Shreya Sawant
Roll no - 55

Experiment No: 2

Aim: Create a Blockchain using Python

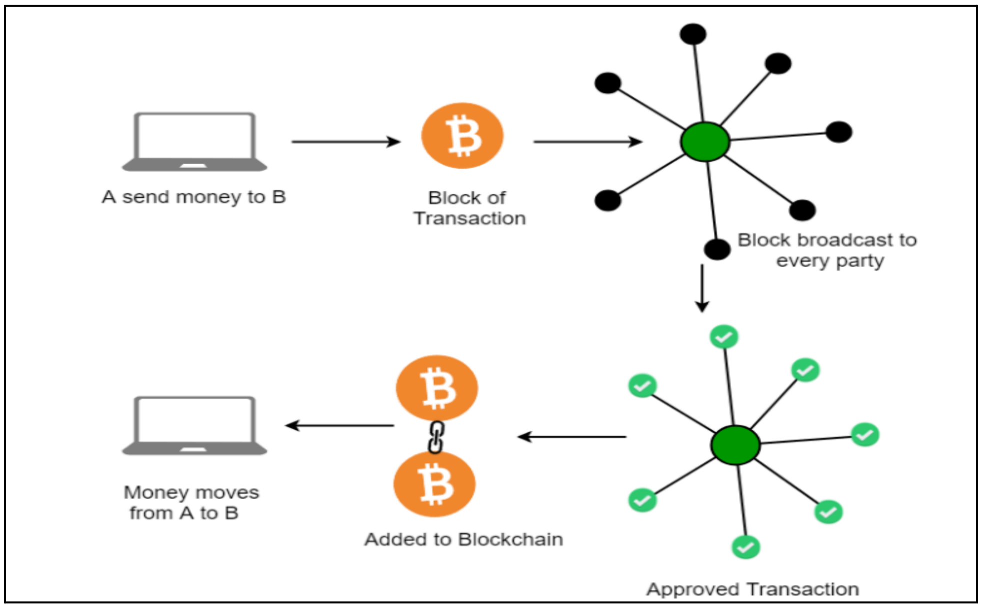
Theory:

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.



Key Features of Blockchain:

- **Decentralization:**
  No single organization or authority has full control over the blockchain. Instead, it is maintained collectively by all participants in the network, reducing dependency on a central system.
- **Transparency:**
  All transactions are visible to authorized participants in the network, promoting trust and accountability among users.
- **Immutability:**
  Once a block is added to the blockchain, altering it would require changing all subsequent blocks and gaining consensus from the majority of the network, making fraud or tampering highly unlikely.

Example:

In a banking system, a bank can modify records, but in blockchain, once a Bitcoin transaction is recorded, it cannot be changed

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. What is Block?

A block is a fundamental component of a blockchain that stores a set of verified transactions. It acts like a page in a digital record book. Each block is connected to the previous one using cryptographic hashes, ensuring the integrity and continuity of the chain.

Components of a Block:

1. Block Header:

It contains important metadata about the block and is used by miners during the mining process.

2. Previous Block Hash:
This links the current block to the previous block, forming a secure chain of blocks.

3. Timestamp:
It records the exact date and time when the block was created.

4. Nonce:
A unique number used in the mining process. Miners repeatedly change this value to find a valid hash that satisfies the network difficulty.

5. Merkle Root:
A cryptographic representation of all transactions in the block. It helps in verifying whether a specific transaction belongs to the block.

Example of a Block:

Block Number: 105678

Block Header:

- Previous Block Hash:
  0000000000000000000a3f2c9e7b1d45c8fa9a1e7b6d9c2f0a4e3b1c8d9f

- Merkle Root:
  4f3c2a1b9e8d7c6b5a4f3e2d1c9b8a7f6e5d4c3b2a1f9e8d7c6b5a4

- Timestamp: 2026-01-29 14:32:10 UTC

- Nonce: 845932

- Block Hash:
  00000000000000000005d6e8f3c2a9b1e7d4f6c8a2e9b5d3f1c7…

Transactions:

1. Alice → Bob: 0.5 BTC
2. Charlie → Dave: 1.2 BT
3. Eve → Frank: 0.3 BTC

2. Process of Mining

Mining is the process through which new transactions are verified and added to the blockchain. It uses a consensus mechanism called Proof of Work (PoW) to ensure security, integrity, and decentralization of the network. This process requires computational effort and prevents fraudulent activities such as double spending.

Step 1: Collection of Transactions

In the first step, all pending and unconfirmed transactions from the network are collected by miners and grouped together into a new block. These transactions are selected based on their validity and transaction fees.

Example: If Alice sends 50 coins to Bob, this transaction is broadcast to the network and included in a new block by the miner.

Step 2: Creation of Block Header

Once the transactions are collected, the miner creates a block header. This header contains important information such as the hash of the previous block, transaction data, timestamp, and an initial nonce value (usually set to zero). The block header serves as the identity of the new block.

Step 3: Proof of Work (PoW)

In this step, miners compete to solve a complex cryptographic puzzle by repeatedly changing the nonce value. The goal is to generate a hash that meets the network's difficulty requirement, such as having a specific number of leading zeros. This process requires significant computational power, which is why it is called Proof of Work.

Example:
SHA-256(block data + nonce) → 0000ab34cd...

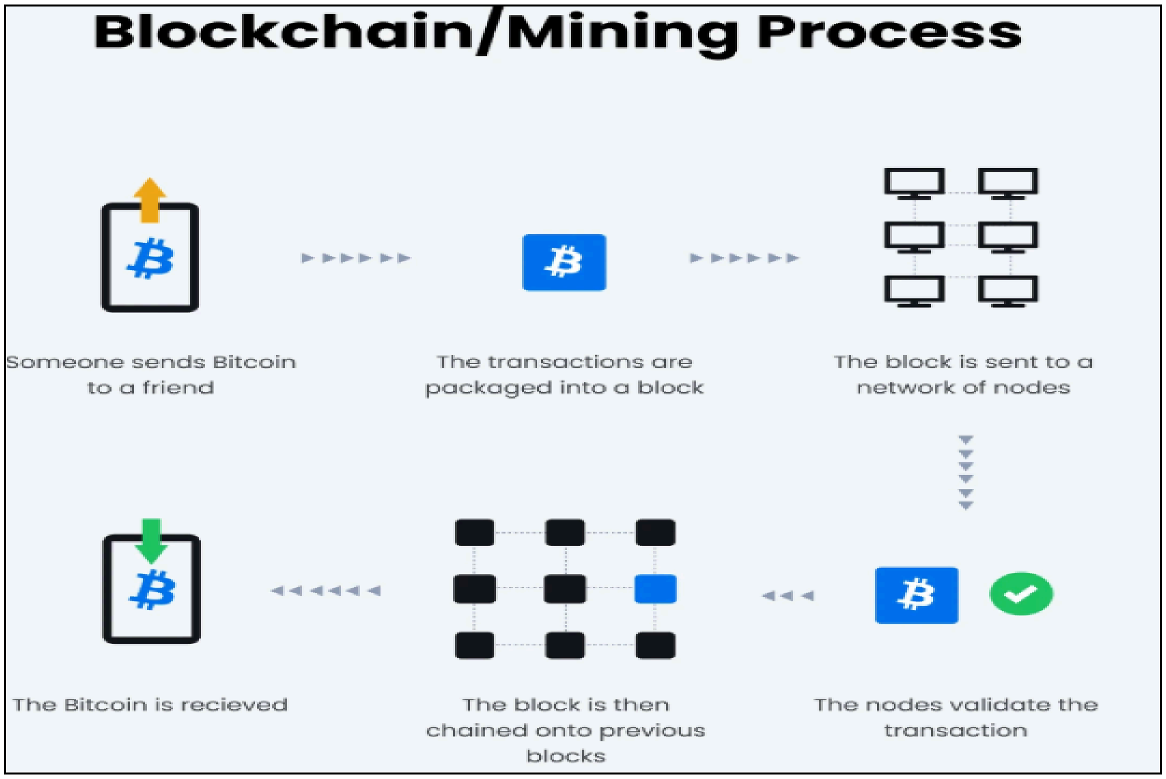
Only when a miner finds such a valid hash is the block considered ready for validation.

Step 4: Block Validation and Broadcasting

After finding a valid hash, the miner broadcasts the newly created block to the entire blockchain network. Other nodes (computers) in the network independently verify the block by checking the correctness of the hash, the validity of the nonce, and the link to the previous block's hash. If any of these are incorrect, the block is rejected.

Step 5: Addition of Block to Blockchain and Reward

If the majority of nodes validate the block successfully, it is permanently added to the blockchain. As a reward for their effort and computational work, the miner receives newly created cryptocurrency along with transaction fees from the included transactions.


Blockchain/Mining Process
Someone sends Bitcoin to a friend
The transactions are packaged into a block
The block is sent to a network of nodes
The nodes validate the transaction
The block is then chained onto previous blocks
The Bitcoin is recieved

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

Block validation is a crucial process that ensures only authentic, correctly formed, and untampered blocks are added to the blockchain. This verification is carried out by all participating nodes in the network, which helps maintain security, integrity, and trust in the system. If a block fails any of the validation checks, it is immediately rejected by the network.

1. Verify Block Hash

In this step, the hash of the block is recalculated using the block's header information, including transaction data, timestamp, previous hash, and nonce. This newly computed hash is then compared with the hash already stored in the block. If both values match, it confirms that the block has not been altered.

Example:
If the calculated hash is 0000ab45cd and it matches the stored hash, the block passes this validation step.

2. Check Proof of Work (PoW)

The next step is to verify whether the block satisfies the network's difficulty requirement. This means checking whether the block's hash contains the required number of leading zeros as defined by the blockchain protocol. This ensures that sufficient computational work was performed during mining.

Example:
If the required format is 0000xxxx, and the block hash begins with 0000f9a2, then the Proof of Work is considered valid.

3. Validate Previous Block Hash

Each block contains the hash of the previous block in the chain. Validators check whether this recorded previous hash exactly matches the actual hash of the last block in the blockchain. This step ensures that the new block is correctly linked to the existing chain and that no tampering has occurred in earlier blocks.

Example:
If the previous block hash stored in the new block is 0000a1b2c3, and this matches the actual hash of the last block, the validation is successful.

4. Verify Transactions in the Block

All transactions included in the block are carefully examined to ensure they are legitimate. Validators check whether the sender has sufficient balance, whether the transaction is properly signed using a digital signature, and whether there is any attempt at double spending.

Example:
If Alice tries to send coins to Bob, the system first confirms that Alice actually has enough balance to complete the transaction. If she does not, the block is rejected.

5. Verify Merkle Root

The Merkle Root is a single hash that represents all transactions in the block. In this step, validators recompute the Merkle Root using the transaction data and compare it with the Merkle Root stored in the block header. If both values match, it confirms that none of the transactions have been modified.

Example:
If the recalculated Merkle Root matches the one in the block header, the block passes this check.

6. Check Timestamp Validity

The timestamp of the block is verified to ensure that it represents a realistic creation time and is not set in the future. This prevents miners from manipulating time records for unfair advantage.

Example:
If the timestamp shows 2026-01-29 10:30 AM, and this is within an acceptable time range, the block is considered valid.

7. Check Block Size and Format

Every blockchain has predefined rules regarding block size and structure. Validators ensure that the new block follows these protocol rules. If the block is too large, improperly formatted, or violates any structural requirement, it is rejected.

8. Validate Genesis Block

The Genesis Block is the very first block in the blockchain and is hardcoded into the system. It has no previous block, so its previous hash is always set to “0.” Validators ensure that this block has not been altered, as it forms the foundation of the entire blockchain.

Final Result

If all the above checks are successfully passed, the block is considered valid and is permanently added to the blockchain.

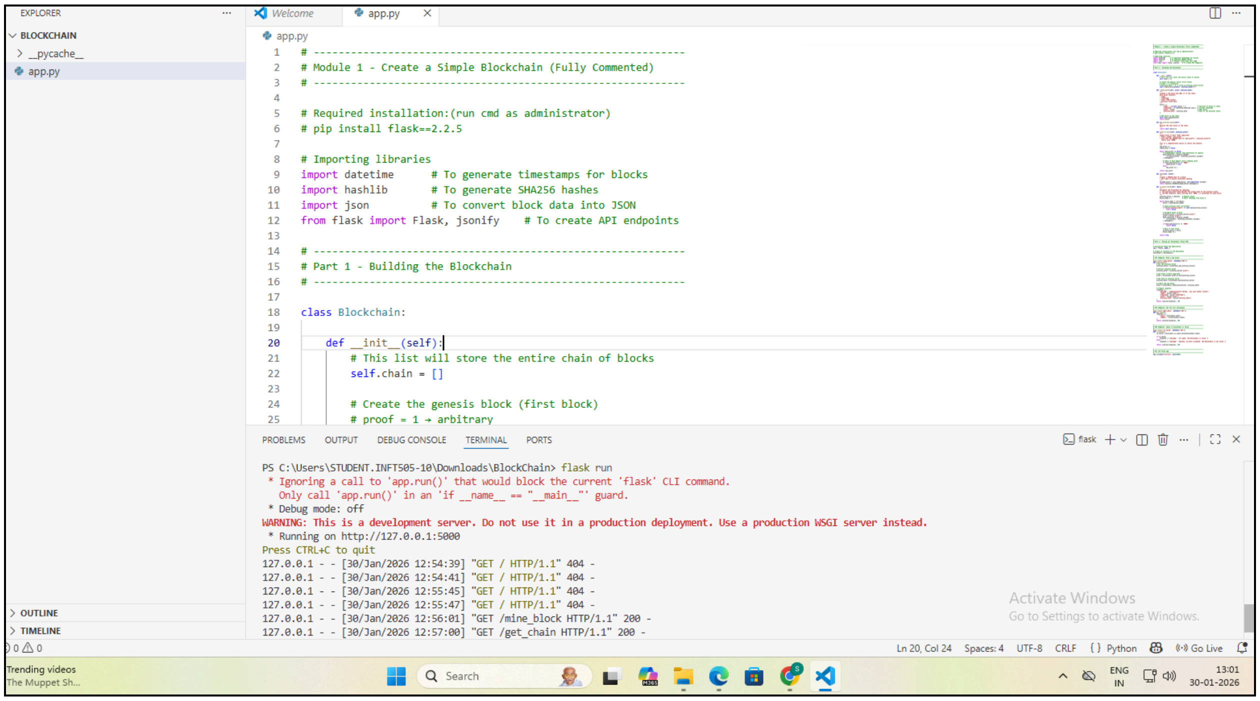
If any of these checks fail, the block is immediately rejected by the network to maintain security and integrity.

Code:

```
# Importing libraries
import datetime
import hashlib
import json
from flask import Flask, jsonify
```

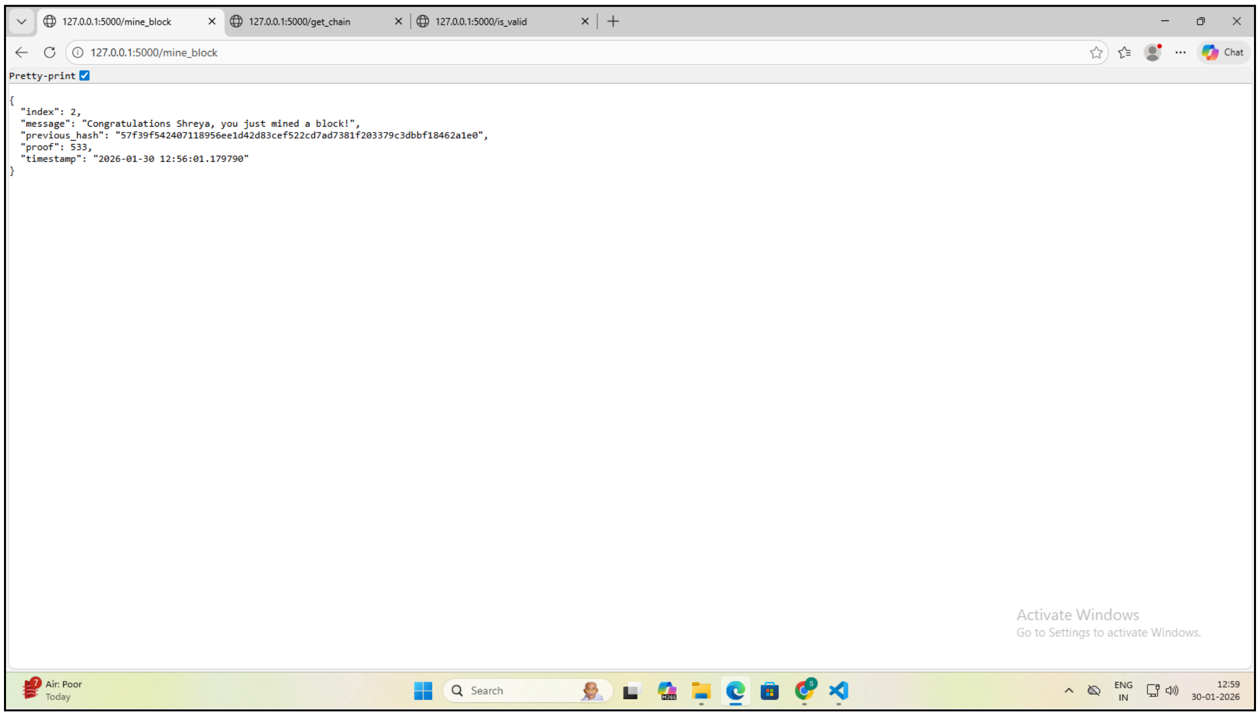
```
def __init__(self):

        self.chain = []

          # Create the genesis block (first block)
        # proof = 1 → arbitrary
        # previous_hash = '0' → since no
previous block exists
        self.create_block(proof=1,
previous_hash='0')

    def create_block(self, proof,
previous_hash):
             block = {
            'index': len(self.chain) + 1,
            'timestamp':
str(datetime.datetime.now()), # Current
timestamp
            'proof': proof,
'previous_hash': previous_hash
        }

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

    def get_previous_block(self):
               return self.chain[-1]

    def proof_of_work(self, previous_proof):
              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):
            encoded_block = json.dumps(block,
sort_keys=True).encode()
        return
hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):

        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)

    # 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:



```python
# ------------------------------------------------------------
# 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
```

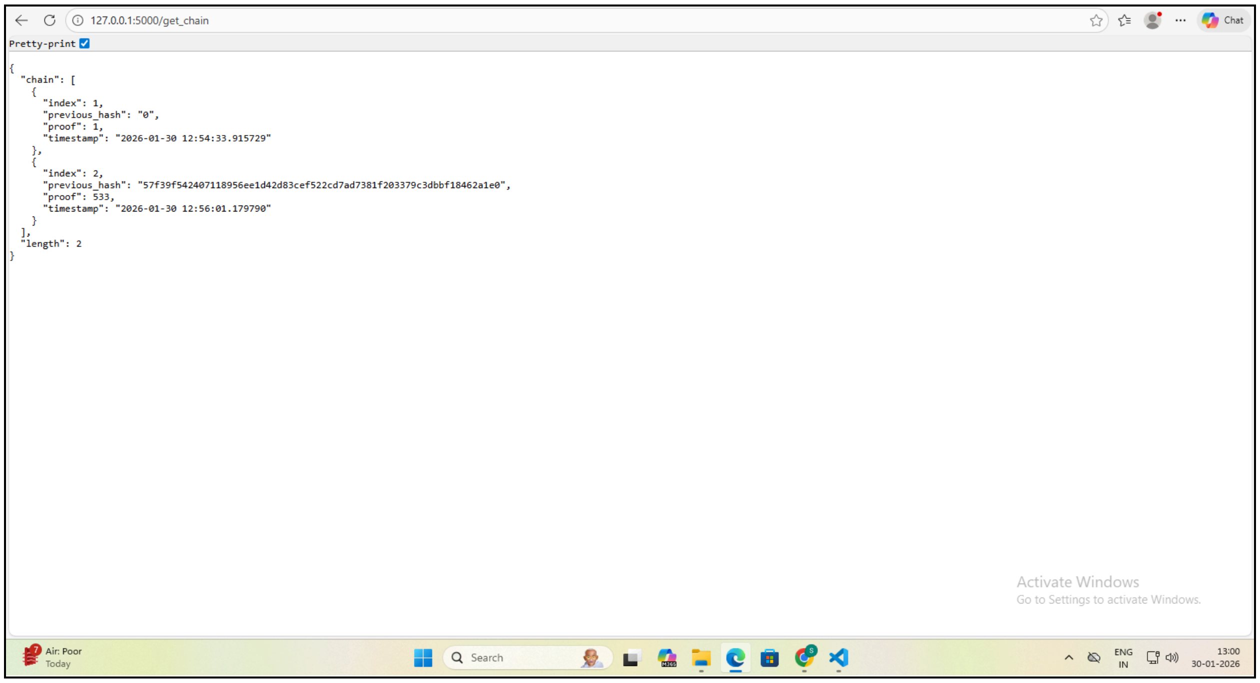
```
PS C:\Users\STUDENT.INFT505-10\Downloads\BlockChain> flask run
 * Ignoring a call to 'app.run()' that would block the current 'flask' CLI command.
   Only call 'app.run()' in an 'if __name__ == "__main__"' guard.
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production 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 -
```



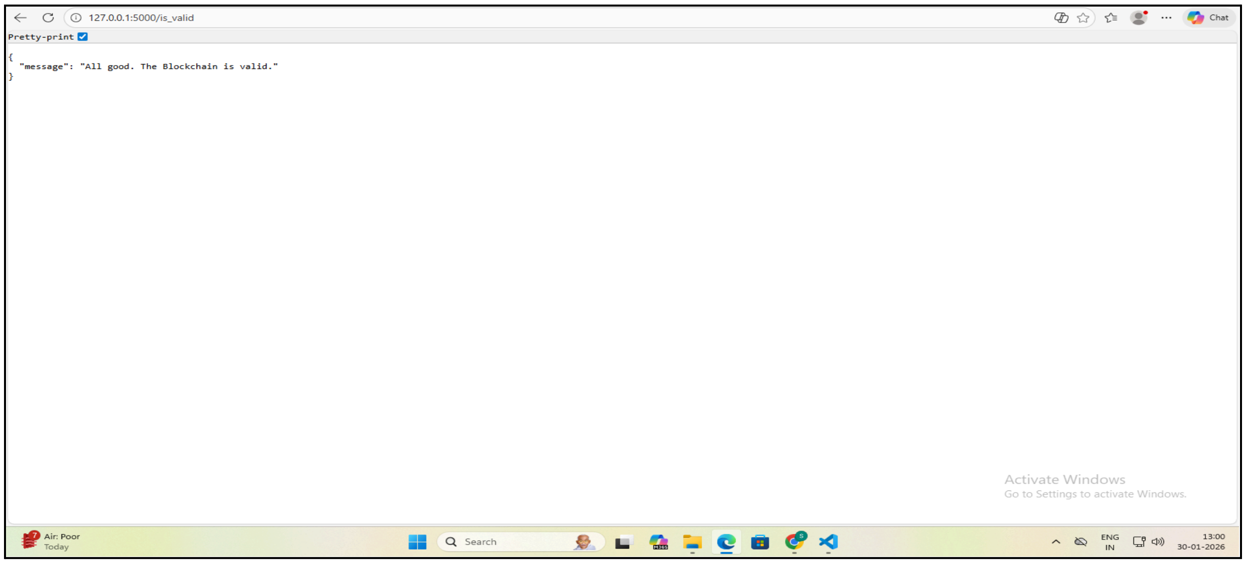
127.0.0.1:5000/mine_block

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"
}
```



```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-01-30 12:54:33.915729"
    },
    {
      "index": 2,
      "previous_hash": "57f39f542407118956ee1d42d83cef522cd7ad7381f203379c3dbbf18462a1e0",
      "proof": 533,
      "timestamp": "2026-01-30 12:56:01.179790"
    }
  ],
  "length": 2
}
```



```
{
  "message": "All good. The Blockchain is valid."
}
```

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.