Table des matières

Chapitre 1

Introduction à JavaScript

JavaScript est un langage de programmation qui permet d'apporter des améliorations au langage HTML, notamment rendre les sites Web dynamiques et interactifs.

JavaScript a été initialement élaboré en 1985 sous le nom de LiveScript par Brendan Eich de chez Netscape en association avec Sun Microsystem. JavaScript est reconnu par tous les navigateurs depuis 1996. Tout comme, le langage HTML ou les feuilles de style CSS, JavaScript est standardisé en 1997 par le comité spécialisé, ECMA (European Computer Manufactures Association) et en 1998 par ISO-16262.

1.1 Qu'est ce que Javascript ?

JavaScript est un langage de script interprété, non typé et orienté objet. Son code est intégré dans le langage HTML. Il est différent du langage Java et sont à des finalités différentes. Sa syntaxe est proche de celle du langage C.

JavaScript permet d'exécuter des programmes du côté client (navigateur) et ainsi de soulager les serveurs qui hébergent les pages web visitées. En plus, l'exécution de Javascript est rapide.

Par exemple, vérifier qu'un champ n'a pa été saisi et en avertir l'utilisateur, au lieu de faire cette vérification par le serveur. Faire des calculs, afficher des messages au visiteur d'une page, modifier le contenu de page web, défiler une série d'images sur l'écran. Déclencher une réaction à une action de l'utilisateur, comme un clique, un survol de souris etc. JavaScript permet aussi de développer des applications Internet.

Cependant, JavaScript ne permet aucune confidentialité au niveau des codes, car celui-ci est accessible à tout visiteur de page Web. Il suffit alors d'afficher le code source pour voir le code en JavaScript . En Javascript et pour la sécurité de votre ordinateur, il est impossible de lire ou d'écrire dans un fichier.

1.2 Où placer le code source Javascript ?

Le code JavaScript est inséré directement dans le document HTML ou mis dans un fichier externe et appelé dans le code HTML. Il y a trois trois façons d'insérer le code JavaScript .

1) Grâce à la balise <script> ... </script> de HTML. Cette balise signale au navigateur qu'il s'agit d'un script JavaScript à interpréter. Une page HTML peut contenir plusieurs balises <script>...</script>, mais elles ne doivent pas être imbriquées. La sytaxe est comme, il suit,

```
. . . code HTML. . .
<script type="text/javascript">
. . . code de javascript . . .
</script>
. . . code HTML. . .
```

Ou

```
<script language="javascript">
. . . code de javascript . . .
</script>
```

Pour les anciens navigateurs qui n'interprètent pas JavaScript on met le code en commentaire :

```
<script type="text/javascript">
<!--
. . . code de javascript . . .
-->
</script>
```

Rappelons que <!- - ... - -> est pour mettre des commentaires en HTML.

2) Grâce aux événements,

```
<balise onEvent ="code JavaScript à insérer">...</balise>
```

Comme exemple d'événement onEvent, on cite onClik (clique souris), onMouseover (survole de souris). Lorsque l'action onEvent est faite par l'utilisateur, le code JavaScript correspondant est déclenché. Nous allons voir les événements plus tard.

3) Insérer du code JavaScript à partir d'un fichier externe (avec l'extension .js) :

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

Le fichier contenant le code JavaScript , peut être sur une machine locale ou à distance, hébergé dans un site web. Un fichier externe ne doit pas contenir la balise <script> . . . </script>.

1.3 JavaScript et les autres langages

JQuery, (http ://jquery.com/) est une bibliothèque JavaScript libre et multi-plateforme créée pour faciliter la programmation en javascript. JQuery est facile à maîtriser.

Ajax (Asynchronous JavaScript and XML) n'est pas un nouveau langage de programmation mais un ensemble de technologies qui permettent la mise à jour du contenu d'une page Web d'une manière rapide et sans chargement complet de celle ci. Ajax combine HTML, JavaScript, CSS, XML, DOM et XMLHttpRequest.

1.4 Outils pour le Javascript

Pour pratiquer du JavaScript il suffit d'avoir un éditeur de texte et un navigateur. Tous les navigateurs (sauf les très anciens) sont équipés d'interpréteurs de code JavaScript .

Chapitre 2

Premiers scripts en JavaScript

1. Contrairement aux langages HTML et CSS, JavaScript est sensible à la casse, c'est à dire il y a une différence entre majuscule et minuscule.

2. Chaque instruction se termine par un point-virgule (comme en langage C).

3. Pour mettre en commentaire le reste d'une ligne on la précède par double slash // et pour mettre en commentaire plusieurs lignes on les met entre /* et on les termine par */ (comme en C, C++). Les commentaires ne peuvent être imbriqués.

2.1 La méthode document.write()

Nous allons expliquer cette nouvelle notation dans le chapitre intitulé programmation orienté objet.

La méthode document.write() permet l'affichage de ses arguments dans le corps du document HTML. Elle prend un ou plusieurs arguments. Elle accepte les balises HTML. La syntaxe est assez simple soit

```
<script type="text/javascript">
document.write("texte1", "texte2", ... ");
</script>
```

ou

```
<script type="text/javascript">
document.write("texte1"+"texte2"+... ");
</script>
```

Le symbole d'addition sert à la concaténation.

On peut aussi écrire une variable, soit la variable

```
<script type="text/javascript">
resultat=123;
document.write(resultat);
</script>
```

Pour associer du texte (chaînes de caractères) et des variables, on utilise l'instruction

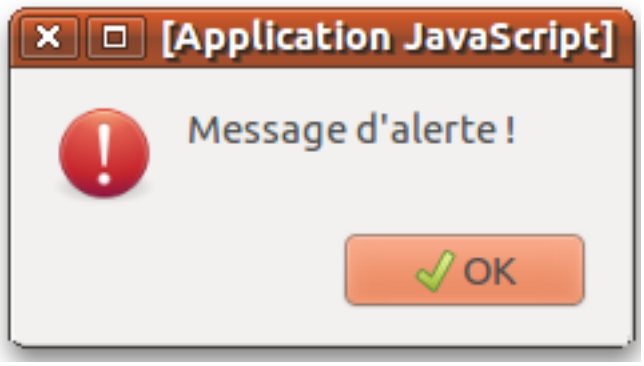

FIGURE 2.1 – Fenêtre alert()

```
<script type="text/javascript">
      ...
document.write("Le résultat est : " + resultat);
</script>
```

On peut utiliser les balises HTML dans document.write() pour produire la mise en forme de son contenu.

```
<script type="text/javascript">
document.write("<br><b>Le résultat est : </b>");
</script>
```

La méthode document.writeln() est proche de document.write() à ceci près qu'elle ajoute un retour chariot à la fin des caractères affichés par l'instruction. Ce qui n'a aucun effet en HTML. On utilise alors
 dans document.write() pour retour à la ligne.

2.2 Les fenêtres de dialogue

2.2.1 La méthode alert()

La méthode alert() ou window.alert() permet d'afficher un message d'alert dans une fenêtre. La syntaxe est

```
<script type="text/javascript">
alert("Message d'alerte");
</script>
```

```
<script type="text/javascript">
alert( "Bienvenue\nJavaScript est facile ! ");
</script>
```

\ permet de créer une nouvelle ligne.

La méthode alert() permet aussi d'afficher le résultat d'un script.

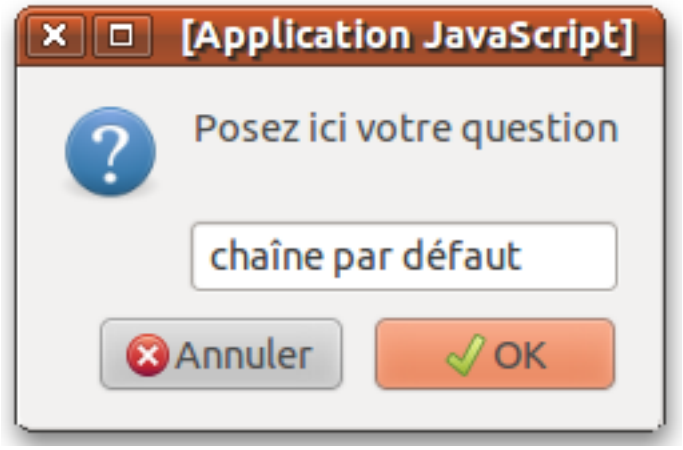

FIGURE 2.2 – Fenêtre prompt()

```
<script type="text/javascript">
res=123;
alert( "Le résultat est :\n"+res );
</script>
```

2.2.2 La méthode prompt()

La méthode prompt() permet de récupérer une information provenant de l'utilisateur et lui donner un nom pour l'utiliser comme variable par la suite. Elle requiert deux arguments : le texte d'invite et la chaîne de caractère par défaut dans le champ de saisie. La syntaxe est :

```
<script type="text/javascript">
var v=prompt(message[,message par defaut optionnel])
</script>
```

Le deuxième argument est optionnel. Exemple :

```
<script type="text/javascript">
var reponse = prompt('Posez ici votre question','chaîne par défaut');
</script>
```

La valeur récupérée par la méthode prompt() est toujours de type chaîne de caractère, comme le montre le script suivant

```
<script type="text/javascript">
n=prompt( "saisir un entier ");
t=typeof(n); // affiche le type
alert("ce que vous avez saisi est de type : "+t )
</script>
```

Si on désire convertir cette valeur en nombre, on peut utiliser la fonction parseInt() (pour la convertir en un nombre entier) ou parseFloat() (pour la convertir en nombre décimal).

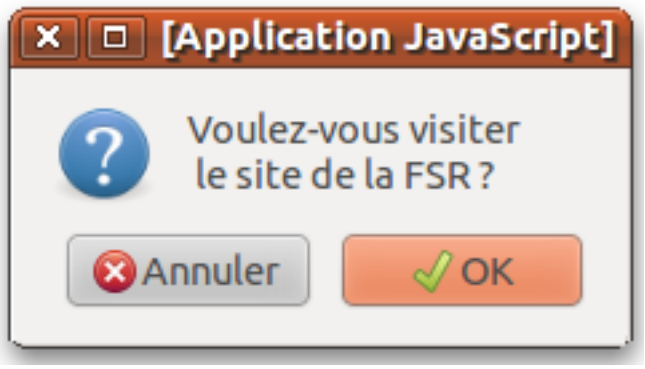

FIGURE 2.3 – Fenêtre confirm()

2.2.3 La méthode confirm()

La méthode confirm() prend un seul argument de type chaîne de caractère, et affiche une fenêtre de confirmation avec un message et deux boutons au choix : "OK" et "Annuler". Cette méthode retourne true si on clique "OK" et false si on clique "Annuler". Syntaxe

```
<script type="text/javascript">
confirm("Message");
</script>
```

confirm() est souvent utilisé dans une structure conditionnelle par exemple :

```
<script type="text/javascript">
if(confirm("Voulez-vous visiter\n le site de la FSR ?"))
{document.location.href='http://www.fsr.ac.ma'; }
</script>
```

permet de choisir une option : "OK" on est amené sur le lien de la FSR et "Annuler" rester sur la page courante.

`\n` permet d'insérer un retour à la ligne.

2.3 Quelques fonctions de base

parseFloat(chaîne) : permet de transformer une chaîne de caractères en un nombre flottant si possible, sinon renvoie NaN (Not a Number)

Syntaxe : parseFloat(string)

parseFloat("3.14");

parseFloat("AB2"); // retourne NaN

parseInt(string[, base]) : Analyse une chaîne de caractères et retourne un nombre entier dans la base spécifiée. La base peut être 2 (binaire), 8 (octal), 10 (décimal), 16 (hexadécimal).

```
<script type="text/javascript">
c=parseInt('100',16); document.write(c) //retourne 256
c=parseInt('A',16); document.write(c) //retourne 10
c=parseInt('A',8); document.write(c) //retourne NaN
```

```
c=parseInt('16',8); document.write(c) //retourne 14
c=parseInt(65,2); document.write(c) //retourne NaN
c=parseInt(1000001,2); document.write(c) //retourne 65
</script>
```

eval : évalue une chaîne de caractère représentant une expression JavaScript, une instruction ou une suite d'instructions JavaScript.

Par exemple : eval("2 + 3"); retourne 5.

Chapitre 3

Programmation en JavaScript

3.1 Variables

Les variables stockent des données qui peuvent être modifiées lors de l'exécution d'un programme. Le nom d'une variable doit commencer par une lettre non accentuée, un tiret bas _ ou un dollar $, et peut comporter uniquement des lettres non accentuées, chiffres, tiret bas _ ou dollar $ (mais jamais d'espace, caractère spécial, symbole de ponctuation ou autres).

Les caractères accentués comme à, é, è, ç etc ne peuvent être employés dans les noms de variables mais uniquement dans les chaînes de caractères.

Un nom de variable ne peut être un nom réservé de JavaScript : abstract, boolean, break, byte, case catch char class, default do double, else extends, false final finally float, goto, if, implements, import, in, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, var, void, while, with.

JS n'impose pas l'initialisation des variables au moment de leur création. JS est non typé. Il n'est pas nécessaire de déclarer le type d'une variable en JS. On peut déclarer une variable par

var x ;

Le symbole d'affectation est le signe =.

Il est possible de définir plusieurs variables en une seule instruction, en les séparant par virgule :

var x=3, y=6 ;

La déclaration de variable peut se faire de deux façons :

- Explicitement, par le mot clef var.

Par exemple :

var Numero = 1 ; var nom = "Fatima" ;

- Implicitement : sans utiliser var. Par exemple :

Numero = 1 ; Prenom = "Nadia" ;

Utiliser var ou non ?

Dans un script, les variables déclarées avec le mot clef var ou non, en dehors de toute fonctions, seront toujours globales. C'est à dire, on peut les exploiter partout dans le document.

Dans une fonction (définit par le mot clé function, voir chapitre sur les fonctions), une variable déclarée par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le document. Par contre, dans une fonction, si la variable est déclarée sans le mot var, alors sa portée sera globale.

3.2 Types de données

En JavaScript, nous avons les types suivants :

1. Number : nombres entiers ou décimaux. Par exemple 21 ou 3.14 (séparateur décimal étant le point).

2. String : chaîne de caractère : suite de caractères à placer toujours entre ' ' ou " ", (sinon cette chaîne de caractère sera considérée comme une variable).

3. Boolean : type dont les valeurs possibles sont true et false.

4. Null : indique l'absence de valeur : aucune valeur pour l'objet n'est présente

5. Undefined : l'unique valeur possible est undefined. C'est le type d'une variable avant qu'elle ne soit affectée.

L'opérateur typeof permet de déterminer le type d'une variable ou d'une expression.

Syntaxe : typeof(x) ou typeof x

```
var i = 1;
typeof i; //retourne number
var titre="JavaScript est génial";
typeof titre; //retourne string
var choix = true;
typeof choix; //retourne boole
var V;
alert(" type de V : "+(typeof V)); // retourne undefined
```

Conversion de types.

Les conversions possibles sont de types primitifs en chaînes de caractères et de chaînes de caractères en nombres entiers ou réels. Les types primitifs correspondent à des pseudo- objets. Ils peuvent posséder des méthodes,

```
var b = true;
alert(" type de b : "+(typeof b));
var v = b.toString();
alert(" type de v : "+(typeof v));
```

Pour les nombres, la méthode toString peut être utilisée avec un paramètre pour spécifier la base souhaité. Par défaut, la base décimale est utilisée. Il est possible de spécifier, la base binaire par 2 , hexadécimale par 16 ou n'importe qu'elle autre base.

```
var N = 5;
var v1 = N.toString();
var v2 = N.toString(2); // 101
```

Les fonctions parseInt et parseFloat convertissent les chaînes de caractères représentant des nombres en nombres entiers ou décimaux. ParseInt peux prendre 2 arguments : la chaine à convertir et sa base.

```
var v = parseInt("11001"); //retourne 11001
var v = parseInt("11001",2); //retourne 25
```

Constante	Explication
undefined	la variable qui prend cette valeur a été déclarée mais n'a pas été initialisée.
null	la variable n'existe pas.
Infinity	représente l'infini positif. Permet notamment de vérifier si il y a une division par zéro.
NaN	équivaut à la définition de l'IEEE de Not a Number .

TABLE 3.1 – Constantes prédéfinies de JS

Les opérateurs + et - permettent de convertir une chaîne de caractères en nombre de la même façon que la méthode parseInt . Le - change en plus le signe du nombre.

3.3 Constantes

Une constante est une variable qui ne change pas de valeur. Pour définir une constante, il suffit d'utiliser un nom de constante et de lui affecter une valeur au moyen de l'opérateur d'affectation =. Par exemple pi=3.14 ; ou const pi=3.14 ; Il existe des constantes prédéfinies telles que dans la Table 3.1 :

Toute valeur différente de null, NaN, undefined, 0 et la chaîne de caractères vide est évaluée par défaut par JavaScript comme true. Var x=3 ; if(x) ?

3.4 Opérateurs arithmétiques, booléens et de comparaison

Pour développer des programmes, en plus des variables à manipuler, nous disposons de nombreux opérateurs pour les traiter.

3.4.1 Les opérateurs de calcul

JS utilise les opérations arithmétiques classiques. Dans les exemples, la valeur initiale de x sera toujours égale à 11.

Signe	Nom	Signification	Exemple	Résultat
+	plus	addition	x + 3	14
-	moins	soustraction	x - 3	8
*	multiplié par	multiplication	x*2	22
/	divisé	par division	x /2	5.5
%	modulo	reste de la division par	x%7	4
=	a la valeur	affectation	x=5	5

3.4.2 Les opérateurs de comparaison

JS utilise les opérateurs suivants pour la comparaison de valeurs. Ces opérateurs sont utilisés dans les structures conditionnelles que nous allons voir plus tard.

Pour les exemple, on suppose x=11.

Signe	Nom	Exemple	Résultat
==	comparaison de valeurs	x==11	true
===	comparaison de valeur et type	x=='11'	false
<	inférieur	x<11	false
<=	inférieur ou égal	x<=11	true
>	supérieur	x>11	false
=<	supérieur ou égal	x>=11	true
!=	différent	x!=11	false
!==	différence de valeur et type	x!=='11'	true

```
var b=2>5;
alert(" type de b : "+(typeof b)); \\ retourne boolean
```

Important. Il ne faut pas confondre le = et le == (deux signes =). Le = est un opérateur d'affectation de valeur tandis que le == est un opérateur de comparaison. Cette confusion est une source classique d'erreur de programmation.

3.4.3 Les opérateurs associatifs

On appelle ainsi les opérateurs qui réalisent un calcul dans lequel une variable intervient des deux côtés du signe = (ce sont donc en quelque sorte également des opérateurs d'attribution). Dans les exemples suivants x vaut toujours 11 et y aura comme valeur 5.

Signe	Description	Exemple	Signification	Résultat
+=	plus égal	x += y	x = x + y	16
-=	moins égal	x -= y	x = x - y	6
*=	multiplié égal	x *= y	x = x * y	55
/=	divisé égal	x /= y	x = x / y	2.2

3.4.4 Les opérateurs d'incrémentation

Ces opérateurs vont augmenter ou diminuer la valeur de la variable d'une unité. Ce qui sera fort utile, par exemple, pour mettre en place des boucles. Dans les exemples x vaut 3.

Signe	Description	Exemple	Signification	Résultat
x++	incrémentation (x++ est le même que x=x+1)	y = x++	3 puis plus 1	4
x- -	décrémentation (x- - est le même que x=x-1)	y= x- -	3 puis moins 1	2

En incrémentant ou décrémentant avec le double signe (++ ou - -) avant le nom de la variable, l'opération d'incrémentation ou de décrémentation sera prioritaire sur l'assignation. Par contre, si le double signe est après le nom de la variable, l'assignation sera prioritaire sur l'incrémentation ou la décrémentation.

3.4.5 Les opérateurs logiques

Aussi appelés opérateurs booléens, ses opérateurs servent à composer deux ou plusieurs conditions.

Signe	Nom	Exemple	Signification
&&	et	(condition1) && (condition2)	est vrai si les 2 conditions sont vraies, sinon fausse
||	ou	(condition1) || (condition2)	est vrai si au moins une des 2 conditions est vraie, sinon fausse
!()	non logique	!(condition)	donne vrai si 'condition' est fausse, sinon fausse

3.4.6 Opérateur sur les chaînes de caractères

L'opérateur + permet aussi de concaténer deux chaînes de caractères. Par exemple "bon"+"jour" donne "bonjour".

3.5 Structures conditionnelles

Seul le premier if et le bloc qui le qui sont nécessaires.

3.5.1 La structure if

```
if (condition1)
      {
       code qui sera interpété si la condition1 est vraie
       }
else if (condition2)  //optionnel
      {
      code qui sera interpété si la condition2 est vraie
       }
else    //optionnel
       {
      code qui sera interpété si toutes
       les conditions ci-dessus sont fausses
      };
```

La condition doit être toujours entourée de parenthèses ( ).
Les accolades { } ne sont obligatoires qu'en cas d'instructions multiples.
Il est possible d'utiliser autant de blocs else if que nécessaire.
La séquence else (optionnelle) est interprétée si toutes les 'condition' sont fausses.
Exemple.

```
      <script type="text/JavaScript">
      var n=prompt('Saisir votre âge');
      if (n>=0 && n<=14){alert('Vous êtes un enfant')}
      else if (n>=15 && n<=24){alert('Vous êtes un adolescent')}
      else if (n>=25 && n<=64){alert('Vous êtes un adulte')}
```

```
      else if (n>=65 && n<=120){alert('Vous êtes un aînés')}
      else {alert('âge érroné')}
      </script>
```

3.5.2 Une autre structure conditionnelle

```
var variable=(condition) ? Valeur1 : Valeur2 ;
```

Si 'condition' est vraie, alors valeur1 est affectée à 'variable', sinon c'est valeur2 qui lui est affectée.

Par exemple :

x=(x>=0)?x :(-x);

tranche=(age>18)? 'majeur' :'mineur';

salut=(heure<=18)?"Bonjour" :"Bonsoir";

3.5.3 switch

La syntaxe est

```
switch(variable)
{
case valeur1 : instructions à faire si variable=valeur1 ; break ;
case valeur2 : instructions à faire si variable=valeur2 ; break ;
...
default : instructions à faire, par défaut, si
             variable est différente de valeur1, valeur2 etc ;
}
```

switch ne peut tester que l'égalité de valeurs et non pas l'inégalité.

Le contenu des case (valeur1, etc) doit être un élément déjà évalué. On ne peut y insérer des expressions. Pour éviter de tester tous les cas, l'utilisation de break permet de sortir du switch dès l'action terminée. Le default permet d'effectuer une action si la variable testée ne coïncide avec aucune des valeurs proposées dans les case précédent.

Il faut préciser un break pour chaque case afin de sortir du switch, sinon le switch va lire et exécuter le code contenu dans chaque case à la suite.

Exemple :

```
<script type="text/JavaScript">
var n=parseFloat(prompt('Saisir un nombre'));
switch (n){
      case 1: alert('Janvier'); break;
      case 2: alert('Février'); break;
      case 3: alert('Mars'); break;
      ...
      default :alert('Saisie érronée')
}
```

```
</script>
```

3.6 Structures itératives

Les structures itératives ou boucles permettent d'exécuter un bloc de code un certain nombre de fois, tant qu'une certaine condition est vraie. La condition donnée doit être fausse à un moment donné, sinon la boucle devient infinie ! JavaScript définit quatre types de boucles, for , for...in , while et do...while .

3.6.1 La boucle for

```
for (départ_compteur; condition_continuation; incrémentation)
{
code à interpréter ;
}
```

Exemple :

```
<script type="text/javascript">
for (var k=1; k<=10; k++)
  {
  document.write(7*k+"<br>");
  };
</script>
```

3.6.2 La boucle for in

```
for (variable in object)
  {
  code à interpréter ;
  }
```

```
<script type="text/javascript">
var personne={Nom:"Ali", Prénom:"Mohammed", age:25};
for (x in personne){
document.write(personne[x] + " ");
}
</script>
```

3.6.3 while

```
while (condition)
{
code à exécuter à chaque passage ;
}
```

```
<script type="text/javascript">
k=1;
while (k<=10)
  {document.write(7*k+' ');
    k++;
  };
</script>
```

3.6.4 do while

```
do
{
  code qui sera interprété à chaque itération
}
while (condition);
```

La boucle do – while est intéressante si besoin pour une raison ou pour une autre d'effectuer au moins un passage dans une boucle pour faire fonctionner un script. L'initialisation est exécutée avant tout passage dans la boucle for, tandis que l'incrémentation est exécutée à la fin de chaque passage dans la boucle for.

JavaScript définit les mots-clé break et continue afin de modifier l'exécution des boucles. Le premier offre la possibilité d'arrêter l'itération d'une boucle et de sortir de son bloc d'exécution, et le second de forcer le passage à l'itération suivante. Les traitements suivants de l'itération courante ne sont alors pas effectués.

break : permet d'interrompre prématurément une boucle for ou while.

Continue : permet de sauter une instruction dans une boucle for ou while et de continuer ensuite le bouclage (sans sortir de celui-ci comme le fait break).

Exemple

```
<script type="text/JavaScript">
var res = '';
for (i = 0; i < 10; i++) {
      if (i >= 3 && i<=5) { continue; }
      text = res + i+' ';}
document.write(res)
</script>
```

```
<script type="text/JavaScript">
var i = 0;
while (i < 10) { if (i == 5) {break;}
      i = i + 1;
      document.write(i)}
</script>
```