

Client-side Scripting



Préparé par :

**Neila Ben Lakhal PhD@ Tokyo
Institute of Technology Japan**

E-mail : neila.benlakhal@gmail.com

Les caractéristiques de JavaScript

- Javascript a été initialement inventé par Brendan Eich en 1995 et développé par Netscape et s'appelait alors **LiveScript**.
- Il est inspiré de nombreux langages, notamment de [Java](#) mais en simplifiant la syntaxe pour les débutants.
- C'est un langage de scripts qui incorporé aux balises Html, permet d'améliorer la présentation et l'interactivité des pages Web en les rendant dynamiques.
- Pour faire simple, un script est, par opposition à un langage compilé, **un langage qui s'interprète**. Ici, l'interprète du JavaScript, c'est le navigateur du visiteur (le client).
- Ces scripts vont être **gérés et exécutés par le browser** lui-même **SANS** devoir faire appel aux ressources du serveur.

Les caractéristiques de JavaScript..

- Les versions récentes du langage JavaScript ont pour origine les spécifications de la norme ECMA-262 définissant ECMAScript :
 - Version 1.0 (March 1996) supporté par IE et Netscape.
 - Puis les Versions 1.1 (August 1996), 1.2, 1.3, et 1.4 .
 - Version 1.5 (November 2000) supportée par IE, Netscape, Firefox, Opera, Safari et chrome.
 - Version 1.8.5 (2010).
- Javascript est un **langage script orienté objets**, ce qui signifie que vous pouvez créer des classes et instancier des objets.
- Javascript propose en standard un certains nombres de classes que vous pourrez utiliser à votre aise pour créer vos programmes.
- Il sera donc possible d'avoir des syntaxes du type `objet.methode(parametre);`

DOM ?

- Les scripts JS peuvent Interagir avec le document HTML via l'interface **Document Object Model (DOM)**, (standardisé par le W3C) fournie par le navigateur (on parle alors de HTML dynamique ou DHTML).
- Selon le W3C: « DOM est une API qui permet aux programmes et scripts de naviguer dans l'arbre des éléments d'un document, d'en lire les nœuds et de modifier dynamiquement le contenu, la structure et le style de documents de type HTML, XHTML et XML ».

DOM

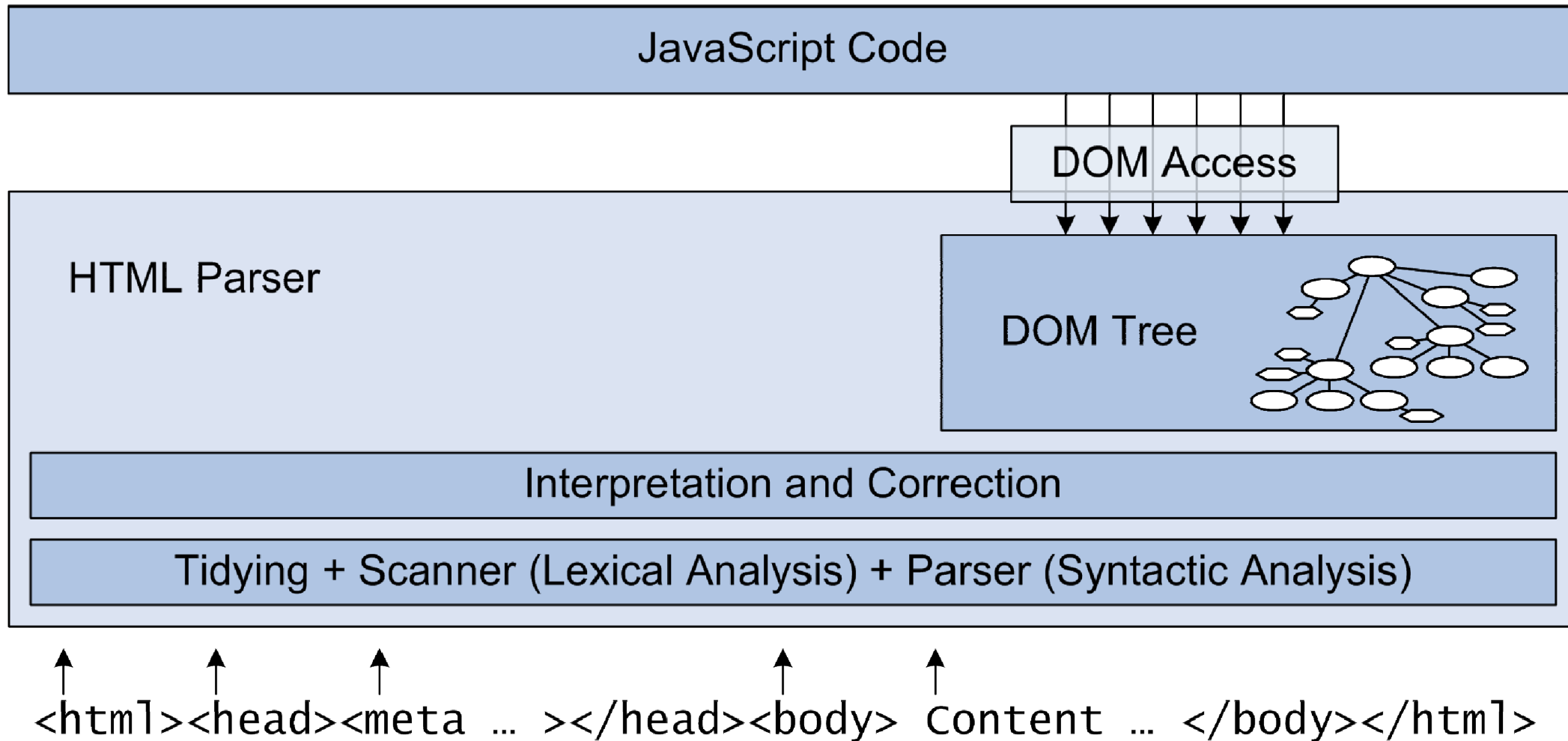
■ Les versions de DOM:

- DOM0 (proposé par Netscape); date de 1998, DOM1 la première version proposée par W3C
- DOM2 la version standard et stable
- DOM3 (working Draft)
- La dernière version DOM 3 date de 2004.
- Seul la version 1 et 2 sont finalisées.
- DOM comprend deux parties :
 - Une partie DOM core : qui s'applique aussi à XML, avec quelques petites différences, et une autre spécifique au HTML.
 - Une partie DOM events (event handlers)

Les étapes d'interprétation des pages HTML

- Recevoir la page .html du serveur en tant que document de type text/HTML (MIME)
- Parser le document et corriger les différentes erreurs potentielles
- Interpréter le document comme si il contenait pas d'erreurs
- Apporter les ressources reliées à la page (avec un GET: CSS, images, JavaScript, ...)
- Construire l'arbre DOM du document en se basant sur l'interprétation zéro erreur
- Appliquer les règles de styles CSS (interne, externe etc.)
- Visualiser le document (structure)
- Commencer à exécuter le code JS
- Détecter les événements du clavier, souris, timer etc. et exécuter le code.
- Tout détruire et reprendre les mêmes étapes quand l'utilisateur quitte vers une page différente.

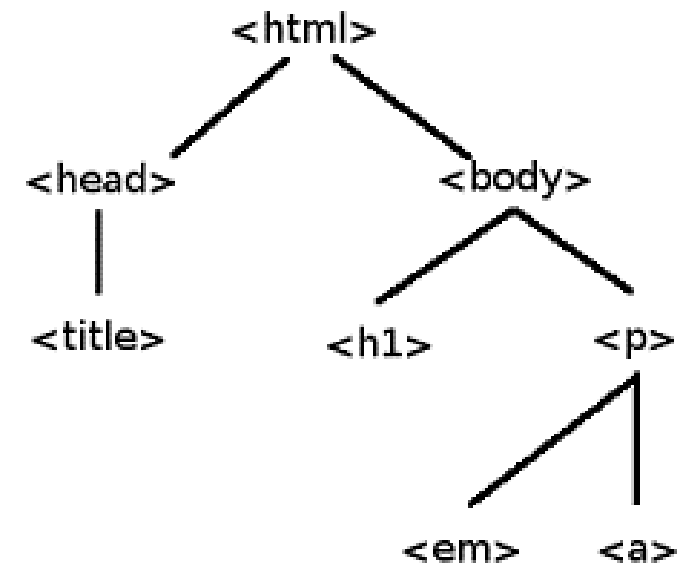
Fonctionnement interne du navigateur



Notion d'arbre

- On peut schématiser une page HTML par un arbre, comme celui-ci:

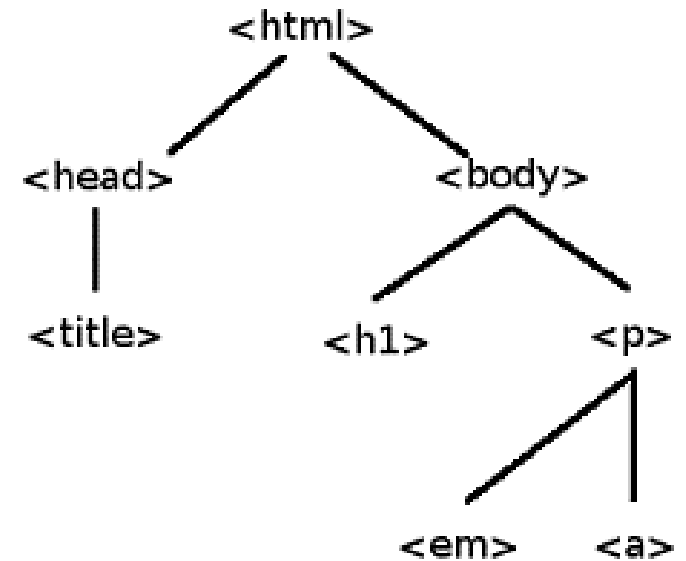
```
<html>
  <head>
    <title>...</title>
  </head>
  <body>
    <h1>...</h1>
    <p>...<em>...</em>...<a>...</a>
    </p>
  </body>
</html>
```



- Pourquoi faire ?
- Lorsque l'on va manipuler ces pages avec du JavaScript, **il va falloir se balader dans notre page → comprendre comment la page est ordonnée devient alors très important.**

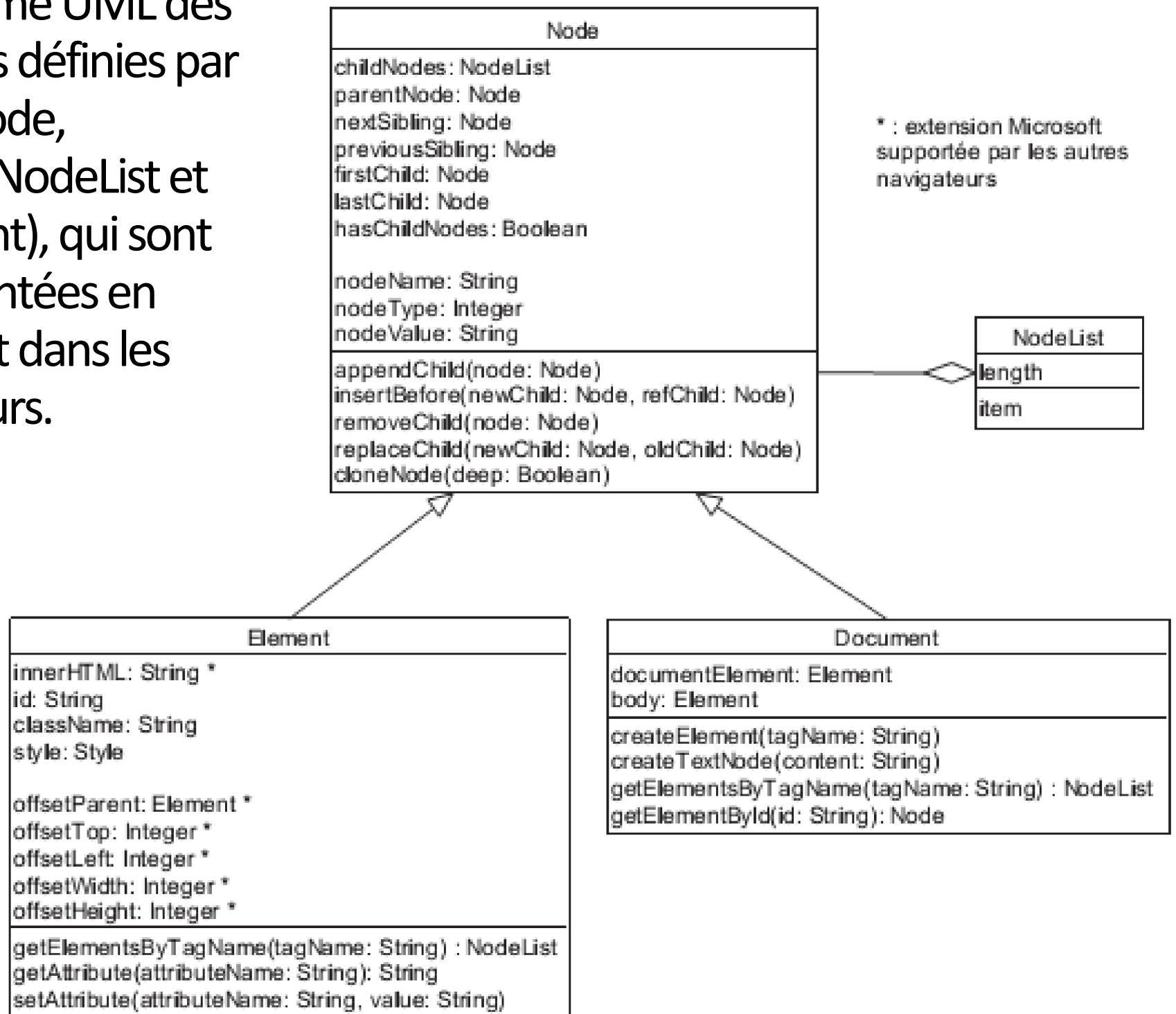
La notion de nœuds

- Dans notre exemple,
- HTML est le **parent** de HEAD et BODY.
- H1 est un **enfant** de BODY.



- En langage Javascript, chaque séparation d'une branche en plusieurs s'appelle un **node** (**nœud**). Nous avons alors le nœud HTML, le nœud HEAD, etc. Il existe deux grands types de node :
 - **élément** : les nœuds que l'on va appeler éléments sont les balises du XHTML que vous connaissez.
 - **texte** : ces nœuds sont en fait du texte brut qui se trouve entre les balises.

- diagramme UML des interfaces définies par DOM (Node, Element, NodeList et Document), qui sont implémentées en JavaScript dans les navigateurs.



Les implémentations de Javascript

- Les implémentations actuelles de Javascript sont multiples (exemples) :
 - Safari utilise [SquirrelFish JavaScript engine](#)
 - Chrome utilise [V8 JavaScript engine](#) (Google's open source JavaScript engine écrit en C++)
 - Etc.
- Par soucis de compatibilité, et pour garantir que votre code satisfait la propriété « **cross-browsing** », il y a un test qui a été mis en place:
- Le Test ACID3 (<http://acid3.acidtests.org/>) a été défini par Web standards c'est un score sur 100 indiquant si votre navigateur permet de développer des applications web 2.0 dynamique principalement tester le support de JS, mais aussi de :

ACID3 Test

DOM2 Core
DOM2 Events
DOM2 HTML
DOM2 Range
DOM2 Style (getComputedStyle, ...)
DOM2 Traversal (NodeIterator, TreeWalker)
DOM2 Views (defaultView)
ECMAScript
HTML4 (<object>, <iframe>, ...)
HTTP (Content-Type, 404, ...)
Media Queries

Selectors (:lang, :nth-child(), combinators, dynamic changes, ...)
XHTML 1.0
CSS2 (@font-face)
CSS2.1 ('inline-block', 'pre-wrap', parsing...)
CSS3 Color (rgba(), hsla(), ...)
CSS3 UI ('cursor')
data: URIs
SVG (SVG Animation, SVG Fonts, ...)

Remarque : il y a aussi ACID1 (HTML4+CSS1) et ACID2 (CSS1+CSS2)
(http://en.wikipedia.org/wiki/Web_Standards_Project)

What is JavaScript Framework?

- En écrivant du code Javascript, ils faut prendre en considération cette diversité dans les moteurs d' exécution .
- Très complexe à faire
- Solution : une diversité de librairie (Framework) ont vu le jour
 - Un Framework JavaScript est un ensemble d'applications et de fonctionnalités regroupés (API) : boîte à outils qui contient toutes les tâches répétitives et courantes.
 - Un Framework permet de gagner beaucoup de temps dans vos développements, facilite la manipulation de DOM, faire des requêtes AJAX, écrire des applications plus robustes et plus riches en fonctionnalités.
 - Un Framework permet d'alléger le développeur des taches récurrentes pour se concentrer sur des taches plus complexes : Il n'est plus nécessaire de faire le développement de zéro.

What is JavaScript Framework?

- Principal apport des Frameworks JS : *Cross-browsing* (garantir la compatibilité du code JS avec divers navigateurs)
- La diversité des plateformes Web fait qu'il est difficile de garantir que votre code est compatible avec toute plateforme (cross-browsing property)
 - Environnement PC : Windows, MacOS, Linux
 - Environnements mobile : iPhone, Android, Blackberry, S60, and JavaME/JavaFX
 - Les navigateurs : le premier s'appelait worldwideweb en 1990 (l'ancien Nexus), suivi par Mosaic en 1993 puis IE en 1995
 - Chrome, Safari, Opera, IE, Maxthon, Konqueror, Opera, etc..
- À ce jour, 500+ Frameworks ont vu le jour !!!!

Exemples de Framework JS

- **Prototype**: (facilite de développement d'applications Web dynamiques) <http://www.prototypejs.org/>
- **Script.aculo.us**: ("dédié interface riche") <http://script.aculo.us/>
- **Mootools**: (pour des fonctionnalités JS avancées)
<http://mootools.net/>
- **extJS**: permet de construire des applications web interactives (initialement une extension à la bibliothèque Javascript YUI de Yahoo) (<http://tutoriel.lyondev.fr/pages/33-ExtJs3.html>) et (<http://docs.sencha.com/ext-js/4-1/>)

(lien vers les librairies les plus utilisées)

<http://net.tutsplus.com/articles/web-roundups/20-javascript-frameworks-worth-checking-out/>)

Des variantes de Javascript

- Il y en a ceux qui ont décidé même de définir leurs propres langages de script pour développer des applications Web , par exemple :
 - Google Apps Script is a JavaScript cloud scripting language that lets you extend Google Apps and build web applications. Scripts are developed in Google Apps Script's browser-based script editor, and they are stored in and run from Google's servers.
(<https://script.google.com>)
 - [Processing.js](#) (visualization environment , digital art, interactive animations, educational graphs, video games, etc.)
 - Etc.

Exemple Google apps Script


The screenshot displays the Google Apps Script editor. At the top, a browser window shows the URL <https://script.google.com/macros/d/M6PMSXD83rA6XJ6Zd93oNhl32gawrAsk/edit?splash=yes>. Below the browser, the editor title is "my first script". A menu bar includes "File", "Edit", "View", "Run", "Publish", "Resources", and "Help". A yellow notification box states "Trying to reach google.com...". A blue "Share" button is in the top right. The main editor area shows a script named "Code.gs" with the following code:

```
1 function createAndSendDocument() {
2   // Create a new document with the title 'Hello World'
3   var doc = DocumentApp.create('Hello World');
4
5   // Add a paragraph to the document
6   doc.appendParagraph('This document was created by my first Google Apps Script.');
```

Exemple de fonctionnalités possible par google apps scripts

Tutorials - Google Apps Sc x

← → ↺ 🏠 <https://developers.google.com/apps-script/articles> ☆ 📄 📄 📄 XML technologies a... 📄 1: JavaScript and Bro... 📄 a Beat Signer | Vrije U... 📄 40+ Useful HTML5 E... 📄 HTML5, continued ... 📄 Intro to jQuery | Cou... 📄 COMP9321 12s2 - W... » 📁 その他のブックマーク

 Google Developers Rechercher 🔍 Neila.BenLakhal@gmail.com Sign out

Accueil **Produits** Événements Vitrine En direct Groupes

Google Apps Script

Commentaires sur ce document

Google Apps Script Overview

- Core Concepts
 - Building Your First Script
- Development Environment
- Execution Methods for Scripts
- Building User Interfaces
 - Serving Content
- Storing Data
- Events and Triggers
- Publishing Scripts
 - Best Practices
 - Troubleshooting
 - Quotas
 - Glossary

Tutorials

These tutorials are designed to help you start using Google Apps Scripts more quickly. Some of these tutorials focus on the basics, some provide an in-depth analysis of a complex script, while others address specific nuances of the Google Apps Script system itself.

Basics and Custom Spreadsheet Functions

- [Your First Script](#) - This tutorial covers the basics of writing and executing a script, demonstrating how to create a Google Document and send an email.
- [Your First Custom Function](#) - This tutorial teaches you how to create custom spreadsheet functions which can be used as part of normal Spreadsheet formulas.
- [Sending Emails from a Spreadsheet](#) - This tutorial shows how to use Spreadsheet data to send emails to different people.
- [Reading Spreadsheet Data using JavaScript Objects](#) - This tutorial guides you through the steps of easily reading structured data from a Spreadsheet and creating JavaScript objects to facilitate access to the data.
- [Writing Spreadsheet Data using JavaScript Objects](#) - This tutorial guides you through the steps of easily reading structured data from a Spreadsheet and populating a second Spreadsheet with different views of the data.
- [Removing Duplicate Rows in a Spreadsheet](#) - This tutorial shows how to remove duplicates when manipulating data in Apps Script.

Integration with Google Products and Third Party Services

Syntaxe Javascript



A quel emplacement insérer le code Javascript ?

- Il existe 2 façons d'inclure du JavaScript :
 - Soit **DANS** la page HTML:
 - Grâce aux événements (**event handler**) dans des éléments HTML en tant que attribut/valeur.
 - Grâce à la balise `<SCRIPT> ...</SCRIPT>`.
 - Soit **HORS** de la page HTML:
 - En mettant le code dans un fichier .js **externe**

Intégrer du code JS dans HTML

Méthode 1

- Pour insérer le code dans une balise, une nouvelle propriété est nécessaire. Il s'agit du **gestionnaire d'événements** (event handler).
 - Cette propriété est caractéristique du JS : elle suffit à dire au navigateur : "attention, ce qui suit est du JS".
 - Elle porte le **nom de l'événement** qui doit **déclencher le script** (c'est pour cela qu'on parle de "gestionnaire d'événements").
 - Elle contient comme valeur **le script à exécuter** lors de cet événement.

Intégrer du code JS dans HTML

- Pour résumer, la balise en question aura cette forme-là :

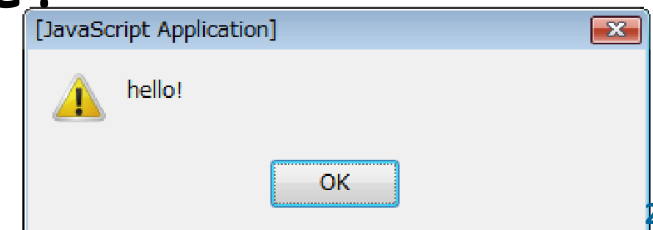
```
<nomdelabalise ... monEventHandler="monscript" />
```

- Le script **monscript** est exécuté quand l'événement **monEventHandler** est déclenché;
- Exemple: Nous allons utiliser **un lien hypertexte** :

```
<a href="http://google.com" onClick="alert('hello! ')">google</a>
```

- **L'événement** déclenchant le script sera un **clic de souris** sur ce lien. Le nom de cet événement est : **onClick**.
- Le script à exécuter sera l'ouverture d'une **boîte de dialogue en premier plan avant d'ouvrir la page de google** :

```
alert('hello !')
```



Intégrer du code JS dans HTML

- Il est possible d'écrire du JavaScript directement à la place de l'adresse d'un lien, en le faisant précéder de **javascript:** comme dans cet exemple :

```
<a href="javascript:alert('hello');">Hello</a>
```

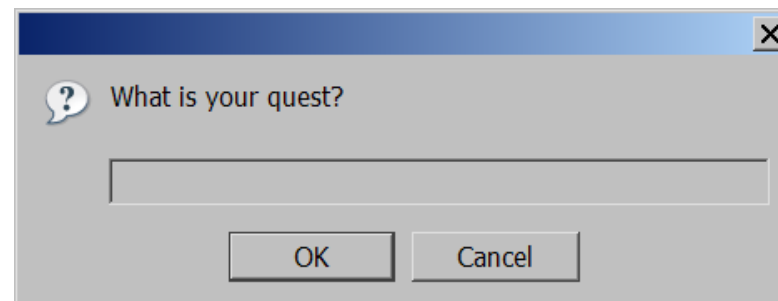
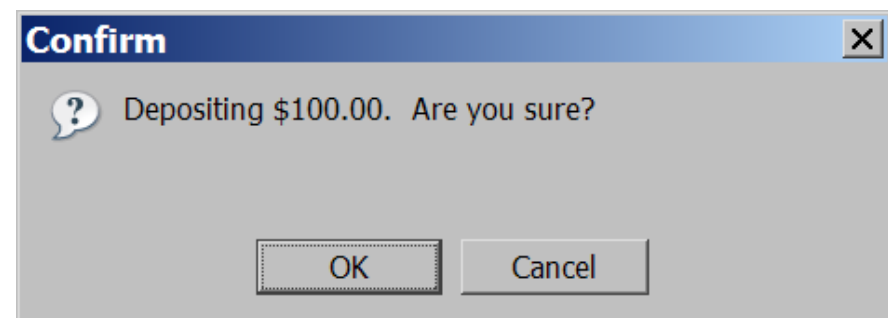
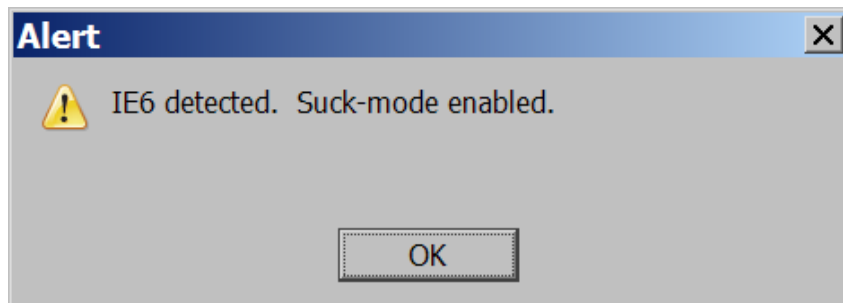
- Les 3 écritures sont équivalentes et permettent seulement d'ouvrir une boîte de dialogue :

```
<a href="#" onClick="alert('hello! ')">Hello</a>
```

```
<a onClick="alert('hello! ')">Hello</a>
```

Les boites de dialogue

```
alert("message"); // message  
confirm("message"); // returns true or false  
prompt("message"); // returns user input string
```



Autres gestionnaires d'événements

- Il existe d'autres événements que le clic de souris, leur syntaxe est la même : **onevent**, event étant le nom de l'événement.
- Les événements s'appliquent à la plupart des balises (sauf événements particuliers, relatifs à des balises précises).
- En voici quelques-uns :
 - **onClick** : au clic (gauche) de la souris,
 - **ondblclick** : lors d'un double clic,
 - **onmouseover** : au passage de la souris,
 - **onmouseout** : lorsque la souris "sort" de cet élément (en quelque sorte, le contraire de onmouseover),
 - **onload** : au chargement de la page,
 - **onunload** : lorsqu'on quitte la page (ou qu'on change de page).
 - Etc..

Autres gestionnaires d'événements

Exemples:

- Voici une image contenant plusieurs gestionnaires d'événements :

```

```

- Et voici la balise **<body>** pour créer une page disant "Bonjour" et "Au revoir" :

```
<body      onload="alert('Bonjour !');"
          onunload="alert('Au revoir !');">
  <!-- corps de la page -->
</body>
```

Intégration de script: méthode 2

Méthode 2 : écrire le script entre **<script>** et **</script>**

- Il faut commencer par préciser au navigateur que notre script est du JavaScript. Pour cela, on rajoute la propriété **type="text/javascript"**

Ce qui nous donne ceci :

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

```
/* votre code javascript se trouve ici c'est déjà plus pratique  
pour un script de plusieurs lignes ceci est un commentaire de  
plusieurs ligne */
```

```
</script>
```

Dans l'en-tête, ou dans le corps de la page ?

- On peut placer l'élément **<script>** :
 - Entre **<body>** et **</body>** : Sont à placer les scripts à exécuter au chargement de cette dernière (lorsque le navigateur "lira" le code, il l'exécutera en même temps).
 - Entre **<head>** et **</head>** : Sont à placer dans l'en-tête les éléments qui doivent être exécutés plus tard (lors d'un événement particulier, par exemple). Dans ce cas, le code n'est pas écrit "en vrac", nous apprendrons plus loin comment l'organiser.
- Exemple:
 - une boîte de dialogue indiquant le début du chargement de la page (donc, le code est à placer au début du corps de la page),
 - une autre indiquant la fin du chargement de celle-ci (donc, à la fin du corps).

Intégration de script: méthode 2 dans le <head> et/ou le <body>

```
<html>
  <head>
    <title>titre de page</title>
    <script type="text/javascript">
      // code ici
    </script>
  </head>
  <body>
    <script type="text/javascript">
      // code ici
    </script>
  </body>
</html>
```


Script dans le corps du texte

- **Exécution directe:** le code s'exécute automatiquement lors du chargement de la page HTML dans le navigateur.
- La squelette de la page HTML est :

```
<html>
```

```
<head>
```

```
  <title> ..... </title>
```

```
</head>
```

```
<body>
```

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

```
    // place du code JavaScript
```

```
  </script>
```

```
</body>
```

```
</html>
```

Script dans l'entête

- **Exécution différée:** le code est d'abord lu par le navigateur, stocké en mémoire, pour ne s'exécuter que sur demande express.
- La squelette de la page HTML est :

<html>

<head>

```
<title> ..... </title> <script type="text/javascript">  
  // place du code JavaScript  
</script>
```

</head>

<body>

```
<- - place du code événement - ->
```

</body>

</html>

Exemple 1: Script placé dans le corps du texte.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head><body>
    <!-- script pour le debut du chargement -->
    <script type="text/javascript">
        alert('Debut du chargement de la page');
    </script>
    <!-- ici se trouve le contenu de la page web -->
    <p>    Vous testez un script...<br />    Enjoy!!! </p>
    <!-- script pour la fin du chargement -->
    <script type="text/javascript">
        alert('Fin du chargement de la page');
    </script>
</body>
</html>
```

Placer le code JS dans un fichier séparé

- Comme pour le CSS, on peut très bien placer notre code dans un fichier indépendant. On dit que **le code est importé depuis un fichier externe**. L'extension du fichier externe contenant du code JavaScript est **.js**.
- On va indiquer aux balises le nom et l'emplacement du fichier contenant notre (ou nos) script(s), grâce à la propriété `src` (comme pour les images).
- Syntaxe :

`<head>...`

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

`...</head>`

Exemple

```
<html><head><title>exemples de déclenchements</title>
<script type="text/javascript">
    function saluer() {
    alert("bonjour !");
    }
</script></head><body>
<h1>exécution immédiate</h1>
<script type="text/javascript">
saluer();
</script>
<h1>exécution sur événement onclick</h1>
<form><input type="button" name="bouton" value="salut" onclick="saluer();" />
</form>
<h1>exécution sur protocole javascript:</h1>
<a href="javascript:saluer();">pour saluer</a></body></html>
```

Notations Javascript

■ Commentaire :

■ Sur une ligne : `// ... commentaire ...`

■ Sur plusieurs lignes : `/* ...
commentaire
... */`

■ Séparateur d'instructions :

■ Comme en C++/Java, les instructions se terminent par un `;` (point virgule) :

■ Un retour à la ligne est aussi interprété comme fin de l'instruction.

■ Groupement d'instructions

■ Accolades : `{ ... instructions ... }`

De quoi se compose le "code" ?

```
<html>

<head>
  <title>JavaScript Page</title>
</head>

<body>
  <script type="text/javascript">
    // silly code to demonstrate output

    document.write("<p>Hello world!</p>");

  </script>

  <p>Here is some static text as well.</p>

</body>
</html>
```

- **document.write** écrit du texte sur une page Web, Le texte à afficher peut inclure des balises HTML,
 - **\n** qui insère un retour à la ligne,
 - **\t** pour insérer une tabulation,
- Ces balises sont interprétées par le navigateurs lors de l'affichage du texte en question,

Déclaration de variables 1/2

- Pour définir une variable : ***var nomvariable=valeur***
 - Pas de typage explicite : détection automatique par l'interpréteur
 - Les types disponibles en JavaScript sont :
 - Toute variable non définie est de type **Undefined** et a pour valeur **undefined**
 - Null** : le type Null ne peut avoir qu'une valeur null
 - Boolean** : le type Boolean peut prendre deux valeurs **true** ou **false**
 - "falsey" values: 0, 0.0, NaN, "", null, and undefined
 - "truthy" values: anything else
 - String** : est du texte
 - Number** : est un nombre
 - Object** : est un objet
- ```
var ned = null; // ned existe, mais a pour valeur null
var caroline; //caroline a pour valeur undefined : n'existe pas encore
```

# Déclaration de variables 2/2

## ■ Nomenclature des variables :

- Nom de variable **sensible à la casse** (**myVar** et **myvar** sont 2 variables différentes).
- Les variables nommées ne peuvent pas contenir les espaces, ou commencer par tout nombre, et tous les symboles de ponctuation excepté le soulignage ( ) sont restreints.

## ■ La portée de variables

- Les variables déclarées ou initialisées en dehors d'un corps de fonction ont une portée globale, rendant lui accessible à tous les autres le rapport dans le même document.
- Les variables déclarées ou initialisées dans un corps de fonction a une portée locale, le rendant accessible seulement aux instructions dans le même corps de fonction.

# Déclaration de variables : exemple

<body>...

<script type="text/javascript">

*//First we will declare a few variables and assign values to them*

var myText = "Good day!";

var myNum = 5;

*//Note that myText is a string and myNum is numeric.*

*//Next we will display these variables to the user.*

document.write(myText);

*//To concatenate strings in JavaScript, use the '+' operator*

document.write("My favourite number is "+ myNum);

</script>...

# Portée des variables: exemple 1

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

```
 var altitude = 5;
```

```
 // VARIABLE GLOBAL (déclaration explicite)
```

```
 function square() {
```

```
 base = 17;
```

```
 //VARIABLE GLOBAL (déclaration implicite: sans le var)
```

```
 sqr = 0.5*(base + altitude);
 return sqr;
 }
```

Quand vous déclarez dans une fonction des variables sans le mot-clé **var**, alors ces variables sont globales.

```
 function perimeter() {
```

```
 var side = 7.5; //VARIABLE LOCAL
```

```
 prm = 2*side + base;
 return prm;
 }
```

Vous obtenez une variable locale par la déclaration de la variable avec **var** à l'intérieur d'une fonction

```
</script>
```

# Portée des variables: exemple 2

```
<SCRIPT type="text/javascript">
var a = 12; //variable globale
var b = 4;
function MultipliePar2(b) {
var a = b * 2; //variable locale
return a; }
document.write("Le double de ",b," est
 ",MultipliePar2(b));
//affichera le double de ,4, est ,8
document.write("La valeur de a est ",a);
//affichera la valeur de a est , 12
</SCRIPT>
```

# Portée des variables: exemple 3

```
<SCRIPT type="text/javascript">
var a = 12; //variable globale
var b = 4;
function MultipliePar2(b) {
var a = b * 2; // variable locale globale
return a; }
document.write("Le double de ",b," est
 ",MultipliePar2(b));
//affichera le double de ,4, est ,8
document.write("La valeur de a est ",a);
//affichera la valeur de a est , 12 8
</SCRIPT>
```

# Les types de données et les variables dans JS

📁 La conversion de types :

🌐 **Nombre --> chaîne:** par l'ajout de deux apostrophes

```
var count = 10;
var s1 = "" + count; // "10"
```

🌐 **Chaîne --> nombre:**

**parseInt(chaine\_a\_convertir)**  
**parseFloat(chaine\_a\_convertir)**

```
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("boo"); // NaN
```

```
<html>
<head>
 <title>Data Types and
 Variables</title>
</head>
<body>
 <script type="text/javascript">
 var x, y;

 x= 1024; y=x; x = "foobar";
 document.write("<p>y = " + y +
 "</p>");
 document.write("<p>x = " + x +
 "</p>");
 document.write ("type x :" +
 typeof(x));
 </script>
</body>
</html>
```



# Interactive Pages Using Prompt

## ■ Utilisation de la fonction prompt:

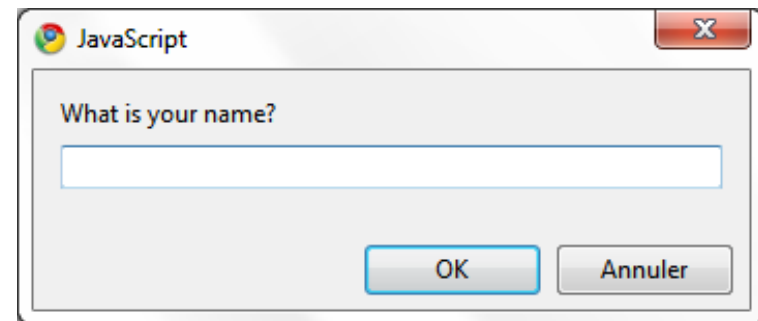
```
<html>
<head>
 <title>Interactive page</title>
</head>

<body>
 <script type="text/javascript">
 userName = prompt("What is your name?", "");

 userAge = prompt("Your age?", "");
 userAge = parseFloat(userAge);

 document.write("Hello " + userName + ".")
 if (userAge < 18) {
 document.write(" Do your parents know " +
 "you are online?");
 }
 else {
 document.write(" Welcome friend!");
 }
 </script>
</body></html>
```

- 1<sup>er</sup> argument: le message qui apparait dans la fenêtre de dialogue,
- 2<sup>ème</sup> argument: une valeur par défaut, si l'utilisateur n'insert rien,



# Exemple

```
<html>
```

```
<head> <title>Data Types conversion</title></head>
```

```
<body>
```

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

```
var nombre1 = prompt('Premier nombre ?');
```

```
var nombre2 = prompt('Deuxieme nombre ?');// on convertit en nombres décimaux
```

```
var resultat = parseFloat(nombre1) + parseFloat(nombre2);
```

```
alert("La somme de ces deux nombres est " + resultat);
```

```
var nombre1 = 8;
```

```
var nombre2 = 12;
```

```
alert(nombre1 + ' ' + nombre2);
```

```
</script></body></html>
```

# JavaScript Array

- **Array** est utilisé pour stocker une séquence d'éléments, accessible via un index. Comme JS est faiblement typée, les éléments d'un même tableau peuvent être de **types différents**.
- Pour créer un tableau, il faut allouer de l'espace en utilisant **new** ou en lui assignant des valeurs **directement**.
- Le premier élément du tableau est **indexé à 0**.

Exemples :

```
var items = new Array(10); // allocation d'espace pour 10 elts
var lItems = new Array(); // if no size given, will adjust dynamically
var lItems = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
```

# JavaScript Arrays

- Pour accéder a un élément du tableau : **[]** (comme C++/Java)

```
var items = new Array(10); // allocation d'espace pour 10 elts
```

```
for (i = 0; i < 10; i++)
{
 items[i] = 0; // stores 0 at each index
}
```

- **length** indique le nombre d'éléments d'un tableau.

```
for (i = 0; i < items.length; i++)
{ document.write(items[i] + "
"); // displays elements
}
```

# Tableaux associatifs

## ■ Principe

- L'indice est une chaîne de caractères,
- Exemple: Chargement d'une page HTML en fonction du jour de la semaine:

```
var semaine = new Array();
semaine["lundi"] = "cours.html";
semaine["dimanche"] = " weekend.html";
```

# Opérateurs de comparaison

- **Les opérateurs d'égalité** (fonctionnent aussi avec des chaînes de caractères)
  - **$a = b$**  si les deux valeurs sont égales, alors on a true, sinon false.
  - **$a \neq b$**  : si les deux valeurs sont différentes, alors on a true, sinon false.
- **Opérateurs de comparaison de valeurs numériques :**
  - **$a < b$**  : si **a** est inférieur à (plus petit que) **b**, alors on a true, sinon false.
  - **$a > b$**  : si **a** est supérieur à (plus grand que) **b**, alors on a true, sinon false.
  - **$a \leq b$**  : si **a** est inférieur ou égal à **b**, alors on a true, sinon false.
  - **$a \geq b$**  : si **a** est supérieur ou égal à **b**, alors on a true, sinon false.

# Incrémentation / décrémentation

- Il existe, de la même manière, les opérateurs suivants:
  - += (on retranche la valeur de la deuxième variable à celle de la première),
  - -= (on retranche la valeur de la deuxième variable à celle de la première),
  - \*= (on multiplie la valeur de la première variable par celle de la deuxième),
  - /= (idem mais avec une division) et %=.
- **Incrémentation / décrémentation**
- Lorsque l'on veut augmenter de 1 la valeur d'une variable on utilise la notation : `variable++`;
- De même, pour **décrémenter** (diminuer la valeur de 1) une variable, le code est le suivant : `variable--`;
- `variable += X`; équi. a `variable = variable + X`; de même pour `-=`, `/=`, `*=` et `%=`.

# Les opérateurs logiques.

- Il s'agit de l'opérateur ET et de l'opérateur OU.
- En JS, on les note **&& (ET)** et **|| (OU)**.

Exemple:

```
var taille = prompt('Combien mesurez-vous ? (en m)');
var poids = prompt('Combien pesez-vous ? (en kgs)');
costaud = (taille>=2 && poids>=90);
alert('Vous êtes costaud : ' + costaud);
```

- La négation : Symbole "NON" (noté !)

Exemple: `var mineur = !(age >= 18);`



# Conversion implicite dans les expressions

- Les opérateurs logiques `>` `<` `>=` `<=` `&&` `||` `!==` `!=` font une conversion automatique de types avec d'évaluer une expression:

```
5 < "7" //is true
```

```
42 == 42.0 //is true
```

```
alert("5.0" == 5) //affichera true
```

- `===` and `!==` sont plus stricts et vérifient le type des deux opérandes:

```
alert("5.0" === 5) //affichera false
```

# Les instructions conditionnelles

## Syntaxe :

```
if (condition)
 instructions
else
 instructions
```

```
switch (expression)
{
 case valeur1 : instructions
 break;
 case valeur2 : instructions
 break;

 ...
 default : instructions
 break;
}
```

- break; arrête une boucle
- continue; passe à l'itération suivante de la boucle

# If else

## Example:

```
<script type="text/javascript" >
 //If the time is less than 10, you will get a "Good morning"
 greeting.
 //Otherwise you will get a "Good day" greeting.
 var d = new Date() // créer un objet de type Date
 var time = d.getHours() //retourne l'heure actuelle
 if (time < 10)
 {
 document.write("Good morning!")
 }
 else
 {
 document.write("Good day!")
 }
</script>
```

# Les boucles

```
while (condition)
 instructions
```

```
initialisation;
while (condition)
{
 instructions
 incrémentation
}
```

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

```
for (initialisation; condition; incrémentation)
{
 instructions
}
```

```
<html>
 <body>
 <script type="text/javascript">
 var i=0
 for (i=0;i<=10;i++)
 {
 document.write("The number is " + i)
 document.write("
")
 }
 </script>
 </body> </html>
```



# Exemple : switch

```
<script type="text/javascript">
 var d = new Date()
 theDay = d.getDay()
 switch (theDay)
 {
 case 5:
 document.write("Finally Friday")
 break
 case 6:
 document.write("Super Saturday")
 break
 case 0:
 document.write("Sleepy Sunday")
 break
 default:
 document.write("I'm really looking forward to this weekend!")
 }
</script>
```

# Les fonctions

- Emplacement de la déclaration
  - Dans l'entête de la page,
  - Utilisation de la syntaxe : `function nom_fonction(param1, ...)`
- Le corps de la fonction est délimité par `{ ... }`
- Contenu :
  - déclaration des variables locales, propres à la fonction,
  - instructions réalisés par la fonctions,
  - instruction `return` pour renvoyer une valeur ou un objet ,
  - cette instruction n'est pas obligatoire vu qu'il y a des fonctions qui ne renvoie pas de valeur,
  - Les paramètres et la valeur de retour n'ont pas de type prédéfinis, contrairement a Java /C ++ car JS est faiblement typé.

# Appel de fonction

- Peut avoir lieu à n'importe quel endroit de la page :
  - dans d'autres fonctions, dans le corps de la page.
  - Utilisation de : `nom_fonction(param1, ... );`

```
<HTML>
 <HEAD>
 <SCRIPT type="text/javascript">
 // déclaration de fonction
 function bonjour(prenom)
 {
 document.write("Bonjour "+prenom+"
");
 }
 </SCRIPT>
 </HEAD>
 <BODY>
 <SCRIPT type="text/javascript">
 bonjour("Toto");
 </SCRIPT></BODY></HTML>
```

# User-Defined Functions

- Il est possible de limiter la portée d'une variable ,
- Si lors de la première utilisation d'une variable, elle est précédé par `var`, alors il s'agit de variable locale à la fonction.

```
function isPrime(n)
// Assumes: n > 0
// Returns: true if n is prime, else false
{ if (n < 2) {
 return false; }
 else if (n == 2) {
 return true;
 }
 else {
 for (var i = 2; i <= Math.sqrt(n); i++) {
 if (n % i == 0) {return false;}
 }
 return true;
 }
}
```

## L'objet Math:

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

méthodes:

abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan

Propriétés : E, PI

## Math.sqrt(n) :

Retourne la racine carrée du nombre réel passé en paramètre.



# Function Example

```
<html>
<head>
 <title>Prime Tester</title>

 <script type="text/javascript">
 function isPrime(n)
 // Assumes: n > 0
 // Returns: true if n is prime
 {
 // CODE AS SHOWN ON PREVIOUS SLIDE
 }
 </script>
</head>

<body>
 <script type="text/javascript">
 testNum = parseFloat(prompt("Enter a positive integer", "7"));

 if (isPrime(testNum)) {
 document.write(testNum + " is a prime number.");
 }
 else {
 document.write(testNum + " is not a prime number.");
 }
 </script></body></html>
```

■ La définition d'une fonction est ajoutée (généralement) dans l'entête **<head>**

**<head>** est chargé en mémoire en premier lieu, donc la fonction est définie avant l'exécution du code du **<body>**. Donc la fonction peut être utilisée dans le corps du texte.

# DOM+Javascript



# DOM

- Le HTML DOM définit le document de HTML comme **collection d'objet**. Les objets ont des méthodes et des propriétés.
- L'objet de **document** est le parent de tous autres objets dans des documents de HTML, qui est le fils de l'objet **window**.
- **La manipulation dynamiquement** des objets:
  - de l'interface (navigator, window, ...)
  - du document (body, form, table, ...)

**(objets créés automatiquement par le navigateur lors du chargement de mon document )**
- Comment ? DOM propose de représenter un document sous la forme d'un **arbre**. Toutes les balises HTML sont donc des **nœuds de l'arbre** et les feuilles sont soit des balises sans contenu, soit le texte de la page HTML.

# Notion d'objet prédéfini dans JS

## Objets prédéfinis du navigateur gérés par JS :

- **window** object
  - **navigator** object
  - **history** object
  - **location** object Etc.
- Dans une page Web, l'objet le plus élevé dans la hiérarchie est la fenêtre du navigateur : **window**.
- **alert()**, **confirm()** et **prompt()** sont des méthodes de **window**

**window**

**frames**

**navigator**

**history**

**location**

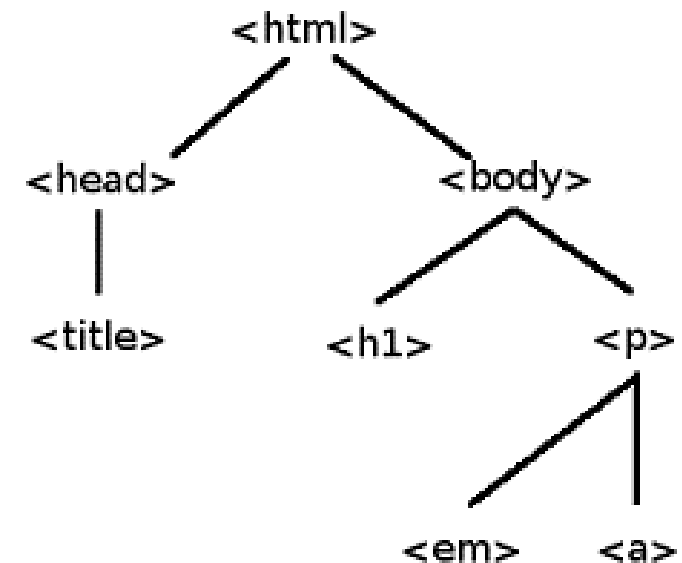
**document**

- forms
- elements
- images
- anchors
- links

# Notion d'arbre

- On peut schématiser une page HTML par un arbre, comme celui-ci:

```
<html>
 <head>
 <title>...</title>
 </head>
 <body>
 <h1>...</h1>
 <p>.........<a>...
 </p>
 </body>
</html>
```



- Pourquoi faire ?
- Lorsque l'on va manipuler ces pages avec du JavaScript, **il va falloir se balader dans notre page → comprendre comment la page est ordonnée devient alors très important.**
- Comment accéder aux différents objets???**

# Méthode 1: Accès direct a un élément

- L'objet **document**, sous objet de **window**, est le parent de tous autres objets dans un documents de HTML. Il faut passer par lui pour accéder à **tous** les autres.
- L'objet de **document.body** représente l'élément de `<body>` des documents de HTML.
- **Méthode 1 : accès direct**
- Comme il y a 1 seul objet body associé à 1 objet document, avec **document.body**

# Méthode 1: Accès direct a un élément- exemple

- Exemple de HTML DOM : comment la couleur de fond d'un document HTML peut être changée en jaune quand l'utilisateur clique là-dessus.?

```
<html><head>
<script type="text/javascript">
function ChangeColor(){
document.write("The background color is: ");
document.write(document.body.bgColor);
document.body.bgColor="yellow";}
</script>
</head>
<body bgcolor="red" onclick="ChangeColor()">
Click on this document!
</body></html>
```

- Comment accéder à un élément paragraphe, table, etc. d'un document HTML sachant qu'il peut y avoir 0 à +sieurs occurrences de cet élément dans un même document HTML?
- Méthode d'accès directe n'est plus valable!



# Méthode 2: id pour se positionner dans une page HTML

- La 2ème méthode pour retrouver un élément dans notre page:
  - Avec les **id** et les **class** que vous avez utilisé pour mettre en forme votre page, les id sur une page étaient uniques pour permettre l'identification (id) à coup sûr d'un élément dans une page.
- Les ids + DOM permet de « se promener » sur notre page.
- **Méthode 2 : se positionner dans un document grâce à un id :**

```
var elem= document.getElementById("mon_id");
```

- ATTENTION à l'écriture : get**E**lement**B**y**I**d !!!!!
- Nous créons une variable que l'on place à l'endroit voulu grâce à la méthode **getElementById** qui prend en argument l'**id**.

# id pour se positionner dans une page

## HTML Manipulation de CSS : exemple

```
<html>
 <head>
 <script type="text/javascript" >
 function Test()
 {document.getElementById("paragraphe").style.color = "blue" ;}
 </script>
 </head>
 <body>
 <p id="paragraphe" style="color:red">un texte</p>
 Test
 </body></html>
```

- Explication : L'exemple contient un paragraphe avec le nom **id paragraphe** et un lien que si on le clique, il appelle la fonction Test(). Cette fonction change la propriété CSS color du paragraphe, de telle sorte que le paragraphe perde sa couleur rouge et devienne bleu.

# Manipulation de style

**Syntaxe :** `document.getElementById("id").style.property="value"`

## Règles générales :

- On peut changer de cette façon n'importe quelle propriété CSS, d'un nœud ayant "id" pour identifiant
- Le nom de la propriété en JavaScript est identique au nom CSS, sauf que les traits d'unions sont remplacés par une majuscule sur la lettre suivante.
- Les valeurs des propriétés en JavaScript sont identiques aux valeurs CSS (mais doivent être mises entre guillemets).

## Exemple :

```
<html><head> <script type="text/javascript">
function setFont(){
document.getElementById("p1").style.fontFamily="arial,sans-serif";}
</script></head><body><p id="p1">This is an example paragraph.</p><form>
<input type="button" onclick="setFont()" value="Change font"
/></form></body></html>
```

# Méthode 3: name pour se positionner dans une page HTML

- De même il est possible de se positionner dans un document grâce à un **name** prédéfini:

Syntaxe : `window.document.getElementsByName("nom")`

- Attention : `getElementsByName` ( avec un "s" )**

Retourne un tableau (**Array**) d'objets HTML ayant "nom" défini comme valeur de l'attribut **name**

```
<form name="f">
<input name="fruit" type="checkbox" value="B" />Banane

<input name="fruit" type="checkbox" value="K" />Kiwi

<input name="fruit" type="checkbox" value="R" />Raisin

<input type="button"
onclick="var x=document.getElementsByName('fruit'); alert(x.length);"
value="how many elements named 'fruit'?"/>
</form>
</body>
</html>
```

# Methode 4: utilisation de `document.getElementsByTagName()`

- **Méthode 4** : permet de **récupérer toutes les balises identiques** à partir du document, ou d'un nœud inférieur (dans l'arbre).
  - Vous pouvez utiliser cette méthode pour compter le nombre de balises qu'il y a sur votre page, par exemple, ou pour naviguer dans votre document si vous connaissez bien sa structure.

- **Exemple :**

```
var titreList = document.getElementsByTagName("h1");
```

```
titreList.length; // nombre de h1 dans le document
```

```
titreList.item(2); // accéder à la 3ème balise h1 rencontrée
```

```
titreList[2]; // accéder à la 3ème balise h1 rencontrée
```

- Dans tous les cas, quand le nœud voulu n'existe pas, la méthode utilisée renvoie **undefined**. Vous avez donc désormais la possibilité de vous promener à volonté dans votre fichier XHTML.

# Utilisation de `document.getElementsByTagName()` Manipulation de CSS : exemple

■ Exemple: `document.getElementsByTagName("img");`

- Cet exemple va retourner un tableau contenant tous les images `<img>` de la page. Il est ainsi possible d'accéder à chacun des éléments. Si on veut accéder au second `<img>` de la page, on tapera:

`document.getElementsByTagName('img')[1];`

■ Propriétés :

- On retrouve presque toujours les mêmes attributs qu'en HTML.
- Exemple: Modifier l'adresse d'une image et ses dimensions :

```
monImage = document.getElementsByTagName("img")[1];
monImage.src = "banniere.jpg";
monImage.width = "800";
monImage.height = "200";
```

# Méthode 5 : utilisation des méthodes prédéfinies des objets

■ **Méthode 5 :** se positionner dans un document grâce à des méthodes qui vont nous permettre de parcourir les nœuds d'un document :

```
var node = document.getElementById("mon_id");
var parent = node.parentNode;
//place la variable parent sur le noeud parent de node

var childList = node.childNodes;
//récupère tous les enfants de node dans un tableau childNodesList

var child1 = node.firstChild;
//récupère le premier enfant de node

var childx = node.lastChild;
//récupère le dernier enfant de node

var frerePrec = node.previousSibling;
//récupère le frère précédent de node
var frereSuiv = node.nextSibling;
//récupère le frère suivant
```



# Ajouter du code XHTML dans une page (1/2)

- Pour ajouter des balises proprement, il faut tout construire brique par brique. Le principe est assez simple :
  - D'abord, on doit (1) récupérer toutes les briques ; ensuite (2) on fait des assemblages ; et enfin (3) on accroche le tout là où on veut.

## (1) pour récupérer nos briques:

```
var titre = document.createElement("h1");
//Ici on crée une balise de type h1.
//Si on voyait ce qu'on a crée, on aurait : <h1></h1>
```

```
titre.setAttribute("class", "Titre_rouge");
//Ici on ajoute un attribut à notre balise
```

```
var lien = document.createElement("a");
lien.setAttribute("href", "toto.html");
var texte = document.createTextNode("Titraillle");
//Ici on a créé un texte
```



# Ajouter du code XHTML dans une page (2/ 2)

(2) Pour assembler nos briques, et surtout mettre le tout dans notre page XHTML :

```
lien.appendChild(texte);
```

```
//La, on accroche en dernier enfant (donc premier,
//puisque c'est le seul) le text node créé juste au-dessus.
// On aura donc un lien de ce type :
//<lien url="toto.html">Titraillle</lien>
```

```
titre.appendChild(lien);
```

```
//là, on accroche notre lien à la balise h1
```

■ On a donc un groupe de balises virtuelles sous cette forme prêtes à être mises dans le XHTML :

```
<h1>Titraillle</h1>
```

# Méthode relationnelle d'ajout de balise

- Il existe 2 autres méthodes prédéfinies pour accrocher une balise à une autre, qui sont :

```
parent.insertBefore(child, referenceChild);
```

//Ici, on accroche notre balise child à la balise parent, et dans l'ordre, elle se retrouve juste avant la balise referenceChild

```
parent.replaceChild(newChild, oldChild);
```

//Ici, on remplace la balise enfant de parent oldChild par la balise newChild

# Manipulation de style

```
node.className="nouvelle_classe"
```

//pour changer le nom de la classe CSS à laquelle appartient le noeud.

```
node.style.borderStyle="valeur"
```

//pour changer le style de bordure d'un noeud.

## ■ Règles générales :

- On peut changer de cette façon n'importe quelle propriété CSS, d'un noeud.
- Le nom de la propriété en JavaScript est identique au nom CSS, sauf que les traits d'unions sont remplacés par une majuscule sur la lettre suivante.
- Les valeurs des propriétés en JavaScript sont identiques aux valeurs CSS (mais doivent être mis entre guillemets).

# DOM: objet form

- L'objet **form** est un sous-objet de **document**. Cet objet représente les formulaires de la page HTML.
- Propriétés
  - **name** : nom (unique) du formulaire
  - **method** : méthode de soumission (0=GET, 1=POST)
  - **action** : action déclenchée par la validation du formulaire
  - **target** : fenêtre de destination de la réponse (si elle existe)
  - **elements[ ]** : tableau des éléments du formulaires
  - **length** : nombre d'éléments du formulaire

# DOM: objet form

## ■ Méthodes

■ `submit()` : soumet le formulaire

■ `reset()` : réinitialise le formulaire

## ■ Événements

■ `onSubmit(method)` : action à réaliser lorsque le formulaire est soumis

■ `onReset(method)` : action à réaliser lorsque le formulaire est réinitialisé

# Méthodes et événements des éléments du Formulaire

## ■ Méthodes spécifiques aux formulaires

- **focus()** : donne le focus à cet élément (pour une zone de texte, place le curseur à l'intérieur)
- **blur()** : enlève le focus de cet élément (en quelque sorte le contraire de **focus**)
- **Click()** : simule un clic de souris sur cet élément
- **select()** : sélectionne ("surligne") le texte de ce champ

## ■ Évènements spécifiques aux formulaires

- **onFocus** : lorsque l'élément reçoit le focus (pour une zone de texte, quand on place le curseur à l'intérieur)
- **onBlur** : lorsque l'élément perd le focus (en quelque sorte le contraire de **onFocus**)
- **onChange** : lorsque la valeur / l'état de l'élément change (quand on coche la case, qu'on modifie le texte, etc.)
- **onSelect** : lorsqu'on sélectionne (quand on "surligne") le texte de ce champ

# DOM: Examples

## ■ Exemple de formulaire :

```
...<body>
 <form name="general">
 <input type="text" name="champ1" value="default">
 </form>
</body>...
```

## ■ Accès à l'objet correspondant au formulaire précédent: il y a trois possibilités :

```
document.forms["general"]
document.forms[0]
document.general
```

# DOM: Examples

## ■ Accès aux éléments du formulaire

■ 3 possibilités :

```
document.forms["general"].elements["champ1"]
```

```
document.forms["general"].elements[0]
```

```
document.forms["general"].champ1
```

## ■ Accès aux propriétés d'un élément

```
document.forms["general"].elements["champ1"].value
```



# Text Boxes (zones de texte)

- Une zone de texte permet une saisie de l'utilisateur (et pourrait aussi être utilisé pour l'affichage)
- Le code JavaScript peut accéder au contenu dans l'exemple ci-dessous comme : `document.BoxForm.userName.value`

```
...<body>
<form id="BoxForm" name="BoxForm">
<p> Enter your name here:
<input type="text" name="userName" id="userName" size="12" value="" />

<input type="button" value="Click Me"
onclick="alert('Thanks, ' + document.forms['BoxForm'].userName.value +
 ', I needed that.');" /></p>
</form>
</body>
```

# Read/Write Text Boxes

- De même, on peut changer le contenu d'une zone par affectation
- Note: le contenu est du texte brut, sans formatage HTML
- Les contenus sont accessibles comme une **chaîne (String)**, utiliser **parseFloat** ou **parseInt** pour convertir en nombre :

```
<body>
 <form id="BoxForm" name="BoxForm">
 <p> Enter a number here:
 <input type="text" size="12" name="number" id="number" value="2" />

 <input type="button" value="Double"
 onclick="document.forms['BoxForm'].number.value=
 parseFloat(document.forms['BoxForm'].number.value) * 2;" />
 </p>
 </form>
</body>
```

# Text Box Events

```
<html>
 <head>
 <title> Fun with Text Boxes </title>
 <script type="text/javascript">
 function FahrToCelsius(tempInFahr)
 // tempInFahr is a number (Fahrenheit)
 // Returns temperature in Celsius
 {
 return (5/9)*(tempInFahr - 32);
 }
 </script>
 </head>

 <body>
 <form id="BoxForm" id="BoxForm">
 <p> Temperature in Fahrenheit:
 <input type="text" name="Fahr" size="10" value="0"
 onchange="document.forms['BoxForm'].Celsius.value =
FahrToCelsius(parseFloat(document.forms['BoxForm'].Fahr.value));"/>
 <tt>----</tt>
 <input type="text" name="Celsius" size="10" value=""
 onfocus="blur();" />
 in Celsius </p>
 </form> </body></html>
```

■ **onchange** : Événement qui se déclenche lorsque le contrôle est modifié. Par exemple, lorsqu'on change le contenu d'un objet d'un formulaire et qu'on quitte cet objet .

■ **Onfocus** : Événement qui se déclenche lorsque l'élément reçoit le focus (devient actif) soit par action de l'outil de pointage (souris), soit par la navigation tabulée (touches du clavier).

■ **blur()** : La méthode javascript **blur()** d'un objet HTML qui permet de supprimer le focus clavier d'une balise HTML .

Utilité: ne pas permettre la modification du champs par une saisie.

# Text Box Validation

- Qu'arrive-t-il si jamais l'utilisateur entre un texte et pas un nombre dans la zone de texte Fahr de l'exemple précédent ?
- Solution: ajouter des instructions de validation
  - Initialiser la zone de texte avec une valeur valide ou le vide
  - Lorsque l'evt **onchange** est activé, vérifier que la nouvelle valeur est valide (sinon, reset)

# Text Box Validation : exemple

Verify.js

```
function VerifyNum(textBox) {
 // Assumes: textBox is a text box
 // Returns: true if textBox contains a number, else false + alert
 var boxValue = parseInt(textBox.value);
 if (isNaN(boxValue)) {
 // isNaN fonction predefinie de JS
 //Retourne true si boxValue n'est pas un nombre

 alert("You must enter a number value!");
 textBox.value = "";
 return false;
 }
 return true;
}
```

# Validation Example

```

<html><head><title> Fun with Text Boxes </title>
 <script type="text/javascript" src="verify.js">
 </script>
 <script type="text/javascript">
 function FahrToCelsius(tempInFahr)
 {
 return (5/9)*(tempInFahr - 32);
 }
 </script>
</head>
<body>
 <form id="BoxForm" name="BoxForm">
 <p> Temperature in Fahrenheit:
 <input type="text" name="Fahr" size="10" value="0"
 onchange="if (VerifyNum(this)) { // "this" refers to current element
 document.forms['BoxForm'].Celsius.value =
 FahrToCelsius(parseFloat(this.value));
 }" />
 <tt>---></tt>
 <input type="text" name="Celsius" size="10" value="" onfocus="blur();" />
 in Celsius </p>
 </form> </body></html>

```

# Text Areas : rappel

- Dans un formulaire, **TEXT** est limite a une seule ligne input/output
- **TEXTAREA** est similaire en fonctionnalité a TEXT, mais peut spécifier n'importe quel nombre de ligne ou de colonne.

```
<textarea name="TextAreaName" rows="NumRows" cols="NumCols">
```

Initialisation Text

```
</textarea>
```

■ *Note:* contrairement a TEXT, **TEXTAREA** a une balise fermante.

🌐 L'initialisation de TEXTAREA est mise entre la paire de balise.

■ Exactement comme pour l'élément TEXT, le texte de **TEXTAREA** ne peut être formaté en utilisant HTML.

# TEXTAREA Example

```
<html>
<head>
 <title> Fun with Textareas </title>
 <script type="text/javascript" src="verify.js"> </script>
 <script type="text/javascript">
 function Table(low, high, power){
 // Results: displays table of numbers between low & high, raised to power
 var message = "i: i^" + power + "\n-----\n";
 for (var i = low; i <= high; i++)
 {
 message = message + i + ": " + Math.pow(i, power) + "\n";
 }
 document.forms['AreaForm'].Output.value = message;
 }
 </script>
</head>
```



# TEXTAREA Example - suite

```

<body><form id="AreaForm">
 <div style="text-align: center;">
 <p> Show the numbers from <input type="text" name="lowRange" size="4"
 value="1" onchange="VerifyNum(this);" />
 to <input type="text" name="highRange" size="4"
 value="10" onchange="VerifyNum(this);" />
 raised to the power of <input type="text" name="power" size=3
 value=2 onchange="VerifyNum(this);" />

 <input type="button" value="Generate Table"
onclick="Table(parseInt(document.forms['AreaForm'].lowRange.value),
parseInt(document.forms['AreaForm'].highRange.value),
parseInt(document.forms['AreaForm'].power.value));" />

 <textarea name="Output" rows="20" cols="15">initialisation</textarea>
 </p></div></form></body></html>

```



# Check buttons (cases a cocher)

```
<html><head>
 <title> Check Boxes </title>
 <script type="text/javascript">
 function processCB(){
 var boxes = document.forms['BoxForm'].cb.length;
 var s="";
 for (var i = 0; i < boxes; i++)
 {
 if (document.forms['BoxForm'].cb[i].checked){
 s = s + document.forms['BoxForm'].cb[i].value + " ";
 }
 }
 if (s == "")
 { s = "nothing"; }
 alert("You selected " + s);
 }</script> </head>
```

# Check buttons (cases a cocher) la suite

```
<body>
 <form id="BoxForm">
 <p>Which of these things is unavoidable in life
 (select one or more)?

 <input type="checkbox" name="cb" value="Death"
 />Death

 <input type="checkbox" name="cb" value="Taxes"
 />Taxes

 <input type="checkbox" name="cb" value="Robbie"
 />Robbie Williams

 <input type="button" value="Done"
 onclick="processCB()" />
 </p>
</form></body></html>
```

# Check buttons (using a slightly different method)

```
<html><head> <title> Using checkboxes </title>
 <script type="text/javascript">
 function processCB()
 { var s="";
 var cb=document.forms['BoxForm'].elements['cb'];
 for (var i = 0; i < cb.length; i++)
 { if (cb[i].checked)
 { s = s + cb[i].value + " "; }
 }
 if (s == "")
 { s = "nothing"; }
 alert("You selected " + s); }
 </script>
</head>
<body>
 <form id="BoxForm">
 <p> Which of these things is unavoidable in life (select one or more)?

 <input type="checkbox" name="cb" value="Death" />Death
 <input type="checkbox" name="cb" value="Taxes" />Taxes
 <input type="checkbox" name="cb" value="Robbie Williams" />Robbie Williams

 <input type="button" value="Done" onclick="processCB()" /> </p>
 </form> </body> </html>
```

# DOM: objet Navigator

- L'objet **navigator** est un sous-objet de **window**. Cet objet donne des informations sur le navigateur utilisé
- Propriétés
  - **appName** : nom de code interne du navigateur
  - **appVersion** : nom réel du navigateur
  - **language** : version du navigateur
  - **platform** : langage du navigateur (fr, en) sous Navigator 4
  - **plugins** : système d'exploitation (MacPPC)ilisé
  - **plugins[]** : tableau des plugins installés chez le client
  - ...

# Notion d'objet prédéfini dans JS

- Cet objet **window** contient entre autres l'objet **document** qui lui même contient tous les objets contenus dans la page Web (paragraphes, formulaires, etc...).
- Si une page Web contient plusieurs cadres (frames), l'un objet **window** contiendra un tableau d'objet **frame**.
- L'objet **navigator** contient des informations concernant votre navigateur

```
<html><body>
<script type="text/javascript">
document.write("Name: " + navigator.appName);
//affichera Name: nom de votre navigateur exécutant ce script
</script>
</body></html>
```

# objet Document

- L'objet **document** est un sous-objet de **window**. Cet objet représente la page HTML affichée dans le navigateur.
- Propriétés
  - **forms[ ]** :tableau des formulaires de la page
  - **forms.length** :nombre de formulaire(s) de la page
  - **links[ ]** :tableau des liens de la page
  - **links.length** :nombre de lien(s) de la page
  - **anchors[ ]** : tableau des ancrs internes (<A NAME= ...>)
  - **anchors.length** :nombre de d'ancre(s) interne(s)
  - **images[ ]** :tableaux des images
- *Remarque : les tableaux contiennent les éléments dans l'ordre de leur apparition dans le code HTML!!!!*

# objet document

## ■ Propriétés

- **title** : titre du document
- **location** : URL du document
- **lastModified** : date de dernière modification
- **referrer** : URL de la page d'où arrive l'utilisateur
- **bgColor** : couleur de fond
- **fgColor** : couleur du texte
- **linkColor, vlinkColor, alinkColor** : couleurs utilisées pour les liens hypertextes



# DOM: Exemples

■ Exemple : donner le nombre de formulaire dans le document

```
<html>
```

```
<body>
```

```
<form name="Form1"></form>
```

```
<form name="Form2"></form>
```

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

```
<p>Number of forms:
```

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

```
document.write(document.forms.length);
```

```
//affichera number of forms:3
```

```
</script></p>
```

```
</body>
```

```
</html>
```

# DOM: objet document

## ■ Méthodes

- `write(string)` : écrit une chaîne dans le document
- `writeln(string)` : idem + caractère de fin de ligne
- `clear()` : efface le document
- `close()` : ferme le document

# document Object

- Internet Explorer, Firefox, Opera, etc. permettent l'accès à votre document HTML en utilisant l'objet document

```
<html>
<head>
 <title>Documentation page</title>
</head>

<body>
 <table width="100%">
 <tr>
 <td><i>
 <script type="text/javascript">
 document.write(document.URL);
 </script>
 </i></td>
 <td style="text-align: right;"><i>
 <script type="text/javascript">
 document.write(document.lastModified);
 </script>
 </i></td>
 </tr>
 </table>
</body>
</html>
```

**document.write(...)**

Affiche du text sur le page,

**document.URL**

Donne l'adresse URL de votre page HTML,

**document.lastModified**

Donne la date et l'heure de modification du code HTML de la page en cours,

# navigator Object

- navigator.appName : le nom du navigateur utilisé,
- navigator.appVersion : la version du navigateur,

```
<!-- MSIE.css -->

a {text-decoration:none;
 font-size:larger;
 color:red;
 font-family:Arial}
a:hover {color:blue}
```

```
<!-- Netscape.css -->

a {font-family:Arial;
 color:white;
 background-color:red}
```

```
<html>
<head>
 <title>Dynamic Style Page</title>

 <script type="text/javascript">
 if (navigator.appName == "Netscape") {
 document.write('<link rel=stylesheet ' +
 'type="text/css" href="Netscape.css">');
 }
 else {
 document.write('<link rel=stylesheet ' +
 'type="text/css" href="MSIE.css">');
 }
 </script>
</head>

<body>
Here is some text with a
link.
</body>
</html>
```

# objet Window

- L'objet **window** représente la fenêtre de votre navigateur. Ça va être un objet très utilisé, car il possède de nombreux sous-objets.
- Propriétés
  - **frames[ ]** : tableau de frames
  - **frames.length** : nombre de frames
  - **self** : fenêtre courante
  - **opener** : la fenêtre (si elle existe) qui a ouvert la fenêtre courante
  - **parent** : parent de la fenêtre courante, si la fenêtre courante est une sous-partie d'un frameset
  - **top** :fenêtre principale (qui a crée toutes les fenêtres)
  - **status** : message dans la barre de statut
  - **defaultstatus** : message par défaut de la barre de statut
  - **name** : nom de la fenêtre
  - ...

# objet Window

## ■ Méthodes

- `alert(string)` : ouvre une boîte de dialogue avec le message passé en paramètre
- `Confirm(string)` : ouvre une boîte de dialogue avec les boutons OK et cancel
- `prompt(string)` : affiche une fenêtre de saisie
- `resizeBy(l,h)` : pour rétrécir ou agrandir la fenêtre
- `resizeTo(l,h)` : largeur et hauteur de la fenêtre à agrandir ou réduire
- `scroll(int x, int y)` : positionnement aux coordonnées (x,y)
- `open(URL, string name, string options)` : ouvre une nouvelle fenêtre contenant le document identifié par l'URL
- `close()` : ferme la fenêtre
- ...

# Window : exemple

- Les boîtes d'alerte sont très bien pour l'affichage des messages court mais ne sont pas bien adapté pour l'affichage de beaucoup de texte, ou de texte nécessitant une mise en forme
- Elles sont pas intégrées dans la page, oblige l'utilisateur à fermer explicitement la boîte de dialogue.
- QUESTION: Peut-on utiliser à la place **document.write**?
- NON : on risque d'écraser la page en cours, y compris les éléments .
- Solution : Ouvrir une nouvelle fenêtre du navigateur et y écrire .

# Window : autres méthodes

Ouvrir une nouvelle fenêtre du navigateur et y écrire :

```
<script type="text/javascript">
var OutputWindow = window.open();
// open a window and assign a name to that object
```

```
OutputWindow.document.open();
// open that window for writing
```

```
OutputWindow.document.write("un texte");
// write text to that window as before
OutputWindow.document.close();
// close the window
</script>
```



# Window Example 2

```
<html>
<head>
 <title> Fun with Buttons </title>
 <script type="text/javascript">
 function Help()
 // Results: displays a help message in a separate window
 {
 var OutputWindow = window.open();
 OutputWindow.document.open();

 OutputWindow.document.write("This might be a context-" +
 "sensitive help message, depending on the " +
 "application and state of the page.");

 }
 </script>
 </head>
 <body>
 <form id="ButtonForm">
 <p> <input type="button" value="Click for Help"
 onclick="Help();" /> </p>
 </form>
 </body>
</html>
```

# Window Example Refined

```
<html>
<head>
 <title> Fun with Buttons </title>
 <script type="text/javascript">
 function Help()
 // Results: displays a help message in a separate window
 {
 var OutputWindow =
 window.open("", "", "status=0,menubar=0,height=200,width=200");
 OutputWindow.document.open();

 OutputWindow.document.write("This might be a context-" +
 "sensitive help message, depending on the " +
 "application and state of the page.");

 }
 </script>
</head>
<body>
 <form id="ButtonForm">
 <p> <input type="button" value="Click for Help"
 onclick="Help();" /> </p>
 </form>
</body></html>
```

Les arguments de `window.open` : 1<sup>st</sup> arg spécifie HREF , 2<sup>nd</sup> arg spécifie internal name, et 3<sup>rd</sup> arg spécifie window properties (e.g., size)

# Manipulation de window : exemple

```
<html><head>
 <script type="text/javascript">
 function OpenWindow() {
 Info = open("fichier.html", "secondefenetre") ;}
 </script></head>
 <body onload="OpenWindow()">
 <p>Fermer la fenetre</p>
 </body></html>
```

- Explication : L'exemple **ouvre à la lecture du fichier une deuxième fenêtre du nom de Info.**
- Dans le fichier est défini un lien. Si l'utilisateur clique sur le lien, la deuxième fenêtre est fermée.

# Manipulation objet document : Exemples

## Le nombre de lien hypertexte interne dans la page:

```
<html>
<body>

HTML Tutorial

CSS Tutorial

XML Tutorial

<a>SQL Tutorial

<p>Number of named anchors:
<script type="text/javascript">
document.write(document.anchors.length);
</script>
</p>

</body>
</html>
```

### Your Result:

HTML Tutorial  
CSS Tutorial  
XML Tutorial  
SQL Tutorial

Number of named anchors: 3

## Le nombre d'image dans la page:

```
<html>
<body>

<script type="text/javascript">
document.write("This document contains: " + document.images.length
+ " images.");
</script>

</body>
</html>
```

### Your Result:



This document contains: 2 images.

# JavaScript & Timeouts

- **setTimeout** : peut être utilisée pour définir une exécution différée d'un script.

`setTimeout(JavaScriptCodeToBeExecuted, MillisecondsUntilExecution)`

- Exemple : un lien vers la nouvelles page quand une page change d'emplacement
- Exemple :

```
<input type="button" value="click me"
onclick="setTimeout('window.alert(\'Hello!\')', 2000)" />
```

Une boite de message qui contient Hello sera afficher après 2 secondes

# Exemple (une redirection/va charger la page newhome.html après 3sec)

```
<html>
 <head>
<script type="text/javascript">
 function Move()
 // Results: sets the current page contents to be newhome.html
 {
 self.location.href = "newhome.html";
 }
</script>
</head>
<body onload="setTimeout('Move()', 3000);">
 <p> This page has moved to newhome.html. </p>
</body>
</html>
```

# Another Timeout Example

```
<html> <head>
<script type="text/javascript">
 function timeSince()
 // Assumes: document.forms['CountForm'].countdown exists in the page
 // Results: every second, recursively writes current countdown in the box
 {
 // CODE FOR DETERMINING NUMBER OF DAYS, HOURS, MINUTES, AND SECONDS
 // UNTIL GRADUATION (see the file for this code!!!)

 document.forms['CountForm'].countdown.value=
 days + " days, " + hours + " hours, " +
 minutes + " minutes, and " + secs + " seconds";

 setTimeout("timeSince();", 1000);
 }
</script>
</head>
<body onload="timeSince();">
 <form id="CountForm">
 <div style="text-align: center;">
 <p> Countdown to Graduation

 <textarea name="countdown" id="countdown" rows="4" cols="15"
 style="font-family: Courier;" onfocus="blur();"></textarea> </p>
 </div></form> </body></html>
```

# Validation formulaire HTML4/HTML5

- Les scripts de validation de formulaires sont à supprimer pour les formulaires en HTML5:
- Zones à saisir obligatoirement
- Zones qui prennent le focus au chargement de la page
- Zones de type Number, e-mail, date, etc.
- Comment faire actuellement alors que HTML5 n'est pas encore un standard ?



## ■ Programmer les formulaires avec HTML5

### ■ Vérifier si l'attribut est supporté avec :

```
function checkAttribute(element, attribute) {
 var test = document.createElement(element);
 if (attribute in test)
 { return true; }
 else { return false; }
}
```

### ■ Par exemple, un test pour l'attribut HTML5 placeholder :

```
if (!checkAttribute('input', 'placeholder')) {
 // No support for placeholders, so add them with JS
}
```