

JavaScript

Table des matières

Introduction.....	5
Avantages.....	6
Limitations.....	7
Ajouts (add-ons) des navigateurs.....	7
Compresser son code javascript.....	8
Les bases de JavaScript.....	8
Principe de base.....	8
La balise <script>.....	8
Attribut type.....	8
Attribut src.....	8
La place de JavaScript dans la page HTML.....	9
Dans une balise <script>.....	9
Dans une balise existante.....	9
Dans un fichier externe.....	9
La séparation des instructions.....	9
Casse de JavaScript.....	9
La balise <noscript>.....	10
Les commentaires JavaScript.....	10
Deux types de données simples.....	10
Les nombres.....	10
Les chaînes.....	11
Le bon usage des guillemets.....	11
Les opérations sur les variables.....	11
Les opérations sur les nombres.....	11
Les opérateurs mathématiques.....	11
Le modulo.....	12
La concaténation des chaînes.....	12
Principe.....	13
Création d'une fonction.....	13
Gestion des valeurs.....	14
Les arguments.....	14
La valeur de retour.....	14
Quelques fonctions de JavaScript.....	15
Interaction.....	15
Manipulation de variables.....	16
Principe général.....	16
Les opérateurs de comparaison et les opérateurs logiques.....	16
Opérateurs de comparaison.....	16
Les six opérateurs de base.....	16
Deux autres opérateurs : comparaison de la valeur et du type.....	17
Opérateurs logiques.....	17
La négation.....	17
Et et ou.....	17
Un exemple de fonction renvoyant un boolean.....	18
Principe.....	18

If / else – Si / sinon.....	18
Principe.....	19
While – Tant que.....	19
For – Pour.....	20
Langage à objets.....	20
Principe.....	20
Utiliser un objet.....	21
Le mot clef this.....	22
Création d'un d'objet par instance de classe.....	23
Le Modèle Objet du Document (Document Object Model en anglais).....	23
Principe.....	23
Illustration.....	23
Adressage d'un objet du DOM.....	24
Propriétés de l'objet.....	25
Méthodes appliquées à l'objet et Arguments de la méthode.....	25
Référentiel.....	26
Déclaration.....	26
Type.....	26
Nombre.....	26
Chaînes de caractères.....	27
Booléens.....	27
Tableaux.....	27
Déclaration.....	28
Index.....	28
Tableau à deux dimensions.....	28
Syntaxe.....	29
Invocation.....	29
Passage des Arguments.....	30
Imbrication.....	30
Retour de résultats.....	31
Variables locales et globales.....	31
Opérateur d'affectation.....	33
Opérateurs arithmétiques.....	33
Opérateur de Concaténation.....	34
Opérateurs d'Affectation.....	34
Opérateurs de Comparaison.....	34
Opérateurs Logiques.....	35
Priorité des opérateurs.....	35
Événements possibles.....	36
Syntaxe.....	37
Principaux Gestionnaires d'Événements.....	37
Structure Conditionnelle IF ... ELSE.....	37
If ... else.....	38
Else ... if.....	38
Opérateur ternaire.....	39
Structure conditionnelle SWITCH ... CASE.....	39
Structure itérative FOR.....	40
Structure itérative FOR ... IN.....	41
Structure itérative WHILE.....	42
Structure itérative DO ... WHILE.....	42

Instruction BREAK.....	43
Instruction CONTINUE.....	43
L'objet Window.....	44
L'objet Location.....	44
L'objet History.....	45
L'objet Document.....	45
L'objet Images.....	45
L'objet Form.....	46
Les éléments de formulaire.....	46
Pour tous les objets Éléments de Formulaire.....	46
Objets Text, Password, Hidden, FileUpload, Textarea.....	46
Objets Button, Submit, Reset.....	46
Objets Radio, Checkbox.....	47
Objet Select.....	47
Objet Options.....	47
L'objet Navigator.....	47
L'objet String.....	47
L'objet Array.....	48
L'objet Math.....	48
L'objet Date.....	49
Mots-clé réservé.....	49

Ce document est divisée 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 librairies JavaScript.

La contrepartie de l'utilisation de ces librairies 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 developper** 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 developper 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 prenoms = ['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(prenoms[1]);
```

Le résultat affiché est Paul.

Un tableau peut contenir lui-même un tableau :

```
var prenoms = ['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(prenoms[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 `function` 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 différent 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
<code>==</code>	égal à
<code>!=</code>	différent de
<code>></code>	supérieur à
<code>>=</code>	supérieur ou égal à
<code><</code>	inférieur à
<code><=</code>	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 booleen

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, ...

Modèle-objet de JavaScript

Langage à objets

|| PRINCIPE

Le JavaScript est un langage orienté objet. Toute variable dans JavaScript est, en fait, un objet. Un