Apprendre Javascript - Les fonctions

Les fonctions

Les fonctions sont des blocs de code qui peuvent être appelés pour être exécutés. Elles sont très utiles pour réutiliser du code, et pour rendre le code plus lisible et plus facile à maintenir.

Déclarer une fonction

Pour déclarer une fonction, on utilise le mot-clé `function`, suivi du nom de la fonction, suivi des parenthèses `()`, et enfin des accolades `{}` pour délimiter le bloc de code de la fonction.

```
function maFonction() {
    console.log("Hello world");
}
```

Appeler une fonction (l'exécuter)

Pour appeler une fonction, on utilise son nom suivi des parenthèses `()`.

```
maFonction();
```

Les paramètres

Les fonctions peuvent prendre des paramètres, qui sont des valeurs que l'on passe à la fonction pour qu'elle les utilise. Ce sont comme des variables qui sont déclarées dans la déclaration de la fonction, et qui prennent la valeur des arguments que l'on passe à la fonction quand on l'appelle.

On peut déclarer autant de paramètres que l'on veut, en les séparant par des virgules.

```
function direBonjour(prenom, nom) {
    console.log(`Bonjour ${prenom} ${nom}`);
}
```

Valeurs par défaut des paramètres

On peut donner des valeurs par défaut aux paramètres, qui seront utilisées si on n'en passe pas à la fonction. On utilise l'opérateur `=` pour cela. Si on ne passe pas de valeur à un paramètre, il prendra la valeur par défaut.

Cependant, l'ordre des paramètres doit être respecté.

```
function direBonjour(prenom = "inconnu", nom = "inconnu") {
    console.log(`Bonjour ${prenom} ${nom}`);
}

direBonjour("Jean", "Dupont"); // Affiche "Bonjour Jean"
direBonjour(); // Affiche "Bonjour inconu inconnu"
```

La valeur de retour

Les fonctions peuvent retourner une valeur, qui est la valeur que la fonction renvoie quand on l'appelle. Pour cela, on utilise le mot-clé `return`, suivi de la valeur que l'on veut retourner. Si on ne met pas de `return`, la fonction retourne `undefined`. On peut retourner n'importe quelle valeur, y compris des objets, des tableaux, ou même d'autres fonctions.

On peut ensuite stocker la valeur retournée dans une variable, ou l'utiliser directement.

```
function direBonjour(prenom = "inconnu", nom = "inconnu") {
    return `Bonjour ${prenom} ${nom}`;
}

direBonjour("Jean", "Dupont"); // Retourne "Bonjour Jean Dupont" mais ne fait rien
car on ne stocke pas la valeur retournée
let message = direBonjour("Jean", "Dupont"); // Stocke la valeur retournée dans la
variable message
console.log(message); // Affiche "Bonjour Jean Dupont"
```

Utilité de la valeur de retour

Les fonctions qui retournent une valeur sont très utiles pour traiter des données, ou pour effectuer des opérations sur des données. Par exemple, on peut utiliser une fonction pour calculer un résultat, et retourner ce résultat pour l'utiliser ailleurs dans le code.

La fonction retournera toujours le même résultat pour les mêmes paramètres, ce qui permet de l'utiliser partout où on a besoin de ce résultat. On appelera cela une fonction pure.

```
function calculerPrixAvecTaxes(prixHT, tauxTaxes) {
    return (prixHT * tauxTaxes) / 100;
}

function trouverLaMoyenne(notes) {
    let somme = 0;
    for (let note of notes) {
        somme += note;
```

```
    }
    return somme / notes.length;
}
```

Portée des variables dans une fonction et un bloc d'instructions (if, for, while, switch, etc.)

Les variables déclarées dans une fonction sont locales à cette fonction, c'est-à-dire qu'elles ne sont accessibles que dans cette fonction. Elles ne sont pas accessibles en dehors de la fonction.

Celà permet de ne pas polluer l'espace de nom global avec des variables qui ne sont utilisées que dans une fonction et d'éviter d'écraser des variables globales.

En d'autres mots, on peut redéclarer une variable dans une fonction sans que cela n'ait d'impact sur la variable globale du même nom.

```
let x = 10; // On déclare une variable x en dehors de la fonction

function maFonction() {
    let x = 20; // On redéclare x, mais ce n'est pas la même variable que celle déclarée en dehors de la fonction

    console.log(x); // Affiche 20
    if (true) {
        let x = 30; // On redéclare x, mais ce n'est pas la même variable que celle déclarée dans la fonction ou celle déclarée en dehors de la fonction
        console.log(x); // Affiche 30
    }
}

console.log(x); // Affiche 10
```

Utilisation de `var` pour déclarer des variables

Avant ES6, on utilisait le mot-clé `var` pour déclarer des variables. Les variables déclarées avec `var` sont accessibles dans toute la fonction dans laquelle elles sont déclarées, mais pas en dehors de cette fonction. Cependant, les blocs d'instructions (if, for, while, switch, etc.) n'ont pas de portée propre, et les variables déclarées avec `var` dans un bloc d'instructions sont accessibles en dehors de ce bloc. **C'EST POURQUOI IL EST RECOMMANDÉ D'UTILISER `let` ET `const` À LA PLACE DE `var`.**

```
function maFonction() {
    var x = 20; // On déclare x avec var

    console.log(x); // Affiche 20
    if (true) {
```

```
        var x = 30; // On redéclare x, mais c'est la même variable que celle
déclarée dans la fonction. On écrase l'espace mémoire de la variable x
        console.log(x); // Affiche 30
    }
    console.log(x); // Affiche 30
}
```

Fonctions anonymes

Les fonctions déclarées sous forme d'expression peuvent être anonymes, c'est-à-dire qu'elles n'ont pas de nom. Cela est utile pour stocker une fonction dans une variable, ou pour passer une fonction en argument à une autre fonction.

Les fonctions anonymes sont souvent utilisées pour les événements ou pour les fonctions de rappel (callback) (voir le point suivant).

```
let maFonction = function () {
    console.log("Hello world");
};

// Sert à ajouter un écouteur de clic à la fenêtre
window.addEventListener("click", function () {
    console.log("Click !");
});

// Sert à exécuter une fonction après un certain délai
setTimeout(function () {
    console.log("3 secondes se sont écoulées");
}, 3000);
```

Fonctions de rappel (callback)

Les fonctions de rappel sont des fonctions qui sont passées en paramètre à une autre fonction, et qui seront appelées par cette fonction à un moment donné.

Au moment de déclarer la fonction de rappel, on ne met pas les parenthèses `()` après le nom de la fonction, sinon la fonction serait exécutée immédiatement. On met seulement le nom de la fonction, sans les parenthèses. C'est un peu comme si on mettait une fonction dans une variable temporairement pour pouvoir l'exécuter plus tard.

L'avantage est aussi qu'on peut remplacer la fonction de rappel selon les besoins, sans avoir à modifier la fonction qui appelle la fonction de rappel.

```
function direBonjour(prenom, callback) {
    console.log(`Bonjour ${prenom}`);
```

```
    // On appelle la fonction de rappel
    callback();
}

function direAuRevoir() {
    console.log("Au revoir");
}

function direSalut() {
    console.log("Salut");
}
direBonjour("Jean", direAuRevoir); // Affiche "Bonjour Jean" puis "Au revoir"
direBonjour("Maxime", direSalut); // Affiche "Bonjour Maxime" puis "Salut"
```

Fonctions fléchées

Les fonctions fléchées sont une autre manière de déclarer des fonctions. Elles sont plus courtes et plus lisibles que les fonctions classiques, et elles ont une portée différente des variables. Elles sont plus récentes que les fonctions classiques, et elles ont été introduites avec ES6.

Pour déclarer une fonction fléchée, on enlève le mot-clé `function` et on utilise une flèche `=>` entre les parenthèses `()` et les accolades `{}`.

Si la fonction ne prend pas de paramètres, on met seulement les parenthèses `()`, et si la fonction ne fait qu'un retour, on peut omettre les accolades `{}` et le mot-clé `return`.

```
let direBonjour = (prenom, nom) => {
    return `Bonjour ${prenom} ${nom}`;
};
//Version simplifiée
let direBonjour = (prenom) => console.log(`Bonjour ${prenom}`);
```

Pour le moment, nous allons utiliser uniquement les fonctions classiques car nous verrons le contexte (`this`) des fonctions fléchées plus tard à la fin de la session.