Group Details:

1. Aadi Aarya Chandra (200101002)
2. Pradeep Kumar (200101080)
3. Prakhar Pandey (200101081)
4. Pratham Pekamwar (200101087)

File Structure and Functionalities:

1. helper.py:

- **computeMerkelRoot(data):** (Placeholder) Calculates the Merkle Root of a set of transactions to ensure data integrity.
- **computeBlockHeaderHash(header):** Computes the hash of a block header using SHA-256 for identification and immutability.

2. block.py:

- **BlockHeader class:** Defines the structure of a block header containing metadata such as version, difficulty target, previous block hash, timestamp, nonce, miner ID, and Merkle root.
- **Block class:** Represents a complete block in the blockchain, including transaction data, header information, block height, block hash, and a timestamp for difficulty adjustments.
- **Methods:** Provides access to block attributes and a `display` function to print block details.

3. main.py:

- Serves as the main program entry point.
- Creates a `Blockchain` instance.
- Offers an interactive loop for users to:
  - Add new blocks with custom data.
  - Insert sample blocks for testing.
  - Visualize the current blockchain structure.
  - Observe dynamic difficulty adjustments.
  - Exit the program.

4. blockchain.py:

- **Blockchain class:** Manages the entire blockchain, including the genesis block, chain height, and a list of all blocks.
- **Methods:**
  - **propagateBlockToPeers(block):** (Placeholder) Shares new blocks with other nodes in a decentralized network.
  - **listenToPeers():** (Placeholder) Receives new blocks from other nodes for verification and addition.

- **addBlock(block):** Appends a verified block to the blockchain and updates relevant attributes like chain height.
- **verifyBlock(block):** Ensures block validity before adding it to the chain by checking:
  - Previous block hash
  - Timestamp
  - Merkle root
  - Transactions (placeholder)
  - Difficulty target
  - Block hash
  - Block height
- **calculateBlockDifficulty(block):** Adjusts the mining difficulty target based on the time taken to mine the previous 2016 blocks, ensuring consistent block generation times.
- **displayBlockchain():** Prints information about all blocks in the chain.
- **verifyTX(txData):** (Placeholder) Verifies the validity of transactions within a block.
- **verifyAndAdd(toBeAddedBlock):** Combines verification and addition of a new block into a single operation.

How to Use the Project:

1. Setup:

- Ensure you have Python 3 installed on your system.
- Clone or download the project files to your local machine. Github Link: https://github.com/sirus-max/mini_blockchain.git

2. Running the Program:

- Open a terminal or command prompt and navigate to the project directory.
- Run the following command to start the interactive program:

```
python main.py
```