

INTRODUCTION JAVASCRIPT CÔTÉ CLIENT

Ahmed Lounis

- JavaScript
 - ▣ Aspects généraux
 - ▣ Les bases de la programmation en JavaScript
- JavaScript au service du web
 - ▣ DHTML : DOM versus JavaScript
 - ▣ JavaScript : un langage orienté événements
 - Gestionnaire d'événements
 - ▣ JavaScript et CSS
- AJAX
- JavaScript et la programmation orientée objet



- ❑ **JavaScript** est un langage de programmation de scripts (interprété) qui peut être utilisé dans les pages web interactives, côté client.
- ❑ Il est intégré aux pages HTML.

- **JavaScript** (*langage de scripting objet*) est un langage de programmation de scripts qui peut être utilisé dans les pages web interactives.
- Le langage a été créé en 1995 par Brendan Eich (appelé à l'époque livescript) pour le compte de Netscape.
 - ▣ Ensuite l'alliance Netscape-Java a donné lieu à Javascript.
- La syntaxe de base est intentionnellement similaire à Java et C pour réduire le nombre de nouveaux concepts nécessaires pour apprendre le langage.
- ECMA a proposé un standard « ECMAScript » (ISO) : ECMA 262, ISO/CEI 16262
 - ▣ Pendant longtemps, (l'époque jQuery), il s'agissait de l'ES5, la version 5 de l'ECMAScript.
 - ▣ Depuis juin 2015, une nouvelle version est disponible : l'ES6 (Une profonde transformation)
 - ▣ La mise à jour du standard JavaScript est désormais annuelle (avec peu de nouveautés). ECMAScript 2018 pour l'année 2018.
 - ▣ Tous les navigateurs modernes supportent l'ES6 depuis un moment, et les frameworks majeurs (Angular, React, Vue...) utilisent tous cette nouvelle version de JavaScript.
- **JavaScript** est actuellement à la version 1.8.6, est une implémentation du standard ECMA-262.

- C'est un langage orienté objets un peu rudimentaire (dite à prototype)
- C'est un langage peu typé :
 - Les types sont associés aux valeurs et pas aux variables.
 - La même variable peut prendre plusieurs types successivement.
- Il offre des types de base,
 - une syntaxe proche des langages de programmation comme le C et Java,
- Pour les programmeurs avancés, il fonctionne également comme un langage orienté objet et comme un langage procédural.
- Le code peut être lu par tout le monde;
- Les liaisons sont dynamiques, donc la référence des objets n'est pas vérifiée au chargement.
- Il peut modifier la structure, le contenu de la page HTML et son style (CSS)

Le code JavaScript :

- peut changer le contenu d'un élément HTML, la valeur d'un attribut HTML
 - `document.getElementById("demo").innerHTML = "Hello JavaScript";`
 - `document.getElementById("image").src = "newPict.png";`
 - `document.getElementById("demo").style.fontSize = "35px";`
 - `document.getElementById("demo").style.display = "none";`
 - `document.getElementById("demo").style.display = "block";`
- est ajouté dans le HTML via la balise script dans body ou head. Plusieurs scripts peuvent être ajoutés.
- permet de définir des fonctions :
 - ```
function myFunction() {
 document.getElementById("demo").innerHTML = "Paragraph
changed.";
}
```
  - peut être appelé lorsque un évènement se produit:  
`<button type="button" onclick="myFunction()">Try it</button>`

- Utilisation de la balise `<script>`;

```
<script type="text/javascript"> // ou <script>
alert("mon premier texte en JavaScript !!!!")
</script>
```

- Inclusion d'un fichier externe;

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

- Directement dans les balises en associant du JavaScript aux événements;

```
<INPUT TYPE="button" VALUE="ce paragraphe contient une phrase importante"
onClick='alert("Nous sommes tous égaux")' >
```

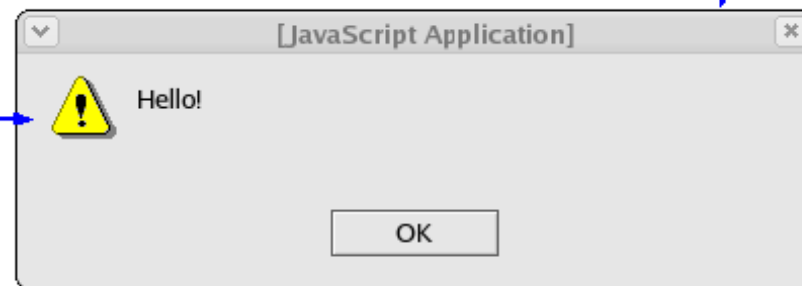
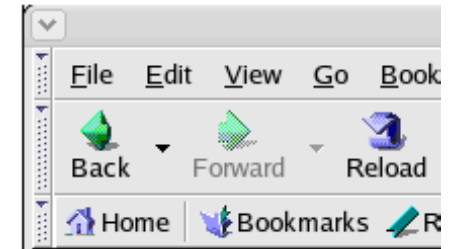
Cette dernière est à éviter!!

# Utilisation de la balise <script>

8

js02.htm

```
<html>
<head><!-- fichier js02.htm -->
 <script>
 function pushbutton() {
 alert("Hello!");
 }
 </script>
</head>
<body>
 <form>
 <input type="button" name="Button1"
 value="Push me" onclick="pushbutton()">
 </form>
</body>
</html>
```





# Inclusion d'un fichier externe;

9

js02.htm

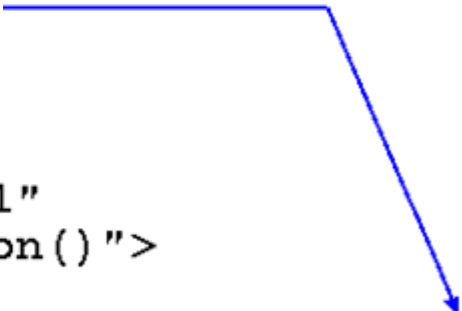


js021.htm

+

js022.js

```
<html>
<head>
 <script src="js022.js">
 </script>
</head>
<body>
<form>
 <input type="button" name="Button1"
 value="Push me" onclick="pushbutton()">
</form>
</body>
</html>
```



```
function pushbutton() {
 alert("Hello!");
}
```

- DOM (Document Object Model) :  
[https://developer.mozilla.org/fr/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model)
- Stockage
  - ▣ Web Storage
  - ▣ IndexedDB
  - ▣ API Cache => le future
- Géolocalisation : <https://developer.mozilla.org/fr/docs/Web/API/Geolocation>
- Graphique 2D et 3D :
  - ▣ Canvas : [https://developer.mozilla.org/fr/docs/Web/API/Canvas\\_API](https://developer.mozilla.org/fr/docs/Web/API/Canvas_API)
  - ▣ WebGL : [https://developer.mozilla.org/fr/docs/Web/API/WebGL\\_API](https://developer.mozilla.org/fr/docs/Web/API/WebGL_API)
- Audio et vidéo :
  - ▣ HTMLMediaElement
  - ▣ WebRTC
- Etc.

- ❑ Twitter : <https://developer.twitter.com/en/docs>
- ❑ Google Maps : <https://developers.google.com/maps/>
- ❑ Facebook : <https://developers.facebook.com/docs/>
- ❑ Etc.

## □ Sécurité :

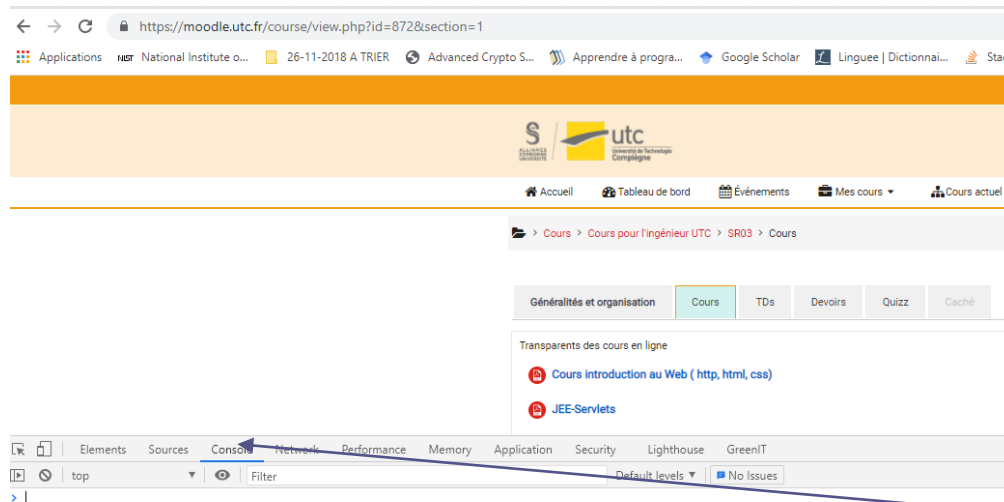
- ▣ onglet du navigateur constitue un périmètre séparé dans lequel s'exécute le code (en termes techniques ces périmètres sont des « environnements d'exécution ») ce qui signifie que, dans la plupart des cas, le code de chaque onglet est exécuté complètement séparément, et le code d'un onglet ne peut affecter directement le code d'un autre onglet ou d'un autre site. [Mmdn Mozilla ]

## □ Ordre d'exécution :

- ▣ De Haut vers le bas => faire attention aux références d'objets.

JavaScript peut afficher les données avec plusieurs manières:

- Ecrire dans un élément html en utilisant **innerHTML**.
- Ecrire dans la sortie html en utilisant **document.write()**.
- Ecrire dans une boite de dialogue alert en utilisant **window.alert()**.
- Ecrire dans la console du navigateur en utilisant **console.log()**.



F12 sous chrome  
et puis onglet  
console

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p>My First Paragraph</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
document.getElementById("demo").innerHTML = 5 + 6;
```

```
</script>
```

```
</body>
```

```
</html>
```

Voir :

<https://developer.mozilla.org/fr/docs/Web/API/Document>

<https://developer.mozilla.org/fr/docs/Web/API/Element>

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p>My first paragraph.</p>
```

```
<script>
```

```
document.write(5 + 6);
```

```
</script>
```

```
</body>
```

```
</html>
```

Voir :

<https://developer.mozilla.org/fr/docs/Web/API/Document>

- L'utilisation du document.write après que la page est chargée, efface tout l'ancien contenu HTML.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p>My first paragraph.</p>
```

```
<button type="button" onclick="document.write(5 +
6)">Try it</button>
```

```
</body>
```

```
</html>
```

Voir :

<https://developer.mozilla.org/fr/docs/Web/API/Document>

[Tester](#)



# La sortie : window.alert()

17

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p>My first paragraph.</p>
```

```
<script>
```

```
window.alert(5 + 6);
```

```
</script>
```

```
</body>
```

```
</html>
```

<https://developer.mozilla.org/fr/docs/Web/API/Window>

# La sortie : console.log()

18

```
□ <!DOCTYPE html>
 <html>
 <body>

 <script>
 console.log(5 + 6);
 </script>

 </body>
 </html>
```

<https://developer.mozilla.org/fr/docs/Web/API/Console/log>

## Attention à la méthode `window.print`

19

- Javascript ne dispose d'aucune méthode `print` pour afficher dans une page HTML.
- La méthode `window.print` permet d'imprimer la page en cours

```
<!DOCTYPE html>
<html>
<body>
```

```
<button onclick="window.print()">Print this page</button>
```

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

- Les commentaires : deux formes possibles :
  - ▣ // commentaire jusqu'en fin de ligne
  - ▣ /\* commentaire jusqu'à \*/
- La déclaration de fonctions
  - ▣ `function nom() { instructions }`
  - ▣ `function nom(param1,...,paramN) { instructions }`
- L'instruction return (sortir d'une fonction)
  - ▣ `return, return expression`

- *Exemple:*

```
function getPluriel(nb) {
 if nb > 1 {
 return "s"
 }
 return " " ; }

```

Portée des variables (Var)  
Globale : visible depuis n'importe où.  
Locale : visible que dans la fonction courante.

```
function test_visibilite() {
 var test = 5;
}
test = 2;
test_visibilite();
alert (test);

```

Rappel : portée par bloc (let ) : visible uniquement dans le bloc où la variable est déclarée {...}.

## □ JavaScript reconnaît les types primitifs suivants :

### ▣ les nombres : 42 ou 3.1415

- Il n'y a pas de distinction entre entiers et réels.
- Les réels utilisent "." pour la virgule et e ou E pour la partie exposant.
- ```
var y = 123e5;    // 123000000
var z = 123e-5;   // 0.00123
```

▣ les booléens : true ou false

▣ les chaînes : "Bonjour !"

▣ les tableaux (Array) : ``` let tab = ['fruit', 32, [0, 1, 2]]; var cars=["Saab","Volvo","BMW"]; ou var cars=new Array("Saab","Volvo","BMW"); cars[0]; // renverra Saab ```

▣ les objets (Object)

```
var person={
  firstname : "John",
  lastname  : "Doe",
  id        : 5566
};
```

▣ null

▣ indéfini (Undefined)

- Les chaînes
 - ▣ On peut utiliser " ou ' pour délimiter une chaîne.
 - ▣ Des caractères spéciaux peuvent être utilisés à l'intérieur :
 - \ b pour un espacement arrière,
 - \ f pour un saut de page,
 - \ n pour un saut de ligne,
 - \ r pour un retour au début de ligne,
 - \ t pour une tabulation.
- Undefined (aucune valeur) vs null (effacer les données)
- New : déclarer le type d'une variable
 - ▣ var carname=new String;
 - ▣ var x= new Number;
 - ▣ var y= new Boolean;
 - ▣ var cars= new Array;
 - ▣ var person= new Object;
 - ▣ let carname=new String;
 - ▣ let x= new Number;
 - ▣ let y= new Boolean;
 - ▣ let cars= new Array;
 - ▣ let person= new Object;

- La déclaration de variables
 - ▣ Syntaxe : `let (ou var) nom1 [= valeur1] [..., nomN [= valeurN]]`.
 - exemple : `let i = 0, j = 2;`
- JavaScript est faiblement typé : les variables ne sont pas associées à un type particulier.
 - ▣ Exemple :
 - `let i = 4.25;`
 - `i = "Bonjour";`

Dans une expression faisant intervenir des nombres et des chaînes les nombres sont convertis en chaîne.

- ▣ `message = " Il y a " + 107 + "étudiants inscrits au SR" + 03`

Déclaration des variables (suite)

24

- **Variables globale**

```
var carName = "Volvo";  
  
// code here can use carName  
  
function myFunction() {  
    // code here can also use carName  
}
```

- **Variable locale (fonction)**

```
// code here can NOT use  
carName  
  
function myFunction() {  
    var carName = "Volvo";  
    // code here CAN use carName  
}  
  
// code here can NOT use  
carName
```

- **Variable locale à un bloc (ES6) :**

```
Let  
{  
    let x = 2;  
}  
// x can NOT be used here
```

-

- **Constante (ES6)**

```
const PI = 3.141592653589793;
```

- **Variables globales dans HTML**

- Les variables globales sont accessibles via l'objet window

```
var carName = "Volvo";  
// code here can use window.carName
```

ES6 2015 a introduit deux nouveaux mots clés : **let** et **const**.

Déclaration des variables (suite)

25

- ❑ Variables déclarées avec Let :
 - ▣ Portée : bloc ({...})
 - ▣ Double déclaration : non => SyntaxError
 - ▣ Utilisation de la variable avant sa déclaration : non => ReferenceError
- ❑ Variables déclarées avec Var
 - ▣ Portée : Globale (à l'extérieur) ou fonction
 - ▣ Double déclaration : oui
 - ▣ Utilisation de la variable avant sa déclaration (voir Javascript Hoisting) :
oui
 - Hoisting est (pour de nombreux développeurs) un comportement inconnu ou négligé de JavaScript => bugs au niveau des programmes
 - Pour éviter ces bugs il est recommandé de déclarer toutes les variables au début de chaque portée

- Opérateurs arithmétiques
+, -, /, *, **** (ES2016)**, %, --, ++
- Les opérateurs logiques
&&, || et !
- Les opérateurs de comparaison
==, !== (égalité et inégalité stricte) => moins d'erreurs dans le code
>, <, >=, <=,
==, !== : vérifier la valeur et non pas le type
- Les opérateurs binaires
& (and), | (or), ~ (not), ^ (xor), << (décalage à gauche), >> (décalage à droite), >>> (décalage à droite non signé)
- Opérateurs sur les chaînes
+, +=

- Opérateurs d'affectation
=, +=, -=, *=, **=, &=, |=, >>=, etc.
- Les opérateurs spéciaux :
 - ▣ *new* permet de créer un objet.
 - ▣ *typeof* permet de déterminer le type de l'opérande.
 - Syntaxe : *typeof expression* ou *typeof (expression)*.
 - ▣ *Instanceof* permet de déterminer si un objet est une instance d'un type objet (elle retourne True ou False)
 - ▣ Void permet d'évaluer une expression donnée et de renvoyer undefined
- Méthode « toString »
 - ▣ let num=32;
 - ▣ let str=num.toString();
- L'objet « Number »
 - ▣ let str= '32';
 - ▣ let num=Number(str)

- Des objets prédéfinis : Array, Boolean, Date, Math (fonction : random, floor, ...), Number, Set, Maps, Object, String (RegExp [search(), replace(),])...
 - Pour chaque objet il y a des constructeurs, propriétés et des méthodes prédéfinies, ex. pour *Array* : voir https://www.w3schools.com/js/js_arrays.asp
 - Constructeur: `Array(n)`, `Array(el1, el2, ..., eln-1)`,
 - Propriétés : `index`, `input`, `length`..
 - Méthodes : `concat`, `join`, `pop`, `push`, `sort`, `toString`...
 - L'objet *Math* est un objet non instanciable possédant des propriétés et des méthodes
 - ...
 - Les objets DOM...
- Les fonctions prédéfinies
 - `eval()`, `escape()/unescape()`, (ou `encodeURIComponent/decodeURI`)

- Evalue et exécute le code javascript contenu dans chaine.
- Exemple : <SCRIPT language=javascript>
var a=1;
chaine="a = a + 10";
eval(chaine);
document.write("Valeur de a = "+a);
</SCRIPT>

Attention : L'exécution de JavaScript à partir d'une chaîne de caractères constitue un risque de sécurité énorme. Il est beaucoup trop facile pour un mauvais acteur d'exécuter du code arbitraire lorsque vous utilisez eval()

- L'attribut length :
 - ▣ let name = 'Ahmed Lounis';
 - ▣ name.length;
- La méthode indexOf
 - ▣ name.indexOf('Lounis');
- La méthode slice
 - ▣ name.slice(0,5);
 - ▣ name.slice(6); // renverra Lounis
- Méthode pour la casse
 - ▣ let exData = 'My NaMe Is AhmeD';
 - ▣ exData.toLowerCase();
 - ▣ exData.toUpperCase();
- La méthode replace
 - ▣ name.replace('Ahmed','Adouane');

Méthodes (ou attributs) de l'objet Array

31

- L'attribut length :
 - ▣ `let tab = ['pomme', 'banane'];`
 - ▣ `tab.length;`
- La méthode `split` : retourne un tableau à partir d'une chaîne de caractère
 - ▣ `let myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';`
 - ▣ `let myArray = myData.split(',');`
- La méthode `join` : retourne une chaîne de caractère à partir d'un tableau
 - ▣ `let myNewString = myArray.join(',');`
- La méthode `toString`
 - ▣ `let dogNames = ["Rocket","Flash","Bella","Slugger"];`
 - ▣ `dogNames.toString(); //Rocket,Flash,Bella,Slugger`
- Les méthodes `push ()` et `pop ()` : ajouter ou supprimer un élément à la fin du tableau
- Les méthodes `unshift()` et `shift()` : ajouter ou supprimer un élément au début du tableau

- ❑ La boucle `do ... while` est :
 - ❑ `do instruction-ou-bloc while (condition).`
- ❑ La boucle `while` :
 - ❑ `while (condition) instruction-ou-bloc`
- ❑ La boucle `for`, la boucle `for ... in` :
 - ❑ `for (initialisation ;condition ;incrémentation) instruction-ou-bloc.`
 - ❑ `for (variable in objet) instruction-ou-bloc`
- ❑ Le test `if ... Else`
- ❑ Expressions conditionnelles « `? :` »
 - ❑ La forme est `condition ? val1 : val2.`
 - ❑ Exemple : `m=(age>=18) ? "majeur" : "mineur"`
- ❑ Le branchement multiple `switch`
- ❑ L'instruction `break` : sortir immédiatement de la boucle
- ❑ L'instruction `continue` : aller vers l'itération suivante

Boucle et branchement (suite)

33

- ```
while (i<5)
{
 x=x + "The number is " + i + "
";
 i++;
}
```
- ```
do
{
    x=x + "The number is " + i + "<br>";
    i++;
}
while (i<5);
```
- ```
cars=["BMW", "Volvo", "Saab", "Ford"];
let i=0;
for (;cars[i];)
{
 document.write(cars[i] + "
");
 i++;
}
```

# Boucle et branchement (suite)

34

- ```
for (var i=0, len=cars.length; i<len; i++)  
{  
    document.write(cars[i] + "<br>");  
}
```
- ```
var person={fname:"John", lname:"Doe", age:25};

for (x in person)
{
 txt=txt + person[x];
}
```
- ```
const name = "ABCDEF";  
  
for (const x of name) {  
    // code block to be executed  
}
```
- ```
const letters = ["a", "b", "c"];

for (const x of letters) {
 // code block to be executed
}
```

- L'instruction :

- `try` {  
    *Block of code to try*  
}  
    `catch(err)` {  
        *Block of code to handle errors*  
    }  
    `finally` {  
        *Block of code to be executed regardless of the try / catch result*  
    }

- Objet Error

- { Name : " ",  
    Description : ""  
}

Six noms d'erreur standardisés : EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError

- L'instruction throw : définir une erreur personnalisée

- `if(x > 10) throw "is too high";`

- Built-in HTML validation :

- Combiner entre Javascript et les règles de validation fournies par HTML

- `<input id="demo" type="number" min="5" max="10" step="1">`

- L'utilisation de : **"use strict"**; au début d'un script ou une fonction
  - ▣ Aide à écrire un Code plus sûr en évitant les mauvaises syntaxes (une mauvaise syntaxe déclenchera une erreur dans ce mode car elle n'est pas acceptée)
- Pratiques non acceptées avec **"use strict"** :
  - ▣ `x = 3.14;` // x n'est pas déclaré. Tous les variables et objets doivent être déclarés avant leur utilisation
  - ▣ `let x = 3.14;` // ou `function x(){...}`  
`delete x;` // la suppression d'une variable (ou d'une fonction) n'est pas permise
  - ▣ La duplication des noms n'est pas permise.
  - ▣ Etc. Voir  
[https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict\\_mode](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict_mode)

37

## JavaScript au service du web

- A ses débuts, JavaScript était utilisé pour des fins d'animation, effets graphiques...,
- Depuis JavaScript est principalement utilisé dans les pages web interactives; facilitant la vie de l'utilisateur mais aussi apporter une valeur ajoutée au fournisseur de pages web.
- Différentes utilisations de JavaScript (ajoute un comportement dynamique à la page au niveau client ):
  - Créer des effets visuels, trier les colonnes d'un tableau;
  - Contrôle de formulaires html;
  - Outils de mesure d'audience, questionnaire des bandes publicitaires...
  - Ajax: aide au remplissage automatique de formulaire...

## □ JavaScript et la sécurité

- ▣ JavaScript ne peut pas accéder aux fichiers stockés sur le site du client;
- ▣ JavaScript ne peut pas accéder aux pages html venant d'autres sites.
- ▣ JavaScript n'autorise pas l'installation de programmes exécutables sur le site client;

## □ Limites de JavaScript

- ▣ Compatibilité navigateurs et plateformes;
- ▣ Code JavaScript non visible par les moteurs de référencement;

- **JavaScript** est destiné à faire du **HTML dynamique**, (*Dynamic HTML*, ou **DHTML**) !
- **DHTML** est un nom générique donné à l'ensemble des techniques utilisées par l'auteur d'une page web pour que celle-ci soit capable de se modifier elle-même en cours de consultation dans le navigateur web.



- Les technologies que le DHTML met en œuvre sont :
  - ▣ Le HTML, nécessaire pour présenter le document ;
  - ▣ Les feuilles de style (CSS), permettant de définir un style pour plusieurs objets et le positionnement de ceux-ci sur la page ;
  - ▣ Le modèle objet de document (DOM), proposant une hiérarchie d'objets, afin de faciliter leur manipulation ;
  - ▣ Le JavaScript, langage de script essentiel pour définir des événements utilisateur ;
  - ▣ Ajax permet de dynamiser les échanges avec le serveur.

## □ Animer des éléments:

- L'animation = modification des propriétés (position, hauteur, largeur, visibilité, couleur ...) ou en utilisant leur méthodes (fonctions associées à un élément).
- Cela ne peut se faire qu'à l'aide de :
  - d'un code JavaScript, permettant de modifier les propriétés des éléments suite à des événements utilisateurs (clic sur la souris, déplacement de la souris, ...),
  - + une structuration des éléments dans la page définie par le DOM (*Document Object Model*).

# Document Object Model

43

- Le **Document Object Model** (ou **DOM**) (est un **W3C standard**) décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents.
- DOM est souvent identifié par une arborescence de la structure d'un document et de ses éléments.
  - P.ex. chaque élément généré à partir du balisage comme, dans le cas de HTML, un paragraphe, un titre ou un bouton de formulaire, y forme un nœud.
- DOM est utilisé pour pouvoir modifier facilement ou accéder au contenu des pages web.
- A partir d'un arbre DOM donné, il est aussi possible de générer des documents dans le langage de balisage voulu;

Les liens hiérarchiques entre les noeuds définissent la structure du document et sont représentés dans l'arbre DOM.

Il y a trois types de noeuds DOM dans une arborescence :

**Les noeuds « elements ».** *L'etiquete associée au nœud donne le nom de l'élément;*

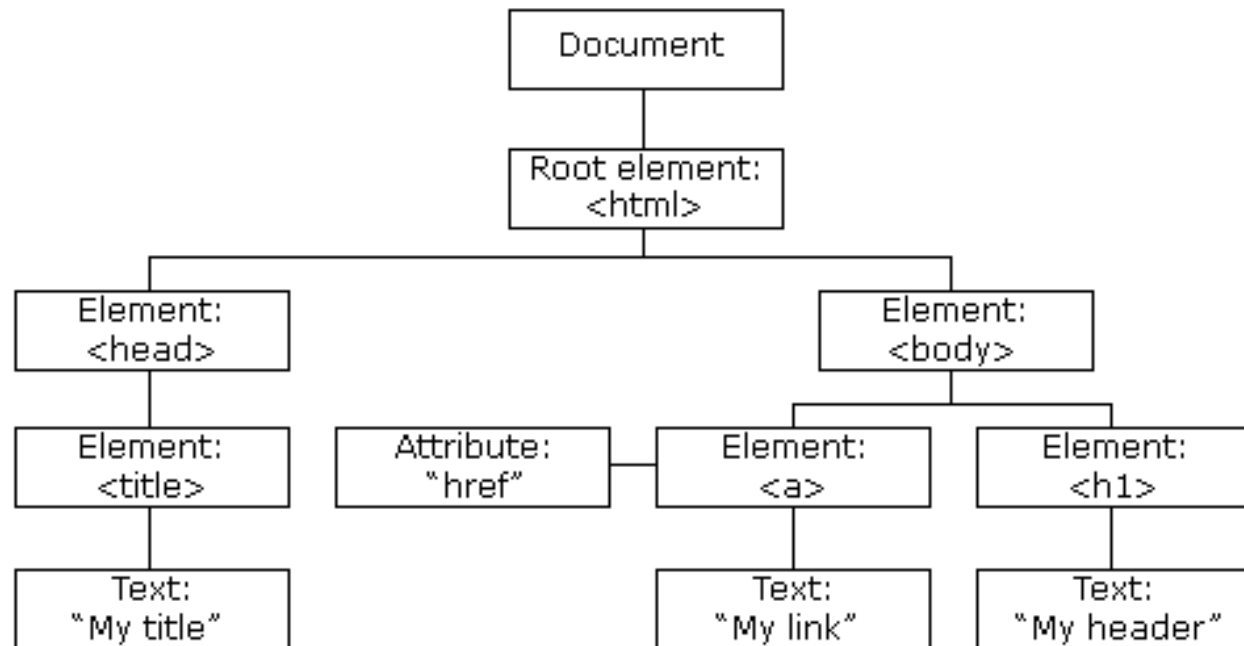
**Les noeuds « text ».** *Le texte contenu dans un élément apparait comme un nœud texte dans l'arbre DOM.*

**Les noeuds « attribut ».** *Les attributs d'un élément html sont représentés comme des noeuds attributs dans l'arbre DOM.*

# L'arborescence DOM

45

```
<html>
<head>
<title> My title</title>
</head>
<body>
<h1> My header</h1>
 My link
</body>
</html>
```



# Dom versus JavaScript

46

- Alors que JavaScript est le langage de programmation qui permet d'opérer sur les objets DOM et de les manipuler, le DOM fournira les méthodes et les propriétés nécessaires pour lire, modifier, mettre à jour et supprimer des éléments du document examiné.
- L'interface **Document** est la première à être définie dans la spécification DOM Level 1 Core, et `document` est un objet hôte implémentant cette interface. Cet objet document contient tout ce qui figure dans une page web.
- L'interface **Document** définit les opérations et les requêtes qui peuvent se produire sur un document HTML.

# Objets HTML et DOM

47

- *Window* est un objet qui correspond à la fenêtre dans laquelle s'affiche une page Web.
- *Location* est un sous-objet de *Windows*. C'est un objet coté-client (donc créé par le navigateur). *Location* est actuellement une interface intégrée à DOM et aussi un attribut de *HTMLDocument*.
- L'objet *Document* défini dans la spécification DOM permet d'accéder au contenu d'une page HTML. Les méthodes les plus utilisées de l'objet sont *getElementByTagName* et *getElementById*, *open*, *write*...
- *History* est un des sous-objets de *Windows* qui sert d'interface avec l'historique de navigation conservé par le navigateur.
- L'interface *Node* est définie dans la spécification DOM 2, elle permet d'accéder à la structure d'un document HTML vu comme une arborescence d'élément, et de modifier cette structure.
- L'interface *NodeList* est un objet DOM 2, qui permet d'accéder aux éléments d'une page Web ou d'un fichier XML. C'est un élément dynamique, tous les changements de structure dans la page modifient le contenu de *NodeList*.

# Window (I)

48

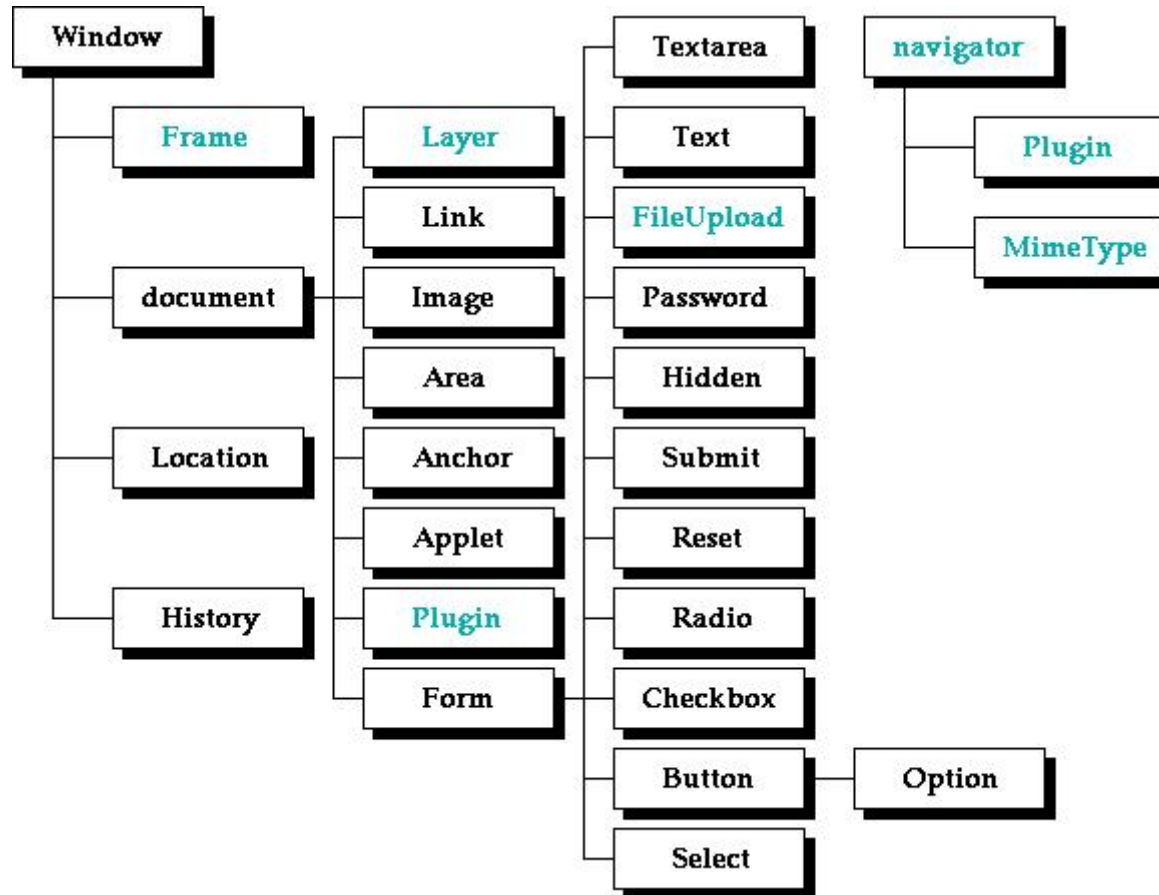


- *Window* est un objet qui correspond à la fenêtre (onglet) dans laquelle s'affiche une page Web. Une telle fenêtre peut être créée dynamiquement.
- **Attributs de window**
  - ▣ **length** Nombre de frames. Lecture seule.
  - ▣ **name** Nom de la fenêtre.
  - ▣ **status** Texte de la barre d'état.
  - ▣ **parent** La fenêtre parent d'une fenêtre donnée.
  - ▣ ...
- **Propriétés de création de fenêtre**
  - ▣ `scrollbars`, `statusbar`, `toolbar`, `menubar`, `resizable`, `directories`.



# Window (II)

49

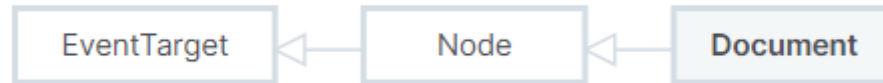


## □ Objets contenus dans window

- ▣ **Document** : Désigne une page, celle que contient une fenêtre.
- ▣ **History** : Liste des pages précédemment vues dans la même fenêtre.
- ▣ **Location** : Désigne l'URL d'une page, celle que contient la fenêtre.
- ▣ **Screen** : Désigne l'écran et contient les propriétés: width, height, availWidth, availHeight, colorDepth.
- ▣ **Navigator**
- ▣ **LocalStorage** ...

## □ Méthodes de window

- ▣ **open(url, nom [, liste de propriétés])** Ouvre une nouvelle fenêtre. L'URL et le nom sont donnés en paramètres ainsi que la liste des options.
- ▣ **close()** Ferme la fenêtre. Exemple: x.close(); window.close();
- ▣ **alert(message)** Affiche un message.
- ▣ ...



- **Attributs de *Document*** (ce sont des attributs en lecture seule, définis lors de la création de la page.)
  - ▣ *DocumentType* **doctype** (Le doctype définit en tête de page.)
  - ▣ *DOMImplementation* **implementation**
  - ▣ *Element* **documentElement** Un *Element* est un objet qui représente une balise et est doté de méthodes.
  - ▣ *String* **title**,
  - ▣ *String* **URL**,
  - ▣ *HTMLElement* **body**,
  - ▣ *HTMLCollection* **links**,
  - ▣ *HTMLCollection* **forms**,
  - ▣ *String* **cookie**
  - ▣ ...

## □ Méthodes de *Document*.

### ▣ *Element* **createElement**(*String*)

Crée un élément, une balise, dont le nom est donné, et retourne un objet *Element*.

### ▣ *Element* **getElementById**(*String*)

Retourne un élément dont on donne l'identificateur...

### ▣ void **open**() Crée un nouveau document.

### ▣ void **close**() Ferme le document courant.

### ▣ void **write**(*String*) Ecrit dans le document. Warning: Use of the document.write() method is strongly discouraged.

### ▣ void **writeln**(*String*) Ecrit la chaîne en paramètre et ajoute un saut à la ligne. Warning: Use of the document.writeln() method is strongly discouraged.

### ▣ ...

Soit donné le lien suivant :

`http://www.xul.fr:80/ecmascript/tutoriel/window-location.php#content?name=value`

`[http:][//[www.xul.fr]:[80]][/ecmascript/tutoriel/windows-location.php]#[content]?[name=value]`

`[protocol][host][port][pathname][hash][search]`

**protocol** http:

**host** www.xul.fr:80

**hostname** www.xul.fr

**port** 80

**pathname** ecmascript/tutoriel/window-location.php

**hash** content

**search** name=value

**href** correspond à l'URL complète.

## Utilisation de location

Les propriétés de *location* peuvent être lues ou modifiées.

Par exemple on affiche le chemin de la page ainsi:

```
document.write(location.pathname)
```

Et on change la page ainsi:

```
location.href = "url"
```

et d'autres possibilités avec `replace()`, `reload()`...

- **Méthodes de history** : Elle servent à passer d'une page à la précédente ou la suivante ou aller à une page donnée.
  - ▣ **back()** Charger la page précédente.
  - ▣ **forward()** Charger la page suivante, après un retour en arrière.
  - ▣ **go(x)** Charger une page dont on donne le numéro dans la liste ou l'URL.  
Exemple: `history.go(-2)` pour annuler les deux derniers clics.

- L'interface *Node* est définie dans la spécification DOM 2, elle permet d'accéder à la structure d'un document HTML vu comme une arborescence d'élément, et de modifier cette structure.
- Propriétés
  - ▣ *String* **nodeName** Le nom de la balise comme <table>, <pre> etc.
  - ▣ *String* **nodeType** Catégorie de nœud: élément, texte, attribut, etc.
  - ▣ *String* **nodeValue** La valeur du nœud lorsque cela à un sens. Tous les nœuds n'ont pas de valeur, dans ce cas, vaut *null*.
  - ▣ *NamedNodeMap* **Element.attributes** Liste les attributs d'un nœud. Lecture seule.
- Méthodes
  - ▣ *Node* **appendChild**(*Node* ajouté) Ajoute un nœud à la liste des nœuds enfants.
  - ▣ *Node* **cloneNode**(*Boolean*) Clône un Node et, éventuellement, tout son contenu. Par défaut, il duplique le contenu de ce nœud.
  - ▣ *Boolean* **hasAttributes**() Retourne true ou false, selon qu'il a des attributs ou non.
  - ▣ *Boolean* **hasChildNodes**() Retourne true ou false selon qu'il contient d'autres nœuds ou non.
  - ▣ *Node* **removeChild**(*Node* supprimé) Supprimer un nœud ou une branche. Retourne l'élément supprimé.
  - ▣ *Node* **replaceChild**(*Node* nouveau, *Node* ancien) Remplace le second argument par le premier.
  - ▣ *Remove()* : *supprimer un élément html. Elle risque de ne pas fonctionner dans les anciens navigateurs*

# L'interface *Node* : un exemple

56

```
<div id="container">
 <h1 name="hclasse" id="hid"> Niveau 1 </h1>
 <p title="paraclasse" id="pid"> C'est un paragraphe
</p>
 <h3 name="hclasse"> Fin de la page </h3>
</div>

<script type="text/javascript">
 var parentID = document.getElementById("container") ;
 document.write("afficher les noeuds de type element et texte" + "</br>");
 var newP = document.createElement("p") ;
 parentID.appendChild(newP) ;
 var pTag = document.getElementsByTagName("p") ;
 document.write(pTag[1].nodeName + "</br>") ;
 var newT = document.createTextNode("Bonjour") ;
 newP.appendChild(newT) ;
 document.write(pTag[1].firstChild.nodeValue + "</br>") ;
 var oldH = document.getElementById("hid");
 parentID.replaceChild(newP,oldH) ;
</script>
```



- L'interface *NodeList* est un objet DOM 2, qui permet d'accéder aux éléments d'une page Web ou d'un fichier XML.
  - ▣ C'est un élément dynamique, tous les changements de structure dans la page modifient le contenu de *NodeList*.
  - ▣ *NodeList* a un seul attribut (*int* **length** Nombre de *Nodes* dans la liste), et une seule **Méthode** *Node* **item(int)** qui retourne le *Node* d'indice donné en argument.

Exemple:

```
var dnl = document.getElementsByTagName("a");
for(i = 0; i < dnl.length; i++)
 var lien = dnl.item(i);
```

*Le code ci-dessus permet d'avoir la liste des liens dans une page puisqu'ils sont introduits par la balise <a>. On obtient ensuite chacun des liens grâce à la méthode item() de NodeList.*

- Un événement est une action engendrée par l'utilisateur.
- Chaque balise peut gérer un événement : sur l'apparition d'un événement du code JavaScript peut être lancé.

Syntaxe <balise ManipulateurÉvénement = "code JavaScript" )

```
<input type= "button" value =
"Cliquer ici"
onclick = "alert ('Merci !');">
```

*Alternativement :*

```
<form name= "f1" >
<input name= "b1" type= "button" value=
"Cliquer ici" >
</form>
document.f1.b1.onclick = function(){alert
('Merci !'); };
```

```
<button onclick="alert ('Merci !')"> Cliquer
ici</button>
```

Alternativement :

```
<button > Cliquer ici</button>
var btn =
document.querySelector('button');
btn.onclick = function() {alert ('Merci !');};
ou :
```

```
btn.addEventListener('click', =
function(){alert ('Merci !'); });
```

- événements page et fenêtre
  - ▣ *onabort* - s'il y a une interruption dans le chargement
  - ▣ *onerror* - en cas d'erreur pendant le chargement de la page
  - ▣ *onload* - après la fin du chargement de la page
  - ▣ *onbeforeunload* ; *onunload* ; *onresize* ;
- événements souris
  - ▣ *onclick* - sur un simple clic
  - ▣ *ondblclick* - sur un double clic
  - ▣ *onmousedown* - lorsque que le bouton de la souris est enfoncé, sans forcément le relâcher
  - ▣ *onmousemove* - *onmouseout* - *onmouseover* – *onmouseup*;
- événements clavier
  - ▣ *onkeydown* - lorsqu'une touche est enfoncée
  - ▣ *onkeypress* - lorsqu'une touche est pressée et relâchée
  - ▣ *onkeyup* - lorsqu'une touche est relâchée
- événements formulaire
  - ▣ *onsubmit* - quand le formulaire est validé (*via* un bouton bouton de type "submit" ou une fonction submit())
  - ▣ *onreset* - lors de la remise à zéro du formulaire (*via* un bouton "reset" ou une fonction reset())
  - ▣ *onselect*; *onblur*; *onchange*; *onfocus*;
- Référence des événements : <https://developer.mozilla.org/fr/docs/Web/Events>

# Associer du JavaScript aux événements

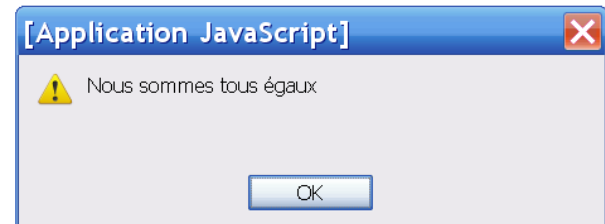
60

```
<html>
<head>
<title>Exemple JavaScript</title>
</head>
<body>
<h1>Exemple JavaScript</h1>
<INPUT TYPE="button" VALUE="ce paragraphe contient une phrase importante"
onclick='alert("Nous sommes tous égaux")' >

</BODY>
</HTML>
```

## Exemple JavaScript : associer du javascript aux événements

ce paragraphe contient une phrase importante



# Associer du JavaScript aux événements : contrôle de formulaires (I)

61

Veillez remplir les champs suivants :

Civilité\* : ☐ Mr ☐ Mme ☐ Mlle

Nom\* :

Prénom\* :

Date de naissance\* :

Adresse\* :

Code postal\* :

Ville\* :

Pays\* : Choisissez votre pays

Téléphone :

Fax :

Mobile :

E-mail\* :

☐ J'accepte les conditions générales d'utilisation du site\*

(\*) Champs obligatoires

- Associer des fonctions javascript pour contrôler comment les champs sont remplis :
  - ▣ Tester si la civilité est renseignée,
  - ▣ tester le champ E-mail,
  - ▣ tester si dans les champs (code postal, téléphone, fax) est de valeur entière,
  - ▣ Tester la date...
  - ▣ Tester l'acceptation des conditions générales
  
- ▣ Créer une fonction testSaisie() à associer à onSubmit().

# Associer du JavaScript aux événements : contrôle de formulaires (II)

62

```
<BODY>
<FORM name="form1" onSubmit="return testSaisie()">
 <P>Veuillez remplir les champs suivants :</P>
 <P>Civilité* :
 <INPUT type="radio" name="civ" value="Mr">Mr
 <INPUT type="radio" name="civ" value="Mme">Mme
 <INPUT type="radio" name="civ" value="Mlle">Mlle

 ...
 <P>(*) Champs obligatoires</P>
 <P>
 <INPUT type="submit" name="Submit" value="Envoyer">
 <INPUT type="reset" name="Submit2" value="Rétablir">

</P>
</FORM>
```

```
function testMail(email)
{
 var posArobase;
 posArobase = email.indexOf("@");
 if (posArobase == -1) return false;
 var posPoint;
 posPoint = email.lastIndexOf(".");
 if ((posPoint == -1) || (posPoint < posArobase)) return false;
 return true;
}
```

```
function testNumerique(valeur)
{
 if (valeur == parseFloat(valeur)) return true;
 else return false;
}
```

```
function testRadio(nomForm,nomGroupe)
{
 var compteur;
 compteur = 0;
 while (compteur < nomForm.elements[nomGroupe].length)
 {
 if (nomForm.elements[nomGroupe][compteur].checked)
 return true;
 compteur++;
 }
 return false;
}
```

# Gestion des événements clavier

63

```
<form name="leformulaire">
Code postal : <input type="text" name="codepostal">

Ville : <input type="text" name="ville">
</form>
<script type="text/javascript">
document.forms["leformulaire"].elements["codepostal"].onkeypress = traiterCodepostal
function traiterCodepostal(e) {
 var txtCarOk="0123456789";
 var car="";
 var isCarOk=false;
 car=String.fromCharCode(e.charCode);
 if (txtCarOk.indexOf(car)>=0) {
 isCarOk=true;
 }
 if ((e.charCode==0)&&(e.keyCode>0)) {
 return true;
 }

 if (isCarOk) {
 if (document.forms["leformulaire"].elements["codepostal"].value.length<5) {
 document.forms["leformulaire"].elements["codepostal"].value+=car;
 }
 }
 return false;
 }
</script>
```

# JavaScript et les cookies

64

- ❑ Pas de méthodes natives au JavaScript pour gérer les cookies.
- ❑ On utilise la propriété *cookie* de l'objet *document* : Exemple :  
`document.cookie = "username=John Smith; expires=Thu, 18 Dec 2018 12:00:00 UTC; path=/";`
- ❑ On écrit une fonction `setCookie()` et une autre `getCookie()` .

```
function setCookie(cname, cvalue, exdays) {
 var d = new Date();
 d.setTime(d.getTime() + (exdays*24*60*60*1000));
 var expires = "expires=" + d.toUTCString();
 document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
function getCookie(cname) {
 var name = cname + "=";
 var decodedCookie = decodeURIComponent(document.cookie);
 var ca = decodedCookie.split(';');
 for(var i = 0; i <ca.length; i++) {
 var c = ca[i];
 while (c.charAt(0) == ' ') {
 c = c.substring(1);
 }
 if (c.indexOf(name) == 0) {
 return c.substring(name.length, c.length);
 }
 }
 return "";
}
```



- JavaScript permet de modifier les styles CSS:
  - ▣ modifier les styles en ligne d'un élément :
    - ex. changement de couleurs d'un élément...
      - On peut accéder à un élément via `document.getElementById...`
  - ▣ modifier la classe d'un élément :
    - On peut accéder à une classe via l'attribut `className` de tout élément html...
    - Toute élément CSS est accessible via JavaScript et DOM selon la règle:
      - Le nom ne change pas si absence de tiret. Sinon supprimer le tiret et mettre en majuscule la première lettre qui le suit.
  - ▣ manipuler les feuilles de style elles-mêmes:
    - Il s'agit d'activer et désactiver des feuilles de style
      - Utiliser la propriété *disabled* pour les éléments `<link>` et `<style>`

- ▣ HTML permet d'encapsuler du texte dans les balises afin de procéder à sa mise en page;
- ▣ Les CSS permettent de définir la présentation des balises... ils peuvent être conditionnées (p.ex. en fonction du media);
- ▣ JavaScript est un langage intégré au HTML.
- ▣ DHTML est la réunion de HTML, CSS et Javascript. Il permet grâce au DOM de réécrire la page HTML sans la recharger



# AJAX

# Ajax: motivation et principe de fonctionnement

68

- Ajax (*Asynchronous JavaScript + XML*) est l'utilisation conjointe de plusieurs technologies comme XHTML, CSS, DOM, XML et/ou JSON, tout lié par JavaScript qui fait appel au mécanisme XMLHttpRequest.
- Ajax permet l'échange d'informations entre le navigateur et le serveur web sans recharger la totalité de la page.
  - Il permet de garder des pages web affichées et avoir des performances accrues;
  - Analyser et travailler avec des documents XML;
  - Pouvoir continuer le travail sans nécessairement attendre la réponse du serveur.

## Avantages :

- Une économie de ressources côté serveur et de bande passante puisque la page n'est pas systématiquement transmise pour une mise à jour
- Une meilleure réactivité et dynamique de l'application web

## Inconvénients :

- Complexité liée à l'utilisation de plusieurs technologies côté client et serveur
- inconvénients liés à l'utilisation de Javascript : difficulté pour déboguer, différences d'implémentation selon le navigateur, code source visible, ...

# Synchrone versus asynchrone

69

Principe: *l'exécution du code JavaScript continue en parallèle du traitement de la requête Ajax (asynchrone).*

- ▣ Une requête est envoyée au serveur (grâce à XMLHttpRequest),
- ▣ Le serveur envoie la réponse;
- ▣ Le JavaScript (et non le navigateur) réceptionnera et traitera la réponse.

Prenons l'exemple de la gestion d'un panier de commande sur un site de commerce électronique.

En mode synchrone l'internaute ajoute un article dans le panier.

- Une requête est envoyée au serveur
- Le navigateur **attend** la réponse du serveur
- Le serveur valide l'ajout
- Le serveur renvoi le contenu du nouveau panier
- Le navigateur rafraichit la zone d'affichage du panier
- L'internaute **continue** ses "achats"

En mode asynchrone l'internaute ajoute un article dans le panier.

- Une requête est envoyée au serveur
- Le navigateur continue son exécution
- L'internaute **continue** ses "achats"
- Le serveur valide l'ajout
- Le serveur rappelle le navigateur et communique le contenu du nouveau panier
- Le navigateur rafraichit la zone d'affichage du panier

L'objet *XMLHttpRequest* constitue la clé de voute d'Ajax.

## Propriétés

Onload : (on lui affecte une fonction avec les instructions à exécuter lorsque la requête est reçu);

Onreadystatechange : (on lui affecte une fonction avec les instructions de traitement de la réponse);

readyState : l'état d'avancement de la requête de 0 (non initialisée) à 4 (*complétée*).

responseText / responseXML : la réponse du serveur à la requête en texte (i.e. objet reconnu par JavaScript) ou formatée en un objet XML.

Status : état de la réponse HTTP du serveur. (200 OK)

## Méthodes

open(*method*, *url*, *bool*) : (si mode asynchrone *bool* vaut true et sinon false)

send(*data*) : Envoie la requête HTTP au serveur (quand il n'y a pas de données la valeur est null)

...

```
req= new XMLHttpRequest();
req.open("GET", location.href, false);
req.send(null);
if(req.readyState == 4)
alert("Requête terminée !");
```

# Ajax : comment ça marche (II)

71

## Lancement d'une requête (asynchrone) HTTP

```
req = new XMLHttpRequest();
...
req.onreadystatechange = function() {
 // instructions de traitement de la réponse };
...
req.open('GET', 'http://url...', true);
req.send(null);
```

## Gestion de la réponse du serveur

```
if (req.readyState == 4) { // réponse reçue
}
else { // pas encore prête
}
...

if (req.status == 200) { // OK !
}
else { // il y a eu un problème avec la requête
}
```

```
function loadDoc() {
 const xhttp = new XMLHttpRequest();
 xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200)
) {
 document.getElementById("demo").innerHTML = t
his.responseText;
 }
 };
 xhttp.open("GET", "ajax_info.txt", true);
 xhttp.send();
}
```

# Ajax : comment ça marche (III)

72

Il y a des subtilités dans l'utilisation des méthodes GET et POST :

GET - attention à l'écriture de l'argument url

déclarer une variable pour récupérer la valeur du champ cible du formulaire et ensuite utiliser la fonction javascript *escape* :

```
var var1 = document.forms[index1].elements[index2].value;
var url = "http://adresse-web-serveur.nom-procedure?var1=" + escape(var1);
```

POST :

Utiliser la propriété *setRequestHeader* de l'objet *XMLHttpRequest*

```
req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
```

Ensuite, préparer la donnée (même démarche que pour url ci-dessus) à envoyer au serveur :

```
var data= "id=" + escape (username) + "&password=" + escape(password);
req.send(data);
```



# Format de données

## XML versus JSON

73

- JSON (*JavaScript Object Notation*) est un format récursif compatible avec JavaScript structuré comme un objet JavaScript sauvegardé dans un fichier.

```
<?xml version="1.0" ?>
<root>
<menu>Fichier</menu>
<commands>
<item>
 <title>Nouveau</value>
 <action>CreateDoc</action></item>
<item>
 <title>Ouvrir</value>
 <action>OpenDoc</action> </item>
<item>
 <title>Fermer</value>
 <action>CloseDoc</action> </item>
</commands>
</root>
```

### Format XML

```
{
 "menu": "Fichier",
 "commandes": [
 {
 "title": "Nouveau",
 "action": "CreateDoc"
 },
 { "title": "Ouvrir",
 "action": "OpenDoc"
 },
 {
 "title": "Fermer",
 "action": "CloseDoc"
 }
]
}
```

### Format JSON

Pour utiliser un fichier JSON:

charger le fichier en tant que texte, (méthode *XMLHttpRequest.responseText*).

utiliser la fonction JavaScript *eval()*:

```
var jtxt = eval('(' + req.responseText + ')');
```

# Exemples d'une requête AJAX

74

- [https://www.w3schools.com/js/js\\_ajax\\_examples.asp](https://www.w3schools.com/js/js_ajax_examples.asp)
  - ▣ Plusieurs exemples : requête simple, requêtes sur des entêtes, charger un fichier XML, requêtes sur des serveurs web (PHP, ASP), requêtes DB

# Accès aux APIs

## API Fetch

75

- ❑ Une interface pour récupérer des ressources
- ❑ A un objectif similaire à XMLHttpRequest très utilisé pour l'approche AJAX.
- ❑ Fonctionnalités plus souple, plus puissante
- ❑ Modèle requête <-> réponse
- ❑ Définition générique pour les objets de type Request et Response
  - Permet une utilisation étendue de ces objets :
    - service workers,
    - l'API Cache,
    - Autres programmes ou mécanismes qui manipulent ou modifient des requêtes et des réponses.

## Traitement synchrone

```
const btn = document.querySelector('button');
btn.addEventListener('click', () => {
 alert(`Vous avez cliqué sur moi !`);

 let pElem = document.createElement('p');
 pElem.textContent = `Il s'agit d'un paragraphe nouvellement ajouté.`;
 document.body.appendChild(pElem);
});
```

- Un seul thread qui exécute les instructions, une instruction peut s'exécuter à la fois, et tout le reste est bloqué jusqu'à la fin d'une opération.

## □ Asynchrone

```
let response = fetch('myImage.png'); // la récupération est asynchrone
let blob = response.blob(); // t returns a promise that resolves with a Blob
// afficher votre blob d'image à l'écran d'une manière ou d'une autre
```

## □ Deux types :

- ▣ Fonctions de rappels asynchrones(callbacks)
- ▣ Promesses (Promises).



# Un langage orienté objet

79

- Un objet est une encapsulation de variables appelées propriétés et de fonctions appelées méthodes.
- ```
let person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```
- Les propriétés sont des boîtes contenant des valeurs primitives, des méthodes et/ou des objets.
- Les valeurs primitives peuvent être prises dans les types primitifs (Undefined, Null, Boolean, Number ou String).
- Une méthode est simplement une fonction associée à un objet.

ES5 ne supporte pas la notion de classe habituelle des langages orientés objet. ES6 (ECMAScript2015) a introduit les classes.

Les objets ne sont pas des instances de classes, mais sont chacun équipés de constructeurs permettant de générer leurs propriétés, et notamment une propriété de prototypage qui permet d'en générer des objets héritiers personnalisés.

- Tout objet possède un ensemble de propriétés
 - ▣ `<objet>.<propriété>`
- Pour définir une propriété :
 - ▣ Lui associer une valeur (`eleve.nom = "Lutton"`);
 - ▣ On peut aussi le voir comme un élément d'un tableau (`eleve["nom"] = "Lutton"`);
- Une propriété peut être aussi un autre objet.
Il est possible de rajouter des propriétés dynamiquement.
- Une méthode est une fonction associée à un objet
 - ▣ `<objet>.<méthode> = <fonction>`
 - ▣ `this` est utilisé pour référencer l'objet courant
 - ▣ `with` est utilisé quand on veut exécuter plusieurs instructions avec un objet.

```
with(document)
{
  open();
  write("Cette page a pour titre " + title);
  close();
}
```


- On peut créer de nouveaux objets en JavaScript:

- ▣ Définir un type d'objet en créant une fonction
- ▣ Créer une instance en utilisant l'opérateur ***new***

- Un exemple :

```
function eleve(nom, prenom) {  
  this.nom = nom;  
  this.prenom = prenom;  
}  
P = new eleve ("Goldman", "Jean-Jacques");
```

- ▣ Copie d'objets :

L'affectation classique n'est pas suffisante;

Pour copier un objet il faudra créer un nouveau objet avec *new*, qui instancie une nouvelle zone mémoire.

- Soit r un objet défini avec deux propriétés : $r.largeur$ et $r.hauteur$.

- ▣ Pour calculer la surface du rectangle :

```
function calculerAireduRectangle(r) {return r.largeur*r.hauteur;}
```

Ou sinon:

```
function Rectangle(l,h) {  
  this.largeur = l;  
  this.hauteur = h;  
  this.aire= function () {return this.largeur*this.hauteur;}  
}
```

Aucune des solutions n'est optimale!!

Car avec la deuxième solution, au niveau mémoire la méthode « aire » n'est pas partagée mais il existe plusieurs versions : une version par instance

Une propriété particulière *prototype*

```
function Rectangle(l,h) {  
  this.largeur = l;  
  this.hauteur = h;  
}
```

L'objet prototype est destiné à détenir les méthodes et les propriétés qui peuvent être partagées par toutes les instances.

```
Rectangle.prototype.aire= function () {return this.largeur*this.hauteur;}
```

L'objet prototype est associé au constructeur et tout objet que celui-ci initialise hérite exactement du même ensemble de propriétés du prototype.

- ES6 introduit les classes en JavaScript :
 - ▣ On utilise le mot clé « class » pour définir une classe.
 - ▣ Les propriétés de la classe sont initialisées à l'intérieur d'une méthode « constructor »
- Exemple de définition d'une classe

```
class Uv {  
    constructor(name) {  
        this.nom = name;  
    }  
    effectif () {  
        return 107;  
    }  
}
```

- Exemple d'instanciation d'un objet en utilisation notre classe

```
uv= new Uv("AI13");  
Uv.effectif();
```

Remontée des déclarations (hoisting)

85

- ❑ Les déclarations de fonctions sont remontées dans le code.
- ❑ Ce n'est pas le cas pour les déclarations de classes. Ainsi, il est nécessaire de déclarer la classe avant de l'instancier. Dans le cas contraire, on obtient une `ReferenceError` :

```
const p = new Rectangle(); // ReferenceError
```

```
class Rectangle {}
```

- L'héritage est possible.
 - ▣ Le mot clé **extends**
 - ▣ La méthode **super ()** :
donne accès aux propriétés
et méthodes de la classe
mère.

```
class Car {  
  constructor(name) {  
    this.brand = name;  
  }  
  
  present() {  
    return 'I have a ' + this.brand;  
  }  
}  
  
class Model extends Car {  
  constructor(name, mod) {  
    super(name);  
    this.model = mod;  
  }  
  show() {  
    return this.present() + ', it is a ' + this.model  
  }  
}  
  
mycar = new Model("Ford", "Mustang");  
mycar.show();
```

Les fonctions fléchées

87

```
hello = function() {  
  return "Hello World!";  
}
```



```
hello = () => {  
  return "Hello World!";  
}
```

- Fonction fléchée : est une définition avec une syntaxe simplifiée (courte)
- Si la fonction a un seul return et ce return est une valeur :
 - ▣ `hello = () => "Hello World!";`
- On peut passer des paramètres via les parenthèses :
 - ▣ `hello = (val) => "Hello " + val;`
- On peut oublier les parenthèses, si on a un seul paramètre à passer :
 - ▣ `hello = val => "Hello " + val;`
- This
 - ▣ Dans une définition régulière d'une fonction, « this » représente l'objet qui a appelé la fonction.
 - ▣ Dans une fonction fléchée, « this » représente l'objet `Header`, peu importe qui a appelé la fonction

- ❑ <https://www.w3schools.com/js>
- ❑ <http://www.w3.org/>
- ❑ [*Traduction française: Référence du DOM Gecko*](http://developer.mozilla.org/en/docs/Gecko_DOM_Reference)
- ❑ <http://www.commentcamarche.net/>
- ❑ <http://www.developpez.com/>
- ❑ <http://pagesperso-orange.fr/arsene.perez-mas/javascript/cours/document.htm>
- ❑ <http://www.jmdoudoux.fr/java/dej/chap-ajax.htm>