

# Introduction : applications Web

**Dans cette unité, vous serez initié aux applications Web.**

L'objectif de cette unité est de vous présenter les applications web. Vous découvrirez les applications web et les applications monopages (SPA) et en quoi elles diffèrent des sites web statiques. Les connaissances acquises dans les unités précédentes vous seront très utiles pour continuer à présenter et à mettre en forme vos informations.

Après cette unité, vous serez capable de :

- Définir ce qu'est une application Web et comment elle est construite
- Connaître les différences entre une application Web et un site Web statique
- Définir ce qu'est une application monopage (SPA)

L'apprentissage est une activité sociale. Quel que soit votre projet, n'hésitez pas à communiquer avec la communauté Codecademy sur les [forums](#).

N'oubliez pas de consulter régulièrement la communauté, notamment en demandant des révisions de code pour vos projets et en partageant vos révisions avec d'autres personnes dans la [catégorie « projets »](#), ce qui peut vous aider à consolider vos acquis.

## Qu'est-ce qu'une application Web ?

**Découvrez les applications Web et ce qu'implique leur création.**

Qu'est-ce qu'une application Web ?

Une application web est un logiciel qui ne nécessite aucune installation et est accessible depuis un serveur distant via un navigateur web. Conçues pour l'interaction, les applications web permettent aux utilisateurs d'échanger et de consommer des données entre le navigateur et le serveur web. Cette interaction peut être aussi simple que la connexion à un compte ou aussi complexe qu'un paiement par carte bancaire.

Quelle est la différence entre un site Web et une application Web ?

Bien que les termes « site web » et « application web » soient souvent utilisés de manière interchangeable, ils peuvent désigner des choses quelque peu différentes. Le plus souvent, un site web est défini comme un ensemble de pages contenant des informations, interconnectées et accessibles via un navigateur web. Une application web, quant à elle, est un logiciel applicatif exécuté sur un serveur web et accessible par l'utilisateur via un navigateur web.

Si l'on peut dire qu'un site web se définit par son contenu, une application web se définit par son interaction avec l'utilisateur. De ce fait, les applications web sont nettement plus complexes que les sites web statiques, tant en termes d'architecture générale que de fonctionnalités.

### Architecture des applications Web

Afin de faciliter ce flux complexe de données, les applications web sont généralement conçues avec différentes couches. Le paradigme de conception le plus courant est une conception à trois couches : une couche de présentation (navigateur web), une couche applicative (serveur) et une couche de stockage (base de données). Dans ce système, la couche de présentation est chargée de relayer les données utilisateur à la couche applicative, qui peut les traiter et effectuer diverses opérations, y compris leur transfert vers la couche de stockage pour les « sauvegarder ».

Les applications web peuvent souvent devenir très complexes. Dans ce cas, une conception à trois couches peut s'avérer insuffisante. L'ajout de couches supplémentaires peut alors être nécessaire pour gérer cette complexité. Par exemple, l'intégration d'une couche d'intégration entre les couches application et stockage peut contribuer à fournir une interface uniforme pour l'accès aux données, isolant ainsi la couche application des modifications apportées à la couche stockage.

Si vous souhaitez en savoir plus sur la technologie qui sous-tend vos applications web préférées, installez l' [extension Chrome Wappalyzer](#) . En naviguant sur un site, vous pourrez cliquer dessus et obtenir la liste des différentes technologies utilisées.

# Qu'est-ce qu'un SPA ?

Découvrez ce qu'est une application monopage (SPA) et pourquoi vous l'utiliserez.

## Introduction

À ce stade de votre parcours de développement web, vous avez appris le HTML pour créer des pages web, le CSS pour les styliser et le JavaScript pour apporter une touche de magie à vos pages. Fort de ces compétences fondamentales, vous êtes désormais prêt à explorer la pratique passionnante du développement web des applications monopages (SPA). Les SPA allient le contenu des sites web traditionnels à l'expérience utilisateur fluide des applications mobiles. Apprendre à développer et à maintenir des SPA est une aventure passionnante qui met les développeurs au défi d'améliorer l'expérience web des utilisateurs du monde entier.

## Applications multipages

Avant de lire cet article, vous devez vous familiariser avec le concept d'application web. Qu'il s'agisse de diffuser du contenu via des sites web statiques ou d'interactivité via des applications web, cet article fait référence à une structure de fichiers multipages hébergée sur un serveur. Chaque fois que de nouvelles données sont nécessaires à l'affichage du navigateur, une requête est envoyée au serveur, qui renvoie un nouvel ensemble de fichiers d'échange. Pour un site web statique, cette approche est généralement satisfaisante, mais les applications web qui nécessitent une interaction plus rapide et plus complexe peinent parfois à suivre.

Imaginez un restaurant où vous ne pouvez manger qu'une partie de votre repas à la fois. Pour une commande comprenant un burger, des frites et une salade, le serveur (et le serveur !) n'apporte que le burger. Après quelques bouchées, vous avez envie de frites ; il ramène donc le burger en cuisine et ne rapporte que les frites. Il a peut-être même dû attendre que les frites soient prêtes, c'est-à-dire préparées, avant de les remettre à table. Répétez la demande, préparez et servez avec la salade, puis le burger, puis encore des frites, et vous voilà dans le restaurant le plus étrange et le plus inefficace qui soit. Heureusement, cela n'arrive pas dans le secteur alimentaire, mais il se passe quelque chose de similaire sur le web.

Dans la vidéo suivante, un utilisateur navigue sur [Wikipédia](#). À droite, vous reconnaîtrez l'onglet « Éléments » des outils de développement Chrome, qui affiche un fichier HTML associé à la page dans le navigateur.

Chaque clic sur un lien affiche une nouvelle page dans DevTools, ce qui signifie qu'une requête a été envoyée au serveur et qu'un nouvel ensemble de fichiers a été envoyé pour affichage dans le navigateur. C'est comme notre hamburger, nos frites et notre salade de notre restaurant préféré. Dans une application web où l'interactivité est essentielle, ces requêtes de fichiers peuvent ralentir l'expérience utilisateur. C'est là qu'intervient l'application monopage.

### Applications d'une seule page

[Wikipédia définit](#) une application monopage (SPA) comme « une application web qui interagit avec le navigateur web en réécrivant dynamiquement la page web actuelle avec les nouvelles données d'un serveur web, au lieu de la méthode par défaut qui charge de nouvelles pages entières par le navigateur ». Le terme « application monopage » désigne généralement une application composée d'une seule page constamment mise à jour par JavaScript. Les requêtes au serveur sont désormais plus rapides, car elles contiennent uniquement les données nécessaires à la mise à jour de l'affichage. Les SPA sont des applications complètes, exécutées dans le navigateur tout en restant connectées à un serveur pour la mise à jour des données de l'application.

Si notre restaurant préféré adopte l'approche monopage, lorsque nous commandons, le serveur nous apporte immédiatement le burger, les frites et la salade. Nous pouvons ainsi déguster ce que nous voulons sans que le serveur n'ait à faire des allers-retours en cuisine. Nous pouvons parfois avoir besoin de ketchup pour les frites ou de vinaigrette pour la salade, mais ces ingrédients sont livrés rapidement et déjà préparés. Comme pour une application monopage, vous recevez presque tout ce dont vous avez besoin à l'avance, mais vous pouvez toujours demander des éléments plus petits pour chaque plat.

Nous avons maintenant une vidéo des interactions utilisateur sur la [page d'accueil de React](#), conçue comme une application monopage. Cette fois, dans le panneau « Éléments », chaque interaction ne modifie que certaines parties du fichier HTML.

Le fichier reste constant, tandis que la logique JavaScript côté client ne modifie que ce qui est nécessaire à la mise à jour de la vue. Les requêtes de données sont récupérées beaucoup plus rapidement que les fichiers devant être traités sur un serveur puis affichés dans le navigateur. Les SPA privilégient la rapidité d'interaction utilisateur et les temps d'affichage du navigateur.

### Cadres SPA

Pour créer une SPA, vous pouvez utiliser JavaScript pour contrôler toute la logique requise. En pratique, la complexité des SPA évolue rapidement ; il n'est donc pas recommandé d'utiliser uniquement JavaScript. Heureusement, plusieurs outils permettent de créer une SPA. Ces outils facilitent de nombreuses tâches, du contrôle de l'affichage de la page à la gestion du développement de l'application.

- [React](#) est une bibliothèque JavaScript populaire pour la création d'applications monopages. Elle se concentre sur la création de composants capables de s'afficher différemment en fonction de l'état actuel de l'application et des données utilisateur.
- [Vue.js](#) est un framework qui utilise la création de modèles au sein d'un même fichier HTML, tandis que la logique applicative contrôle le rendu. Cette approche est parfois considérée comme plus traditionnelle et donc plus facile à maîtriser.
- D'autres bibliothèques et frameworks incluent [AngularJS](#), [Ember.js](#), [ExtJS](#), [Knockout.js](#) et [Meteor.js](#). Bien que partageant tous des objectifs similaires, ils adoptent chacun une approche différente pour créer des applications sur mesure.

## Avantages et inconvénients du SPA

Alors, votre prochain projet devrait-il être une application monopage ? Peut-être. Peut-être pas. Comme pour la plupart des projets de développement, vous devez peser le pour et le contre avant de décider comment mettre en œuvre un nouveau projet.



VS.



## Avantages

- Les SPA sont rapides. Leur principal atout réside dans leur interface utilisateur similaire à celle d'une application de bureau ou mobile. En éliminant les demandes de nouveaux fichiers et en utilisant uniquement de plus petites quantités de données provenant du serveur, les SPA offrent une interface en temps réel avec leurs utilisateurs.
- La réutilisation du code est un atout majeur des SPA, car elle permet de gagner du temps au sein d'un même projet et entre plusieurs projets. De nombreuses bibliothèques et frameworks SPA recommandent que les composants soient suffisamment génériques pour pouvoir être réutilisés d'un projet à l'autre.

- Les SPA simplifient la migration du code vers une application mobile. Avec une SPA, le back-end de l'application alimente l'interface front-end découpée en données. Cette séparation des tâches permet de créer une interface utilisateur mobile tout en préservant la logique back-end de l'application.

## Inconvénients

- Les SPA nécessitent davantage de fichiers à exécuter au démarrage, ce qui peut allonger le temps de chargement de l'application. C'est un point à prendre en compte si l'utilisateur souhaite éviter de consulter un site dont le chargement est trop long. Le temps de chargement des SPA peut être réduit en chargeant stratégiquement les ressources tout au long de l'exécution de l'application.
- L'optimisation pour les moteurs de recherche (SEO) présente certains inconvénients en matière de SPA. Les moteurs de recherche, comme Google ou DuckDuckGo, indexent les pages d'un site web pour classer le contenu. Cela peut s'avérer complexe avec une seule page, qui peut ne contenir de contenu qu'après son chargement par JavaScript. Le SEO est un domaine en constante évolution ; des stratégies existent donc déjà pour atténuer ces inconvénients.
- Les SPA peuvent ne pas fonctionner comme prévu dans le navigateur. Par exemple, le bouton « Retour » ou l'historique de navigation peuvent réagir différemment lors de l'utilisation d'une application monopage. Cela peut être frustrant pour les utilisateurs qui attendent certaines fonctionnalités de leur navigateur.

## Conclusion

Les applications monopages offrent une meilleure expérience utilisateur lorsqu'elles s'exécutent dans un navigateur web. Elles constituent le choix idéal pour les applications nécessitant une interaction en temps réel ou complexe avec leurs utilisateurs. Créer une SPA ne se résume pas à quelques fichiers HTML, CSS et JavaScript, mais leur complexité est minimisée par des frameworks comme React et Vue.js. Même si votre prochaine application n'est pas une SPA, savoir la mettre en œuvre est une compétence essentielle pour un développeur front-end.

Prêt à créer vos propres SPA ? Découvrez nos [formations en développement web](#) pour vous lancer.

## Revue : Applications Web

**Dans cette unité, vous avez appris à connaître les applications Web.**

Félicitations ! L'objectif de cette unité était de vous présenter les applications web. Vous avez découvert les applications web et les applications monopages (SPA) et en quoi elles diffèrent des sites web statiques. Utilisez les connaissances acquises dans les unités précédentes pour présenter et styliser vos informations.

Après avoir terminé cette unité, vous êtes désormais capable de :

- Définir ce qu'est une application Web et comment elle est construite
- Connaître les différences entre une application Web et un site Web statique
- Définir ce qu'est une application monopage (SPA)

Si vous souhaitez en savoir plus sur ces sujets, voici quelques ressources supplémentaires :

- Ressource : [Résultats du sondage : « Sites » vs « Applications »](#)

L'apprentissage est une activité sociale. Quel que soit votre projet, n'hésitez pas à communiquer avec la communauté Codecademy sur les [forums](#). N'oubliez pas de consulter régulièrement la communauté, notamment en demandant des révisions de code pour vos projets et en partageant vos révisions avec d'autres personnes dans la [catégorie « projets »](#), ce qui peut vous aider à consolider vos acquis.

## Introduction : React, partie I

**Dans cette unité, vous serez initié à React.**

L'objectif de cette unité est de vous présenter React, la célèbre bibliothèque JavaScript. React vous aidera à créer des interfaces web plus performantes et évolutives grâce à la création de composants. Un aperçu d'ES6 et des concepts fonctionnels de JavaScript vous aidera à comprendre React.

Après cette unité, vous serez capable de :

- Comprendre les concepts de programmation ES6+ et JavaScript fonctionnel
- Découvrez ce qu'est un DOM virtuel et comment il est utilisé dans React
- Apprendre JSX
- Construire les premiers composants React

L'apprentissage est une activité sociale. Quel que soit votre projet, n'hésitez pas à communiquer avec la communauté Codecademy sur les [forums](#). N'oubliez pas de consulter régulièrement la communauté, notamment en demandant des révisions de code pour vos projets et en partageant vos révisions avec d'autres personnes dans la [catégorie « projets »](#), ce qui peut vous aider à consolider vos acquis.

# Mise à jour JavaScript

**Préparez-vous avec un rappel JavaScript avant de commencer à apprendre React.**

Avant de plonger dans l'univers de React, passons en revue quelques concepts fondamentaux de JavaScript. Les ressources ci-dessous vous aideront à mieux comprendre la syntaxe JavaScript et à l'appliquer à la création d'applications React.

- Document de référence : [Réintroduction de MDN à JavaScript](#) — Utilisez-le comme guide pour rafraîchir vos connaissances en syntaxe fondamentale
- Article : [Introduction au JavaScript fonctionnel](#) – Découvrez l'histoire de JS et ses paradigmes de programmation

Dans le prochain article, vous découvrirez également la déstructuration, une syntaxe largement utilisée dans les applications React.

# Déstructuration avec JavaScript

**Apprenez une nouvelle syntaxe pour gérer les objets et les tableaux en JavaScript.**

Qu'est-ce que la déstructuration ?

La déstructuration, ou affectation de déstructuration, est une fonctionnalité JavaScript qui facilite l'extraction de données à partir de tableaux et d'objets, introduite dans [la version ES6 de JavaScript](#).

À ce stade, nous supposons que vous savez déjà extraire des données de tableaux et d'objets. Cela signifie que vous n'avez pas besoin de déstructuration : la déstructuration est une forme de sucre syntaxique, ce qui signifie qu'elle simplifie l'écriture de certaines expressions, généralement en étant plus courte et plus claire que d'autres. Même si vous ne l'utilisez pas vous-même, vous la verrez probablement dans le code de quelqu'un d'autre ; il est donc important de la comprendre.

## Déstructuration des tableaux

Les structures de données telles que les tableaux et les objets peuvent être très utiles pour stocker de grandes quantités de données. La conversion des éléments d'un tableau en variables individuelles peut être fastidieuse :

```
let cars = ['ferrari', 'tesla', 'cadillac'];
```

Si nous voulions accéder à ces voitures individuellement et les affecter à des variables, nous pourrions faire ceci :

```
let cars = ['ferrari', 'tesla', 'cadillac'];
let car1 = cars[0];
let car2 = cars[1];
let car3 = cars[2];
console.log(car1, car2, car3); // Prints: ferrari tesla
cadillac
```

Nous pouvons utiliser la déstructuration pour raccourcir notre code et le rendre plus concis :

```
let cars = ['ferrari', 'tesla', 'cadillac'];
let [car1, car2, car3] = cars;
console.log(car1, car2, car3); // Prints: ferrari tesla
cadillac
```

Dans le code ci-dessus, nous avons créé trois variables (`car1`, `car2`, `car3`) correspondant aux trois éléments du `cars` tableau. Chaque nom de variable du nouveau tableau se verra attribuer la valeur de l'élément correspondant. Comme vous pouvez le constater, nous avons réussi à réaliser ce que nous avons fait précédemment avec trois lignes de code, en une seule ! Imaginez le nombre de lignes de code économisées avec un tableau de 10 éléments !

Pratiquons :

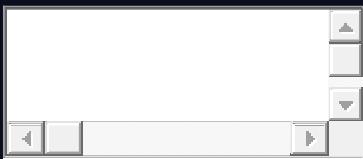
Question de codage

## Questions

Déstructurez les éléments du `colors` tableau en nouvelles variables `color1`, `color2`, et `color3`. Si cela est fait correctement, les couleurs devraient être imprimées dans cet ordre : « bleu », « rouge », « violet ».

Code

```
1  
2  
3  
4  
5  
6
```



Courir

Exécutez votre code pour vérifier votre réponse

Déstructuration des objets

Nous pouvons également utiliser l'affectation déstructurante avec des objets. Examinons un cas d'utilisation simple où nous capturons les propriétés d'un objet avec de nouvelles variables :

```
let destinations = { x: 'LA', y: 'NYC', z: 'MIA' };  
let x = destinations.x;  
let y = destinations.y;  
let z = destinations.z;  
console.log(x, y, z); // Prints LA NYC MIA
```

Avec la syntaxe de déstructuration simplifiée, nous accédons aux valeurs en faisant correspondre les noms de variables aux noms de propriétés.

```
let destinations = { x: 'LA', y: 'NYC', z: 'MIA' };
let { x, y, z } = destinations;
console.log(x, y, z); // Prints LA NYC MIA
```

Grâce à la syntaxe de déstructuration, nous pouvons créer de nouvelles variables directement à partir des propriétés d'un objet. Dans ce cas, nous avons pris les valeurs de `destination.x`, `destination.y` et `destination.z` et les avons stockées dans de nouvelles variables `x`, `y`, `z`, respectivement.

## Question de codage

### Questions

Déstructurez les éléments de l' `planets` objet afin qu'ils soient nommés `x`, `y`, et `z`. Si cela est fait correctement, les planètes devraient être imprimées dans cet ordre : « Saturne », « Mars », « Neptune ».

### Code

The image shows a Scratch script editor interface. It features two identical scripts stacked vertically. Each script consists of a green `repeat [ ] end` control block at the top, followed by a `say [ ] for [ ]` control block, and a `change [ ] by [ ]` control block at the bottom. The `repeat` loop has a value of 3. The `say` block contains the text "Neptune". The `change` block has a variable named "planete" and a value of 1. On the right side of the screen, there is a vertical column of numbers from 1 to 6, likely representing the script index or step number. The entire interface is set against a dark background.

Courir

## Vérifier la réponse



Tu l'as eu !

Paramètres de fonction de déstructuration

Les arguments de fonction sont un autre domaine où la déstructuration est utile. Au lieu d'accepter un objet complet comme argument, une fonction peut utiliser la déstructuration pour capturer des propriétés spécifiques sous forme de paramètres nommés.

```
let truck = {  
    model: '1977 Mustang convertible',  
    maker: 'Ford',  
    city: 'Detroit',  
    year: '1977',  
    convertible: true  
};  
  
const printCarInfo = ({model, maker, city}) => {  
    console.log(`The ${model}, or ${maker}, is in the city  
${city}.`);  
};  
  
printCarInfo(truck);  
// Prints: The 1977 Mustang convertible, or Ford, is in the  
city Detroit.
```

La `printCarInfo` fonction utilise la déstructuration d'objet pour créer trois variables de paramètres : `model`, `maker` et `city`. Lorsque la fonction est appelée avec l' `truck` objet, ces paramètres reçoivent les valeurs correspondantes de cet objet.

Notez que nous n'avons pas besoin d'utiliser toutes les propriétés de l' `truck` objet : nous créons uniquement des variables de paramètres pour les valeurs dont nous avons besoin.

Question de codage

## Questions

Définissez une fonction nommée `printPlantInfo()` pour que ce code enregistre la chaîne :

```
'The Prairie Rose, or Rosa arkansana, is in the kingdom Plantae'
```

Utilisez la déstructuration pour extraire les paramètres de la fonction dans la déclaration de fonction.

Code

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```



La ligne précédente 9 contenait une erreur

La rose des prairies, ou Rosa arkansana, appartient au règne des Plantae



Courir



Tu l'as eu !

Conclusion

La déstructuration permet de décompresser les valeurs de tableaux et d'objets et de les affecter à des variables ou des paramètres. La déstructuration offre d'autres possibilités intéressantes : renommer des variables, leur attribuer des valeurs par défaut, ignorer des valeurs, etc. Nous vous encourageons à consulter [la documentation MDN et à expérimenter](#). Bon code !

## React : le DOM virtuel

Découvrez comment le DOM virtuel de React empêche toute manipulation inutile du DOM.

Le problème

La manipulation du DOM est au cœur du web interactif moderne. Malheureusement, elle est aussi beaucoup plus lente que la plupart des opérations JavaScript.

Cette lenteur est aggravée par le fait que **la plupart des frameworks JavaScript mettent à jour le DOM bien plus souvent qu'ils ne le devraient**.

Par exemple, imaginons que vous ayez une liste contenant dix éléments. Vous cochez le premier élément. La plupart des frameworks JavaScript reconstruiraient *la liste entière*. C'est dix fois plus de travail que nécessaire ! Un seul élément a été modifié, mais les neuf autres sont reconstruits exactement comme avant.

Reconstruire une liste n'est pas un problème majeur pour un navigateur web, mais les sites web modernes peuvent nécessiter d'importantes manipulations du DOM. L'inefficacité des mises à jour est devenue un problème sérieux.

Pour résoudre ce problème, les gens de React ont popularisé ce qu'on appelle le *DOM virtuel*.

Le DOM virtuel

Voici un rappel de ce qui se passe dans les coulisses [du DOM virtuel](#).

Dans React, à chaque objet DOM correspond un « objet DOM virtuel ». Un objet DOM virtuel est une *représentation* d'un objet DOM, comme une copie allégée.

Un objet DOM virtuel possède les mêmes propriétés qu'un objet DOM réel, mais il n'a pas la capacité de l'objet réel à modifier directement ce qui est affiché à l'écran.

La manipulation du DOM est lente. La manipulation du DOM virtuel est beaucoup plus rapide, car rien n'est dessiné à l'écran. La manipulation du DOM virtuel est comparable à la modification d'un plan, plutôt qu'au déplacement de pièces dans une maison réelle.

Comment cela aide

Lorsque vous effectuez le rendu d'un élément JSX, chaque objet DOM virtuel est mis à jour.

Cela semble incroyablement inefficace, mais le coût est insignifiant car le DOM virtuel peut se mettre à jour très rapidement.

Une fois le DOM virtuel mis à jour, React compare le DOM virtuel avec un *instantané* du DOM virtuel pris juste avant la mise à jour.

En comparant le nouveau DOM virtuel avec une version antérieure à la mise à jour, React identifie *précisément les objets du DOM virtuel qui ont été modifiés*. Ce processus est appelé « diffing ».

Une fois que React sait quels objets virtuels du DOM ont été modifiés, il met à jour ces objets, *et uniquement ceux-là*, dans le DOM réel. Dans notre exemple précédent, React serait suffisamment intelligent pour reconstruire l'élément de liste coché et laisser le reste inchangé.

Cela fait une grande différence ! React ne peut mettre à jour que les parties nécessaires du DOM. La réputation de performance de React repose en grande partie sur cette innovation.

En résumé, voici ce qui se passe lorsque vous essayez de mettre à jour le DOM dans React :

1. L'ensemble du DOM virtuel est mis à jour.
2. Le DOM virtuel est comparé à son état avant sa mise à jour. React identifie les objets modifiés.
3. Les objets modifiés, et les objets modifiés uniquement, sont mis à jour sur le DOM *réel*.
4. Les modifications apportées au DOM réel entraînent un changement de l'écran.

Pourquoi React ?

1 min

React.js est une bibliothèque JavaScript développée par les ingénieurs de Facebook. Voici quelques raisons pour lesquelles les gens choisissent de programmer avec React :

- React est *rapide*. Les applications créées avec React peuvent gérer des mises à jour complexes tout en restant rapides et réactives.
- React est *modulaire*. Au lieu d'écrire des fichiers de code volumineux et denses, vous pouvez écrire de nombreux fichiers plus petits et réutilisables. La modularité de React peut être une solution intéressante aux [problèmes de maintenabilité](#) de JavaScript .
- React est *évolutif*. Les programmes volumineux affichant de nombreuses données variables sont ceux où React est le plus performant.
- React est *flexible*. Vous pouvez l'utiliser pour des projets intéressants qui n'ont rien à voir avec la création d'applications web. Le potentiel de React est encore à explorer. [Il reste encore beaucoup à explorer](#) .
- React est *populaire*. Bien que cela n'ait pas grand-chose à voir avec sa qualité, comprendre React améliore vos chances d'emploi.

Si vous débutez avec React, ce cours est fait pour vous ; aucune connaissance préalable de React n'est requise. Nous commencerons par les bases et progresserons progressivement. À la fin, vous serez prêt à programmer en React avec une compréhension approfondie de votre activité.

Bonjour le monde

2 min

Jetez un œil à la ligne de code suivante :

```
const h1 = <h1>Hello world</h1>;
```

Copy to Clipboard

Quel genre de code hybride étrange est-ce ? Est-ce du JavaScript ? Du HTML ? Ou autre chose ?

Il semble que ce soit du JavaScript, car il commence `const` et se termine par `;`. Si vous essayez de l'exécuter dans un fichier HTML, cela ne fonctionnerait pas.

Cependant, le code contient également `<h1>Hello world</h1>`, qui ressemble exactement à du HTML. Cette partie ne fonctionnerait pas si vous essayiez de l'exécuter dans un fichier JavaScript.

Le mystère révélé

<1 min

Revoyez la ligne de code que vous avez écrite. Ce code appartient-il à un fichier JavaScript, HTML ou ailleurs ?

La réponse est... un fichier JavaScript ! Malgré ses apparences, votre code ne contient en réalité aucun code HTML.

La partie qui ressemble à du HTML, `<h1>Hello world</h1>`, est quelque chose appelé

## JSX

Aperçu : Documents Chargement de la description du lien.

Qu'est-ce que JSX ?

1 min

## JSX

Preview: Docs JavaScript XML (JSX) is a syntax extension of JavaScript that provides highly functional and reusable markup code. It is used to create DOM elements which are then rendered in the React DOM. JSX provides a neat visual representation of the UI.

est une extension de syntaxe pour JavaScript. Elle a été conçue pour être utilisée avec React. Le code JSX ressemble beaucoup au HTML.

Que signifie « extension de syntaxe » ?

Dans ce cas, cela signifie que JSX n'est pas un JavaScript valide. Les navigateurs Web ne peuvent pas le lire !

Si un fichier JavaScript contient du code JSX, il devra être *compilé*. Cela signifie qu'avant que le fichier n'atteigne un navigateur web, un *compilateur JSX* traduira tout code JSX en JavaScript standard.

Les serveurs de Codecademy disposent déjà d'un compilateur JSX installé ; vous n'avez donc pas à vous en soucier pour le moment. Nous vous

expliquerons ultérieurement comment configurer un compilateur JSX sur votre ordinateur.

Éléments JSX

1 min

Une unité de base de

JSX

Preview: Docs JavaScript XML (JSX) is a syntax extension of JavaScript that provides highly functional and reusable markup code. It is used to create DOM elements which are then rendered in the React DOM. JSX provides a neat visual representation of the UI.

est appelé un *élément JSX*.

Voici un exemple d'élément JSX :

```
<h1>Hello world</h1>
```

Copy to Clipboard

Cet élément JSX ressemble exactement à du HTML ! La seule différence notable est qu'il se trouve dans un fichier JavaScript, et non dans un fichier HTML.

Éléments JSX et leur environnement

3 minutes

JSX

Aperçu : Documents Chargement de la description du lien

Les éléments sont traités comme *des expressions JavaScript*. Ils peuvent être placés partout où les expressions JavaScript le permettent. Cela signifie qu'un élément JSX peut être enregistré dans une variable, passé à une fonction, stocké dans un objet ou un tableau, etc.

Voici un exemple d'élément JSX enregistré dans une variable :

```
const navBar = <nav>I am a nav bar</nav>;
```

Copy to Clipboard

Voici un exemple de plusieurs éléments JSX stockés dans un objet :

```
const myTeam = {  
  center: <li>Benzo Walli</li>,  
  powerForward: <li>Rasha Loa</li>,  
  smallForward: <li>Tayshaun Dasmoto</li>,  
  shootingGuard: <li>Colmar Cumberbatch</li>,  
  pointGuard: <li>Femi Billon</li>  
};
```

Copy to Clipboard

Attributs dans JSX

5 minutes

## JSX

Aperçu : Documents Chargement de la description du lien

les éléments peuvent avoir *des attributs*, tout comme les éléments HTML.

Un attribut JSX s'écrit avec une syntaxe de type HTML : un *nom*, suivi d'un signe égal, puis d'une *valeur*. Cette *valeur* doit être entourée de guillemets, comme ceci :

```
my-attribute-name="my-attribute-value"
```

Copy to Clipboard

Voici quelques éléments JSX avec *des attributs* :

```
<a href='http://www.example.com'>Welcome to the Web</a>;  
  
const title = <h1 id='title'>Introduction to React.js: Part  
I</h1>;
```

Copy to Clipboard

Un seul élément JSX peut avoir plusieurs attributs, tout comme en HTML :

```
const panda = <img src='images/panda.jpg' alt='panda'  
width='500px' height='500px'>;
```

JSX imbriqué

5 minutes

Vous pouvez *imbriquer* des éléments JSX dans d'autres éléments JSX, comme en HTML.

Voici un exemple d' `<h1>`élément JSX, *imbriqué* dans un `<a>`élément JSX :

```
<a href="https://www.example.com"><h1>Click me!</h1></a>
```

Copy to Clipboard

Pour rendre cela plus lisible, vous pouvez utiliser des sauts de ligne et des retraits de style HTML :

```
<a href="https://www.example.com">
  <h1>
    Click me!
  </h1>
</a>
```

Copy to Clipboard

Si une expression JSX occupe plusieurs lignes, vous devez l'entourer de parenthèses. Cela peut paraître étrange au premier abord, mais on s'y habitue :

```
( 
  <a href="https://www.example.com">
    <h1>
      Click me!
    </h1>
  </a>
)
```

Copy to Clipboard

Les *expressions JSX imbriquées* peuvent être enregistrées comme variables, transmises à des fonctions, etc., tout comme les expressions JSX non imbriquées ! Voici un exemple d' expression JSX *imbriquée* enregistrée comme variable :

```
const theExample = (
  <a href="https://www.example.com">
```

```
<h1>
  Click me!
</h1>
</a>
);
```

## Éléments externes JSX

3 minutes

Il y a une règle que nous n'avons pas mentionnée : une expression JSX doit avoir exactement *un* élément le plus externe.

En d'autres termes, ce code fonctionnera :

```
const paragraphs = (
  <div id="i-am-the-outermost-element">
    <p>I am a paragraph.</p>
    <p>I, too, am a paragraph.</p>
  </div>
);
```

Copy to Clipboard

Mais ce code ne fonctionnera pas :

```
const paragraphs = (
  <p>I am a paragraph.</p>
  <p>I, too, am a paragraph.</p>
);
```

Copy to Clipboard

La première balise d'ouverture et la dernière balise de fermeture d'une expression JSX doivent appartenir au même élément JSX !

Il est facile d'oublier cette règle et de se retrouver avec des erreurs difficiles à diagnostiquer.

Si vous remarquez qu'une expression JSX comporte plusieurs éléments externes, la solution est généralement simple : envelopper l'expression JSX dans un `<div>` élément.

## Rendu JSX

Vous avez appris à écrire [JSX](#)

Aperçu : Documents Chargement de la description du lien

Des éléments ! Il est temps d'apprendre à les *représenter*.

Rendre *une* expression JSX signifie la faire apparaître à l'écran.

### Instructions

1. Point de contrôle 1 passé

1 .

Le code suivant rendra une expression JSX :

```
const container = document.getElementById('app');
const root = createRoot(container);
root.render(<h1>Hello world</h1>);
```

Explication du rendu JSX

7 minutes

Examinons le code que vous venez d'écrire dans le dernier exercice.

```
const container = document.getElementById('app');
const root = createRoot(container);
root.render(<h1>Hello world</h1>);
```

[Copy to Clipboard](#)

Avant de commencer, il est essentiel de comprendre que React s'appuie sur deux choses pour restituer : quel contenu restituer et où placer le contenu.

Avec cela à l'esprit, regardons la première ligne :

```
const container = document.getElementById('app')
```

[Copy to Clipboard](#)

Cette ligne :

- Utilise `document` qui représente notre page Web.
- Utilise la `getElementById()` méthode de `document` pour obtenir `Element` représentant l'élément HTML avec l'ID passé (`app`).
- Stocke l'élément dans `container`.

Dans la ligne suivante :

```
const root = createRoot(container)
```

Copy to Clipboard

Nous utilisons `createRoot()` la `react-dom/client` bibliothèque, qui crée une racine React `container` et la stocke dans un fichier `root`. Elle peut être utilisée pour générer une expression JSX. Il s'agit de la partie « où placer le contenu » du rendu React.

Enfin, la dernière ligne :

```
root.render(<h1>Hello world</h1>)
```

Copy to Clipboard

utilise la `render()` méthode de `root` pour restituer le contenu passé en argument. Nous passons ici un `<h1>` élément qui affiche `Hello world`. Il s'agit de la partie « Quel contenu restituer » du rendu React.

Passer une variable à render()

6 minutes

Dans l'exercice précédent, nous avons vu comment créer une racine React en utilisant `createRoot()` et utiliser sa `render()` méthode pour rendre

## JSX

Preview: Docs JavaScript XML (JSX) is a syntax extension of JavaScript that provides highly functional and reusable markup code. It is used to create DOM elements which are then rendered in the React DOM. JSX provides a neat visual representation of the UI.

.

L' `render()` argument de la méthode n'a pas besoin d'être JSX, mais il doit être évalué comme une expression JSX. L'argument peut également être une variable, à condition que cette variable soit évaluée comme une expression JSX.

Dans cet exemple, nous enregistrons une expression JSX sous forme de *variable* nommée `todoList`. Nous passons ensuite `todoList` l'argument `render()` :

```
const toDoList = (
  <ol>
    <li>Learn React</li>
    <li>Become a Developer</li>
  </ol>
);

const container = document.getElementById('app');
const root = createRoot(container);
root.render(toDoList);
```

## Le DOM virtuel

Une particularité de `render()` la méthode d'une racine React est qu'elle *ne met à jour que les éléments DOM qui ont changé*.

Cela signifie que si vous effectuez le rendu exact de la même chose deux fois de suite, le deuxième rendu ne fera rien :

```
const hello = <h1>Hello world</h1>

// This will add "Hello world" to the screen:
root.render(hello, document.getElementById('app'));

// This won't do anything at all:
root.render(hello, document.getElementById('app'));
```

Copy to Clipboard

C'est important ! La simple mise à jour des éléments DOM nécessaires contribue largement au succès de React. Ceci est rendu possible grâce au *DOM virtuel* de React .

Revoir

1 min

Félicitations ! Vous avez appris à créer et à restituer

[JSX](#)

Aperçu : Documents Chargement de la description du lien

Éléments ! C'est la première étape pour maîtriser React.

Dans cette leçon, nous avons appris que :

- React est un framework front-end modulaire, évolutif, flexible et populaire.
- JSX est une extension de syntaxe pour JavaScript qui nous permet de traiter le HTML comme des expressions.
  - Ils peuvent être stockés dans des variables, des objets, des tableaux et bien plus encore !
- Les éléments JSX peuvent avoir des attributs et être imbriqués les uns dans les autres, tout comme en HTML.
- JSX doit avoir exactement un élément externe, et d'autres éléments peuvent être imbriqués à l'intérieur.
- `createRoot()` from `react-dom/client` peut être utilisé pour créer une racine React sur l'élément DOM spécifié.
- Une méthode racine React `render()` peut être utilisée pour restituer JSX à l'écran.
- La méthode d'une racine React `render()` met uniquement à jour les éléments DOM qui ont changé à l'aide du DOM virtuel.

Au fur et à mesure que vous en apprendrez davantage sur React, vous découvrirez des choses puissantes que vous pouvez faire avec JSX, certains problèmes JSX courants et comment les éviter.

### **classe vs nom de classe**

5 minutes

Cette leçon couvrira des sujets plus avancés

### [JSX](#)

Aperçu : Documents Chargement de la description du lien

Vous apprendrez quelques astuces puissantes et quelques erreurs courantes à éviter.

La grammaire de JSX est globalement la même que celle du HTML, mais il existe des différences subtiles à prendre en compte. La plus fréquente concerne le mot `class`.

En HTML, il est courant d'utiliser `class` comme nom d'attribut :

```
<h1 class="big">Title</h1>
```

Copy to Clipboard

En JSX, le mot « ! » est interdit. `class` Il faut utiliser `className` :

```
<h1 className="big">Title</h1>
```

Copy to Clipboard

C'est parce que JSX est traduit en JavaScript et `class` est un mot réservé dans JavaScript.

Lorsque JSX est *rendu*, les attributs JSX `classNames` sont automatiquement rendus sous forme `class` d'attributs.

### **Etiquettes à fermeture automatique**

Une autre erreur JSX courante concerne *les balises à fermeture automatique*.

Qu'est-ce qu'une balise à fermeture automatique ?

La plupart des éléments HTML utilisent deux balises : une *balise d'ouverture* (`<div>`) et une *balise de fermeture* (`</div>`). Cependant, certains éléments HTML, comme `<img>` et `<input>` n'en utilisent qu'une seule. La balise d'un élément à balise unique n'est ni une balise d'ouverture ni une balise de fermeture ; c'est une *balise autofermante*.

Lorsque vous écrivez une balise à fermeture automatique en HTML, il est *facultatif* d'inclure une barre oblique immédiatement avant le crochet angulaire final :

```
// Fine in HTML with a slash:  
<br />
```

```
// Also fine, without the slash:  
<br>
```

Copy to Clipboard

Cependant, en JSX, il est nécessaire d'inclure la barre oblique. Si vous écrivez une balise autofermante en JSX et oubliez la barre oblique, vous générerez une erreur :

```
// Fine in JSX:  
<br />
```

```
// NOT FINE AT ALL in JSX:  
<br>
```

JavaScript dans votre JSX dans votre JavaScript

2 min

Jusqu'à présent, nous nous sommes concentrés sur l'écriture

## JSX

Aperçu : Documents Chargement de la description du lien

expressions. C'est similaire à l'écriture de morceaux de HTML, mais à l'intérieur d'un fichier JavaScript.

Dans cette leçon, nous allons ajouter quelque chose de nouveau : du JavaScript standard, écrit à l'intérieur d'une expression JSX, écrit à l'intérieur d'un fichier JavaScript.

### **Instructions**

1. Point de contrôle 1 passé

À partir de la ligne 7, écrivez soigneusement le code suivant.

```
root.render(<h1>2 + 3</h1>);
```

Copy to Clipboard

Que pensez-vous qui apparaîtra dans le navigateur ?

### Accolades bouclées en JSX

Le code du dernier exercice ne s'est pas comporté comme prévu. Au lieu d'additionner 2 et 3, il a affiché « 2 + 3 » sous forme de chaîne de texte. Pourquoi ?

Cela s'est produit parce que 2 + 3 qu'il est situé entre les balises `<h1>` et `</h1>`.

Tout code entre les balises d'un élément JSX sera lu comme du JSX, et non comme du JavaScript classique ! JSX n'ajoute pas de nombres : il les lit comme du texte, comme du HTML.

Vous avez besoin d'un moyen d'écrire du code qui dit : « Même si je suis situé entre des balises JSX, traitez-moi comme du JavaScript ordinaire et non comme du JSX. »

Vous pouvez le faire en enveloppant votre code dans *des accolades*.

### Instructions

1. Point de contrôle 1 passé

1.

Ajoutez une paire d'accolades au code du dernier exercice afin que votre expression JSX ressemble à ceci :

```
<h1>{2 + 3}</h1>
```

Copy to Clipboard

Tout ce qui se trouve à l'intérieur des accolades sera traité comme du JavaScript normal.

20 chiffres de Pi en JSX

6 minutes

Vous pouvez désormais injecter du JavaScript standard dans des expressions JSX ! Cela sera extrêmement utile.

Dans l'éditeur de code, vous pouvez voir une expression JSX qui affiche les vingt premiers chiffres de pi.

Étudiez l'expression et remarquez ce qui suit :

- Le code est écrit dans un fichier JavaScript. Par défaut, il sera traité comme du JavaScript standard.
- Recherchez `<div>` sur la ligne 5. À partir de là, jusqu'à `</div>`, le code sera traité comme JSX.
- Rechercher `Math`. À partir de là, jusqu'à `(20)`, le code sera à nouveau traité comme du JavaScript normal.
- Les accolades elles-mêmes ne seront pas traitées comme du JSX ou du JavaScript. Ce sont *des marqueurs* qui signalent le début et la fin d'une injection JavaScript dans JSX, à l'instar des guillemets qui signalent les limites d'une chaîne.

## Variables dans JSX

Lorsque vous injectez du JavaScript dans

### JSX

Aperçu : Documents Chargement de la description du lien

, que JavaScript fait partie du même environnement que le reste du JavaScript dans votre fichier.

Cela signifie que vous pouvez accéder aux variables à l'intérieur d'une expression JSX, même si ces variables ont été déclarées en dehors du bloc de code JSX.

```
// Declare a variable:  
const name = 'Gerdo';  
  
// Access your variable inside of a JSX expression:  
const greeting = <p>Hello, {name}!</p>;
```

[Copy to Clipboard](#)

Attributs variables dans JSX

6 minutes

Lors de l'écriture

## JSX

Aperçu : Documents Chargement de la description du lien

, il est courant d'utiliser des variables pour définir *des attributs* .

Voici un exemple de la façon dont cela pourrait fonctionner :

```
// Use a variable to set the `height` and `width` attributes:  
  
const sideLength = "200px";  
  
const panda = (  
    
);
```

Copy to Clipboard

Notez que dans cet exemple, `<img />` chaque attribut « » a sa propre ligne. Cela peut rendre votre code plus lisible si vous avez beaucoup d'attributs pour un même élément.

*Les propriétés d'objet* sont également souvent utilisées pour définir des attributs :

```
const pics = {  
  panda: "http://bit.ly/1Tqltv5",  
  owl: "http://bit.ly/1XGtKM3",  
  owlCat: "http://bit.ly/1Upbczi"  
};  
  
const panda = (  
  <img  
    src={pics.panda}  
    alt="Lazy Panda" />  
);  
  
const owl = (  
  <img  
    src={pics.owl}  
    alt="Unimpressed Owl" />  
);  
  
const owlCat = (  
  <img  
    src={pics.owlCat}
```

```
    alt="Ghastly Abomination" />
);
```

Copy to Clipboard

## Écouteurs d'événements dans JSX

7 minutes

### JSX

Aperçu : Documents Chargement de la description du lien

Les éléments peuvent avoir *des écouteurs d'événements*, tout comme les éléments HTML. Programmer en React implique de travailler constamment avec des écouteurs d'événements.

Pour créer un écouteur d'événements, attribuez un *attribut* spécial à un élément JSX . Voici un exemple :

```
<img onClick={clickAlert} />
```

Copy to Clipboard

*Le nom* d'un attribut d'écouteur d'événement doit être du type `onClick` ou `onMouseOver` : le mot `on` suivi du type d'événement que vous écoutez. Consultez la [liste des composants courants dans la documentation React](#) pour découvrir les noms d'événements pris en charge.

*La valeur* d'un attribut d'écouteur d'événements doit être une fonction. L'exemple ci-dessus ne fonctionnerait que s'il `clickAlert` agissait d'une fonction valide définie ailleurs :

```
function clickAlert() {
  alert('You clicked this image!');
}

<img onClick={clickAlert} />
```

Copy to Clipboard

Notez qu'en HTML, *les noms* des écouteurs d'événements sont écrits en minuscules, comme «`onclick` or » `onmouseover`. En JSX, ils sont écrits en camelCase, comme «`onClick` or » `onMouseOver`.

Conditions JSX : instructions If qui ne fonctionnent pas  
1 min

Excellent travail ! Vous avez appris à utiliser des accolades pour injecter du JavaScript dans un

## JSX

Aperçu : Documents Chargement de la description du lien  
expression!

Voici une règle que vous devez connaître : vous ne pouvez pas injecter une `if` instruction dans une expression JSX.

Ce code va casser :

```
(  
  <h1>  
    {  
      if (purchase.complete) {  
        'Thank you for placing an order!'  
      }  
    }  
  </h1>  
)
```

Copy to Clipboard

Que faire si vous souhaitez qu'une expression JSX s'affiche, mais uniquement dans certaines circonstances ? Vous ne pouvez pas injecter d'`if` instruction.  
Que faire ?

Vous disposez de nombreuses options. Dans les prochaines leçons, nous explorerons quelques méthodes simples pour écrire des

*conditions* (expressions qui ne sont exécutées que sous certaines conditions) en JSX.

Conditions JSX : instructions « If » efficaces

8 minutes

Comment pouvez-vous écrire une *condition* si vous ne pouvez pas injecter une `if` instruction dans JSX ?

Une option consiste à écrire une `if` déclaration et à *ne pas* l'injecter dans JSX

Aperçu : Documents Chargement de la description du lien

.

Regardez **if.js**. Suivez l' `if` instruction, de la ligne 8 à la ligne 20.

**if.js** fonctionne car les mots `if` et `else` sont *pas* injectés entre les balises JSX. L' `if` instruction est externe et aucune injection JavaScript n'est nécessaire.

Il s'agit d'une manière courante d'exprimer des conditions dans JSX.

```
const container = document.getElementById('app');
const root = createRoot(container);
function coinToss() {
  // This function will randomly return either 'heads' or 'tails'.
  return Math.random() < 0.5 ? 'heads' : 'tails';
}

const pics = {
  kitty: 'https://content.codecademy.com/courses/React/react_photo-kitty.jpg',
  doggy: 'https://content.codecademy.com/courses/React/react_photo-puppy.jpeg'
};
let img;

// if/else statement begins here:
```

```
if(coinToss() === 'heads'){
  img = <img src={pics.kitty}/>
}else{
  img = <img src={pics.doggy}/>
}
root.render(img);
```

Conditions JSX : l'opérateur ternaire

6 minutes

Il existe une manière plus compacte d'écrire des conditions dans JSX :  
l'*opérateur ternaire*.

L'opérateur ternaire fonctionne de la même manière en React qu'en JavaScript standard. Cependant, il apparaît étonnamment souvent en React.

Rappelons son fonctionnement : vous écrivez `x ? y : z`, où `x`, `y` et `z` sont des expressions JavaScript. Lors de l'exécution de votre code, `x` est évalué comme « vrai » ou « faux ». Si `x` est vrai, alors l'opérateur ternaire entier renvoie `y`. Si `x` est faux, alors l'opérateur ternaire entier renvoie `z`.

Voici comment vous pourriez utiliser l'opérateur ternaire dans un

[JSX](#)

Aperçu : Documents Chargement de la description du lien

expression:

```
const headline = (
  <h1>
    { age >= drinkingAge ? 'Buy Drink' : 'Do Teen Stuff' }
  </h1>
);
```

[Copy to Clipboard](#)

Dans l'exemple ci-dessus, si `age` est supérieur ou égal à `drinkingAge`, alors `headline` sera égal à `<h1>Buy Drink</h1>`. Sinon, `headline` sera égal à `<h1>Do Teen Stuff</h1>`.

Conditions JSX : &&

6 minutes

Nous allons aborder une dernière manière d'écrire des conditionnelles dans React : l' `&&`opérateur.

Comme l'opérateur ternaire, `&&il` n'est pas spécifique à React, mais il apparaît très souvent dans React.

Dans les deux derniers exercices, vous avez écrit des instructions qui rendraient parfois un chaton et d'autres fois un chien. `&&n'`aurait *pas* été le meilleur choix pour ce code.

`&&fonctionne` mieux pour les conditionnels qui parfois effectuent une action mais d'autres fois ne font *rien du tout* .

Voici un exemple :

```
const tasty = (
  <ul>
    <li>Applesauce</li>
    { !baby && <li>Pizza</li> }
    { age > 15 && <li>Brussels Sprouts</li> }
    { age > 20 && <li>Oysters</li> }
    { age > 25 && <li>Grappa</li> }
  </ul>
);
```

[Copy to Clipboard](#)

Si l'expression à gauche de est `&&évaluée` comme vraie, alors le

[JSX](#)

Aperçu : Documents Chargement de la description du lien

à droite de `&&sera` affiché. Cependant, si la première expression est fausse, le JSX à droite de `&&sera` ignoré et non affiché.

.map en JSX

8 minutes

La `.map()`méthode array est souvent utilisée en React. Il est conseillé de prendre l'habitude de l'utiliser avec JSX.

Si vous souhaitez créer une liste d'éléments JSX, utiliser `.map()`est souvent la méthode la plus efficace. Cela peut paraître étrange au premier abord :

```
const strings = [ 'Home', 'Shop', 'About Me' ];

const listItems = strings.map(string => <li>{string}</li>);
```

```
<ul>{listItems}</ul>
```

Copy to Clipboard

Dans l'exemple ci-dessus, nous commençons avec un tableau de chaînes. Nous appelons `.map()` ce tableau et l' `.map()` appelle renvoie un nouveau tableau de `<li>` chaînes.

Sur la dernière ligne de l'exemple, notez que cela `{listItems}` sera évalué comme un tableau, car c'est la valeur renvoyée par `.map()!` `<li>` Les JSX n'ont pas besoin d'être dans un tableau comme celui-ci, mais ils *peuvent* l'être.

```
// This is fine in JSX, not in an explicit array:  
<ul>  
  <li>item 1</li>  
  <li>item 2</li>  
  <li>item 3</li>  
</ul>  
  
// This is also fine!  
const liArray = [  
  <li>item 1</li>,  
  <li>item 2</li>,  
  <li>item 3</li>  
];  
  
<ul>{liArray}</ul>
```

Clés

6 minutes

Lorsque vous faites une liste dans

[JSX](#)

Aperçu : Documents Chargement de la description du lien

, parfois votre liste devra inclure quelque chose appelé `keys`:

```
<ul>  
  <li key="li-01">Example1</li>  
  <li key="li-02">Example2</li>  
  <li key="li-03">Example3</li>  
</ul>
```

Copy to Clipboard

A `key` est un attribut JSX. Son *nom* est `key`. Sa *valeur* doit être unique, similaire à celle d'un `id` attribut.

`keys` Ne faites rien de visible ! React les utilise en interne pour suivre les listes. Si vous n'utilisez pas les clés au bon moment, React risque de mélanger accidentellement les éléments de votre liste dans le mauvais ordre.

Toutes les listes ne nécessitent pas nécessairement `keys`. Une liste nécessite `.keys`

1. Les éléments d'une liste conservent *leur mémoire* d'un rendu à l'autre. Par exemple, lors du rendu d'une liste de tâches, chaque élément doit se souvenir s'il a été coché. Les éléments ne doivent pas devenir amnésiques lors du rendu.
2. L'ordre d'une liste peut être modifié. Par exemple, une liste de résultats de recherche peut être modifiée d'un rendu à l'autre.

Si aucune de ces conditions n'est remplie, inutile de vous inquiéter `keys`. En cas de doute, n'hésitez pas à les utiliser !

#### Instructions

1. Point de contrôle 1 passé

1 .

À la ligne 10, donnez `<li>` un `key` attribut à l'élément.

Quelle devrait être *la valeur key* de ?

Chacun `key` doit être une chaîne unique que React peut utiliser pour associer correctement chaque élément rendu à son élément correspondant dans le tableau.

Ainsi, pour chaque élément de `people`, nous devons générer une valeur unique. Comment parvenir à produire des clés uniques ?

Tout d'abord, ajoutez un `i` paramètre à `.map()` la fonction interne de afin de pouvoir accéder à l'index unique de chaque personne :

```
const peopleList = people.map((person, i) =>
```

Copy to Clipboard

Vous pouvez désormais obtenir une clé unique sur chaque boucle en ajoutant l'attribut suivant à l'`<li>` élément :

```
key={'person_' + i}
```

Copy to Clipboard

React.createElement

3 minutes

Vous pouvez écrire du code React sans utiliser

## JSX

Aperçu : Documents Chargement de la description du lien  
du tout!

La majorité des programmeurs React utilisent JSX, mais vous devez comprendre qu'il est possible d'écrire du code React sans lui.

L'expression JSX suivante :

```
const h1 = <h1>Hello world</h1>;
```

Copy to Clipboard

peut être réécrit sans JSX, comme ceci :

```
const h1 = React.createElement(  
  "h1",  
  null,  
  "Hello world"  
);
```

Lorsqu'un élément JSX est compilé, le compilateur *transforme* l'élément JSX en la méthode que vous voyez ci-dessus : `React.createElement()`. Chaque élément JSX est secrètement un appel à `React.createElement()`.

Nous n'entrerons pas en détail dans son `React.createElement()` fonctionnement, mais consultez [la documentation `React.createElement\(\)`](#) pour en savoir plus.

## Composants React

4 min

Les applications React sont constituées de **composants**.

Qu'est-ce qu'un composant ?

Un composant est un petit morceau de code réutilisable chargé d'une tâche. Cette tâche consiste souvent à afficher du code HTML et à le réafficher lorsque certaines données changent.

Jetez un œil au code ci-dessous. Il créera et affichera un nouveau composant React :

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function MyComponent() {
  return <h1>Hello world</h1>;
}

ReactDOM.createRoot(
document.getElementById('app')
).render(<MyComponent />);
```

Beaucoup d'entre eux vous semblent inconnus, mais pas d'inquiétude. Nous allons décortiquer ce code, petit morceau à la fois. À la fin de cette leçon, vous saurez comment créer un composant React !

## Importer React

3 minutes

Le premier composant React créé lors du dernier exercice a commencé par importer `react`. La ligne qui a effectué cette opération est :

```
import React from 'react';
```

Cela crée un objet nommé `React`, qui contient les méthodes nécessaires à l'utilisation de la bibliothèque React. Ce dernier `React` est importé depuis le `'react'` package, qui doit être installé dans votre projet en tant que dépendance. Grâce à cet objet, nous pouvons commencer à utiliser les fonctionnalités de la `react` bibliothèque !

En important la bibliothèque, nous pouvons utiliser des fonctionnalités importantes telles que

### Crochets

Aperçu : Documents Chargement de la description du lien

, que nous explorerons en détail plus tard.

Pour les prochains exercices, nous travaillerons sur deux fichiers : **App.js** et **index.js** . Dans une application React, le fichier **App.js** constitue généralement le niveau supérieur de votre application, et **index.js** le point d'entrée.

## Importer ReactDOM

3 minutes

Une autre importation dont nous avons besoin en plus de React est **ReactDOM** :

```
import ReactDOM from 'react-dom/client';
```

[Copy to Clipboard](#)

Les méthodes importées depuis 'react-dom' interagissent avec le DOM.

Les méthodes importées depuis React 'react' ne traitent pas du tout du DOM. Elles n'interagissent pas directement avec quoi que ce soit qui ne fasse pas partie de React.

Pour clarifier : le DOM est *utilisé* dans les applications React, mais il n'en fait pas *partie*. Après tout, le DOM est également utilisé dans de nombreuses applications non React. Les méthodes importées depuis React 'react' sont exclusivement destinées à des applications React pures, comme la création de composants ou l'écriture.

### [JSX](#)

Aperçu : Documents Chargement de la description du lien

éléments.

Créer un composant de fonction

5 minutes

Vous avez appris qu'un *composant React* est un petit morceau de code réutilisable qui est responsable d'une tâche, qui implique souvent le rendu HTML et son nouveau rendu chaque fois que certaines données changent.

Il est utile de considérer les composants comme des éléments plus petits de notre interface. Ensemble, ils constituent les éléments constitutifs d'une application React. Sur un site web, nous pouvons créer un composant pour la barre de recherche, un autre pour la barre de navigation et un autre pour le contenu du tableau de bord.

Voici un autre fait concernant les composants : nous pouvons utiliser des fonctions JavaScript pour définir un nouveau composant React. C'est ce qu'on appelle un **composant de fonction**.

Auparavant, les composants React étaient définis à l'aide de classes JavaScript. Mais depuis l'introduction de

### [Crotchets](#)

Aperçu : Documents Chargement de la description du lien

(quelque chose dont nous discuterons plus tard), les composants de fonction sont devenus la norme dans les applications React modernes.

Après avoir défini notre composant fonctionnel, nous pouvons l'utiliser pour créer autant d'instances de ce composant que nous le souhaitons.

Prenons l'exemple du premier exercice :

```
import React from 'react';

function MyComponent() {
  return <h1>Hello, I'm a functional React Component!</h1>;
}

export default MyComponent;
```

Copy to Clipboard

Sur la troisième ligne, une fonction nommée `MyComponent`. À l'intérieur, elle renvoie un élément React dans

## JSX

Aperçu : Documents Chargement de la description du lien

syntaxe:

```
return <h1>Hello, I'm a functional React Component!</h1>;
```

Copy to Clipboard

Combiné, cela constitue un composant fonctionnel React de base.

Sur la dernière ligne du bloc de code ci-dessus, `MyComponent` est exporté afin de pouvoir être utilisé ultérieurement.

Beaucoup de choses sont encore inconnues, mais vous pouvez en comprendre plus qu'avant ! Continuons !

Nommez un composant fonctionnel

2 min

Bien ! Créer une fonction JavaScript est le moyen de déclarer un nouveau *composant fonctionnel*.

Lorsque vous déclarez un nouveau composant fonctionnel, vous devez lui donner un *nom*. Sur notre composant final, le nom était `MyComponent` :

```
function MyComponent() {  
  return <h1>Hello world</h1>;  
}
```

Copy to Clipboard

Les noms des composants de fonction doivent commencer par une majuscule et sont généralement créés en PascalCase !

## JSX

Aperçu : Documents Chargement de la description du lien

les balises sont compilées, la majuscule indique qu'il s'agit d'un composant React plutôt que d'une balise HTML.

C'est une particularité de React ! Si vous créez un composant, veillez à le nommer en commençant par une majuscule afin qu'il soit interprété comme un composant React. S'il commence par une minuscule, React recherchera un composant intégré tel que «`div` and » `input` à la place, et échouera.

Instructions des composants fonctionnels

2 min

Récapitulons ce que vous avez appris jusqu'à présent ! Retrouvez chacun de ces points dans `App.js` et `index.js` :

- Sur la ligne 1 de `App.js` et `index.js`, `import React from 'react'` un objet JavaScript est créé. Cet objet contient les propriétés nécessaires au fonctionnement de React, telles que `React.createElement()`.
- La ligne 2 du **fichier `index.js`** `import ReactDOM from 'react-dom/client'` crée un autre objet JavaScript. Cet objet contient des méthodes permettant à React d'interagir avec le DOM, telles que `ReactDOM.createRoot()`.
- À la ligne 3 de `App.js`, en écrivant une fonction JavaScript, un composant de fonction a été défini. Nous ne pouvons pas encore voir ce composant, car il s'agit plutôt d'une fabrique qui produit des instances d'elle-même lorsqu'elle est utilisée. Pour le voir, nous devons le restituer dans le DOM.
- Chaque fois que vous créez un composant fonctionnel, vous devez lui attribuer un nom. Ce nom doit être écrit en majuscules Pascal, comme ceci : `UpperCamelCase`.

Ce dont nous *n'avons pas* encore parlé, c'est le *corps* de votre composant de fonction : la paire d'accolades après la définition de la fonction et tout le code entre ces accolades.

Comme toutes les fonctions JavaScript, celle-ci nécessite un corps. Ce corps agira comme un ensemble d'instructions expliquant à votre composant de fonction comment créer un composant React.

Voici à quoi ressemblerait le corps de votre fonction seul, sans le reste de la syntaxe de déclaration de fonction. Vous le trouverez dans **App.js** :

```
return <h1>Hello, this is a function component body.</h1>;
```

Copy to Clipboard

Cela ne ressemble pas à un ensemble d'instructions expliquant comment créer un composant React ! Pourtant, c'est exactement ce que c'est.

Cliquez sur Suivant et nous verrons comment fonctionnent ces instructions.

### Le mot-clé Return dans les composants fonctionnels

4 min

Lorsque nous définissons un composant fonctionnel, nous définissons une fabrique capable de construire la combinaison d'éléments appropriée à chaque fois que nous référençons son nom. Elle le construit en consultant un ensemble d'instructions que vous devez fournir.

Si vous pensez : « Cela ressemble exactement à la fonction d'une fonction JavaScript classique », vous avez raison ! Les composants fonctionnels peuvent être considérés de manière très similaire aux fonctions JavaScript classiques, à la différence que leur rôle consiste à assembler une partie de l'interface à partir d'instructions données !

Parlons un peu plus de ces instructions.

Pour commencer, ces instructions doivent prendre la forme d'un corps de déclaration de fonction. Elles seront donc délimitées par des accolades, comme ceci :

```
function Button() {  
  // Instructions go here, between the curly braces.  
}
```

Copy to Clipboard

Nos instructions peuvent inclure une combinaison de balisage, de CSS et de JavaScript pour produire le résultat souhaité. L'élément que nous devons toujours inclure est une instruction **de retour**.

La fonction est censée produire

[JSX](#)

Aperçu : Documents Chargement de la description du lien

Code permettant d'afficher un élément sur l'écran du navigateur. Ainsi, lorsque nous définissons des composants fonctionnels, nous devons renvoyer un élément JSX.

```
function BackButton() {  
  return <button>Back To Home</button>;  
}
```

Copy to Clipboard

Bien sûr, cela n'affiche pas `<button>Back To Home</button>` encore tout à fait le rendu sur l'écran du navigateur. Nous avons seulement défini notre composant.

Continuons pour voir comment le rendre et pourquoi l'instruction de retour était nécessaire !

## Importation et exportation de composants React

5 minutes

Il nous reste encore un peu de travail à faire avant de pouvoir utiliser notre composant défini et le rendre sur le DOM.

Nous avons mentionné précédemment qu'une application React comporte généralement deux fichiers principaux : **App.js** et **index.js**. Le fichier **App.js** constitue le niveau supérieur de votre application, et **index.js** en est le point d'entrée.

Jusqu'à présent, nous avons défini le composant à l'intérieur de **App.js**, mais comme **index.js** est le point d'entrée, nous devons l'exporter vers **index.js** pour le rendre.

Les composants React sont très utiles car ils sont réutilisables. Nous pouvons séparer, organiser et réutiliser nos composants en les plaçant dans des fichiers distincts et en les exportant là où nous en avons besoin.

Pour les exporter, nous pouvons le préfixer avec la `export` déclaration et préciser s'il s'agit d'une exportation par défaut ou nommée. Dans ce cas, nous nous en tiendrons à `default`. Pour un rappel sur `export`, consultez la [documentation web de MDN](#).

Après la définition du composant de fonction, dans **App.js**, nous pouvons exporter par défaut notre composant comme suit :

```
export default MyComponent;
```

Copy to Clipboard

Nous pouvons accéder à notre fichier **index.js** pour importer notre composant depuis '`./App`':

```
import MyComponent from './App';
```

Copy to Clipboard

Cela nous permettra de l'utiliser `MyComponent` dans **index.js**.

## Utilisation et rendu d'un composant

8 minutes

Maintenant que nous avons un composant de fonction défini, nous pouvons commencer à l'utiliser.

Nous pouvons l'utiliser avec une syntaxe de type HTML qui ressemble à une balise à fermeture automatique :

```
<MyComponent />
```

Copy to Clipboard

Si vous devez imbriquer d'autres composants entre les deux, vous pouvez également utiliser une structure de balise d'ouverture et de fermeture correspondante :

```
<MyComponent>
  <OtherComponent />
</MyComponent>
```

Copy to Clipboard

Cependant, pour afficher notre composant dans le navigateur, nous devons utiliser les méthodes `.createRoot()` et de la bibliothèque. Cela doit être fait dans notre point d'entrée, **index.js** ..`render()``react-dom`

Tout d'abord, nous appelons la `createRoot` méthode pour créer un conteneur racine React pour afficher le contenu. Les applications React possèdent

généralement un seul nœud DOM racine, et tout ce qu'il contient est géré par React DOM.

En d'autres termes, nous donnons `createRoot` un élément DOM dans lequel effectuer le rendu, et React prendra en charge la gestion du DOM à l'intérieur.

Voici un exemple :

```
ReactDOM.createRoot(document.getElementById('app'));
```

Copy to Clipboard

Super ! Décomposons-le un peu plus :

- `document.getElementById('app')` renvoie un élément DOM de **index.html**.
- `.createRoot()` reçoit l'élément DOM comme premier argument et crée une racine pour celui-ci.
- `.createRoot()` renvoie une référence au conteneur racine sur lequel vous pouvez appeler des méthodes comme `.render()`.

Une fois la racine créée, il ne reste plus qu'à appeler la `.render()` méthode sur la racine renvoyée et afficher le composant React comme ceci :

```
ReactDOM.createRoot(document.getElementById('app')).render(<MyComponent />);
```

Copy to Clipboard

À partir de là, React s'affichera `<MyComponent />` dans la racine et le fera apparaître à l'écran.

Dans une application entièrement développée avec React, cette opération n'est nécessaire qu'une seule fois. Une fois configurée, React gère le DOM de votre application et toutes les mises à jour de l'interface utilisateur sont gérées efficacement. L'ajout de composants supplémentaires doit être effectué dans votre fichier **App.js** principal .

**Revoir**

6 minutes

Dans cette leçon, vous avez découvert un concept fondamental de React : les composants.

Avant de partir, voici un récapitulatif :

- Les applications React sont constituées de **composants** .
- Les composants sont responsables du rendu des éléments de l'interface utilisateur.
- Pour créer des composants et les restituer, `react` et `reactDOM` doivent être importés.
- Les composants React peuvent être définis avec des fonctions Javascript pour créer **des composants de fonction** .
- Les noms des composants de fonction doivent commencer par une lettre majuscule et la casse Pascal est la convention de nommage adoptée.
- Les composants de fonction doivent renvoyer certains éléments React dans

## JSX

Aperçu : Documents Chargement de la description du lien

syntaxe.

- Les composants React peuvent être exportés et importés d'un fichier à un autre.
- Un composant React peut être utilisé en appelant le nom du composant dans une syntaxe de balise à fermeture automatique de type HTML.
- Le rendu d'un composant React nécessite d'utiliser `.createRoot()` pour spécifier un conteneur racine et d'appeler la `.render()` méthode dessus.

Ouf ! C'était beaucoup, mais les composants sont au cœur de React et c'est l'une des raisons pour lesquelles React est un outil si puissant !

Utiliser JSX multiligne dans un composant

9 minutes

Dans cette leçon, vous apprendrez quelques méthodes courantes qui

## JSX

Aperçu : Documents Chargement de la description du lien

Les composants JSX et React fonctionnent ensemble. Vous vous familiariserez avec les composants JSX et React tout en découvrant de nouvelles astuces.

Jetez un œil à ce code HTML :

```
<blockquote>
  <p>
    The world is full of objects, more or less interesting; I do not
    wish to add any more.
  </p>
  <cite>
    <a target="_blank"
        href="https://en.wikipedia.org/wiki/Douglas_Huebler">
      Douglas Huebler
    </a>
  </cite>
</blockquote>
```

Copy to Clipboard

Comment pourriez-vous faire en sorte qu'un composant React renvoie ce code HTML ?

Sélectionnez **QuoteMaker.js** pour voir une façon de procéder.

L'élément clé à noter `QuoteMaker` est la présence de parenthèses dans l'`return` instruction, aux lignes 4 et 16. Jusqu'à présent, vos `return` instructions de composants de fonction ressemblaient à ceci, sans aucune parenthèse :

```
return <h1>Hello world</h1>;
```

Copy to Clipboard

Cependant, une expression JSX multiligne doit toujours être entourée de parenthèses ! C'est pourquoi `QuoteMaker` l'instruction `return` de est entourée de parenthèses.

Utiliser un attribut variable dans un composant

10 minutes

Jetez un œil à cet objet JavaScript nommé `redPanda`:

```
const redPanda = {
  src:
  'https://upload.wikimedia.org/wikipedia/commons/b/b2/Endangered
```

```
_Red_Panda.jpg',
  alt: 'Red Panda',
  width: '200px'
};
```

Copy to Clipboard

Comment pourriez-vous rendre un composant React avec une image `redPanda` et ses propriétés ?

Sélectionnez **RedPanda.js** pour voir une façon de procéder.

Notez toutes les injections JavaScript entre accolades dans l'`return` instruction. Vous pouvez, et vous le ferez souvent, injecter du JavaScript dans [JSX](#)

Aperçu : Documents Chargement de la description du lien

à l'intérieur de la `return` déclaration.

Mettre la logique dans un composant de fonction

11 min

Un composant fonctionnel doit contenir une `return` instruction. Cependant, ce n'est pas *tout*.

Vous pouvez également mettre des calculs simples qui doivent être effectués avant de retourner votre

[JSX](#)

Aperçu : Documents Chargement de la description du lien

élément dans le composant de fonction.

Voici un exemple de quelques calculs qui peuvent être effectués à l'intérieur d'un composant de fonction :

```
function RandomNumber() {
  //First, some logic that must happen before returning
  const n = Math.floor(Math.random() * 10 + 1);
  //Next, a return statement using that logic:
  return <h1>{n}</h1>
```

```
}
```

Copy to Clipboard

Attention à cette erreur courante :

```
function RandomNumber() {
  return (
    const n = Math.floor(Math.random() * 10 + 1);
    <h1>{n}</h1>
  )
}
```

Dans l'exemple ci-dessus, la ligne avec la `const n` déclaration provoquera une erreur de syntaxe, car elle devrait venir avant le `return`.

Utiliser une condition dans un composant de fonction

10 minutes

Comment pourriez-vous utiliser une instruction *conditionnelle* à l'intérieur d'un composant de fonction ?

Sélectionnez **TodaysPlan.js** pour voir une façon de le faire.

Notez que l' `if` instruction est située à *l'intérieur* du composant de fonction, mais *avant* l' `return` instruction.

```
import React from 'react';

const fiftyFifty = Math.random() < 0.5;

// New function component starts here:
function TonightsPlan(){
  if(fiftyFifty === true){
    return <h1>Tonight I'm going out WOOO</h1>
  }
  if(fiftyFifty === false){
    return <h1>Tonight I'm going to bed WOOO</h1>
  }
}
```

```
}
```

```
export default TonightsPlan;
```

Écouteur d'événements et gestionnaires d'événements dans un composant  
6 minutes

Vos composants fonctionnels peuvent inclure des gestionnaires d'événements. Grâce à eux, nous pouvons exécuter du code en réponse à des interactions avec l'interface, comme un clic.

```
function MyComponent(){
  function handleHover() {
    alert('Stop it. Stop hovering.');
  }
  return <div onHover={handleHover}></div>;
}
```

Copy to Clipboard

Dans l'exemple ci-dessus, le gestionnaire d'événements est `handleHover()`. Il est transmis comme accessoire à l'élément JSX `<div>`. Nous aborderons les accessoires plus en détail dans une prochaine leçon, mais pour l'instant, il est important de comprendre qu'il s'agit d'informations transmises à une balise JSX.

La logique de ce qui doit se produire au `<div>` survol de l'élément est contenue dans la `handleHover()` fonction. La fonction est ensuite transmise à l'`<div>` élément.

Les fonctions de gestionnaire d'événements sont définies à l'intérieur du composant de fonction et, par convention, commencent par le mot « `handle` » suivi du type d'événement qu'il gère.

Il y a une petite particularité à laquelle il faut prêter attention. Revoyez cette ligne :

```
return <div onHover={handleHover}></div>
```

Copy to Clipboard

La `handleHover()`fonction est passée sans les parenthèses habituelles lors de l'appel d'une fonction. En effet, la passer avec la valeur « `as` » `handleHover`indique qu'elle ne doit être appelée qu'une fois l'événement survenu. Si elle est passée avec la valeur « `as` » `handleHover()`, la fonction sera immédiatement déclenchée ; soyez donc prudent !

Revoir

1 min

Félicitations ! Vous avez terminé la leçon sur les composants React.

Voici un bref récapitulatif des concepts introduits dans cette leçon :

- Les composants de fonction peuvent renvoyer plusieurs

### JSX

Aperçu : Documents Chargement de la description du lien

lignes en imbriquant les éléments dans un élément parent.

- Les attributs variables peuvent être utilisés à l'intérieur d'un composant React avec des injections JavaScript.
- Les composants React prennent en charge la logique en plaçant les instructions logiques au-dessus des instructions de retour.
- Les composants peuvent renvoyer conditionnellement des éléments JSX en plaçant des instructions conditionnelles à l'intérieur des composants.
- Les composants peuvent répondre aux événements en définissant des gestionnaires d'événements et en les transmettant aux éléments JSX.

Bravo pour ces sujets complexes ! Vous avez passé beaucoup de temps à étudier les composants React de manière isolée ! Il est maintenant temps d'apprendre comment les composants s'intègrent dans leur environnement.

### **Instructions**

Si vous souhaitez mettre en pratique les compétences que vous avez acquises, considérez ces défis :

- Ajoutez une logique au `MyQuote` composant pour produire différentes citations en fonction de différentes conditions.
- Ajoutez à l'écran des éléments capables de réagir à des événements tels qu'un clic ou un survol. Assurez-vous de définir des gestionnaires d'événements pour ces éléments dans le composant.

## Revue : React, partie I

Dans cette unité, vous avez découvert React.

Félicitations ! L'objectif de cette unité était de vous présenter la célèbre bibliothèque JavaScript React. React est un outil puissant qui vous permet de créer des interfaces web performantes et évolutives grâce à la création de composants. Vous avez également étudié ES6 et les concepts fonctionnels de JavaScript pour approfondir votre compréhension de React.

Après avoir terminé cette unité, vous êtes désormais capable de :

- Comprendre les concepts de programmation ES6+ et JavaScript fonctionnel
- Découvrez ce qu'est un DOM virtuel et comment il est utilisé dans React
- Comprendre JSX
- Créer des composants React

Si vous souhaitez en savoir plus sur ces sujets, voici quelques ressources supplémentaires :

- Documentation : [Documentation React : Premiers pas](#)
- Tutoriel : [Introduction à React](#)
- Article : [Penser en React](#)
- Article : [Qu'est-ce que le fork du DOM virtuel React ?](#)
- Ressource : [Awesome React](#)
- Ressource : [Créer une application React](#)

L'apprentissage est social. Quel que soit votre projet, n'hésitez pas à communiquer avec la communauté Codecademy sur les [forums](#). N'oubliez pas de consulter régulièrement la communauté, notamment en demandant des

révisions de code pour vos projets et en partageant vos révisions de code avec d'autres personnes dans la [catégorie « projets »](#), ce qui peut vous aider à consolider vos acquis.