# Le Javascript

# Introduction

## Définition

Le Javascript est un petit langage de script qui, incorporé aux balises HTML ou XHTML (E**X**tensible **H**yper**T**ext **M**arkup **L**anguage, fondé sur la syntaxe définie par XML, plus récente, mais plus simple que celle définie par SGML sur laquelle repose HTML et qui oblige une structure plus strict et moins permissive pour être validée), permet d'effectuer des opérations sur les éléments de la page HTML qui le contient. Ce langage est inspiré plus ou moins directement du langage C.

Grâce au modèle objet implémenté, ce langage nous permet d'accéder à tous les éléments d'une page HTML, de les modifier, ou de prendre en compte n'importe quelle action de l'utilisateur sur cette page. Il peut ouvrir par la suite une porte aux développeurs web pour passer à des langages plus complets comme le Java, dont la syntaxe est très proche du Javascript.

Le principal avantage de ce langage est dû au fait qu'il est interprété par la machine cliente, qui va afficher la page par l'intermédiaire du navigateur. Toutes les actions qu'il va effectuer seront donc prises en compte immédiatement, sans que l'on ait besoin de recharger la page, ce qui permet un dynamisme important et une réponse immédiate aux actions de l'utilisateur.

Le HTML avec sa philosophie de structuration de contenu est essentiellement statique. Le besoin s'est donc fait sentir très rapidement d'y ajouter un peu de mouvement et surtout d'interactivité avec l'utilisateur comme par exemple :

- animer du texte ou des images,
- réagir à l'action de la souris,
- détecter le type et la version du navigateur,
- vérifier la saisie dans les formulaires,
- effectuer des calculs simples,
- demander une confirmation,
- afficher la date et l'heure,
- gérer les menus de navigation,
- rediriger le visiteur vers une autre page,
- etc …

## Bref historique

Le langage Javascript est apparu en 1995 avec le navigateur Netscape 2, créant à l'époque un vif intérêt et une petite révolution chez les développeurs et utilisateurs du web.

Microsoft, principal concurrent de Netscape Corporation à l'époque et alors en retard technologiquement dans le domaine de l'Internet, a intégré rapidement Javascript au navigateur Internet Explorer 3. En effet, il ne fallait pas de licence pour utiliser le Javascript. Par contre, il n'était pas question pour Microsoft d'acheter une licence à Netscape pour en adapter le code. C'est alors que JScript est né, variante du Javascript selon Microsoft, qui reprenait la plupart des fonctions du Javascript et en élargissait le champ d'application.

Par soucis de compatibilité et face aux problèmes rencontrés, les deux firmes ont fini par accepter de participer à une standardisation. L'organisme choisi fut l'ECMA (European Computer Manufactures Association). En 1997, le Javascript a été standardisé sous le nom ECMAScript sous la référence ECMA-262. Dans la réalité, cette standardisation n'a apporté que la syntaxe de base mais chaque éditeur a continué ses développements indépendamment l'un de l'autre.

Durant le long déclin de Netscape, différentes versions de Javascript sont apparues. Parallèlement à l'évolution (sans révolution) du langage est apparu un autre facteur de divergence entre les éditeurs des navigateurs, dans la façon de définir et d'accéder à la structure de modèle objet du langage.
Le W3C a réagi à cette incompatibilité en créant le DOM (Document Object Model). Cette standardisation a été globalement bien suivie, mais il subsiste encore des différences notoires d'interprétation que nous verrons plus tard.

## Les limites du langage

Globalement, Javascript est bien reconnu par tous les navigateurs. Pour des applications assez basiques, le problème de compatibilité ne se posent pas. Néanmoins, dès que l'on passe à des applications plus sophistiquées, il s'avère parfois utile d'écrire un code spécifique adapté à certains navigateurs et éventuellement à chaque version de ces derniers.

Le Javascript étant un langage interprété côté client, des problèmes de sécurité se sont rapidement posés qui ont été paliés par la suppression de fonctionnalités. Mais cette relative sécurité a un prix qui se traduit par de sévères limites techniques dont la principale est qu'il ne permet pas de lire et d'écrire sur le disque dur ou sur tout autre périphérique du visiteur. La seule exception est celle qui permet à Javascript d'interagir avec la zone réservée aux cookies.
En outre, aucune instruction n'est prévue en Javascript pour générer une connexion vers un autre ordinateur ou un serveur. Cela n'est possible que par la création d'un objet XMLHTTPREQUEST qui est un élément indispensable à la mise en place d'une application de type AJAX, mais nous aborderons cela plus tard

# Dans le vif du sujet

## La déclaration du langage

Javascript est un langage interprété par le navigateur. Il faut donc informer ce dernier que le code qu'il risque de rencontrer est du Javascript.
Au niveau des balises META, la validation XHTML1.0 Strict imposait de déterminer le langage principal utilisé dans la page mais aussi les langages complémentaires. Cette déclaration se faisait par l'intermédiaire de la ligne :

*<meta http-equiv="Content-Script-Type" content="text/javascript" />*

Cette balise signale que la page comporte ou peut comporter un script et que ce script, en mode texte, est du Javascript. Elle autorise ainsi l'utilisation du gestionnaire d'évènement via les attributs des balises (*exemple : onclick="javascript:alert('Bonjour');"*).

Le code Javascript sera signalé au navigateur en entourant celui-ci des balises

*<script> … </script>*

Cette balise peut être utilisée à tout endroit du code HTML, mais en général la convention veut qu'on la place dans la partie <head></head> ou en fin de la page HTML juste avant le </body>. Il faut néanmoins se souvenir que le navigateur charge la page HTML de manière séquentielle. Il lit d'abord l'en-tête puis le corps de page, ligne par ligne dans l'ordre d'écriture. Si on fait appel à du code Javascript, alors que celui-ci n'a pas encore été défini dans la page et donc interprété par le navigateur, ce dernier ne peut, en toute logique, pas le prendre en charge et provoque donc une erreur bloquant l'ensemble des scripts suivant.
Dans sa forme stricte, on doit spécifier au navigateur le langage utilisé par le script car il existe d'autres langages pouvant s'implémenter dans une page HTML :

*<script type="text/javascript"> … </script>*

Dans la pratique, on rencontre fréquemment la déclaration abrégée sans le type du fait que le Javascript est devenu le langage de référence des structures HTML5.
En HTML, on a pris l'habitude d'entourer le code Javascript des balises

*<!-- … // -->*

pour cacher le code à l'intention des navigateurs qui ne reconnaissent pas le Javascript. Remarquons que les balises *<!-- … -->* sont des commentaires HTML et les barres obliques *//* du commentaire Javascript.
En XHTML, il faut entourer le code Javascript par :

```
//<![CDATA[
...
//]]>
```

Cette notation, issue du XML, permet d'informer le navigateur de ne pas analyser et interpréter ce qu'il y a entre les marquages. L'analyseur syntaxique va seulement récupérer le contenu de cette section, sans chercher à repérer des balises, des entités ou toute autre spécificité syntaxique XML.
On peut aussi déléguer le code Javascript dans un fichier externe. Dans ce cas, on doit inclure le fichier contenant le source. On utilise également la balise *<script>*, mais avec du contenu vide. L'attribut *src* de cette balise indique l'URL du fichier à inclure (pouvant se trouver sur ce serveur ou sur un autre serveur).
Le nom du fichier Javascript doit posséder l'extension *.js*.

*<script type="text/javascript" src="http://www.domain.tld/js/script.js"></script>*

## La syntaxe de base

Javascript est un langage orienté objet que l'on écrit sous forme de lignes de texte.
Ces lignes de texte sont organisées en instructions, blocs d'instructions et commentaires.
Les instructions sont simplement des lignes de codes contenant une succession de symboles et d'objets. Chaque ligne de code correspond à une instruction.
Toutefois, il est recommandé d'utiliser des séparateurs d'instruction à la fin de chacune d'elles. Le séparateur en Javascript est le point-virgule.

Les groupes d'instructions qui sont placés entre des crochets sont appelés des blocs d'instructions. Ces blocs sont utilisés par exemple dans la déclaration de fonctions ou de blocs conditionnels :

```
function premierScript() {
        var i, str;
        i = 1;
        str = 'Bonjour';
        if( i === 1) {
                alert( str );
        }
}
```

Les commentaires sont des lignes de code qui ne seront pas interprétées par le navigateur. Ils sont délimités par les caractères *//* placés en début de ligne et répétés à chaque ligne.
Dans le cas où vous désireriez placer plusieurs lignes successives de commentaire, vous pouvez également utiliser les caractères ouvrant/fermant */* … */*.

## Les variables

Les variables peuvent être déclarées de deux manières différentes.
La première consiste à utiliser l'instruction explicite *var* qui permet de réserver un espace mémoire pour une variable donnée sans pour autant lui assigner de valeur.
Il est également possible de déclarer une variable de manière implicite sans utiliser cette instruction, simplement en assignant directement une valeur à cette variable.

```
var i;
str = 'Bonjour;
```

*Rappel : Le Javascript est "case sensitive". Il fait donc la différence entre les minuscules et les majuscules. Ainsi, la variable var a est différente de var A.*
*Rappel : Le Javascript repose sur l'alphabet ASCII qui est composé de 128 caractères. Ainsi, le code doit être écrit sans caractères spéciaux. La nomenclature de Javascript impose aux noms de variables de toujours commencer par une lettre, un underscore ou un signe dollar. De même, le nom d'une variable ne peut contenir que des lettres, chiffres, underscores.*
Contrairement à de nombreux langages de programmation, Javascript ne nécessite pas de déclaration de type pour ses variables. Ce dernier est déterminé automatiquement par la dernière valeur qui lui est assignée. Lorsqu'une opération est effectuée entre deux variables, Javascript essaiera toujours de convertir les variables dans un type différent si cela est nécessaire.

### Accessibilité d'une variable

Les variables *globales* sont des variables qui ont été déclarées hors de blocs d'instructions. Elles sont alors accessibles par n'importe quelle fonction ou bloc d'instructions, ainsi que par des instructions globales.

Les variables *locales* sont, quant à elles, déclarées dans des blocs d'instructions ou des fonctions. Seul ce bloc d'instructions ou cette fonction pourra alors y faire appel.

## Les types de données

Javascript dispose de 6 types de données qui peuvent être utilisés :

- *les chaînes de caractères,*
- *les nombres,*
- *les objets,*
- *les booléens,*
- *null,*
- *undefined.*

Il existe des fonctions de conversion de type afin de forcer l'utilisation de l'un ou l'autre.

```
i = 1;
str = '2';
alert( i + str ); // affichera 12
alert( i + parseInt( str ) ); // affichera 3
```

### Les chaînes de caractères

Elles sont constituées d'une suite de caractères quelconques, encadrée par des *guillemets* ou des *apostrophes* (si la chaîne contient le délimiteur, ce caractère peut être échappé par un *backslash* afin d'être interprété comme un simple caractère et non comme un délimiteur de la chaîne). Elle peut avoir n'importe quelle taille.
Les chaînes de caractères en Javascript sont également des objets disposant de propriétés et de méthodes qui leurs sont propres.
Elles se concatènent avec le caractère +.

### Les nombres

Javascript supporte tous types de nombres : entiers, décimaux, ou encore des nombres déclarés sur des bases différentes de la base décimale.
Il existe une valeur spéciale pour ce type de données qui est la valeur *NaN*. Elle indique une valeur non numérique (*Not a Number*).

### Les booléens

Un booléen prend la valeur *true* ou *false*.
Il faut savoir que dans les comparaisons, toute valeur numérique estimée à la valeur zéro est considérée comme valant false.

### Le type null

Ce type de données indique une valeur qui n'a pas de valeur et qui ne signifie rien.
Elle est obtenue notamment lorsque l'on fait appel à une variable ou à un objet qui n'a pas encore été déclaré ou instancié.

### Le type undefined

Une valeur *undefined* est une valeur par défaut qui est assignée à une variable qui a été déclarée, mais à laquelle aucune valeur particulière n'a été assignée.

## Les opérateurs

### Les opérateurs arithmétiques

Les opérateurs arithmétiques sont tous les opérateurs permettant d'effectuer des opérations de calcul sur des variables (addition, soustraction, multiplication, …) ainsi que les opérateurs d'incrémentation.

| Opérateur | Nom | Description |
|---|---|---|
| + | plus | addition |
| - | moins | soustraction |
| * | multiplié | multiplication |
| / | divisé | division |
| % | modulo | reste de la division par |
| = | égal | affectation de la valeur |
| ++ | incrément | incrémentation |
| -- | décrément | décrémentation |

### Les opérateurs logiques

Les opérateurs logiques sont utilisés lors de tests conditionnels. Il renvoient toujours des valeurs booléennes.

| Opérateur | Nom |
|---|---|
| < | inférieur |
| > | supérieur |
| <= | inférieur ou égal |
| >= | supérieur ou égal |
| == | égalité |
| != | inégalité |
| ! | NON Logique |
| && | ET Logique |
| \|\| | OU Logique |
| ?: | Opérateur conditionnel |

| | |
|---|---|
| === | Identité |
| !== | Non identité |

Les opérateurs associatifs

Les opérateurs d'assignation associent deux opérateurs d'affectation. C'est une forme abrégée pour noter les opérateurs de calcul.

| Opérateur | Nom | Description |
|---|---|---|
| = | assignation directe | x = y |
| += | assignation après addition | x = x + y |
| -= | assignation après soustraction | x = x - y |
| *= | assignation après multiplication | x = x * y |
| /= | assignation après division | x = x / y |

Les opérateurs divers

| Opérateur | Description |
|---|---|
| delete | Suppression d'une propriété ou d'un élément |
| typeof | Détermination du type d'une expression |
| void | Évaluation de l'expression et retour de la valeur undefined |

## Les structures conditionnelles

Javascript dispose de plusieurs instructions permettant d'effectuer différents types de contrôles.

### L'instruction if

On peut écrire les tests sous deux formes :

- en ligne si la condition ne permet d'exécuter qu'une seule ligne d'instruction
  *if( i === 1 ) alert( 'La variable vaut : ' + i );*
- en bloc d'instructions

```
if( i !== 0 ) {
        i++;
        alert( 'La variable a été incrémentée' );
}
```

### L'instruction if … else

```
if( i === 1 ) alert( 'La variable vaut : ' + i );
else {
        i++;
        alert( 'La variable a été incrémentée' );
}
```

### La boucle while

Cette instruction permet d'effectuer en boucle un bloc d'instructions tant que la condition est vérifiée :

```
a = 0;
b = 0;
while( a < 5 ) {
        b = b + a;
        a++;
}
```

Attention au test de cette instruction afin de ne pas créer de boucle infinie !

### La boucle do … while

A la différence de la boucle *while*, l'instruction *do … while* va exécuter une première fois le bloc d'instructions avant que le test ne soit effectué.

```
a = 1;
b = 1;
do b = b + a;
while( b == 2 )
```

## La boucle for

La première partie du *for* contient les instructions d'initialisation.
La deuxième partie correspond au test conditionnel (s'il est vrai on continue)
La troisième partie est exécutée à chaque fin de boucle.
Le bloc d'instructions est exécuté lorsque la condition est vraie.

```
a = 0;
b = 0;
for( a; a < 5; a++ ) b = b + a;
```

## La boucle for … in

Cette instruction permet d'exécuter un bloc d'instructions pour chaque élément contenu dans un tableau, ou pour chaque propriété d'un objet.
Par exemple, le code suivant écrira dans le document l'ensemble des propriétés de l'objet *window* :

```
for( item in window ) document.write( item . '<br />' );
```

## L'instruction switch

Pour l'utiliser, nous plaçons les différents cas par des mots-clés *case* dans le bloc de l'instruction *switch*. Seul le bloc contenu dans le cas spécifié est exécuté, à condition d'inclure l'instruction *break* à la fin de chaque cas.

```
a = 1;
b = 1;
switch( a ) {
        case 0:
                b--;
                break;
        case 1:
                b++;
                break;
        case 2:
                b = 0;
                break;
}
```

## Les objets

Javascript est un langage orienté objet. De ce fait, chacun des composants du langage et de l'interpréteur dans le navigateur sont aussi des objets auquel le langage vous permet d'accéder et qu'il vous permet de manipuler.
Il existe une notion de hiérarchie dans les objets Javascript existant.

### Les objets du navigateur

- *navigator* est particulièrement utile pour adapter la modalité d'exécution des scripts lors des divergences d'implémentation de certaines instructions Javascript selon les navigateurs.
- *window* est la fenêtre du navigateur.
- *document* est le document HTML contenu dans cette fenêtre.
  Nous retrouverons dans cet objet, tous les éléments HTML créés dans la page.
  Ainsi, si nous considérons dans la page HTML un formulaire contenant 2 boutons radio, nous utiliserons le chemin suivant pour accéder à la valeur de la première occurrence de ce bouton :
  *(window.)document.form.radio[0].value*
  L'objet *window* n'est pas utile dans l'instruction du fait qu'il est unique.

Cette façon de procéder pour accéder aux éléments HTML est la première manière mise en place par le Javascript. Depuis 1998 et l'apparition du DOM, d'autres procédés ont vu le jour.

## Créer un objet

Un objet est une variable (globale ou locale) de classe *Object*. La classe *Object* est une classe générale Javascript dont tout objet découle.

Un objet se crée par l'intermédiaire de *new Object()*.

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

On peut aussi l'écrire en notation JSON (**J**ava**s**cript **O**bject **N**otation) :

```
var obj = {};
```

Si l'on veut créer une propriété pour cet objet, il suffit de l'initialiser :

```
var obj = new Object();
obj.prop = 'Prop1';
```

ou en JSON

```
var obj = { prop: 'Prop1' };
```

Si l'on veut créer une méthode sur un objet, il suffit de la définir :

```
var obj = new Object();
obj.prop = 'Prop1';
obj.dump = function () {
        alert( this.prop );
}

obj.dump(); // utilise la méthode dump de l'objet obj.
```

ou en JSON

```
var obj = {
        prop: 'Prop1',
        dump : function () {
                alert( this.prop );
        }
};

obj.dump(); // utilise la méthode dump de l'objet obj.
```

Quelques objets intrinsèques

- L'objet *String* permet de déclarer une chaîne de caractères. Comme nous l'avons vu, une chaîne de caractères peut être déclarée directement, sans passer par la fonction constructeur de l'objet.
  L'objet *String* dispose d'une seule propriété principale : *length* (retourne la longueur de la chaîne) et de plusieurs méthodes accessibles comme par exemple :
  - *substring( int debut[, int fin] )* : extrait une sous-chaîne en partant de l'indice *début* et jusqu'à l'indice *fin*
    ```
    var url = 'http://www.domain.tld/maCategorie/maPage.html';
    var protocol = url.substring( 0, 4 ); // retourne http
    ```
  - *indexOf( str sousChaine, int debut )* : retourne la position d'une sous-chaîne dans une chaîne, à partir de la position *debut*. Sinon la méthode retoune *-1*
    ```
    var url = 'http://www.domain.tld/maCategorie/maPage.html';
    var domaine = url.substring( 7, url.indexOf( '/', 7 ) ); // retourne www.domain.tld
    ```
- L'objet *Array* permet de déclarer un tableau Javascript.
  Tout comme l'objet *String*, l'objet *Array* dispose d'une seule propriété et de plusieurs méthodes.

```
var monTableau = [14,86,45,22];
// Ajouter des éléments à la fin du tableau
var longueur = monTableau.push(254,123,500); // retourne 7
// Supprimer le dernier élément du tableau
var supprElement = monTableau.pop(); // retourne '500';
// Supprimer le premier élément du tableau
var supprElement = monTableau.shift(); // retourne '14';
// Ajouter des éléments au début du tableau
var longueur = monTableau.unshift(-1, 0); // retourne 7
// Trier le tableau
monTableau.sort(); // [-1,0,22,45,86,123,254]
// Parcourir le tableau
for( var i=0; i < monTableau.length; i++ ) alert( monTableau[i] );
```

## Quelques méthodes Javascript

Passons en revue quelques méthodes de l'objet *window*:

- *alert()* : affiche une boite de dialogue comportant un message qui reproduit la valeur de l'argument qui lui a été fourni.
  Cette boite de dialogue bloque le programme tant que l'utilisateur n'aura pas cliqué sur OK.
- *confirm()* : affiche une boite de dialogue avec deux boutons; OK et Annuler. En cliquant sur OK, la méthode renvoie la valeur *true* et bien entendu *false* si on a cliqué sur Annuler.
- *prompt()* : propose une boite de dialogue composée d'un champ comportant une entrée à compléter par l'utilisateur.

## Mise en pratique

1. Soit un élève ‘toto’ qui a une série de notes : créer une fonction pour calculer la moyenne des notes et faire afficher cette moyenne dans la page.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice Javascript</title>
	</head>
	<body>
		<h1>Moyenne de Toto</h1>
		<p id="moyenne"></p>
		<script type=”text/javascript”>
		/* <![CDATA[ */
		function calcMoy( t_notes ) {
			var moyenne = 0; // Initialisation de la moyenne
			for( var i = 0; i<t_notes.length; i++ ) { // Pour chaque note,
				moyenne += parseFloat( t_notes[i] ); // Ajout des notes
			}
			moyenne /= parseInt( t_notes.length ); // Calcul de la moyenne

			return moyenne;
		}

		toto = [15,14,16,12,18];
		document.getElementById( 'moyenne' ).innerHTML = calcMoy( toto );
		/* ]]> */
		</script>
	</body>
</html>
```

2. Soit ce même élève auquel on souhaite ajouter autant de nouvelles notes que voulu : pouvoir renseigner les nouvelles notes via une boite de dialogue, stocker ces notes et actualiser la moyenne affichée.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice Javascript</title>
	</head>
	<body>
		<h1>Moyenne de Toto</h1>
		<p id="moyenne"></p>
		<script type=”text/javascript”>
		/* <![CDATA[ */
		function calcMoy( t_notes ) {
			var moyenne = 0; // Initialisation de la moyenne
			for( var i = 0; i<t_notes.length; i++ ) { // Pour chaque note,
				moyenne += parseFloat( t_notes[i] ); // Ajout des notes
			}
			moyenne /= parseInt( t_notes.length ); // Calcul de la moyenne

			return moyenne;
		}
		function isNumber( n ) {
			return !isNaN( parseFloat( n ) ) && isFinite( n );
		}
		function ajouterNote( t_notes, note ) { t_notes.push( note ); }

		toto = [];
		do {
			var saisie = prompt( 'Saisissez une note :', '0' )
			if( saisie!=null && isNumber( saisie ) ) {
				ajouterNote( toto, saisie );
				document.getElementById( 'moyenne' ).innerHTML =
calcMoy( toto );
			} else alert( 'Votre saisie doit être numérique' );
		} while( confirm( 'Ajouter une nouvelle note ?' ) )
		/* ]]> */
		</script>
	</body>
</html>
```

# Le DOM

Le DOM (pour Document Object Model) définit un mode standardisé pour accéder et mettre à jour tous les éléments d'un document HTML, XHTML ou XML.
Il faut préciser que le DOM n'est pas en soi un langage de balise ou de programmation mais simplement une manière de percevoir, de parcourir et de manipuler un document HTML ou XML, en utilisant des méthodes et des propriétés spécifiques.
En résumé, le DOM est simplement l'interface permettant de lier le langage de script utilisé et le code HTML de la page.

Comme pour le langage HTML, le DOM a une spécification proposée par le W3C. La version actuelle est DOM Niveau 3 mais la version suivante est en cours de développement est a déjà été annoncée sans date précise de sortie.

## Le concept de noeud (node)

Selon le DOM, tout composant ou élément d'un document HTML, XHTML ou XML constitue un noeud. Ces différents noeuds sont en relation les uns avec les autres. Il est alors possible de représenter un document selon une structure arborescente.

### La hiérarchisation des noeuds

Les noeuds ont une relation hiérarchique entre eux : si on considère par exemple le code * Texte * nous compterons 2 noeuds, un noeud pour la balise * * et un noeud de texte qui comprend son contenu.
Selon ce mode hiérarchique, tous les noeuds forment l'arborescence ou l'arbre du document. Cet arbre commence par le noeud *document* et se termine par les noeuds *texte* pour les niveaux les plus éloignés, en passant par les noeuds *element* des balises.
Ainsi, nous pourrions représenter la hiérarchie suivante :

*document -> html -> head -> title -> Texte*
*-> body -> p -> Texte*

### Les propriétés de l'objet Node

Les propriétés de relation :

- parentNode: renvoie le noeud parent d'un noeud
- firstChild: renvoie le premier enfant d'un noeud
- lastChild: renvoie le dernier enfant d'un noeud
- childNodes: stocke une liste de tous les noeuds enfant disponibles à partir d'un noeud
- previousSibling: renvoie le noeud frère/soeur précédent d'un noeud
- nextSibling: renvoie le noeud frere/soeur suivant d'un noeud

Les propriétés d'état :

- nodeName: indique le nom du noeud sélectionné
- nodeType: indique le type de noeud rencontré
- nodeValue: permet d'obtenir ou de changer la valeur d'un noeud de type *texte*

## L'accès au DOM

On peut citer par exemple :

- *getElementById* : parcours le document HTML à la recherche d'un noeud unique qui a été spécifié par l'attribut *id* d'un champ HTML
- *getElementsByName* : permet de sélectionner tous les éléments portant un nom donné, spécifié par l'attribut *name*. Le retour de la liste de noeuds est stocké dans un tableau.
- *getElementsByTagName* : parcours le document à la recherche de toutes les balises d'un type spécifique, signalé en argument. Le retour de la liste de noeuds est stocké dans un tableau.
- *querySelector()* : renvoie le premier élément trouvé.
- *querySelectorAll()* : retrouve un tableau d'éléments (éventuellement vide) correspondant au sélecteur CSS fourni.

### La création d'un élément dans l'arborescence

On utilise *document.createElement( '[balise]' )* pour créer l'élément. Ce dernier est ensuite ajouté en tant que fils d'un autre élément parent par *parent.appendChild( fils )*. Le fils est inséré à la fin des éléments fils du parent. Pour l'insérer en tant que frère d'un autre élément, on utilisera plutôt *parent.insertBefore( fils, frere )* qui insère le fils juste avant le frère.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice Javascript</title>
	</head>
	<body>
		<p id="par1">Ceci est le paragraphe 1</p>
		<script type="text/javascript">
		/* <![CDATA[ */
		var p = document.createElement( 'p' );
		p.innerHTML = '<span>Ceci est un paragraphe ajouté et contenu dans une balise span</span>';
		p.id = 'para2';
		document.body.appendChild( p );

		var h = document.createElement( 'h1' );
		var hTxt = document.createTextNode( 'Titre de la page' );
		document.insertBefore( h, p );
		/* ]]> */
		</script>
	</body>
</html>
```

## La suppression d'un élément dans l'arborescence

Pour supprimer un élément fils dans la descendance d'un élément parent, on utilisera *parent.removeChild( fils )*.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice Javascript</title>
	</head>
	<body>
		<p>Cliquer pour supprimer le paragraphe 1</p>
		<p>Cliquer pour supprimer le paragraphe 2</p>
		<p>Cliquer pour supprimer le paragraphe 3</p>
		<script type="text/javascript">
		/* <![CDATA[ */
		var ps = document.querySelectorAll( 'p' );
		for( var i = 0; i < ps.length; i++ ) {
			var p = ps[i];
			p.onclick = function () {
				this.parentNode.removeChild( this );
			}
		}
		/* ]]> */
		</script>
	</body>
</html>
```

# Les gestionnaires d'événements

Les gestionnaires d'événements vont apporter l'élément d'interactivité qui caractérise le Javascript. Par leur intermédiaire, les actions du visiteur mettent en oeuvre les outils de programmation du Javascript.
La gestion de ces événements (déclenchés généralement par l'internaute) fait le lien entre le langage XHTML et le Javascript.

## Listeners pour gérer les événements

Ce sont des méthodes d'interception des événements qui peuvent se produire sur les éléments d'une page HTML. Ils permettent d'effectuer un traitement Javascript lorsqu'un événement donné se produit sur un élément précis de l'arborescence du DOM.
Une forme d'écriture de ceux-ci consiste à utiliser les attributs HTML *onclick, onmouseover, onmouseout, etc* …

```
<p onclick="javascript:this.parentNode.removeChild( this );">Cliquez pour supprimer</p>
```

L'inconvénient de cette méthode est que l'on ne peut pas ajouter un traitement supplémentaire lors du clic sans modifier le code HTML ou le code Javascript déjà écrit.
L'utilisation de la méthode *addEventListener()* va le permettre.

### Utilisation de addEventListener

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice Javascript</title>
		<script type="text/javascript">
		/* <![CDATA[ */
		window.addEventListener( 'load', function ( e ) {
			setTimeout( function() {
				document.getElementById( 'bandeauImg' ).src =
'images/bandeau1.jpg';
				var lien = document.createElement( 'a' );
				lien.href = '';
				lien.id = 'lien';
				lien.innerHTML= 'Cliquer pour modifier l\'image';
				document.body.appendChild( lien );

				var el = document.getElementById( 'lien' );
				el.addEventListener( 'click', function ( e ) {
					var img = document.getElementById( 'bandeauImg' );
					img.src = 'images/bandeau2.jpg';

					e.preventDefault();
				}, false );
			}, 3000 );
		}, false /* si true, useCapture indique que tous les événements du type spécifié seront
dispatchés vers l'EventListener avant d'être envoyé à une cible plus bas dans l'arbre */ );
		/* ]]> */
		</script>
	</head>

	<body>
		<figure id="thumb-wrapper"><img alt="Image de bandeau" id="bandeauImg"
src="images/loader.gif" /></figure>
	</body>
</html>
```

## Mise en pratique

Créer un formulaire avec un contrôle de saisie avant l'envoi et lors de la perte de focus sur les champs.

# L'objet XMLHttpRequest

L'objet XMLHttpRequest permet d'envoyer des requêtes HTTP vers un serveur, de recevoir des réponses et de mettre à jour un partie de votre page web. En mode asynchrone, cette mise à jour se réalise sans avoir à recharger la page et donc de façon totalement transparente pour l'utilisateur.

L'objet XMLHttpRequest s'utilise donc dans une architecture de type client/serveur.
Il est créé par le moteur Javascript du navigateur et est alors utilisé pour effectuer une requête HTTP vers le serveur. La réponse est fournie par ce dernier au navigateur. A l'aide du XHTML, le résultat est ensuite affiché dans le navigateur.

L'avantage des applications AJAX réside principalement dans la diminution de la bande passante car seules les données réclamées sont affichées sans avoir à recharger tout le document.

## Créer un objet XMLHttpRequest

Pour le navigateur Internet Explorer dans ses versions inférieures à 6.0, il fallait passer par des méthodes ActiveX propriétaires pour créer une instance de l'objet. Cela est dû au fait que l'objet XMLHttpRequest a été initialement implémenté en 1999 dans IE 5.0 suite à un développement de Microsoft XML Core Services avant d'être repris par le projet Mozilla comme objet natif de Javascript en 2002. Depuis IE 7.0, le navigateur de Microsoft a à son tour intégré la requête XMLHTTP en tant qu'objet natif de Javascript.

Pour tous les navigateurs récents, l'objet natif XMLHttpRequest est donc utilisé de cette manière :

```
var xhr = new XMLHttpRequest();
```

On peut tester l'objet *window* pour connaitre la bonne déclaration à faire en fonction du navigateur :

```
if( window.XMLHttpRequest ) {
        var xhr = new XMLHttpRequest();
} else if( window.ActiveXObject ) {
        try {
                var xhr = new ActiveXObject( Msxml2.XMLHTTP' );
        } catch ( e ) {
                var xhr = new ActiveXObject( 'Microsoft.XMLHTTP' );
        }
} else {
        alert( 'Votre navigateur ne supporte pas la technologie AJAX.' );
}
```

## Les propriétés et les méthodes

Les propriétés :

- readyState: retourne l'état de la requête
- onreadystatechange: gestionnaire d'événements qui prend en charge les changements d'état de la requête
- status: renvoie le code numérique de la réponse du serveur HTTP
- statusText: renvoie le message lié au code numérique de la réponse du serveur HTTP
- responseText: réponse du serveur sous forme de chaîne de caractères
- reponseXML: réponse du serveur sous forme d'un document XML

Les méthodes :

- open(): initialise la requête selon une série de paramètres fournis en arguments
- send(): effectue la requête, avec éventuellement l'envoi de données
- getAllResponseHeaders(): renvoie l'ensemble de l'en-tête HTTP de la réponse sous la forme d'une chaîne de caractères
- abort(): annule la requête

Exemple :

Au clic sur un bouton, affichez dans une ligne de texte le contenu d'un fichier texte contenant la phrase "Je viens du serveur".

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8" />
		<title>Exercice AJAX</title>

		<script type="text/javascript">
		/* <![CDATA[ */
		function getxhr() {
			var xhr = null;
			if( window.XMLHttpRequest ) {
				xhr = new XMLHttpRequest();
			} else if( window.ActiveXObject ) {
				try {
					xhr = new ActiveXObject( Msxml2.XMLHTTP' );
				} catch ( e ) {
					xhr = new ActiveXObject( 'Microsoft.XMLHTTP' );
				}
			} else {
				alert( 'Votre navigateur ne supporte pas la technologie AJAX.' );
			}
			xhr.onreadystatechange = function () {
				if( xhr.readyState==4 && xhr.status==200 ) { // readyState==4 signifie
		que le fichier a été transmis.
					document.getElementById( 'txt' ).innerHTML = 'Reçu du
			serveur - ' + xhr.responseText;
				}
			}
			xhr.open( 'GET', 'test.txt', true ); // true devient false pour désactiver le mode
		asynchrone.
			xhr.send( null );
		}
		/* ]]> */
		</script>
	</head>
	<body>
		<form action="" name="frmAjax">
			<input onclick="getxhr()" type="button" value="Envoyer la requête" />
			<p id="txt"></p>
		</form>
	</body>
</html>
```

# jQuery

```
<!--[if lt IE 9]>
<script src="http://code.jquery.com/jquery-1.12.0.min.js"></script>
<![endif]-->
<!--[if gte IE 9]><!-->
<script src="http://code.jquery.com/jquery-2.2.0.min.js"></script>
<!--<![endif]-->
```

If you’d like to try jQuery 2.0 on web sites where you still need to support IE 6, 7, and 8, you can use conditional comments. All browsers except old IE will get the second script and ignore the first.