> Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

Valeurs de retour des fonctions

Il y a un dernier concept essentiel sur les fonctions dont nous devons discuter : **les valeurs de retour**. Certaines fonctions ne renvoient pas de valeur significative, mais d'autres le font. Il est important de comprendre quelles sont leurs valeurs, comment les utiliser dans votre code et comment faire en sorte que les fonctions renvoient des valeurs utiles. Nous allons aborder ces différents sujets dans cet article.

Prérequis :	Notions de base en informatique, compréhension élémentaire de HTML et CSS, avoir lu Premiers pas en JavaScript, et Fonctions — blocs de code réutilisable.
Objectif :	Comprendre les valeurs de retour, et comment les utiliser.

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

Les **valeurs de retour** sont, comme leur nom l'indique, les valeurs retournées par une fonction après son exécution. Vous en avez déjà rencontré plusieurs fois sans forcément y avoir pensé explicitement.

Revenons à un exemple déjà vu (tiré d'un article précédent de cette série) :

JS

```
const monTexte = "Il fait froid";
const nouveauTexte = monTexte.replace("froid", "chaud");
console.log(nouveauTexte); // Devrait afficher "Il fait chaud"
```

```
// la fonction replace() s'applique aux chaînes de caractères
// et remplace une sous-chaîne par une autre puis retourne
// une nouvelle chaîne avec le remplacement effectué
```

La fonction `replace()` est invoquée sur la chaîne de caractères `monTexte`, et nous lui passons deux paramètres :

- La chaîne à trouver (`"froid"`)
- La chaîne de remplacement (`"chaud"`)

Lorsque cette fonction a fini de s'exécuter, elle retourne une valeur qui est une nouvelle chaîne avec le remplacement effectué. Dans le code ci-dessus, nous sauvegardons cette valeur avec la variable `nouveauTexte`.

Si vous regardez la page de référence MDN sur la fonction `replace()`, vous verrez une section intitulée Valeur de retour. Il est utile de savoir et de comprendre quelles sont les valeurs retournées par les fonctions et c'est pourquoi cette information est présente sur les différentes pages de référence des fonctions JavaScript sur MDN.

Certaines fonctions ne retournent pas de valeur (pour ces cas, nos pages de référence indiquent `void` ou `undefined` comme valeur de retour). Par exemple, dans la fonction `displayMessage()` construite dans l'article précédent, aucune valeur spécifique n'est retournée comme résultat de la fonction appelée. Il y a seulement une boîte qui apparaît, et c'est tout !

Généralement, une valeur de retour est utilisée lorsque la fonction est une étape dans un programme. Ces valeurs intermédiaires doivent être d'abord évaluées par une fonction, le résultat renvoyé pourra être ensuite utilisé dans l'étape suivante du programme et ainsi de suite jusqu'à obtenir la valeur finale désirée.

Utiliser des valeurs de retour dans vos fonctions

Pour retourner une valeur d'une fonction que vous avez créée, vous devez utiliser le mot-clef `return`. Nous avons vu son utilisation dans l'exemple `random-canvas-circles.html`. La fonction `draw()` dessine 100 cercles aléatoires dans un élément HTML `<canvas>` :

JS

```
function draw() {
  ctx.clearRect(0, 0, WIDTH, HEIGHT);
  for (let i = 0; i < 100; i++) {
    ctx.beginPath();
    ctx.fillStyle = "rgba(255,0,0,0.5)";
    ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
    ctx.fill();
  }
}
```

À chaque itération de la boucle, on fait trois fois appel à la fonction `random()` pour générer respectivement une valeur aléatoire pour les *coordonnées x et y* du cercle, ainsi que pour son *rayon*. La fonction `random()` prend un seul paramètre, un nombre entier, et elle retourne un nombre entier aléatoire compris entre 0 et ce nombre. Voici à quoi elle ressemble :

JS

```
function random(nombre) {
  return Math.floor(Math.random() * nombre);
}
```

Cela peut aussi s'écrire ainsi :

JS

```
function random(nombre) {
  const resultat = Math.floor(Math.random() * nombre);
  return resultat;
}
```

Mais la première version est plus rapide à écrire, et plus compacte.

La fonction retourne le résultat de `Math.floor(Math.random() * nombre)` chaque fois qu'elle est appelée. Cette valeur de retour apparaît à l'endroit où la fonction a été appelée, puis le code continue. Si, par exemple, nous exécutons la ligne suivante :

JS

```
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
```

Si les trois appels `random()` retournent respectivement les valeurs 500, 200 et 35, la fonction `arc()` s'exécutera de cette façon :

JS

```
ctx.arc(500, 200, 35, 0, 2 * Math.PI);
```

Les appels de fonctions qui sont imbriqués dans d'autres appels sont exécutés en premier, puis leur valeur de retour est substitué pour que la fonction englobante puisse être exécutée et ainsi de suite.

Apprentissage actif : créer notre propre fonction renvoyant une valeur

Écrivons nos propres fonctions avec des valeurs de retour.

1. Pour commencer, faites une copie locale du fichier `function-library.html` à partir de GitHub. Il s'agit d'une simple page HTML contenant un champ texte `<input>` et un paragraphe. Il y a également un élément `<script>` qui référence ces éléments HTML dans deux variables. Cette page vous permettra d'entrer un nombre dans le champ texte, et affichera différents nombres en lien avec celui saisi dans l'espace en dessous.
2. Ajoutons quelques fonctions dans `<script>`. Sous les deux lignes existantes de JavaScript, ajoutez les définitions des fonctions suivantes :

   JS

   ```
   function carre(nombre) {
     return nombre * nombre;
   }

   function cube(nombre) {
     return nombre * nombre * nombre;
   }

   function factorielle(nombre) {
     if (nombre < 0) return undefined;
     if (nombre === 0) return 1;
     let x = nombre - 1;
     while (x > 1) {
       nombre *= x;
       x--;
   ```

```
  }
  return nombre;
}
```

Les fonctions `carre()` et `cube()` sont plutôt évidentes, elles retournent le carré et le cube du nombre donné en paramètre. La fonction `factorielle()` retourne la factorielle du nombre donné.

3. Ensuite, nous allons ajouter un moyen d'afficher des informations relatives au nombre saisi dans le champ texte. Ajoutez le gestionnaire d'évènement suivant à la suite des fonctions :

   JS

```
input.addEventListener("change", () => {
  const nombre = parseFloat(input.value);
  if (isNaN(nombre)) {
    para.textContent = "Vous devez saisir un nombre.";
  } else {
    para.textContent = `Le carré de ${nombre} vaut ${carre(nombre)}. `;
    para.textContent += `Le cube de ${nombre} vaut ${cube(nombre)}. `;
    para.textContent += `La factorielle de ${nombre} vaut ${factorielle(
      nombre,
    )}. `;
  }
});
```

4. Sauvegardez votre code, chargez-le dans votre navigateur et testez-le.

Voici quelques explications sur la fonction `addEventListener()` qui a été ajoutée à l'étape 3 :

- En ajoutant un gestionnaire d'évènement pour `change`, on exécute la fonction correspondante chaque fois que l'évènement `change` se déclenche sur le champ de saisie (c'est-à-dire lorsqu'une nouvelle valeur est saisie dans `input` et soumise (par exemple, entrez une valeur, puis désélectionnez le champ en appuyant sur `Tab` ou `Entrée`)). Lorsque cette fonction anonyme s'exécute, la valeur saisie dans le champ (`input.value`) est stockée dans la constante `nombre`.
- L'instruction `if` affiche un message d'erreur si la valeur saisie n'est pas un nombre. La condition vérifie si l'expression `isNaN(nombre)` retourne `true`. La fonction `isNaN()`

teste si la valeur `nombre` n'est pas un nombre : si c'est le cas, elle retourne `true`, et sinon, elle retourne `false`.

- Si la condition retourne `false`, la valeur `nombre` est un nombre et la fonction affiche une phrase à l'intérieur du paragraphe qui indique le carré de la valeur, son cube et sa factorielle. Pour cela, la fonction appelle les fonctions `carre()`, `cube()`, et `factorielle()` pour calculer les valeurs requises.

> **Note :** Si vous rencontrez des difficultés pour faire fonctionner cet exemple, vous pouvez vérifier le code en le comparant à la version finale (en anglais) sur GitHub (vous pouvez également voir la démo en anglais).

À votre tour !

À ce stade, c'est à vous de vous lancer et d'écrire vos propres fonctions et de les ajouter. Pouvez-vous ajouter une fonction qui calcule la racine carrée ou la racine cubique du nombre, une fonction qui calcule la circonférence d'un cercle avec ce rayon ?

Voici quelques conseils supplémentaires à propos des fonctions :

- Profitez-en pour ajouter *la gestion des erreurs* dans vos fonctions. C'est généralement une bonne idée de vérifier que tous les paramètres nécessaires sont valides et que tous les paramètres facultatifs ont une valeur par défaut fournie. De cette façon, votre programme sera moins susceptible de générer des erreurs.
- Réfléchissez à la création d'une *bibliothèque de fonctions*. Au fur et à mesure que vous avancerez dans votre parcours de programmeuse ou programmeur, vous commencerez à réaliser des fonctions similaires, encore et encore. C'est une bonne idée que de créer votre propre bibliothèque de fonctions utilitaires pour faire ce genre de choses. Vous pourrez ainsi les réutiliser dans du nouveau code.

Évaluez vos compétences !

Vous voici à la fin de cet article. Avez-vous bien retenu les informations importantes ? Pour le vérifier avant d'aller plus loin, vous pouvez vous évaluer sur les fonctions JavaScript (en-US).

sur leur syntaxe et leurs fonctionnalités.

Si vous n'avez pas compris quelque chose, n'hésitez pas à relire l'article, ou contactez-nous pour obtenir de l'aide.

Voir aussi

Le guide sur les fonctions de la référence JavaScript

Un guide détaillé couvrant des informations plus avancées sur les fonctions.

La page du glossaire sur les fonctions de rappel (*callbacks* en anglais)

Il arrive souvent en JavaScript de passer une fonction à une autre fonction *comme argument*. Cet argument est alors appelé au sein de la deuxième fonction. Ce concept va au-delà de ce premier guide, mais n'hésitez pas à vous familiariser avec cette notion.

This page was last modified on 26 oct. 2023 by MDN contributors.