

Experiment :03
Blockchain Lab

Aim: Create a Cryptocurrency using Python and perform mining in the Blockchain created.

Theory:

1. Overview of Blockchain

Blockchain is a **distributed, decentralized, and immutable digital ledger** used to record transactions securely across a peer-to-peer (P2P) network. Unlike traditional centralized databases, blockchain does not rely on a single authority. Instead, every participating node maintains a copy of the ledger.

Each block in a blockchain consists of:

- **Transaction data**
- **Timestamp**
- **Hash of the previous block**
- **Current block hash**

The blocks are cryptographically linked using hashes, ensuring data integrity. Any modification in a block changes its hash and breaks the chain, making tampering computationally infeasible.

2. Cryptocurrency and Blockchain Simulation

A cryptocurrency is a **digital asset** that uses cryptographic techniques and blockchain technology to secure transactions and control the creation of new units. In this experiment, a **simple cryptocurrency (Hadcoin)** is implemented using Python to demonstrate real-world blockchain behavior.

The system simulates:

- Decentralized nodes
 - Transaction handling
 - Mining process
- Consensus among peers

3. Mining Process

Mining is the process through which:

- Pending transactions are collected
 - A new block is created
 - A cryptographic puzzle (Proof-of-Work) is solved
 - The block is added to the blockchain
-
- In Proof-of-Work:
 - Miners repeatedly calculate hashes
 - A valid hash must satisfy a predefined difficulty condition (e.g., starting with leading zeros)
- Once found, the block is broadcast to all peers

The miner who successfully mines the block receives a **mining reward**, which is added as a transaction in the block.

4. Transactions in the Blockchain Network

A transaction represents the transfer of cryptocurrency between participants and contains:

- **Sender address**
- **Receiver address**
- **Amount**

Transactions are first added to a **temporary transaction pool (mempool)**. These pending transactions are later included in a mined block. After a block is successfully mined, the transactions are removed from the pool to prevent duplication.

5. Mempool (Memory Pool)

The mempool is a temporary storage area where **unconfirmed transactions** are kept before being mined into a block.

Role of mempool:

- Stores pending transactions
- Helps miners select transactions for the next block
- Prevents double spending
- Improves transaction organization and efficiency

In this experiment, the code is modified to ensure that transactions are **cleared from the mempool after being added to a block**.

6. Peer-to-Peer (P2P) Network

A P2P network allows nodes to communicate directly without a central server. In this lab:

- Three nodes run on ports **5001, 5002, and 5003**
- Each node maintains its own blockchain
- Nodes connect to peers using HTTP requests

This setup demonstrates decentralization, where no single node controls the network.

7. Challenges in P2P Networks

Some common challenges include:

- **Consensus management:** Ensuring all nodes agree on the same blockchain
- **Network latency:** Delay in block propagation
- **Forks:** Multiple versions of the blockchain
- **Security risks:** Sybil attacks, double spending
- **Scalability:** Handling large numbers of transactions

8. Consensus Mechanism – Longest Chain Rule

To maintain consistency, this experiment uses the **Longest Chain Rule**:

- If multiple chains exist, the longest valid chain is accepted
- Nodes compare their chain with peers
- The shorter chain is replaced automatically

This ensures synchronization and agreement across all nodes.

9. Public and Private Blockchain Implementation

- **Public Blockchain:** Multiple nodes participate and share the blockchain openly
- **Private Blockchain:** Controlled environment with selected nodes

This experiment demonstrates both concepts by allowing controlled node participation while maintaining decentralized behavior.

10. Tools and Libraries Used

Python : Used to implement blockchain logic, mining, hashing, and networking.

Flask: A lightweight web framework used to:

- Create REST APIs
- Handle blockchain requests such as mining, adding transactions, and chain replacement

1. Add Transactions - invoke `add_transactions()` as a POST request.

The screenshot shows a Postman interface with a POST request to `http://127.0.0.1:5001/add_transaction`. The request body contains the following JSON:

```
1
2   "amount": 1000,
3   "receiver": "pooja",
4   "sender": "4208036bca004b56b2e456b4daff0059"
5
6
```

The response status is **201 CREATED** with a response time of 3 ms and a response size of 226 B. The response body is:

```
{}
  "message": "This transaction will be added to Block 4"
```

2. mining - `mine_block()`

The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:5001/mine_block`. The response status is **200 OK** with a response time of 25 ms and a response size of 621 B. The response body is:

```

2   "index": 4,
3   "message": "Congratulations, you just mined a block!",
4   "previous_hash": "ed3cc6f96be22e405a6ed26538e3c0a71a79779d682eff6e7e713be9ec690104",
5   "proof": 21391,
6   "timestamp": "2026-01-27 11:26:58.840324",
7   "transactions": [
8     {
9       "amount": 1,
10      "receiver": "Richard",
11      "sender": "4208036bca004b56b2e456b4daff0059"
12    },
13    {
14      "amount": 1000,
15      "receiver": "pooja",
16      "sender": "4208036bca004b56b2e456b4daff0059"
17    }
]
```

3.fetch the chain - get_chain()

HTTP http://127.0.0.1:5001/get_chain

GET http://127.0.0.1:5001/get_chain

Send

Docs Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (5) Test Results | ⏱

200 OK • 4 ms • 533 B • ⓘ | ⌂

{ } JSON ▾ ▶ Preview ⌂ Visualize | ↴

```
1 {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "0",
6       "proof": 1,
7       "timestamp": "2026-01-27 11:09:25.029105",
8       "transactions": []
9     },
10    {
11      "index": 2,
12      "previous_hash": "6d617bf1d8543b614960a1afcdbe1e1bb5f00e0e840c15c815761faffe79dec0",
13      "proof": 533,
14      "timestamp": "2026-01-27 11:12:17.003830",

```

4. node - invoke connect_node()

HTTP http://127.0.0.1:5001/connect_node

POST http://127.0.0.1:5001/connect_node

Send

Docs Params Authorization Headers (8) Body ● Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON ▾ Schema Beautify

Body Cookies Headers (5) Test Results | ⏱

201 CREATED • 5 ms • 325 B • ⓘ | ⌂

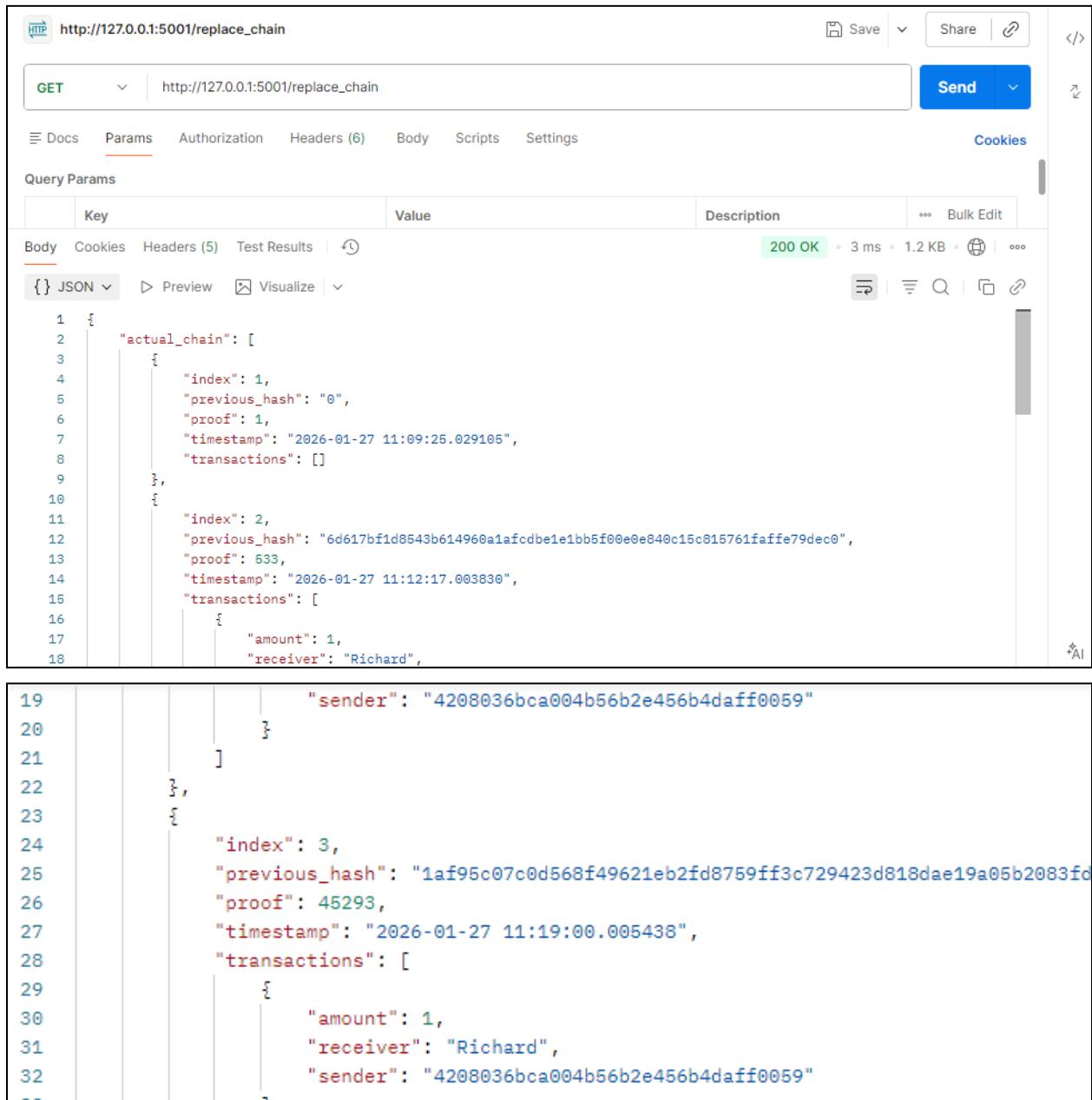
{ } JSON ▾ ▶ Preview ⌂ Visualize | ↴

```
1 {
2   "nodes": [
3     "http://127.0.0.1:5000",
4     "http://127.0.0.1:5002"
5   ]

```

```
1 {
2   "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
3   "total_nodes": [
4     "127.0.0.1:5002",
5     "127.0.0.1:5000"
6   ]
7 }
```

5. replace the longest chain - replace_chain()



The screenshot shows a Postman interface with the following details:

- URL:** `http://127.0.0.1:5001/replace_chain`
- Method:** GET
- Headers:** (6)
- Body:** (JSON) - The response body is displayed as JSON code.
- Response Status:** 200 OK
- Response Time:** 3 ms
- Response Size:** 1.2 KB

```
1 {  
2   "actual_chain": [  
3     {  
4       "index": 1,  
5       "previous_hash": "0",  
6       "proof": 1,  
7       "timestamp": "2026-01-27 11:09:25.029105",  
8       "transactions": []  
9     },  
10    {  
11      "index": 2,  
12      "previous_hash": "6d617bf1d8543b614960a1afcdbe1e1bb5f00e0e840c15c815761faffe79dec0",  
13      "proof": 533,  
14      "timestamp": "2026-01-27 11:12:17.003830",  
15      "transactions": [  
16        {  
17          "amount": 1,  
18          "receiver": "Richard",  
19          "sender": "4208036bca004b56b2e456b4daff0059"  
20        }  
21      ]  
22    },  
23    {  
24      "index": 3,  
25      "previous_hash": "1af95c07c0d568f49621eb2fd8759ff3c729423d818dae19a05b2083fd",  
26      "proof": 45293,  
27      "timestamp": "2026-01-27 11:19:00.005438",  
28      "transactions": [  
29        {  
30          "amount": 1,  
31          "receiver": "Richard",  
32          "sender": "4208036bca004b56b2e456b4daff0059"  
33        }  
34      ]  
35    }  
36  ]  
37}  
38
```

```

"index": 4,
"previous_hash": "ed3cc6f96be22e405a6ed26538e3c0a71a79779d682eff6e7e713be9ec690104",
"proof": 21391,
"timestamp": "2026-01-27 11:26:58.840324",
"transactions": [
    {
        "amount": 1,
        "receiver": "Richard",
        "sender": "4208036bca004b56b2e456b4daff0059"
    },
    {
        "amount": 1000,
        "receiver": "pooja",
        "sender": "4208036bca004b56b2e456b4daff0059"
    },
    {
        "amount": 1,
        "receiver": "Richard",
        "sender": "4208036bca004b56b2e456b4daff0059"
    }
],
"message": "All good. The chain is the largest one."
}

```

Conclusion

This experiment demonstrates the practical implementation of a cryptocurrency using Python and blockchain principles. By simulating a P2P network with multiple nodes, mining, transactions, and consensus mechanisms, the experiment provides hands-on understanding of how real-world blockchain systems operate. It highlights the importance of decentralization, security, and consensus in distributed ledger technologies.