

Les bonnes pratiques de la programmation avec JS

Table des matières

I. Contexte	3
II. Introduction aux bonnes pratiques en JavaScript	3
A. Les bonnes pratiques en JavaScript.....	3
B. Le guide de style Airbnb.....	4
C. Exercice : Quiz.....	6
III. Quelques bonnes pratiques en JavaScript	7
A. Quelques bonnes pratiques en JavaScript	7
B. Exercice : Quiz.....	10
IV. L'essentiel	11
V. Auto-évaluation	11
A. Exercice	11
B. Test.....	13
Solutions des exercices	13

I. Contexte

Durée : 1 heure

Environnement de travail : VS Code / Repl it

Contexte

L'écriture de code de qualité est essentielle pour garantir la fiabilité et les performances de vos applications en JavaScript. Les bonnes pratiques en JavaScript regroupent des règles et des conventions de codage qui peuvent vous aider à maintenir cette qualité élevée, à optimiser la performance et la sécurité, et à améliorer la maintenabilité de votre code.

Dans ce cours, nous allons passer en revue différentes bonnes pratiques recommandées pour le développement en JavaScript, ainsi que les avantages et les inconvénients de leur utilisation. Nous allons également explorer l'utilisation du guide de style Airbnb pour JavaScript, qui fournit des directives précises pour écrire du code clair, cohérent et facile à lire.

Ce cours vous permettra de dégrossir l'essentiel des règles de bonnes pratiques, mais gardez à l'esprit que ces bonnes pratiques peuvent évoluer, cela peut donc faire l'objet d'une veille au cours de votre vie professionnelle.

Attention

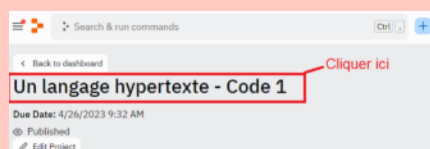
Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

- 1) Vous connecter à votre compte sur <https://replit.com/> (ou créer gratuitement votre compte)
- 2) Rejoindre la Team Code Studi du module via ce lien : <https://replit.com/teams/join/mmurvlgipIpxuasordloklIb bqskoim-programmer-avec-javascript>

Une fois ces étapes effectuées, nous vous conseillons de rafraîchir votre navigateur si le code ne s'affiche pas.

En cas de problème, redémarrez votre navigateur et vérifiez que vous avez bien accepté les cookies de connexion nécessaires avant de recommencer la procédure.

Pour accéder au code dans votre cours, cliquez sur le nom du lien Replit dans la fenêtre. Par exemple :



II. Introduction aux bonnes pratiques en JavaScript

A. Les bonnes pratiques en JavaScript

Définition

Les bonnes pratiques en JavaScript sont des conventions de codage qui permettent de maintenir la qualité du code, d'améliorer la lisibilité et la maintenabilité, et de réduire les erreurs et les bogues. Les avantages de suivre les bonnes pratiques sont nombreux, tels que :

- **Amélioration de la qualité du code** : les bonnes pratiques aident à écrire un code cohérent, organisé et facile à comprendre. Cela permet aux développeurs de travailler plus efficacement et de réduire le temps nécessaire pour déboguer le code.
- **Réduction des erreurs et des bogues** : les bonnes pratiques en JavaScript permettent de détecter les erreurs de code et les bogues plus rapidement, ce qui facilite leur correction. Cela permet également d'améliorer la fiabilité de l'application et d'augmenter la satisfaction des utilisateurs.

- **Facilitation de la collaboration** : les bonnes pratiques en JavaScript permettent aux membres de l'équipe de travailler plus efficacement et de collaborer plus facilement. En suivant les mêmes conventions de codage, les membres de l'équipe peuvent comprendre plus facilement le code écrit par les autres.

Cependant, il y a également quelques inconvénients à suivre les bonnes pratiques en JavaScript :

- **Temps supplémentaire nécessaire lors du développement** : le suivi des bonnes pratiques en JavaScript peut nécessiter un temps supplémentaire pour mettre en place les outils nécessaires et pour apprendre à les utiliser correctement. Cependant, ce temps supplémentaire est souvent compensé par une réduction des erreurs et des bogues.
- **Possibilité de surcharge cognitive** : si les bonnes pratiques sont trop complexes ou difficiles à comprendre, cela peut entraîner une surcharge cognitive pour les développeurs. Cela peut rendre le développement plus difficile et moins efficace. Il est donc important de mettre en place des bonnes pratiques qui correspondent à l'équipe, et qu'elles ne deviennent pas un fardeau pour les développeurs.

Définition Refactoring

Le *refactoring* (ou « *remaniement de code* » en français) est le processus consistant à modifier le code source d'un programme sans en changer le comportement externe, dans le but d'améliorer sa lisibilité, sa maintenabilité, sa qualité et/ou ses performances. Le refactoring ne consiste pas à ajouter de nouvelles fonctionnalités ou à corriger des bogues, mais plutôt à optimiser la structure du code existant.

Le refactoring peut inclure des tâches telles que la suppression de code inutile, le renommage de variables, la réorganisation de fonctions et de classes pour faciliter leur compréhension, la normalisation de la syntaxe, la simplification des expressions, l'extraction de méthodes et la réduction des dépendances entre les différentes parties du code.

B. Le guide de style Airbnb

Le guide de style Airbnb est un ensemble de règles et de recommandations pour écrire du code JavaScript clair, cohérent et facile à lire. Il est utilisé par de nombreuses entreprises et développeurs pour maintenir la qualité du code et améliorer la lisibilité et la maintenabilité. Il est, comme son nom l'indique, créé par Airbnb.

Il est publié sur GitHub sous la forme d'un référentiel open source (GitHub¹) et est conçu pour être utilisé avec un outil appelé ESLint, qui est un linter JavaScript qui peut identifier et signaler certaines erreurs dans votre code. Il est accessible aussi *via* ce lien : Airbnb JavaScript Style Guide².

En utilisant le guide de style Airbnb avec ESLint, les développeurs peuvent vérifier leur code en temps réel pour s'assurer qu'il respecte les normes de codage et les bonnes pratiques recommandées par Airbnb. Cela permet de maintenir la qualité du code, d'améliorer la lisibilité et la maintenabilité, et de réduire les erreurs et les bogues.

Le JavaScript style guide Airbnb couvre différents aspects : la déclaration de variables, la gestion des erreurs, la structure des fichiers, les fonctions, les classes, les chaînes de caractères, les tableaux, les objets, et bien plus encore. Le guide de style Airbnb fournit également des exemples de code pour illustrer les conventions de codage recommandées.

Le JavaScript style guide Airbnb est un outil très utile pour les développeurs JavaScript, car il fournit des normes de codage cohérentes et recommandées pour écrire du code clair, cohérent et facile à lire. Cela évite aux équipes de créer leurs propres conventions, qui seraient donc différentes d'une entreprise à l'autre.

¹ <https://github.com/airbnb/javascript>

² <https://airbnb.io/javascript/>

Ainsi, les bonnes pratiques en JavaScript sont essentielles pour maintenir la qualité du code, améliorer la lisibilité et la maintenabilité, et réduire les erreurs et les bogues. Le guide de style Airbnb est un outil utile pour maintenir la qualité du code en fournissant des règles et des conventions de codage claires et cohérentes. Bien que l'utilisation du guide de style Airbnb puisse avoir quelques inconvénients, les avantages l'emportent largement sur les inconvénients, en particulier pour les équipes de développement travaillant sur de grands projets.

Complément ESLint

ESLint est un outil d'analyse statique de code open source qui permet de détecter et de signaler les problèmes de code JavaScript, tels que les erreurs de syntaxe, les erreurs de style et les problèmes liés à la qualité du code. Le but principal de cet outil est d'aider les développeurs à écrire du code JavaScript propre, cohérent et sans erreurs.

ESLint utilise un système de règles configurables pour vérifier votre code et vous fournir des rapports sur les problèmes rencontrés. Vous pouvez configurer ces règles en fonction de vos préférences ou des normes de votre équipe de développement. En outre, vous pouvez étendre les fonctionnalités d'ESLint en utilisant des plugins ou en écrivant vos propres règles personnalisées.

L'intégration d'ESLint dans votre flux de travail de développement peut vous aider à maintenir la qualité du code, à détecter les erreurs potentielles avant l'exécution et à faciliter la collaboration entre les membres de l'équipe en favorisant un style de code cohérent.

Méthode Comment utiliser ESLint

Installez ESLint : pour installer ESLint, vous devez utiliser un gestionnaire de paquets tel que npm ou yarn (ESLint¹). Si vous utilisez npm, vous pouvez installer ESLint en exécutant la commande suivante dans votre terminal :

```
1 npm init @eslint/config
```

Exécutez ESLint : une fois que vous avez configuré ESLint, vous pouvez exécuter ESLint en utilisant la commande suivante dans votre terminal :

```
1 npx eslint yourfile.js
```

Où « *yourfile.js* » est le fichier que vous souhaitez analyser.

Vous pouvez également exécuter ESLint sur plusieurs fichiers en même temps en spécifiant des patterns de fichiers :

```
1 npx eslint "src/**/*.js"
```

Cette commande exécutera ESLint sur tous les fichiers .js dans le dossier src et ses sous-dossiers.

Exemple Un retour de ESLint

Si vous avez un fichier JS avec :

```
1 function test() {
2   var a = 1;
3   var _b = 2;
4   console.log(a)
5 }
6
7 test();
```

[cf.]

Lorsque vous exécutez ESLint sur ce fichier avec la configuration par défaut, vous obtenez le retour suivant :

```
1 2:7 error 'a' is assigned a value but never used      no-unused-vars
2 3:7 warning '_b' is defined but never used            no-unused-vars
```

¹ <https://eslint.org/docs/latest/use/getting-started>

Cela indique qu'il y a 2 erreurs, l'une pour la variable `a` qui est toujours inutilisée, et une alerte pour la variable `_b` qui est définie, mais jamais utilisée, en fonction de la configuration des règles.

Vous allez pouvoir ensuite configurer votre ESLint, pour vérifier ou non certaines erreurs. Par exemple, en ajoutant la règle `'no-var'`, ESLint avertira chaque fois que vous déclarez une variable avec le mot clé `var`, ce qui est une mauvaise pratique. Avec le même code JavaScript, nous aurions donc ce retour :

```
1 2:3 Unexpected var, use let or const instead. (no-var)
2 3:3 Unexpected var, use let or const instead. (no-var)
3 3:7 '_b' is assigned a value but never used. (no-unused-vars)
```

C. Exercice : Quiz

[solution n°1 p.15]

Question 1

Le refactoring est le processus consistant à améliorer la qualité du code sans changer son comportement externe.

- ☐ Vrai
- ☐ Faux

Question 2

Quel est l'avantage principal de l'utilisation de bonnes pratiques en JavaScript ?

- ☐ Améliorer la performance du code
- ☐ Faciliter la maintenance du code
- ☐ Rendre le code plus court

Question 3

ESLint est un outil qui permet de détecter les erreurs de syntaxe et les erreurs potentielles dans le code JavaScript.

- ☐ Vrai
- ☐ Faux

Question 4

Quel est le but principal du guide Airbnb de JavaScript ?

- ☐ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de l'entreprise Airbnb
- ☐ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de la communauté JavaScript
- ☐ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de la W3C

Question 5

Le guide Airbnb de JavaScript diminue les performances en travail de groupe, mais augmente les performances en travail individuel.

- ☐ Vrai
- ☐ Faux

III. Quelques bonnes pratiques en JavaScript

A. Quelques bonnes pratiques en JavaScript

Méthode Utiliser const et let au lieu de var

Il est recommandé d'utiliser `const` et `let` pour déclarer les variables en JavaScript, plutôt que `var`. Les variables déclarées avec `const` ne peuvent pas être réaffectées et les variables déclarées avec `let` peuvent être réaffectées. En revanche, les variables déclarées avec `var` ont une portée de fonction et peuvent être réaffectées. Voici un exemple :

```
1 // Bonne pratique
2 const PI = 3.14159;
3 let message = "Bonjour";
4
5 // Mauvaise pratique
6 var x = 10;
```

Méthode Utiliser des fonctions fléchées

Les fonctions fléchées sont une nouvelle syntaxe pour définir des fonctions en JavaScript. Elles ont l'avantage de rendre le code plus concis et plus facile à lire. Voici un exemple :

```
1 // Bonne pratique
2 const add = (a, b) => a + b;
3
4 // Mauvaise pratique
5 function add(a, b) {
6   return a + b;
7 }
```

Méthode Utiliser les templates strings

Les templates strings sont une nouvelle syntaxe pour créer des chaînes de caractères en JavaScript. Elles permettent d'incorporer des expressions JavaScript dans une chaîne de caractères. Voici un exemple :

```
1 // Bonne pratique
2 const name = "John";
3 const message = `Bonjour, ${name}!`;
4
5 // Mauvaise pratique
6 const name = "John";
7 const message = "Bonjour, " + name + "!";
```

Méthode Éviter les boucles for-in pour les tableaux

Il est recommandé d'éviter les boucles `for-in` pour parcourir les tableaux en JavaScript, car elles peuvent engendrer des erreurs. Il est préférable d'utiliser des boucles `for-of` ou `forEach`. Voici un exemple :

```
1 // Bonne pratique
2 const fruits = ["pomme", "banane", "orange"];
3
4 for (const fruit of fruits) {
5   console.log(fruit);
6 }
7
8 fruits.forEach(fruit => console.log(fruit));
```

[cf.]

```
1 // Mauvaise pratique
2 const fruits2 = ["pomme", "banane", "orange"];
3
4 for (const index in fruits2) {
5   console.log(fruits2[index]);
6 }
```

[cf.]

Méthode Éviter les fonctions constructeurs

Il est recommandé d'éviter les fonctions constructeurs pour créer des objets en JavaScript, car elles peuvent engendrer des erreurs. Il est préférable d'utiliser des classes ou des objets littéraux. Voici un exemple :

```
1 // Bonne pratique
2 class Person {
3   constructor(name) {
4     this.name = name;
5   }
6 }
7
8 const john = new Person("John");
9
10 const person = { name: "John" };

1 // Mauvaise pratique
2 function Person(name) {
3   this.name = name;
4 }
5
6 const john = new Person("John");
7
8 const person = new Object({ name: "John" });
```

Méthode Utiliser les méthodes Array

Les méthodes Array sont des méthodes intégrées qui permettent de manipuler les tableaux en JavaScript. Il est recommandé de les utiliser plutôt que de parcourir les tableaux avec des boucles. Voici des exemples de méthodes Array utiles :

```
1 // filter : permet de filtrer les éléments d'un tableau selon un critère
2 const numbers1 = [1, 2, 3, 4, 5];
3
4 const evenNumbers = numbers1.filter(number => number % 2 === 0)
5 console.log(evenNumbers); // [2, 4]
6
7 // map : permet de transformer chaque élément d'un tableau en un autre élément
8 const numbers2 = [1, 2, 3, 4, 5];
9
10 const doubledNumbers = numbers2.map(number => number * 2);
11 console.log(doubledNumbers); // [2, 4, 6, 8, 10]
12
13 // reduce : permet de réduire les éléments d'un tableau en une seule valeur
14 const numbers3 = [1, 2, 3, 4, 5];
15
16 const sum = numbers3.reduce((accumulator, number) => accumulator + number, 0);
17 console.log(sum); // 15
```

[cf.]

Méthode Éviter les boucles imbriquées

Il est recommandé d'éviter les boucles imbriquées en JavaScript, car elles peuvent ralentir les performances de l'application. Il est préférable d'utiliser des méthodes Array comme `map`, `filter` ou `reduce` pour éviter les boucles imbriquées. Voici un exemple :

```
1 // Bonne pratique
2 const matrix = [[1, 2], [3, 4], [5, 6]];
3
4 const flattenedMatrix = matrix.flatMap(row => row);
5 console.log(flattenedMatrix); // [1, 2, 3, 4, 5, 6]
```

[cf.]

```
1 // Mauvaise pratique
2 const matrix = [[1, 2], [3, 4], [5, 6]];
3 const flattenedMatrix = [];
4
5 for (let i = 0; i < matrix.length; i++) {
6   for (let j = 0; j < matrix[i].length; j++) {
7     flattenedMatrix.push(matrix[i][j]);
8   }
9 }
10
11 console.log(flattenedMatrix); // [1, 2, 3, 4, 5, 6]
```

[cf.]

Méthode Éviter les évaluations en cascade

Il est recommandé d'éviter les évaluations en cascade en JavaScript, car elles peuvent rendre le code difficile à lire et à maintenir. Il est préférable d'utiliser des méthodes Array ou des fonctions pour simplifier le code. Voici un exemple :

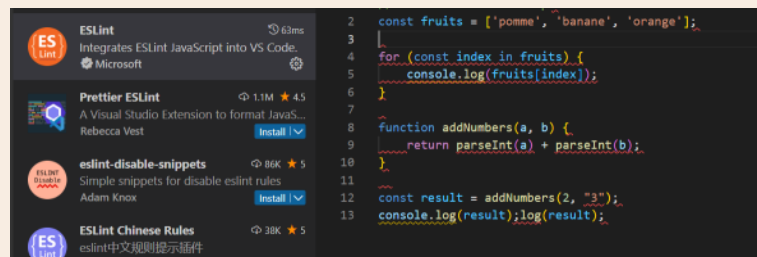
```
1 // Bonne pratique
2 const person = {
3   name: "John",
4   age: 30,
5   address: {
6     city: "Paris",
7     country: "France"
8   }
9 };
10
11 const country = person.address?.country;
12
13 // Mauvaise pratique
14 const person = {
15   name: "John",
16   age: 30,
17   address: {
18     city: "Paris",
19     country: "France"
20   }
21 };
22
23 const country = person.address && person.address.country;
```

Complément

Voici une liste plus exhaustive des bonnes pratiques en JavaScript : developer.mozilla.org¹.

Complément Les extensions

Beaucoup d'extensions existent pour nous signaler les erreurs directement depuis l'éditeur de code. Elles vous seront très pratiques, car elles vous éviteront de devoir taper les commandes ESLint en console. L'extension ESLint développée par Microsoft pourra par exemple vous aider sur VS Code, car elle soulignera le code présentant des erreurs en rouge.



B. Exercice : Quiz

[solution n°2 p.16]

Question 1

Il est recommandé d'utiliser des variables globales en JavaScript autant que possible.

- ☐ Vrai
- ☐ Faux

Question 2

Il est important d'utiliser des commentaires dans le code JavaScript pour expliquer le but et le fonctionnement du code.

- ☐ Vrai
- ☐ Faux

Question 3

Quelle est la meilleure façon de déclarer une variable en JavaScript ?

- ☐ `let variableName;`
- ☐ `var variableName;`
- ☐ `const variableName;`

Question 4

Les boucles imbriquées sont une bonne pratique.

- ☐ Vrai
- ☐ Faux

Question 5

¹ https://developer.mozilla.org/fr/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide/JavaScript

Il est recommandé d'utiliser des boucles `for-in` pour itérer sur les tableaux en JavaScript.

- ☐ Vrai
- ☐ Faux

IV. L'essentiel

Nous avons appris l'importance des bonnes pratiques de développement et du respect des normes de codage pour garantir un code robuste, fiable et facilement maintenable. Nous avons vu comment utiliser des outils tels que le guide de style JavaScript d'Airbnb pour mettre en place des conseils de bonnes pratiques et aider les développeurs à écrire un code cohérent et facilement compréhensible.

Nous avons appris comment utiliser des outils de linting tels que ESLint pour détecter les erreurs de syntaxe, les problèmes de style et les mauvaises pratiques de codage dans notre code. Nous avons vu comment configurer ces outils pour qu'ils respectent les normes de codage de notre entreprise et comment les intégrer dans notre flux de travail de développement.

En résumé, la mise en place de bonnes pratiques de développement et le respect des normes de codage sont des éléments clés pour garantir un code robuste et facilement maintenable. Les outils tels que le guide de style JavaScript d'Airbnb ou ESLint peuvent nous aider à atteindre ces objectifs en fournissant des conseils de bonnes pratiques, en détectant les erreurs de codage.

V. Auto-évaluation

A. Exercice

Vous êtes développeur dans une entreprise, et vous recevez cet e-mail de votre leader technique.

Objet : refactoring fichier JS pour se conformer aux bonnes pratiques

Bonjour,

J'espère que vous allez bien. Nous avons récemment identifié un fichier JavaScript dans notre codebase qui ne suit pas les bonnes pratiques de codage. Nous aimerions que vous corrigiez ce fichier en suivant les directives ci-dessous.

Le fichier en question présente plusieurs problèmes :

- Il utilise `var` pour déclarer des variables au lieu de `let` ou `const`,
- Il y a des boucles imbriquées,
- Il utilise une boucle `for-in` pour parcourir un tableau.

Question 1

[solution n°3 p.17]

Votre mission consiste à revoir le fichier et à apporter les modifications nécessaires pour le mettre en conformité avec nos standards de développement.

Voici le contenu du fichier HTML et du fichier JS concernés :

index.html :

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Exemple de page</title>
7 </head>
8 <body>
9   <h1>Exemple de page HTML</h1>
10  <ul id="result"></ul>
11
```

```
12 <script src="script.js"></script>
13 </body>
14 </html>
```

script.js :

```
1 var data = [
2   {id: 1, value: "Item 1"},
3   {id: 2, value: "Item 2"},
4   {id: 3, value: "Item 3"}
5 ];
6
7 var resultElement = document.getElementById("result");
8
9 function processItems() {
10   for (var key in data) {
11     var item = data[key];
12     var listItem = document.createElement("li");
13     listItem.innerHTML = "ID : " + item.id + ", Value : " + item.value;
14     resultElement.appendChild(listItem);
15
16     for (var i = 0; i < data.length; i++) {
17       if (data[i].id === item.id) {
18         data[i].processed = true;
19       }
20     }
21   }
22 }
23
24 function checkProcessedItems() {
25   var allProcessed = true;
26   for (var key in data) {
27     allProcessed = allProcessed && data[key].processed;
28   }
29   return allProcessed ? "All items processed" : "Some items not processed";
30 }
31
32 processItems();
33 console.log(checkProcessedItems());
```

N'hésitez pas à me contacter si vous avez des questions ou besoin de clarifications. Une fois les modifications effectuées, merci de soumettre vos changements pour revue.

Cordialement

Vous êtes développeur dans une entreprise de vente de vêtements et de chaussures en ligne et votre responsable vous confie une mission.

Question 2

[solution n°4 p.17]

Votre responsable vous annonce que l'un des stagiaires du département avait commencé à rédiger du code pour afficher une liste de produits et leur prix avant la fin de sa période de stage. Toutefois, après un rapide coup d'œil, vous vous apercevez que ce code ne respecte pas les bonnes pratiques de la programmation en JavaScript.

Votre mission va donc être de corriger le code suivant :

```
1 const products = [
2   { name: 'T-shirt', price: 20 },
3   { name: 'Jeans', price: 50 },
4   { name: 'Sneakers', price: 80 }
5 ];
6
```

```
7 for (const index in products) {  
8   console.log(products[index].name + ' - ' + products[index].price);  
9 }
```

B. Test

Exercice 1 : Quiz

[solution n°5 p.18]

Question 1

L'utilisation de bonnes pratiques en JavaScript peut réduire le temps de développement.

- ☐ Vrai
- ☐ Faux

Question 2

L'utilisation de bonnes pratiques en JavaScript peut améliorer la qualité du code.

- ☐ Vrai
- ☐ Faux

Question 3

Quel est le but principal d'ESLint ?

- ☐ Traduire le code JavaScript
- ☐ Faire de l'autocomplétion dans le code JavaScript
- ☐ Fournir des suggestions pour améliorer la qualité du code JavaScript

Question 4

Quel est l'avantage de l'utilisation de `let` et `const` plutôt que `var` en JavaScript ?

- ☐ Elle permet de définir des variables globale
- ☐ Elle permet de définir des variables locales
- ☐ Elle permet de mieux contrôler la portée des variables

Question 5

Quel est l'avantage de l'utilisation de la méthode `map` pour itérer sur les tableaux en JavaScript ?

- ☐ Elle permet de modifier les éléments du tableau
- ☐ Elle permet de filtrer les éléments du tableau
- ☐ Elle permet de créer un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine


Solutions des exercices

Exercice p. 6 Solution n°1**Question 1**

Le refactoring est le processus consistant à améliorer la qualité du code sans changer son comportement externe.

☒ Vrai

☐ Faux

 Le refactoring permet d'améliorer la qualité du code en le rendant plus lisible, plus maintenable et plus performant, sans changer son comportement externe.


Question 2

Quel est l'avantage principal de l'utilisation de bonnes pratiques en JavaScript ?

☐ Améliorer la performance du code

☒ Faciliter la maintenance du code

☐ Rendre le code plus court


 Les bonnes pratiques en JavaScript, telles que la modularisation du code, la gestion des erreurs et l'utilisation de noms de variables significatifs, permettent de rendre le code plus facile à comprendre et à modifier.

Question 3

ESLint est un outil qui permet de détecter les erreurs de syntaxe et les erreurs potentielles dans le code JavaScript.

☒ Vrai

☐ Faux

 ESLint est un outil de linting qui permet de détecter les erreurs de syntaxe et les erreurs potentielles dans le code JavaScript, en se basant sur un ensemble de règles définies par l'utilisateur.


Question 4

Quel est le but principal du guide Airbnb de JavaScript ?

☒ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de l'entreprise Airbnb

☐ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de la communauté JavaScript

☐ Fournir des directives pour écrire du code JavaScript conforme aux normes de codage de la W3C


 Le but principal du guide Airbnb de JavaScript est de fournir des directives pour écrire du code JavaScript conforme aux normes de codage de l'entreprise Airbnb. Le guide Airbnb est un ensemble de bonnes pratiques recommandées pour la programmation JavaScript, développé par l'entreprise Airbnb. Il est largement utilisé par la communauté JavaScript et fournit des directives claires pour écrire un code JavaScript clair, lisible, performant et maintenable.

Question 5

Le guide Airbnb de JavaScript diminue les performances en travail de groupe, mais augmente les performances en travail individuel.

☐ Vrai

☒ Faux

 Le guide Airbnb de JavaScript est justement très utile lors du travail de groupe, car il permet que tout le monde code avec les mêmes pratiques, et donc le code sera plus unifié.


Exercice p. 10 Solution n°2

Question 1

Il est recommandé d'utiliser des variables globales en JavaScript autant que possible.

☐ Vrai

☒ Faux


 Il est recommandé d'éviter autant que possible les variables globales en JavaScript, car cela peut entraîner des conflits et des bugs dans le code.

Question 2

Il est important d'utiliser des commentaires dans le code JavaScript pour expliquer le but et le fonctionnement du code.

☒ Vrai

☐ Faux

 Les commentaires aident les autres développeurs (et vous-même à l'avenir) à comprendre le but et le fonctionnement du code.


Question 3

Quelle est la meilleure façon de déclarer une variable en JavaScript ?

☒ `let variableName;`

☐ `var variableName;`

☒ `const variableName;`


 Il est recommandé d'utiliser `const` autant que possible, car cela garantit que la variable ne sera pas réaffectée accidentellement.

Question 4

Les boucles imbriquées sont une bonne pratique.

☐ Vrai

☒ Faux

 Les boucles imbriquées pourraient générer des pertes de performance. Il vaut mieux utiliser les méthodes de la classe Array.

Question 5

Il est recommandé d'utiliser des boucles `for-in` pour itérer sur les tableaux en JavaScript.

☐ Vrai

☒ Faux

Q Il est recommandé d'utiliser des boucles `for-of` pour itérer sur les tableaux en JavaScript, car cela garantit que chaque élément du tableau sera traité une fois et une seule. Les boucles `for-in` peuvent entraîner des bugs si des propriétés non numériques sont présentes dans le tableau.

p. 11 Solution n°3

- Remplacer `var` par `const` pour déclarer les variables immuables (`data`, `resultElement`).
- Remplacer la boucle `for-in` par la méthode `forEach()` pour parcourir le tableau `data`.
- Utiliser la méthode `every()` au lieu d'une boucle pour vérifier si tous les éléments du tableau ont été traités.

Code corrigé :

```
1 const data = [
2   {id: 1, value: "Item 1"},
3   {id: 2, value: "Item 2"},
4   {id: 3, value: "Item 3"}
5 ];
6
7 const resultElement = document.getElementById("result");
8
9 function processItems() {
10   data.forEach((item) => {
11     const listItem = document.createElement("li");
12     listItem.innerHTML = `ID : ${item.id}, Value : ${item.value}`;
13     resultElement.appendChild(listItem);
14
15     item.processed = true;
16   });
17 }
18
19 function checkProcessedItems() {
20   const allProcessed = data.every((item) => item.processed);
21   return allProcessed ? "All items processed" : "Some items not processed";
22 }
23
24 processItems();
25 console.log(checkProcessedItems());
```

[cf.]

p. 12 Solution n°4

Voici le résultat attendu :

```
1 const products = [
2   { name: 'T-shirt', price: 20 },
3   { name: 'Jeans', price: 50 },
4   { name: 'Sneakers', price: 80 }
5 ];
6
7 for (const product of products) {
8   console.log(`${product.name} - ${product.price}`);
9 }
```

[cf.]


Exercice p. 13 Solution n°5

Question 1

L'utilisation de bonnes pratiques en JavaScript peut réduire le temps de développement.

☒ Vrai

☐ Faux


 Les bonnes pratiques en JavaScript, telles que la modularisation du code et la gestion des erreurs, permettent de rendre le développement plus efficace en réduisant le temps passé à rechercher et à corriger les bugs.

Question 2

L'utilisation de bonnes pratiques en JavaScript peut améliorer la qualité du code.

☒ Vrai

☐ Faux

 Les bonnes pratiques en JavaScript, telles que l'utilisation de noms de variables significatifs et la séparation des responsabilités dans le code, permettent de rendre le code plus facile à comprendre et à maintenir, ce qui améliore la qualité du code.


Question 3

Quel est le but principal d'ESLint ?

☐ Traduire le code JavaScript

☐ Faire de l'autocomplétion dans le code JavaScript

☒ Fournir des suggestions pour améliorer la qualité du code JavaScript

 ESLint est un outil de linting qui permet de détecter les erreurs potentielles et de fournir des suggestions pour améliorer la qualité du code JavaScript.


Question 4

Quel est l'avantage de l'utilisation de `let` et `const` plutôt que `var` en JavaScript ?

☐ Elle permet de définir des variables globale

☐ Elle permet de définir des variables locales

☒ Elle permet de mieux contrôler la portée des variables

 L'utilisation de `let` et `const` en JavaScript permet de mieux contrôler la portée des variables et d'éviter les bugs liés à la réutilisation de variables.

Question 5

Quel est l'avantage de l'utilisation de la méthode `map` pour itérer sur les tableaux en JavaScript ?

☐ Elle permet de modifier les éléments du tableau

☐ Elle permet de filtrer les éléments du tableau

☒ Elle permet de créer un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine

Q La méthode `map` en JavaScript permet de créer un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine, ce qui est utile dans de nombreux cas.