Chapitre 5
JavaScript orienté WebApp

1. Les variables

1.1 Quelques précisions

Inutile de refaire un cours complet de JavaScript ici, ce n'est pas le but de ce livre. Seuls certains points bien précis sont abordés ici, car ils sont très utiles, voire nécessaires, pour la conception d'une WebApp.

Pour les lecteurs qui souhaitent trouver les notions fondamentales du développement en JavaScript, les Éditions ENI éditent des livres sur le sujet, notamment le livre "Apprendre à développer avec JavaScript" dans la collection Ressources Informatiques.

Pour commencer, pas besoin d'éditeur de texte, les premiers exemples ne nécessitent que la console du navigateur. Pour cela, il faut ouvrir un nouvel onglet, accéder aux outils de développement en appuyant sur [F12], puis cliquer sur l'onglet **Console**. Les commandes sont entrées en bas et le résultat s'affiche à la suite.

1.2 La déclaration implicite

Si l'on se laisse aller, JavaScript est d'une redoutable simplicité quant à l'utilisation des variables. Il suffit d'en nommer une pour qu'elle existe.

```
a=15
```

Il est alors possible de l'afficher. Pour cela, il suffit d'entrer le nom de cette variable.

```
a
```

Par contre, il n'est pas possible d'utiliser ou d'afficher une variable qui n'existe pas. Les deux exemples qui suivent provoqueront une erreur de référence.

```
a=z
```

```
y
```

Ceci mène rapidement à un joli chaos, car toute variable déclarée implicitement a une portée globale. Il est donc nécessaire de mettre un peu d'ordre dans tout cela. Heureusement, les mots-clés `var` et `let` sont là pour y parvenir.

1.3 var

Utilisé en dehors d'une fonction, `var` crée une variable globale, elle pourra donc être vue dans toutes les fonctions. On obtient le même résultat avec une déclaration implicite, mais le procédé de stockage en mémoire est différent.

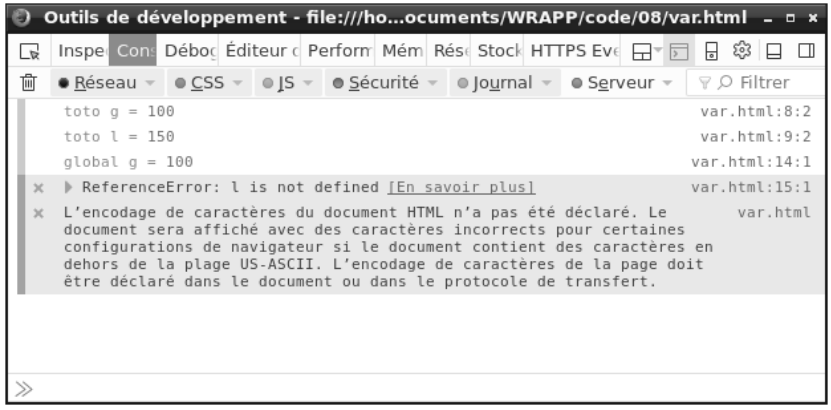
En revanche, lorsque l'on utilise `var` pour déclarer une variable au sein d'une fonction, cela devient une variable locale.

L'exemple suivant nécessite d'être sauvé dans un fichier HTML afin de pouvoir être interprété par le navigateur. Il faut ensuite afficher la console de développement pour voir le résultat de l'exécution du code, c'est ainsi que fonctionnent tous les exemples de ce chapitre.

```
<script>

var g=100; // Variable globale

function toto()
{
 var l=150; // Variable locale
 console.log("toto g =",g);
 console.log("toto l =",l);
}

toto();

console.log("global g =",g);
console.log("global l =",l); // Erreur : l n'est pas globale

</script>
```

Deux variables sont déclarées : `g` qui est globale et `l` qui est locale à la fonction `toto`. Lors de l'exécution, les variables `g` et `l` sont bien visibles depuis la fonction `toto`. On peut aussi afficher `g` au niveau global.

Par contre, l'affichage de `l` au niveau global provoque une erreur, car la variable n'a pas été déclarée à ce niveau.



Le second message d'erreur est dû au fait que la page HTML ne contient pas d'indication d'encodage des caractères. Ce qui n'est pas un problème pour réaliser des tests sur des variables.

Il est possible de déclarer une variable `l` au niveau global pour qu'il n'y ait plus d'erreur. Cette variable a alors deux valeurs différentes selon que l'on est au niveau global ou local.

```
var g=100; // Variable globale
var l=200; // Autre variable globale
```

Il suffit de retirer le mot-clé `var` devant les déclarations de variables pour se rendre compte qu'il est vraiment important de les utiliser systématiquement.

```
<script>

g=100; // Variable globale
l=200; // Autre variable globale

function toto()
{
 l=150; // Variable qui sera aussi globale !
 console.log("toto g =",g);
 console.log("toto l =",l);
}

toto();

console.log("global g =",g);
console.log("global l =",l);

</script>
```

Le résultat est alors le suivant.

La déclaration de `l` dans la fonction a écrasé la déclaration de `l` au niveau global.

Il faut toujours veiller à ne pas oublier de déclarer au moyen du mot-clé `var` toutes les variables qui sont utilisées au niveau d'une fonction.

1.4 let

L'inconvénient d'une déclaration locale avec le mot-clé `var`, c'est qu'elle est valable pour la totalité de la fonction. Pourtant, il peut être intéressant de définir des variables qui sont locales au niveau d'un bloc, comme on peut le faire en C, C++, Java, ou d'autres langages.

Le mot-clé `let` permet donc de déclarer des variables avec une plus grande finesse, en limitant leur portée au bloc d'accolades dans lequel elles ont été déclarées. C'est ce qui est fait dans cet exemple qui se charge d'afficher une table de multiplication dans la console.

```
<script>
mul();

function mul()
{
	console.log("TABLE DE MULTIPLICATION");
	for(let a=1;a<=10;a++) // Boucle A
	{
		let text="";
		for(let b=1; b<=10; b++) // Boucle B
		{
			text+=(a*b)+" ";
			if(a*b<10) text+=" ";
		}
		console.log(text);
//		console.log(a,b); // Provoque une erreur
	}
//	console.log(text); // Provoque une erreur
}
</script>
```

La variable locale `text` est définie à l'intérieur de la `Boucle A`. Si on décommente la ligne `console.log(text)`, située en dehors de la `Boucle A`, on aura un message d'erreur, car `text` n'est définie que dans la `Boucle A`. Par contre si l'on remplace la ligne `let text=""` par `var text=""`, il n'y aura plus d'erreur.