# Les fonctions

Les fonctions peuvent être divisées en plusieurs groupes: d'abord les fonctions qui font partie du language ("built-in") et celles qui sont définies par le programmeur. Cette deuxième catégorie ne peut rajouter de possibilités au langage et servira plutôt à **réutiliser le code**.
Une autre façon de classer est selon ce que la fonction renvoie: si une fonction ne renvoie rien on l'appelle une **procédure** sinon c'est une **fonction**. Mais cette première appellation ne s'applique pas vraiment au Javascript car une fonction qui "ne renvoie rien" renverra toujours de façon implicite la valeur **undefined**.

L'utilisation d'une fonction est plutôt simple, on écrit son nom suivi de parenthèses pour l'**appeler** ou l'exécuter et à l'intérieur de celles-ci on place les **arguments** (c'est les informations qu'on va lui donner pour travailler) séparés par des virgules.

=> **console.log('hello', ' ', "world!");**

Pour créer une fonction on utilisera le mot-clef **function** suivi d'un espace et du nom qu'on va donnera à la fonction et directement après des parenthèses qui contiendront les **paramètres** (eux aussi séparés par des virgules) suivi au final d'accolades ("brackets") qui contiendront les instructions à effectuer lors de l'exécution. Les **paramètres** sont les noms de variable sous lesquels seront enregistrés les **arguments** données lors de l'appel ! Les arguments sont les valeurs qui seront fournies lors de l'appel à la fonction (soit son exécution).

```
=> function soustraction(num1, num2) {
        console.log(num1 – num2);
}

soustraction(7, 3); // Affiche 4 dans la console
```

On peut voir ici que l'ordre des arguments est important ! Le 7 a été stocké dans la variable **num1** et le 3 dans **num2** à l'intérieur de la fonction. On pourrait donc dire que:

**Les paramètres sont des variables et les arguments les valeurs assignées (num1 = 7; num2= 3)**

On utilisera donc ces paires de paramètres & arguments chaque fois qu'on aura besoin d'avoir des valeurs disponibles à l'utilisation mais qui peuvent changer à chaque appel à la fonction. Pour les fonctions déjà définies dans le langage les paramètres ne sont pas important, on a juste à fournir les arguments vu qu'on ne crée pas nous-mêmes ces fonctions.

Une autre point important dans la définition d'une fonction est le mot-clef **return** qui permet à la fonction de renvoyer une valeur spécifiée; **return** met un terme à l'execution de la fonction, plus aucune instruction dans son corps (entre les accolades) n'est effectuée après **return**.

```
function soustraction(num1, num2) {
	return num1 - num2;
}

var resultat = soustraction(7, 3); // La variable contiendra 4 car la fonction a renvoyé 4, c'est comme si on remplaçait la fonction par ce qu'elle renvoie
```

Côté paramètres on a plusieurs variantes possibles. On a les paramètres simples comme vu précédemment mais également les paramètres par défaut qui permettent de donner une valeur à cette variable dans la fonction lorsque l'argument est manquant ou qu'une valeur **undefined** est fournie. Il faut simplement faire suivre le paramètre du signe égal ("=") et de la valeur à donner par défaut.

```
function soustraction(num1 = 12, num2 = 5) {
	return num1 - num2;
}

var test1 = soustraction(8); // La variable contient 3 car num1 vaut 8 et num2 vaut 5

var test2 = soustraction(undefined, 10); // Contient 2 car num1 vaut 12 et num2 vaut 10

var test3 = soustraction(); // Contient 7, car aucune valeur fournie donc num1 vaut 12 et num2 vaut 5
```

Une autre variante est le paramètre de reste qui permet de prendre un compte un nombre variable d'arguments et de les enregistrer dans un tableau ("array") . Il se place toujours à la fin de la liste de paramètres et se compose de 3 points ("...") suivi du nom de variable, bien qu'il soit très recommandé d'utiliser le nom usuel **rest**.

```
function multiply(...rest) {
	let resultat = 1;

	for(let i = 0; i < rest.length; i++) {
		resultat *= rest[i];
	}

	return resultat;
}


var test = multiply(2, 4, 6, 8); // La variable contient 384 car 2 * 4 * 6 * 8
```

Il existe également une syntaxe raccourcie qui sert principalement de syntaxe pour les **callbacks** (c'est à dire des fonctions passées en arguments) ou pour définir une fonction en une ligne mais qui peut parfaitement s'utiliser comme telle mais nécessite d'être attribuée à une variable, c'est la **fonction flèchée** ("**Arrow function**").

Elle se compose de parenthèses qui entourent une liste de paramètres séparés par des virgules suivi de "=>" (signe égal et un chevron pointant vers la droite) et des brackets qui contiendront les instructions.

```
(num1, num2) => {
	return num1 + num2;
}
```

Vous aurez remarqué que notre fonction flèchée ne possède par de nom et est donc actuellement inutilisable directement. C'est parce que cette notation nous *renvoie une fonction* par opposition à la notation habituelle où la fonction est crée et enregistrée sous le qu'on lui a donné.
Pour l'utiliser comme une variable normale on peut donc l'enregistrer dans une variable.

```
const somme = (num1, num2) => {
	return num1 + num2;
}

const resultat = somme(3, 5); // resultat contient 8
```

Ainsi on pourra l'utiliser comme toute autre fonction avec son nom suivi de parenthèses et ses arguments si nécessaires pour l'appeler. Un avantage de notre méthode est qu'on peut alors utiliser la portée de variable **const** qui interdit tout futur réassignement à notre variable somme et donc peut nous éviter des erreurs.

Mais il existe aussi d'autres avantages qui font d'une fonction flèchée une manière pratique de raccourcir ses fonctions.
On peut omettre les parenthèses lorsqu'un seul paramètre est présent.

```
const presenter = nom => {
        return `Salut je m'appelle ${nom}`
};

const salutations = presenter("Taliyah Aeriel"); // salutations contient donc maintenant la string "Salut je m'appelle Taliyah Aeriel"
```

Une dernière chose: les brackets ne sont pas nécessaires lorsque qu'on veut simplement que la fonction retourne une valeur comme dans l'exemple précédant et alors un **return implicite** est présent.

```
const presenter = nom => `Salut je m'appelle ${nom}`;

const salutations = presenter("Taliyah Aeriel"); // Contient la même chose que l'exemple précédant
```