INVESTIGATION DE FONCTIONNALITÉ

PROBLÉMATIQUE

Comparer deux formulations algorithimique **en Performance** entre les fonctions d'**API** vanilla et le constructeur **FOR** dans la recherche d'affichage

API VANILLA SUR ARRAY

Les méthodes associées à l'objet array sont spécialement développées pour itérer sur les éléments d'une liste. Elles permettent de modifier chaque élément de cette liste et de retourner une nouvelle liste, ou d'exécuter une opération spécifique sur chaque élément de la liste.

AVANTAGES	INCONVÉNIENTS
Plus facile à **maintenir dans un processus d'itération** de l'application	Transforme l'array et demande une bonne gymnastique intellectuelle
Facile à lire pour une compréhension du code fonctionnel	Peut être **difficile à appréhender sans connaitre toutes les subtilités** de fonctionnemenet de l'api dans les nuances
Ils correspondent mieux aux paradigmes de la programmation fonctionnelle	Connaitre l'option la plus adéquate `map()` `filter()` `slice` `splice` … Parfois les nuances ne sont pas évidentes 😢
Possibilité d'utiliser le chaînage de méthodes pour empiler les méthodes les unes après les autres	- -

BOUCLE FOR (ALGORHYMIE CLASSIQUE)

La classique boucle for, quand tu ne connais pas un langage tu fais du for (même si… il peut y avoir plusieurs for dans la forme pour moins d'effort (le mode boomer en cadeau pour la rime) Donc… l'avantage d'un for est que tu sais faire un for() et que tu sais quelle est résultat attendu sans surprise ce qui sera retourné, et ca c'est vraiment bien

AVANTAGES	INCONVÉNIENTS
La structure en **écrire est connue et le résultat est attendu** (Déjà dit mais bon pour les PO féniants je vais le remettre ici)	**Demande plus de ressources mémoires** (ça va tu gères pas des pointeurs en mémoire C, en JS c'est pas la fin du monde)
Facile d'ajouter ou de retirer des actions dans la boucle sans devoir refaire un traitement comme dans une API vanilla	L'écriture est plus couteuse et **la relecture est plus complexe car visuellement moins fonctionnel** et encore j'ai la coloration syntaxique

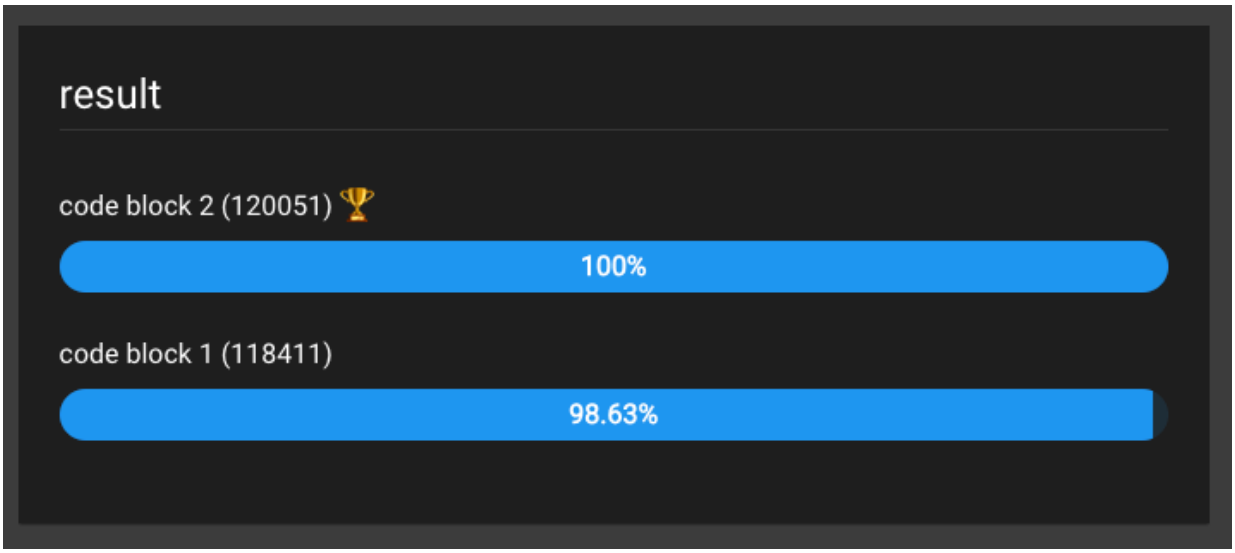
SOLUTION

Dans une situation où une boucle JavaScript native (Vanilla JS) exécute environ **50 itérations sur un fichier local**, l'impact sur la performance est généralement minime et se réduit principalement à la latence du processeur ou au moteur d'exécution JavaScript du

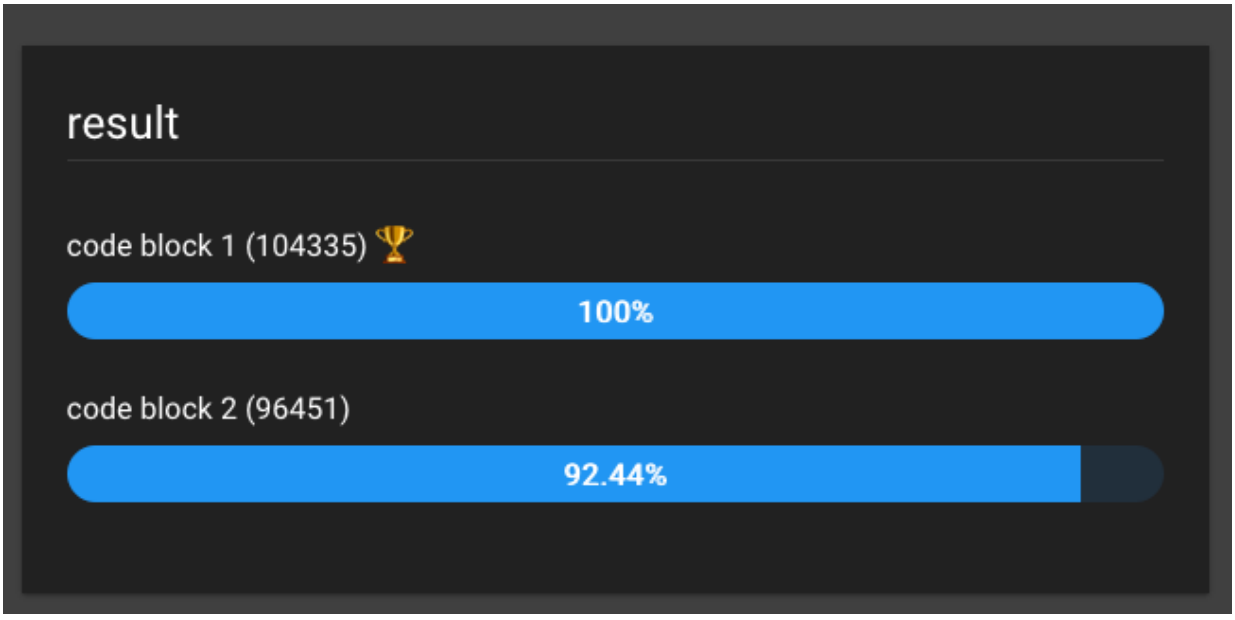
navigateur. Lorsque le code est exécuté par un navigateur, il est recommandé d'utiliser les **APIs Vanilla Array()** pour de **meilleures pratiques et performances**. Cependant, si la structure des données venait à évoluer, il serait judicieux de **reconsidérer cette approche dans un environnement backend** et de comparer les performances des environnements d'exécution tels que `BUN`, `DENO`, et `NODE`. Bien que BUN ait déjà capturé mon affection 😍, des tests comparatifs pourraient **révéler des différences cruciales** adaptées à des besoins spécifiques.

TEST DE PERFORMANCE

Test de performance pour le moteur Chromium (BRAVE)

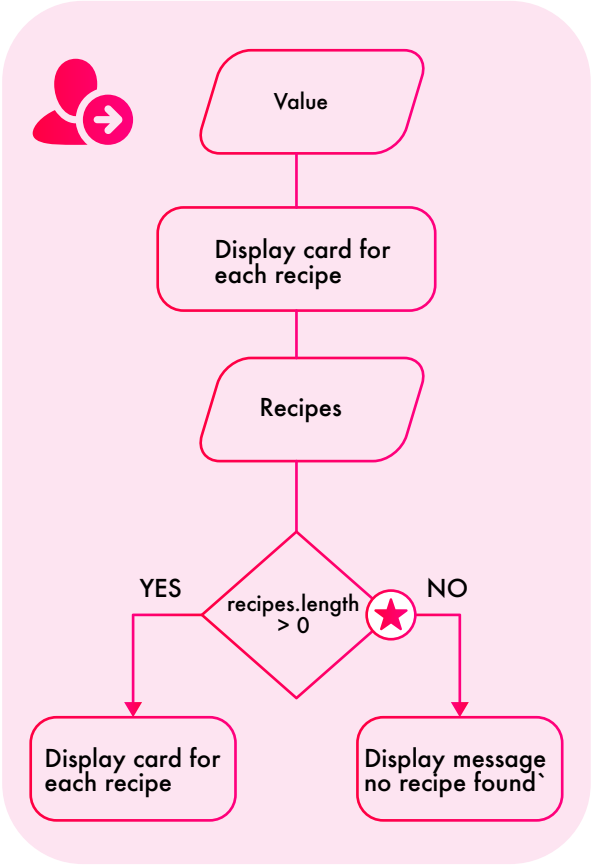


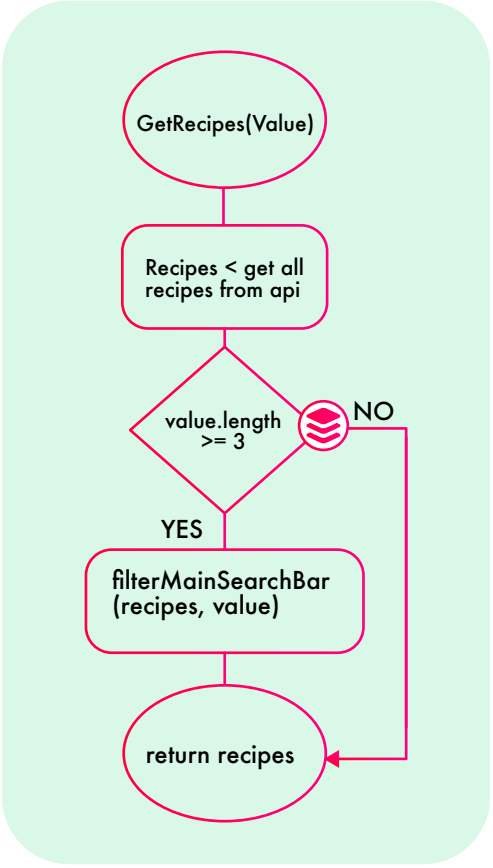
Test de performance pour le moteur Firefox (DEV EDITION)



LES PETITS PLATS

Fonctionnalité API ARRAY()





filterMainsearchBar(recipes, value) → Return recipes.filter(item => isLowerCaseIncluded(item.description, value) || isLowerCaseIncluded(item.name, value) || isFound(item.ingrediend, `ingredient`, value))

isLowerCaseIncluded(value1, value2) → Return value1.toLowerCase().includes(value2.toLowerCase())

isFound(array, property, value) → Return array.find(item => IsLowerCaseIncluded(item[property], value))

LES PETITS PLATS

Fonctionnalité FOR()