✅ Zadanie 1: Utwórz strukturę bloku

Cel: Zaimplementuj klasę `Block`, która zawiera:

- numer bloku (index),
- czas utworzenia,
- dane (np. transakcje),
- hash poprzedniego bloku,
- własny hash.

✅ Zadanie 2: Zaimplementuj funkcję tworzącą hash bloku

Cel: W klasie `Block`, zaimplementuj metodę `calculate_hash`, która:

- tworzy SHA-256 z połączenia `index`, `timestamp`, `data`, `previous_hash`.

✅ Zadanie 3: Utwórz klasę Blockchain

Cel: Zaimplementuj klasę `Blockchain`, która:

- przechowuje listę bloków,
- ma metodę do tworzenia **bloku początkowego** (genesis block),
- umożliwia dodanie nowego bloku.

✅ Zadanie 4: Walidacja łańcucha bloków

Cel: Dodaj do klasy `Blockchain` metodę `is_chain_valid`, która:

- sprawdza, czy wszystkie hashe są poprawne,
- upewnia się, że każdy blok wskazuje poprawny hash poprzedniego bloku.

✅ Zadanie 5: Dodaj proof-of-work (kopanie bloków)

Cel: Dodaj mechanizm kopania z trudnością (`difficulty`), czyli np. hash musi zaczynać się od 2–4 zer (`00...`).

- Dodaj metodę `mine_block` w klasie `Block`, która:
    - zmienia nonce do momentu uzyskania odpowiedniego hasha.

✅ Zadanie 6: Prosty system transakcji

Cel: Zmienna `data` w bloku powinna zawierać listę transakcji (np. słowniki z `sender`, `receiver`, `amount`).

- Dodaj bufor oczekujących transakcji.
- Dodaj metodę `add_transaction`.
- Nowy blok powinien zawierać transakcje z bufora.

✅ Zadanie 7: Weryfikacja

Cel: Weryfikacja bezpieczeństwa blockchain

- Zmień transakcję w dowolnym bloku
- Wykonaj funkcję weryfikującą długość blockchain
    - Wykonaj funkcję weryfikującą każdy block w blockchain
      - poprawność hasha (`block.hash == block.calculate_hash()`),
    - czy `block.previous_hash` == `previous_block.hash`,
    - poprawność `nonce`, jeśli używany jest proof-of-work.
- Dodaj raport błędów: `is_chain_valid()` powinno zwracać nie tylko `True/False`, ale i listę wykrytych błędów.