Chapitre 4

Les fonctions en JavaScript

La notion de fonction existe dans tous les langages de programmation. Une fonction est un sous-programme qui porte un nom et qui peut être utilisé plusieurs fois, par simple appel de son nom. Le nom de fonction suit les mêmes règles que celles de variables.

Avant de l'appeler, une fonction doit être définie. Une fonction peut être placée n'importe où dans un document HTML ; mais il est conseillé de la placer entre <head> ... </head>. Les arguments de fonctions doivent être indépendants du contexte d'un document HTML. Pour rappel, Javascript est sensible à la casse. Ainsi, fonct() et Fonc() sont différentes. En outre, les noms de fonctions dans un même script doivent être différents par exemple.

Les instructions composant la définition d'une fonction ne sont interprétées que jusqu'à l'appel de la fonction.

En Javascript, il existe deux types de fonctions :

- les fonctions prédéfinies dans le langage Javascript, comme parseInt(), eval() etc. Les fonctions membres appelées aussi "méthodes". Chacune d'entre elles est associées à un objet de JavaScript . Par exemple la méthode alert() de l'objet window, document.write de document etc.

- les fonctions écrites par le programmeur.

4.1 Programmation de fonction

Pour définir une fonction, on utilise le mot réservé function. La syntaxe de définition d'une fonction est la suivante.

```
function maFonction(argument1, argument2, ...) {
      ... instructions ... ;
      return resultat
}
```

La mention d'arguments est facultative dans la définition d'une fonction. En cas d'utilisation d'arguments, ils sont séparés par la virgule. Les parenthèses qui suivent le nom de fonction sont obligatoires, même en cas d'absence d'arguments. Ces parenthèses permettent à l'interpréteur JavaScript de distinguer les fonctions des variables. La définition d'une fonction est groupée dans un bloc entre accolades { }.

Pour renvoyer un résultat, il suffit d'écrire le mot clé return suivi de l'expression à renvoyer, sans parenthèses. L'instruction return est facultative et on peut utiliser plusieurs return dans une même fonction.

Une fois une fonction définie, on pourra l'appeler partout dans la suite par son nom. Une fonction peut appeler une autre fonction déjà définie.

Rappelons que le code JavaScript peut être placé comme valeur d'attribut d'événement onClick, onMouseOver, onFocus etc.

<balise onEvent=" code JavaScript "> ... </balise>

Quand ce code JavaScript est long, on l'écrit sous forme d'une fonction nommée maFonction par exemple et on l'appelle comme il suit :

<balise onEvent="maFonction()"> ... </balise>

```
<script type="text/javascript">
function fac(n){
      if (n < 2) {
            return 1;
      } else {
            return n * fac(n-1);
      }
}
</script>
```

On pourra par la suite appeler cette fonction

```
<script type="text/javascript">
var N= fac(7);
document.write(N);
</script>
```

Les arguments passés à une fonction constituent automatiquement un tableau nommé arguments.

Par exemple : une fonction qui fait la somme d'un nombre indéterminé de nombres.

```
<script>
function somme() {
   var s = 0;
   for (i = 0; i < arguments.length; i++) {
      s += arguments[i];
   }
   return s;
}
x = somme(2,4,6,8,10);
y = somme(1,3,5);
document.write(x);
document.write(y);
</script>
```

Écrire Une fonction qui affiche ses arguments.

```
<script type="text/javascript">
function F()
{for(i=0;i<arguments.length;i++)
      document.write(arguments[i]);
}
F("a","b", 123); // retourne ab123
</script>
```

Variable Locale. Une variable déclarée dans une fonction en utilisant var, n'est reconnue que à l'intérieur cette fonction. Cette variable n'est pas reconnue en dehors de sa fonction. On dit que cette variable est locale. Le navigateur efface de la mémoire toute variable locale à une fonction dès que cette fonction est utilisée.

Deux fonctions différentes peuvent utiliser un même nom de variable.

Variable Globale. Une variable globale peut être déclarée n'importe où dans un script en-dehors de toute fonction. On dit qu'elle a une portée globale. Une variable globale est stockée en mémoire, ce qui veut dire qu'elle prend plus de mémoire qu'une variable locale. C'est pourquoi, il faut privilégier l'utilisation des variables locales autant que possible.

4.2 Fonctions de haut niveau prédéfinies

Les fonctions de haut niveau ne sont associées à aucun objet particulier.

Fonction	Signification
encodeURI(string)	Encode l'URI de telle façon que tous les caractères spéciaux sont transformés en séquences de signes ASCII.
decodeURI(string)	Décode une URI qui a été encodé avec encodeURI.
eval()	Evalue et exécute le JS contenu dans une chaine de caractère. Var x=7; chaine="x= x+ 3"; eval(chaine);
IsFinite()	Retourne true si la valeur passée en paramètre fait partie de la plage de nombre que peut traiter JavaScript, sinon retourne false.
isNaN()	Retourne true si la valeur passés en paramètre est un nombre, sinon retourne false.
Number()	convertit un objet en un nombre.
ParseFloat()	convertit une chaîne de caractères en valeur réelle flottante
parseInt()	convertit une chaîne de caractères en valeur entière dans la base spécifiée
String()	convertit un objet en une chaîne.

TABLE 4.1 – Fonctions de haut niveau prédéfinies en JavaScript

La fonction eval() permet d'évaluer une chaîne de caractère sous forme de valeur numérique. On peut stocker dans une chaîne des opérations numériques, des opérations de comparaison, des instructions et même des fonctions.

Par exemple C='3 + 7' ; eval(C) donne comme résultat 10.

Voir les autres fonctions dans le Tableau 4.1. Certaines fonctions comme escape(), unescape() sont devenues obsolètes.