Shreya Sawant
Roll no - 55

Experiment No: 2

Aim: Create a Blockchain using Python

Theory:

1. What is a Blockchain?

Blockchain is a revolutionary technology that functions as a shared, immutable digital ledger. The name "blockchain" comes from its structure data is organized in blocks, with each new block linked to the one before it, forming a continuous chain.

Each block contains crucial data, such as a list of transactions, a timestamp, and a unique identifier called a cryptographic hash. This hash is generated from the block's contents and the hash of the previous block, ensuring that each block is tightly connected to the one before it.

- Blockchain's linked structure makes data tampering detectable by altering hashes and breaking the chain.
- It acts as a distributed database, storing transactions across the network.
- Each transaction is verified by the majority, ensuring legitimacy.
- This decentralization prevents any single party from manipulating the data.

Blockchain is decentralized and distributed, meaning no single authority controls it. Instead, multiple computers (nodes) on a network each have a copy of the blockchain, keeping the ledger synchronized.

2. Process of Mining

Mining is the process of validating transactions and adding new blocks to the blockchain using a consensus mechanism such as Proof of Work (PoW). It ensures the security and integrity of the blockchain network.

Steps involved in the mining process are:
• Transaction Collection: New transactions are collected and grouped into a block.
• Cryptographic Puzzle: Miners attempt to solve a complex mathematical problem by finding a nonce.
• Golden Nonce Discovery: The correct nonce that produces a hash with required

leading zeros is called the Golden Nonce.
• Block Validation: Other nodes verify the block's hash and proof.
• Block Addition: Once validated, the block is added to the blockchain.
• Reward: The miner receives a reward for successfully mining the block.

Mining requires computational effort, which makes altering the blockchain extremely difficult. This process prevents double spending, ensures decentralization, and maintains trust in the network.

3. How to check the validity of blocks in a Blockchain

Block validation ensures that the blockchain remains secure and untampered. Each block is checked against specific rules before being accepted into the chain.

Block validity is verified by:
• Previous Hash Verification: The previous hash stored in the block must match the actual hash of the previous block.
• Proof of Work Validation: The hash must satisfy the difficulty condition (such as leading zeros).
• Transaction Integrity Check: Any change in transaction data invalidates the block hash.
• Sequential Verification: Blocks are validated from the genesis block to the latest block.

If all validation conditions are satisfied, the blockchain is considered valid. This verification process allows blockchain systems to operate without centralized control while maintaining high security and data integrity.

Code:

```
# Importing libraries
import datetime      # To generate timestamps for blocks
import hashlib       # To generate SHA256 hashes
import json          # To convert block data into JSON
from flask import Flask, jsonify    # To create API endpointsclass Blockchain:

    def __init__(self):
        # This list will store the entire chain of blocks
        self.chain = []

        # Create the genesis block (first block)
        # proof = 1 → arbitrary
```

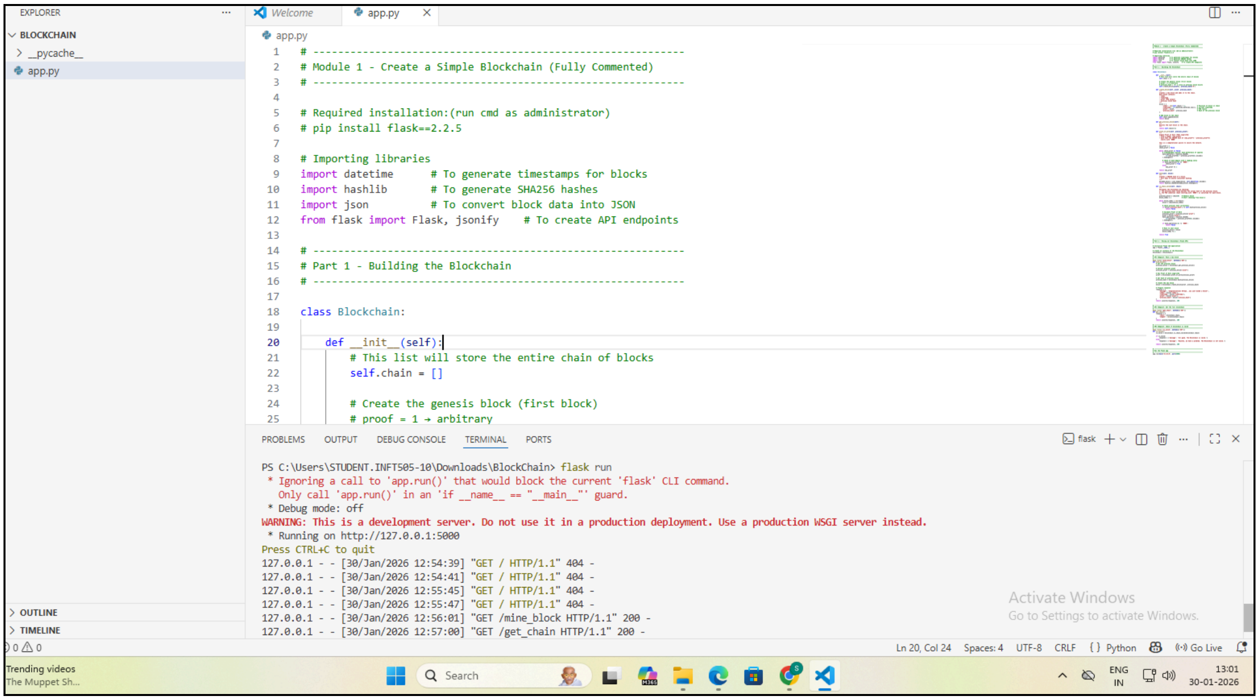
```python
        # previous_hash = '0' → since no previous block exists
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        """
        Creates a new block and adds it to the chain.
        Each block contains:
        - index
        - timestamp
        - proof (PoW output)
        - previous block hash
        """
        block = {
            'index': len(self.chain) + 1,              # Position of block in chain
            'timestamp': str(datetime.datetime.now()), # Current timestamp
            'proof': proof,                            # PoW result
            'previous_hash': previous_hash             # Hash of the previous block
        }

        # Add block to the chain
        self.chain.append(block)
        return block

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

    def proof_of_work(self, previous_proof):
        """
        Simple Proof of Work (PoW) algorithm:
        - Find a number (new_proof)
        - Such that the SHA256 hash of (new_proof^2 - previous_proof^2)
          starts with '0000'

        This is a computational puzzle to secure the network.
        """
        new_proof = 1
```

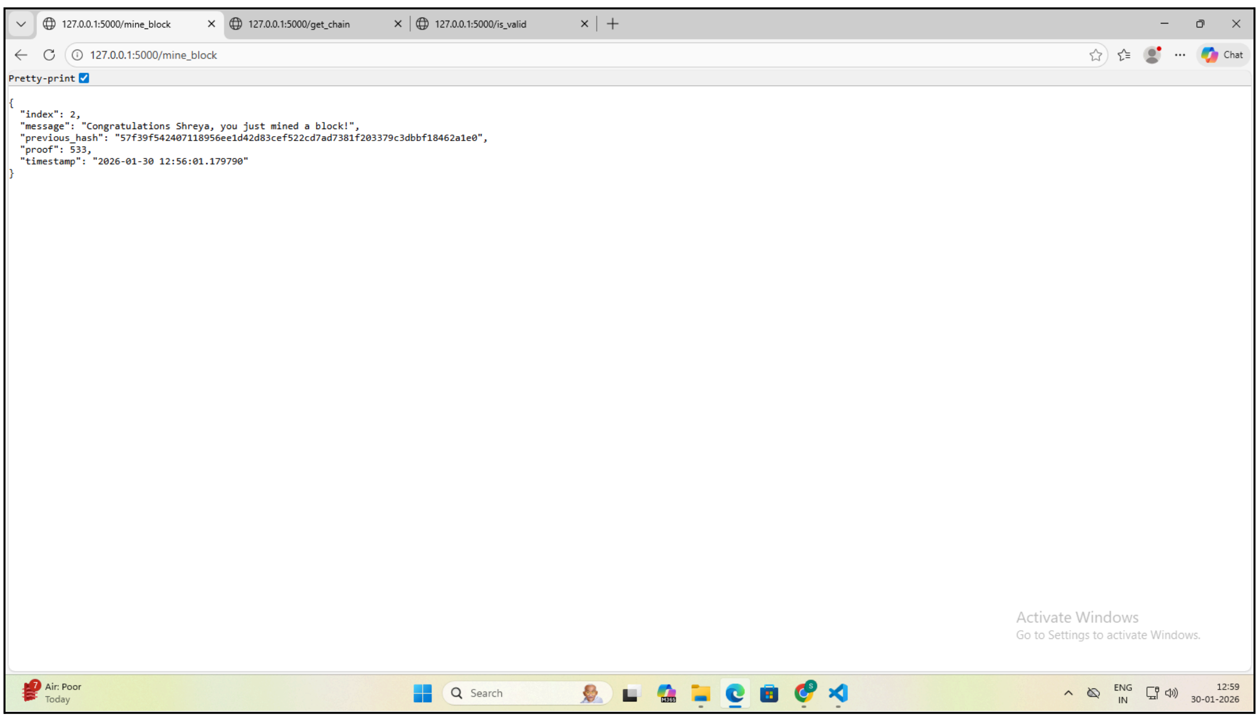
```
        check_proof = False

        while check_proof is False:
            # Cryptographic puzzle: hash difference of squares
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()

            # Check if hash begins with 4 leading zeros
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1

        return new_proof

    def hash(self, block):
        """
        Creates a SHA256 hash of a block.
        - Sort keys to ensure consistent hashing
        """
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        """
        Validates the blockchain by checking:
        1. The previous_hash field matches the actual hash of the previous block.
        2. The PoW condition (hash starting with '0000') is satisfied for each block.
        """
        previous_block = chain[0]   # Genesis block
        block_index = 1             # Start checking from block 2

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

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

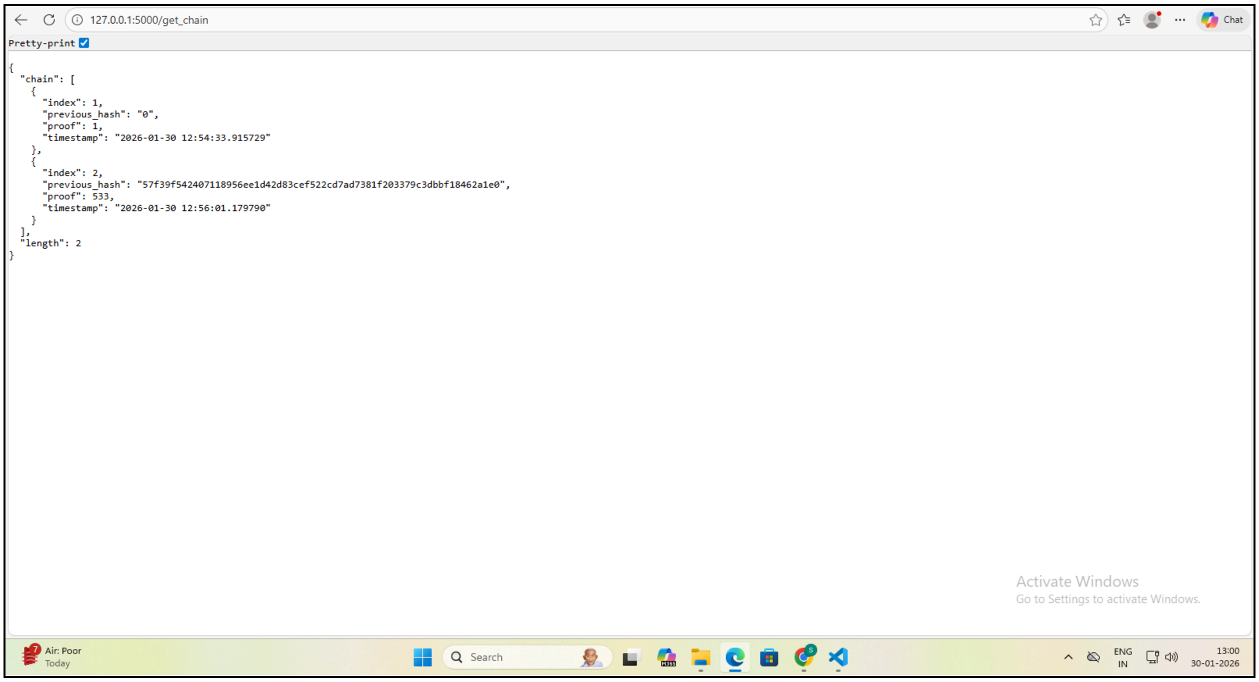
```
            # Validate Proof of Work
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(
                str(proof**2 - previous_proof**2).encode()
            ).hexdigest()

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

            # Move to next block
            previous_block = block
            block_index += 1

        return True

# Initialize Flask web application
app = Flask(__name__)

# Create an instance of the Blockchain
blockchain = Blockchain()
@app.route('/mine_block', methods=['GET'])
def mine_block():
    # Get the previous block
    previous_block = blockchain.get_previous_block()

    # Extract previous proof
    previous_proof = previous_block['proof']

    # Run Proof of Work algorithm
    proof = blockchain.proof_of_work(previous_proof)

    # Get hash of previous block
    previous_hash = blockchain.hash(previous_block)

    # Create the new block
    block = blockchain.create_block(proof, previous_hash)
```

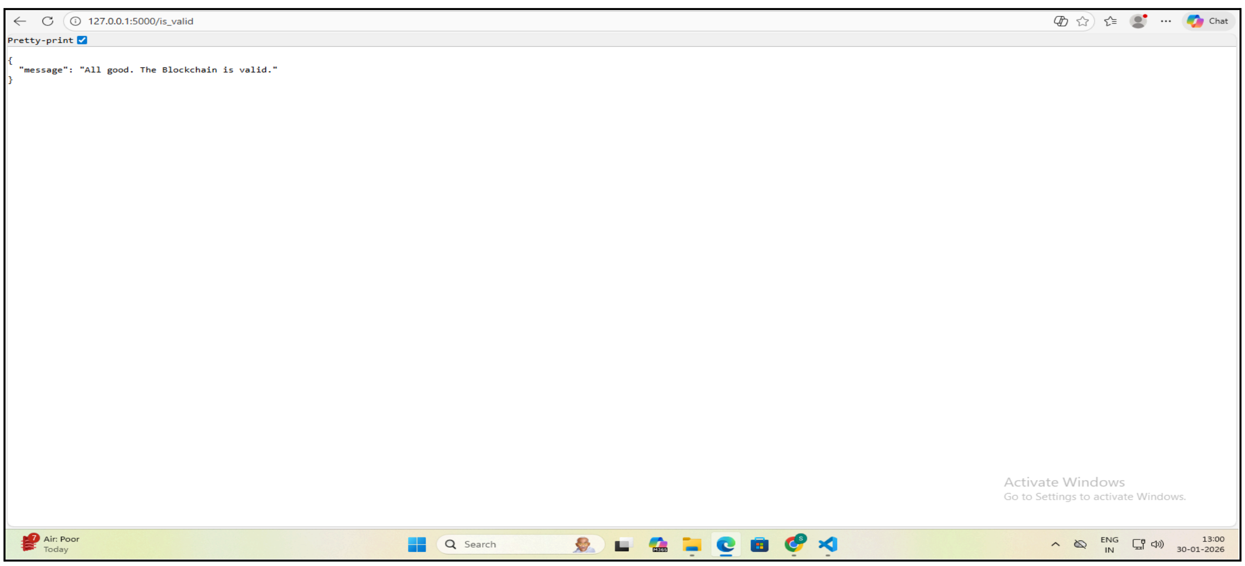
```python
    # Prepare response
    response = {
        'message': 'Congratulations Shreya , you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }
    return jsonify(response), 200
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200

@app.route('/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)

    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}

    return jsonify(response), 200
app.run(host='0.0.0.0', port=5000)
```

Output:


EXPLORER
BLOCKCHAIN
__pycache__
app.py
Welcome
app.py
# --
# Module 1 - Create a Simple Blockchain (Fully Commented)
# --
# Required installation:(run cmd as administrator)
# pip install flask==2.2.5
# Importing libraries
import datetime   # To generate timestamps for blocks
import hashlib   # To generate SHA256 hashes
import json   # To convert block data into JSON
from flask import Flask, jsonify   # To create API endpoints
# --
# Part 1 - Building the Blockchain
# --
class Blockchain:
def __init__(self):
# This list will store the entire chain of blocks
self.chain = []
# Create the genesis block (first block)
# proof = 1 → arbitrary
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\STUDENT.INFT505-10\Downloads\BlockChain> flask run
* Ignoring a call to 'app.run()' that would block the current 'flask' CLI command.
Only call 'app.run()' in an 'if __name__ == "__main__"' guard.
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [30/Jan/2026 12:54:39] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Jan/2026 12:54:41] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Jan/2026 12:55:45] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Jan/2026 12:55:47] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Jan/2026 12:56:01] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:57:00] "GET /get_chain HTTP/1.1" 200 -
OUTLINE
TIMELINE
Activate Windows
Go to Settings to activate Windows.
Ln 20, Col 24
Spaces: 4
UTF-8
CRLF
Python
Go Live
Trending videos
The Muppet Sh...
Search
ENG
IN
13:01
30-01-2026


127.0.0.1:5000/mine_block
127.0.0.1:5000/get_chain
127.0.0.1:5000/is_valid
127.0.0.1:5000/mine_block
Chat
Pretty-print
{
"index": 2,
"message": "Congratulations Shreya, you just mined a block!",
"previous_hash": "57f39f542407118956ee1d42d83cef522cd7ad7381f203379c3dbbf18462a1e0",
"proof": 533,
"timestamp": "2026-01-30 12:56:01.179790"
}
Activate Windows
Go to Settings to activate Windows.
Air: Poor
Today
Search
ENG
IN
12:59
30-01-2026





Conclusion:

In this experiment, we successfully created a simple blockchain using Python and demonstrated how blocks are created, linked, and secured using Proof of Work. The implementation shows how mining works, how new blocks are added to the chain, and how the blockchain's integrity can be verified. This helps in understanding the basic principles of decentralization, security, and immutability in blockchain technology.