JAVASCRIPT

**Qu'est-ce que le Javascript ?**

Le Javascript est un langage de programmation de scripts orienté objet

Un langage de programmation est un langage qui permet auxdéveloppeurs d'écrire du code source qui sera  
analysé par l'ordinateur.  
Un développeur, ou un programmeur, est une personne qui développe des programmes. Ça peut être un professionnel (un ingénieur, un informaticien ou un analyste programmeur) ou bien un amateur

De même, vous pouvez déclarer et assigner des variables sur une seule et même ligne :var myVariable1, myVariable2 = 4, myVariable3;

En Javascript, nos variables sont typées dynamiquement, ce qui veut dire que l'on peut y mettre du texte en premier lieu puis l'effacer et y mettre un nombre quel qu'il soit, et ce, sans contraintes.

**Tester l'existence de variables avec typeof**Il se peut que vous ayez un jour ou l'autre besoin de tester l'existence d'une variable ou d'en vérifier son type. Dans ce genre de situations, l'instruction **typeof** est très utile, voici comment l'utiliser : **var** number = 2;  
alert(**typeof** number); *// Affiche : « number »***var** text = 'Mon texte';  
alert(**typeof** text); *// Affiche : « string »***var** aBoolean = **false**;  
alert(**typeof** aBoolean); *// Affiche : « boolean »*

**Convertir une chaîne de caractères en nombre**

Le concept est simple : il suffit de convertir la chaîne de caractères en nombre.  
Pour cela, vous allez avoir besoin de la fonction parseInt() qui s'utilise de cette manière : **var** text = '1337', number;  
number = parseInt(text);  
alert(**typeof** number); *// Affiche : « number »*alert(number); *// Affiche : « 1337 »*

**Convertir un nombre en chaîne de caractères**

Il est déjà possible de concaténer un nombre et une chaîne sans conversion, mais pas deuxnombres, car ceux-ci s'ajouteraient à cause de l'emploi du +. D'où le besoin de convertir un nombre en chaîne. V oici comment faire :var text, number1 = 4, number2 = 2;  
text = number1 + '' + number2;  
alert(text); // Affiche : « 42 »

**Combiner les opérateurs  
var** condition1, condition2, result;  
Partie 1 : Les bases du Javascript 34/378   
condition1 = 2 > 8; *// false*condition2 = 8 > 2; *// true*result = condition1 && condition2;  
alert(result); *// Affiche « false »*

Il est bien entendu possible de raccourcir le code en combinant tout ça sur une seule ligne, dorénavant toutes les conditions  
seront sur une seule ligne dans ce tutoriel :var result = 2 > 8 && 8 > 2;  
alert(result); // Affiche « false »

**La structure if pour dire « si »**

voyons de quoi est constitué une condition :  
De la structure conditionnelle **if** ;  
De parenthèses qui contiennent la condition à analyser, ou plus précisément le booléen retourné par les opérateurs conditionnels ;  
D'accolades qui permettent de définir la portion de code qui sera exécutée si la condition se vérifie. À noter que nous plaçons ici la première accolade à la fin de la première ligne de condition, mais vous pouvez très bien la placer comme vous le souhaitez (en dessous, par exemple).

**la fonction confirm()**Afin d'aller un petit peu plus loin dans le cours, nous allons apprendre l'utilisation d'une fonction bien pratique : confirm() ! Son utilisation est simple : on lui passe en paramètre une chaîne de caractères qui sera affichée à l'écran et elle retourne un booléen en fonction de l'action de l'utilisateur ; vous allez comprendre en essayant : **if** (confirm('Voulez-vous exécuter le code Javascript de cette page ?'))

{

alert('Le code a bien été exécuté !');

}

**Les ternaires**

**var** startMessage = 'Votre catégorie : ', endMessage,  
adult = confirm('Êtes-vous majeur ?');  
endMessage = adult ? '18+' : '-18';

***Utilisation de continue***Cette instruction est plus rare, car les opportunités de l'utiliser ne sont pas toujours fréquentes. **continue**, un peu comme  
**break**, permet de mettre fin à une itération, mais attention, elle ne provoque pas la fin de la boucle : l'itération en cours est  
stoppée, et la boucle passe à l'itération suivante.

***Portée des variables de boucle***En Javascript, il est déconseillé de déclarer des variables au sein d'une boucle (entre les accolades), pour des soucis de performance (vitesse d'exécution) et de logique : il n'y a en effet pas besoin de déclarer une même variable à chaque passage dans la boucle ! Il est conseillé de déclarer les variables directement dans le bloc d'*initialisation*, comme montré dans les exemples de ce cours. Mais attention : une fois que la boucle est exécutée, la variable existe toujours, ce qui explique que dans l'exemple précédent on puisse récupérer la valeur de i une fois la boucle terminée. Ce comportement est différent de celui de nombreuxautres langages, dans lesquels une variable déclarée dans une boucle est « détruite » une fois la boucle exécutée

**Les fonctions anonymes**

une fonction, lorsqu'elle est déclarée, n'exécute pas immédiatement  
le code qu'elle contient, elle attend d'être appelée. Or, nous, nous souhaitons exécuter ce code immédiatement ! La solution est  
donc d'utiliser ce couple de parenthèses.

**Les attributs**Pour interagir avec les attributs, l'objet Element nous fournit deuxméthodes, getAttribute() et setAttribute() permettant respectivement de récupérer et d'éditer un attribut. Le premier paramètre est le nom de l'attribut, et le deuxième, dans le cas de setAttribute() uniquement, est la nouvelle valeur à donner à l'attribut. Petit exemple :

pour récupérer ou modifier la valeur d'un champ, on utilisera la propriété value

On peut penser que pour modifier l'attribut class d'un élément HTML, il suffit d'utiliser element.**class**. Ce n'est pas possible, car le mot-clé **class** est réservé en Javascript, bien qu'il n'ait aucune utilité. À la place de **class**, il faudra utiliser className

le nom **for** est réservé lui aussi en Javascript (pour les boucles). V ous ne pouvez donc pas modifier l'attribut HTML for d'un **<label>** en écrivant element.**for**, il faudra utiliser element.htmlFor à la place

***Tester le navigateur***Il est possible via une simple condition de tester si le navigateur prend en charge telle ou telle méthode ou propriété. **<body>  
<div** id="myDiv"**>  
<p>**Un peu de texte **<a>**et un lien**</a></p>  
</div>  
<script>  
var** div = document.getElementById('myDiv');   
**var** txt = '';  
**if** (div.textContent) { *// « textContent » existe ? Alors on  
s'en sert !*txt = div.textContent;  
} **else if**(div.innerText) { *// « innerText » existe ? Alors on  
doit être sous IE.*txt = div.innerText + ' [via Internet Explorer]';  
} **else** { *// Si aucun des deux n'existe, cela est sûrement dû  
au fait qu'il n'y a pas de texte*txt = ''; *// On met une chaîne de caractères vide*} a  
lert(txt);  
**</script>  
</body>**Il suffit donc de tester par le biais d'une condition si l'instruction fonctionne. Si textContent ne fonctionne pas, pas de soucis, on prend innerText

Cela dit, ce code est quand même très long et redondant. Il est possible de le raccourcir de manière considérable :  
txt = div.textContent || div.innerText || '';

**var** paragraph = document.getElementById('myP');  
**var** first = paragraph.firstChild;  
**var** last = paragraph.lastChild;  
alert(first.nodeValue);  
alert(last.firstChild.data);   
first contient le premier nœud, un nœud textuel. Il suffit de lui appliquer la propriété nodeValue (ou data) pour récupérer son contenu ; pas de difficulté ici. En revanche, il y a une petite différence avec notre élément **<strong>** : vu que les propriétés nodeValue et data ne s'appliquent *que* sur des nœuds textuels, il nous faut d'abord accéder au nœud textuel que contient notre élément, c'est-à-dire son nœud enfant. Pour cela, on utilise firstChild (et non pas firstElementChild), et ensuite on récupère le contenu avec nodeValue ou data.

***childNodes***La propriété childNodes retourne un tableau contenant la liste des enfants d'un élément

***nextSibling et previousSibling***nextSibling et previousSibling sont deuxpropriétés qui permettent d'accéder respectivement au nœud suivant et au nœud précédent.

Tout comme pour firstChild et lastChild, sachez qu'il existe les propriétés nextElementSibling et  
previousElementSibling qui permettent, elles aussi, de ne récupérer que les éléments HTML. Ces deux  
propriétés ont les mêmes problèmes de compatibilité que firstElementChild et lastElementChild.

**Créer et insérer des éléments**

La création d'un élément se fait avec la méthode createElement(), un sous-objet de l'objet racine, c'est-à-dire document dans la majorité des cas

On utilise la méthode appendChild() pour insérer l'élément. *Append child* signifie « ajouter un enfant », ce qui signifie qu'il nous faut connaître l'élément auquel on va ajouter l'élément créé

**Cloner, remplacer, supprimer…**

Pour cloner un élément, rien de plus simple : cloneNode(). Cette méthode requiert un paramètre booléen ( **true** ou **false**) : si vous désirez cloner le nœud avec (**true**) ou sans (**false**) ses enfants et ses différents attributs.

Pour remplacer un élément par un autre, rien de plus simple, il y a replaceChild(). Cette méthode accepte deuxparamètres : le premier est le nouvel élément, et le deuxième est l'élément à remplacer. Tout comme cloneNode(), cette méthode s'utilise sur tous les types de nœuds (éléments, nœuds textuels, etc.)

Pour insérer un élément, on utilise appendChild(), et pour en supprimer un, on utilise removeChild(). Cette méthode prend en paramètre le nœud enfant à retirer.

***Utiliser les événements***

**<span** onclick="alert('Hello !');"**>**Cliquez-moi !**</span>**

***Bloquer l'action par défaut de certains événements***

il suffit juste d'ajouter le code **return false**; dans notre événement click :

**<a** href="http://www.siteduzero.com" onclick="alert('Vous avez cliqué !'); return false;"**>**Cliquez-moi !**</a>**

**Les événements au travers du DOM**

**Le DOM-0**

**<span** id="clickme"**>**Cliquez-moi !**</span>**

**<script>  
var** element = document.getElementById('clickme');  
element.onclick = **function**() {  
alert("Vous m'avez cliqué !");  
};  
**</script>**

on définit maintenant les événements non plus dans le code HTML mais directement en Javascript.  
Chaque événement standard possède donc une propriété dont le nom est, à nouveau, précédé par les deuxlettres « on ». Cette  
propriété ne prend plus pour valeur un code Javascript brut, mais soit le nom d'une fonction, soit une fonction anonyme. Bref,  
dans tous les cas, il faut lui fournir une fonction qui contiendra le code à exécuter en cas de déclenchement de l'événement

**Le DOM-2**

**<span** id="clickme"**>**Cliquez-moi !**</span>  
<script>  
var** element = document.getElementById('clickme');  
element.addEventListener('click', **function**() {  
alert("Vous m'avez cliqué !");  
}, **false**);  
**</script>**

la suppression des événements ! Celle-ci s'opère avec la méthode removeEventListener() et se fait de manière très simple :element.addEventListener('click', myFunction, **false**); *// On crée l'événement*element.removeEventListener('click', myFunction, **false**); *// On supprime l'événement en lui repassant les mêmes paramètres*

Toute suppression d'événement avec le DOM-2 se fait avec les mêmes paramètres utilisés lors de sa création ! Cependant, cela ne fonctionne pas aussi facilement avec les fonctions anonymes ! Tout événement DOM-2 créé avec une fonction anonyme est particulièrement complexe à supprimer, car il faut posséder une référence vers la fonction concernée, ce qui n'est généralement pas le cas avec une fonction anonyme

***Bloquer l'action par défaut de certains événements***Eh oui, nous y revenons ! Nous avons vu qu'il est possible de bloquer l'action par défaut de certains événements, comme la  
redirection d'un lien vers une page Web. Sans le DOM-2, cette opération était très simple vu qu'il suffisait d'écrire **return false**;. Avec l'objet Event, c'est quasiment tout aussi simple vu qu'il suffit juste d'appeler la méthode preventDefault() !

**<a** id="link" href="http://www.siteduzero.com"**>**Cliquez-moi !**</a>  
<script>  
var** link = document.getElementById('link');  
link.addEventListener('click', **function**(e) {  
e.preventDefault(); *// On bloque l'action par défaut de cet  
événement*alert('Vous avez cliqué !');  
}, **false**);  
**</script>**