Blockchain & Cryptomonnaies

Sujet d'Examen : Création d'un Prototype de Blockchain Simplifiée

Rapport Technique

I. Explication de chaque étape avec les principes techniques et les tests effectués.

Étape 1: Création d'un Bloc

Ce rapport décrit la mise en œuvre d'une classe Block, conçue pour représenter les blocs individuels au sein d'une blockchain. Chaque bloc contient des données telles que les transactions associées, le hachage du bloc précédent, ainsi qu'un hachage unique servant de preuve de travail.

1. Description de la Classe Block

1.1 Attributs de la classe

→ index : un identifiant unique du bloc.
→ transactions : la liste des transactions associées au bloc.
→ timestamp : l'horodatage de la création du bloc.
→ previous_hash : le hachage du bloc précédent, garantissant la continuité dans la chaîne.
→ nonce : une valeur utilisée pour résoudre le problème de preuve de travail.
→ hash : le hachage propre au bloc, calculé en fonction de son contenu.

1.2 Méthodes implémentées

→ __init__ : Initialise les attributs du bloc.
→ generate_hash() : Génère un hachage unique basé sur le contenu du bloc.

2. Étapes de Mise en Œuvre

2.1 Tests effectués

- Création d'un bloc avec des transactions fictives.
- Vérification de la cohérence des hachages après mise à jour des valeurs.
- Validation du mécanisme de preuve de travail.

2.2 Résultats obtenus

- Un bloc valide est généré.

- La preuve de travail est correctement résolue, assurant que le hachage commence par le nombre correct de zéros.

Étape 2 : Modélisation d'une Transaction

Ce rapport détaille l'implémentation et les tests d'une classe Transaction, utilisée pour représenter les échanges entre deux parties dans une blockchain. Une transaction inclut les informations de l'expéditeur, du destinataire, et le montant échangé. Des mécanismes de sérialisation et de validation sont également intégrés.

1. Description de la Classe Transaction

1.1 Attributs de la classe

→ exp_id : Identifiant unique de l'expéditeur.
→ dest_id : Identifiant unique du destinataire.
→ montant : Montant transféré dans la transaction.

1.2 Méthodes de la classe

→ to_json() : Sérialise la transaction au format JSON pour faciliter son stockage ou sa transmission.
→ is_valid() : Vérifie l'intégrité de la transaction en contrôlant :

  - Que tous les champs sont non nuls.
  - Que l'identifiant de l'expéditeur et du destinataire ne sont pas identiques.
  - Que le montant est supérieur à zéro.

2. Étapes de Mise en Œuvre

2.1 Tests effectués

- Création et Sérialisation de Transactions
- Tests sur des Transactions Invalides

2.2 Résultats Observés

Test	Description	Résultat attendu	Résultat observé
Transaction valide	Expéditeur : Alice Destinataire : Bob Montant : 100	Transaction valide	Transaction valide
Transaction avec données identiques	Expéditeur : Nazir Destinataire : Nazir	Non valide, identifiants égaux	Non valide, identifiants égaux
Transaction avec données manquantes	Expéditeur : Ilham Destinataire : None	Non valide, données manquantes	Non valide, données manquantes
Transaction avec montant nul	Expéditeur : Nazir Destinataire : Ilham Montant : 0	Non valide, montant nul	Non valide, montant nul

Étape 3 : Preuve de Travail (Proof of Work)

Le mécanisme de Preuve de Travail (Proof of Work) garantit la sécurité et l'intégrité des blocs dans une blockchain. La méthode implémentée utilise un nonce (nombre arbitraire) pour résoudre un problème de hachage, où le hachage final du bloc doit commencer par un certain nombre de zéros.

1.Description Technique du mécanisme Proof of Work

1.1 Objectif : résoudre le problème de hachage

- Le hachage du bloc doit respecter un modèle spécifique (commençant par un certain nombre de zéros).
- Un nonce est incrémenté pour générer de nouveaux hachages jusqu'à ce que la condition soit satisfaite.

1.2 Méthode

→ Calcul du hachage du bloc en combinant :
  - L'identifiant du bloc.
  - L'horodatage.
  - Les transactions.
  - Le hachage du bloc précédent.
  - La valeur actuelle du nonce.

→ Vérification : le hachage commence par le nombre requis de zéros.

- Si la condition n'est pas respectée :
    - Incrémentation du nonce et recalcul du hachage.
    - Répéter jusqu'à trouver une solution valide.

2. Étapes de Mise en Œuvre

2.1 Tests effectués

- Implémentation dans la classe Block

2.2 Résultats obtenus

- Hachage initial : Affiche un hachage non conforme (n'importe quelle valeur hexadécimale).
- Preuve de Travail terminée
- Le hachage obtenu commence par 4 zéros (0000...).
- Un nonce unique est trouvé.

Étape 4 : Création de la Chaîne de Blocs

Ce rapport décrit la mise en œuvre de la classe Blockchain, conçue pour l'initialisation d'une chaîne contenant un bloc de genèse (premier bloc), l'ajout de nouveaux blocs à la chaîne tout en validant leur intégrité, et l'affichage de l'ensemble de la chaîne dans un format lisible.

1. Description de la Classe Blockchain

1.1 Méthodes de la classe

→ __init__(self) : Initialise la blockchain avec un bloc de genèse.
→ create_genesis_block() : crée le bloc de genèse (le 1er bloc de la chaîne).
→ add_block() : ajoute un nouveau bloc à la chaîne après vérification de son intégrité.
→ is_valid_new_block() : vérifie si un nouveau bloc est valide par rapport au bloc précédent.
→ is_ chain _valid() : vérifie si une chaîne donnée est valide.

2. Étapes de Mise en Œuvre

2.1 Tests effectués

- Création de la classe Blockchain
- Création du bloc de genèse
- Validation de l'intégrité (vérification du lien entre les blocs)
- Affichage clair et lisible de la chaîne complète.

2.2 Résultats obtenus

→ Affichage de données de chaque bloc contenant les valeurs des attributs suivants :

- Index
- Horodatage
- Transactions
- Nonce
- Hachage
- Hachage précédent

Étape 5 : Calcul des Hashes

Ce rapport décrit l'utilisation la bibliothèque hashlib de Python pour créer un hachage basé sur le contenu complet du bloc comprenant : l'index, le timestamp, les transactions, le hash du bloc précédent, et le nonce, pour calculer un hachage basé sur l'intégralité du bloc, et vérifier que les hachages générés sont corrects.

1. Tests Effectués

Les tests garantissent que le calcul du hachage est correct, que le hachage ne se répète pas lorsque le contenu du bloc change, et qu'il est basé sur le contenu complet.

Étape 6 : Simulation d'un Réseau Décentralisé

Ce rapport décrit l'utilisation de la méthode resolve_conflicts pour corriger les différences entre des chaînes concurrentes en adoptant la plus longue.

1. Implémentation du Consensus

→ Règle de la Chaîne la Plus Longue

La règle stipule que, dans un réseau décentralisé, la chaîne valide avec le plus grand nombre de blocs est celle qui est adoptée comme référence.

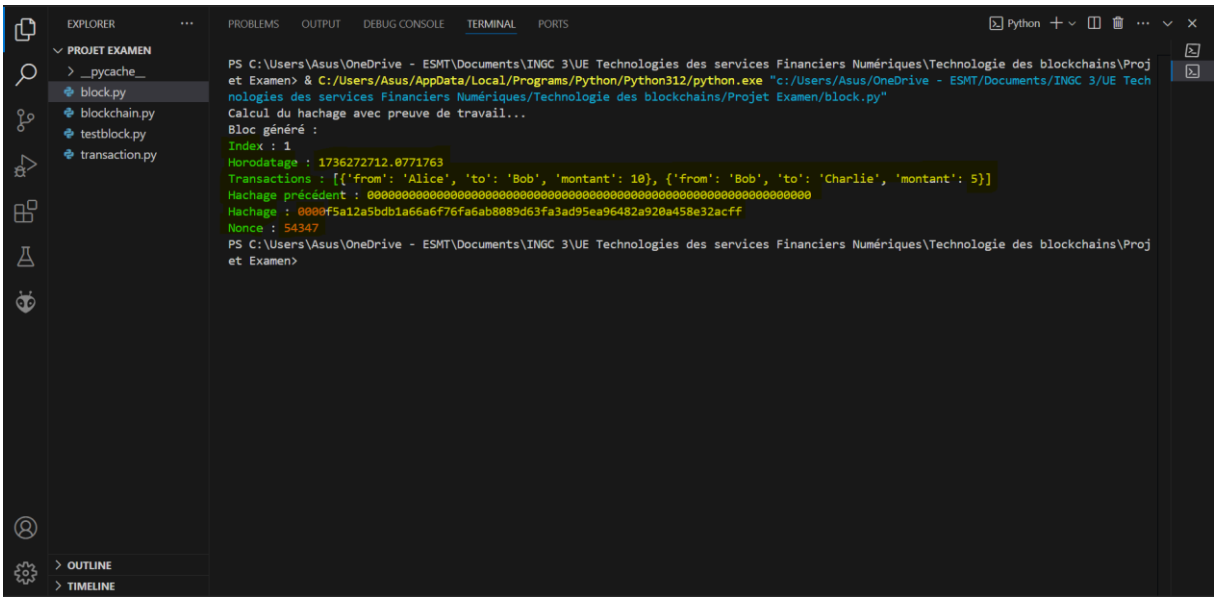
→ Méthode resolve_conflicts

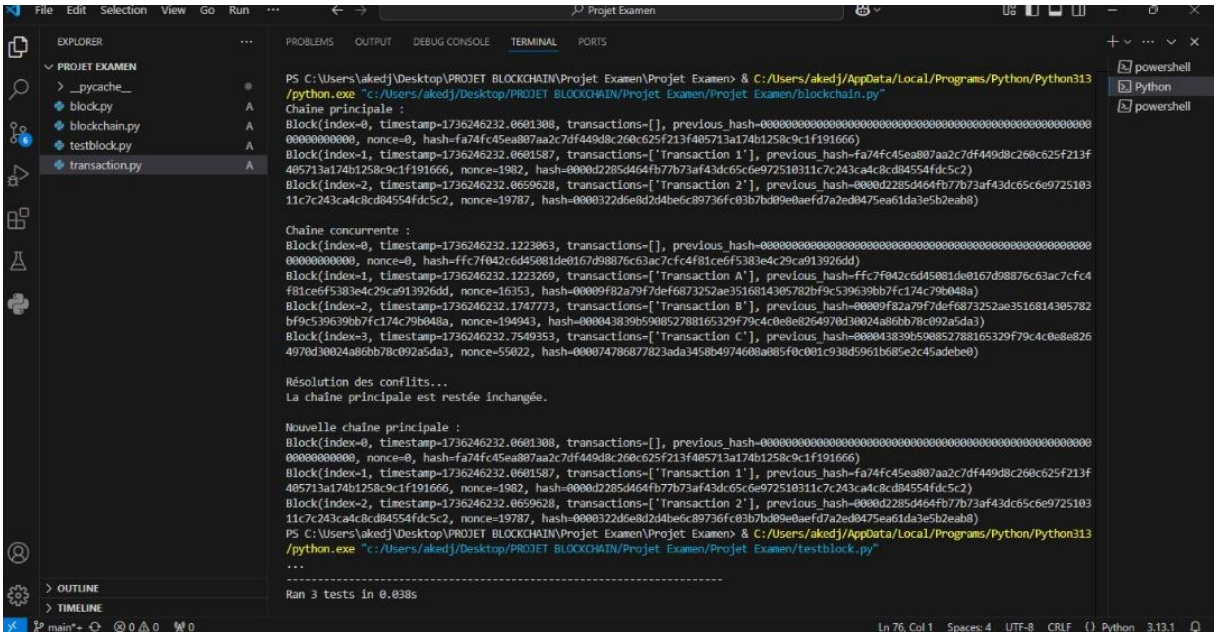
- Parcourt les chaînes des autres nœuds.
- Compare leur longueur avec la chaîne actuelle.
- Si une chaîne concurrente est plus longue et valide, elle remplace la chaîne actuelle.

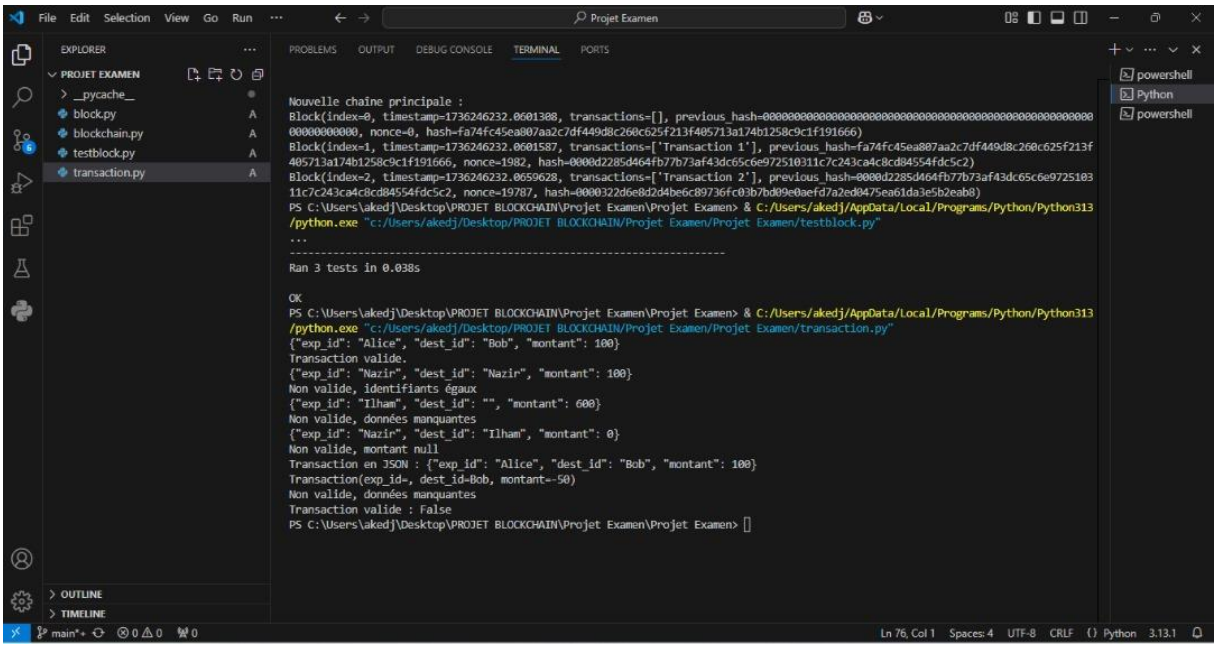
2. Résultat observé

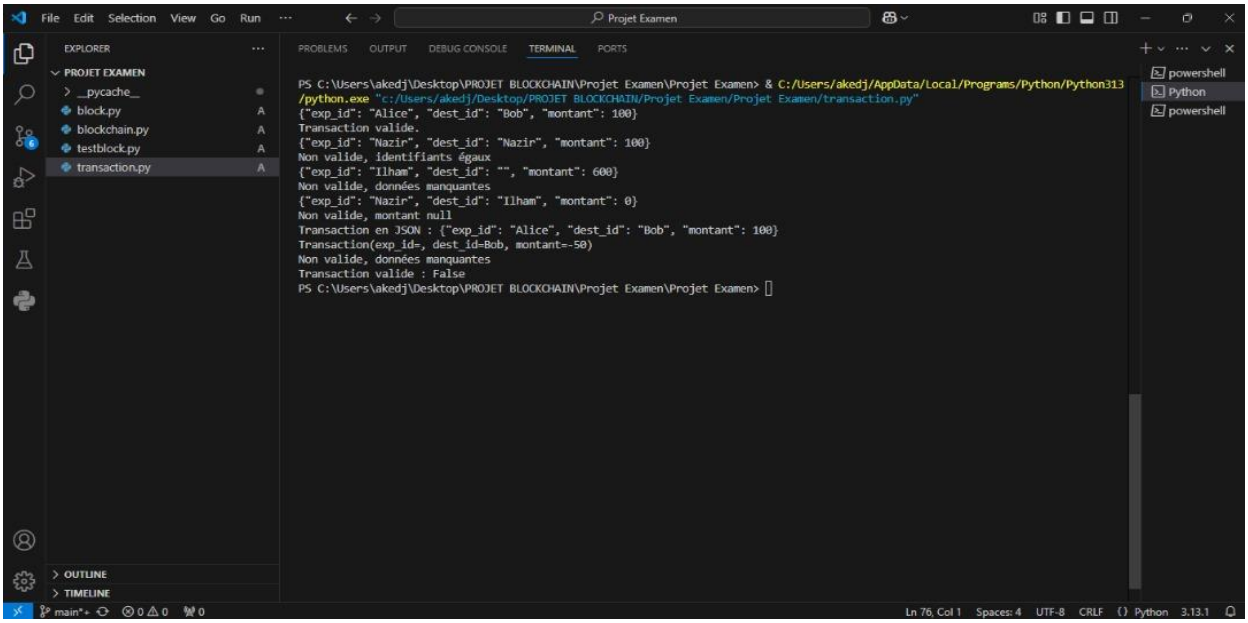
- La chaîne valide la plus longue est adoptée avec succès.

II. Capture d’écran de l'exécution du code






File Edit Selection View Go Run
Projet Examen
EXPLORER
PROJET EXAMEN
__pycache__
block.py
blockchain.py
testblock.py
transaction.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python
powershell
Nouvelle chaîne principale :
Block(index=0, timestamp=1736246232.0601308, transactions=[], previous_hash=0000000000000000000000000000000000000000000000000000000000000000, nonce=0, hash=fa74fc45ea807aa2c7df449d8c260c625f213f405713a174b1258c9c1f191666)
Block(index=1, timestamp=1736246232.0601587, transactions=['Transaction 1'], previous_hash=fa74fc45ea807aa2c7df449d8c260c625f213f405713a174b1258c9c1f191666, nonce=1982, hash=0000d2285d464fb77b73af43dc65c6e972510311c7c243ca4c8cd84554fdc5c2)
Block(index=2, timestamp=1736246232.0659628, transactions=['Transaction 2'], previous_hash=0000d2285d464fb77b73af43dc65c6e972510311c7c243ca4c8cd84554fdc5c2, nonce=19787, hash=0000322d6e8d2d4be6c89736fc03b7bd09e0aefd7a2ed0475ea61da3e5b2eab8)
PS C:\Users\akedj\Desktop\PROJET BLOCKCHAIN\Projet Examen\Projet Examen> & C:/Users/akedj/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akedj/Desktop/PROJET BLOCKCHAIN/Projet Examen/Projet Examen/testblock.py"
...
Ran 3 tests in 0.038s
OK
PS C:\Users\akedj\Desktop\PROJET BLOCKCHAIN\Projet Examen\Projet Examen> & C:/Users/akedj/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akedj/Desktop/PROJET BLOCKCHAIN/Projet Examen/Projet Examen/transaction.py"
{"exp_id": "Alice", "dest_id": "Bob", "montant": 100}
Transaction valide.
{"exp_id": "Nazir", "dest_id": "Nazir", "montant": 100}
Non valide, identifiants égaux
{"exp_id": "Ilham", "dest_id": "", "montant": 600}
Non valide, données manquantes
{"exp_id": "Nazir", "dest_id": "Ilham", "montant": 0}
Non valide, montant null
Transaction en JSON : {"exp_id": "Alice", "dest_id": "Bob", "montant": 100}
Transaction(exp_id=, dest_id=Bob, montant=-50)
Non valide, données manquantes
Transaction valide : False
PS C:\Users\akedj\Desktop\PROJET BLOCKCHAIN\Projet Examen\Projet Examen>
OUTLINE
TIMELINE
main*+
Ln 76, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.1


File Edit Selection View Go Run
Projet Examen
EXPLORER
PROJET EXAMEN
__pycache__
block.py
blockchain.py
testblock.py
transaction.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python
powershell
PS C:\Users\akedj\Desktop\PROJET BLOCKCHAIN\Projet Examen\Projet Examen> & C:/Users/akedj/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akedj/Desktop/PROJET BLOCKCHAIN/Projet Examen/Projet Examen/transaction.py"
{"exp_id": "Alice", "dest_id": "Bob", "montant": 100}
Transaction valide.
{"exp_id": "Nazir", "dest_id": "Nazir", "montant": 100}
Non valide, identifiants égaux
{"exp_id": "Ilham", "dest_id": "", "montant": 600}
Non valide, données manquantes
{"exp_id": "Nazir", "dest_id": "Ilham", "montant": 0}
Non valide, montant null
Transaction en JSON : {"exp_id": "Alice", "dest_id": "Bob", "montant": 100}
Transaction(exp_id=, dest_id=Bob, montant=-50)
Non valide, données manquantes
Transaction valide : False
PS C:\Users\akedj\Desktop\PROJET BLOCKCHAIN\Projet Examen\Projet Examen>
OUTLINE
TIMELINE
main*+
Ln 76, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.1