Les fonctions fléchées

Cette leçon est très importante, les fonctions fléchées sont les fonctions que vous utiliserez le plus en `JavaScript`.

Elles sont très concises, flexibles et il est facile de raisonner avec elles. Il s'agit d'une excellente fonctionnalité que nous recommandons fortement.

La syntaxe des fonctions fléchées

Les fonctions fléchées sont des fonctions anonymes (sans nom) qui ont une syntaxe plus concise :

```
(param1, param2) => {
  // instruction 1;
  // instruction 2;
}
```

Lorsque qu'il n'y a **qu'un seul paramètre, il est possible d'enlever les parenthèses l'entourant** :

```
param1 => {
  // instruction 1;
  // instruction 2;
  return valeur;
}
```

Si il n'y a qu'une seule instruction, il est également possible d'omettre les accolades et même le mot clé `return`.

Dans ce cas, la valeur retournée sera le résultat de l'évaluation de l'expression. On appelle cela le retour implicite.

```
param1 => instruction;
```

Par exemple :

```
const doubler = nombre => nombre * 2;
doubler(21); // 42
```

Valeur de `this`

La valeur de `this` dans une fonction fléchée est liée à son environnement lexical et non au contexte d'exécution contrairement aux fonctions `function`.

Il est donc très intuitif de raisonner sur la valeur de `this` dans la plupart des cas.

Si elle est déclarée sur un objet ou en global, la valeur de `this` sera l'objet global, même en mode strict :

```
"use strict"
const a = () => console.log(this === window);
a(); // true

const b = {
  methode: () => console.log(this === window)
}
b.methode(); // true
```

Dans une fonction elle aura pour valeur le `this` de la fonction.

Donc en mode normal, pour rappel, dans une invocation simple de fonction `this` vaut l'objet global :

```
function func() {
  const a = () => console.log(this === window);
  a();
}
func(); // true
```

En mode strict, ce sera donc `undefined` :

```
"use strict"

function func() {
  const a = () => console.log(this === undefined);
  a();
}
func(); // true
```

Comme une fonction fléchée n'a pas de `this` propre, mais qu'elle utilise celui de sa portée parente, il n'est pas possible d'utiliser `call()`, `apply()` ou `bind()` avec les fonctions fléchées.

Retourner un objet littéral

Pour retourner un objet littéral lorsque vous utilisez une fonction fléchée avec le retour implicite il vous faudra utiliser des parenthèses :

```
const fonction = () => { a: 1 };
fonction(); // undefined
```

En effet, sans parenthèses comme ci-dessus, l'interpréteur `JavaScript` pense que vous déclarez le corps de la fonction. Dans ce cas, il n'y a donc pas de retour implicite, et comme nous n'utilisons pas `return`, la valeur de retour est `undefined`.

Avec des parenthèses, le retour implicite fonctionne car l'interpréteur sait que nous ne déclarons pas le corps de la fonction :

```
const fonction = () => ({ a: 1 });
fonction(); // { a: 1 }
```

Recommandations d'utilisation

Nous recommandons d'utiliser les fonctions fléchées partout où cela est possible sauf dans les cas suivants :

La définition de méthodes sur les objets. Il faut conserver l'utilisation du raccourci syntaxique car dans le cas contraire vous n'aurez pas le comportement attendu pour `this` qui vaudra l'objet global et non pas l'objet. Il faut garder cette écriture :

```
const obj = {
  maMethode() {
    console.log(this === obj); // => true : comportement attendu
  }
};
obj.maMethode();
```

Pour les constructeurs, que nous verrons plus tard. Les fonctions fléchées ne peuvent pas faire office de fonction constructrice.