

JavaScript

Les Bases



Table des matières

Introduction.....	4
Avantages.....	5
Limitations.....	6
Ajouts (add-ons) des navigateurs.....	6
Compresser son code javascript.....	7
Les bases de JavaScript.....	7
Principe de base.....	7
La balise <script>.....	7
Attribut type.....	7
Attribut src.....	7
La place de JavaScript dans la page HTML.....	8
Dans une balise <script>.....	8
Dans une balise existante.....	8
Dans un fichier externe.....	8
La séparation des instructions.....	8
Casse de JavaScript.....	8
La balise <noscript>.....	9
Les commentaires JavaScript.....	9
Deux types de données simples.....	9
Les nombres.....	9
Les chaînes.....	10
Le bon usage des guillemets.....	10
Les opération sur les variables.....	10
Les opérations sur les nombres.....	10
Les opérateurs mathématiques.....	10
Le modulo.....	11
La concaténation des chaînes.....	11
Principe.....	12
Création d'une fonction.....	12
Gestion des valeurs.....	13
Les arguments.....	13
La valeur de retour.....	13
Quelques fonctions de JavaScript.....	14
Interaction.....	14
Manipulation de variables.....	15
Principe général.....	15
Les opérateurs de comparaison et les opérateurs logiques.....	15
Opérateurs de comparaison.....	15
Les six opérateurs de base.....	15
Deux autres opérateurs : comparaison de la valeur et du type.....	16
Opérateurs logiques.....	16
La négation.....	16
Et et ou.....	16
Un exemple de fonction renvoyant un boolean.....	17
Principe.....	17

If / else – Si / sinon.....	17
Principe.....	18
While – Tant que.....	18
For – Pour.....	19
Référentiel.....	20
Déclaration.....	20
Type.....	20
Nombre.....	20
Chaînes de caractères.....	21
Booléens.....	21
Tableaux.....	21
Déclaration.....	22
Index.....	22
Tableau à deux dimensions.....	22
Syntaxe.....	23
Invocation.....	23
Passage des Arguments.....	24
Imbrication.....	24
Retour de résultats.....	25
Variables locales et globales.....	25
Opérateur d'affectation.....	27
Opérateurs arithmétiques.....	27
Opérateur de Concaténation.....	28
Opérateurs d'Affectation.....	28
Opérateurs de Comparaison.....	28
Opérateurs Logiques.....	29
Priorité des opérateurs.....	29
Événements possibles.....	30
Syntaxe.....	31
Principaux Gestionnaires d'Événements.....	31
Structure Conditionnelle IF ... ELSE.....	31
If ... else.....	31
Else ... if.....	32
Opérateur ternaire.....	33
Structure conditionnelle SWITCH ... CASE.....	33
Structure itérative FOR.....	34
Structure itérative FOR ... IN.....	35
Structure itérative WHILE.....	35
Structure itérative DO ... WHILE.....	36
Instruction BREAK.....	37
Instruction CONTINUE.....	37

Ce document est divisé en trois parties. La première, l'introduction, présente l'histoire du JavaScript et ses caractéristiques. La seconde, les bases du JavaScript, suit le déroulement du cours permettant de revenir rapidement aux notions abordées ensemble. La troisième, le référentiel, aborde point par point les différentes facettes du JavaScript de façon approfondie.

Introduction

Historique

JavaScript a été développé par Netscape en 1995, pour la version 2.0 de son navigateur. Initialement, il se nommait Livescript. En s'associant avec Sun (qui développe le langage JAVA) Netscape va le rebaptiser JavaScript. Depuis Sun a été racheté par Oracle en 2009.

Pour concurrencer Netscape, Microsoft va développer deux langages. Tout d'abord, un langage de script dérivé de son langage VisualBasic, le VBscript (Visual Basic Scripting Edition : sous-ensemble du langage Visual Basic for Applications, VBA, un langage propriétaire de Microsoft prévu pour être intégré aux produits Microsoft Office®). Ensuite, son propre langage de script, le Jscript.

En 1996, pour éviter les dérives, les éditeurs décident de s'entendre et confient à ECMA (European Computer Manufacturers Association) le soin de normaliser le langage. Cette norme est la ECMA-262(<http://www.ecma-international.org/publications/standards/Ecma-262.htm>, en anglais). On parle donc fréquemment d'ECMAScript. Ce standard voit officiellement le jour en juin 1997.

Après une seconde version de l'ECMAScript en juin 1998 et une troisième en décembre 1999, une quatrième est mise en chantier. Adobe, qui a développé de son côté l'ActionScript (utilisé pour le Flash) propose de faire adopter dans l'ECMAScript un certain nombre de fonctionnalités de l'ActionScript. D'autres entreprises qui soutiennent le développement de l'ECMAScript sont opposées à cette présence de fonctionnalités issues de l'ActionScript. Finalement, la version 4 de l'ECMAScript est abandonnée pour lancer le développement de la version 5 du langage.

Pour faciliter la transmission d'information, Douglas Crockford développe le JSON (JavaScript Object Notation) et le propose à la communauté pour remplacer le XML. Le JSON permet d'économiser 30 % de ressources par rapport au XML. Le W3C ne veut pas reconnaître ce format de données. Finalement, le JSON va être adopté par la communauté et le W3C le reconnaîtra après coup. La réputation de Douglas Crockford augmente fortement. Ce dernier va être invité à présider la commission œuvrant à la réalisation de la version 5 de l'ECMAScript qui sortira en décembre 2009.

Aujourd'hui, tous les principaux navigateurs intègrent des interpréteurs JavaScript. C'est d'ailleurs l'un des axes principaux d'optimisation des navigateurs étant donné la prolifération de l'usage du JavaScript (un historique complet est disponible à la page : <http://fr.wikipedia.org/wiki/JavaScript>).

JavaScript et Java

Attention : JavaScript et Java n'ont rien de commun.

JavaScript	Java
<ul style="list-style-type: none">• Langage interprété (nécessite un interpréteur JavaScript)• Code intégré au HTML• Langage peu typé• Liaisons dynamiques : les références des objets sont vérifiées au chargement• Accessibilité du code• Sûr : ne peut pas écrire sur le disque dur	<ul style="list-style-type: none">• Langage interprété (nécessite la machine virtuelle Java)• Code (applet) indépendant du document HTML et appelé à partir de la page• Langage fortement typé (déclaration du type de variable)• Liaisons statiques : tous les objets doivent exister au chargement• Confidentialité du code

Pour en savoir plus : <http://www.toutjavascript.com/savoir/savoir04.php3>.

Domaines d'applications

JavaScript est principalement utilisé pour :

- La vérification des saisies dans les formulaires.
- Les calculs (TVA, conversion monétaire...)
- L'affichage des dates et heures.
- La création et lecture de cookies.
- La création de menus de navigation (cette fonctionnalité est de moins en moins utilisée avec les possibilités offerte par le CSS3).
- La création d'effets sur les images (diaporamas, rollover, animations...)
- L'utilisation des événements liés à la navigation (survol, clic, chargement des objets de la page...)
- Le lancement de popups.

Et bien d'autres choses...

Avantages et Limitations

Comme tout langage, JavaScript a des avantages et des inconvénients...

Avantages

- JavaScript est rapide :
 - Il n'est pas compilé, il est donc facile et rapide de créer, modifier le code JavaScript (contrairement aux langages compilés qui passent par une phase de compilation pour chaque modification)
 - Il est léger : le code JavaScript intégré à une page HTML ne pèse que quelques octets. Il est donc rapide à charger par le navigateur de l'internaute.
- JavaScript est simple (quoiqu'il soit possible de le compliquer à loisir !) :
 - Son modèle d'objet (DOM) est très simple à manipuler.
 - La lisibilité de son code en fait un langage très rapide à comprendre.

- Un simple éditeur de texte permet de coder du JavaScript.
- JavaScript n'a besoin d'aucune ressource serveur :
 - Étant interprété par le navigateur de l'internaute, il ne demande aucune ressource au serveur web, contrairement aux langages serveur comme ASP, PHP, etc....

Limitations

- JavaScript n'est pas universel. Chaque interpréteur (donc chaque navigateur) est différent. Un même script ne sera donc pas compris de la même façon par IE, Firefox, Safari ou autre... Il n'est donc pas rare de prévoir deux scripts pour le même effet, l'un pour Firefox ou Chrome, l'autre pour IE (on parle de code cross-browser).
- On ne peut pas accéder à l'ordinateur en dehors du navigateur (pas d'accès au matériel).

Le **Cross-browser** est la possibilité pour toute application web, sous format HTML ou programmée avec un langage de script s'exécutant côté client de supporter plusieurs navigateurs web. Avec le temps, la compréhension des navigateurs converge, notamment grâce à l'utilisation des bibliothèques JavaScript.

La contrepartie de l'utilisation de ces bibliothèques est leur poids : une centaine de kilo-octets en moyenne.

Outils de développement

Pour développer en JavaScript, vous pouvez utiliser n'importe quel éditeur de texte ! Lors du cours, nous utiliserons l'éditeur de texte Open Source Notepad++.

Ajouts (add-ons) des navigateurs

Pour nous aider dans le débogage, il existe plusieurs add-ons créés pour les navigateurs qui simplifient grandement la vie des développeurs web.

Pour Firefox (c'est le navigateur avec les meilleurs add-ons, celui avec lequel il est le plus facile de développer) :

- **Firebug** permet de trouver rapidement la partie du code HTML correspondant à un élément, de visualiser les requêtes HTTP, les requêtes HTTP Asynchrones (c.f méthode AJAX), de retrouver les CSS et le déboguer facilement, et de voir le code javascript d'une page. Indispensable.
- **Web developer** possède plusieurs options utiles, complémentaires à Firebug : possibilité de redimensionner la page à des dimensions spécifiques pour tester la compatibilité, possibilité de désactiver totalement le javascript, possibilité de valider sa page rapidement sur le site w3c.

Pour Internet Explorer

- **IE developer toolbar** est le seul add-on utile pour le développement sur internet explorer. Il ressemble à firebug par certains côtés, mais est moins complet et plus complexe d'utilisation.
- Dans les dernières versions, IE intègre nativement un inspecteur de code de type Firebug.

Compresser son code javascript

Il est possible et recommandé de compresser le code JavaScript grâce à des utilitaires qui vont retirer tous les espaces inutiles et les sauts de ligne. Le code résultant devient illisible, mais permet de gagner en rapidité pour le site internet. (Le gain de poids moyen est de 20 %)

- <http://www.julienlecomte.net/yuicompressor/> le plus efficace mais change le nom des variables (rendant le script totalement incompréhensible par un humain). On appelle cela l'obfuscation.
- <http://javascript.crockford.com/jsmin.html> retire tous les espaces et sauts de ligne.

Les bases de JavaScript

Premiers pas : des notions importantes

Principe de base

Créons une page HTML basique :

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <script type="text/javascript">alert("Hello") ;</script>
  </body>
</html>
```

Résultat : au chargement de la page, une boîte de dialogue s'ouvre et affiche « Hello ». Vous venez de créer votre premier script JavaScript.

La balise <script>

Vous l'avez vu dans l'exemple précédent, JavaScript se place dans une balise <script>.

Les attributs possibles de la balise <script> sont :

|| **ATTRIBUT TYPE**

Définit le type de langage de script utilisé. La valeur utilisée pour JavaScript est « text/javascript »

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

|| **ATTRIBUT SRC**

Définit l'URL du script externe :

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

Nous reviendrons plus tard sur les scripts externes. Si vous travaillez sur d'anciens scripts, vous pourrez rencontrer l'attribut **language** qui définit le type de langage de script utilisé. Cette notation a été déclassée par le W3C au profit de l'attribut "type". Les valeurs possibles pour JavaScript sont "JavaScript", « JavaScript1.1 », "JavaScript1.2 », « JavaScript1.3 ».

```
<script language="JavaScript1.3">instructions</script>
```

La place de JavaScript dans la page HTML

|| DANS UNE BALISE <SCRIPT>

Comme dans notre exemple, le script est placé dans le corps ou dans l'entête de la page HTML.

|| DANS UNE BALISE EXISTANTE

On peut placer un script dans l'événement d'un objet HTML. On utilise alors les gestionnaires d'événements. Les gestionnaires d'événements permettent d'associer un déclenchement de script réaction à une action de l'utilisateur sur une balise ou en réaction à un comportement du navigateur. Par exemple ce script sera déclenché après la fin du chargement du document web par le navigateur :

```
<body onload="alert('coucou !');">
```

|| DANS UN FICHIER EXTERNE

Il est enfin possible de placer un script dans un fichier JavaScript (.js) externe.

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

La séparation des instructions

Un bloc d'instructions JavaScript peut contenir autant d'instructions que nécessaire.

Chaque **instruction** doit alors être **terminée par un point-virgule " ; "**.

Sur un bloc ne comprenant qu'une seule instruction, le point-virgule est facultatif. Cependant, il est conseillé de toujours l'utiliser.

Casse de JavaScript

JavaScript est sensible à la casse (case sensitive en anglais). Par exemple, je définis une variable untruc et je lui donne une valeur. Je fais afficher le type de cette variable (typeof) en mettant en majuscule sa première lettre la seconde fois :

```
<body bgcolor="#0000FF">
  <script type="text/javascript">
    <!-- un peu de code javascript -->
    var unTruc ;
    unTruc = "bidule";
    alert(typeof(untruc)) ;
    alert(typeof(Untruc)) ;
  </script>
</body>
```

A l'affichage, nous obtenons ceci :

string

undefined

La variable Untruc n'existe pas ! Elle est non définie.

Pour en savoir plus sur la casse : <http://fr.wikipedia.org/wiki/Sensibilit%C3%A9>

[%C3 %A0 la casse.](#)

La balise <noscript>

Ces balises, ne déclenchent pas d’affichage pour les navigateurs avec interpréteur JavaScript. Les navigateurs ne disposant pas d’interpréteur ou ceux sur lesquels l’interpréteur est désactivé afficheront le contenu des balises <noscript>.

Les commentaires JavaScript

Nous venons de voir que les commentaires HTML (<!-- -->) ne sont pas compris par JavaScript. Heureusement, JavaScript, dispose d’une syntaxe pour ses propres commentaires :

```
<!-- commentaires HTML -->
<script type="text/javascript">
    // Commentaires JavaScript sur une ligne
    /* Commentaires JavaScript
       sur plusieurs lignes */
</script>
```

Les variables

Dans un exemple précédent nous a été présenté une variable. Celle-ci est l’élément de base du JavaScript. En effet, comme tout langage de programmation, le développeur crée des processus permettant de stocker des informations et de les manipuler. Ces informations sont enregistrées dans des variables.

On peut définir une variable comme un espace de stockage dans lequel on peut enregistrer tout type de données : un nombre, une chaîne ou d’autres éléments plus particuliers que nous aborderons plus tard.

Une variable qui n’existe pas ou dans laquelle on a jamais rien enregistré est équivalente à *undefined*.

L’information *null* représente une information vide. Une variable peut contenir *null*. L’information *null* est donc différente de *undefined*.

Deux types de données simples

|| LES NOMBRES

Une variable peut contenir un nombre. Cela peut sembler trivial, mais il faut préciser qu’il s’agit d’un élément sur lequel on peut effectuer des opérations. Cette distinction est importante, car cela le distingue de la chaîne sur laquelle ces opérations sont impossibles.

```
<script type="text/javascript">
    // Quelques exemples de nombres
    var monPremierNombre = 14 ;
    var monSecondNombre = -67 ;
```

```
var monTroisiemeNombre = 4.5 ;  
</script>
```

Vous remarquez qu'il faut utiliser un point et non une virgule pour séparer l'entier de la décimale. On parle de nombre entier (int) ou à virgule (float).

|| LES CHAÎNES

Une variable peut également contenir une chaîne de caractères (string), c'est-à-dire un texte délimité par des guillemets simples ou double. Il faut noter que ce texte n'a aucun sens ou valeur pour l'ordinateur.

```
<script type="text/javascript">  
  // Quelques exemples de chaînes  
  var monPremierTexte = "maison";  
  var monSecondTexte = 'une jolie maison dans la campagne';  
  var monTroisiemeTexte = '56';  
  var monQuatriemeTexte = 'qu'il fait beau !';  
</script>
```

Ici, 56 est une chaîne. Il n'a pas de valeur pour l'ordinateur. Si vous utilisez un guillemet simple dans votre texte, il faut mettre un antislash avant pour l'échapper.

Le bon usage des guillemets

Vous trouverez dans du code JavaScript des éléments contenu dans des guillemets simples ou doubles. Il est impératif de fermer avec les mêmes guillemets qui ont servi à ouvrir la chaîne. En pratique, il faut se donner une règle pour écrire le code et s'y tenir. Je conseille les guillemets simples pour le JavaScript et les doubles dans le html.

```
<body>  
  <div id="contenu"></div>  
</div>  
  <script type="text/javascript">  
    var maVariable = 'un petit texte';  
    alert(maVariable) ;  
  </script>  
</body>
```

Les opération sur les variables

|| LES OPÉRATIONS SUR LES NOMBRES

| Les opérateurs mathématiques

Les quatre opérateurs mathématiques sont disponibles dans JavaScript (+ - * /) :

```
<script type="text/javascript">  
  var monNombre1 ;  
  var monNombre2 ;  
  monNombre1 = 5 + 6 ;  
  monNombre2 = 3 - 7 ;  
  monNombre1 = 5 * (8 + 6) ;  
  monNombre2 = (751 - 254) / 5 ;
```

```
//on peut faire une opération sur une variable pour en modifier la valeur
monNombre1 = monNombre1 + (5 * monNombre2) ;
</script>
```

Les règles que nous connaissons ordinairement en mathématique restent valables. Ainsi, il n'est pas possible de diviser par 0. De plus, pour les opérations, il faut user de parenthèses pour donner un ordre à ses opérations (deux rappels utiles : <http://fr.wikipedia.org/wiki/Parenthèse> et http://fr.wikipedia.org/wiki/Ordre_des_opérations).

| Le modulo

À ces quatre opérateurs s'ajoute le modulo. Il est représenté par le pourcentage. L'opération avec le modulo donne le reste de la division.

```
<script type="text/javascript">
  var monNombre1 ;
  var monNombre2 ;

  //calcul de 7 modulo 2
  monNombre1 = 7 % 2 ;
  //monNombre1 vaut 1

  //calcul de 13 modulo 5
  monNombre2 = 13 % 5 ;
  //monNombre2 vaut 3
</script>
```

|| LA CONCATÉNATION DES CHAÎNES

La concaténation consiste à assembler deux éléments ou plus. On utilise pour cela le signe plus.

```
<script type="text/javascript">
  var maVariable1 = 'Mon chien';
  var maVariable2 = 'mon chat';
  var resultat ;
  resultat = maVariable1 + ' et ' + maVariable2 ;
  //affichera : Mon chien et mon chat
  alert(resultat) ;
</script>
```

Les tableaux

Un tableau (array) est une variable particulière qui contient plusieurs éléments alors que les chaînes et nombres n'en contiennent qu'un seul. Chaque élément ou valeur de ce tableau a un indice. Si je déclare le tableau suivant :

```
var prenom = ["Pierre", "Paul", "Jacques"] ;
```

Pierre a l'indice 0, Paul le 1 et Jacques le 2. Vous remarquez que l'on débute à 0 et non 1.

Pour appeler une valeur, il suffit d'indiquer l'indice :

```
alert(prenom[1]) ;
```

Le résultat affiché est Paul.

Un tableau peut contenir lui-même un tableau :

```
var prenom = ['hommes et femmes', ["Pierre", "Paul", "Jacques"], ["Sophie", "Hélène", "Sandrine", "Chloé"]];
```

Dans le cas d'un tableau contenant un tableau, on appelle une valeur ainsi :

```
alert(prenom[1][2]);
```

Le résultat affiché est *Jacques*.

Les fonctions

Principe

Nous avons vu à plusieurs reprises `alert()` qui affiche le contenu de la parenthèse. Il s'agit d'une fonction.

D'une façon générale, une fonction permet d'effectuer une ou plusieurs actions. Une fonction peut exiger une ou plusieurs valeurs en entrée, mais cela n'est pas obligatoire. Elle peut donner une valeur en sortie, mais cela n'est pas obligatoire.

Bien entendu, JavaScript possède de nombreuses fonctions, mais on peut également créer ses propres fonctions.

Création d'une fonction

Une fonction est schématiquement construite de la façon suivante :

```
var maFonction = function(){  
    action(s);  
}
```

Ce qui nous donne concrètement :

```
var maFonction = function () {  
    var calcul = 5 + 3;  
};
```

La fonction est stockée dans une variable et je l'appelle ensuite de la façon suivante :

```
maFonction();
```

Nous remarquons que :

- la fonction possède toujours un nom.
- Il faut obligatoirement une parenthèse entre fonction et l'accolade ouvrante, le tout sans espace.
- Après l'accolade, fermante, le point-virgule est obligatoire.

Gestion des valeurs

|| LES ARGUMENTS

Une fonction peut demander au choix de celui qui l'écrit : aucun argument, un argument, plusieurs arguments.

Voici un premier exemple avec un argument :

```
var calculFacile = function(valeur){  
    var resultat = valeur * 5 ;  
    alert(resultat) ;  
} ;  
  
calculFacile(15) ;
```

Il sera affiché ici 75.

Voici un exemple avec trois arguments :

```
var calculFacile = function(nom, poidsEnKilo, prixAuKilo){  
    var calcul = poidsEnKilo * prixAuKilo ;  
    var resultat = 'Mon produit : ' + nom + ', pèse ' + poidsEnKilo + ' kg et coûte '  
+ prixAuKilo + ' €.' ;  
    alert(resultat) ;  
} ;  
  
calculFacile("pomme", 5, 12) ;
```

Attention, il faut fournir obligatoirement le nombre d'arguments demandé par la fonction.

|| LA VALEUR DE RETOUR

Si il est possible de choisir le nombre d'arguments en entrée, la fonction peut au plus avoir une valeur de retour.

Par défaut, une fonction ne retourne aucun résultat. Les variables définies dans la fonction ne sont pas disponibles en dehors. Il faut donc utiliser un return pour rendre disponible une information en dehors de la fonction.

```
var calculFacile = function(){  
    var valeur = 3 ;  
    return valeur ;  
} ;  
var info = calculFacile() ;  
alert(info) ;
```

Ici, la fonction retourne un résultat. Ce résultat est stocké dans la variable info. On dit qu'une fonction est équivalente à ce qu'elle retourne.

Voici un second exemple :

```
var calculFacile = function(){  
    var couleur1 = "rouge";  
    var couleur2 = "bleu";
```

```
var couleur3 = "vert";  
var valeur = couleur1 + ' ' + couleur2 + ' et ' + couleur3 ;  
  
return valeur ;  
};  
var retour = maFonction() ;  
alert(retour) ;
```

On obtient *rouge, bleu et vert*. Nous pourrions aussi écrire :

```
var calculFacile = function(){  
    var couleur1 = "rouge";  
    var couleur2 = "bleu";  
    var couleur3 = "vert";  
    var valeur = couleur1 + ' ' + couleur2 + ' et ' + couleur3 ;  
  
    return valeur ;  
};  
alert(calculFacile()) ;
```

On utilise directement le résultat de l'exécution de la fonction *calculFacile* comme argument de la fonction *alert*.

On peut retourner directement une opération :

```
var calculFacile = function(){  
    var nombre1 = 5 ;  
    var nombre2 = 7 ;  
    var nombre3 = 8 ;  
    var nombre4 = 15 ;  
    return valeur = (nombre1 + nombre2) * nombre3 - nombre4 ;  
};  
var retour = calculFacile() ;  
alert(retour) ;
```

On obtient 81.

Quelques fonctions de JavaScript

JavaScript possède de nombreuses fonctions. À titre d'exemple, en voici quelques unes.

|| *INTERACTION*

Nous avons déjà vu la fonction *alert()* qui affiche l'élément contenu entre les parenthèses. Il existe également *prompt()* et *confirm()* :

- *confirm()* : ici, la boîte de dialogue affiche le texte entre parenthèse, mais l'utilisateur ne peut que confirmer (bouton OK) ou annuler (bouton annuler). Selon le choix de l'utilisateur, *confirm* retournera un des deux booléens.
- *prompt*(« mon message ») : affichage d'une boîte de dialogue avec l'argument entre parenthèse affiché (ici *mon message*). Une zone de saisie de texte est disponible. Cette fonction retourne un texte ou null selon le choix de l'utilisateur.

|| MANIPULATION DE VARIABLES

Nous avons vu précédemment qu'une variable peut contenir différents types d'informations. La fonction `typeof()` permet de savoir le type de la variable évaluée (string...).

Les booléens

Principe général

Un booléen est un type de variable qui peut avoir deux états : vrai ou faux (true/false). Ici, l'affichage renvoie *boolean*.

```
var test=false ;  
alert(typeof(test)) ;
```

False équivaut à 0 et true à 1. On peut donc écrire :

```
test=false ;  
// ou  
test=0 ;  
// et  
test=true ;  
// ou  
test = 1 ;
```

Les opérateurs de comparaison et les opérateurs logiques

Très souvent, il est nécessaire de comparer deux variables, ou plus, entre elles. Pour cela, on dispose des opérateurs de comparaisons et des opérateurs logiques.

|| OPÉRATEURS DE COMPARAISON

Ils sont au nombre de huit.

| Les six opérateurs de base

opérateur	sens
==	égal à
!=	différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Quelques exemples :

```
alert(5 == 7) ;  
// affiche false  
  
alert(13 != 6) ;
```

```
// affiche true

alert (47 <= 5) ;
//affiche false

alert("pierre" != "didier") ;
// affiche true
```

Une erreur courante est d'utiliser un seul =. Pour vérifier une égalité, il faut toujours utiliser == ou ===.

| Deux autres opérateurs : comparaison de la valeur et du type

Ces deux opérateurs comparent non seulement la valeur, mais aussi le type. Souvenez-vous de la fonction typeof() qui donne le type la variable.

opérateur	sens
===	contenu et type égal à
!==	contenu ou type différent de

```
alert('13' == 13) ;
// affiche true, car seule la valeur est évaluée.

alert('13' === 13) ;
//affiche false, car une chaîne n'est pas du même type qu'un nombre.
```

||

OPÉRATEURS LOGIQUES

opérateur	sens
!	non
	ou
&&	et

| La négation

En plaçant un ! avant une valeur, il renvoie false si la valeur est true, sinon il renvoie false

```
alert (! true) ;
//affiche false

alert(! false) ;
//affiche true

alert(!"chien") ;
//affiche false
```

| Et et ou

Dans un test avec l'opérateur ou (||), il suffit que l'un des éléments soit vrai pour que true soit renvoyé. Dans un test avec et (&&), il faut que tous les éléments soient vrai pour que true soit renvoyé.

Comme cela a été déjà expliqué précédemment pour les opérateurs mathématique, il faut utiliser également des parenthèses quand coexistent un `||` (ou) et un `&&` (et) (ou plusieurs bien-sûr) :

```
alert(true || false) ;  
// affiche true, car l'un des deux est vrai  
  
alert(true && false) ;  
// affiche false, car il faut que les deux soient vrais  
  
var valeur = false ;  
valeur =! valeur ;  
alert(valeur) ;  
// affiche true.
```

Un exemple de fonction renvoyant un booléen

Il est commode de savoir si une variable contient ou non un nombre. La fonction `isNaN()` permet de savoir si l'argument n'est pas un nombre. Elle renvoie un booléen.

```
test="cinq";  
alert(isNaN(test)) ;  
test2 = 5 ;  
alert(isNaN(test2)) ;
```

Les résultats sont *true* puis *false*.

Les conditions

Principe

Les blocs conditionnels permettent de faire des tests. En fonction du résultat, il sera possible d'influer sur l'exécution du code : si tel chose est vraie, fait ceci sinon fait cela.

If / else – Si / sinon

Le bloc conditionnel le plus utilisée est le bloc `if – else` (si – sinon). Voici schématiquement son fonctionnement :

```
if(variableATester){  
    action qui s'exécute si la variableATester contient true  
}  
else{  
    action qui s'exécute si la variableATester contient false  
}
```

Un exemple concret :


```
var variableATester = (age >= 18) ;  
if(variableATester){  
    alert('Tu es majeur') ;  
}else{  
    alert('Tu es mineur') ;  
}
```

Si on veut affiner les conditions on peut imbriquer les bloc if else (contrairement à d'autre langage le mot clé « elseif » n'existe pas) :

```
var variableATester = (age >= 18) ;  
if(variableATester){  
    alert('Tu es majeur') ;  
}else{  
    var autreVariableATester = ((age<18) && (age>0)) ;  
    if(autreVariableATester){  
        alert('Tu es mineur') ;  
    }else{  
        alert('Valeur d'âge incorrecte') ;  
    }  
}
```

Les boucles

Principe

La programmation vise, entre autre, à automatiser les tâches répétitives. Pour répondre à ce besoin, il y a les boucles. Toutes les boucles reposent sur la répétition d'une action tant qu'une variable contient true ou une variable équivalente (tableau non vide, texte, nombre autre que 0 ,).

While – Tant que

Voici le schéma de cette boucle :

```
while(maValeurATester){  
    action(s) à réaliser  
}
```

Cela nous donne concrètement

```
var age=0 ;  
while(age<18){  
    age = prompt('Donnez votre âge ?') ;  
}  
alert('Vous êtes majeur') ;
```

Dans cet exemple, tant que l'âge entré par l'utilisateur est inférieur à 18, la question est posée. Si le chiffre est supérieur ou égale à 18, le message est affiché.

Il faut être attentif en utilisant cette boucle. Si la condition ne peut être remplie, on peut tomber dans le cas d'une boucle infinie et ne pas pouvoir en sortir. Dans ce cas, le navigateur plante.

For – Pour

On peut utiliser une boucle while d'une façon un peu particulière si l'utilisateur souhaite que la boucle se répète un certain nombre de fois. Il sera nécessaire de recourir à un compteur de la façon suivante :

```
var i = 0 ;  
while(i < 10){  
    Action(s) à réaliser  
    i = i + 1 ;  
} ;
```

Pour cet usage, il existe un autre type de boucle : for(valeur de départ, valeur maximale, incrément). Il s'agit donc d'une simplification syntaxique de la boucle while que nous venons de voir. Elle s'écrit ainsi :

```
for(var i=0 ; i<10 ; i = i +1){  
    action(s) à réaliser  
}
```

Il faut noter :

- il y a toujours trois éléments entre parenthèse séparés par des points-virgules : la valeur de départ, la valeur maximale à partir de laquelle on sort de la boucle et l'incrément (i +1 pourrait s'écrire i++).
- L'incrément de i se fait toujours après la réalisation des actions. Si je mets (i=0 ; i > 0 ; i++), la boucle sera effectuée une fois, car i aura pour valeur 1 une fois les actions réalisées.
- De façon conventionnelle, on utilise toujours i comme valeur à incrémenter pour une boucle for.

Voici un exemple concret :

```
var fruits = ['les pommes','les poires','le melon','la banane','la pastèque','le pamplemousse',  
    'les kiwis','le raisin'] ;  
var texte ;  
for(i=0 ; i<8 ; i++){  
    texte = 'j\'aime ' + fruits[i] + ' ;'  
    alert(texte) ;  
}
```

La boucle aura huit itérations permettant d'afficher successivement chaque valeur du tableau : les pommes, les poires,...

Référentiel

Les variables

Une variable est une zone mémoire réservée (conteneur). Cette variable a un nom et une valeur. Pour nommer une variable, vous pouvez utiliser des lettres, des chiffres et le tiret bas "_". Néanmoins, le premier caractère du nom de la variable est impérativement une lettre. Il n'y a pas de limite sur le nombre de caractères utilisés pour nommer une variable. JavaScript étant sensible à la casse, la variable toto est différente des variables TOTO et Toto !

Déclaration

Pour déclarer variable, on utilise la syntaxe suivante :

```
var nomDeLaVariable ;
```

Plusieurs variables peuvent être déclarées en même temps :

```
var variable1, variable2, variable3 ;
```

Une fois déclarée, il est possible d'affecter une valeur à la variable :

```
var variable1 ;  
variable1=36 ;
```

Il est également possible d'affecter une valeur dès la déclaration :

```
var variable1, variable2=22, variable3="coucou";  
variable1=62 ;
```

La déclaration par le mot-clé var, n'est pas indispensable. Attention toutefois, il peut exister une différence entre une variable préalablement déclarée, et une variable non déclarée.

Type

JavaScript est un langage faiblement typé. Il n'y a pas de déclaration du type. Les variables sont de plusieurs types possibles.

|| **NOMBRE**

Entier (exprimé dans le système décimal, octal, hexadécimal) ou flottant :

```
variableDecimale=30 ;  
variableOctale=036 ; (de 0 à 7)  
variableHexadecimale=0x1F ;  
variableDecimaleNegative=-17 ;  
variableFlottante=125.36 ;
```

On utilise un "0" initial pour spécifier un nombre exprimé dans le système octal, et un "0x" pour le système hexadécimal. Dans l'exemple, les 3 variables variableDecimale, variableOctale, et variableHexadecimale, ont la même valeur.

|| CHAÎNES DE CARACTÈRES

Les chaînes de caractères sont la combinaison de n'importe quels caractères alphanumériques et spéciaux.

Elles sont délimitées par des guillemets " " (ou apostrophes " ' ").

```
variableString="1 chaîne de caractères »;
```

OU

```
variableString='1 chaîne de caractères';
```

L'interpréteur JavaScript considère que les guillemets commencent et terminent la chaîne, il ne les affiche pas :

```
variableString="1 chaîne de caractères »;  
document.write(variableString) ;
```

Comment alors procéder pour l'affichage de : le "JavaScript" est différent de "java" ?

On utilise les apostrophes :

```
var1='le "JavaScript" est différent de "java"';  
document.write(var1) ;
```

Et pour afficher : le "JavaScript" n'est pas "java" ? Avec des guillemets et des apostrophes ?

On utilise le **caractère d'échappement** : anti-slash " \ " :

```
var2='le "JavaScript" n\'est pas "java"';  
var3="le \"JavaScript\" n'est pas \"java\"";  
document.write(var1) ;
```

Le caractère d'échappement informe l'interpréteur que le caractère qui le suit doit être affiché et non pas considéré comme un caractère ouvrant ou fermant la chaîne.

|| BOOLÉENS

Il existe des mots-clé réservés, notamment les booléens **true** et **false**.

```
varA=true ;  
varB=false ;  
varC=1 ;  
varD=0 ;
```

Les booléens peuvent également s'écrire de manière numérique, ainsi, varA équivaut à varC et varB équivaut à varD.

Tableaux

Jusqu'à présent, nous avons parlé de variables "scalaires". Par scalaire on entend une variable simple : text, entier, etc.

Les scalaires ne peuvent avoir à l'instant T, qu'une seule et unique valeur.

Les tableaux sont des agrégats (ensemble) de variables de type quelconque.

Un tableau est composé d'éléments indexés. Chaque élément ayant son propre type, sa propre valeur, et son propre index. L'indice (index) est le repère de l'élément. Le premier indice est 0. Un

tableau composé de 10 éléments aura donc des éléments indexés de 0 à 9.

|| DÉCLARATION

Un tableau est un objet JavaScript qui se déclare ainsi :

```
var nomDuTableau = new Array() ;
```

Il est possible de passer en argument, le nombre d'éléments du tableau. Par exemple un tableau de 4 éléments

```
nomDuTableau = new Array(4) ;
```

Nous pouvons aussi déclarer un tableau sans passer par la déclaration de l'objet :

```
nomDuTableau2 = ["pomme", "poire", "abricot", "orange"] ;
```

|| INDEX

Pour sélectionner un élément, on utilise la syntaxe suivante : `nomDuTableau[0]`

Le premier élément possède l'index 0, le deuxième l'index 1, le troisième l'index 2... le Nième élément possède l'index N-1...

Voici un tableau :

```
var tableauExemple = new Array() ;  
tableauExemple = ["pomme", "poire", "abricot", "orange", true, 1, "toto"] ;
```

Ce tableau a les 4 éléments suivants :

```
tableauExemple[0] = "pomme";  
tableauExemple[1] = "poire";  
tableauExemple[2] = "abricot";  
tableauExemple[3] = "orange";
```

A tout moment, on peut rajouter un élément supplémentaire au tableau, en indiquant ou non l'indice (sans précision l'élément aura l'indice suivant l'indice du dernier élément) :

```
tableauExemple[4] = "citron";
```

Nous pouvons aussi affecter un indice non consécutif à un élément :

```
tableauExemple[7] = "tomate";
```

Le 7e élément (`tableauExemple[6]`) aura la valeur `undefined`.

|| TABLEAU À DEUX DIMENSIONS

Jusqu'à présent, chaque élément des tableaux était un scalaire quelconque. Nous allons voir maintenant que les éléments d'un tableau peuvent être d'autres tableaux :

```
tableau = new Array() ;
```

```
tableau[0] = new Array() ;
tableau[1] = new Array() ;
tableau[0][0] = "pomme";
tableau[0][1] = "poire";
tableau[0][2] = "abricot";
tableau[0][3] = "orange";
tableau[1][0] = "voiture";
tableau[1][1] = "moto";
tableau[1][2] = "bateau";
tableau[1][3] = "avion";
```

Les fonctions

Syntaxe

Afin de réutiliser facilement certaines instructions JavaScript, il est possible de créer des fonctions. Une fonction est un ensemble d'instructions selon la syntaxe suivante :

```
function nomDeLaFonction(argument1, argument2...){
    instructions ;
}
```

On déclare une fonction en utilisant le mot-clé **function** suivi du nom de la fonction.

Le nom d'une fonction est libre, et obéit aux mêmes restrictions que le nom des variables.

Les arguments de la fonction sont indiqués entre les parenthèses.

Attention : les arguments sont facultatifs, mais les parenthèses sont obligatoires.

Le JavaScript est case sensitive (en français sensible à la casse)

Certains mots-clés tels que new, boolean, etc sont des mots réservés que vous ne pourrez pas utiliser pour nommer une fonction.

Enfin, comme en langage C, chaque instruction se termine par un point-virgule ;.

Le bloc d'instructions est délimité par des accolades.

Invocation

```
function ecrire(){
    document.write(« Hello world !" ) ;
}
```

Lorsque l'on place cette fonction dans la page HTML, il ne se passe rien. C'est normal, une fonction définit une série d'instructions, mais ne les déclenche pas.

Il s'agit donc d'invoquer la fonction pour l'utiliser. Pour invoquer une fonction, il suffit de la nommer, et éventuellement lui signaler les arguments à utiliser :

```
<html>
  <head>
    <script type="text/javascript">
      function ecrire(texteAEcrire){
        document.write(texteAEcrire) ;
      }
    </script>
```

```
</head>
<body>
  <script type="text/javascript">
    ecrire (« Hello World ») ;
  </script>
</body>
</html>
```

On a vu que l'interpréteur ne déclenche pas le bloc d'instructions de la fonction lorsqu'il le rencontre. Par contre, lorsqu'il rencontre l'invocation d'une fonction, il recherche la déclaration de celle-ci dans toute la page, puis exécute les instructions associées.

Passage des Arguments

Dans certains cas, il est intéressant de passer des arguments à une fonction.

Pour cela, il est important que la fonction, lors de sa déclaration, prévoie le passage du bon nombre d'arguments.

Par exemple :

```
function ecrire_prenom(prenom){
  document.write (« Bonjour, ») ;
  document.write(prenom) ;
  document.write(" !") ;
}
ecrire_prenom("Marcel") ;
```

Dans cet exemple l'argument passé a pour valeur " Marcel ". Lors de l'invocation, toutes les occurrences de l'argument prénom, seront remplacées par la valeur " Marcel ".

Il est donc possible de changer la valeur de l'argument à chaque invocation :

```
function ecrire_prenom(prenom){
  document.write (« Bonjour, ») ;
  document.write(prenom) ;
  document.write(" !") ;
}
function br(){
  document.write("<br />") ;
}
ecrire_prenom("Marcel") ;
br() ;
ecrire_prenom("Pierre") ;
br() ;
ecrire_prenom("Paul") ;
br() ;
ecrire_prenom("Jacques") ;
```

Imbrication

Les fonctions peuvent être imbriquées entre elles.

Le code précédent peut alors être remplacé par celui-ci :

```
function ecrire_prenom(prenom){
  document.write (« Bonjour, ») ;
```



```
document.write(prenom) ;
document.write(" !") ;
br() ;
}
function br(){
    document.write("<br/>") ;
}
ecrire_prenom("Marcel") ;
ecrire_prenom("Pierre") ;
ecrire_prenom("Paul") ;
ecrire_prenom("Jacques") ;
```

Retour de résultats

Une fonction peut avoir comme rôle de renvoyer un résultat.

```
<script type="text/javascript">
    function faire_la_moyenne(a, b, c){
        var moyenne=(a+b+c)/3 ;
        return moyenne ;
    }
    document.write(faire_la_moyenne(5,8,2)) ;
</script>
```

Une fonction ne peut renvoyer qu'un seul et unique résultat.

Lorsque la fonction renvoie un résultat, l'appel de la fonction sera alors affecté du résultat de la fonction.

Par défaut, une fonction renvoie le booléen false.

Variables locales et globales

Une variable locale est une variable qui ne peut-être utilisée que dans le contexte où elle a été déclarée.

Une variable globale est une variable qui peut-être utilisée à tout moment du script.

Dans cet exemple var1 est globale puisqu'elle est déclarée hors de toute fonction. La valeur est bien affichée dans le document :

```
<script type="text/javascript">
    var var1 ;
    var1 = " Toto " ;
    document.write(var1) ;
</script>
```

Dans cet exemple var1 est locale puisqu'elle est déclarée dans la fonction. Le script " plante " puisqu'il fait référence à une fonction qu'il ne connaît pas. Rien ne s'affiche :

```
<script type="text/javascript">
    function fonction1(){
        var var1 ;
        var1 = "Toto";
    }
```

```
x = fonction1() ;  
document.write(var1) ;  
</script>
```

Enfin, dans cet exemple `var1` est globale : elle n'est pas explicitement déclarée, et est donc considérée comme globale. La valeur est bien affichée dans le document :

```
<script type="text/javascript">  
  function fonction1(){  
    var1 = "Toto";  
  }  
  x = fonction1() ;  
  document.write(var1) ;  
</script>
```

Fichiers JavaScript

La méthode `write()` de l'objet `document` permet d'écrire dans une page HTML. Elle permet également d'écrire du code HTML :

```
<script type="text/javascript">  
  document.write(« normal <b>gras</b> normal ») ;  
</script>
```

Compte tenu de cette faculté de `write()`, il est tout à fait possible de réaliser en JavaScript un code commun à toutes les pages, par exemple :

```
<script type="text/javascript">  
  document.write("<a href=\"index.html\"> Accueil </a> - <a  
href=\"plan.html\"> Plan du Site </a>  
- <a href=\"mailto : toto@toto.fr\">Contact</a>") ;  
</script>
```

Afin de pouvoir réutiliser ce code, nous pouvons créer un fichier JavaScript externe.

Les fichiers JavaScript sont des fichiers de type texte, créés depuis n'importe quel éditeur de texte (les éditeurs HTML permettent de les créer également).

Ces fichiers possèdent l'extension `.js` et ne contiennent que des instructions JavaScript (pas de balises `<script>`).

Pour lier un document HTML à un fichier JavaScript, on utilise la balise suivante :

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

Opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer,...

Opérateur d'affectation

L'opérateur d'affectation est le signe égal " = ".

Il affecte à l'entité placée à sa gauche, la valeur de l'entité (des entités) située(s) à sa droite :
variable = 5 ; // 5 est affecté à la variable

Opérateur	Signification	Exemple	Résultat
=	affectation	var1 = 5	Var1 vaut 5

Opérateurs arithmétiques

Pour : x = 5 et y = 2

Opérateur	Signification	Exemple	Résultat	Précision
+	addition	x+y	7	
-	Négation / soustraction	x-y	3	
*	Multiplication	x*y	10	
/	Division	x/y	2,5	
%	Modulo	x%y	1	
++	Incrémentation	x++ ++x	6 6	Post-incrémentation Pré-incrémentation
--	Décrémentation	x-- --x	4 4	Post-décrémentation Pré-décrémentation

Pour bien comprendre la différence entre post-incrémentation et la pré-incrémentation, réalisons le code suivant :

```
var x1=5, x2=5, y1, y2 ;
y1=x1++;
y2=++x2 ;
document.write(« x1 : " + x1) ;
document.write("<br/> y1 : ") ;
document.write(y1) ;
document.write("<br/> x2 : ") ;
document.write(x2) ;
document.write("<br/> y2 : ") ;
document.write(y2) ;
```

Dans le premier cas (y1=x1++), on affecte la valeur de x1 à y1 AVANT l'incrément de x1. Dans le second cas (y1=++x1), on affecte la valeur de x1 à y1 APRES l'incrément de x1.

Le modulo est le reste de la division entière de deux nombres.

Opérateur de Concaténation

L'opérateur de concaténation est le signe égal " + ".

Pour : x = "Coucou" et y = " !" »

Opérateur	Signification	Exemple	Résultat
+	Concaténation	x+y	Coucou !

```
var str1="bonjour";
var str2="prénom";
var num1=12 ;
var num2=10 ;

document.write(str1+str2) ;
document.write(str1+num1) ;
document.write(num1+num2) ;
```

L'opérateur de concaténation est le même que l'opérateur d'addition. C'est le type des variables qui détermine le traitement de l'opération.

Si l'une des entités de l'opération est une chaîne, l'opération sera une concaténation.

Si toutes les entités de l'opération sont des nombres, l'opération sera une addition.

Opérateurs d'Affectation

L'opérateur d'affectation se place devant le signe '='

Pour : x = 5 et y = 2

Opérateur	Exemple	Signification	Résultat
+=	x+=y	x=x+y	x=7
-=	x-=y	x=x-y	x=3
=	x=y	x=x*y	x=10
/=	x/=y	x=x/y	x=2,5
%=	x%=y	x=x%y	x=1

Opérateurs de Comparaison

Un opérateur de comparaison compare ses opérandes et renvoie une valeur logique en fonction du résultat de la comparaison. Les opérandes peuvent être des nombres, des chaînes, des valeurs logiques ou des objets.

Pour : x = 5, y = 2, A = true et B = 1 ;

Opérateur	Signification	Exemple	Résultat	Précision
	Égal à	x==5 y==5 A==B	true false true	A==B et A===B true équivaut à 1, mais n'est pas strictement égal à 1
	Strictement égal à	x===5 y===5 A===B	true false false	
!=	Différent de	x !=5 y !=5 A !=B	false true false	
!==	Strictement différent de	x !==5 y !==5 A !==B	false true true	
<	Inférieur à	x<5 y<5	false true	
<=	Inférieur ou égal à	x<=5 y<=5	true true	
>	Supérieur à	x>5 y>5	false false	
>=	Supérieur ou égal à	x>=5 y>=5	true false	

Les opérateurs logiques renvoient toujours un booléen (true, false). Ils sont utilisés dans le test de structures conditionnelles (voir plus loin).

Opérateurs Logiques

Les opérateurs logiques sont généralement employés avec des valeurs booléennes (ou logiques)

Opérateur	Signification	Exemple	Résultat
&&	ET logique	X>5 && X<10	Vrai si X vaut 6,7,8 ou 9
	OU logique	X<3 X>6	Faux si X vaut 3,4,5 ou 6
!	NON logique (NOT)	!(X<5)	Vrai si X est supérieure ou égal à 5

Priorité des opérateurs

Comme en mathématiques, les opérateurs s'effectuent dans un ordre de priorité spécifique. Ci-dessous, l'ordre du priorité : du degré de priorité le plus faible ou degré de priorité le plus élevé. Dans le cas d'opérateurs de priorité égale, c'est le sens de lecture qui prime (de gauche à droite). Il ne faut pas hésiter à utiliser les parenthèses, en cas de doute.

Priorité	Opérateur	Signification
Faible	,	Séparateur de liste
	= += -= *= /= %=	Affectation
		OU logique
	&&	ET logique
	== === != !==	Égalité, différence
	< <= > >=	Relationnels
	+ -	Addition, soustraction, concaténation
	* /	Multiplication, division
	! ++ --	Négation, incrémentation
	()	Appel de fonction, parenthèses
Elevé	. []	Membre

Gestionnaire d'événements

Les Event-Handler (gestionnaires d'événement) sont un maillon important entre HTML et JavaScript. Les gestionnaires d'événements sont notés sous forme d'attributs dans des repères HTML. Étant donné qu'il s'agit de parties intégrantes qui apparaissent en HTML, le consortium W3C les a entre-temps acceptés dans le standard HTML

Événements possibles

Il existe plusieurs événements :

- Lorsque l'internaute clique sur un élément : **Click**
- Lorsque le pointeur de la souris survole un élément : **MouseOver**
- Lorsque le pointeur sort du survol d'un élément : **MouseOut**
- Lorsque l'utilisateur presse sur le bouton de la souris : **MouseDown**
- Lorsque l'utilisateur relâche le bouton de la souris : **MouseUp**
- Lorsque la page est ouverte par le navigateur : **Load**
- Lorsque la page est quittée : **Unload**
- Lorsque la fenêtre est déplacée : **Move**
- Lorsque la fenêtre est redimensionnée : **Resize**
- Lorsque le chargement de l'élément provoque une erreur : **Error**
- Lorsqu'un élément de formulaire prend le focus (devient la zone active) : **Focus**
- Lorsqu'un élément de formulaire perd le focus (n'est plus la zone active) : **Blur**
- Lorsque la valeur d'un élément de formulaire est modifiée par l'internaute : **Change**
- Lorsque l'internaute sélectionne (surbrillance) tout ou partie du contenu d'un champ de texte : **Select**
- Lorsque l'internaute, clique sur le bouton "submit" d'un formulaire : **Submit**
- Lorsque l'internaute, clique sur le bouton "reset" d'un formulaire : **Reset**
- A l'arrêt intempestif du chargement d'une image : **Abort**

- Lorsque l'utilisateur presse sur une touche du clavier : **KeyDown**
- Tant que l'utilisateur maintient une touche du clavier pressée : **KeyPress**
- Lorsque l'utilisateur relâche sur une touche du clavier : **KeyUp**

Syntaxe

Pour utiliser les événements ci-dessus, on utilise la syntaxe suivante :

```
onEvenement="instruction(s) "
```

onEvenement est le gestionnaire d'événements. Celui-ci est placé à l'intérieur de la balise à laquelle il correspond.

Exemples :

```
<input type="button" value="Clic" onclick="alert('Bravo !!!') ;">  
OU  
<body onload="alert('bonjour !') ;" onunload="alert('au revoir !') ;">
```

Attention : Le HTML5, exigeant que tous les attributs de balises soient écrit en minuscule, il est obligatoire d'utiliser la syntaxe minuscule pour les gestionnaires d'événements !!!

Principaux Gestionnaires d'Événements

Événement	Gestionnaire	Objets affectés
Click	onClick	Images, liens, et tous les champs de formulaires
Load Unload	onLoad onUnload	Documents
MouseOver MouseOut	onMouseOver onMouseOut	Images, liens, et tous les champs de formulaires
Focus Blur	onFocus onBlur	Fenêtres et tous les champs de formulaire
Change	onChange	Tous les champs de formulaire (sauf boutons !)
Select	onSelect	Champs text et textarea
Submit	onSubmit	Formulair

Structures de Contrôle

Structure Conditionnelle IF... ELSE

|| **IF... ELSE**

L'instruction if() permet de tester une condition, et, selon le résultat de ce test, de faire telles ou

telles instructions.

La syntaxe est la suivante :

```
if(condition){  
    instructions1 ; // si la condition est remplie (true)  
else {  
    instructions2 ; // si la condition n'est pas remplie (false)  
}
```

On peut traduire cette structure par :

Si (ceci) est vrai { fait ceci } sinon { fait cela }

Afin de tester une condition, on utilise les opérateurs de comparaison et les opérateurs logiques

Exemple :

```
<script type="text/javascript">  
function test(){  
    chiffre_a_tester = document.formulaire.chiffre.value ;  
    if(chiffre_a_tester<5){  
        alert(« Bravo !" ) ;  
    }else{  
        alert(« On a dit : inférieur à 5 !" ) ;  
    }  
}  
</script>  
<form name="formulaire" method="post" action="#">  
Donnez un chiffre inférieur à 5  
<input type="text" name="chiffre">  
<input type="button" value="Go" onclick="test() ;">  
</form>
```

Attention : Le bloc d'instructions else n'est pas obligatoire.

|| ELSE... IF

Il est possible de faire des conditions en cascade :

Exemple :

```
<script type="text/javascript">  
function test(){  
    nombre_a_tester = document.formulaire.nombre.value ;  
    if(chiffre_a_tester<0){  
        alert(« Ce nombre est négatif ») ;  
    }else if(nombre_a_tester>0){  
        alert(« Ce nombre est positif ») ;  
    }else{  
        alert(« Ce nombre est nul ») ;  
    }  
}  
</script>  
<form name="formulaire" method="post" action="">  
Donnez un chiffre inférieur à 5  
<input type="text" name="nombre">  
<input type="button" value="Go" onclick="test() ;">  
</form>
```


|| OPÉRATEUR TERNAIRE

L'opérateur ternaire est un opérateur qui peut se substituer à la structure if... else.

Sa syntaxe est :

```
(condition) ? instructions1 : instructions2 ;
```

Notre exemple de if... else, peut alors être remplacé par :

```
<script type="text/javascript">
function test(){
    chiffre_a_tester = document.formulaire.chiffre.value ;
    (chiffre_a_tester<5) ? alert(« Bravo !") : alert(« On a dit : inférieur à
5 !!!") ;
}
</script>
<form name="formulaire" method="post" action="">
Donnez un chiffre inférieur à 5
<input type="text" name="chiffre">
<input type="button" value="Go" onclick="test() ;">
</form>
```

|| STRUCTURE CONDITIONNELLE SWITCH... CASE

Cette structure permet de tester plusieurs conditions concernant la même expression.

Sa syntaxe est :

```
switch(expression){
    case valeur1 : instructions1 ; break ;
    case valeur2 : instructions2 ; break ;
    case valeurN : instructionsN ; break ;
    ...
    default : instructions4 ;
}
```

L'expression est une variable. Celle-ci est comparée à la succession des valeurs (constantes). A chaque fois que la variable est égale à la valeur testée (case), les instructions suivantes seront réalisées.

L'instruction break permet de sortir du switch afin de ne pas faire les tests suivants.

Exemple :

```
<script type="text/javascript">
function calendrier(){
    numero_mois = Number(document.formulaire.mois.value) ;
    switch(numero_mois){
        case 1 :    nom_mois="janvier"; break ;
        case 2 :    nom_mois="fevrier"; break ;
        case 3 :    nom_mois="mars"; break ;
        case 4 :    nom_mois="avril"; break ;
        case 5 :    nom_mois="mai"; break ;
        case 6 :    nom_mois="juin"; break ;
        case 7 :    nom_mois="juillet"; break ;
        case 8 :    nom_mois="août"; break ;
        case 9 :    nom_mois="septembre"; break ;
    }
}
```

```

        case 10 : nom_mois="octobre"; break ;
        case 11 : nom_mois="novembre"; break ;
        case 12 : nom_mois="décembre"; break ;
    }
    alert(nom_mois) ;
}
</script>
<form name="formulaire" method="post" action="">
Quel mois :
<input type="text" name="mois">
<input type="button" value="Go" onclick="calendrier() ;">
</form>

```

Structure itérative FOR

Cette structure permet de réaliser des instructions tant que la condition perdure, l'expression évaluée s'incrémentant automatiquement.

Sa syntaxe est :

```

for(expression initiale ; condition ; incrément){
    instructions ;
}

```

L'expression initiale est la valeur de la variable de référence au départ de la boucle.

La condition est un test portant sur la variable. La condition est testée à chaque boucle. Lorsque la condition n'est plus remplie, on sort de la boucle.

L'incrément est exécuté à chaque fin de boucle.

Exemple :

```

for(var compteur = 0 ; compteur < 3 ; compteur++){
    document.write(compteur) ;
}

```

Actions de l'interpréteur	Calcul	Valeur de variable
Il évalue l'expression initiale (variable)		0
Il teste si la condition est remplie	0<3 => vrai	0
Il réalise les instructions (affichage de 0)		0
Il incrémente la variable	0++ => 0+1 => 1	1
Il évalue l'expression initiale (variable)		1
Il teste si la condition est remplie	1<3 => vrai	1
Il réalise les instructions (affichage de 0)		1
Il incrémente la variable	1++ => 1+1 => 2	2
Il évalue l'expression initiale (variable)		2
Il teste si la condition est remplie	2<3 => vrai	2
Il réalise les instructions (affichage de 0)		2

Il incrémente la variable	2++ => 2+1 => 3	3
Il évalue l'expression initiale (variable)		3
Il teste si la condition est remplie	3<3 => faux	3
Il sort de la boucle et continue le reste du script		

Structure itérative FOR... IN

La boucle for...in est spécifique aux objets. Elle permet de passer en revue toutes les propriétés d'un objet. Cette boucle est rarement utilisée dans un script, mais elle est très utile au moment du développement pour parcourir les objets intégrés.

Sa syntaxe est :

```
for(variable in tableau){ ou (variable in objet)
    instructions ;
}
```

La boucle for...in n'a pas de condition. Elle s'arrête lorsque toutes les propriétés ont été parcourues.

Cela donne :

```
<html>
  <head><title>Test</title></head>
  <body>
    <script type="text/javascript">
      <!--
      var sortie = "";
      for (var propriete in document){
        sortie = sortie + « document. » + propriete + ": " +
        document[propriete] + "<br>";
        document.write("<h1>Propriétés de l'objet <i> document
        <\i><\h1>") ;
        document.write(sortie) ;
      }
      // -->
    </script>
  </body>
</html>
```

Structure itérative WHILE

Cette structure permet de réaliser des instructions tant que la condition perdure, sans incrémentation définie.

Sa syntaxe est :

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

La condition est un test portant sur la variable. La condition est testée à chaque boucle. Lorsque la condition n'est plus remplie, on sort de la boucle.

Attention : il n'y a pas d'incréméntation prédéfinie, il est donc possible de coder une boucle sans fin, qui peut planter l'ordinateur.

Exemple :

```
var variable=0 ;  
while(variable<3){  
    document.write(variable) ;  
    variable++;  
}
```

Actions de l'interpréteur	Calcul	Valeur variable
Il teste si la condition est remplie	0<3 => vrai	0
Il réalise les instructions (affichage de 0 et incréméntation)	0++ => 0+1 => 1	1
Il teste si la condition est remplie	1<3 => vrai	1
Il réalise les instructions (affichage de 0 et incréméntation)	1++ => 1+1 => 2	2
Il teste si la condition est remplie	2<3 => vrai	2
Il réalise les instructions (affichage de 0 et incréméntation)	2++ => 2+1 => 3	3
Il teste si la condition est remplie	3<3 => faux	3
Il sort de la boucle et continue le reste du script		

Structure itérative DO... WHILE

Cette structure permet de réaliser des instructions puis, si la condition perdure réaliser une boucle. Sa syntaxe est :

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

Attention : il n'y a pas d'incréméntation prédéfinie, il est donc possible de coder une boucle sans fin, qui peut planter l'ordinateur.

Alors que la boucle **while** teste la condition avant de faire les instructions, la boucle **do... while**, réalise les instructions avant de tester la condition. Cette structure réalisera au moins une fois les instructions.

Exemple :

```
variable=5 ;  
do{  
    document.write(« boucle do while ») ;  
    variable++;  
}while(variable<5) ;
```

Instruction BREAK

Nous l'avons vu dans le test **switch case**, l'instruction **break** permet de sortir d'une instruction. Dans le cas des boucles, il peut être nécessaire de sortir de la boucle, alors même que la condition de celle-ci est réalisée. Pour cela, on utilise l'instruction break.

Exemple :

```
for(variable=0 ; variable<10 ; variable++){  
    if(variable==5){  
        break ;  
    }  
    document.write(variable) ;  
}
```

Instruction CONTINUE

Il est parfois utile de sauter un incrément, afin de réaliser une série discontinue. L'instruction continue donne l'ordre d'ignorer la suite des instructions de la boucle et de continuer au niveau suivant de la boucle.

Exemple :

```
for(variable=0 ; variable<10 ; variable++){  
    if(variable==5){  
        continue ;  
    }  
    document.write(variable) ;  
}
```