Mohit Patil **D20A** **04**

Blockchain Experiment 2

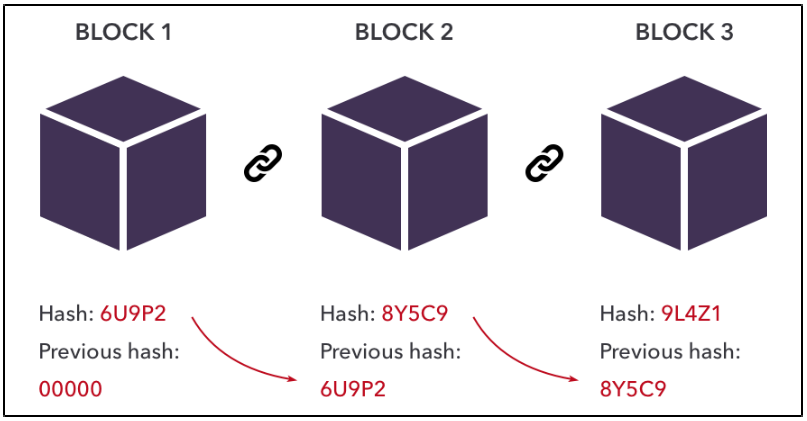
AIM: Create a Blockchain using Python

THEORY:

Q1: What is a Blockchain?

A Blockchain is a distributed and decentralized digital ledger that records transactions across multiple computers in a secure and transparent manner. Each record is stored in a block, and these blocks are linked together using cryptographic hash values, forming a chain.

Once data is added to a block and confirmed, it cannot be altered or deleted, which makes blockchain highly secure and tamper-resistant. Blockchain technology removes the need for a central authority and allows participants in the network to trust the system through cryptographic techniques and consensus mechanisms.



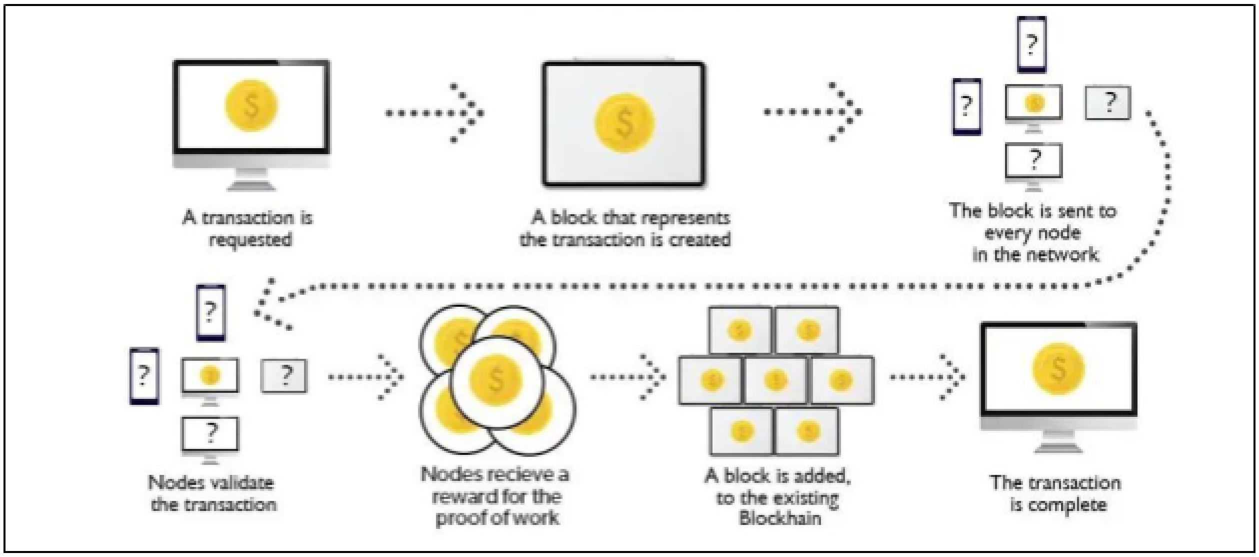
Q2: Process of Mining

Mining is the process of adding new blocks to the blockchain. It involves validating transactions and solving a cryptographic puzzle using computational power.

Steps in the Mining Process:

a. Transactions are collected and grouped into a block.
b. Miners calculate the hash of the block data along with a nonce.
c. The nonce is repeatedly changed until the hash meets the network's difficulty condition (e.g., leading zeros).
d. The first miner to find the valid hash broadcasts the block to the network.
e. Other nodes verify the solution.
f. Once verified, the block is added to the blockchain.

Mining ensures network security, prevents double spending, and maintains consensus among participants.


A transaction is requested
A block that represents the transaction is created
The block is sent to every node in the network
Nodes validate the transaction
Nodes recieve a reward for the proof of work
A block is added, to the existing Blockchain
The transaction is complete

Q3: How to check the validity of blocks in a Blockchain

The validity of blocks in a blockchain is checked to ensure that the data has not been tampered with and that all blocks follow the network rules.

Block Validation Process:

- **Previous Hash Verification:**
  Each block stores the hash of the previous block. If this hash does not match the actual hash of the previous block, the chain is invalid.
- **Proof-of-Work Verification:**
  The block's proof value is checked to confirm that it satisfies the required difficulty condition.
- **Hash Integrity Check:**
  The block data is re-hashed to ensure the hash value remains unchanged.
- **Chain Continuity Check:**
  Blocks are validated sequentially from the genesis block to the latest block.

If all these conditions are satisfied, the blockchain is considered valid and secure.

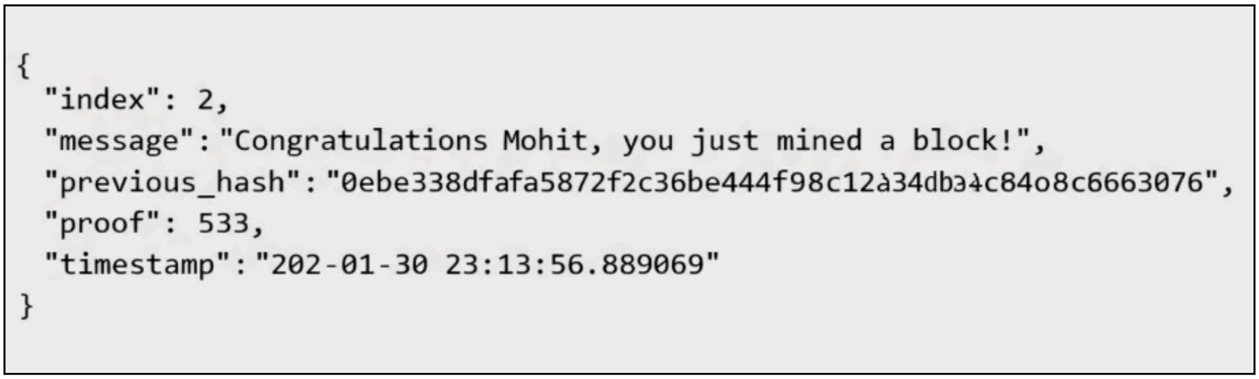
TASKS PERFORMED:

Task: Implementation of a Simple Blockchain with Proof-of-Work and Flask API

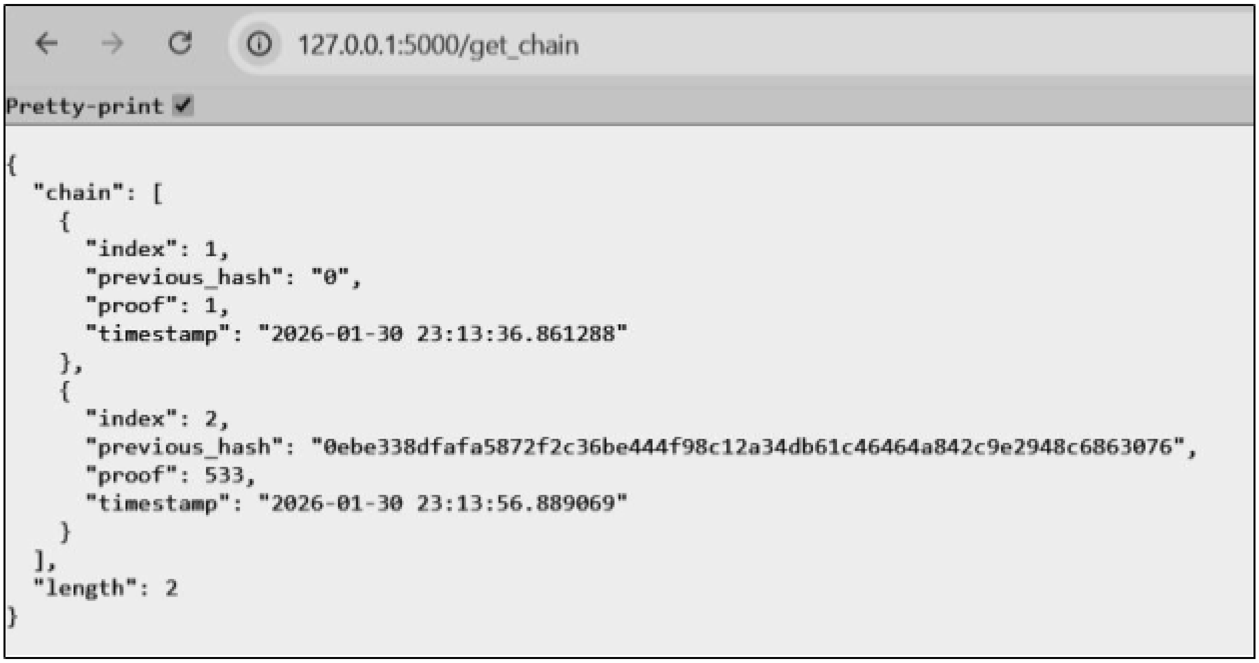
```
# Importing libraries
import datetime          # To generate timestamps for
blocks import hashlib    # To generate SHA-256 hashes
import json              # To convert block data into JSON
format from flask import Flask, jsonify # To create REST API
endpoints

# Part 1 - Building the Blockchain
class Blockchain:


    def _init_(self):
```

```python
        # List to store the blockchain
        self.chain = []

        # Create the genesis block
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        """
        Create a new block and add 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):
        """
        Return the last block in the chain
        """
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        """
        Proof of Work Algorithm:
        Find a number such that the hash of
        (new_proof^2 - previous_proof^2)
        starts with '0000'
        """
        new_proof = 1
        check_proof = False

        while not check_proof:
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1

        return new_proof

    def hash(self, block):
        """
        Create a 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):
        """
        Check whether the blockchain is valid
        """
        previous_block = chain[0]
        block_index = 1

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

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

            # Check proof of work
            previous_proof = previous_block['proof']
            proof = block['proof']

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

            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1

        return True


# Part 2 - Mining the Blockchain (Flask API)
# Create Flask app
app = Flask(_name_)

# Create Blockchain object
blockchain = Blockchain()

# API Endpoint - Mine a new block
@app.route('/mine_block', methods=['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']

    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)

    block = blockchain.create_block(proof, previous_hash)

    response = {
        'message': 'Congratulations Mohit, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }

    return jsonify(response), 200


# API Endpoint - Get full blockchain
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
```

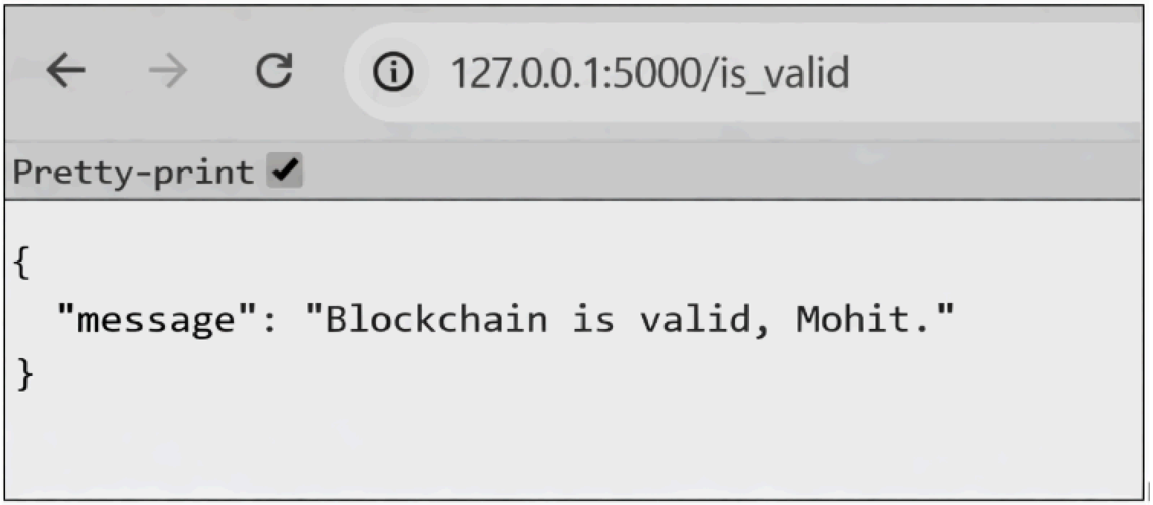
```python
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200


# API Endpoint - Check blockchain validity
@app.route('/is_valid', methods=['GET'])
def is_valid():
    valid = blockchain.is_chain_valid(blockchain.chain)

    if valid:
        response = {'message': 'Blockchain is valid, Mohit.'}
    else:
        response = {'message': 'Blockchain is NOT valid, Mohit.'}

    return jsonify(response), 200


# Run the Flask Application
if _name____ == '_main_':
    app.run(host='0.0.0.0', port=5000)
```



```
{
  "index": 2,
  "message":"Congratulations Mohit, you just mined a block!",
  "previous_hash":"0ebe338dfafa5872f2c36be444f98c12a34db34c84o8c6663076",
  "proof": 533,
  "timestamp":"202-01-30 23:13:56.889069"
}
```



```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-01-30 23:13:36.861288"
    },
    {
      "index": 2,
      "previous_hash": "0ebe338dfafa5872f2c36be444f98c12a34db61c46464a842c9e2948c6863076",
      "proof": 533,
      "timestamp": "2026-01-30 23:13:56.889069"
    }
  ],
  "length": 2
}
```



```
{
  "message": "Blockchain is valid, Mohit."
}
```

CONCLUSION:

In this experiment, a simple blockchain was successfully implemented using Python and Flask. The system allows block mining, viewing the complete blockchain, and checking its validity. Proof-of-Work was used to make block creation secure, while cryptographic hashing ensured that the data stored in blocks remains unchanged. This implementation helps in understanding the basic concepts of blockchain such as immutability, consensus, and decentralization, and provides a good foundation for learning advanced blockchain applications.