Exercices TypeScript

Travaux pratiques

TypeScript Exercice 1

En fonction des données utilisateur définies dans la constante `users`, définissez une interface `User` correspondante et corrigez le type de chaque déclaration et paramètre afin de ne plus utiliser de type ‘unknown’.

TypeScript Exercice 2

Dans cet exercice, nous définissons une nouvelle interface Admin. Nous avons ainsi une interface `User` et une interface `Admin`.

Définissez le type `Person` qui peut être soit un utilisateur commun de type `User` soit un administrateur de type `Admin`. Puis utilisez ce nouveau type dans votre programme afin de corriger les erreurs de compilation.

TypeScript Exercice 3

Dans cet exercice, nous améliorons notre fonction `logPerson`.
Cette fonction affiche maintenant la propriété `occupation` s'il s'agit d'un utilisateur ou la propriété `role` s'il s'agit d'un administrateur.

TypeScript Exercice 4

Afin de mieux gérer les utilisateurs et les administrateurs, nous définissons 2 fonctions `isAdmin` et `isUser` afin de les distinguer.

Corrigez les erreurs TypeScript de la fonction `logPerson` qui sont dûes à l'utilisation des 2 fonctions précédentes.

TypeScript Exercice 5

Pour cet exercice, nous avons défini une fonction `filterUsers` qui filtre les utilisateurs (que les utilisateurs pas les administrateurs) en fonction de critères spécifiques.

Modifiez la définition de la fonction `filterUsers` afin de passer en paramètre uniquement les critères de sélection souhaités.

Vous pouvez également exclure la propriété ‘type’ de la liste des critères possibles

TypeScript Exercice 6

On veut maintenant pouvoir filtrer tout type de personne (`User` et `Admin`).

Corrigez les problèmes de typage de la fonction `filterPersons` pour pouvoir filtrer toute personne et retournez un tableau de `User` s'il s'agit d'un utilisateur (personType='user') ou un tableau de `Admin` s'il s'agit d'un administrateur.

La fonction `filterPersons` doit accepter un type partiel User ou Admin en fonction de `personType`. Le paramètre `criteria` doit avoir un type en fonction du paramètre `personType`.

Le champ `type` ne doit pas être présent dans le paramètre `criteria`.

Vous pouvez créer une fonction `getObjectKeys()` qui retourne le résultat qui convient pour n'importe paramètre afin d'éviter d'effectuer un cast.

TypeScript Exercice 7

Dans cet exercice, on se propose de définir une fonction `swap` qui prend 2 paramètres en entrée de n'importe quel type et qui retourne un tuple de ces paramètres dans l'ordre inverse.
Cette fonction doit accepter n'importe quel type.

TypeScript Exercice 8

Définissez un type PowerUser qui devrait posséder tous les champs de l'interface User et de l'interface Admin (à l'exception du champ `type`).
Le type PowerUser devrait également avoir son propre champ type de valeur `powerUser`.

TypeScript Exercice 9

Il a été décidé de créer 2 types de réponse pour notre API.

`UsersApiResponse` qui doit contenir la réponse de l'API lorsque l'on gère des utilisateurs.
`AdminsApiResponse` qui doit contenir la réponse de l'API lorsque l'on gère des administrateurs.

Malheureusement ce choix nécessite la création de plusieurs types.

Dans cet exercice, on se propose de supprimer les 2 types créés précédemment et de les remplacer par un type générique `ApiResponse`.

TypeScript (autres exercices)

Additionner les valeurs d'un arbre

Ecrire une fonction TypeScript qui additionne les valeurs d'un arbre

L'objet aura 2 propriétés gauche et droite de type number ou null ou un objet.

Relation Eleves Professeur

Créer une sorte de base de données élève, classe, professeur et coder quelques fonctions.

Union Type

créer une fonction qui va vous retourner la

valeur d'un nombre si le paramètre dans la fonction est un nombre unique ou la somme

d'un tableau de nombre si le paramètre passé dans la fonction est un tableau de nombre