4 minutes

Les variables let et const

Les problématiques liées à `var`

En `JavaScript` moderne, il ne faut plus utiliser le `var`.

Nous l'avons présenté car vous le rencontrerez souvent dans des anciennes librairies etc.

Mais la recommandation est d'utiliser le plus possible `const` et quand c'est impossible d'utiliser `let`.

Ces recommandations sont celles de tous les meilleurs experts du langage `JavaScript` : `Douglas Crockford`, `Google`, `Airbnb` etc.

Les codes d'exemple ci-dessous montre quelques uns des principaux problèmes avec l'utilisation des variables.

C'est normal que vous ne les compreniez pas encore car nous n'avons presque rien vu ! Vous pourrez y revenir plus tard quand nous aurons vu les boucles, les fonctions, la portée etc :

https://codesandbox.io/embed/js-c3-l3-1-knkrv

L'utilisation de `const`

`const` permet de déclarer une variable **en créant une référence en lecture seule vers la valeur, et a une portée de bloc**.

Il est obligatoire d'assigner une valeur à la `const` lors de sa déclaration.

Les `const` doivent être utilisées partout où cela est possible.

Avoir une référence en lecture seule signifie que si l'on tente de modifier la référence vers lequel porte une `const` une erreur sera levée :

https://codesandbox.io/embed/js-c3-l3-2-lpg0p

L'interpréteur lève l'exception : la valeur est en lecture seule. C'est pour cette raison que l'on parle de constante.

De la même manière, vous aurez une erreur si vous essayez de déclarer une autre constante avec le même nom :

https://codesandbox.io/embed/js-c3-l3-3-gbksx

Vous aurez enfin une erreur si vous n'assignez pas de valeur à `const` :

https://codesandbox.io/embed/js-c3-l3-3-7d0eg

Par contre, et nous verrons pourquoi dans ce chapitre, il est possible de faire :

```
const personne = {
  prenom: 'Paul'
}

personne.prenom = 'Dupont';

console.log(personne);
```

Copier

Nous verrons ainsi pourquoi les tableaux, les objets, les `Map`, les `Set` (et les `WeakMap` et les `WeakSet`) peuvent être modifiés même lorsque nous utilisons `const`.

La raison en deux mots est que la référence est en lecture seule, mais ici la référence à l'objet n'a pas changé lorsque nous l'avons modifié !

En effet, **il n'est pas possible de modifier la référence associée à l'identifiant de la variable, mais la valeur contenue peut être modifiée**.

Cela sera plus clair lorsque nous verrons la différence entre les valeurs dites primitives et les objets.

L'utilisation de `let`

`let` permet de créer une variable avec une portée de bloc.

Il n'est pas obligatoire d'assigner la variable `let` lors de sa déclaration.

Avec les variables `let`, il est possible de modifier la référence et donc de modifier la valeur primitive contenue dans la variable :

```
let test = 50;
test = 51; // Tout va bien la réaffectation est possible.
```

Copier

Par contre, il y aura une erreur si vous déclarez deux variables avec le même identifiant :

```
let test = 50;
let test = 51; // Erreur : l’identifiant test a déjà été déclaré.
```

Copier

Nous reverrons en détails la portée de bloc mais notez déjà :

```
let test = false; // première variable globale
if (true) {
  let test = true // une seconde variable est déclarée mais limitée dans le bloc.
  console.log(test); // true => car nous sommes dans un bloc { }
}
console.log(test) // false => pas impacté par la variable dans un bloc
```

Copier

Ce que vous devez retenir de cette leçon pour le moment : nous n'utilisons plus `var` en `JavaScript` moderne. Vous devez utiliser `const` la majorité du temps. Vous devez utiliser `let` quand vous ne pouvez pas utiliser `const`.