> 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.

# Construire vos propres fonctions

Dans l'article précédent, nous avons traité essentiellement de la théorie. Le présent article fournira une expérience pratique. Ici vous allez mettre en pratique ces connaissances en construisant vos propres fonctions. Tout au long, nous expliquerons également quelques détails supplémentaires concernant les fonctions.

| | |
|---|---|
| **Prérequis :** | Savoir-faire de base, une compréhension minimale HTML et CSS, premiers pas en JavaScript, Fonctions — blocs de code réutilisables. |
| **Objectif :** | Fournir quelques pratiques de création de fonctions, et expliquer un peu plus les détails associés. |

## Apprentissage actif : Construisons une fonction

La fonction que nous allons construire sera nommée `displayMessage()`. Elle affichera une boîte de message personnalisée sur une page web. Elle fonctionnera comme un substitut personnalisé de la fonction alert() du navigateur. Vous avez déjà vu cela avant, mais nous allons simplement nous rafraîchir la mémoire — essayez le code qui suit dans la console JavaScript de votre navigateur, sur n'importe quelle page que vous aimez :

JS

```
alert("This is a message");
```

La fonction prend un seul argument en paramètre — la chaîne de caractères qui est affichée dans la boîte d'alerte. Vous pouvez essayer de varier la syntaxe de la chaîne pour modifier le message.

La fonction alert() est assez limitée : vous pouvez modifier le message, mais vous ne pouvez pas facilement faire varier autre chose, comme la couleur, une icône, ou autre chose. Nous en construirons une qui s'avérera plus amusante.

> **Note :** Cet exemple devrait fonctionner correctement dans tous les navigateurs modernes, mais elle pourrait avoir un comportement un peu plus inattendu dans un navigateur ancien. Nous recommandons donc de faire cet exercice dans un navigateur moderne tel que Firefox, Opera, ou Chrome.

## La fonction de base

Pour commencer, mettons en place une fonction de base.

> **Note :** Pour les conventions de nommage des fonctions, vous devez suivre les mêmes règles que les conventions de noms de variables. Ce qui est bien, c'est que vous pouvez les différencier — les noms de fonctions se terminent par des parenthèses, pas les variables.

1. Commencez par faire une copie locale du fichier function-start.html . Vous pourrez voir que le code HTML est simple — l'élément body ne contient qu'un seul bouton. Nous avons également ajouté quelques règles CSS de base pour styliser la boîte de message personnalisée, et un élément `<script>` pour écrire notre code JavaScript.
2. Ensuite, ajoutez le code ci-dessous à l'intérieur de l'élément `<script>` :

   JS

   ```
   function displayMessage() {}
   ```

   Nous commençons avec le mot-clé `function`, qui signifie que nous définissons une fonction. Celui-ci est suivi par le nom que nous voulons donner à notre fonction, des parenthèses et des accolades. Tous les paramètres que nous voulons donner à notre

fonction vont à l'intérieur des parenthèses, et le code qui s'exécute lorsque nous appelons la fonction va à l'intérieur des accolades.

3. Enfin, ajoutez le code suivant à l'intérieur des accolades :

```
JS
var html = document.querySelector("html");

var panel = document.createElement("div");
panel.setAttribute("class", "msgBox");
html.appendChild(panel);

var msg = document.createElement("p");
msg.textContent = "This is a message box";
panel.appendChild(msg);

var closeBtn = document.createElement("button");
closeBtn.textContent = "x";
panel.appendChild(closeBtn);

closeBtn.onclick = function () {
  panel.parentNode.removeChild(panel);
};
```

Étant donné qu'il y a pas mal de code à analyser, allons-y pas à pas.

La première ligne utilise une fonction de l'API DOM appelée `document.querySelector()` pour sélectionner l'élément `<html>` et stocker une référence vers cet élément dans une variable appelée `html`, de façon à pouvoir l'utiliser plus tard :

```
JS
var html = document.querySelector("html");
```

La section suivante utilise une autre fonction de l'API DOM appelée `Document.createElement()` pour créer un élément `<div>` et stocker une référence vers lui dans une variable appelée `panel` (Dans la suite de l'article, nous parlerons simplement du panneau `<div>`.). Cet élément sera le conteneur extérieur de notre boîte de message.

Puis, nous utilisons encore une autre fonction de l'API DOM appelée `Element.setAttribute()` pour ajouter un attribut `class` à notre panneau qui aura pour valeur `msgBox`. Ceci rendra plus facile la mise en forme de l'élément — si vous regardez le

CSS de la page, vous verrez que nous utilisons un sélecteur de classe `.msgBox` dans le but de styliser la boîte de message ainsi que son contenu.

Finallement, nous appelons une fonction du DOM nommée `Node.appendChild()` sur la variable `html` créée précédemment, qui insère un élément, en tant qu'enfant, à l'intérieur d'un autre. Nous spécifions le panneau `<div>` (panel) comme l'enfant que nous voulons ajouter à l'intérieur de l'élément `<html>`. Nous avons besoin de le faire puisque l'élément que nous avons créé ne peut pas apparaître de lui-même sur la page — nous avons besoin de préciser où le mettre.

JS

```
var panel = document.createElement("div");
panel.setAttribute("class", "msgBox");
html.appendChild(panel);
```

Les deux sections suivantes font usage des mêmes fonctions `createElement()` et `appendChild()` que nous avons déjà vu pour créer deux nouveaux éléments — l'un `<p>` et l'autre `<button>` — et pour les insèrer dans la page en tant qu'enfant du panneau `<div>`. On utilise leur propriété `Node.textContent` — qui représente le contenu textuel d'un élément — pour insérer un message à l'intérieur du paragraphe, ainsi qu'un 'x' à l'intérieur du bouton. Ce bouton sera cliqué / activé quand l'utilisateur voudra fermer la boîte de message.

JS

```
var msg = document.createElement("p");
msg.textContent = "This is a message box";
panel.appendChild(msg);

var closeBtn = document.createElement("button");
closeBtn.textContent = "x";
panel.appendChild(closeBtn);
```

Finalement, nous utilisons un gestionnaire d'évènements `GlobalEventHandlers.onclick` de sorte qu'un clic sur le bouton déclenche le bout de code chargé de supprimer la totalité du panneau de la page — c'est-à-dire fermer la boîte de message.

Le gestionnaire `onclick` est une propriété disponible sur le bouton (en fait, sur n'importe quel élément de la page) qui pourra se voir transmettre une fonction en paramètre pour spécifier quel morceau de code sera déclenché quand le bouton sera cliqué. Vous en apprendrez bien plus dans notre article sur les évènements. Nous avons passé à notre gestionnaire `onclick` une fonction anonyme, qui contient le code exécuté quand le bouton est cliqué. L'instruction définie dans la fonction utilise la fonction de l'API DOM `Node.removeChild()` pour indiquer que nous tenons à supprimer un élément enfant spécifique de l'élément HTML — dans notre cas le panneau `<div>`.

JS

```
closeBtn.onclick = function () {
  panel.parentNode.removeChild(panel);
};
```

Au final, l'intégralité du bloc de code génère un bloc de code HTML et l'insère dans la page, ce qui ressemble à ça :

HTML

```
<div class="msgBox">
  <p>This is a message box</p>
  <button>x</button>
</div>
```

Ça nous a fait beaucoup de code à passer en revue — ne vous inquiétez pas trop si vous ne vous souvenez pas exactement de comment chaque instruction fonctionne ! Bien que la partie principale sur laquelle nous voulions mettre l'accent ici est la structure de la fonction et son utilisation, nous avons voulu montrer quelque chose d'intéressant pour mettre en valeur cet exemple.

## Appeler la fonction

À présent, nous avons notre fonction définie comme il faut dans notre balise `<script>`, mais il ne se passera rien si on laisse les choses en l'état.

1. Ajoutez la ligne suivante au-dessous de votre fonction pour l'appeler :

   JS

   ```
   displayMessage();
   ```

Cette ligne appelle la fonction en la faisant fonctionner immédiatement. Lorsque vous enregistrez votre code et rechargez la page dans le navigateur, vous voyez la petite boîte de message apparaître immédiatement, une seule fois. Après tout, nous ne l'appelons bien qu'une fois.

2. Maintenant, ouvrez vos outils de développement sur la page d'exemple, allez à la console JavaScript et tapez-y la ligne à nouveau, vous verrez qu'elle apparaît encore une fois ! C'est génial, nous avons maintenant une fonction réutilisable que nous pouvons appeler chaque fois que nous le voulons. Cela dit, nous voulons probablement qu'elle apparaisse en réponse aux actions de l'utilisateur ou du système. Dans une application réelle, une telle boîte de message serait probablement appelée en réponse à de nouvelles données disponibles, si une erreur s'est produite, si l'utilisateur essaie de supprimer son profil ("Êtes vous sûr de vouloir réaliser cette action ?"), ou encore si l'utilisateur ajoute un nouveau contact et que l'opération se termine avec succès, etc. Dans cette démo, nous faisons apparaître le message quand l'utilisateur clique sur le bouton.
3. Supprimez la ligne précédente que vous avez ajoutée.
4. Ensuite, vous sélectionnerez le bouton et stockerez une référence vers celui-ci dans une variable. Ajoutez la ligne suivante à votre code, au-dessus de la définition de fonction :

   JS

   ```
   var btn = document.querySelector("button");
   ```

5. Enfin, ajoutez la ligne suivante à la précédente :

   JS

   ```
   btn.onclick = displayMessage;
   ```

   D'une manière similaire à notre ligne `closeBtn.onclick...` à l'intérieur de la fonction, ici, nous appelons un certain code en réponse à un clic sur un bouton. Mais dans ce cas, au lieu d'appeler une fonction anonyme contenant du code, nous appelons directement notre nom de fonction.
6. Essayez d'enregistrer et de rafraîchir la page, maintenant vous devriez voir la boîte de message s'afficher lorsque vous cliquez sur le bouton.

Vous vous demandez peut-être pourquoi nous n'avons pas inclus les parenthèses après le nom de la fonction. C'est parce que nous ne voulons pas appeler la fonction

immédiatement, seulement après que le bouton aura été cliqué. Si vous modifiez la ligne pour :

JS

```
btn.onclick = displayMessage();
```

Enregistrez et rafraîchissez la page, vous verrez que la boîte de message apparaît sans que le bouton ait été cliqué ! Dans ce contexte, les parenthèses sont parfois appelées "opérateur d'appel / invocation de fonction". Vous ne les utilisez que lorsque vous souhaitez exécuter la fonction immédiatement dans la portée actuelle. Dans le même ordre d'idée, le code à l'intérieur de la fonction anonyme n'est pas exécuté immédiatement, car il se trouve à l'intérieur de la portée de la fonction.

Si vous avez essayé la dernière expérimentation, assurez-vous d'annuler la dernière modification avant de poursuivre.

## Améliorer la fonction à l'aide de paramètres

En l'état, la fonction n'est pas très utile — on ne veut pas montrer le même message par défaut à chaque fois. Améliorons la en ajoutant quelques paramètres, ils permettront d'appeler la fonction avec différentes options.

1. Premièrement, mettons à jour la première ligne :

   JS

   ```
   function displayMessage() {
   ```

   par :

   JS

   ```
   function displayMessage(msgText, msgType) {
   ```

   Maintenant, quand nous appelons la fonction, nous pouvons fournir deux valeurs de variables entre les parenthèses : une pour spécifier le message à afficher dans la boîte, l'autre pour le type de message.

2. Pour faire usage du premier paramètre, mettez à jour la ligne suivante à l'intérieur de votre fonction :

   JS

```
msg.textContent = "This is a message box";
```

avec :

JS

```
msg.textContent = msgText;
```

3. Vous devez maintenant mettre à jour votre appel de fonction pour inclure un texte de message mis à jour. Modifiez la ligne suivante :

   JS

   ```
   btn.onclick = displayMessage;
   ```

   par ce bloc :

   JS

   ```
   btn.onclick = function () {
     displayMessage("Woo, this is a different message!");
   };
   ```

   Si nous voulons spécifier des paramètres à l'intérieur des parenthèses pour la fonction que nous appelons, alors nous ne pouvons pas l'appeler directement — nous avons besoin de la mettre à l'intérieur d'une fonction anonyme de sorte qu'elle n'est pas dans la portée immédiate et n'est donc pas appelée immédiatement. Maintenant, elle ne sera pas appelée tant que le bouton ne sera pas cliqué.

4. Rechargez et essayez le code à nouveau et vous verrez qu'il fonctionne toujours très bien, sauf que maintenant vous pouvez également modifier le message à l'intérieur du paramètre pour obtenir des messages différents affichés dans la boîte !

## Un paramètre plus complexe

Passons au paramètre suivant. Celui-ci va demander un peu plus de travail — selon la valeur du paramètre `msgType`, la fonction affichera une icône et une couleur d'arrière-plan différentes.

1. Tout d'abord, téléchargez les icônes nécessaires à cet exercice (warning et chat) depuis GitHub. Enregistrez-les dans un nouveau dossier appelé `icons` dans le même répertoire que votre fichier HTML.

> **Note :** icônes warning et chat trouvés sur iconfinder.com, et créés par Nazarrudin Ansyari . Merci !

2. Ensuite, trouvez le CSS à l'intérieur de votre fichier HTML. Nous ferons quelques changements pour faire place aux icônes. Tout d'abord, mettez à jour la largeur `.msgBox` en changeant :

   CSS
   ```
   width: 200px;
   ```

   par :

   CSS
   ```
   width: 242px;
   ```

3. Ensuite, ajoutez les lignes à l'intérieur de la règle CSS `.msgBox p { ... }` :

   CSS
   ```
   padding-left: 82px;
   background-position: 25px center;
   background-repeat: no-repeat;
   ```

4. Maintenant, nous devons ajouter du code à notre fonction `displayMessage()` pour gérer l'affichage de l'icône. Ajoutez le bloc suivant juste au dessus de l'accolade fermante " `}` " de votre fonction :

   JS
   ```
   if (msgType === "warning") {
     msg.style.backgroundImage = "url(icons/warning.png)";
     panel.style.backgroundColor = "red";
   } else if (msgType === "chat") {
     msg.style.backgroundImage = "url(icons/chat.png)";
     panel.style.backgroundColor = "aqua";
   } else {
     msg.style.paddingLeft = "20px";
   }
   ```

   Ici, quand `msgType` a la valeur `'warning'`, l'icône d'avertissement est affichée et le fond du panneau prend la couleur rouge. Si `msgType` a la valeur `'chat'`, l'icône de chat est affichée et l'arrière-plan du panneau est bleu. Si le paramètre `msgType` n'a pas de valeur du tout (ou s'il a une valeur totalement différente), alors la partie du code

si aucun paramètre `msgType` n'est fourni, ce qui signifie qu'il s'agit d'un paramètre facultatif !

5. Nous allons tester notre fonction mise à jour, essayez de mettre à jour l'appel `displayMessage()` :

   JS

   ```
   displayMessage("Woo, this is a different message!");
   ```

   par soit l'un ou l'autre :

   JS

   ```
   displayMessage("Your inbox is almost full — delete some mails", "warning");
   displayMessage("Brian: Hi there, how are you today?", "chat");
   ```

   Vous pouvez voir à quel point notre petite (plus tant que cela maintenant) fonction est devenue utile :

> **Note :** Si vous avez des difficultés à mettre en œuvre cet exemple, n'hésitez pas à verifier votre code par rapport à la version définitive sur GitHub (aussi, vous pouvez tester la démo ), ou nous demander de l'aide.

## Conclusion

Vous êtes venu à bout de cette activité, félicitations ! Cet article vous a amené à travers tout le processus de construction d'une fonction pratique personnalisée, qui avec un peu plus de travail pourrait être transposée dans un projet réel. Dans l'article suivant, nous allons conclure l'apprentissage des fonctions en expliquant un autre concept connexe essentiel — les valeurs de retour.

This page was last modified on 3 août 2023 by MDN contributors.