Ansh Sarfare **D20A - 51**

Experiment -2

Aim: Create a Blockchain using Python

Theory:

1. What is Blockchain

Blockchain is an innovative technology that works as a shared, tamper-resistant digital ledger. Its name comes from the way data is structured — information is stored in blocks, each connected to the previous one, forming a continuous chain.

Each block contains important information such as a list of transactions, a timestamp, and a cryptographic hash, which uniquely identifies the block. The hash is generated from both the block's content and the previous block's hash, which links the blocks together securely.

- This linked design ensures that any attempt to alter data becomes immediately noticeable, as it breaks the chain.
- Blockchain serves as a distributed database, storing transaction data across all network participants.
- Transactions are verified collectively by the network, maintaining legitimacy and trust.
- Because it is decentralized, no single entity can manipulate the data.

In a blockchain network, multiple computers (called nodes) hold copies of the ledger. Once a transaction is added and confirmed, it becomes practically immutable, meaning it cannot be changed or deleted without consensus from the network.

2. What is a Block in Blockchain

A block is a digital container that securely stores verified transactions. It is a key building block of the blockchain. Each block contains:

- Transaction Data – details of the transaction such as sender, receiver, and amount.
- Timestamp – records the exact creation time of the block.
- Unique Hash – a cryptographic fingerprint of the block, ensuring its integrity.

- Previous Block Hash – links the block to the one before it, maintaining an unbroken chain.

By storing this information, blocks create a chronological and transparent record of all transactions across the network.

3. Components of a Blockchain Block

Key Components Include:

- Transaction Data: Captures details of who made the transaction, what was exchanged, and when.
- Timestamp: Marks when the block was created, maintaining chronological order.
- Hash: A unique identifier for the block, produced from its contents. Any change in data changes the hash.
- Previous Hash: Connects this block to the preceding one, forming a secure chain.

4. What is Mining

Mining is the process of validating transactions and adding new blocks to the blockchain. It is crucial for network security, decentralization, and transparency.

In cryptocurrencies like Bitcoin, mining uses Proof of Work (PoW), where miners solve complex computational puzzles. Miners compete to validate transactions and append blocks, receiving rewards for success.

Steps in Mining Process

4.1 Transaction Initiation

- Users submit transactions to the blockchain network.
- These transactions are broadcast to all nodes.

4.2 Transaction Pool (Mempool)

- Unconfirmed transactions are collected in a temporary area called the mempool.
- Miners select transactions from this pool to include in new blocks.

4.3 Block Creation

A new block is formed containing:

- Selected transactions
- Hash of the previous block
- Timestamp
- Nonce (a random number used for hashing)

4.4 Hash Generation

- Miners compute a cryptographic hash (e.g., SHA-256) of the block.
- The hash must satisfy the network's difficulty rules, such as starting with a certain number of zeros.

4.5 Proof of Work

- Miners repeatedly adjust the nonce and recompute the hash until a valid hash is found.
- The first miner to find a valid hash proves they have completed the PoW.

4.6 Block Verification

- The mined block is broadcast to all network nodes.
- Nodes verify:
  - Hash correctness
  - Transaction authenticity
  - Proper block structure

4.7 Adding Block to Blockchain

- Once verified, the block is permanently added to the chain.
- The chain remains tamper-proof and synchronized across all nodes.

4.8 Reward Distribution

- The miner who successfully adds the block receives:
    - New coins (block reward)
    - Transaction fees from transactions in the block

5. How to Check Block Validity

Block validity ensures that a block adheres to all blockchain rules before it is accepted. Key checks include:

- Block Structure Verification: Confirm the block has all required elements like header, previous hash, timestamp, nonce, and transactions.
- Previous Hash Validation: Compare the block's previous hash with the hash of the latest block to maintain the chain's integrity.
- Hash and Proof of Work Verification: Recompute the block hash and ensure it meets the network's difficulty conditions.
- Transaction Validation: Ensure every transaction has valid digital signatures, sufficient balance, and no double-spending.
- Merkle Root Verification: Recalculate the Merkle root from all transactions and compare it with the stored root.
- Timestamp Verification: Ensure the block's timestamp is valid and sequential relative to the previous block.
- Block Size and Rule Compliance: Check that the block follows size limits and other protocol rules.
- Consensus Rule Verification: Ensure the block adheres to the network's consensus mechanisms, confirming it is legitimate.

Program:

```
# 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
        # 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):
```

```
    Return 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
# Part 2 - Mining our Blockchain (Flask API)
# Initialize Flask web application
app = Flask(__name__)

# Create an instance of the Blockchain
```

```
blockchain = Blockchain()
# API Endpoint: Mine a new block
@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, 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 the full blockchain
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
```

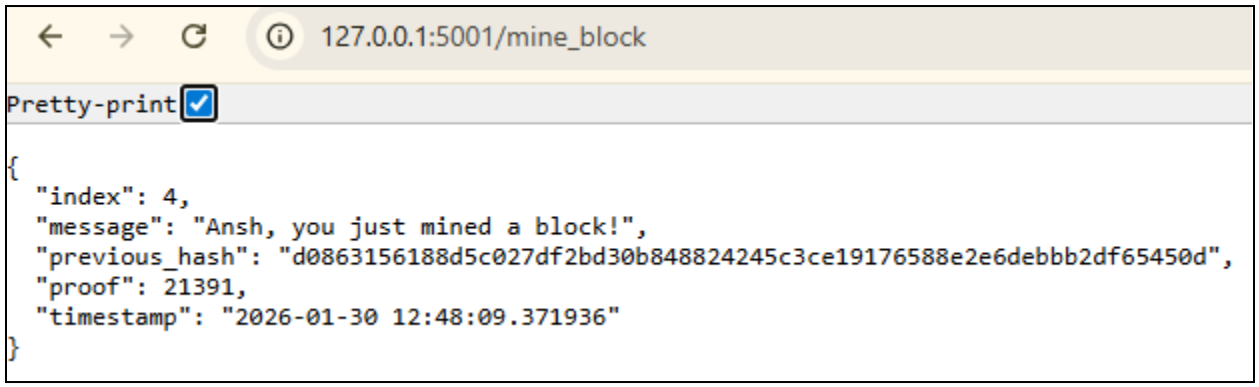
```
    return jsonify(response), 200
# API Endpoint: Check if blockchain is valid
@app.route('/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}
    return jsonify(response), 200
# Run the Flask app
app.run(host='0.0.0.0', port=5000)
```

Output:

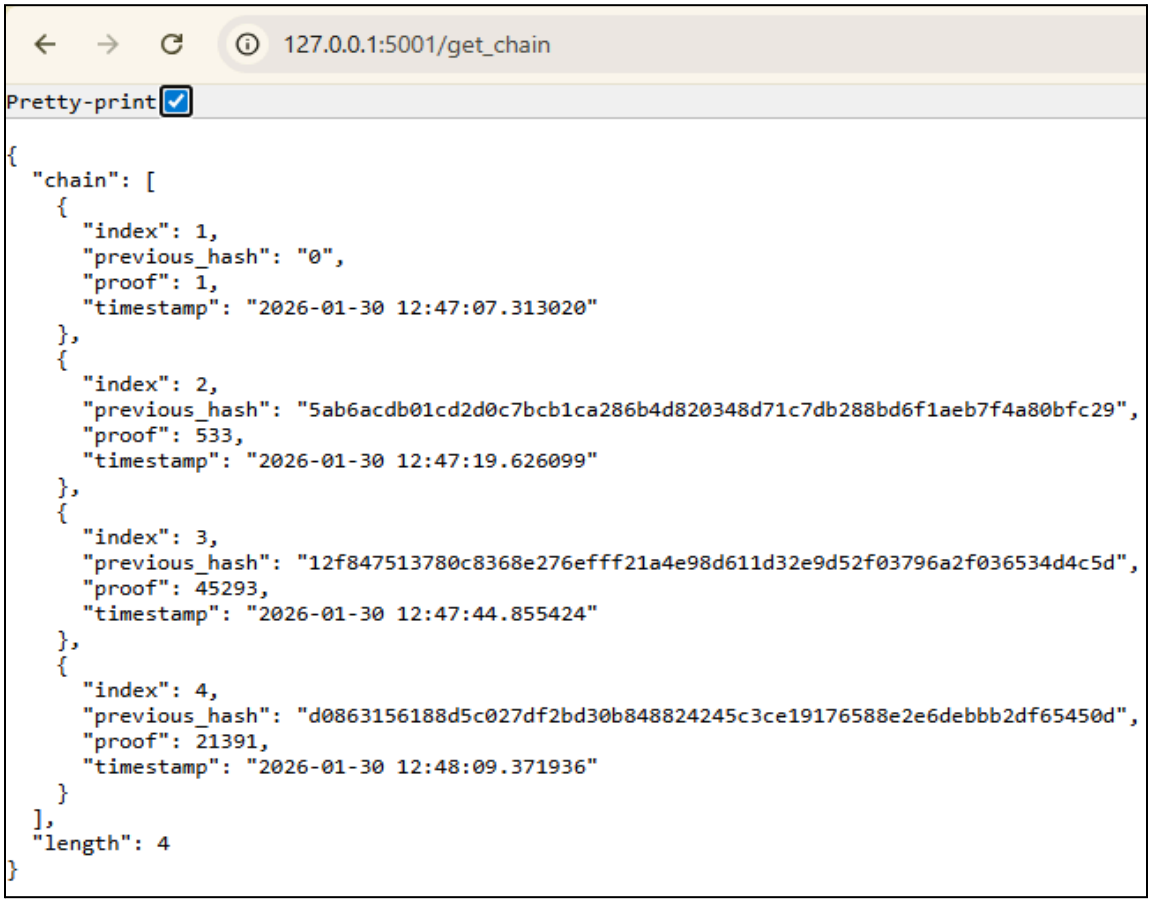
```
C:\Users\STUDENT.INFT505-03\Desktop>cd blockchain lab

C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>python -m venv venv

C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>venv\Scripts\activate.bat

(venv) C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>pip install flask==2.2.5
Collecting flask==2.2.5
  Downloading Flask-2.2.5-py3-none-any.whl.metadata (3.9 kB)
Collecting Werkzeug>=2.2.2 (from flask==2.2.5)
  Downloading werkzeug-3.1.5-py3-none-any.whl.metadata (4.0 kB)
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip

(venv) C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>
(venv) C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\student.inft505-03\desktop\blockchain lab\venv\lib\site-packages (25.1.1)
Collecting pip
  Using cached pip-25.3-py3-none-any.whl.metadata (4.7 kB)
Using cached pip-25.3-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.1.1
    Uninstalling pip-25.1.1:
      Successfully uninstalled pip-25.1.1
Successfully installed pip-25.3

(venv) C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>python blockchain.py
```

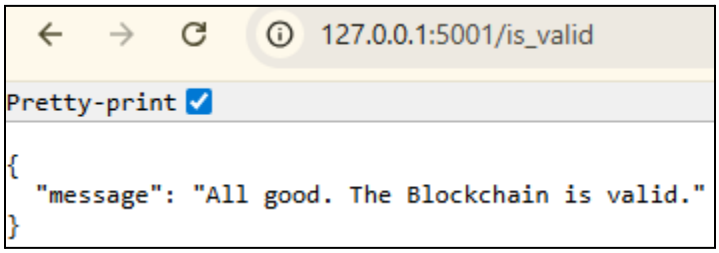
```
(venv) C:\Users\STUDENT.INFT505-03\Desktop\blockchain lab>python blockchain.py
 * Serving Flask app 'blockchain'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.32.230:5001
Press CTRL+C to quit
127.0.0.1 - - [30/Jan/2026 12:47:19] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:47:39] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:47:44] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:47:47] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:48:04] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:48:09] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:48:11] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:48:25] "GET /is_valid HTTP/1.1" 200 -
```




```
{
  "index": 4,
  "message": "Ansh, you just mined a block!",
  "previous_hash": "d0863156188d5c027df2bd30b848824245c3ce19176588e2e6debbb2df65450d",
  "proof": 21391,
  "timestamp": "2026-01-30 12:48:09.371936"
}
```


```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-01-30 12:47:07.313020"
    },
    {
      "index": 2,
      "previous_hash": "5ab6acdb01cd2d0c7bcb1ca286b4d820348d71c7db288bd6f1aeb7f4a80bfc29",
      "proof": 533,
      "timestamp": "2026-01-30 12:47:19.626099"
    },
    {
      "index": 3,
      "previous_hash": "12f847513780c8368e276efff21a4e98d611d32e9d52f03796a2f036534d4c5d",
      "proof": 45293,
      "timestamp": "2026-01-30 12:47:44.855424"
    },
    {
      "index": 4,
      "previous_hash": "d0863156188d5c027df2bd30b848824245c3ce19176588e2e6debbb2df65450d",
      "proof": 21391,
      "timestamp": "2026-01-30 12:48:09.371936"
    }
  ],
  "length": 4
}
```



Conclusion

We successfully created a basic blockchain using Python and Flask that supports block mining, chain retrieval, and validity checks. Proof-of-Work ensures security by making block creation computationally intensive, while cryptographic hashing guarantees immutability. The implementation demonstrates core blockchain principles decentralization, immutability, and consensus in a simplified environment, providing a strong foundation for building more advanced blockchain systems.