Langage JavaScript

Le langage JavaScript permet d'exécuter des programmes simples comme la vérification d'un formulaire, ou très élaborés comme un jeu ou une application. Ce langage est interprété en natif par les navigateurs, sans plugin ajouté, ce qui est un grand avantage par rapport à d'autres langages qui nécessitent l'installation d'un plugin comme Java.

Les principales utilisations du langage JavaScript sont les suivantes :

- Animation des pages HTML par la gestion des événements (événement = action de l'utilisateur, clic souris, clavier, etc.)
- Contrôle de saisie dans les formulaires avant leur envoi au serveur pour traitement
- Informations sur l'internaute : détection du navigateur utilisé, langue, système d'exploitation
- Menu déroulant, galerie d'images animées, slider, effet lightbox
- Jeux en ligne
- Applications en ligne

Attention : Ne pas confondre le langage JavaScript avec le langage Java qui est un autre langage développé par *Oracle*.

Le traditionnel *Hello World !*

Le code peut être placé dans la zone d'entête <head> ou dans la partie <body> de la page HTML. Le navigateur exécute le code dès qu'il le rencontre.

Ici dans <*body*> :

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
        </head>
        <body>
                <p id="texte"></p>
                <script>
                document.getElementById("texte").innerHTML = "Hello World!";
                </script>
        </body>
</html>
```

Là dans <head>, mais le code ci-dessous ne fonctionnera pas car on mentionne un *id='texte'* qui n'a pas encore été lu par le navigateur.

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <script>
                document.getElementById("texte").innerHTML = "Hello World!";
                </script>
        </head>
        <body>
                <p id="texte"></p>
        </body>
</html>
```

Une manière de rectifier le code ci-dessus est d'écrire le code suivant :

```
<!DOCTYPE html>
<html>
        <head>
```

```
                <meta charset="utf-8">
                <script>
                function affichehello() {
                        document.getElementById("texte").innerHTML = "Hello World!";
                }
                </script>
        </head>
        <body>
                <p id="texte"></p>
                <script>
                affichehello();
                </script>
        </body>
</html>
```

La fonction est tout d'abord lue dans la zone d'entête sans être exécutée. Elle sera exécutée dès qu'elle sera appelée, ce qui est le principe des fonctions. Elle est appelée après la balise <p> et le navigateur connait alors l'*id='texte'*.

Plus élégant encore, l'usage de l'évènement *onload* qui appelle la fonction seulement après le chargement de tout le contenu de <body>.

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <script>
                function affichehello() {
                        document.getElementById("texte").innerHTML = "Hello World!";
                }
                </script>
        </head>
        <body onload="affichehello()">
                <p id="texte"></p>
        </body>
</html>
```

Pour finir sur les sommets de l'élégance, la fonction affublée d'un paramètre pour écrire le texte de son choix.

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <script>
                function affiche(deschoses) {
                        document.getElementById("texte").innerHTML = deschoses;
                }
                </script>
        </head>
        <body onload="affiche('Sean Connery a la classe !')">
                <p id="texte"></p>
        </body>
</html>
```

Retenons les points suivants :

- La balise <script> indique que le langage JavaScript sera utilisé dans le contenu de cette balise.
- Il est préférable de placer les scripts en bas de la page juste avant </body>, car bien souvent, le JavaScript agit sur des balises de la page qui doivent être chargées en premier. On peut aussi utiliser l'événement *onload* sur la balise <body>.
- Comme pour le PHP, chaque instruction se termine par un point-virgule, mais un retour à la ligne fonctionne également.
- Les chaînes de caractères sont placées entre guillemets ou entre apostrophes.

Il est également possible de placer du code JavaScript dans un fichier externe avec l'extension *.js* :

Fichier principal : **page.html**

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<script src="truc.js"></script>
	</head>
	<body>
		<header id="debut"></header>
		<script>
		miaou();
		</script>
	</body>
</html>
```

Fichier relié : **truc.js**

```
function miaou() {
	document.getElementById("debut").innerHTML = "<h1>Miaouuu !</h1>";
}
```

Syntaxe du langage

Elle est très proche de celle du langage C ou PHP.

Types et variables

Les types de variables peuvent être :

- Number (nombres entiers ou à virgule) :

  1, 0, -5, 10.85, -5.9

- String (chaine de caractère, texte) :

  "*toto*", "", '*cuicui*'

- Boolean (booléen) :

  true, false

Les mots clés *const*, *let* et *var* permettent la déclaration des variables.

Exemples :

```
const a = 2;
const b = "coucou";
let truc = true;
let truc = 450.53;
var c = 55
```

Si la valeur de la variable ne change pas durant l'exécution du programme, utilisez *const*. Si la valeur de la variable est amenée à changer, utilisez *let*.

```
const pi = 3.14159;
pi = 3.14; // cette ligne provoque une erreur car pi est en lecture seule (const)

let age = 30;
age = 31; // cette ligne fonctionne car age peut être réassigné (let)
```

L’instruction let permet de déclarer une variable locale dont la portée est celle du bloc courant :

```
let i = 1;

if (i == 1) {
        let i = 2;
        alert(i); // affiche 2
}

alert(i); // puis 1
```

A l’inverse, l’instruction var permet de déclarer une variable globale :

```
var i = 1;

if (i == 1) {
        var i = 2;
        alert(i); // affiche 2
}

alert(i); // puis 2
```

Attention, le langage *JavaScript*, comme le langage *C*, est sensible à la casse : la variable *maChaine* est différente de *Machaine*, par exemple.

Les tableaux

Les tableaux sont des variables numérotées qui utilisent le même nom auquel on ajoute un numéro entre crochets.

```
const toto = new Array(3); // Tableau déclaré pour 3 éléments. Attention au A majuscule
toto[0] = 12; // on commence à l’indice 0
toto[1] = "coucou"; // chaque élément peut être de type différent
toto[2] = 10.5;
alert(toto.length); // Affiche le nombre d’éléments : 3 (de 0 à 2)
```

Voici un exemple où le tableau est utilisé dans une boucle for :

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
        </head>
        <body>
        <script>
        const photos = new Array(3);
        photos[0] = "chat";
        photos[1] = "chien";
        photos[2] = "hamster";
        for (let i = 0 ; i < 3 ; i++)
            alert(photos[i]);
        </script>
        </body>
</html>
```

Un exemple équivalent où le tableau est déclaré et alimenté directement :

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
        </head>
        <body>
        <script>
        const photos = ["chat", "chien", "hamster"];
```

```
        for (let i = 0 ; i < 3 ; i++)
            alert(photos[i]);
        </script>
        </body>
</html>
```

Opérateurs arithmétiques

Le langage JavaScript utilise les opérateurs arithmétiques traditionnels :

toto + titi	Addition	Somme de toto et titi
toto – titi	Soustraction	Reste de la différence de titi et toto
toto * titi	Multiplication	Produit de toto par titi
toto / titi	Division	Quotient de toto par titi

Les instructions conditionnelles

On appelle structure conditionnelle les instructions qui permettent de tester si une condition est vraie ou non, ce qui permet de donner de l'interactivité à vos scripts par exemple.

L'instruction if

L'instruction if est la structure de test la plus basique, on la retrouve dans tous les langages. Elle permet d'exécuter une série d'instructions si jamais une condition est réalisée.

La syntaxe de cette expression est la suivante :

```
if (condition réalisée) {
      Liste d'instructions;
}
```

Remarques :

- La condition doit être entre des parenthèses.
- S'il n'y a qu'une instruction, les accolades ne sont pas nécessaires.

Il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||) :

```
if (condition1 && condition2)
// teste si les deux conditions sont vraies
```

```
if (condition1 || condition2)
// exécute les instructions si l'une ou l'autre des deux conditions est vraie
```

L'instruction if … else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter en cas de non réalisation de la condition…

L'expression *if … else* permet d'exécuter une autre série d'instructions en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante :

```
if (condition réalisée) {
     Liste d'instructions;
}
else {
     Autre série d'instructions;
}
```

Les boucles

Les boucles permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée.

La façon la plus commune de faire une boucle, est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter un certain nombre de fois la même série d'instructions.

Trois paramètres sont à préciser dans la syntaxe for :

- le nom de la variable qui sert de compteur et sa valeur de départ,
- la condition pour laquelle la boucle s'arrête (condition qui teste si la valeur du compteur dépasse une limite),
- une instruction qui incrémente (ou décrémente) le compteur.

```
for (départ du compteur; condition; pas du compteur) {
     liste d'instructions;
}
```

Exemple :

```
for (let i = 1; i < 4; i++) {
        alert(i);
}
```

Cette boucle affiche 3 fois la valeur de i, c'est-à-dire *1, 2, 3*

Elle commence à *i=1*, vérifie que *i* est bien strictement inférieur à *4*, jusqu'à atteindre la valeur *i=4* pour laquelle la condition ne sera plus réalisée, la boucle s'arrêtera alors et le programme se poursuivra après la boucle.

L'instruction *alert(i)* dans la boucle est un bon moyen pour vérifier la valeur du compteur pas à pas.

Il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle :

```
for (let i = 0; i <= 10; i++)
```

exécute 11 fois la boucle (*i* de *0* à *10* inclus)

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

Syntaxe :

```
while (condition réalisée) {
      liste d'instructions;
}
```

Cette instruction exécute la liste d'instructions tant que (*while* est un mot anglais qui signifie tant que) la condition est réalisée. La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur (voire une brèche dans l'espace-temps).

Les boîtes d'alerte

Boite d'alerte

```
alert("message d'alerte");
```

Affiche une simple boîte de message avec le texte « message d'alerte »

Confirmation

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
	</head>
	<body>
		<script>
		if (confirm("L'élément sera supprimé. Êtes-vous sûr ?"))
			document.write("Élément supprimé.");
		else
			document.write("Suppression annulée.");
		</script>
	</body>
</html>
```

Cette fonction affiche une boîte de message et offre à l'utilisateur la possibilité de confirmer (*OK* ou *Annuler*). Elle retourne *true* si l'utilisateur clique sur *OK*, *false* s'il clique sur *Annuler*.