Shreyash G. Dhekane　　　　*D20A*　　　　*Roll No.23*

Blockchain Exp 2

Aim:- Create a blockchain in Python

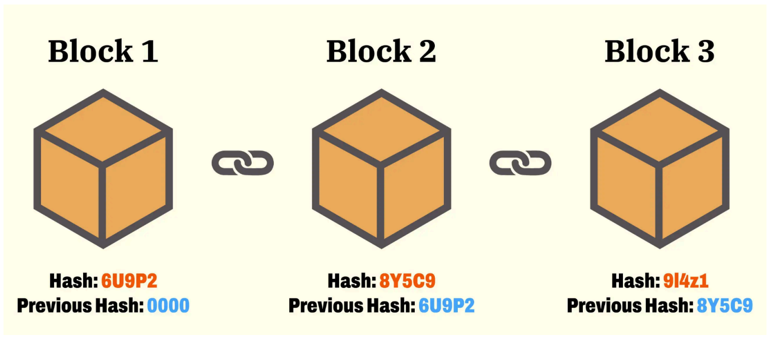
1. What is a Blockchain ?

A blockchain is a distributed, decentralized digital ledger that is used to record transactions in a secure, transparent, and permanent manner. Unlike traditional databases that are controlled by a central authority, a blockchain operates over a peer-to-peer network where every participant (node) maintains a copy of the ledger. This eliminates the need for intermediaries and increases trust among participants.

The data in a blockchain is stored in blocks, and each block contains transaction data, a timestamp, and the cryptographic hash of the previous block. These blocks are linked together sequentially, forming a chain. Because each block depends on the hash of the previous one, modifying data in any block would require changing all subsequent blocks, which is practically impossible. This property makes blockchain tamper-resistant and immutable.

Main characteristics of blockchain include:

- Decentralization: Control is distributed among network nodes instead of a single authority
- Immutability: Once data is recorded, it cannot be altered or deleted
- Transparency: All transactions are visible to authorized participants
- Security: Cryptographic techniques protect data from unauthorized changes
- Trustless system: Trust is achieved through consensus, not intermediaries



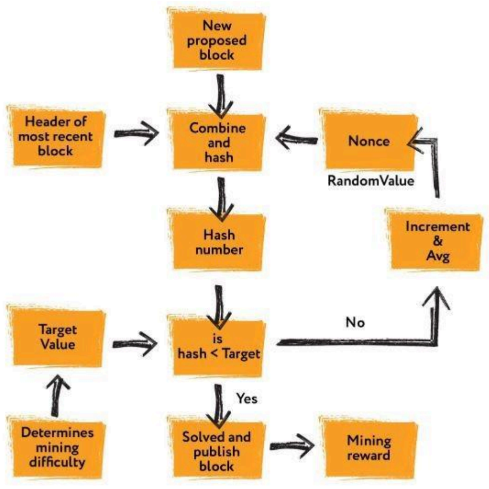
2. Process of Mining a Blockchain

Mining is the process through which new blocks are added to the blockchain and transactions are verified. It plays a crucial role in maintaining the security and integrity of the blockchain network. Mining is commonly associated with blockchains that use the Proof of Work (PoW) consensus mechanism.

In this process, miners collect pending or unconfirmed transactions from the network and organize them into a candidate block. The miner then tries to find a special value called a nonce that, when combined with the block data and hashed, produces an output that satisfies the network's difficulty requirement. This requires extensive computational power and repeated trial-and-error.

Steps involved in the mining process:

- Collection of unconfirmed transactions from the network
- Creation of a new block containing transactions and previous block hash
- Repeated hashing with different nonce values
- Finding a hash that meets the required difficulty target
- Broadcasting the newly mined block to the network
- Verification of the block by other nodes
- Rewarding the successful miner with cryptocurrency



3. How to Check the Validity of a Block in Blockchain ?

Before a block is accepted into the blockchain, it must undergo a validation process to ensure its correctness and compliance with network rules. Block validation is performed by network nodes to maintain consistency, security, and trust across the blockchain.

The first step in validation is checking whether the block's hash is correctly computed using the block's data and nonce. The hash must also satisfy the difficulty level defined by the network. If the hash does not meet this requirement, the block is considered invalid.

Key checks performed during block validation:

- Verification of the block hash and difficulty criteria
- Matching the previous block's hash with the blockchain history
- Validation of all transactions inside the block
- Ensuring no double spending has occurred
- Confirming the block follows consensus rules

If all these conditions are satisfied, the block is accepted and added to the blockchain. If any rule is violated, the block is rejected, protecting the blockchain from fraud and manipulation.

<u>Code</u>:-

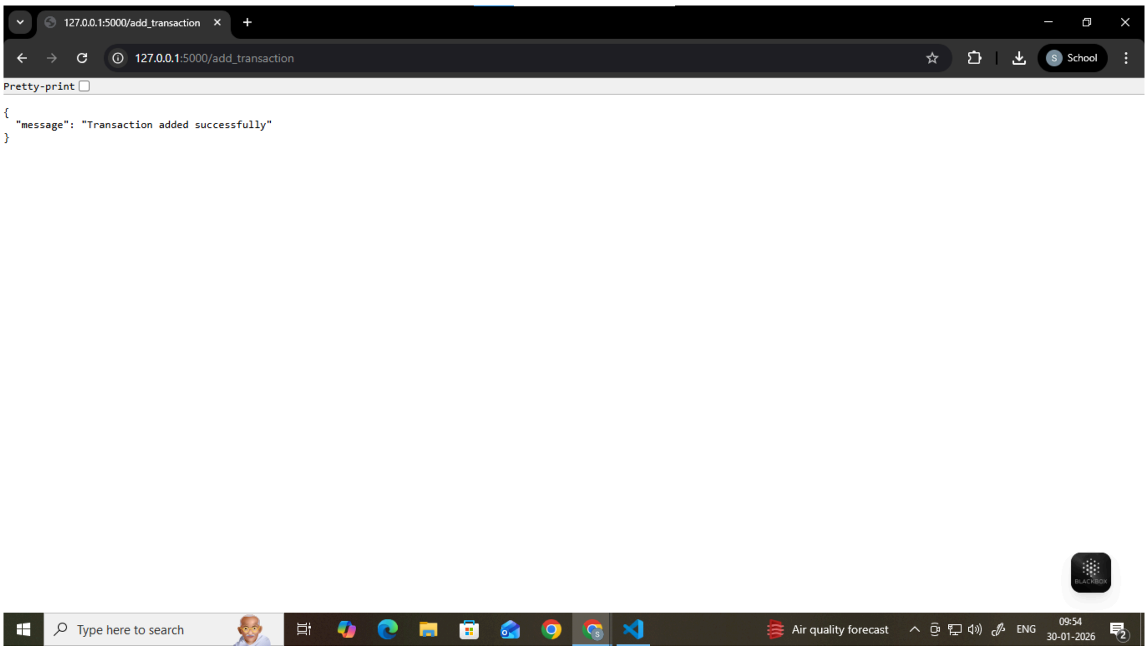
`blockchain.py`

```
import datetime
import hashlib
import json

class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
# Transaction list
        self.difficulty = 3
# "000" leading zeros
        self.create_genesis_block()

    def create_genesis_block(self):
        genesis_block = {
            'index': 1,
            'timestamp':
str(datetime.datetime.now()),
            'transactions': [],
            'nonce': 0,
            'previous_hash': '0'
        }
        genesis_block['hash'] =
self.calculate_hash(genesis_block)

self.chain.append(genesis_block)

    def create_Transactions(self,
sender, receiver, amount):
        transaction = {
            'sender': sender,
            'receiver': receiver,
            'amount': amount
        }

self.transactions.append(transaction)

    def calculate_hash(self, block):
        block_copy = block.copy()
        block_copy.pop('hash', None)
        encoded_block =
json.dumps(block_copy,
sort_keys=True).encode()
        return
hashlib.sha256(encoded_block).hexdiges
t()

    def proof_of_work(self, block):
        nonce = 0
        while True:
            block['nonce'] = nonce
            hash_value =
self.calculate_hash(block)
            if
hash_value.startswith('0' *
self.difficulty):
                return nonce,
hash_value
            nonce += 1

    def mine_block(self):

        if len(self.transactions) ==
0:
            return None   # Mine only
if transactions exist

        previous_block =
self.chain[-1]

        block = {
            'index': len(self.chain) +
1,
            'timestamp':
str(datetime.datetime.now()),
            'transactions':
self.transactions,
            'nonce': 0,
            'previous_hash':
previous_block['hash']
        }

        nonce, hash_value =
self.proof_of_work(block)
        block['nonce'] = nonce
        block['hash'] = hash_value

        self.transactions = []
        self.chain.append(block)
        return block

    def is_chain_valid(self):
        for i in range(1,
len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i -
1]

            if
current['previous_hash'] !=
previous['hash']:
                return False

            recalculated_hash =
self.calculate_hash(current)
            if recalculated_hash !=
current['hash']:
                return False

            if not
current['hash'].startswith('0' *
self.difficulty):
                return False

        return Tru
```

app.py

```
from flask import Flask, jsonify
from blockchain import Blockchain

app = Flask(__name__)
blockchain = Blockchain()

@app.route('/add_transaction',
methods=['GET'])
def add_transaction():

blockchain.create_Transactions("Alice"
, "Bob", 50)
    return jsonify({'message':
'Transaction added successfully'}),
200

@app.route('/mine_block',
methods=['GET'])
def mine_block():
    block = blockchain.mine_block()
    if block is None:
        return jsonify({'message': 'No
transactions to mine'}), 400
    return jsonify({
        'message': 'Block mined
successfully!',
        'block': block
    }), 200

@app.route('/get_chain',
methods=['GET'])
```

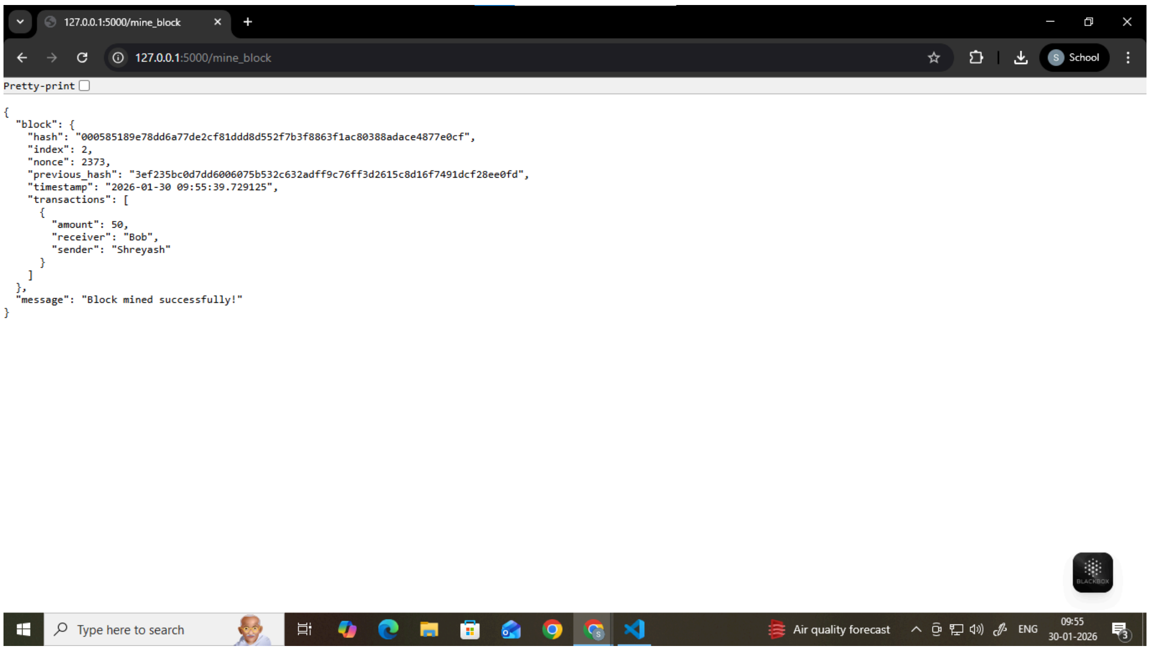
```
def get_chain():
    return jsonify({
        'chain': blockchain.chain,
        'length':
len(blockchain.chain)
    }), 200

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

if __name__ == '__main__':
    print(" Blockchain running with 3
leading zero difficulty")
    app.run(debug=True)
```



127.0.0.1:5000/add_transaction

Pretty-print

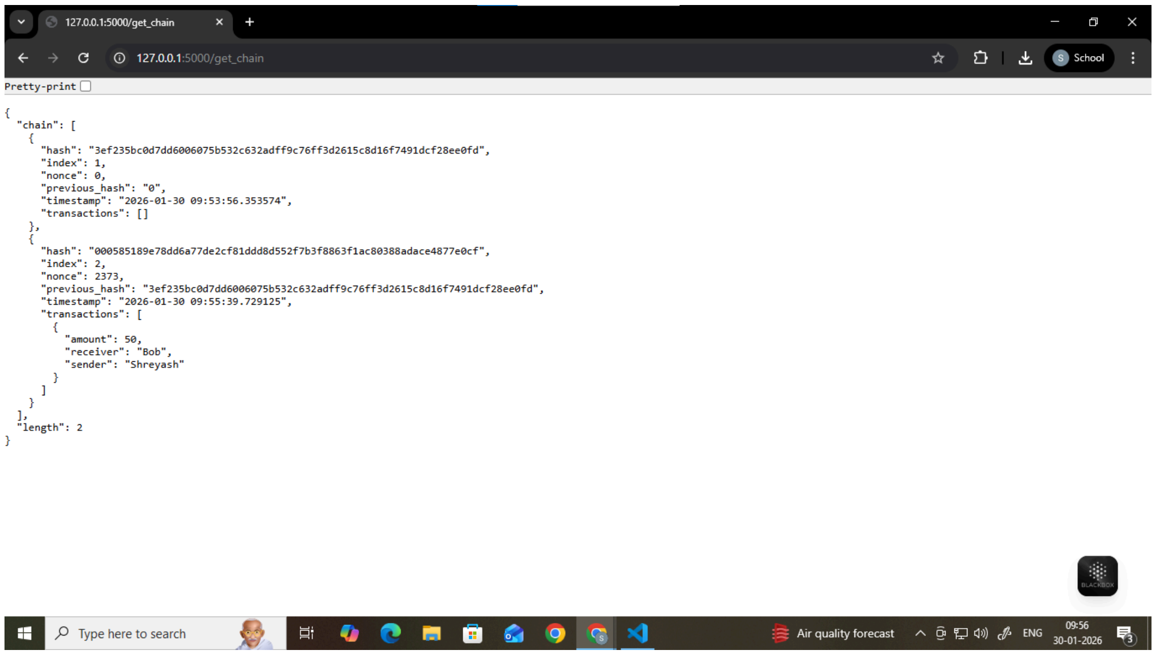
```
{
  "message": "Transaction added successfully"
}
```



127.0.0.1:5000/mine_block

Pretty-print

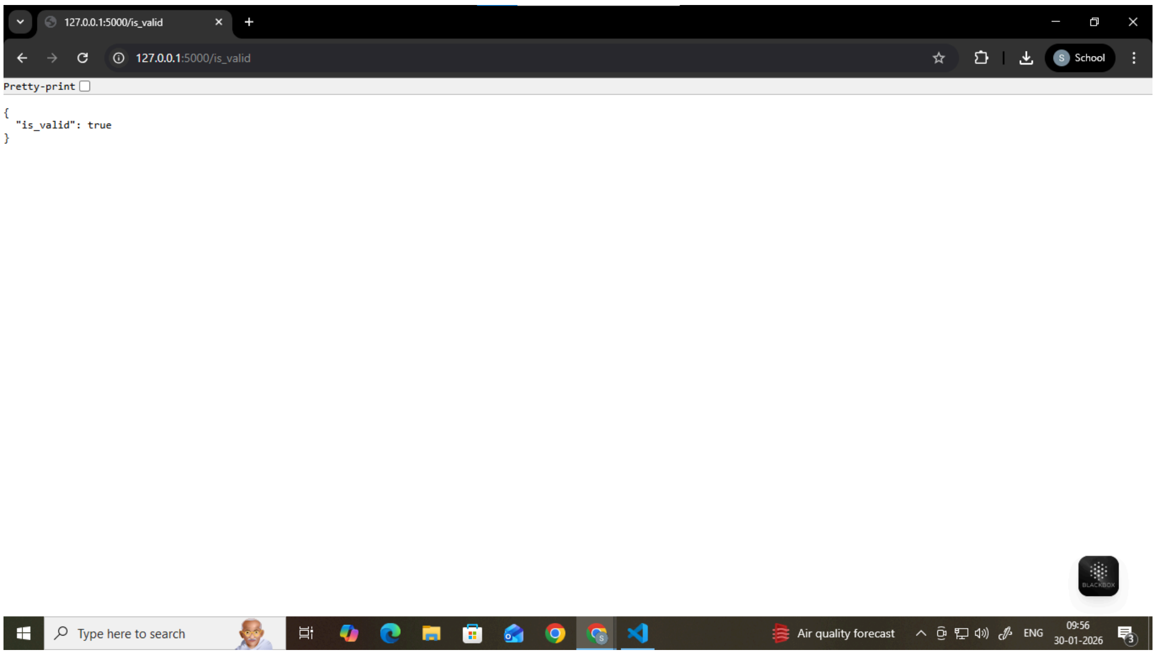
```
{
  "block": {
    "hash": "000585189e78dd6a77de2cf81ddd8d552f7b3f8863f1ac80388adace4877e0cf",
    "index": 2,
    "nonce": 2373,
    "previous_hash": "3ef235bc0d7dd6006075b532c632adff9c76ff3d2615c8d16f7491dcf28ee0fd",
    "timestamp": "2026-01-30 09:55:39.729125",
    "transactions": [
      {
        "amount": 50,
        "receiver": "Bob",
        "sender": "Shreyash"
      }
    ]
  },
  "message": "Block mined successfully!"
}
```



127.0.0.1:5000/get_chain

Pretty-print

```
{
  "chain": [
    {
      "hash": "3ef235bc0d7dd6006075b532c632adff9c76ff3d2615c8d16f7491dcf28ee0fd",
      "index": 1,
      "nonce": 0,
      "previous_hash": "0",
      "timestamp": "2026-01-30 09:53:56.353574",
      "transactions": []
    },
    {
      "hash": "000585189e78dd6a77de2cf81ddd8d552f7b3f8863f1ac80388adace4877e0cf",
      "index": 2,
      "nonce": 2373,
      "previous_hash": "3ef235bc0d7dd6006075b532c632adff9c76ff3d2615c8d16f7491dcf28ee0fd",
      "timestamp": "2026-01-30 09:55:39.729125",
      "transactions": [
        {
          "amount": 50,
          "receiver": "Bob",
          "sender": "Shreyash"
        }
      ]
    }
  ],
  "length": 2
}
```



127.0.0.1:5000/is_valid

Pretty-print

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

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.