Questions d'Évaluation

I. Compréhension Technique

1. Gestion des limites d'exposition

Mon smart contract gère les limites d'exposition en comparant systématiquement l'exposition nette (différence entre les expositions longues et courtes) avec la limite définie pour chaque contrepartie. En cas de dépassement de cette limite, un événement `LimiteDepassee` est émis, ce qui permet de notifier immédiatement les parties prenantes.

Mesures de sécurité :

- Validation des entrées pour éviter les valeurs négatives ou incohérentes.
- Vérification systématique des limites dans la fonction `mettreAJourExposition`.
- Désactivation des contreparties à risque élevé pour prévenir les abus.

2. Calcul des risques

Mon modèle de calcul du risque intègre :

- **Scores de crédit :** Utilisés comme multiplicateurs dans le calcul du score de risque pour refléter la solvabilité de la contrepartie.
- **Exposition courante :** Base principale pour évaluer le risque, calculée à partir des expositions longues et courtes.
- **Historique des transactions :** Bien que non explicitement stocké, les événements émis (`ExpositionMiseAJour`, `LimiteDepassee`) permettent de suivre les modifications des expositions pour évaluer les tendances.
  Le risque est calculé via la formule :

$$\text{Score de Risque} = \frac{\text{Exposition Courante}}{\text{Limite d'Exposition}} \times \frac{100}{\text{Score de Crédit}}$$

3. Efficacité en termes de gas

J'ai optimisé l'utilisation du gas en :

- Minimisant les écritures dans la blockchain (par exemple, en mettant à jour uniquement les champs nécessaires).
- Utilisant des structures comme `mapping` pour un accès direct aux données.
- Évitant des calculs inutiles en externalisant certains calculs complexes au frontend.
  Résultat : Les fonctions courantes consomment un gas limité, ce qui les rend adaptées aux environnements Ethereum avec des frais élevés.

5. Gestion des cas limites

- **Congestion du réseau :** Les utilisateurs peuvent retenter les transactions en cas de retard. L'interface Streamlit affiche des messages d'erreur clairs.
- **Transactions échouées :** Les transactions invalides n'altèrent pas l'état du contrat. Les erreurs sont capturées côté frontend.
- **Entrées invalides :** Toutes les entrées (adresses, montants) sont validées avant d'appeler une fonction du contrat intelligent, empêchant ainsi des abus ou des erreurs.

6. Stratégie de test et résultats

J'ai adopté une stratégie de test en trois étapes :

- **Tests unitaires :** Chaque fonction a été testée individuellement dans Remix et Ganache. Exemple : Ajout et mise à jour des contreparties.
- **Tests d'intégration :** Vérification des interactions entre Streamlit et le contrat intelligent via Web3.py.
- **Validation manuelle :** Contrôle des événements générés et vérification de la cohérence des données dans MetaMask.
  Résultat : Tous les scénarios principaux ont fonctionné correctement, y compris les cas limites.

II. Compréhension de la Gestion des Risques

1. Comparaison avec la gestion traditionnelle

- **Blockchain :**
  - **Avantages :** Transparence accrue, immutabilité des données, automatisation via des smart contracts.
  - **Limitations :** Frais de transaction, dépendance aux réseaux Ethereum, et complexité technique.
- **Gestion traditionnelle :**
  - **Avantages :** Plus de flexibilité dans les processus, support des systèmes existants.
  - **Limitations :** Moins de transparence, dépendance aux audits humains, risque d'erreur ou de manipulation.

2. Scénarios de risque

- **Augmentation soudaine de l'exposition :** Le contrat émet un événement d'alerte (`LimiteDepassee`) pour notifier ce dépassement.
- **Détérioration du score de crédit :** La mise à jour du score de crédit impacte immédiatement le calcul des risques.
- **Transactions multiples simultanées :** Ethereum gère les transactions via un mécanisme FIFO, garantissant que l'état du contrat reste cohérent.

3. Améliorations potentielles

- Ajouter un historique des transactions au niveau du contrat pour un suivi détaillé.
- Intégrer des modèles prédictifs pour anticiper les risques basés sur des tendances passées.

- Prendre en compte des facteurs externes comme les fluctuations du marché pour ajuster dynamiquement les limites d'exposition.

III.Implémentation et Innovation

1. Fonctionnalités supplémentaires

- **Émissions d'événements :** Les événements comme `LimiteDepassee` et `ExpositionMiseAJour` améliorent la traçabilité.
- **Désactivation des contreparties :** Cette fonctionnalité prévient les risques liés à des contreparties inactives ou non fiables.
- **Calcul du ratio de couverture :** Utile pour évaluer la capacité d'une contrepartie à couvrir son exposition avec son collatéral.
  Valeur métier : Ces fonctionnalités renforcent la sécurité et l'efficacité du système pour des applications financières réelles.

2. Applications potentielles

- **Dans le monde réel :** Ce système peut être utilisé par les institutions financières pour gérer les risques de crédit en temps réel.
- **Modifications nécessaires :** Passage à un réseau public sécurisé (ex. Ethereum Mainnet), audit de sécurité approfondi, et ajout de fonctionnalités comme l'intégration avec des outils KYC.

3. Réflexion sur le développement

- **Défis rencontrés :**
  - Gestion des frais de gas élevés.
  - Problèmes de compatibilité entre Web3.py et certaines versions de Python.
- **Solutions apportées :**
  - Optimisation des fonctions Solidity pour réduire le gas.
  - Utilisation de versions stables et testées des outils.
- **Ce que je ferais différemment :**
  - Planifier des tests approfondis dès le début pour éviter des ajustements tardifs.
  - Documenter chaque étape de développement pour accélérer l'intégration avec des équipes externes.