EXERCICES SUR LES PROMESSES

Exercice 1 : Création d'une promesse simple

Objectif : L'objectif de cet exercice est de vous familiariser avec la création de promesses en JavaScript. Vous allez créer une promesse qui se résout ou se rejette après un certain délai.

Instructions :

1. **Création de la promesse :**
- Commencez par créer une nouvelle promesse en utilisant le constructeur Promise.
- La promesse doit prendre une fonction comme argument. Cette fonction aura deux paramètres : resolve et reject.
- À l'intérieur de cette fonction, utilisez la fonction setTimeout pour définir un délai de 2 secondes.

2. **Résolution ou rejet de la promesse :**
- À l'intérieur du setTimeout, utilisez une condition pour décider si la promesse doit être résolue ou rejetée.
- Par exemple, vous pouvez utiliser Math.random() pour générer un nombre aléatoire entre 0 et 1. Si ce nombre est supérieur à 0,5, résolvez la promesse avec le message "Promesse résolue !". Sinon, rejetez la promesse avec le message "Promesse rejetée !".

Conseils :
- La fonction Math.random() génère un nombre aléatoire entre 0 (inclus) et 1 (exclus).
- La fonction setTimeout permet d'exécuter une fonction après un certain délai, spécifié en millisecondes.

Résultat attendu :
Lorsque vous exécutez votre code, vous devriez voir s'afficher l'état de la promesse et le résultat après 2 secondes, en fonction du résultat aléatoire.

Exercice 2 : Utilisation de then et catch

Utilisez la promesse créée dans l'exercice 1. Ajoutez des gestionnaires then et catch pour gérer la résolution et le rejet de la promesse.

Résultat attendu :
Lorsque vous exécutez votre code, vous devriez voir s'afficher "Promesse résolue !" ou "Promesse rejetée !" après 2 secondes, en fonction du résultat aléatoire.

Exercice 3 : Promesse avec des données utilisateur

Objectif : L'objectif de cet exercice est de vous familiariser avec la création de promesses qui traitent des données fournies par l'utilisateur. Vous allez créer une promesse qui vérifie l'âge d'un utilisateur et détermine s'il est majeur ou mineur.
Contexte : Imaginez que vous développez une application qui nécessite que les utilisateurs soient majeurs pour accéder à certaines fonctionnalités. Avant de donner accès à ces fonctionnalités, vous devez vérifier l'âge de l'utilisateur.

Instructions :

1. **Préparation :**
- Créez une fonction verifierUtilisateur qui prend en argument un objet utilisateur.
- Cet objet utilisateur devrait avoir au moins deux propriétés : nom et age.

2. **Création de la promesse :**
- À l'intérieur de la fonction, créez une promesse.
- Si l'âge de l'utilisateur est supérieur ou égal à 18 ans, la promesse doit être résolue avec le message "Utilisateur majeur".
- Si l'âge de l'utilisateur est inférieur à 18 ans, la promesse doit être rejetée avec le message "Utilisateur mineur".

3. **Testez votre fonction :**
- Appelez la fonction verifierUtilisateur avec un objet utilisateur ayant un âge supérieur ou égal à 18 ans et affichez le résultat.
- Appelez à nouveau la fonction avec un objet utilisateur ayant un âge inférieur à 18 ans et affichez le résultat.

Conseils :
- Utilisez les méthodes then et catch pour gérer la résolution et le rejet de la promesse lors des tests.
- Assurez-vous de bien comprendre la différence entre la résolution et le rejet d'une promesse.

Résultat attendu :
- Lorsque vous testez avec un utilisateur majeur, vous devriez voir le message "Utilisateur majeur" s'afficher.
- Lorsque vous testez avec un utilisateur mineur, vous devriez voir le message "Utilisateur mineur" s'afficher.

Exercice 4 : Promesse avec une opération mathématique

Objectif : L'objectif de cet exercice est de vous familiariser avec la création de promesses qui traitent des opérations mathématiques. Vous allez créer une promesse qui effectue une division et gère le cas où le dénominateur est zéro.
Contexte : Imaginez que vous développez une calculatrice en ligne. L'une des fonctionnalités de cette calculatrice est de diviser deux nombres. Cependant, comme la division par zéro n'est pas définie en mathématiques, vous devez gérer ce cas particulier.

Instructions :

1. **Préparation :**

Créez une fonction diviser qui prend en argument deux nombres : a (le numérateur) et b (le dénominateur).

2. **Création de la promesse :**

- À l'intérieur de la fonction, créez une promesse.
- Si le dénominateur b est différent de zéro, la promesse doit être résolue avec le résultat de la division de a par b.
- Si le dénominateur b est égal à zéro, la promesse doit être rejetée avec le message "Division par zéro !".

3. **Testez votre fonction :**

- Appelez la fonction diviser avec deux nombres de votre choix où le dénominateur n'est pas zéro et affichez le résultat.
- Appelez à nouveau la fonction diviser avec un dénominateur égal à zéro et affichez le résultat.

Conseils :

- Utilisez les méthodes then et catch pour gérer la résolution et le rejet de la promesse lors des tests.
- Assurez-vous de bien comprendre pourquoi la division par zéro est un problème et comment la promesse peut aider à gérer ce cas.

Résultat attendu :

- Lorsque vous testez avec un dénominateur différent de zéro, vous devriez voir le résultat de la division s'afficher.
- Lorsque vous testez avec un dénominateur égal à zéro, vous devriez voir le message "Division par zéro !" s'afficher.

Exercice 5 : Promesse avec une simulation d'appel API

Objectif : L'objectif de cet exercice est de vous familiariser avec la création de promesses qui simulent des appels API. Vous allez créer une promesse qui simule la récupération d'informations sur un utilisateur depuis une API externe.

Contexte : Imaginez que vous développez une application web qui nécessite de récupérer des informations sur un utilisateur depuis une base de données distante via une API. Cependant, les appels API peuvent échouer pour diverses raisons (problèmes de réseau, serveur distant en panne, etc.). Vous devez donc gérer ces éventuelles erreurs.

Instructions :

1. **Préparation :**

Créez une fonction appelAPI qui ne prend aucun argument.

2. **Création de la promesse :**

- À l'intérieur de la fonction, créez une promesse.
- Utilisez la fonction setTimeout pour simuler un délai de 3 secondes, représentant le temps que prendrait un véritable appel API.
- Après ce délai, utilisez Math.random() pour simuler une chance de succès de l'appel API.
- Si l'appel est "réussi" (par exemple, si Math.random() renvoie un nombre supérieur à 0,5), résolvez la promesse avec un objet contenant des informations sur un utilisateur fictif, comme { nom: "Alice", age: 25 }.
- Si l'appel échoue (par exemple, si Math.random() renvoie un nombre inférieur ou égal à 0,5), rejetez la promesse avec le message "Erreur lors de l'appel API".

3. **Testez votre fonction :**

- Appelez la fonction appelAPI et utilisez les méthodes then et catch pour gérer la résolution et le rejet de la promesse.
- Affichez les informations de l'utilisateur si l'appel est réussi, ou le message d'erreur si l'appel échoue.

Résultat attendu :
Lorsque vous exécutez votre code, vous devriez voir s'afficher les informations de l'utilisateur ou le message d'erreur après 3 secondes, en fonction du résultat aléatoire.