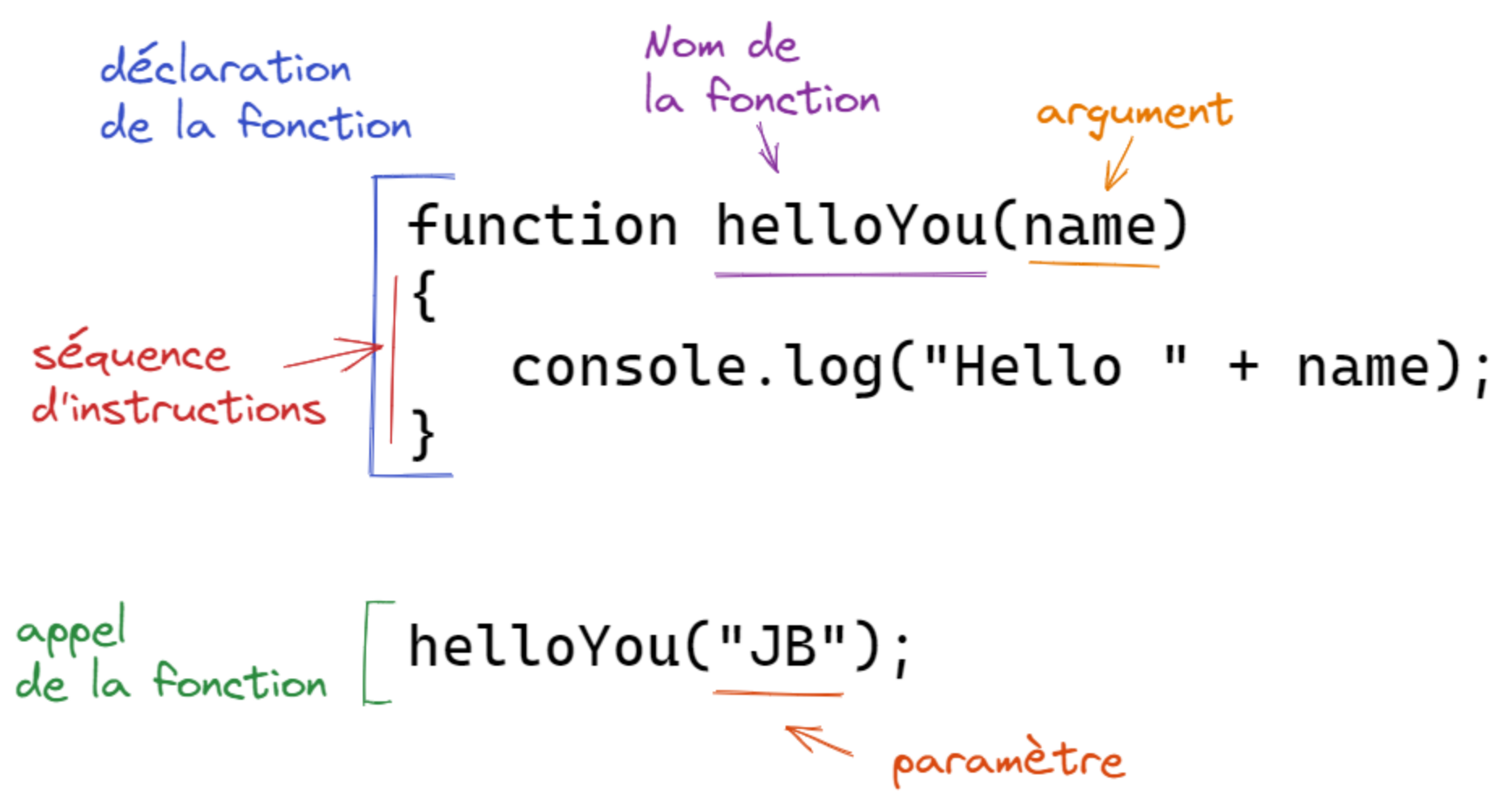
Dynamiser vos sites avec Javascript - Les fonctions anonymes, fléchées et récursives en JS

Rappels sur les fonctions



Des précisions

Une fonction permet d'**effectuer des opérations répétitives**.

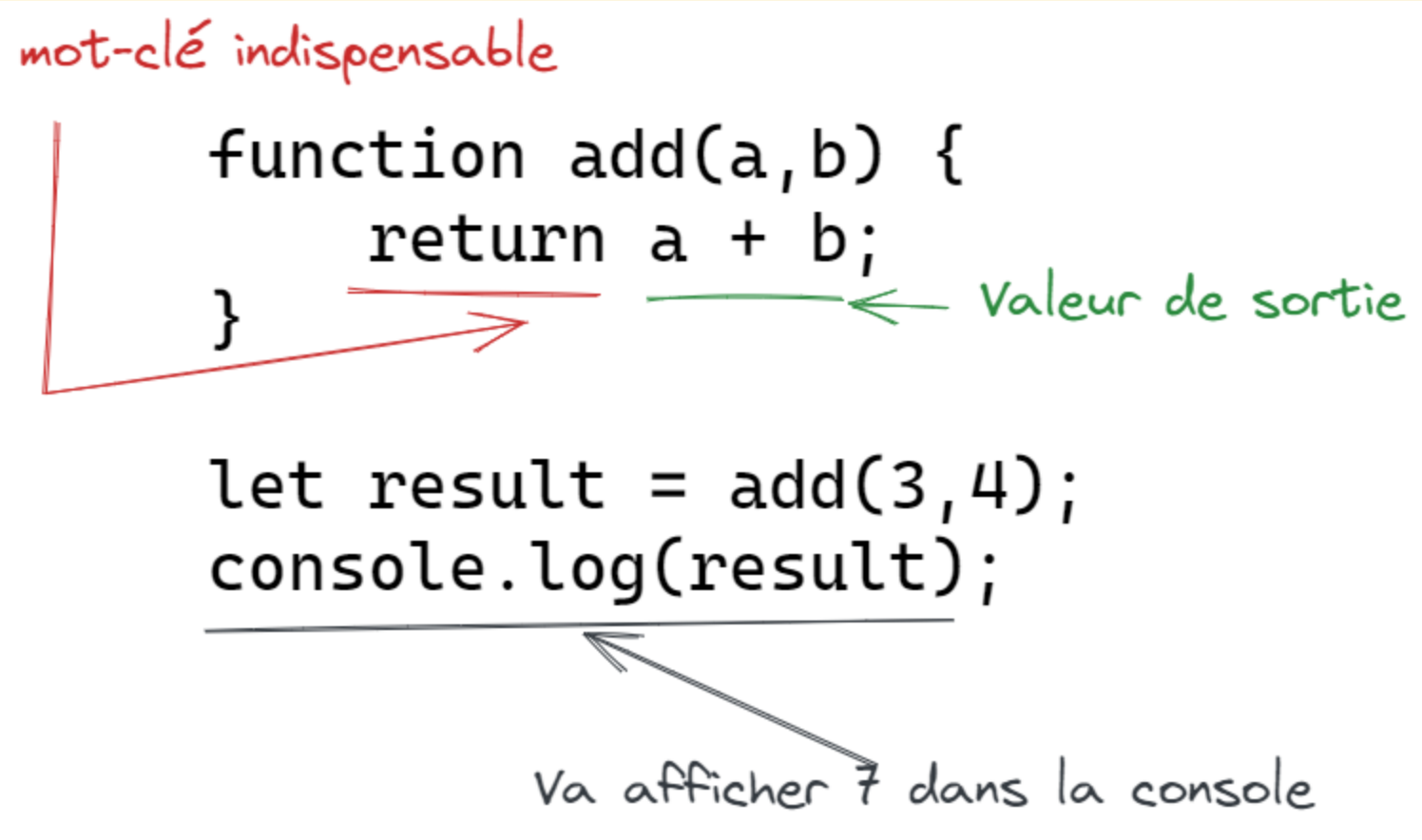
Plutôt que d'écrire sans arrêt un même sous-programme, on va l'englober dans une **séquence d'instructions**.

En **déclarant** un nom à cette séquence, on va permettre à chaque **appel** de ce nom suivi de `()` d'exécuter la dite séquence.

On peut personnaliser l'exécution de la fonction grâce à des **paramètres lors de l'appel**.

Ces paramètres seront juxtaposés aux **arguments de la déclaration**.

Le retour d'une fonction



Parfois, on veut **récupérer la sortie** de l'exécution d'une fonction.
On appelle ça **un retour**.

NB: On ne peut retourner qu'une seule valeur.

Les fonctions anonymes

C'est quoi: Une fonction qui n'a pas de nom

Utilité: lorsque le code de notre fonction n'est appelé que peu de fois, et n'est pas réutilisé

```
function() {
    console.log("coucou toi!");
}
```

NB: A contrario, une fonction qui intègre son nom dans la déclaration s'appelle **une fonction nommée**

Appeler une fonction anonyme

2 façons courantes:

- Enfermer le code de notre fonction dans une variable et utiliser la variable comme une fonction
- Utiliser une autre fonction pour déclencher son exécution

Stocker dans une variable

```
let helloYou = function() {
    console.log("Coucou toi!");
}

helloYou();
helloYou();
helloYou();

helloYou = 3;

helloYou(); // BIPPPPPP ERREUR.
/* A ce moment-là, la variable helloYou ne contient plus une fonction...
...mais un nombre */
```

Aucune différence d'exécution par rapport à une fonction nommée!

Déclencher exécution par une autre fonction

```
function botSpeaking(myAnonymousFunction) {
    console.log("bipbip!");
    myAnonymousFunction();
}

botSpeaking(function() {
    console.log("Je vais faire des calculs...");
    // des calculs
})

botSpeaking(function() {
    // Processus de folie...
    alert("Je vais conquérir le monnnnnnde! MOUAHAHA");
})

// Chaque fonction anonyme s'exécutera après l'affichage de `bipbip!`
```

Les fonctions fléchées

Toute fonction peut être écrite différemment, avec une syntaxe plus courte.

Comment? grâce à une spécification apportée par l'**ES6** (JS 2015)

```
function sayHello() {
    console.log("coucou Toi!");
}

// C'EST LA MEME CHOSE
let sayHello = () => {
    console.log("coucou Toi!");
}
```

D'autres raccourcis

- Pas besoin de `()` si 1 seul argument.

```
let sayHello = name => {
    console.log("coucou " + name);
}
```

- Si une seule ligne de code, pas besoin de `{}`

```
let sayHello = name => console.log("coucou " + name);
```

- Si expression simple (sur une ligne), pas besoin du mot clé return

```
function add(a,b) {
    return a+b;
}

let add = (a,b) => a+b;
```

- Si retour d'une expression litérale comme un objet, alors l'entourer de `()`

```
// ERROR!
let descriptionToto = () => {age: 2, name: "Toto"}

// GOOD
let descriptionToto = () => ({age: 2, name: "Toto"})
```

Les fonctions récursives

C'est quoi la récursivité?

Quand une fonction s'appelle elle-même jusqu'à atteindre une condition d'arrêt

```
function decompte( ... ) {
    ...
    decompte( ... );
    ...
}
```

L'intérêt

S'apparente à des boucles, mais plus performante qu'elles.

Surtout si boucles dans des boucles.

```
// Un exemple de boucle dans une boucle
for(let i=0; i<9; i++) {
    for(let j=0;j<9;j++) {
        // ...
    }
}
```

NB: On parle alors de **complexité d'un algorithme**

Une fonction de compte à rebours

```
// Algo classique
function countdown(nb) {
    for(let i=nb; i>=0; i--) {
        console.log(i);
    }
}


// Algo récursif
function countdown_rec(nb) {
    if(nb >= 0) {
        console.log(nb);
        countdown_rec(nb-1);
    }
}
```

Les 3 besoins d'une fonction résursive

- **La condition d'arrêt**: trouver quand la récursivité doit stopper
- **La résolution du problème**: réduire le problème à sa plus petite forme
- **L'appel récursif**: changement obligatoire dans les paramètres de son appel!