

[Welcome home !](#)[Règlement intérieur](#)[Poster une issue](#)[JavaScript](#)[Environnement de dev](#)[👋 Semaine 1](#)[Terminal](#)[Variables](#)[Conditions](#)[Boucles](#)[👋 Semaine 2](#)[Manipuler des chaînes de caractères](#)[Fonctions](#)[🎥 Vidéos](#)[Introduction et concept](#)[Notations possibles](#)[Déclaration et Appel](#)[Ajouter des paramètres à vos fonctions](#)[Valeur de retour](#)[Tableaux](#)[👋 Semaine 3](#)[Déroulement d'un programme](#)[Les booléens](#)[Les valeurs truthy et falsy](#)[Notion de scope](#)[Conditions d'existence](#)[Conditions imbriquées](#)[👋 Semaine 4](#)[Objets](#)[Requêtes à API](#)

Qu'est-ce qu'une valeur de retour ?

Fonction avec `console.log()`

Lorsque nous écrivons une fonction en JavaScript, il y a plusieurs façons de la concevoir. Dans certains cas, nous souhaitons simplement afficher un message à l'utilisateur, tandis que dans d'autres cas, nous souhaitons **renvoyer une valeur** à l'endroit où la fonction a été appelée.

Lorsque nous utilisons `console.log()` dans une fonction, cela signifie que nous voulons simplement afficher un message à l'utilisateur :

```
const displayMessage(message) {  
  console.log(message);  
}  
  
displayMessage("Bienvenue !");
```

Avec le code ci-dessus, en appelant la fonction, le message `"Bienvenue !"` s'affichera dans le terminal.

Mais un affichage dans le terminal n'est pas "tangible", dans le sens où cela ne représente pas une valeur en tant que telle !

Pour reprendre l'exemple de notre pizzeria : imaginez qu'il appelle la fonction `makingPizza` pour créer une pizza... Si la fonction fait un `console.log()`, alors c'est comme si la fonction contenait d'afficher la pizza dans le terminal, plutôt que de la créer réellement !

Si vous voulez que l'appel de cette fonction "crée" une pizza, et pouvoir utiliser celle-ci, alors il faudra que la fonction "retourne" une pizza.