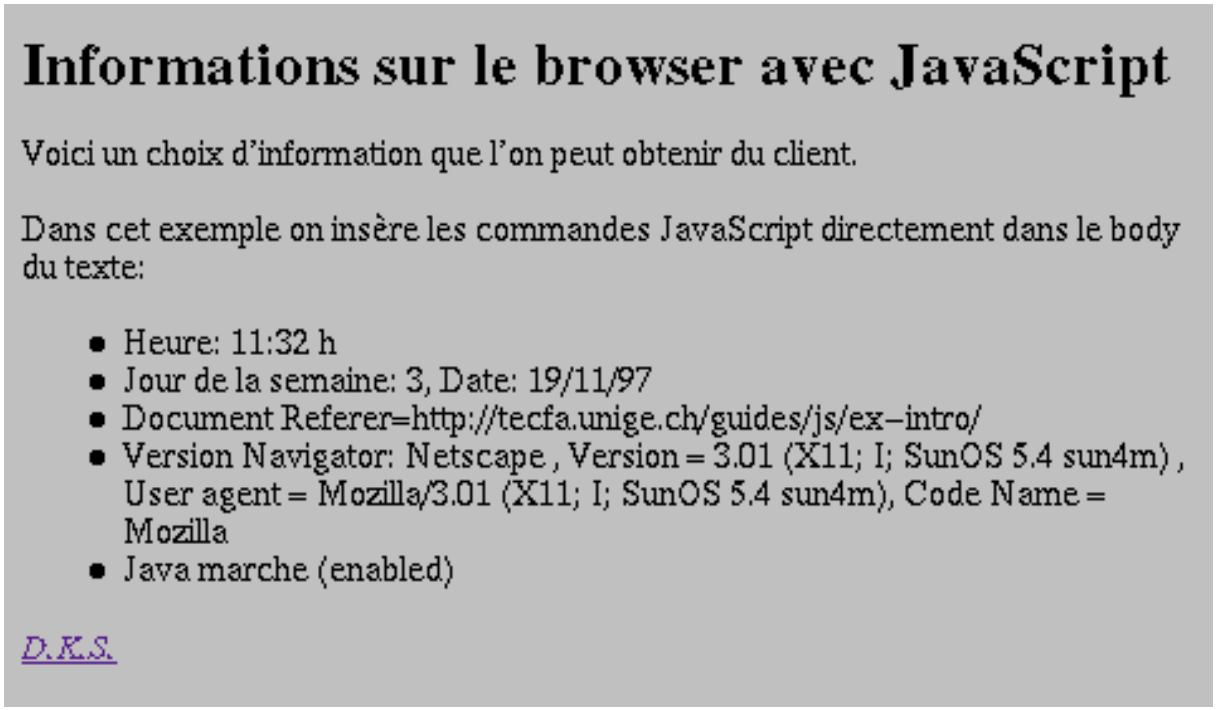
5. Manipulation du browser et de fenêtres

5.1 Informations sur le navigateur

- Avec JavaScript on peut obtenir de l'information sur le client que vous utilisez
- Cette exemple montre aussi comment générer du HTML avec Javascript

Exemple 5-1: Information sur le browser avec Javascript

url: **Pour voir: http://tecfa.unige.ch/guides/js/ex-intro/info.html**



Informations sur le browser avec JavaScript

Voici un choix d'information que l'on peut obtenir du client.

Dans cet exemple on insère les commandes JavaScript directement dans le body du texte:

- Heure: 11:32 h
- Jour de la semaine: 3, Date: 19/11/97
- Document Referer=http://tecfa.unige.ch/guides/js/ex-intro/
- Version Navigator: Netscape , Version = 3.01 (X11; I; SunOS 5.4 sun4m) , User agent = Mozilla/3.01 (X11; I; SunOS 5.4 sun4m), Code Name = Mozilla
- Java marche (enabled)

D.K.S.

Voici le code:

```
<script language="JavaScript">

// pour connaître l'heure et la date il faut d'abord créer un objet Date.
today = new Date()

document.write("<li>Heure: ",today.getHours(),":",today.getMinutes(), " h ")

document.write("<li>Jour de la semaine: ", today.getDay(), ", Date: ", today.getDate(),"/",
today.getMonth()+1,"/",today.getYear());

document.write("<li>Document Referer=", document.referrer);

document.write("<li>Version Navigator: ", navigator.appName, " , Version = ", navigator.appVersion, " ,
User agent = ", navigator.userAgent, ", Code Name = ", navigator.appCodeName);

if (navigator.javaEnabled()) {
   document.write("<li>Java marche (enabled)");
   }
   else
   document.write("<li>Java ne marche pas (faites quelque chose!)");

</script>
```

- document.write() est une méthode pour "écrire" dans le document courant.
- Important: Il n'est pas possible de modifier ce qui a été écrit sans recharger la page.

5.2 Ouvrir une nouvelle fenêtre

Exemple 5-2: Création d'une nouvelle fenêtre I

url: **http://tecfa.unige.ch/guides/js/ex-intro/fenetre.html**

- Notez comment on appelle les méthodes: `nom_objet.méthode()`
- la méthode `window.open()` permet d'ouvrir une nouvelle fenêtre du browser (voir la documentation pour les paramètres)
- Lorsqu'on ouvre une nouvelle fenêtre, ca crée un objet qu'il faut stocker dans une variable (ici: `win`) afin de pouvoir l'utiliser

```
win = window.open("", "Resultats", .....);
```

- la méthode `xxx.document.open` initialise l'objet document de la fenêtre (dans lequel on va "écrire")

```
win.document.open();
```

- "document" est une propriété de l'objet xxx qui contient l'objet document associé avec le window xxx.
- xxx.document.writeln () permet d"écrire" du code HTML
- xxx.document.close va finaliser la page (afficher le contenu pour l'utilisateur).

```
win.document.close();
```

- Note: document.writeln ("bla") (sans le xxx) aurait été utilisé pour écrire quelque chose dans la page HTML courante, mais comme on écrit depuis la page courante vers une autre fenêtre, il faut utiliser:

```
win.document.writeln ("Salut");
```

Voici le code:

```
function ouvrir () {
  // on crée un nouvelle fenêtre
  win = window.open("", "Resultats",        "width=250,height=150,status=1,resizable=1");
  // on ouvre l'acces au contenu de la fenêtre
  win.document.open();
  // Ici on devrait d'abord écrire une jolie entête HTML <head> ....
  // Mais on s'en passe et on affiche juste un message
  win.document.writeln ("<h1>Message secret</h1>");
  win.document.writeln ("Bonjour cher ami!");
  // On rajoute un bouton pour fermer cette fenetre
  win.document.writeln ("<hr><center><form><input type='button' value='FERMER'
onClick='window.close()'>");
  // Et on finalise, rien n'est affiché avant ce document.close () !!
  win.document.close();
}
....

<hr><form>
<input type="button" name="ProcessButton" value="Voir un truc"
       onClick="ouvrir()">
</form><hr>
```

Exemple 5-3: Création d'une nouvelle fenêtre II

Cet exemple est plus compliqué, il montre

- comment on peut faire un bouton permettant de fermer la fenêtre
- comment écrire une fonction un peu générale
- comment utiliser une variable pour "construire" le contenu (c'est idiot d'écrire pleins de xx.document.writeln)

url: **Voir: http://tecfa.unige.ch/guides/js/exemples/createw.html**

6. Traitement de formulaires avec Javascript

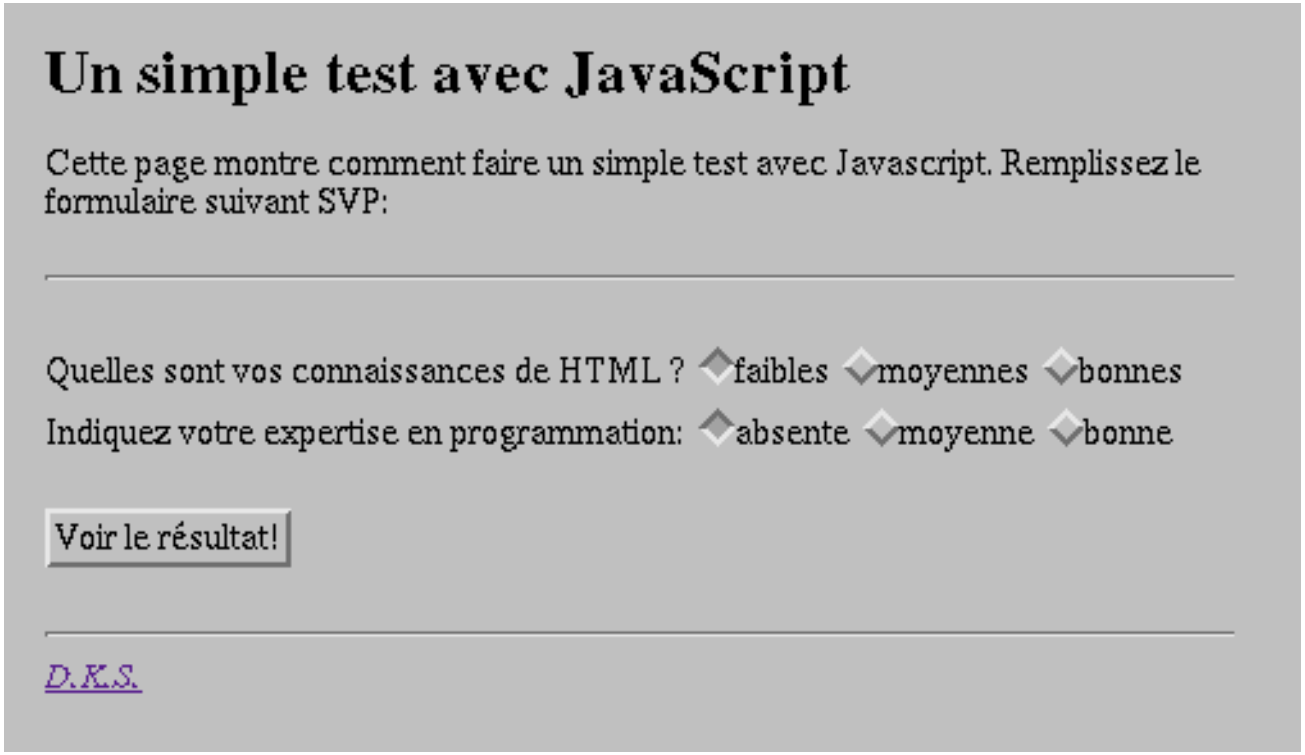
6.1 Un simple quiz

A. Solution style "grand-mère"

Exemple 6-1: Simple quiz avec Javascript

url: **cf. http://tecfa.unige.ch/guides/js/ex-intro/test1.html**

url: **pour XHTML cf. http://tecfa.unige.ch/guides/js/ex-intro/test1.xhtml**



Voici le code:

```
<HTML>
  <HEAD>
    <TITLE>Un simple test avec JavaScript (18-Nov-1997)</TITLE>
   <script language="JavaScript">
     // Initialisation de variables
      var q1 = 1;
      var q2 = 1;
    // calcul
      function calculer () {
      score = q1 + q2;
      alert("Sur une échelle qui va de 2 à 6 vous avez " + score)
      }
</script>
</HEAD>
  <BODY>
    <H1>Un simple test avec JavaScript</H1>
Cette page montre comment faire un simple test avec Javascript.
Remplissez le formulaire suivant SVP: <P>
    <hr><form>
    Quelles sont vos connaissances de HTML ?
    <input type="radio" name="choice" value="1" onClick="q1=1" checked>faibles
    <input type="radio" name="choice" value="2" onClick="q1=2">moyennes
    <input type="radio" name="choice" value="3" onClick="q1=3">bonnes
    <br>
    Indiquez votre expertise en programmation:
    <input type="radio" name="choice2" value="1" onClick="q2=1" checked>absente
    <input type="radio" name="choice2" value="2" onClick="q2=2">moyenne
    <input type="radio" name="choice2" value="3" onClick="q2=3">bonne
    <P>
    <input type="button" name="ProcessButton" value="Voir le résultat!"
           onClick="calculer()">
    </form><hr>
    </HTML>
```

A retenir:

- L'utilisation de "onClick":
  - Chaque fois que l'utilisateur clique sur un bouton radio, on enregistre la valeur du bouton dans une variable:

```
<input type="radio" name="choice" value="1" onClick="q1=1" checked>faibles
<input type="radio" name="choice" value="2" onClick="q1=2">moyennes
<input type="radio" name="choice" value="3" onClick="q1=3">bonnes
```

  - Notez que la variable "q1" a été définie préalablement dans le code
- L'appel à la fonction du calcul lors du submit:

```
<input type="button" name="ProcessButton" value="Voir le résultat!"
onClick="calculer()">
```

- Le calcul:

```
function calculer () {
     score = q1 + q2;
     alert("Sur une échelle qui va de 2 à 6 vous avez " + score)
     }
```

- Affichage des résultats
  - on utilise un simple "alert", ce qui n'est pas très beau.
  - Voir **http://tecfa.unige.ch/guides/js/ex-intro/test2.html !**
  - ou s'inspirer de l'exemple 5-2 "Création d'une nouvelle fenêtre I" [28]

B. Traitement de formulaire style "DOM ancien"

- Il n'est pas nécessaire d'enregistrer l'effet de chaque "click", mais on peut aussi avec JS interroger l'état d'un formulaire en utilisant l'objet "document"
- C'est une solution plus difficile à comprendre pour les débutants, mais plus efficace au niveau écriture de code.

Chaque élément de la page correspond à un objet qui s'insère dans un arbre hiérarchique. "***les boutons radio réponse1 sont dans le formulaire questions qui se trouve dans le document par défaut".***

On nomme l'objet en écrivant sa hiérarchie :

- document.questions.reponse1

Voici quelques exemples pour accéder à des propriétés ou à des méthodes :

- soumettre le formulaire ***questions*** (méthode):

```
document.question.submit();
```

- mettre le nombre d'éléments du formulaires ***questions*** dans une variable (propriété):

```
elements = document.questions.length;
```

Exemple 6-2: Traitement du formulaire II : interrogation des valeurs

url: **http://tecfa.unige.ch/guides/tie/html/interactive-intro/examples/value-form.html**
url: **(code) /guides/tie/html/interactive-intro/examples/value-form.txt**

Voici la fonction ***displayScore*** modifiée et commentée:

```
  function displayScore () {
    document.questions.submit.checked = false;

    // on déclare la variable score et on initialise à 0
    var score = 0;

    // on regarde combien d'éléments il y a dans le formulaire
    elements = document.questions.length;

    // on fait ensuite un boucle sur le nombre d'éléments pour les passer en revue un
par un.
    var i;
    for (i = 0; i < elements; i++) {
      // on regarde si l'élément est "checked". Si oui, on calcule le score
      if (document.questions.elements[i].checked) {
 score = score + eval(document.questions.elements[i].value);
      }
    }
    // feedback
    ...
```

Exemple 6-3: Interrogation d'un formulaire (autre exemple)

url: **Voir: http://tecfa.unige.ch/guides/js/exemples/interactif2.html**

C. Traitement de formulaire style DOM W3C

Voici une manière de faire assez moderne:

- Il n'y plus de code JavaScript dans le HTML
- Les événements sont associés à des noms de fonction par des méthodes d'inialisation

url: **Voir http://tecfa.unige.ch/guides/js/ex-intro/test-dom.html**

Code HTML:

```
<form id="quiz" action="#" method="get">
<input type="submit" value="Voir le résultat!" /> </form>
```

Code JavaScript:

```
// Ceci associe la fonction init à l'événement charger la page
// Quand cela charge, le navigateur va chercher la fonction et l'exécuter
window.onload = init;

// Cette fonction associe une éventhandler function "doit" à l'évévenement
// où l'utilisateur soumet la forme

function init () {
  document.getElementById("quiz").onsubmit = doit;
}
```

Voir les slides sur DOM et JS:

url: **http://tecfa.unige.ch/guides/tie/html/js-intro/js-dom.html**

6.2 Vérification d'un formulaire I

- Voi**r: http://tecfa.unige.ch/guides/js/ex-intro/test1-verif.html**

A retenir:

- Il y a pleins de façons de faire, voir les sites "WebMaster"
- Ici on fait simple, car on a peu de questions:
  - on initialise au départ les variables q1 et q2 à -1

```
var q1 = -1;
var q2 = -1;
```

  - on écrit une fonction qui teste si une de ces variables est encore sous 0

```
function verif () {
  if ( (q1 < 0) || (q2 < 0) ) {
    alert ("Il faudrait remplir tous les champs SVP !");
    return false;
  }
  else return true;
}
```

  - notez que cette fonction retourne **`false`** si c'est le cas et **`true`** si tout va bien.
  - Ensuite, la fonction calculer appelle tout d'abord cette fonction verif. Si verif retourn false on sort, sinon on continue .....

```
function calculer () {
  // Si verif retourn false on quitte le navire
  if (!verif()) return ;
  // sinon on continue
  score = q1 + q2;
  alert("Sur une échelle qui va de 2 à 6 vous avez " + score) ;}
```

- ... le reste est pareil

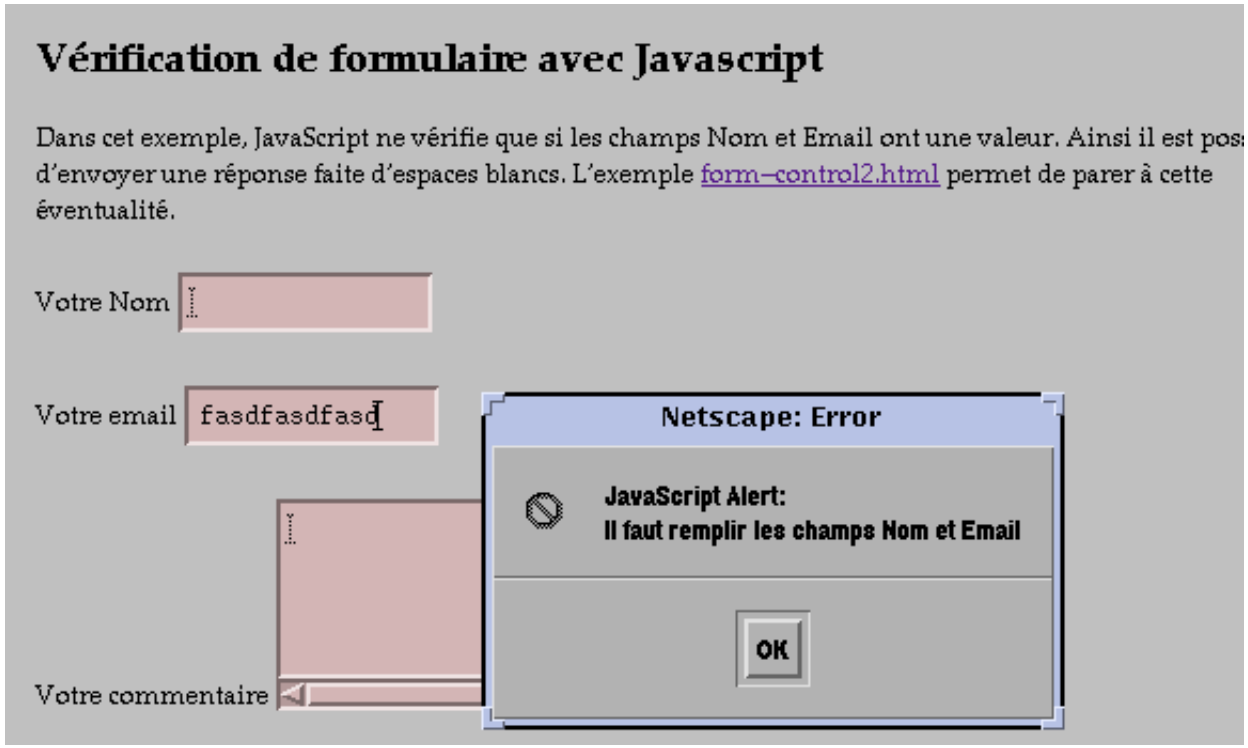
6.3 Vérification d'un formulaire (II)

Exemple 6-4: Vérification d'un formulaire

url: **http://tecfa.unige.ch/guides/js/ex-intro/form-control.html**

Cet exemple reprend le formulaire discuté dans le module "Les formulaires HTML" et y ajoute deux fonctions Javascript permettant de vérifier que l'utilisateur n'a pas laissé de champs vides.

- Beaucoup de sites utilisent JS pour vérifier un formulaire avant de l'envoyer à un CGI pour traitement.



A remarquer:

- Le bouton qui permet d'envoyer le contenu du formulaire est différent de l'exemple «Envoyer le contenu d'un formulaire avec email» [p. 21]:
  - Lorsque l'utilisateur clique sur le bouton "envoi" la fonction 'checkForm' est appelée avec le contenu du formulaire en argument: checkForm(this.form)

  AVANT

  ```
  <input type="submit" value="Envoi">
  ```

  APRES (avec vérification):

  ```
  <input type="button" value="Envoi"  onClick="checkForm(this.form)">
  ```

- La fonction "checkForm" appelle la fonction "checkBlank", qui vérifie:
  - (1) est-ce que les champs en question ont une valeur (est-ce que l'utilisateur a tapé quelque chose).
  - Si la question (1) reçoit une reponse positive, alors la fonction checkBlank retourne la valeur true qui signifie dans notre cas, 'tout est en ordre'.
  - Dès lors le test dans la fonction checkForm est satisfait et l'appel form.submit() envoie le contenu du formulaire.
- A faire mieux:
  - La fonction checkBlank ne détecte pas si l'utilisateur a utilise un espace blanc dans sa réponse.
  - L'exemple 6-4 "Vérification d'un formulaire" [39] permet de parer à cette éventualité.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function checkBlank(input,msg)
{
  if (input.value == null || input.value.length == 0) {
    alert ("Il faut remplir les champs Nom et Email");
    return false;
  }
  return true;
}
function checkForm(form)
{
  if (
      !checkBlank(form.nom) ||
      !checkBlank(form.email)) {
    return false;
  }
  form.submit();
    alert ("Merci pour votre reponse ...");
  return true;
}
</SCRIPT>
</HEAD>
<BODY>
<h1>Vérification de formulaire avec Javascript</h1>
<form enctype="application/x-www-form-urlencoded"
  action="mailto:Patrick.Jermann@tecfa.unige.ch" method=post>
Votre Nom <input type="text" name="nom" size=15> <p>
Votre email <input type="text" name="email" size=15> <p>
Votre commentaire <textarea name="comment" rows=5 cols=30></textarea><p>
<input type="button" value="Envoi" onClick="checkForm(this.form)">
<input type="reset" value="Effacer">
</form>
</BODY></HTML>
```

A. Vérification d'un formulaire (III)

url: **Cf.** http://tecfa.unige.ch/guides/js/ex-intro/form-control2.html

- La fonction checkForm appelle la fonction checkBlank qui vérifie deux choses:

  1) est ce que les champs en question ont une valeur (est-ce que l'utilisateur a tapé quelque chose).

  2) Est-ce que les champs sont remplis par autre chose que des espaces ?

  Si les questions 1) et 2) ont une reponse positive alors la fonction "checkBlank" retourne la valeur true qui signifie dans notre cas, 'tout est en ordre'. Dès lors le test dans la fonction chekForm est satisfait et l'appel form.submit() envoie le contenu du formulaire.

Voici la fonction checkBlank modifiée:

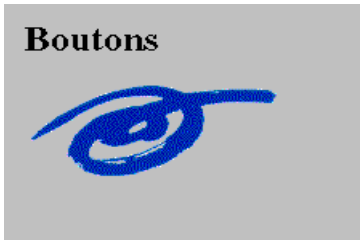
(voir la source de la page HTML ci-dessus pour l'ensemble)

```
function checkBlank(input,msg)
{
   if (input.value == null || input.value.length == 0) {
    alert ("Il faut remplir les champs Nom et Email");
    return false;
  }
  var str = input.value;
  for (var i = 0; i < str.length ;i++){
    var ch = str.substring(i,i+1);
    if ( ch == " ") {
      alert (msg );
      return false;
    }
  }
  return true;
}
```

7. HTML Dynamique (ou presque)

7.1 Les boutons JavaScript

url: **Cf. http://tecfa.unige.ch/guides/js/ex-intro/switch-buttons.html**




```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
off_logo = new Image(120,25);
off_logo.src = "logo.gif";
on_logo = new Image(120,25);
on_logo.src = "logo_on.gif";

function hiLite(imgDocID,imgObjName) {
      document.images[imgDocID].src = eval(imgObjName + ".src")
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="http://agora.unige.ch/"
     onMouseOver="hiLite('logo','on_logo')"
     onMouseOut="hiLite('logo','off_logo')">
<IMG SRC="newimages/logo.gif" BORDER=0  NAME="logo"></A>
</BODY></HTML>
```

7.2 Menus déroulants

- Faire un menu déroulant est assez difficile
- Il faut tenir compte des différents browsers (sauf si vous misez sur les toutes dernières générations)
- Il existe pleins de sites qui donnent des scripts
  voir les Webmaster's sites: http://tecfa.unige.ch/guides/toolbox.html

Exemple 7-1: Les HierMenus de webreference.com

- Un menu DHTML hiérarchique qui marche avec la plupart des clients WWW
  url: http://webreference.com/dhtml/hiermenus/
- Il faut bien lire les instructions (et surtout respecter les nom des array)
  url: http://webreference.com/dhtml/hiermenus/instructions/