TD 3

Exercice 1 : Fonctions Pures

1 - Création d'une fonction pure simple

- Écrire une fonction pure qui prend un tableau de nombres et retourne un nouveau tableau avec chaque nombre doublé.

2 - Test de la pureté

- Tester la fonction avec différents ensembles de données.
- Identifier les caractéristiques qui font que la fonction est pure.

3 - Exemple d'une fonction impure

- Créer une fonction qui modifie une variable extérieure (effet de bord) et expliquer pourquoi elle est impure.

Points de discussion

- Pourquoi éviter les effets de bord ?
- Quels types de bugs ou de problèmes peuvent survenir dans un code non pur ?

Exercice 2 : Immutabilité

1 - Ajout d'un élément sans modification du tableau

- Écrire une fonction qui ajoute un élément à un tableau sans modifier le tableau d'origine.

2 - Comparaison avec une approche mutable

- Modifier directement un tableau en utilisant `.push()` et observer les changements sur le tableau d'origine.

3 - Modification immuable d'un objet

- Écrire une fonction qui modifie une propriété d'un objet sans altérer l'original.

Points de discussion

- Pourquoi l'immutabilité est-elle essentielle en programmation fonctionnelle ?
- Quels avantages cela présente-t-il pour la gestion des états dans des applications

complexes ?

Exercice 3 : Transparence Référentielle

1 - Transparence dans une fonction simple

- Créer une fonction qui calcule la somme de deux nombres.
- Remplacer les appels de fonction dans le code par les valeurs retournées et tester le comportement.

2 - Transparence dans des expressions complexes

- Introduire des fonctions imbriquées (somme et multiplication par exemple) et remplacer les appels par leurs valeurs.
- Tester si le programme reste identique.

3 - Absence de transparence référentielle

- Créer une fonction avec des effets de bord (ex: `Math.random()` ou modification d'une variable globale) et expliquer pourquoi elle n'est pas transparente.

Points de discussion

- Pourquoi la transparence référentielle est-elle cruciale pour rendre un code plus facile à lire et à raisonner ?
- Comment la transparence référentielle permet-elle d'optimiser ou de simplifier un code lors du développement ?

Exercice 4 : Fonctions d'ordre supérieur

1 - Application d'une fonction à chaque élément d'un tableau

- Écrire une fonction d’ordre supérieur qui prend une fonction et l’applique à chaque élément d’un tableau.

2 - Création de fonctions retournant des fonctions

- Écrire une fonction qui retourne une autre fonction (par exemple, une fonction qui crée un multiplicateur).

3 - Utilisation de fonctions d'ordre supérieur intégrées

- Expérimenter avec les méthodes comme `filter()`, `reduce()`, et `forEach()` pour transformer des tableaux.

Points de discussion

- En quoi les fonctions d’ordre supérieur facilitent-elles la modularité du code ?
- Comment ce concept peut-il être utilisé pour encapsuler des comportements et les réutiliser dans plusieurs contextes ?

Exercice 5 : Combinaison des principes

1 - Manipulation de tableaux d'objets

- Travailler sur un tableau d'objets représentant des produits avec un prix.
- Appliquer une réduction à chaque produit tout en conservant l'immutabilité.

2 - Calcul total avec `reduce`

- Calculer le prix total après réduction en utilisant `reduce`.

3 - Refactorisation pour pureté et réutilisabilité

- Réécrire le code en appliquant les principes vus, comme découper en plus petites fonctions pures.

Points de discussion

- Quel est l'impact de l'immuabilité dans cet exemple ?
- Pouvez-vous voir où la transparence référentielle intervient dans ce code ?
- Comment la combinaison de ces principes conduit-elle à un code plus fiable et facile à maintenir ?
- Quelles améliorations peuvent être apportées pour rendre le code plus fonctionnel ?