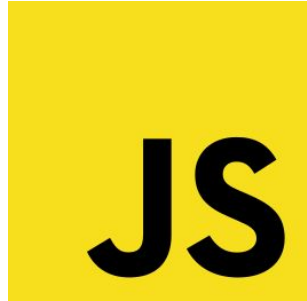


Javascript



JS

Nous allons introduire le (vaste) sujet des fonctions en Javascript.



# JS

Nous allons introduire le (vaste) sujet des fonctions en Javascript.

Voici un des grands principes du développement : **do not repeat yourself**

Nous allons introduire le (vaste) sujet des fonctions en Javascript.

Voici un des grands principes du développement : **do not repeat yourself**



Nous allons introduire le (vaste) sujet des fonctions en Javascript.

Voici un des grands principes du développement : **do not repeat yourself**



Cela signifie que l'on ne doit pas écrire un bout de code qui fait sensiblement la même chose à plusieurs endroits différents.

Nous allons introduire le (vaste) sujet des fonctions en Javascript.

Voici un des grands principes du développement : **do not repeat yourself**



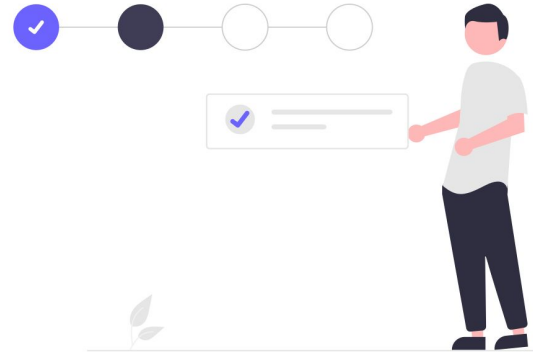
Cela signifie que l'on ne doit pas écrire un bout de code qui fait sensiblement la même chose à plusieurs endroits différents.

Une façon d'éviter de se répéter est de créer des **fonctions**. Ce sont des **blocs de code** que l'on peut **exécuter** autant de fois que l'on veut, sans avoir à réécrire les instructions systématiquement.

## Objectifs

1. **Écrire et exécuter** des fonctions en JS
2. Éviter de se répéter (**D.R.Y**)

# Écrire et exécuter des fonctions





## Créer une fonction

Pour **déclarer une fonction**, on va utiliser le mot clé **function** suivi du nom de la fonction.

## Créer une fonction

Pour **déclarer une fonction**, on va utiliser le mot clé **function** suivi du nom de la fonction.

```
function helloWorld() {  
  console.log("Hello,");  
  console.log("World!");  
}
```

## Créer une fonction

Pour **déclarer une fonction**, on va utiliser le mot clé **function** suivi du nom de la fonction.

```
function helloWorld() {  
    console.log("Hello,");  
    console.log("World!");  
}
```

Pour **appeler** (exécuter/invoquer) la fonction, il faut simplement écrire son nom **suivi par des parenthèses** :

## Créer une fonction

Pour **déclarer une fonction**, on va utiliser le mot clé **function** suivi du nom de la fonction.

```
function helloWorld() {  
    console.log("Hello,");  
    console.log("World!");  
}
```

Pour **appeler** (exécuter/invoquer) la fonction, il faut simplement écrire son nom **suivi par des parenthèses** :

```
helloWorld();
```

## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

Les paramètres sont des **données** que la fonction va prendre en entrée. Ces données seront donc disponibles **dans le contexte de la fonction** afin que cette dernière puisse les manipuler.

Voici un exemple :

## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

Les paramètres sont des **données** que la fonction va prendre en entrée. Ces données seront donc disponibles **dans le contexte de la fonction** afin que cette dernière puisse les manipuler.

Voici un exemple :



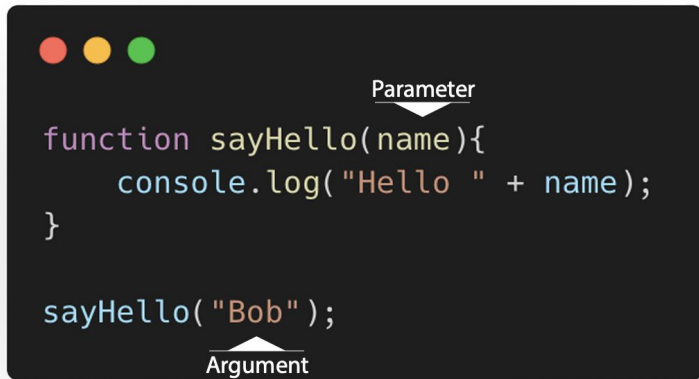
```
function sayHello(name){  
    console.log("Hello " + name);  
}  
  
sayHello("Bob");
```

## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

Les paramètres sont des **données** que la fonction va prendre en entrée. Ces données seront donc disponibles **dans le contexte de la fonction** afin que cette dernière puisse les manipuler.

Voici un exemple :



```
function sayHello(name){  
    console.log("Hello " + name);  
}  
  
sayHello("Bob");
```

The code is displayed in a dark-themed editor with three colored window control buttons (red, yellow, green) in the top-left corner. Annotations are present: 'Parameter' with a downward arrow pointing to the `name` parameter in the function signature, and 'Argument' with an upward arrow pointing to the `"Bob"` value in the function call.

Dans cet exemple, on a créé une **fonction** nommée **sayHello** qui accepte une donnée nommée **name** en **paramètre**.

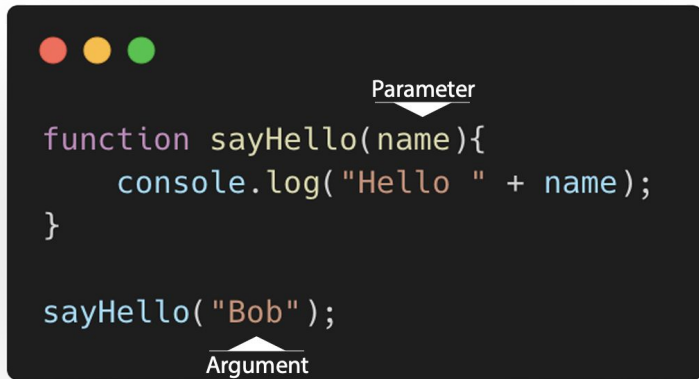


## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

Les paramètres sont des **données** que la fonction va prendre en entrée. Ces données seront donc disponibles **dans le contexte de la fonction** afin que cette dernière puisse les manipuler.

Voici un exemple :



```
function sayHello(name){  
  console.log("Hello " + name);  
}  
  
sayHello("Bob");
```

The code is displayed in a dark-themed editor with syntax highlighting. A label 'Parameter' with a downward arrow points to the 'name' parameter in the function signature. A label 'Argument' with an upward arrow points to the 'Bob' argument in the function call.

Dans cet exemple, on a créé une **fonction** nommée **sayHello** qui accepte une donnée nommée **name** en **paramètre**.

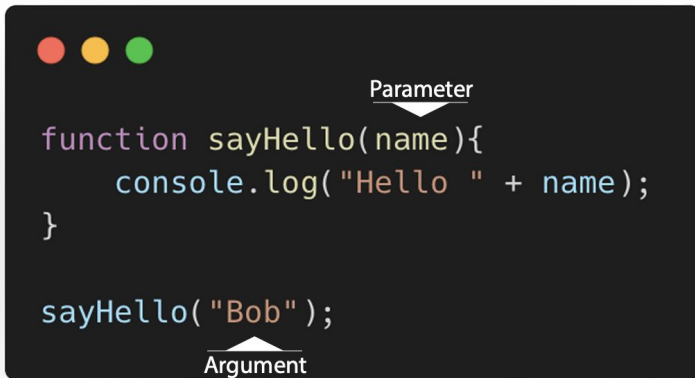
Ce **paramètre se verra assigné la valeur de l'argument** donné entre les parenthèses de la fonction lors de son appel.

## Paramètres/Arguments d'une fonction

Une fonction peut accepter un ou plusieurs **paramètres**.

Les paramètres sont des **données** que la fonction va prendre en entrée. Ces données seront donc disponibles **dans le contexte de la fonction** afin que cette dernière puisse les manipuler.

Voici un exemple :



```
function sayHello(name){  
    console.log("Hello " + name);  
}  
  
sayHello("Bob");
```

The code is displayed in a dark-themed editor. Above the parameter `name` in the function signature, there is a label `Parameter` with a downward-pointing arrow. Below the argument `"Bob"` in the function call, there is a label `Argument` with an upward-pointing arrow.

Dans cet exemple, on a créé une **fonction** nommée **sayHello** qui accepte une donnée nommée **name** en **paramètre**.

Ce **paramètre se verra assigné la valeur de l'argument** donné entre les parenthèses de la fonction lors de son appel.

On peut appeler le paramètre comme on le souhaite : on l'a appelé **name**, mais on aurait pu donner le nom **firstName** à la place (ou quelque chose d'autre, tant que c'est explicite).

## Paramètres/Arguments d'une fonction

Un exemple avec plusieurs **paramètres** :

```
function sayHello(firstName, lastName) {  
  console.log("Hello " + firstName + " " + lastName);  
}  
  
sayHello("Mickey", "Mouse");
```

Hello Mickey Mouse

## La concaténation

Le fait d'accoler plusieurs chaînes les unes à la suite des autres (dans notre exemple "Hello" + "Mickey" + " " + "Mouse") est appelé **concaténation**.

## La concaténation

Le fait d'accoler plusieurs chaînes les unes à la suite des autres (dans notre exemple "Hello" + "Mickey" + " " + "Mouse") est appelé **concaténation**.

```
console.log("Hello " + name);
```

## La concaténation

Le fait d'accoler plusieurs chaînes les unes à la suite des autres (dans notre exemple "Hello" + "Mickey" + " " + "Mouse") est appelé **concaténation**.

```
console.log("Hello " + name);
```

Avec cette méthode, on peut donc prendre plusieurs chaînes de caractères (autant que l'on souhaite) pour n'en faire qu'une :

## La concaténation

Le fait d'accoler plusieurs chaînes les unes à la suite des autres (dans notre exemple "Hello" + "Mickey" + " " + "Mouse") est appelé **concaténation**.

```
console.log("Hello " + name);
```

Avec cette méthode, on peut donc prendre plusieurs chaînes de caractères (autant que l'on souhaite) pour n'en faire qu'une :

```
console.log("Hello" + "I'm" + "Bob");
```

## L'interpolation de chaînes

Pour assembler des chaînes ensemble, il y a une autre méthode un peu plus récente : **l'interpolation**.



## L'interpolation de chaînes

Pour assembler des chaînes ensemble, il y a une autre méthode un peu plus récente : **l'interpolation**.

Comment ça marche ? Il suffit d'écrire la chaîne avec des "backticks" (``) à la place des guillemets simples ou doubles. Dès que l'on souhaite insérer une **expression** (par exemple une variable) dans la chaîne, on va utiliser le signe dollar (\$) suivi d'accolades ({} à l'intérieur desquelles on va pouvoir placer notre expression.

Voici un exemple concret :

## L'interpolation de chaînes

Pour assembler des chaînes ensemble, il y a une autre méthode un peu plus récente : **l'interpolation**.

Comment ça marche ? Il suffit d'écrire la chaîne avec des "backticks" (``) à la place des guillemets simples ou doubles. Dès que l'on souhaite insérer une **expression** (par exemple une variable) dans la chaîne, on va utiliser le signe dollar (\$) suivi d'accolades ({}), à l'intérieur desquelles on va pouvoir placer notre expression.

Voici un exemple concret :

```
const name = "Donald Duck";  
console.log(`Hello, I'm ${name}`);  
console.log(`You can write Js expressions: ${1 + 1}`);  
console.log(`You can write Js expressions: ${1 < 1}`);
```

```
Hello, I'm Donald  
Duck  
You can write Js  
expressions: 2  
You can write Js  
expressions: false
```

## Return

On vient de voir que les paramètres d'une fonction représentent **ses entrées**. Mais une fonction peut également produire **une sortie** qu'on appelle **valeur de retour**.

## Return

On vient de voir que les paramètres d'une fonction représentent **ses entrées**. Mais une fonction peut également produire **une sortie** qu'on appelle **valeur de retour**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

On vient de voir que les paramètres d'une fonction représentent **ses entrées**. Mais une fonction peut également produire **une sortie** qu'on appelle **valeur de retour**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

Dans ce code, on a créé une fonction pour calculer la somme de deux nombres.

## Return

On vient de voir que les paramètres d'une fonction représentent **ses entrées**. Mais une fonction peut également produire **une sortie** qu'on appelle **valeur de retour**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

Dans ce code, on a créé une fonction pour calculer la somme de deux nombres.

Cette dernière accepte en paramètres les nombres à additionner : **a** et **b**.

## Return

On vient de voir que les paramètres d'une fonction représentent **ses entrées**. Mais une fonction peut également produire **une sortie** qu'on appelle **valeur de retour**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

Dans ce code, on a créé une fonction pour calculer la somme de deux nombres.

Cette dernière accepte en paramètres les nombres à additionner : **a** et **b**.

La somme (**a + b**) est **renvoyée à l'endroit où on a appelé la fonction** grâce au mot-clé **return**.

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```



## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.
- Le code de la fonction est exécuté et **renvoie la valeur 3**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.
- Le code de la fonction est exécuté et **renvoie la valeur 3**.
- A ce moment là, tout se passe donc comme si on avait **console.log(3)** au lieu de **console.log(sum(1, 2))**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.
- Le code de la fonction est exécuté et **renvoie la valeur 3**.
- A ce moment là, tout se passe donc comme si on avait **console.log(3)** au lieu de **console.log(sum(1, 2))**.

Il n'est pas obligatoire de toujours spécifier une valeur de retour.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.
- Le code de la fonction est exécuté et **renvoie la valeur 3**.
- A ce moment là, tout se passe donc comme si on avait **console.log(3)** au lieu de **console.log(sum(1, 2))**.

Il n'est pas obligatoire de toujours spécifier une valeur de retour.

**Si rien n'est spécifié, la fonction renverra `undefined` par défaut.**

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

## Return

Maintenant, que se passe-t-il exactement quand on exécute la fonction ?

- On appelle la fonction **sum** avec les **arguments 1 et 2**.
- Le code de la fonction est exécuté et **renvoie la valeur 3**.
- A ce moment là, tout se passe donc comme si on avait **console.log(3)** au lieu de **console.log(sum(1, 2))**.

```
function sum(a, b){  
  return a + b;  
}  
console.log(sum(1, 2));
```

Il n'est pas obligatoire de toujours spécifier une valeur de retour.

Si rien n'est spécifié, la fonction renverra **undefined** par défaut.

**⚠** L'utilisation du mot-clé "return" stoppe immédiatement l'exécution de la fonction. Les lignes de code après un "return" ne seront donc jamais prises en compte.

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :



## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :

- **name** (l'identifiant de l'utilisateur)

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :

- **name** (l'identifiant de l'utilisateur)
- **password** (son mot de passe).

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :

- **name** (l'identifiant de l'utilisateur)
- **password** (son mot de passe).

Cette dernière va retourner **true** ou bien **false** en fonction de la validité des informations de connexion fournies en paramètres.

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :

- **name** (l'identifiant de l'utilisateur)
- **password** (son mot de passe).

Cette dernière va retourner **true** ou bien **false** en fonction de la validité des informations de connexion fournies en paramètres.

Une fois la fonction définie, on crée 2 variables **userName** et **userPassword** qui seront initialisées avec l'entrée de l'utilisateur, via un prompt

## Return

```
function login(name, password) {  
  if(name === "Bob" && password === "secret") {  
    return true;  
  }  
  else {  
    return false  
  }  
}  
  
let userName = prompt("What's your name?");  
let userPassword = prompt("What's your password?");  
  
if(login(userName, userPassword)) {  
  console.log("Welcome!");  
}  
else {  
  console.log("Wrong credentials...");  
}
```

Dans ce code, on crée une fonction **login** qui accepte deux paramètres :

- **name** (l'identifiant de l'utilisateur)
- **password** (son mot de passe).

Cette dernière va retourner **true** ou bien **false** en fonction de la validité des informations de connexion fournies en paramètres.

Une fois la fonction définie, on crée 2 variables **userName** et **userPassword** qui seront initialisées avec l'entrée de l'utilisateur, via un prompt

On vérifie ensuite si la valeur de retour de la fonction login appelée avec les arguments **userName** et **userPassword** est **true**. Si c'est le cas, on affiche "Welcome !", sinon on affiche "Wrong credentials...".

## Portée (scope) - Contexte

En Javascript, dès que l'on écrit du code, **le contexte est très important** : **on ne peut pas utiliser une variable déclarée à l'intérieur d'une fonction en dehors de cette dernière.**

## Portée (scope) - Contexte

En Javascript, dès que l'on écrit du code, **le contexte est très important** : on ne peut pas utiliser une variable déclarée à l'intérieur d'une fonction en dehors de cette dernière.

```
function sayMyName() {  
  let name = "Pierre";  
  console.log(name);  
  /// fonctionne bien dans le contexte de la fonction  
}  
  
function sayMyFullName() {  
  let lastName = "Gerard";  
  console.log(lastName + ' ' + name);  
  // ne fonctionne pas  
}
```

## Portée (scope) - Contexte

En Javascript, dès que l'on écrit du code, **le contexte est très important** : on ne peut pas utiliser une variable déclarée à l'intérieur d'une fonction en dehors de cette dernière.

```
function sayMyName() {  
  let name = "Pierre";  
  console.log(name);  
  /// fonctionne bien dans le contexte de la fonction  
}  
  
function sayMyFullName() {  
  let lastName = "Gerard";  
  console.log(lastName + ' ' + name);  
  // ne fonctionne pas  
}
```

Par exemple, dans ce cas, la variable **name** sera disponible uniquement dans le contexte de **sayMyName**, pas dans le contexte global.



## Portée (scope) - Contexte

En Javascript, dès que l'on écrit du code, **le contexte est très important** : on ne peut pas utiliser une variable déclarée à l'intérieur d'une fonction en dehors de cette dernière.

```
function sayMyName() {  
  let name = "Pierre";  
  console.log(name);  
  /// fonctionne bien dans le contexte de la fonction  
}  
  
function sayMyFullName() {  
  let lastName = "Gerard";  
  console.log(lastName + ' ' + name);  
  // ne fonctionne pas  
}
```

Par exemple, dans ce cas, la variable **name** sera disponible uniquement dans le contexte de **sayMyName**, pas dans le contexte global.

**La fonction s'exécutera donc dans son propre contexte et aura son propre espace mémoire.**

## Portée (scope) - Contexte

En Javascript, dès que l'on écrit du code, **le contexte est très important : on ne peut pas utiliser une variable déclarée à l'intérieur d'une fonction en dehors de cette dernière.**

```
function sayMyName() {  
  let name = "Pierre";  
  console.log(name);  
  /// fonctionne bien dans le contexte de la fonction  
}  
  
function sayMyFullName() {  
  let lastName = "Gerard";  
  console.log(lastName + ' ' + name);  
  // ne fonctionne pas  
}
```

Par exemple, dans ce cas, la variable **name** sera disponible uniquement dans le contexte de **sayMyName**, pas dans le contexte global.

**La fonction s'exécutera donc dans son propre contexte et aura son propre espace mémoire.**

>> [Vidéo récap sur les fonctions](#)

Pratiquons !

