Name : Devansh Wadhwani **Class: D20A** **Roll No: 64**

BLOCKCHAIN LAB

EXPERIMENT NO - 2

AIM : Cryptography in Blockchain, Merkle root Tree Hash.

THEORY :

What is Blockchain?

Blockchain is a distributed, decentralized, and immutable digital ledger used to record transactions or data securely across multiple systems. Instead of storing data in a single central server, blockchain maintains a chain of blocks, where each block contains a group of transactions and is cryptographically linked to the previous block.

Each block in a blockchain consists of:

- Transaction data
- Timestamp
- Hash of the previous block
- Proof (Nonce) generated using Proof-of-Work

The blocks are connected using cryptographic hash functions (SHA-256), which ensures that once data is recorded, it cannot be altered without changing all subsequent blocks. This property provides data integrity and tamper resistance.

Blockchain operates on a peer-to-peer network, where all participants maintain a copy of the ledger. Any new block added to the chain must be validated using a consensus mechanism, such as Proof-of-Work (PoW), which requires solving a computationally difficult cryptographic puzzle.

In this experiment, blockchain is implemented in a single-node environment to demonstrate its core working principles:

- Block creation
- Proof-of-Work mining
- SHA-256 hashing
- Chain validation

Although simplified, this implementation effectively demonstrates how blockchain ensures security, transparency, immutability, and trust without relying on a centralized authority.

Key Features of Blockchain

- Decentralization – No central control over data
- Immutability – Data cannot be modified once stored
- Transparency – All transactions are verifiable
- Security – Cryptographic hashing protects data
- Consensus Mechanism – Validates blocks before addition

Blockchain technology provides a secure and reliable method for storing and validating data. The experimental implementation demonstrates the fundamental concepts of blockchain, making it suitable for academic and practical learning purposes.

Block Components of Blockchain

A block is the basic unit of a blockchain. Each block stores data and is cryptographically linked to the previous block, forming a secure chain.
The main components of a block are explained below:

1. Block Index

- Indicates the position of the block in the blockchain.
- The first block is called the Genesis Block and has index 1 (or 0 in some systems).

2. Timestamp

- Records the date and time when the block is created.
- Helps in maintaining the chronological order of blocks.

3. Transaction Data

- Contains the actual information or transactions stored in the block.
- In cryptocurrency, this includes sender, receiver, and amount.
- In experiments, it may be simple text or sample data.

4. Previous Block Hash

- Stores the hash of the previous block.
- This creates a link between blocks, forming a chain.
- If any block is altered, its hash changes and breaks the chain.

5. Nonce (Proof)

- A number used in the Proof-of-Work (PoW) process.
- Miners change the nonce to generate a valid hash.
- Ensures that block creation requires computational effort.

6. Current Block Hash

- A SHA-256 cryptographic hash of the block’s contents.
- Ensures data integrity and immutability.
- Even a small change in block data produces a completely different hash.

Block Structure Summary

```
Block {
  Index
  Timestamp
  Transactions / Data
  Previous Hash
  Nonce (Proof)
  Current Hash
}
```

Importance of Block Components

- Ensures security using cryptographic hashing
- Maintains immutability of data
- Enables verification and validation of blockchain
- Prevents tampering and fraud

Process of Mining in Blockchain

Mining is the process of adding new blocks to the blockchain by solving a cryptographic puzzle using a consensus mechanism called Proof-of-Work (PoW). It ensures the security, integrity, and validity of the blockchain.

Step-by-Step Mining Process

1. Transaction Collection

- New transactions are collected from users or the network.
- These transactions are temporarily stored before being added to a block.

2. Block Formation

- A new block is created containing:
  - Transaction data
  - Timestamp
  - Hash of the previous block
  - Nonce (initially set to a value like 0)

3. Proof-of-Work Puzzle

- The miner tries to find a nonce such that:

```
SHA-256(Block Data + Nonce) starts with required leading zeros
```

- This makes mining computationally difficult.

4. Hash Calculation

- The SHA-256 hash function is applied repeatedly.
- Each nonce produces a different hash output.
- The miner continues until a valid hash is found.

5. Block Validation

- Once a valid hash is found:
    - The block is considered mined
    - Other nodes verify the solution
    - The hash is checked against difficulty rules

6. Block Addition

- The verified block is added to the blockchain.
- The chain remains intact by linking the new block's `previous_hash`.

7. Reward (Conceptual)

- In real blockchains, miners receive:
    - Cryptocurrency rewards
    - Transaction fees

Mining Process Flow

```
Collect Transactions

        ↓

Create Block

        ↓

Apply Proof-of-Work

        ↓

Find Valid Hash

        ↓

Validate Block

        ↓

Add Block to Blockchain
```

Purpose of Mining

- Prevents fraudulent block creation
- Ensures network consensus
- Maintains blockchain immutability
- Secures the ledger against attacks

Check Validity of Blocks in Blockchain

The validity of blocks in a blockchain is verified to ensure that the data has not been altered and that the chain remains secure and trustworthy. Block validation is performed using cryptographic hashing and consensus rules such as Proof-of-Work (PoW).

Steps to Check Block Validity

1. Verify Previous Hash Link

- Each block stores the hash of the previous block.
- The system recalculates the hash of the previous block.
- If:

```
stored previous_hash ≠ calculated hash(previous block)
```

the block is invalid

2. Validate Proof-of-Work

- The nonce (proof) is checked to ensure it satisfies the PoW condition.
- The hash generated using:

```
SHA-256(proof² - previous_proof²)
```

must start with the required number of leading zeros.

- If the condition fails, the block is rejected.

3. Recalculate Block Hash

- The block’s contents are hashed again using SHA-256.
- Any change in data produces a different hash.
- This confirms data immutability.

4. Validate Chain Sequentially

- Validation starts from the Genesis Block.
- Each block is checked in sequence until the last block.
- If all blocks satisfy hash linkage and PoW, the chain is valid.

5. Detect Tampering

- If even a single block is modified:
    - Its hash changes
    - The next block’s `previous_hash` becomes invalid
    - Chain validation fails

Validation Logic Flow

```
Start from Genesis Block
        ↓
Check Previous Hash
        ↓
Verify Proof-of-Work
        ↓
Recalculate Hash
        ↓
Repeat for Next Block
```

Validation in Experimental Implementation

In the Python blockchain experiment:

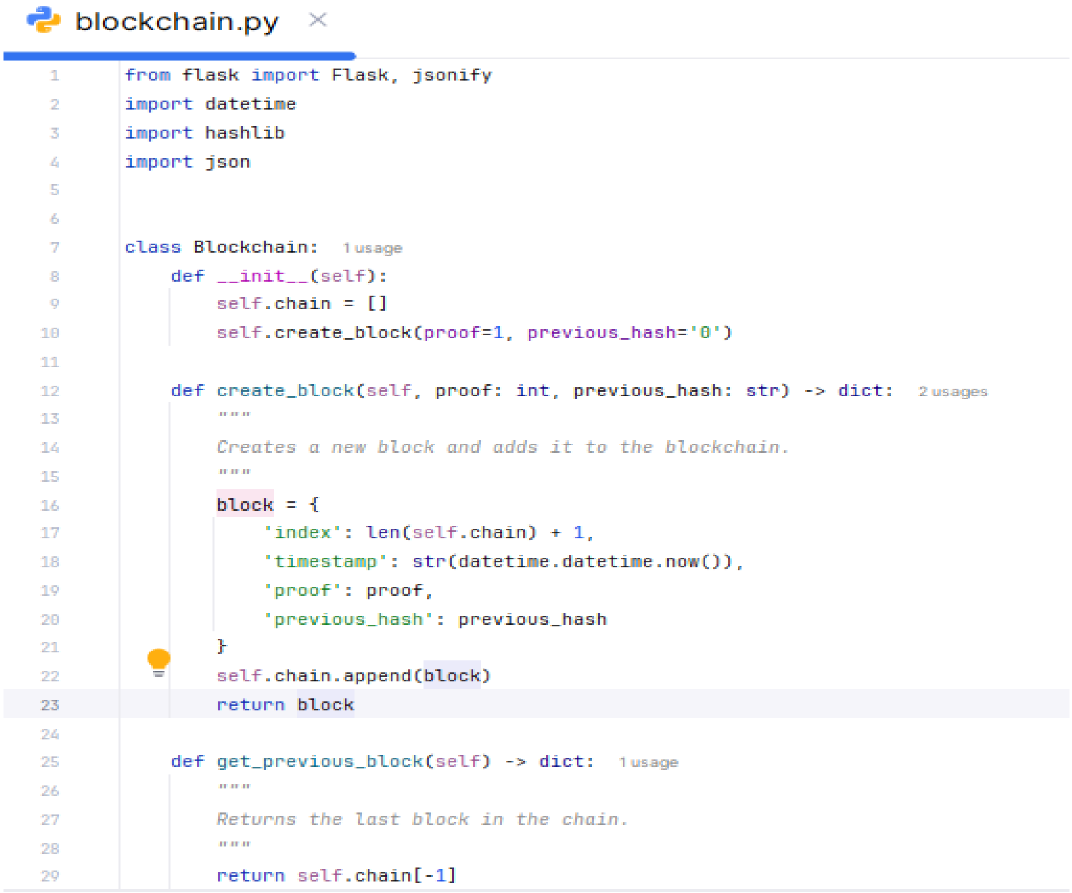
- `is_chain_valid()` method is used
- It checks:
  - Correct hash linkage
  - Valid Proof-of-Work
- Returns `True` if the blockchain is valid, otherwise `False`

Importance of Block Validation

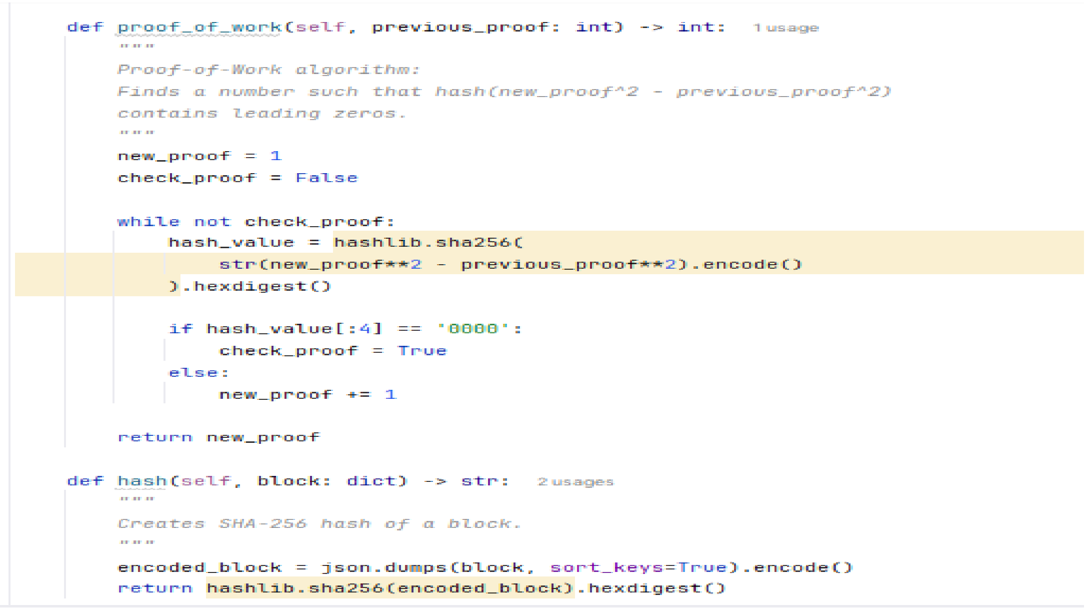
- Ensures data integrity
- Prevents unauthorized modifications
- Maintains trust in the blockchain
- Secures the distributed ledger

Block validation is essential for maintaining the security and reliability of blockchain systems. By verifying hashes and Proof-of-Work, blockchain ensures that all blocks remain authentic and untampered.

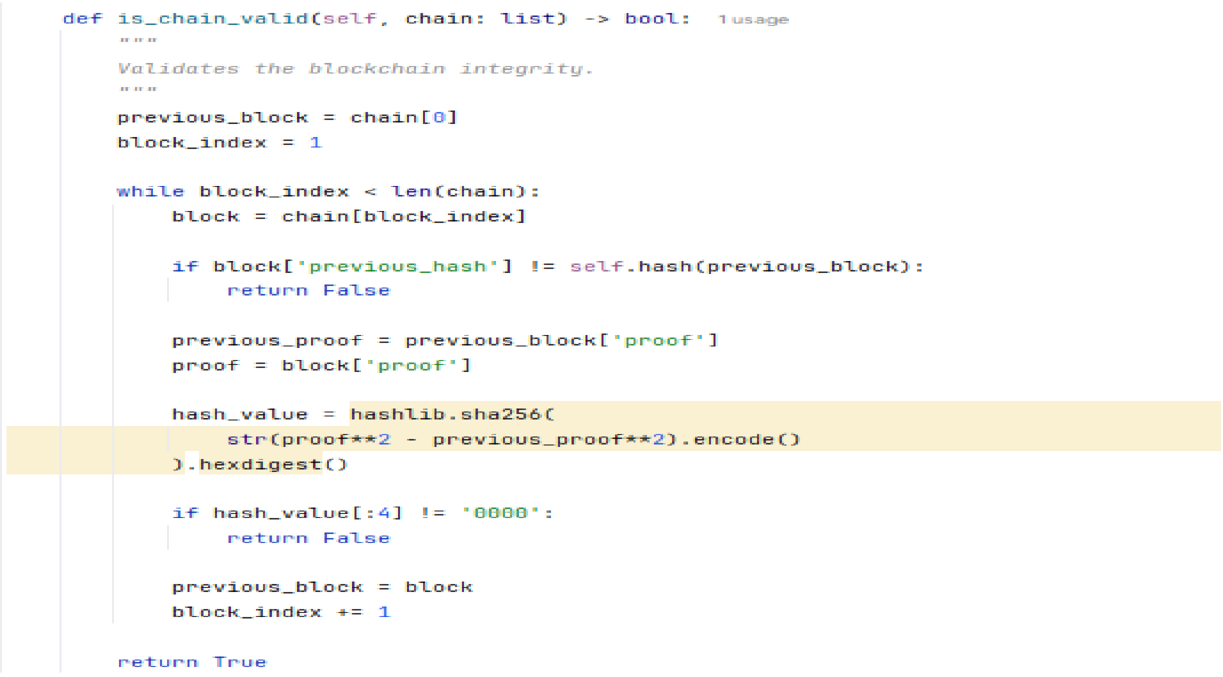
CODE :



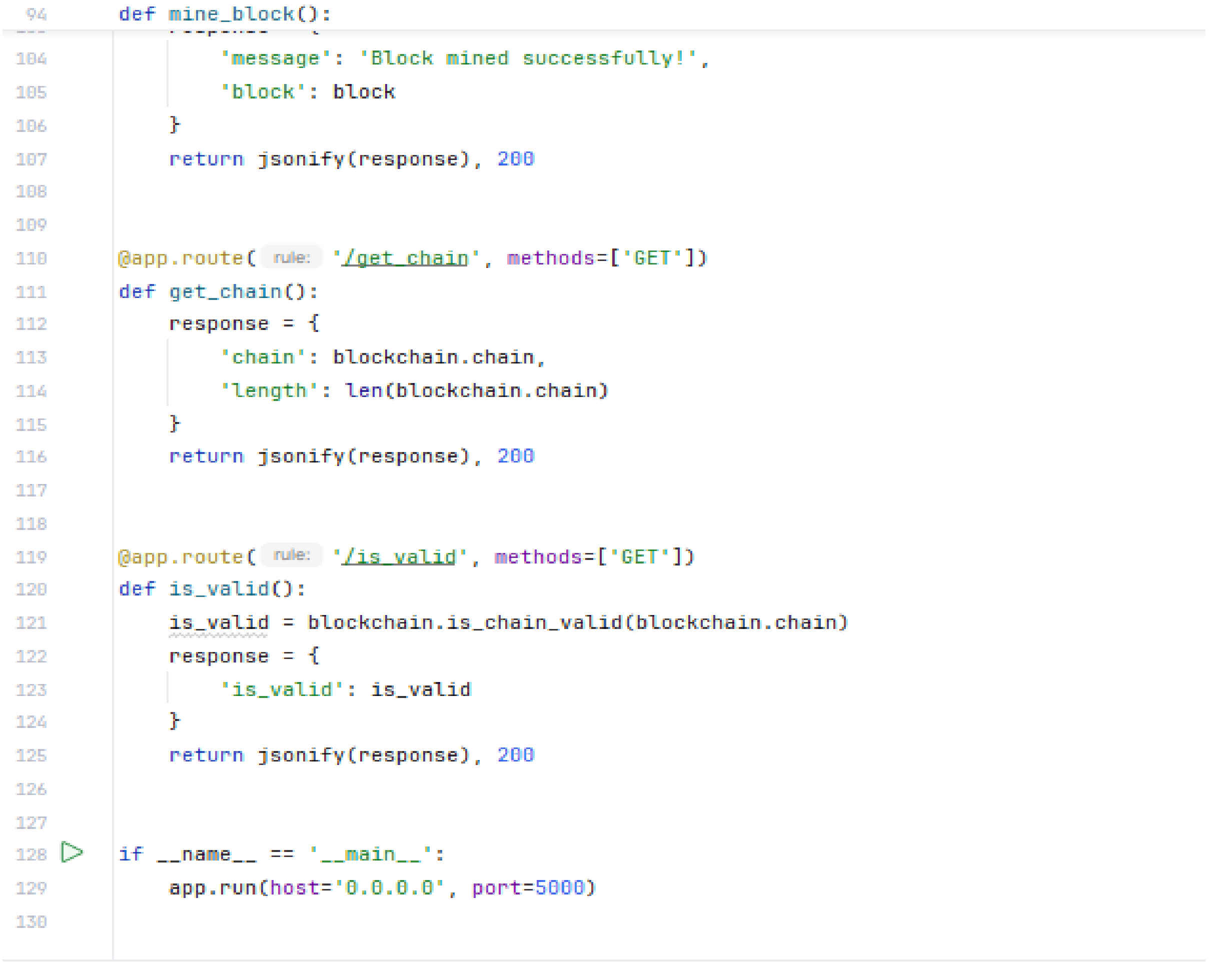
```python
from flask import Flask, jsonify
import datetime
import hashlib
import json


class Blockchain:  1 usage
    def __init__(self):
        self.chain = []
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof: int, previous_hash: str) -> dict:  2 usages
        """
        Creates a new block and adds it to the blockchain.
        """
        block = {
            'index': len(self.chain) + 1,
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash
        }
        self.chain.append(block)
        return block

    def get_previous_block(self) -> dict:  1 usage
        """
        Returns the last block in the chain.
        """
        return self.chain[-1]
```



```python
    def proof_of_work(self, previous_proof: int) -> int:  1 usage
        """
        Proof-of-Work algorithm:
        Finds a number such that hash(new_proof^2 - previous_proof^2)
        contains leading zeros.
        """
        new_proof = 1
        check_proof = False

        while not check_proof:
            hash_value = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()

            if hash_value[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1

        return new_proof

    def hash(self, block: dict) -> str:  2 usages
        """
        Creates SHA-256 hash of a block.
        """
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()
```



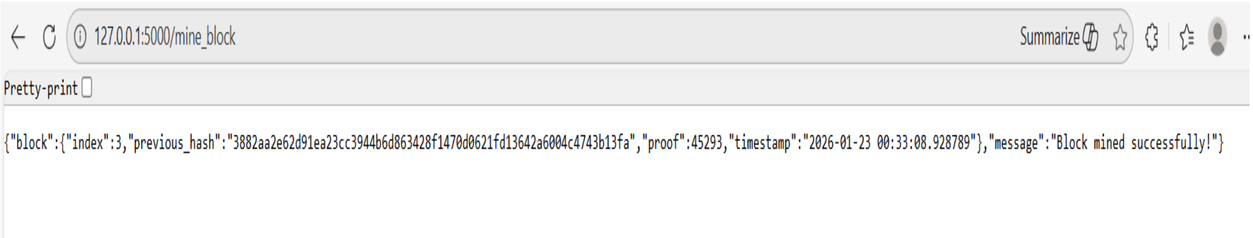
```
    def is_chain_valid(self, chain: list) -> bool:  1 usage
        """
        Validates the blockchain integrity.
        """
        previous_block = chain[0]
        block_index = 1

        while block_index < len(chain):
            block = chain[block_index]

            if block['previous_hash'] != self.hash(previous_block):
                return False

            previous_proof = previous_block['proof']
            proof = block['proof']

            hash_value = hashlib.sha256(
                str(proof**2 - previous_proof**2).encode()
            ).hexdigest()

            if hash_value[:4] != '0000':
                return False

            previous_block = block
            block_index += 1

        return True
```



```
def mine_block():
            'message': 'Block mined successfully!',
            'block': block
        }
    return jsonify(response), 200


@app.route( rule: '/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200


@app.route( rule: '/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    response = {
        'is_valid': is_valid
    }
    return jsonify(response), 200


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

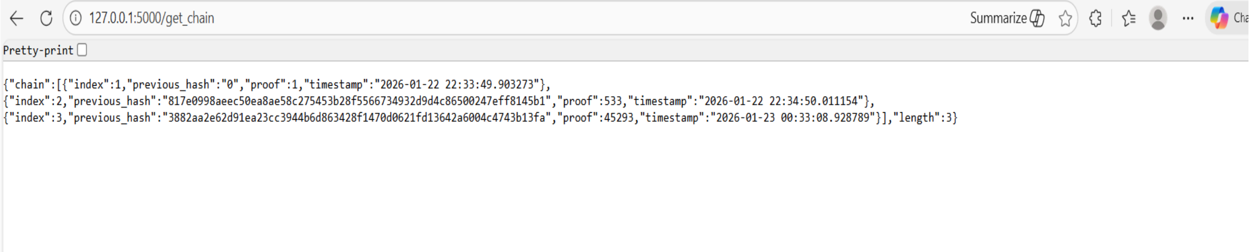
OUTPUT :

Mine_block



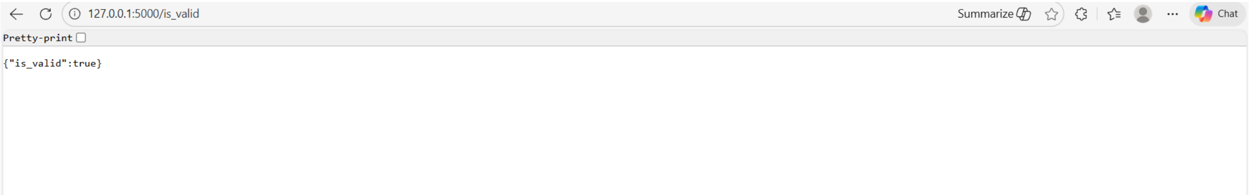
```
{"block":{"index":3,"previous_hash":"3882aa2e62d91ea23cc3944b6d863428f1470d0621fd13642a6004c4743b13fa","proof":45293,"timestamp":"2026-01-23 00:33:08.928789"},"message":"Block mined successfully!"}
```

get_chain



```
{"chain":[{"index":1,"previous_hash":"0","proof":1,"timestamp":"2026-01-22 22:33:49.903273"},
{"index":2,"previous_hash":"817e0998aeec50ea8ae58c275453b28f5566734932d9d4c86500247eff8145b1","proof":533,"timestamp":"2026-01-22 22:34:50.011154"},
{"index":3,"previous_hash":"3882aa2e62d91ea23cc3944b6d863428f1470d0621fd13642a6004c4743b13fa","proof":45293,"timestamp":"2026-01-23 00:33:08.928789"}],"length":3}
```

is_valid



```
{"is_valid":true}
```

Conclusion:

The blockchain integrity was verified by validating previous hash links and Proof-of-Work conditions, ensuring immutability and tamper resistance. The Flask-based Web API enabled interaction with the blockchain by allowing block mining, chain retrieval, and validation through HTTP requests. It was performed in Python 3 environment.