

Client-side Scripting



Préparé par :

Neila Ben Lakhal

phD@ Tokyo Institute of Technology Japan

E-mail : neila.benlakhal@gmail.com

Javascript?

- **JavaScript (JS)** is a lightweight interpreted programming language with first-class functions (functions are treated as variables: they can be passed as arguments to other functions, returned as the values from other functions, and assigned to variables etc.)
- While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, e.g., Adobe Acrobat.
- JavaScript supports object-oriented, and procedural programming.
- JavaScript® is a registered trademark of Oracle in the U.S. and other countries.

Caractéristiques de JS

- 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.
- JS est un **langage script orienté objets**, ce qui signifie que vous pouvez créer des classes et instancier des objets.
- JS propose en standard un certains nombres de classes que vous pourrez utiliser à votre aise pour créer vos programmes.

Histoire 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](#), C et C++.
- 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) (fully supported by all modern browsers)
 - ECMAScript 2018 (Draft)

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++)
 - Mozilla utilise SpiderMonkey [JavaScript](#) engine C/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)

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 tout 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)
- **extJS**: permet de construire des applications web interactives (initialement une extension à la bibliothèque Javascript YUI de Yahoo)
- **Angular.js** (old version), **angular** (By Google)
- **React.js** (By Facebook)
- **Node.js** (built on Chrome's V8 JavaScript engine)
- **Etc.**

Une liste exhaustive : <https://www.javascripting.com/>

Des variantes de Javascript

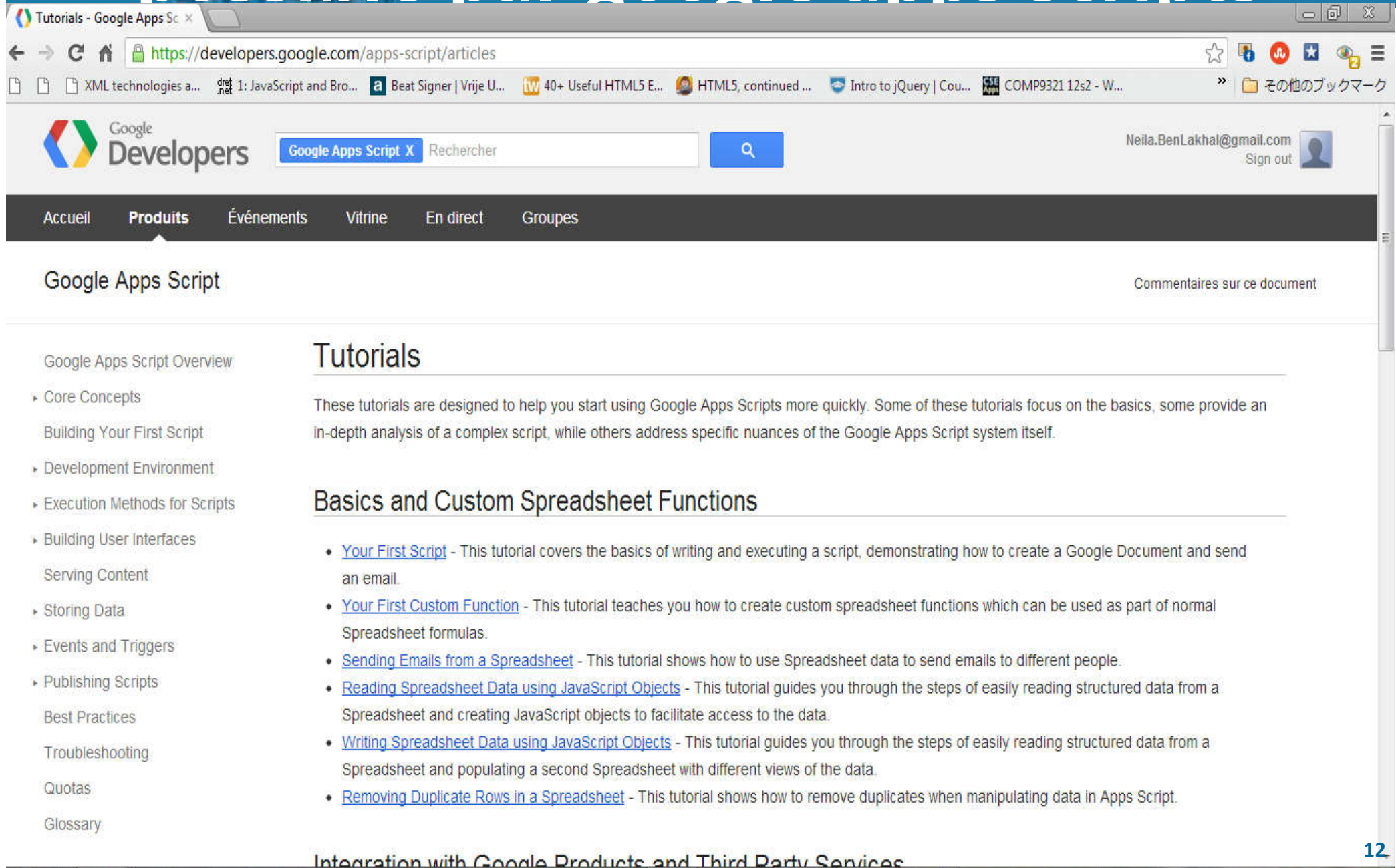
- Il y en a ceux qui ont décidé même de définir leurs propres langages de script pour développer des applications Web, par exemple :
 - Google Apps Script is a JavaScript cloud scripting language that lets you extend Google Apps and build web applications. Scripts are developed in Google Apps Script's 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. The browser address bar shows the URL: <https://script.google.com/macros/d/M6PMSXD83rA6XJ6Zd93oNhi32gawrAslk/edit?splash=yes>. The page title is "my first script". The menu bar includes File, Edit, View, Run, Publish, Resources, and Help. A yellow notification box states "Trying to reach google.com...". The toolbar contains icons for undo, redo, save, share, and a dropdown menu currently showing "createAndSendDocument". The left sidebar shows a file explorer with "Code.gs" selected. The main editor area displays the following JavaScript 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



The screenshot shows a web browser window displaying the Google Developers page for Google Apps Script. The browser's address bar shows the URL <https://developers.google.com/apps-script/articles>. The page header includes the Google Developers logo, a search bar with the text "Google Apps Script X", and a user profile for "Neila.BenLakhall@gmail.com" with a "Sign out" link. The main navigation bar contains links for "Accueil", "Produits", "Événements", "Vitrine", "En direct", and "Groupes". The page title is "Google Apps Script", and there is a link for "Commentaires sur ce document". The left sidebar lists various topics: "Google Apps Script Overview", "Core Concepts" (with sub-links for "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", and "Glossary"), "Tutorials", "Basics and Custom Spreadsheet Functions", and "Integration with Google Products and Third Party Services". The main content area is titled "Tutorials" and contains a paragraph: "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." Below this, the "Basics and Custom Spreadsheet Functions" section lists several tutorials:

- [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 :

```
<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.



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>
```

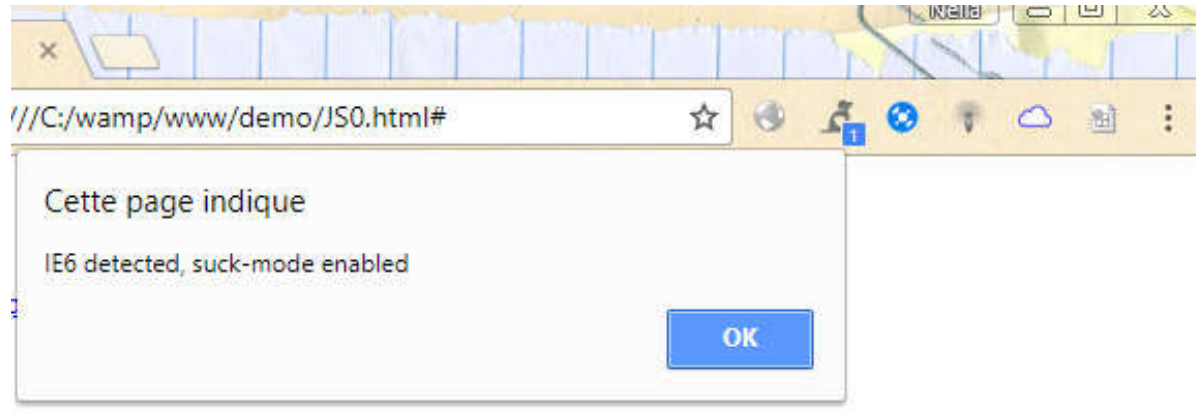
- Les 3 écritures sont équivalentes et permettent seulement d'ouvrir une boîte de dialogue :

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

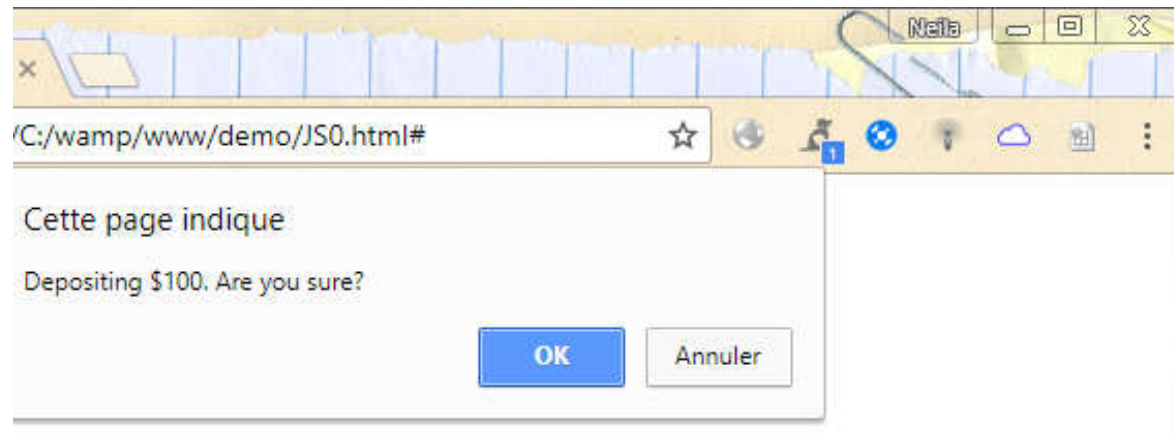
```
<a onClick="alert('hello! ')">Hello</a>
```

Les boites de dialogue

`Red Alert`

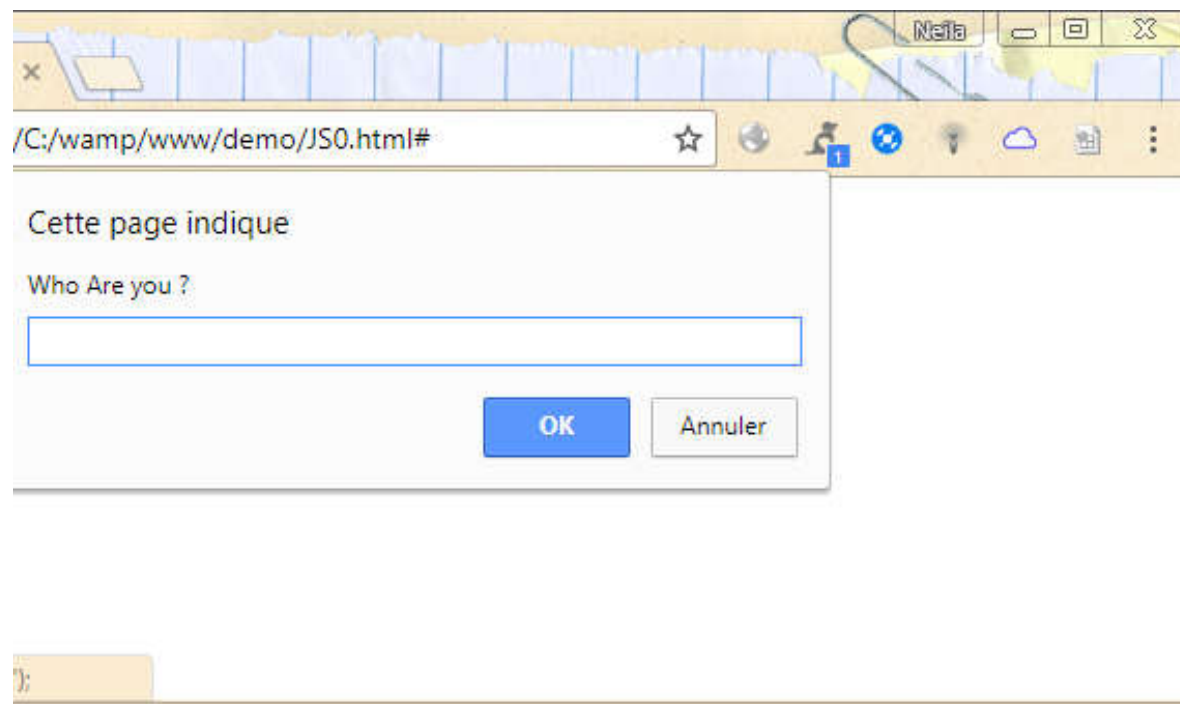


`Confirm`
//returns true or false



Les boites de dialogue

```
<a href="javascript:prompt('Who Are you ?');">Prompt</a>  
// 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,
 - 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" :

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

Inconvéniant méthode 1

- Si un même événement est à détecter dans plusieurs balises du même types, il y aura redondance!!
- Il est recommandé d'alléger la partie body du code JS.

Intégration de script: méthode 2

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

<script>

/ 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>

■ 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).

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<title>titre de page</title>
```

```
<script>
```

```
// code ici : exécution différée
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<script>
```

```
// code ici : exécution directe
```

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

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

```
  </script>
```

```
</head>
```

```
<body>
```

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

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

Exemple

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8">
<script>
function saluer() {
alert("bonjour");
}
</script>
</head><body>
<h1>exécution immédiate</h1>
<script>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>
```

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 src="monscript.js"></script>
...</head>

Exemple

```
<!DOCTYPE html>
<html><head><title>exemples de déclenchements</title>
<script src="monscript.js"></script>
</head><body>
<h1>exécution immédiate</h1>
<script>
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>
```

```
//monscript.js
```

```
function saluer() {
  alert("bonjour !");
}
```


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 ... }`

Déclaration de variables 1/2

- Les types disponibles en JavaScript sont :
 - **String** : est du texte
 - **Number** : est un nombre
 - **Object** : est un objet (Function, Array, Date et RegExp)
 - **Undefined** : 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

Déclaration de variables 2/3

- Pour définir une variable : `var|let nomvariable=valeur;`
- Pas de typage explicite : détection automatique par l'interpréteur

```
var nulle= null;
```

```
// nulle existe, mais a pour valeur null
```

```
var caroline;
```

```
//caroline a pour valeur undefined :n'existe pas
```

```
var age = 9; // Number
```

```
var weight = 27.4; // Number
```

```
var Name = "Sarah"; //String
```

```
var name='autrechaine'; //String
```

```
console.log(typeof(caroline)); //undefined
```

Type String

```
var x=10.1;
console.log(typeof(x));// Number
var Name= "Sarah BEN";
var fName = Name.substring(0, Name.indexOf(" "));
console.log(fName);// "Sarah"
var taille= Name.length;
console.log(taille);// 9
'hello'.length; // 5
```

Methodes: [charAt](#), [charCodeAt](#), [fromCharCode](#), [indexOf](#), [lastIndexOf](#),
[replace](#), [split](#), [substring](#), [toLowerCase](#), [toUpperCase](#)

Length : property

Concatenation with + : 1 + 1 is 2, but "1" + 1 is "11"

Type Boolean

```
var iLikeJS = true;
var ieIsGood = "IE6" > 0;
console.log(ieIsGood); // false
if ("web dev is great")
{ /* true */ }
if (0) { /* false */ }
```

- Any value can be used as a Boolean
 - "falsey" values: 0, 0.0, NaN, "", null, and undefined
 - "truthy" values: anything else
- Converting a value into a Boolean explicitly:
var boolValue = Boolean(*otherValue*);

Déclaration de variables 2/3

■ 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>
```

```
//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.
```

```
</script>
```

```
<h1>Any other code..</h1>
```

```
<script>
```

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

```
document.write(myText);
```

```
//document.write writes text in the web page,
```

```
//The text can be formatted with HTML tags
```

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

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

```
</script>...
```


Portée des variables: exemple 1

```
<script>
  var altitude = 5; // VARIABLE GLOBAL
function declareVar(){
  myVar=17; // VARIABLE GLOBALE
  var my2Var=15; // VARIABLE LOCALE
  console.log(my2Var); // affiche 15
}
```

Quand vous déclarez dans une fonction des variables sans le mot-clé `var`, alors ces variables sont **globales**: affectation de valeur sans déclaration de la variable.

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

```
declareVar(); // affichera 15
console.log (altitude) // affichera 5
console.log(myVar); // affichera 17
console.log(my2Var); // affichera une erreur
</script>
```

Portée des variables: exemple 2

```
<script>
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>
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>
```

Variables de bloc

```
let a;
```

```
let name = 'Sarah';
```

■ **let** allows you to declare block-level variables.

■ The declared variable is available from the function block it is enclosed in.

Example

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<script>
function letVariable(){
// name *is* visible here

let name="sarah as bloc variable"; // the same as if declared with var
// to make it visible implicate declaration
document.write(name+"<br/>");
}
</script>
</head><body>
<script>
letVariable();
// name *is* not visible out here
document.write(name+"<br/>");
</script></body>
```

Exemple de variable de bloc

```
for (let myVariable = 0; myVariable < 5;  
myVariable++) {  
  // myVariable is visible here  
  document.write(myVariable); //01234  
}  
document.write(myVariable);  
// myVariable *is* not visible out here  
// Error Undefined Variable
```

Exemple let, var et implicite

```
let x=10;
console.log(x);// Affichera 10 considéré comme Globale
function local(){
let Y=12; console.log(Y);
} local();//Affichera 12
console.log(Y); // Erreur Considéré comme Locale
function Notglobal(){
var Z=13;
console.log(Z);}
Notglobal();// Affichera 13
console.log(Z);//Erreur
function global(){
W=14; console.log(W);
}
global();//Affichera 14
console.log(W);//Affichera 14
```

La conversion de types

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

```
var count = 10; var s1 = "" + count;  
console.log(typeof(s1)) // affichera string
```

Chaîne --> nombre:

```
parseInt(chaine_a_convertir)  
parseFloat(chaine_a_convertir)
```

```
x= '1024'; y=parseInt(x);  
console.log("type of x :" + typeof(x));  
//affichera string  
console.log("type of y :" + typeof(y));  
//affichera number  
console.log (x+y); //affichera 10241024  
console.log(x-y); //affichera 0
```


Data Types conversion:

parseInt | parseFloat

```
<!DOCTYPE html>
<html><head>
  <title>Data Type conversion</title>
</head>
<body>
  <script>
    userName = prompt("What is your name?", "");

    userAge = prompt("Your age?", "");
    //userAge = parseInt(userAge);

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

Data Types conversion:

parseInt | parseFloat

```
<!DOCTYPE html>
```

```
<html>
```

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

```
<body>
```

```
<script>
```

```
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);
```

```
</script>
```

```
</body>
```

```
</html>
```

Déclaration de constante

■ **const** allows you to declare variables whose values are never intended to change. The variable is available from the function block it is declared in.

```
const Pi = 3.14; // variable Pi is set
```

```
Pi = 1; // Error because you cannot change a constant.
```

■ **Constant** are also case sensitive, like variables.

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 Items = new Array(); // if no size given, will adjust dynamically
var IItems = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
var divers = ['t', 1978, [0, 1, 2]];
```

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] + "<br/>");  
// displays elements  
}
```

JavaScript Arrays

```
var myList = [ , 'JS' , , 'PHP' ];
```

```
console.log(myList);
```

```
/*affichera :
```

```
(4) [empty, "JS", empty, "PHP"]
```

```
  1: "JS"
```

```
  3: "PHP"
```

```
length: 4 */
```

```
console.log(myList[2]); //affichera undefined
```

```
myList[5] = 'html';
```

```
console.log(myList[5]); // 'html'
```

```
console.log(Object.keys(myList)); // ["1", "3", "5"]
```

```
console.log(myList.length); // 6
```

Increase/decrease Array Size

```
myList.length = 10; // allocation dynamique  
console.log(Object.keys(myList));  
// ["1", "3", "5"]  
console.log(myList.length); // 10
```

```
myList.length = 2; // suppression d'éléments  
console.log(Object.keys(myList)); // ['1']  
console.log(myList.length); // 2
```


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";
```

Accès à la clé et au contenu

```
<script>
var myArray = new Array();
myArray['one'] = 1;
myArray['two'] = 2;
myArray['three'] = 3; // show the values stored
for (var i in myArray) {
  console.log('key is: ' + i + ', value is: ' +
myArray[i]);
}
</script>
```



Les methodes du tableau

■ Methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift

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" (on le note !)

Exemple:

```
mineur = !(age >= 18);
```

Conversion implicite dans les expressions

- Les opérateurs logiques `>` `<` `>=` `<=` `&&` `||` `!==` `!=` font une **conversion automatique** de types avant 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" === 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

```
<script>
```

```
//If the time is less than 10, you will get "Good morning"  
//Otherwise you will get a "Good day"
```

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

</body>
</html>
```

Exemple : switch

```
<script>
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.
- On peut appeler une fonction avant sa déclaration.
- Utilisation de : `nom_fonction(param1, ...);`

```
<HTML>
```

```
<HEAD>
```

```
<SCRIPT>
```

```
// déclaration de fonction
```

```
function bonjour(prenom) {  
    document.write("Bonjour "+prenom+"<br/>");  
}
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY>
```

```
<SCRIPT>
```

```
// appel de la fonction
```

```
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:

```
three = Math.floor(Math.PI);
```

méthodes:

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

Propriétés : E, PI

Exemple:

Math.sqrt(n) : Retourne la racine carrée du nombre réel passé en paramètre.

Function Example

```
<html><head><title>Prime Tester</title>
<script>
function isPrime(n) {// CODE PREVIOUS SLIDE }
</script></head>
<body>
<script>
testNum = parseInt(prompt("Enter a positive integer"));
if (isPrime(testNum)) {
document.write(testNum + " <b>is</b> a prime number.");
}
else
{
document.write(testNum + " <b>is not</b> a prime number.");
}
</script>
</body></html>
```

Anonymous functions

■ Une fonction anonyme est une fonction dont le nom a été omis vu qu'elle va pas être appelé plusieurs fois, c'est un bloque de code simple et qui sera exécuté toute de suite, elle va pas être réutiliser.

■ Exemple :

```
<a id="mylink" href="#">link</a>
```

```
<script>
```

```
mylink.onclick = function() {
```

```
  alert('hello');
```

```
  // I can put as much code
```

```
  // inside here as I want
```

```
}
```

```
</script>
```

IIFE (Immediately Invokable Function Expression)

- Quand une fonction est appelée une et une seule fois une forme pratique est IIFE.
- Déclaration et appel de la fonction en une même instruction SANS attribuer un nom à la fonction.

Exemple :

```
(function () {  
  var x = "Hi!!";  
  alert(x);  
})  
(); // I will invoke myself
```

Fonction anonyme /expression

- Une fonction JS peut être stocker dans une variable :

```
var expression=function (a,b) { return a*b;}  
console.log(expression); //affiche Le code de la fonction  
console.log(expression(5,10)); // affichera 50
```

- Une fonction JS peut être utiliser dans une expression :

```
function somme (a,b) { return a*b;}  
var z=somme(10,20)*10;  
console.log(z);//affichera 2000
```

Fonction dans une fonction

■ Une fonction peut être définie dans le corps d'une d'autre :

■ Exemple:

```
function myBigFunction()
```

```
{ myValue=10;
```

```
  subFunction1();
```

```
}
```

```
function subFunction1() { console.log(myValue); }
```

```
myBigFunction();//affichera 10
```

```
subFunction1();//affichera 10
```

Les objets

- “An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.”
- En JS, vous pouvez créer vos propres objets :
- Il y a plusieurs manières de créer un objet :

- **Méthode 1:** La notation JS:

```
var person = {};
```

- **Méthode 2:** Le constructeur Object() (besoin de plusieurs instances):

```
var person1 = new Object();
```

- **Méthode 3:** Une fonction constructeur :

```
function Person(name) {...};
```

```
var person1 = new Person('Bob');
```

Exemple d'objets : méthode 1

```
var person = {};  
var person = {  
  name: ['Bob', 'Smith'],  
  age: 32, gender: 'male',  
  greeting: function() { alert('Hi! I\'m ' + this.name[0]  
+ '.'); } };  
console.log(person.name[0])//Bob  
person.age//32  
person.greeting()
```

Exemple d'objets : méthode 2

```
var person1 = new Object();  
person1.name = 'Chris';  
person1['age'] = 38;  
person1.greeting = function() { alert('Hi! I\'m '  
+ this.name + '.'); };
```

```
var person2 = Object.create(person1);  
/*person2 aura les mêmes propriétés et valeurs que  
person1*/  
console.log(person2.name) // affichera Chris  
person2.greeting()
```


Exemple d'objets : méthode 3

```
function Person(name) {  
  this.name = name;  
  this.greeting = function() { alert('Hi! I\'m ' +  
    this.name + '.'); };  
}  
  
var person1 = new Person('Bob');  
var person2 = new Person('Sarah');  
Console.log(person1)  
  
/* affichera :  
Person {name: "Bob", greeting: f}  
  greeting:f ()  
  name:"Bob" */
```

Exercice

■ On considère la fonction constructeur d'objet suivante :

```
function Person(first, last, age, gender, interests)
{ this.name = { 'first': first, 'last' : last };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
  this.bio = function()
  { alert(this.name.first + ' ' + this.name.last + ' is ' +
    this.age + ' years old. He likes ' + this.interests[0] + '
    and ' + this.interests[1] + '.'); };
  this.greeting = function() { alert('Hi! I\'m ' +
    this.name.first + '.'); };
}
```

Exercice

- Création d'une instance :

```
var person1 = new Person('Bob', 'Smith', 32,  
    'male', ['music', 'skiing']);
```

- On souhaite personnaliser les messages de la fonction bio() et greeting() pour qu'ils s'adaptent en fonction du genre et du nombre d'activités favorites de la personne.

Les APIs JAVASCRIPTS



Web APIs

- JS disposent de plusieurs API qui permettent d'enrichir vos codes avec diverses fonctionnalités :
- **APIs pour manipuler les documents chargés dans le navigateur:** [DOM \(Document Object Model\) API](#)
- APIs pour ramener un contenu du serveur : [AJAX](#)
- APIs HTML5:
 - API Canvas pour Dessiner
 - API web storage (session/cookies)
- APIs externe (third party) : API Google, Facebook, Twitter etc.

<https://www.programmableweb.com/category/all/apis>

<https://developer.mozilla.org/en-US/docs/Web/API>

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
- DOM4 (Recommandation 2015)

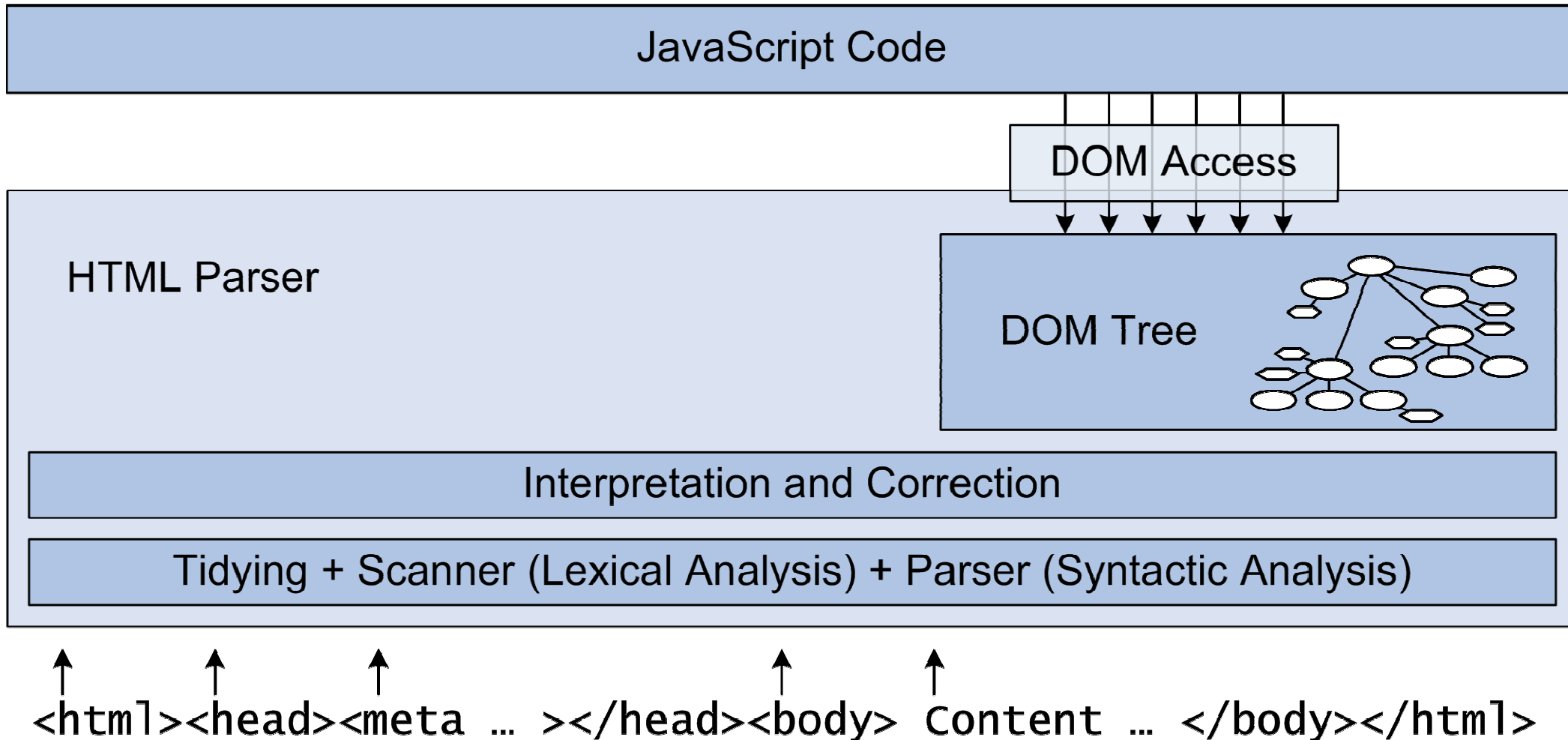
■ DOM comprend deux parties :

- 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
- 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.

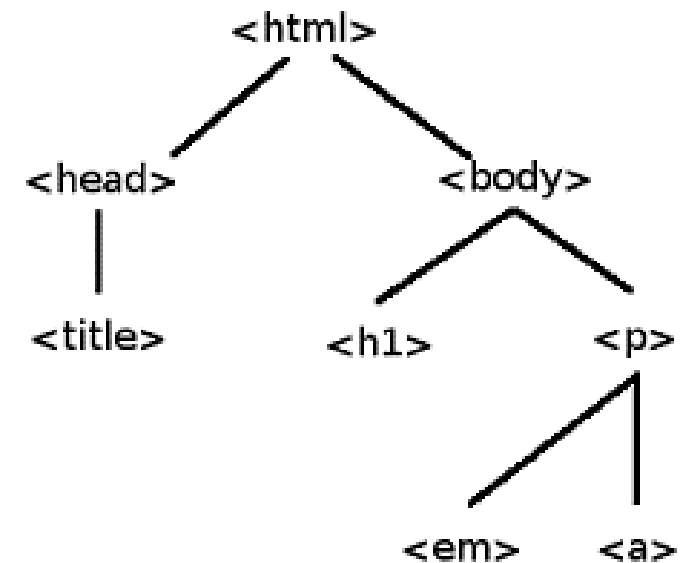
Fonctionnement interne du navigateur



Notion d'arbre

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

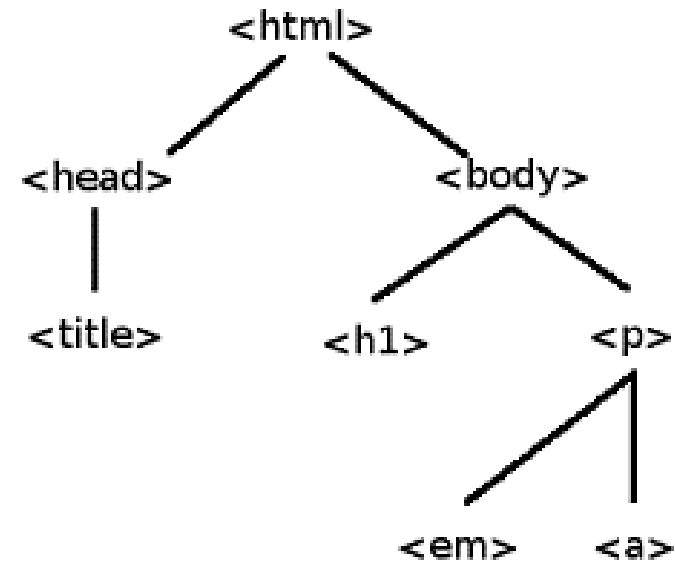
```
<html>
  <head>
    <title>...</title>
  </head>
  <body>
    <h1>...</h1>
    <p>...<em>...</em>...<a>...</a>
  </p>
  </body>
</html>
```



- 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 HTML que vous connaissez.
 - **texte** : ces nœuds sont en fait du texte brut qui se trouve entre les balises.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<!--je suis un commentaire-->
<title>Bienvenue!!</title>
</head>
<body>
<a href="http://www.w3schools.com/">
ceci est un lien vers le site W3C</a>
<h1>Ceci est un titre</h1>
<p>Ceci est un paragraphe.</p>
<b>Ce texte est en gras</b><br/>
<i>Ce texte est en italique</i><br/>
</body>
</html>

```

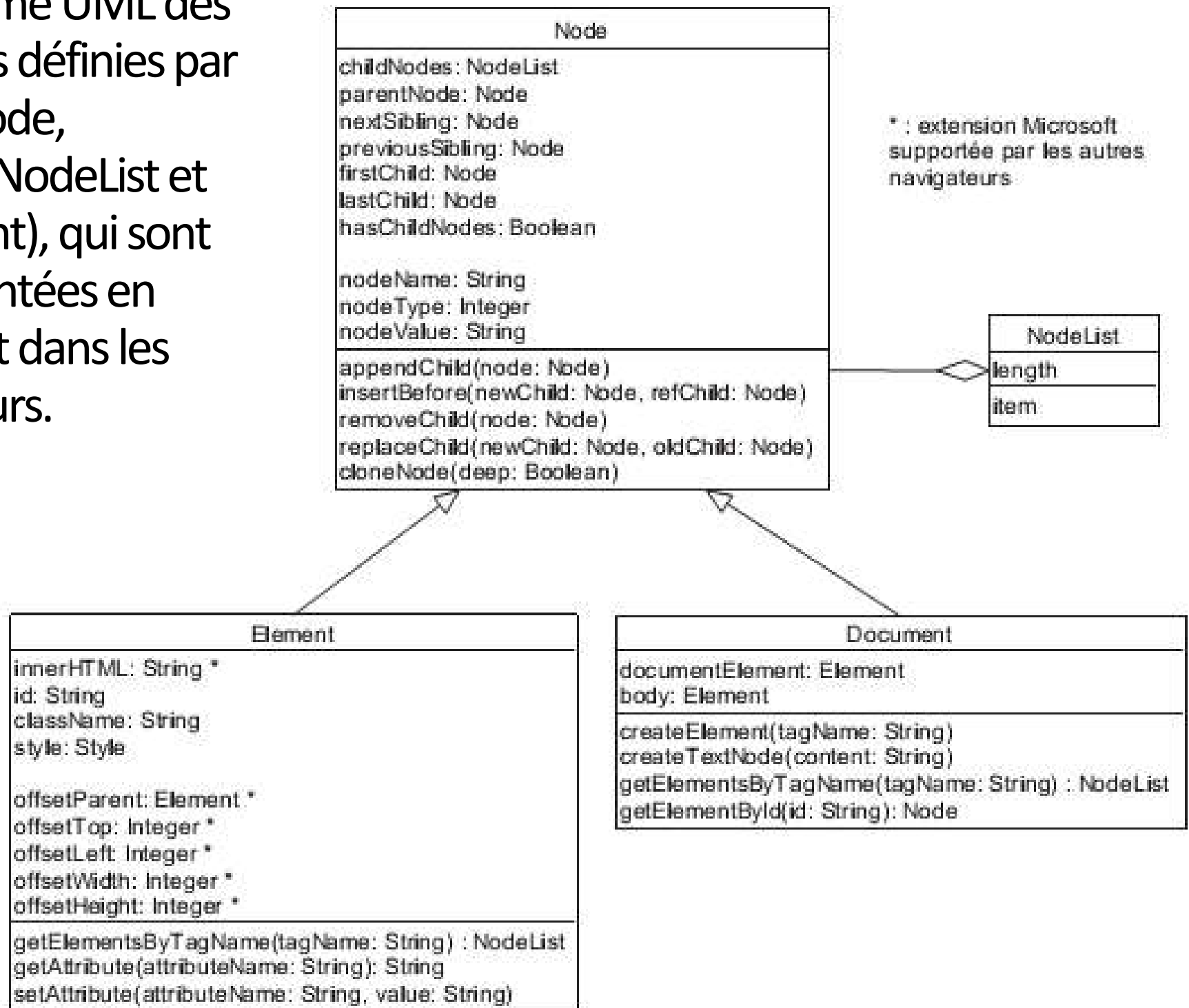


This DOM tree diagram was created
 using Ian Hickson's [Live DOM viewer](https://software.hixie.ch/utilities/js/live-dom-viewer/)
 (https://software.hixie.ch/utilities/js/live-dom-viewer/)

Types de noeuds (node)

- **Element node:** An element, as it exists in the DOM.
- **Root node:** The top node in the tree, which in the case of HTML is always the HTML node (other markup vocabularies like XML will have different root elements).
- **Child node:** A node directly inside another node. For example, A is a child of BODY in the above example.
- **Descendant node:** A node anywhere inside another node.
- **Parent node:** A node which has another node inside it. For example, BODY is the parent node of A in the above example.
- **Sibling nodes:** Nodes that sit on the same level in the DOM tree. For example, H1 and P are siblings in the above example.
- **Text node:** A node containing a text string.

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



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

navigator

history

location

document

- forms
- elements
- images
- anchors
- links

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>
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>
```

Méthode 2

- 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>
      function Test()
      {document.getElementById("paragraphe").style.color = "blue" ;}
    </script>
  </head>
  <body>
    <p id="paragraphe" style="color:red">un texte</p>
    <a href="#" onclick="Test()">Test</a>
  </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.

Recommendation

```
document.getElementById("paragraphe").style.color = "blue"
```

- Cette écriture n'est pas vraiment recommandée !
- Essayez de laisser la modification de style avec du CSS !!

Correction :

- Au lieu de :

```
document.getElementById("paragraphe").style.color = "blue" ;
```

- Optez pour :

```
document.getElementById("paragraphe").className = "enBleu";
```

- Et déclarer au niveau de vos style la classe « enBleu » :

```
.enBleu {color:enBleu}
```

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>
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<br />
<input name="fruit" type="checkbox" value="K" />Kiwi<br />
<input name="fruit" type="checkbox" value="R" />Raisin <br /><br />
<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 HTML.

Utilisation de `document.getElementsByTagName()`

Manipulation de CSS : exemple

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

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

Methode 6 : querySelector

- Returns the first Element within the document that matches the specified group of selectors.

```
element = document.querySelector(selectors);  
/*selectors is a string containing one or more CSS  
selectors separated by commas.*/
```

Example :

```
var e1 = document.querySelector(".myclass");  
//returns the first element in the document with the class  
"myclass"
```

Methode 7 : querySelectorAll

- Returns an Array of Elements within the document that matches the specified group of selectors.

```
element = document.querySelector(selectors);  
/*selectors is a string containing one or more CSS  
selectors separated by commas.*/
```

Example :

```
var e1 = document.querySelector(".myclass");  
//returns an array of elements in the document with the  
//class "myclass"
```

Exemple

*C:\Users\NBL\workspaceNeon\TP\WebContent\QuerySelector.html - Notepad

Fichier Édition Recherche Affichage Encodage Langage Paramétrage Macro Exécution TextFX Compléments Documents ?

lesClients.php clientChangeComputerUnit.php Page92.html htmlexp.html expJavaScript.html QuerySelector.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style type="text/CSS">
5     .myClass {
6         color: red;
7     }
8 </style>
9 </head>
10 <body>
11     <p class="myClass">This is the first paragraph</p>
12     <p class="myClass">This is the second paragraph</p>
13     <p>You get the idea...</p>
14     <h1>some other code..</h1>
15     <script type="text/javascript">
16         allParas = document.querySelectorAll(".myClass");
17         for (var i = 0; i < allParas.length; i++) {
18             allParas[i].style.color = "yellow";
19         }
20         var Para = document.querySelector("p").style.backgroundColor = "red";
21     </script>
22 </body>
23 </html>
```

Méthode 8 : utilisation des méthodes prédéfinies des objets

■ **Méthode 8 :** 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 HTML :

```
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">Titraille</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><a href="toto.html">Titraille</a></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).

exercice

- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side web APIs/Manipulating documents](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents)
[\(shopping list\)](#)
- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building blocks/Image gallery](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Image_gallery)

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="" />
<br />
<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" />
    <br /><br />
    <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>
      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));"/>
      &nbsp; <tt>----</tt> &nbsp;
      <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));
      }" />
    &nbsp; <tt>---&gt;</tt> &nbsp;
    <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 src="verify.js"> </script>
  <script>
    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);" />
    <br/><br/>
    <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));" /><br/><br/>
    <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>
    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)?<br/><br/>
    <input type="checkbox" name="cb" value="Death"
      />Death<br/>
    <input type="checkbox" name="cb" value="Taxes"
      />Taxes<br/>
    <input type="checkbox" name="cb" value="Robbie"
      />Robbie Williams<br/>
    <br/> <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>
    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)?<br/><br/>
        &nbsp; <input type="checkbox" name="cb" value="Death" />Death<br/> &nbsp; &nbsp;
        <input type="checkbox" name="cb" value="Taxes" />Taxes<br/> &nbsp; &nbsp;
        <input type="checkbox" name="cb" value="failure" />failure<br/> <br/>
        <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 ancres internes ()
 - **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: Examples

■ 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>
          document.write(document.URL);
        </script>
      </i></td>
      <td style="text-align: right;"><i>
        <script>
          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>
    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
<a href="javascript:alert('GO AWAY')">link</a>.
</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
 - **self** : fenêtre courante
 - **opener** : la fenêtre (si elle existe) qui a ouvert la fenêtre courante
 - **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 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>
```

```
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>
    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>
    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` : 1st arg spécifie HREF , 2nd arg spécifie internal name, et 3rd 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><a href="" onclick="Info.close()">Fermer la fenetre</a></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>

<a name="html">HTML Tutorial</a><br />
<a name="css">CSS Tutorial</a><br />
<a name="xml">XML Tutorial</a><br />
<a>SQL Tutorial</a>

<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>

<br />

<br /><br />

<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>
  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 <a href="newhome.html">newhome.html</a>. </p>
</body>
</html>
```

Another Timeout Example

```
<html> <head>
<script>
  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 <br />
      <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 ?

Validation formulaire HTML4/HTML5

- 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  
}
```