Javascript - Syntaxe du langage

Source : wikipedia.org

Table des matières

Variable

Les variables en JavaScript n'ont pas de type défini, et n'importe quelle valeur peut être stockée dans n'importe quelle variable. Les variables peuvent être déclarées avec `var` . Ces variables ont une portée lexicale et une fois la variable déclarée, on peut y accéder depuis la fonction où elle a été déclarée. Les variables déclarées en dehors d'une fonction et les variables utilisées sans avoir été déclarées en utilisant `var`, sont globales (peuvent être utilisées par tout le programme).

Voici un exemple de déclaration de variables et de valeurs globales :

```
x = 0; // Une variable globale
var y = 'Hello!'; // Une autre variable globale

function f(){
  var z = 'foxes'; // Une variable locale
  twenty = 20; // Globale car le mot-clef var n'est pas utilisé
  return x; // Nous pouvons utiliser x ici car il s'agit d'une
variable globale
}
// La valeur de z n'est plus accessible à partir d'ici
```

Type de données

Nombres

Les nombres en JavaScript sont représentés en binaire comme des IEEE-754 Doubles, ce qui permet une précision de 14 à 15 chiffres significatifs JavaScript FAQ 4.2 **(en)**.
Comme ce sont des nombres binaires, ils ne représentent pas toujours exactement les nombres décimaux, en particulier les fractions.
Ceci pose problème quand on formate des nombres pour les afficher car JavaScript n'a pas de méthode native pour le faire. Par exemple:
alert(0.94 - 0.01); // affiche 0.9299999999999999
En conséquence, l'arrondi devrait être utilisé dès qu'un nombre est formaté pour l'affichage **(en)**. La méthode toFixed() ne fait pas partie des spécifications de l'ECMAScript et est implémentée différemment selon l'environnement, elle ne peut donc être invoquée.
Les nombres peuvent être spécifiés dans l'une de ces notations :

```
345;    // un "entier", bien qu'il n'y ait qu'un seul type
numérique en JavaScript
34.5;   // un nombre flottant
3.45e2; // un autre nombre flottant, équivalent à 345
0377;   // un entier octal égal à 255
```

```
0xFF;   // un entier hexadecimal égal à 255, les lettres A-F
peuvent être en minuscules ou en majuscules
```

Le constructeur Number peut être utilisé pour réaliser une conversion numérique explicite :

```
var myString = "123.456"
var myNumber = Number( myString );
```

Chaînes de caractères

En Javascript la chaîne de caractères est considérée comme une suite de caractères. Une chaîne de caractères en JavaScript peut être directement créée en plaçant des caractères entre quotes (doubles ou simples) :

```
var salutation = "Hello, world!";
var salutmartien = 'Bonjour, amis terriens';
```

Opérateurs

L'opérateur '+' est surchargé; il est utilisé pour la concaténation de chaîne de caractères et l'addition ainsi que la conversion de chaînes de caractères en nombres.

```
// Concatène 2 chaînes
var a = 'Ceci';
var b = ' et cela';
alert(a + b);  // affiche 'Ceci et cela'

// Additionne deux nombres
var x = 2;
var y = 6;
alert(x + y); // affiche 8

// Concatène une chaîne et un nombre
alert( x + '2'); // affiche 22

// Convertit une chaîne en nombre
var z = '4';   // z est une chaîne (le caractère 4)
alert( z + x); // affiche 42
alert( +z + x);// affiche 6
```

Affectations

= Affectation

+= Ajoute et affecte
-= Soustrait et affecte
*= Multiplie et affecte
/= Divise et affecte

```
var x = 1;
x *= 3;
document.write( x );  // affiche: 3
x /= 3;
document.write( x );  // affiche: 1
x -= 1;
document.write( x );  // affiche: 0
```

Comparaisons
== Égal à
!= Différent de
> Supérieur à
>= Supérieur ou égal à
< Inférieur à
<= Inférieur ou égal à

Booléens
Le langage Javascript possède 3 opérateurs booléens :
&& and (opérateur logique ET)
|| or (opérateur logique OU)
! not (opérateur logique NON)

Dans une opération booléenne, toutes les valeurs sont évaluées à `true` (VRAI), à l'exception de :
- la valeur booléenne `false` (FAUX) elle-même
- le nombre 0
- une chaîne de caractères de longueur 0
- une des valeurs suivantes : *null undefined NaN*

On peut faire explicitement la conversion d'une valeur quelconque en valeur booléenne par la fonction `Boolean` :

```
Boolean( false );      // rend false (FAUX)
Boolean( 0 );          // rend false (FAUX)
Boolean( 0.0 );        // rend false (FAUX)
Boolean( "" );         // rend false (FAUX)
Boolean( null );       // rend false (FAUX)
Boolean( undefined ); // rend false (FAUX)
Boolean( NaN );        // rend false (FAUX)
```

```
Boolean ( "false" );  // rend true (VRAI)
Boolean ( "0" );      // rend true (VRAI)
```

L'opérateur booléen unaire NOT *!* évalue d'abord la valeur booléenne de son opérande, puis rend la valeur booléenne opposée :

```
var a = 0;
var b = 9;
!a; // rend true, car Boolean( a ) rend false
!b; // rend false, car Boolean( b ) rend true
```

Structures de contrôle

Si Sinon

```
 if (expression1) {
   //instructions réalisées si expression1 est vraie;
 } else if (expression2) {
   //instructions réalisées si expression1 est fausse et
expression2 est vraie;
 } else {
   //instructions réalisées dans les autres cas;
 }
```

Dans chaque structure *if..else*, la branche *else* :

- est facultative
- peut contenir une autre structure *if..else*

Opérateur conditionnel

L'opérateur conditionnel crée une expression qui, en fonction d'une condition donnée, prend la valeur de l'une ou l'autre de 2 expressions données. Son fonctionnement est similaire à celui de la structure *si..sinon* qui, en fonction d'une condition donnée, exécute l'une ou l'autre de 2 instructions données. En ce sens on peut dire que l'opérateur conditionnel est aux expressions ce que la structure *si..sinon* est aux instructions.

```
 var resultat = (condition) ? expression1 : expression2;
```

Le code ci-dessus, qui utilise l'opérateur conditionnel, a le même effet que le code ci-dessous, qui utilise une structure *si..sinon* :

```
 if (condition) {
   resultat = expression1;
 } else {
   resultat = expression2;
 }
```

Contrairement à la structure *si..sinon*, la partie correspondant à la branche *sinon* est obligatoire.

```
var resultat = (condition) ? expression1; // incorrect car il manque la valeur à prendre si condition est fausse
```

Switch

L'instruction switch remplace une série de *si..sinon* pour tester les nombreuses valeurs que peut prendre une expression :

```
switch (expression) {
  case UneValeur:
    // instructions réalisées si expression=UneValeur;
    break;
  case UneAutreValeur:
    // instructions réalisées si expression=UneAutreValeur;
    break;
  default:
    // instructions réalisées dans les autres cas;
    break;
}
```

Boucle For

Syntaxe pour plusieurs lignes d'instructions :

```
for (initialisation;condition;instruction de boucle) {
  /*
     instructions exécutées à chaque passage dans la boucle
     tant que la condition est vérifiée
  */
}
```

Boucle While

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

Fonctions

Une fonction est un bloc d'instructions avec une liste de paramètres (éventuellement vide). Elle possède généralement un nom et peut renvoyer une valeur.

```
function nom_fonction(argument1, argument2, argument3) {
  instructions;
  return expression;
}
```

Exemple de fonction : l'algorithme d'Euclide. Il permet de trouver le plus grand commun diviseur de deux nombres, à l'aide d'une solution géométrique qui soustrait le segment le plus court du plus long:

```
function gcd(segmentA, segmentB) {
  while (segmentA != segmentB) {
    if (segmentA > segmentB)
      segmentA -= segmentB;
    else
      segmentB -= segmentA;
  }
  return segmentA;
}
println(gcd(60, 40)); // imprime 20
```

Tableaux

Un tableau est un ensemble d'éléments repérés par leur indice, qui est un nombre entier. En JavaScript, tous les objets peuvent être formés d'un ensemble d'éléments, mais les tableaux sont des objets spéciaux qui disposent de méthodes spécifiques (par exemple, `join`, `slice`, et `push`).

Les tableaux ont une propriété `length` qui représente la longueur du tableau, c'est-à-dire le nombre d'éléments qu'il peut contenir. La propriété length d'un tableau est toujours supérieure à l'indice maximal utilisé dans ce tableau (N.B. Les indices de tableaux sont numérotés à partir de zéro). Si on crée un élément de tableau avec un indice supérieur à `length`, `length` est automatiquement augmentée. Inversement, si on diminue la valeur de `length`, cela supprime automatiquement les éléments qui ont un indice supérieur ou égal. La propriété `length` est la seule caractéristique qui distingue les tableaux des autres objets.

Les éléments de tableau peuvent être accédés avec la notation normale d'accès aux propriétés d'objet :

```
monTableau[1];
```

La déclaration d'un tableau peut se faire littéralement ou par l'utilisation du constructeur `Array` :

```
monTableau = [0,1,,,4,5];              // crée un tableau de
longueur 6 avec 4 éléments
monTableau = new Array(0,1,2,3,4,5);   // crée un tableau de
longueur 6 avec 6 éléments
monTableau = new Array(365);           // crée un tableau vide de
longueur 365
```

Tableaux associatifs

Grâce à la déclaration littérale, on peut créer des objets similaires aux tableaux associatifs d'autres languages :

```
chien["nom"] = "caramel";
chien["couleur"] = "brun";
```