Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

Hoisting

Le hoisting (*en français, "hissage"*) est un terme que vous *ne* trouverez dans aucune prose de spécification normative avant l'ECMAScript® 2015 . Le hissage a été conçu comme une façon générale de penser à la manière dont les contextes d'exécution (précisément, les phases de création et d'exécution) fonctionnent en JavaScript. Toutefois, le concept peut être un peu déroutant à première vue.

Conceptuellement, par exemple, une définition stricte du hissage suggère que les déclarations de variables et de fonctions sont déplacées physiquement en haut de votre code, mais ce n'est pas ce qui se passe en fait. A la place, les déclarations de variables et de fonctions sont mises en mémoire pendant la phase de *compilation*, mais restent exactement là où vous les avez tapées dans votre code.

En apprendre plus

Exemple technique

L'un des avantages du fait que JavaScript met en mémoire les déclarations des fonctions avant d'exécuter un quelconque segment de code, est que cela vous permet d'utiliser une fonction avant que nous ne la déclariez dans votre code. Par exemple :

```
function nomChat(nom) {
  console.log("Le nom de mon chat est " + nom);
}

nomChat("Tigrou");
/*
Le résultat du code ci-dessus est : "Le nom de mon chat est Tigrou"
*/
```

Le fragment de code ci-dessus est la façon dont vous vous attendez à écrire le code pour qu'il fonctionne. Voyons maintenant ce qui se passe quand nous appelons la fonction avant de la déclarer :

```
nomChat("Chloé");

function nomChat(nom) {
  console.log("Le nom de mon chat est " + nom);
}
/*
Le résultat du code ci-dessus est : "Le nom de mon chat est Chloé"
*/
```

Même si nous appelons d'abord la fonction dans notre code, avant que la fonction ne soit écrite, le code fonctionne néanmoins. Cela est dû à la façon dont l'exécution de contexte fonctionne en Javascript.

Le hissage fonctionne tout aussi bien avec d'autres types de données et d'autres variables. Les variables peuvent être initialisées et utilisées avant d'être déclarées. Mais elles ne peuvent pas être utilisées sans initialisation.

Exemple technique

```
num = 6;
num + 7;
var num;
/* Ne donne aucune erreur tant que num est déclarée*/
```

JavaScript hisse seulement les déclarations, pas les initialisations. Si vous utilisez une variable qui est déclarée et initialisée après l'avoir utilisée, sa valeur sera indéfinie. Les deux exemples ci-dessous montrent le même comportement.

```
var x = 1; // Initialise x
console.log(x + " " + y); // Affiche '1 undefined'
var y = 2; // Initialise y


// Le code suivant se comportera de la même façon que le code précédent:
var x = 1; // Initialise x

var y; // Déclare y
console.log(x + " " + y); // Affiche '1 undefined'
y = 2; // Initialise y
```

Références techniques

- JavaScript: Understanding the Weird Parts - Cours d'Udemy.com
- instruction var - MDN
- déclaration function - MDN

declaration function - MDN

Last modified: 23 févr. 2022, by MDN contributors