NAME: KUNAL PUNJABI | D20A | 46

BLOCKCHAIN LAB

EXPERIMENT NO - 3

AIM

To design and implement a cryptocurrency using Python and demonstrate the mining process in a blockchain environment.

THEORY

1. Blockchain Overview

Blockchain is a decentralized and distributed digital ledger used to securely record transactions across multiple computers called nodes. Unlike traditional systems, it does not rely on a central authority. Instead, every node in the network stores and maintains an identical copy of the ledger.

Each block in a blockchain contains:

- A set of transactions
- A timestamp
- The hash of the previous block
- Its own unique cryptographic hash

Blocks are linked together using cryptographic hash values, forming a secure and continuous chain. Each block depends on the hash of the previous block, so even a small change in data will modify the hash and break the chain. This makes unauthorized changes easy to detect.

Because of this structure, blockchain ensures data integrity, transparency, and immutability, meaning once data is recorded, it cannot be altered.

- **Transactions:** A group of verified records representing the transfer of data or digital assets between participants.

- **Timestamp:** Stores the exact date and time when a block is created, helping maintain the correct order of blocks.

- **Previous hash:** A reference to the hash of the earlier block, which securely links blocks together.

- **Current block hash:** A unique cryptographic value used to identify the block and verify that its data remains unchanged.

2. Mining

Mining is the process through which new blocks are added to the blockchain.

It includes the following steps:

- Collecting pending transactions
- Creating a block header
- Solving a Proof-of-Work (PoW) problem

Collecting pending transactions

Before a new block is created, all unconfirmed transactions in the network are collected. These transactions are checked for validity and then prepared to be added to the next block.

Creating a block header

After selecting the transactions, a block header is created. It contains important details such as the previous block's hash, timestamp, nonce, and Merkle root, which together define the structure of the block.

Solving Proof-of-Work (PoW)

In this step, miners solve a complex mathematical problem by repeatedly changing a nonce value until a valid hash that meets the difficulty requirement is found. Successfully solving this problem allows the block to be added to the blockchain.

Mining involves competition among miners, and the first miner to find a valid hash adds the block to the chain and receives a reward.

3. Multi-Node Blockchain Network

In this experiment, a multi-node blockchain network is simulated using three nodes running on different ports (5001, 5002, and 5003).

Each node:

- Operates independently
- Maintains its own copy of the blockchain
- Communicates with other nodes to share updates

Operates independently

Each node functions on its own and validates data without depending on any central authority.

Maintains its own copy of the blockchain

Every node stores a complete copy of the blockchain, which improves security and ensures fault tolerance.

Communicates with peer nodes

Nodes exchange newly mined blocks and updates with each other to keep the entire network consistent.

Overall, each node works independently while continuously sharing information with other nodes, ensuring synchronization and reliability across the network.

4. Consensus Mechanism

Consensus is a method used to achieve agreement among all nodes in a blockchain network.

Key points of consensus:

- Ensures all nodes have the same and correct version of the blockchain
- Removes the need for a central authority
- Prevents double spending and fraudulent activities
- Uses algorithms such as Proof-of-Work and the Longest Chain Rule
- Maintains trust, security, and synchronization in the network

A consensus mechanism allows nodes to independently verify transactions and blocks before accepting them. It resolves conflicts between different blockchain versions and ensures that only valid data is added, keeping the decentralized system secure and consistent.

5. Transactions and Mining Reward

A transaction in a cryptocurrency system includes:

- **Sender address:** Identifies the account sending the cryptocurrency
- **Receiver address:** Identifies the account receiving the cryptocurrency
- **Transaction amount:** Specifies how much cryptocurrency is being transferred

When a miner successfully adds a new block to the blockchain, they receive a mining reward for their effort.

6. Chain Replacement

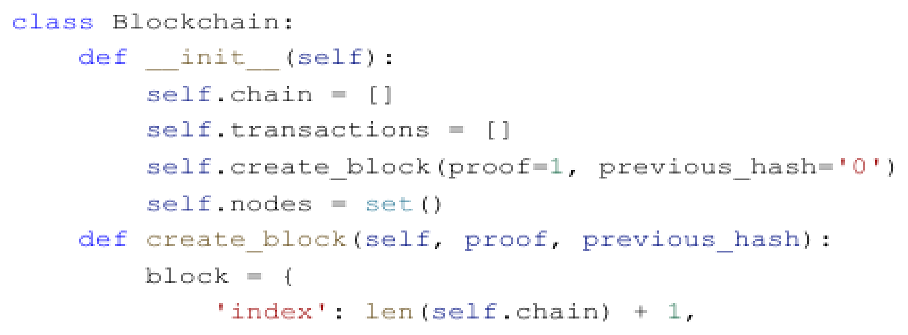
Chain replacement is used to maintain consistency across the blockchain network.

When the `/replace_chain` endpoint is activated:

- A node requests blockchain data from other nodes
- The received chains are validated
- The node replaces its own chain if a longer and valid chain is found

This process helps resolve conflicts between different versions of the blockchain and ensures that all nodes remain synchronized.

CODE :



```
class Blockchain:
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof=1, previous_hash='0')
        self.nodes = set()
    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
```

```python
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash,
            'transactions': self.transactions
        }
        self.transactions = []
        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 not check_proof:
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1
        return new_proof
```

```python
    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]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(
                str(proof**2 - previous_proof**2).encode()
```

```python
            ).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True

    def add_transaction(self, sender, receiver, amount):
        self.transactions.append({
            'sender': sender,
            'receiver': receiver,
            'amount': amount
        })
        previous_block = self.get_previous_block()
        return previous_block['index'] + 1

    def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)
    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain
        if longest_chain:
            self.chain = longest_chain
            return True
        return False

app = Flask(__name__)

node_address = str(uuid4()).replace('-', '')
```

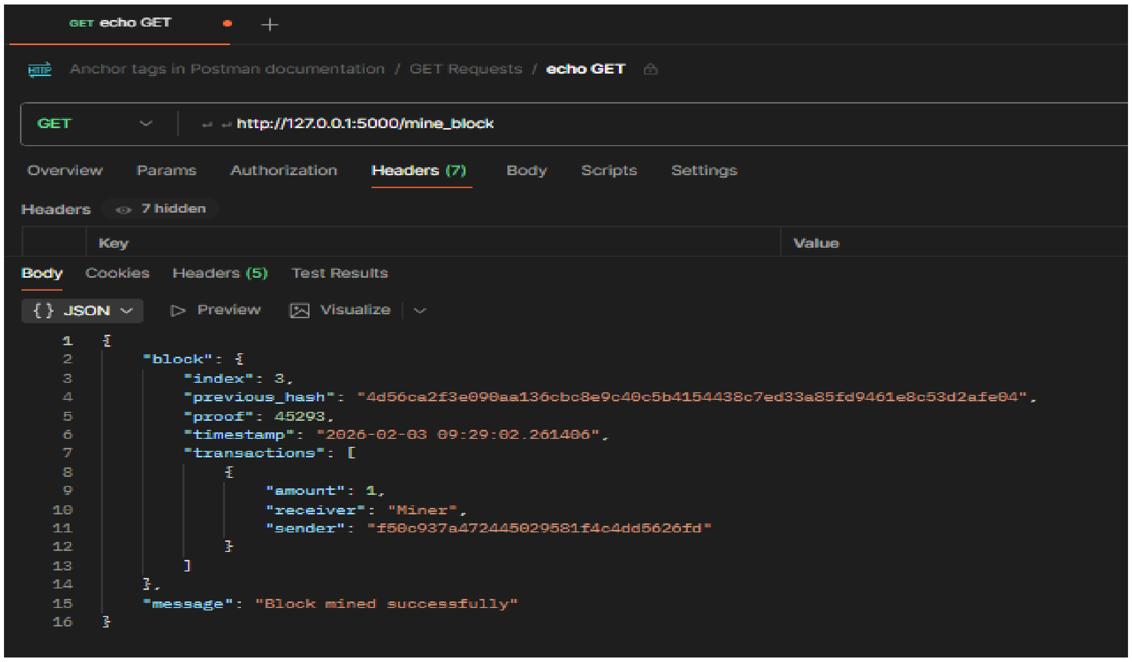
```
blockchain = Blockchain()

@app.route('/mine_block', methods=['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    blockchain.add_transaction(
        sender=node_address,
        receiver='Richard',
        amount=1
    )
    block = blockchain.create_block(proof, previous_hash)
    response = {
        'message': 'Congratulations, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
        'transactions': block['transactions']
    }
    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': 'The Blockchain is not valid.'}
```

```python
    return jsonify(response), 200

@app.route('/add_transaction', methods=['POST'])
def add_transaction():
    json_data = request.get_json()
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in json_data for key in transaction_keys):
        return 'Some elements of the transaction are missing', 400
    index = blockchain.add_transaction(
        json_data['sender'],
        json_data['receiver'],
        json_data['amount']
    )
    response = {
        'message': f'This transaction will be added to Block {index}'
    }
    return jsonify(response), 201

@app.route('/connect_node', methods=['POST'])
def connect_node():
    json_data = request.get_json()
    nodes = json_data.get('nodes')
    if nodes is None:
        return "No node", 400
    for node in nodes:
        blockchain.add_node(node)
    response = {
        'message': 'All the nodes are now connected.',
        'total_nodes': list(blockchain.nodes)
    }
    return jsonify(response), 201

@app.route('/replace_chain', methods=['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {
            'message': 'The chain was replaced by the longest one.',
            'new_chain': blockchain.chain
        }
```

```python
    else:
        response = {
            'message': 'The chain is already the largest one.',
            'actual_chain': blockchain.chain
        }
    return jsonify(response), 200

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

OUTPUT :

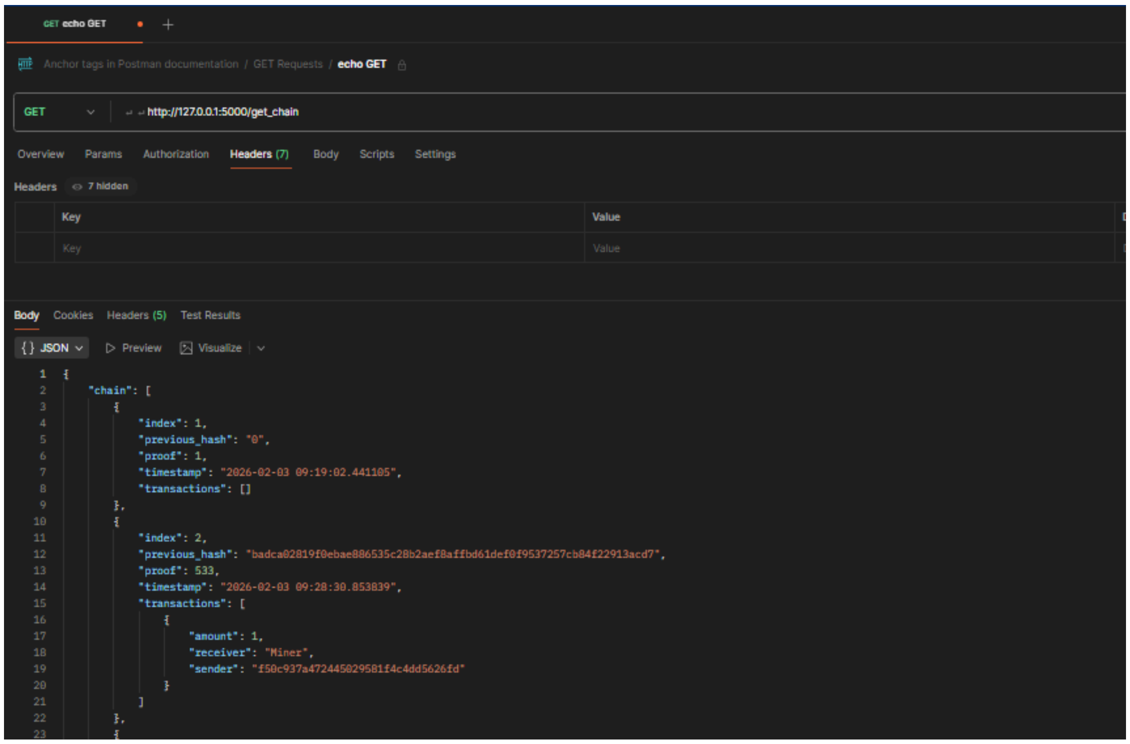


GET echo GET

Anchor tags in Postman documentation / GET Requests / echo GET

GET http://127.0.0.1:5000/mine_block

```
{
    "block": {
        "index": 3,
        "previous_hash": "4d56ca2f3e090aa136cbc8e9c40c5b4154438c7ed33a85fd9461e8c53d2afe04",
        "proof": 45293,
        "timestamp": "2026-02-03 09:29:02.261406",
        "transactions": [
            {
                "amount": 1,
                "receiver": "Miner",
                "sender": "f50c937a472445029581f4c4dd5626fd"
            }
        ]
    },
    "message": "Block mined successfully"
}
```

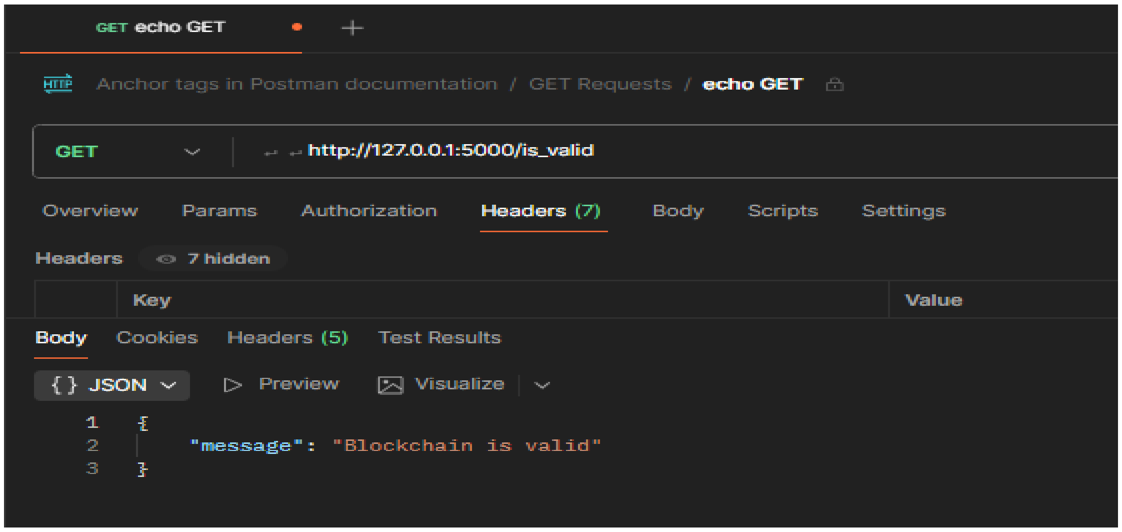


GET echo GET

Anchor tags in Postman documentation / GET Requests / echo GET

GET http://127.0.0.1:5000/get_chain

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-03 09:19:02.441105",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "badca02819f0ebae886535c28b2aef8affbd61def0f9537257cb84f22913acd7",
            "proof": 533,
            "timestamp": "2026-02-03 09:28:30.853839",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Miner",
                    "sender": "f50c937a472445029581f4c4dd5626fd"
                }
            ]
        },
        {
```


GET echo GET
Anchor tags in Postman documentation / GET Requests / echo GET
Overview Params Authorization Headers (7) Body Scripts Settings
Headers 7 hidden
Key Value
Body Cookies Headers (5) Test Results
{} JSON Preview Visualize

GET http://127.0.0.1:5000/is_valid

```
{
    "message": "Blockchain is valid"
}
```

Conclusion:

This implementation demonstrates how blockchain technology works by creating a decentralized ledger using Python. The code shows how blocks are securely linked using cryptographic hashes, how transactions are recorded, and how **Proof-of-Work** is used to mine new blocks. The use of consensus through chain validation and replacement ensures data integrity and synchronization across nodes. Overall, the system highlights the core principles of blockchain such as **decentralization, transparency, security, and immutability.**