**Code Documentation**

# Explanation of The Code

This code is a simple blockchain system. It builds a genesis block and 3 blocks with Merkle tree roots and initiates and handles transactions between users. The code adheres to the specifications given before.

1. Key Pair Generation:
   * The function begins by producing a pair of private and public keys for each of the six users. There is a "System" user and six normal users (User1 through User6). Each user's keys are saved in a dictionary.

1. Balances Initialization:
   * A dictionary is used to maintain track of each user's balance. According to the conditions, "User1" starts with 100 coins.

1. Transaction Initialization (**initiate\_transaction function**):
   * This function creates and processes transactions. As input, it requires the sender, receiver, and amount to be transferred. This is how it works:
     + - It determines whether or not the sender's balance is adequate to execute the transaction.
       - If the balance is adequate, the money is deducted from the sender and added to the balance of the receiver.
       - It produces a new public/private key pair for the recipient after each transaction for future transactions.
       - It preserves the receiver's old public and private keys.
       - The transaction data is returned, which includes the sender, receiver, amount, balances before and after the transaction, old public key (if it exists), transaction hash, and signature.

1. Merkle Root Calculation (**merkle\_root function**):
   * The Merkle root is calculated from a collection of transaction hashes using this function. It combines the hashes recursively until a single Merkle root is found.

1. Genesis Block Creation:
   * The code creates a list of genesis transactions in which the "System" allocates 100 coins to "User1." The initiate\_transaction function is used to execute these transactions, and the resultant transaction hashes are used to generate the genesis Merkle root with the merkle\_root function.

1. Block Generation (**generate\_block function**):

● This function generates a block. It accepts the block number, the hash of the preceding block, and a list of transactions as input. This is what it does:

* + - * + Using the merkle\_root function, it computes the Merkle root of the transactions in the block.
        + It generates the block header, which contains the block number, the hash of the preceding block, and the Merkle root.
        + It then uses SHA-256 to compute the hash of the block header.
        + The method outputs the block header as well as the information of each transaction included within the block.

1. Loop to Generate Blocks:

● A loop is used to construct three blocks, each of which contains four transactions. For each new block, the hash of the preceding block is used.

# Output Explanation

The code simulates a basic blockchain system with user accounts, transactions, and block creation.

**Genesis Block:**

The code starts by creating the Genesis Block, which is the first block in the blockchain.

Block Number: 0 - This is the first block.

Previous Block Header: None - Since it's the first block, there is no previous block.

Merkle Root: (a hexadecimal hash) - The Merkle root of the transactions in this block.

**Transactions in Genesis Block:**

The System allocates 100 coins to Alice in the Genesis block. The transactions are printed with sender, receiver, and amount.

**Transactions in Other Block:**

Three more blocks are created, each with its block header, block number and previous block header. For each block, there are four transactions created, showing the sender, receiver, amount, and balances. The sender's public key, the receiver's new public key, and the receiver's old public key are displayed in hexadecimal format. The verification status is shown for each transaction.

Each block has 4 transactions within it, along with the Block number, Previous Block Header and Merkle Root.

**Remaining Balances:**

After all the blocks are processed, the remaining balances of each user are displayed.











