Name: Prasad Jawale **Class**: D16AD **Roll**: 20

Blockchain Lab 2

Aim: To create a blockchain using python

Theory: Creating a blockchain from scratch using Python involves implementing the core principles and components of a blockchain system. Here's a theoretical overview of the key elements you'll need to understand when building a basic blockchain:

- Blocks and Chain: A blockchain is a distributed ledger composed of a chain of blocks. Each block contains a set of transactions and a reference (usually a hash) to the previous block in the chain, forming a linked list. The linking ensures the chronological order of transactions and adds security through immutability.
- Transactions: Transactions represent the data being stored on the blockchain. In a cryptocurrency blockchain like Bitcoin, transactions include sender and receiver addresses, transaction amount, digital signatures, and other relevant information. For non-cryptocurrency blockchains, the content of transactions can vary widely.
- Hash Functions: Cryptographic hash functions like SHA-256 are used to create unique, fixed-size hashes for blocks and transactions. Hashes are essential for data integrity and linking blocks together. Any change in the block's content results in a completely different hash.
- Mining and Proof of Work: Mining is the process by which new blocks are added to the blockchain. Miners (computers or nodes) compete to solve a computationally intensive mathematical puzzle, often called Proof of Work (PoW), to create a new block. The first miner to solve the puzzle broadcasts the new block to the network, and other nodes verify its validity.

- Consensus Mechanism: Blockchain networks use a consensus mechanism to agree on the validity of transactions and blocks. PoW, Proof of Stake (PoS), and Delegated Proof of Stake (DPoS) are common consensus mechanisms. PoW ensures that miners put in computational work to validate transactions and create new blocks.
- Decentralization: One of the core principles of a blockchain is decentralization, meaning that no single entity or authority has control over the network. Nodes (computers) in the network maintain copies of the blockchain and collectively validate transactions and maintain the ledger.
- Security: Security is paramount in blockchain. Cryptographic techniques are used for secure transactions and block validation. Private and public keys, digital signatures, and encryption are used to protect data.
- Peer-to-Peer Networking: Blockchain networks are typically decentralized and rely on peer-to-peer networking protocols. Nodes communicate with each other to propagate transactions and blocks, ensuring network integrity.
- Smart Contracts (Optional): Some blockchains, like Ethereum, support smart contracts, which are self-executing code that runs on the blockchain. Smart contracts enable programmable and automated transactions and are written in languages like Solidity.
- Testing and Validation: Extensive testing and validation are necessary to ensure that the blockchain operates correctly and securely. This includes testing the consensus mechanism, validating transactions, and checking the overall network health.

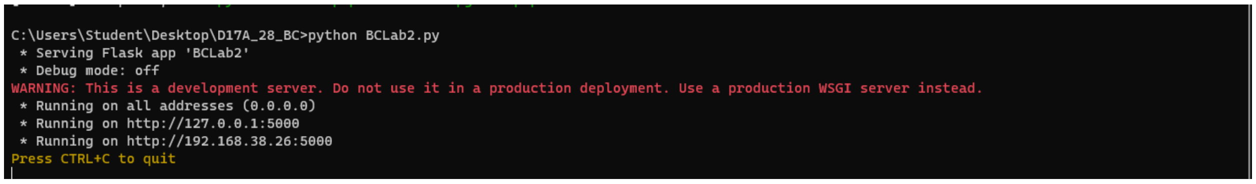
Creating a blockchain from scratch can be a complex and time-consuming task, but it provides valuable insights into blockchain technology. You can start by implementing a basic blockchain with the above principles in mind and gradually add more features and complexity as you become more comfortable with the concepts. There are also open-source blockchain libraries and frameworks available in Python that can help streamline the development process.

Program:

```
import datetime
import hashlib
import json
from flask import Flask, jsonify, request

def build_merkle_tree(transactions):
    if len(transactions) == 0:
        return None

    if len(transactions) == 1:
        return transactions[0]

    # Recursive construction of the Merkle Tree
    while len(transactions) > 1:
        if len(transactions) % 2 != 0:
            transactions.append(transactions[-1])

        new_transactions = []
        for i in range(0, len(transactions), 2):
            combined = transactions[i] + transactions[i+1]
            hash_combined = hashlib.sha256(
                    combined.encode('utf-8')).hexdigest()
            new_transactions.append(hash_combined)

        transactions = new_transactions

    return transactions[0]


class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
        self.mine_genesis_block()


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


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

    def remove_transactions(self):
        self.transactions = []

    def proof_of_work(self, block):
        block['proof'] = 1
        while True:
            encoded_block = json.dumps(block, sort_keys = True).encode()
            hash_operation = hashlib.sha256(encoded_block).hexdigest()
            if hash_operation.startswith("0000"):
                self.chain.append(block)
                self.remove_transactions()
                return block['proof']
            block['proof'] += 1


    def hash(self, previous_block):
        encoded_block = json.dumps(previous_block, sort_keys = True).encode()
        return hashlib.sha256(encoded_block).hexdigest()


    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            hash_operation = hashlib.sha256(json.dumps(block, sort_keys =
True).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True

    def mine_genesis_block(self):
        merkle_root = build_merkle_tree(self.transactions)
        genesis_block = self.create_block(previous_hash='0', merkle_root=merkle_root,
block_transactions=self.transactions)
        self.proof_of_work(genesis_block)

# Creating a Web App
app = Flask(__name__)

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_hash = blockchain.hash(previous_block)

    merkle_root = build_merkle_tree(blockchain.transactions)
    block = blockchain.create_block(previous_hash, merkle_root,blockchain.transactions)

    blockchain.proof_of_work(block)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'merkle_root': block['merkle_root'],
                'transactions':block['transactions']}
    return jsonify(response), 200

# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200
```

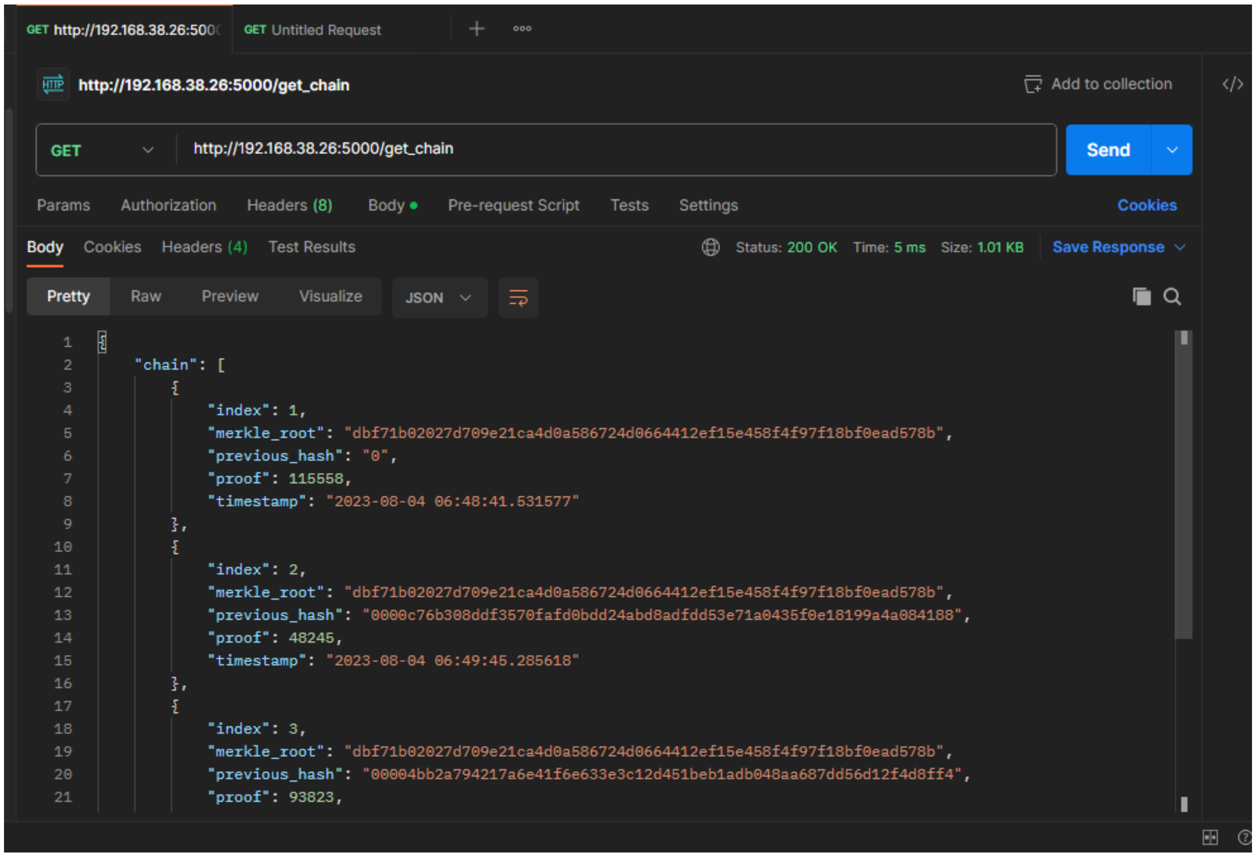
```
# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have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

@app.route('/add_transaction', methods = ['POST'])
def add_transaction():
    data = request.get_json()
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in data for key in transaction_keys):
        return 'Some elements of the transaction are missing', 400

    sender = data['sender']
    receiver = data['receiver']
    amount = data['amount']
    current_transaction = f'{sender} -> {receiver}: ${amount}'
    blockchain.transactions.append(current_transaction)

    return jsonify('Transaction added'), 200

# Running the app
app.run(host = '0.0.0.0', port = 5000)
```

Starting flask server

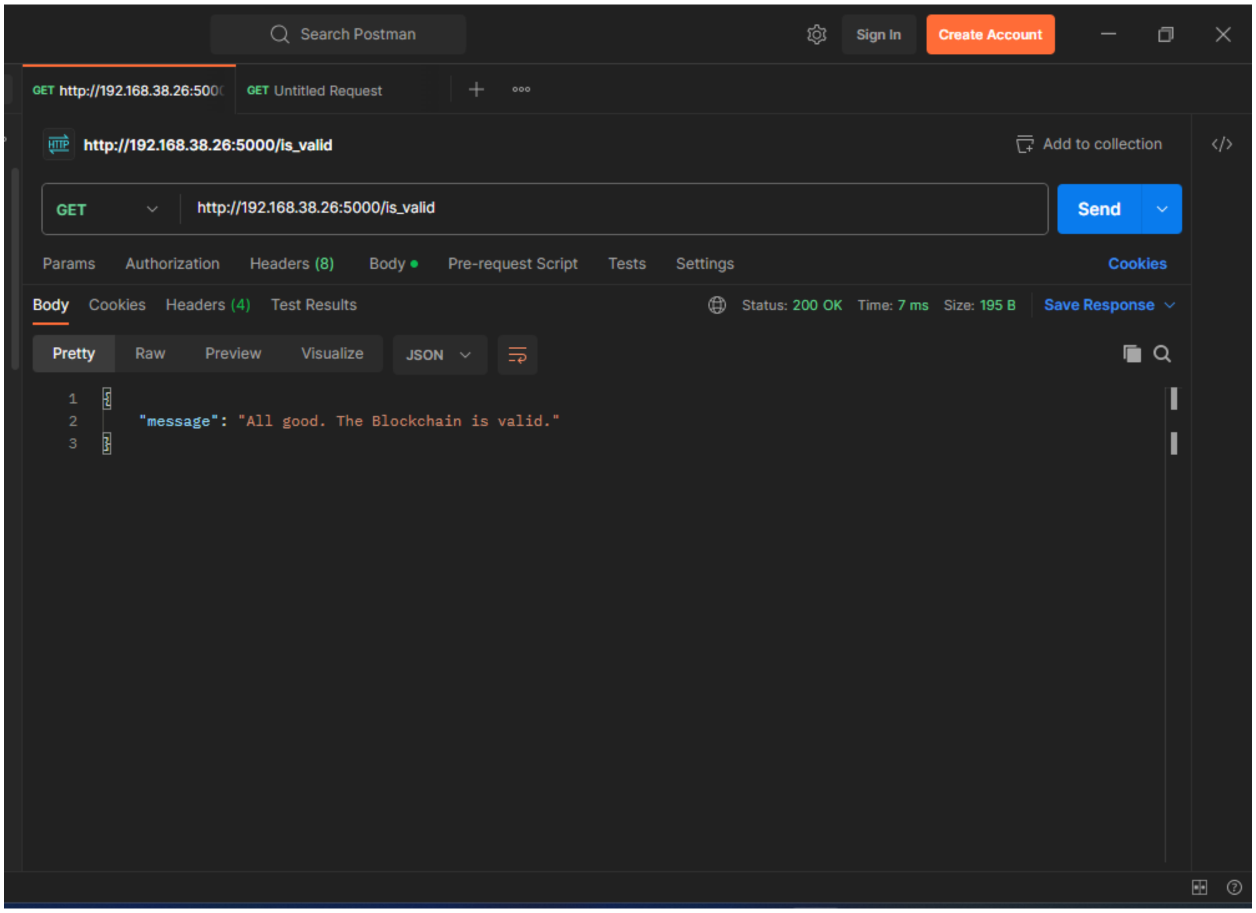


```
C:\Users\Student\Desktop\D17A_28_BC>python BCLab2.py
 * Serving Flask app 'BCLab2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.38.26:5000
Press CTRL+C to quit
```

HTTP Requests on Postman



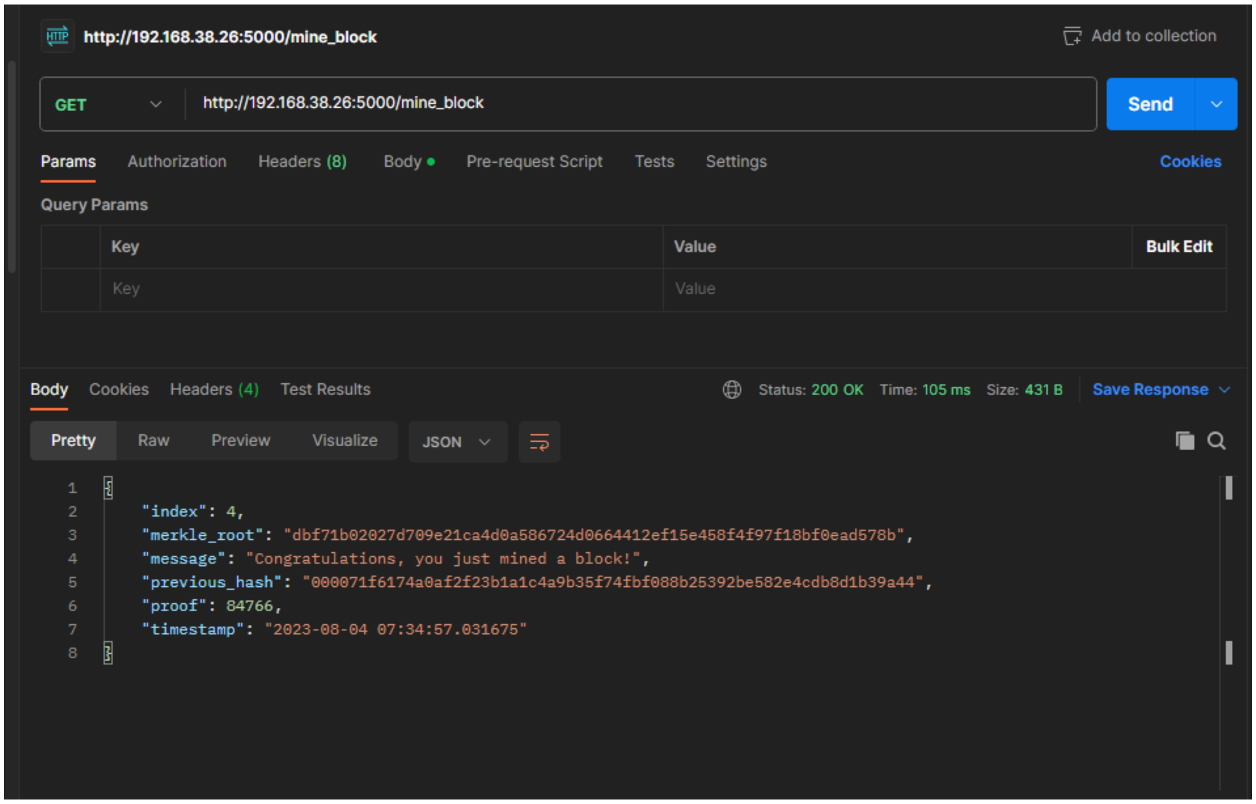
```
{
    "chain": [
        {
            "index": 1,
            "merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
            "previous_hash": "0",
            "proof": 115558,
            "timestamp": "2023-08-04 06:48:41.531577"
        },
        {
            "index": 2,
            "merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
            "previous_hash": "0000c76b308ddf3570fafd0bdd24abd8adfdd53e71a0435f0e18199a4a084188",
            "proof": 48245,
            "timestamp": "2023-08-04 06:49:45.285618"
        },
        {
            "index": 3,
            "merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
            "previous_hash": "00004bb2a794217a6e41f6e633e3c12d451beb1adb048aa687dd56d12f4d8ff4",
            "proof": 93823,
```



GET http://192.168.38.26:5000/is_valid

Status: 200 OK Time: 7 ms Size: 195 B

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



```
{
    "index": 4,
    "merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "000071f6174a0af2f23b1a1c4a9b35f74fbf088b25392be582e4cdb8d1b39a44",
    "proof": 84766,
    "timestamp": "2023-08-04 07:34:57.031675"
}
```

Conclusion: Blockchain has been implemented in Python