**Javascript - Concepts à connaître avant d'apprendre React**

Avant de plonger dans React, il est essentiel de maîtriser certaines notions fondamentales de JavaScript, car React est une bibliothèque JavaScript. Nous allons réviser les principales notions à connaître.

[Template Literals (template strings) 2](#_5hzkii2jffhu)

[L'opérateur ternaire 3](#_1qs3kgs1j9d6)

[Noms de propriétés abrégés 4](#_djzw6wu1z8uk)

[Fonctions fléchées 5](#_vw6nrwtigghu)

[Spread operator 6](#_h318f9k1q2m2)

[Le destructuring 8](#_hy8x2th3sett)

[Import / Export 10](#_2d3y4at8qaki)

[Méthodes des tableaux 12](#_41shqgn80f28)

## Template Literals (template strings)

[MDN: Template Literals](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Template_literals)

Les template literals sont une amélioration des chaînes de caractères traditionnelles en JavaScript, C’est une fonctionnalité introduite dans ECMAScript 6 (ES6) offrant une manière plus propre et plus lisible d'incorporer des expressions dynamiques dans du texte.  
Ils sont couramment utilisés pour la génération de texte dynamique dans les applications web, notamment dans les frameworks comme React pour rendre les composants plus flexibles et interactifs.

Pour déclarer un template literal, entourez votre chaîne de caractères avec des backticks (` `), comme ceci :

| const name = "Alice"; const message = `Bonjour, ${prenom} !`; console.log(message); // Affiche : Bonjour, Alice ! |
| --- |

Exercice

[Create Strings using Template Literals](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/create-strings-using-template-literals)

## L'opérateur ternaire

[MDN: L'opérateur ternaire](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)

L'opérateur ternaire est un moyen de déclarer une condition de manière concise. Il est souvent utilisé pour évaluer une expression conditionnelle et retourner une valeur en fonction de cette condition. L'opérateur ternaire a la forme suivante :

| condition ? valeur\_si\_vrai : valeur\_si\_faux |
| --- |

La condition est évaluée en premier. Si la condition est vraie (true), l'opérateur ternaire renvoie la "valeur\_si\_vrai". Si la condition est fausse (false), il renvoie la "valeur\_si\_faux". Voici un exemple simple :

| const age = 20; const message = age >= 18 ? "Majeur" : "Mineur"; console.log(message); // Affiche : "Majeur" |
| --- |

Exercice

[Use the Conditional (Ternary) Operator](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/use-the-conditional-ternary-operator)

## Noms de propriétés abrégés

[Alsacreations : raccourcis de propriétés](https://www.alsacreations.com/astuce/lire/1899-Les-raccourcis-de-proprietes-property-shorthand-en-JavaScript.html)

Les nom de propriétés abrégés (“Shorthand property names”) en JavaScript simplifient la création d'objets en permettant de définir des propriétés avec le même nom que la variable qui stocke leur valeur, ce qui rend le code plus court et plus lisible. Cette fonctionnalité est largement utilisée pour améliorer la lisibilité du code JavaScript moderne.

Voici un exemple pour illustrer :

| // Sans utilisation des "Shorthand property names" const nom = "Alice"; const age = 30;  const personne = {  nom: nom,  age: age };  // Avec utilisation des "Shorthand property names" const nom = "Alice"; const age = 30;  const personne = {  nom,  age }; |
| --- |

Exercices

[Write Concise Object Literal Declarations Using Object Property Shorthand](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/write-concise-object-literal-declarations-using-object-property-shorthand)

[Write Concise Declarative Functions with ES6](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/write-concise-declarative-functions-with-es6)

## Fonctions fléchées

[MDN : Fonctions fléchées](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

Les fonctions fléchées (arrow functions) sont une syntaxe plus concise et plus simple pour définir des fonctions en JavaScript. Elles ont été introduites dans ECMAScript 6 (ES6) et sont devenues une caractéristique populaire du langage en raison de leur syntaxe plus légère.

Voici un exemple pour illustrer :

| // Fonction traditionnelle function additionner(a, b) {  return a + b; }  // Fonction fléchée const additionner = (a, b) => {  return a + b; };  // Encore plus concis const additionner = (a, b) => a + b; |
| --- |

Exercices

[Use Arrow Functions to Write Concise Anonymous Functions](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-arrow-functions-to-write-concise-anonymous-functions)

## Spread operator

[MDN: Spread syntax](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Spread_syntax)

[MDN: Rest parameters](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Functions/rest_parameters)

Le "spread operator" (ou opérateur de diffusion) est une fonctionnalité de JavaScript qui permet de récupérer les éléments d'un tableau ou les propriétés d'un objet pour les utiliser dans un nouvel emplacement. Cela peut être utile dans de nombreuses situations pour copier, concaténer, étendre ou simplifier les données.

Pour les tableaux

Vous pouvez utiliser le spread opérator pour créer une copie d'un tableau existant. Cela peut être utile pour éviter de modifier l'original accidentellement :

| const tableauOriginal = [1, 2, 3]; const copieTableau = [...tableauOriginal]; |
| --- |

Vous pouvez utiliser le spread opérator pour concaténer plusieurs tableaux en un seul :

| const tableau1 = [1, 2, 3]; const tableau2 = [4, 5]; const concatene = [...tableau1, ...tableau2]; // [1, 2, 3, 4, 5] |
| --- |

Vous pouvez ajouter des éléments à un tableau existant :

| const tableau = [1, 2, 3]; const nouveauTableau = [...tableau, 4, 5]; |
| --- |

Pour les objets

Comme pour les tableaux, vous pouvez utiliser l'opérateur de diffusion pour créer une copie d'un objet existant :

| const objetOriginal = { nom: 'Alice', age: 30 }; const copieObjet = { ...objetOriginal }; |
| --- |

Vous pouvez étendre un objet en lui ajoutant de nouvelles propriétés ou en remplaçant des propriétés existantes :

| const objetBase = { a: 1, b: 2 }; const objetEtendu = { ...objetBase, b: 3, c: 4 }; // objetEtendu devient { a: 1, b: 3, c: 4 } |
| --- |

Exercices

[Use the Spread Operator to Evaluate Arrays In-Place](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-the-spread-operator-to-evaluate-arrays-in-place)

[Use the Rest Parameter with Function Parameters](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-the-rest-parameter-with-function-parameters)

## Le destructuring

[MDN : Destructuring](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

Le destructuring en JavaScript est une technique qui vous permet d'extraire des informations d'objets ou de tableaux de manière simple et concise. Cela simplifie la manipulation des données en permettant aux développeurs d'extraire rapidement des valeurs et de les assigner à des variables.  
La déstructuration peut être utilisée avec des objets ou des tableaux.

Destructuration d'objets

Lors de la déstructuration d'objets, vous pouvez extraire des valeurs des propriétés d'un objet et les assigner à des variables portant le même nom que les propriétés :

| const personne = { nom: 'Alice', age: 30 };  // Extraction des valeurs des propriétés de l'objet personne const { nom, age } = personne;  console.log(nom); // Affiche : 'Alice' console.log(age); // Affiche : 30 |
| --- |

Destructuration de tableaux

Lors de la déstructuration de tableaux, vous pouvez extraire des éléments d'un tableau et les assigner à des variables en fonction de leur position dans le tableau :

| const fruits = ['pomme', 'banane', 'fraise'];  // Extraction des éléments du tableau fruits const [premier, deuxieme, troisieme] = fruits;  console.log(premiere); // Affiche : 'pomme' console.log(deuxieme); // Affiche : 'banane' console.log(troisieme); // Affiche : 'fraise' |
| --- |

Exercices

[Use Destructuring Assignment to Extract Values from Objects](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-to-extract-values-from-objects)

[Use Destructuring Assignment to Assign Variables from Objects](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-to-assign-variables-from-objects)

[Use Destructuring Assignment to Assign Variables from Nested Objects](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-to-assign-variables-from-nested-objects)

[Use Destructuring Assignment to Assign Variables from Arrays](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-to-assign-variables-from-arrays)

[Destructuring via rest elements](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-with-the-rest-parameter-to-reassign-array-elements)

[Use Destructuring Assignment to Pass an Object as a Function's Parameters](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-destructuring-assignment-to-pass-an-object-as-a-functions-parameters)

## Import / Export

[MDN : Import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import)  
[MDN : Export](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export)

Les instructions import et export sont utilisées dans JavaScript pour gérer l'importation et l'exportation de modules (fichiers JavaScript) réutilisables. Cette fonctionnalité permet de découper votre code en bout de code réutilisables.

L'instruction export est utilisée pour rendre des éléments (fonctions, variables, objets, etc.) d'un module disponibles pour d'autres modules qui souhaitent les utiliser.

Exportation nommée

Vous pouvez exporter des éléments individuels en leur attribuant un nom. Cela permet d'importer ces éléments spécifiquement dans d'autres modules :

| // Dans un fichier "module.js" export const maVariable = 42; export function maFonction() {  // ... }  // Dans un autre fichier import { maVariable, maFonction } from './monModule.js'; |
| --- |

Exportation par défaut

Vous pouvez également exporter un élément par défaut dans un module. Il ne peut y avoir qu'un seul élément exporté par défaut par module :

| // Dans un fichier "monModule.js" const valeurParDefaut = 42; export default valeurParDefaut;  // Dans un autre fichier import maValeurParDefaut from './monModule.js'; |
| --- |

Exercices

[Use export to Share a Code Block](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use-export-to-share-a-code-block)

[Reuse JavaScript Code Using import](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/reuse-javascript-code-using-import)

[Use \* to Import Everything from a File](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/use--to-import-everything-from-a-file)

[Create an Export Fallback with export default](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/create-an-export-fallback-with-export-default)

[Import a Default Export](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/es6/import-a-default-export)

## Méthodes des tableaux

[MDN : Array](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array)

Les méthodes sur les tableaux sont des fonctions intégrées à JavaScript qui peuvent être utilisées pour effectuer des opérations sur des tableaux (ou des listes) de données. Ces méthodes facilitent la manipulation, la transformation et la gestion de données stockées dans des tableaux. Voici quelques-unes des méthodes les plus couramment utilisées en React :

map()

Crée un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine :

| const tableauOriginal = [1, 2, 3, 4, 5];  // Utilisation de la méthode map() pour créer un nouveau tableau const tableauDouble = tableauOriginal.map((element) => element \* 2);  console.log(tableauDouble); // Affiche : [2, 4, 6, 8, 10] |
| --- |

filter()

Crée un nouveau tableau contenant uniquement les éléments du tableau d'origine qui satisfont une condition spécifiée.

| const chiffres = [1, 2, 3, 4, 5, 6];  // Utilisation de la méthode filter() pour créer un nouveau  // tableau contenant uniquement les nombres pairs const chiffresPairs = chiffres.filter((element) => element % 2 === 0);  console.log(chiffresPairs); // Affiche : [2, 4, 6] |
| --- |

find()

Recherche le premier élément du tableau qui satisfait une condition spécifiée et renvoie cet élément.

| const chiffres = [1, 3, 4, 5];  // Utilisation de la méthode find() pour trouver le premier chiffre  // pair const chiffrePair = chiffres.find((element) => element % 2 === 0);  console.log(chiffrePair); // Affiche : 4 |
| --- |

Exercices

[Use the map Method to Extract Data from an Array](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/functional-programming/use-the-map-method-to-extract-data-from-an-array)

[Use the filter Method to Extract Data from an Array](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/functional-programming/use-the-filter-method-to-extract-data-from-an-array)

[Use Higher-Order Functions map, filter, or reduce to Solve a Complex Problem](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/functional-programming/use-higher-order-functions-map-filter-or-reduce-to-solve-a-complex-problem)