

# Experiment 3

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

## Theory:

### 1. Blockchain Overview

Blockchain is a distributed and decentralized digital ledger technology used to securely record transactions across multiple computers (nodes). Instead of storing data in a single centralized database, blockchain distributes copies of the ledger across all participating nodes in the network.

The blockchain consists of a sequence of blocks, where each block contains:

- A list of transactions
- A timestamp indicating when the block was created
- The cryptographic hash of the previous block
- Its own unique hash generated using a cryptographic algorithm (e.g., SHA-256)

The inclusion of the previous block's hash creates a chain structure, ensuring data integrity. Once a block is added to the blockchain, its contents become immutable, meaning any modification would require recalculating the hashes of that block and all subsequent blocks, which is computationally impractical.

### 2. Mining Process

Mining is the process through which new blocks are created and added to the blockchain. It involves the following steps:

1. Collecting pending transactions from the transaction pool.
2. Forming a new block containing these transactions.
3. Solving a computational puzzle using the Proof-of-Work (PoW) mechanism.
4. Generating a valid hash that satisfies the network's difficulty condition.
5. Adding the successfully mined block to the blockchain.
6. Broadcasting the new block to all connected nodes in the network.

Mining ensures network security, transaction validation, and block creation. Miners who successfully mine a block are rewarded with cryptocurrency as an incentive for contributing computational power.

### 3. Multi-Node Blockchain Network

A multi-node blockchain network consists of multiple independent blockchain nodes operating simultaneously. In this experiment, three nodes are simulated using different port numbers: 5001, 5002, and 5003.

Each node:

- Runs as an independent instance of the blockchain application
- Maintains its own local copy of the blockchain
- Can communicate with other nodes in the network
- Shares newly mined blocks and transactions with peers

This distributed setup demonstrates how blockchain eliminates the need for a central authority while maintaining data consistency across the network.

#### **4. Consensus Mechanism**

To ensure all nodes maintain a consistent version of the blockchain, a consensus mechanism is required. In this experiment, the Longest Chain Rule is used.

According to this rule:

- If multiple versions of the blockchain exist, the node accepts the longest valid chain.
- The longest chain represents the greatest amount of computational work performed.
- This mechanism resolves conflicts that arise when different nodes mine blocks simultaneously.

Consensus ensures that all nodes eventually agree on a single, authoritative transaction history.

#### **5. Transactions and Mining Reward**

A blockchain transaction typically includes: Sender's address, Receiver's address, Amount transferred.

Before mining, transactions remain in a pending state. When a block is mined:

- All pending transactions are added to the block.
- A reward transaction is automatically generated.
- This reward pays the miner for successfully mining the block.

The mining reward acts as an economic incentive and introduces new cryptocurrency into the system.

#### **6. Chain Replacement Mechanism**

Chain replacement is used to maintain consistency among nodes. When the /replace\_chain endpoint is called:

- The node requests the blockchain from all connected peer nodes.
- It compares the received chains with its current chain.
- If a longer and valid chain is found, the node replaces its own blockchain.
- If no longer chain exists, the current chain is retained.

This mechanism ensures fault tolerance, consistency, and synchronization across the distributed blockchain network.

## 1. Connect Node 2 and 3 to Node 1

The screenshot shows a Postman collection named "mahvish". A POST request is made to "http://127.0.0.1:5001/connect\_node". The request body is a JSON object with a "nodes" array containing two URLs: "http://127.0.0.1:5002" and "http://127.0.0.1:5003". The response status is 201 (CREATED) with a response time of 13 ms and a size of 0.16 kb. The response body contains a message: "Hey Mahvish! All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes: ["127.0.0.1:5003", "127.0.0.1:5002"]". The operating system taskbar at the bottom shows various icons and the date/time as 06-02-2026.

## 2. Connect Node 1 and 3 to Node 2

The screenshot shows a Postman collection named "mahvish". A POST request is made to "http://127.0.0.1:5002/connect\_node". The request body is a JSON object with a "nodes" array containing two URLs: "http://127.0.0.1:5001" and "http://127.0.0.1:5003". The response status is 201 (CREATED) with a response time of 13 ms and a size of 0.16 kb. The response body contains a message: "Hey Mahvish! All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes: ["127.0.0.1:5003", "127.0.0.1:5001"]".

### 3. Connect Node 1 and 2 to Node 3

The screenshot shows a Postman collection named "mahvish". A POST request is made to "http://127.0.0.1:5003/connect\_node". The request body contains the following JSON:

```
{  
  "nodes": [  
    "http://127.0.0.1:5001",  
    "http://127.0.0.1:5002"  
  ]  
}
```

The response status is 201 (CREATED) with a response time of 13 ms and a size of 0.16 kb. The response body contains the following JSON:

```
{  
  "message": "Hey Mahvish! All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",  
  "total_nodes": ["127.0.0.1:5001", "127.0.0.1:5002"]  
}
```

### 4. Add a transaction to node 1

The screenshot shows a POST request to "http://127.0.0.1:5001/add\_transaction". The request body contains the following JSON:

```
{  
  "sender": "Mahvish",  
  "receiver": "Shreya",  
  "amount": 58  
}
```

The response status is 201 (CREATED) with a response time of 8 ms and a size of 0.05 kb. The response body contains the following JSON:

```
{  
  "message": "This transaction will be added to Block 2"  
}
```

### 5. Mine block in node 1

Name: mahvish

Save Share Generate ▾

http://127.0.0.1:5001/mine\_block

GET ↗ Send

Params Body Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

Body 16 Headers 5 Raw 8

200 (OK) 11 ms 0.34 kb

Body

```
{"index":2,"message":"Congratulations, you just mined a block!","previous_hash":"48262b4e2674679bde80930040d14888e5912c16f0d910bc000179dedce6dd2b","proof":533,"timestamp":"2026-02-06 12:32:57.975439","transactions":[{"amount":58,"receiver":"Shreya","sender":"Mahvish"}, {"amount":1,"receiver":"Richard","sender":"51d2d30e4a7e4fa3bb549dbaf3734a99"}]}
```

## 6. Add second transaction to node 1

Name: mahvish

Save Share Generate ▾

http://127.0.0.1:5001/add\_transaction

POST ↗ Send

Params Body 6 Auth Headers 6 Raw 13

None  JSON  Form (url-encoded)  XML  Custom

```
{
  "sender": "mahvish",
  "receiver": "shreya",
  "amount": 508
}
```

Body 3 Headers 5 Raw 8

201 (CREATED) 10 ms 0.05 kb

HTTP Message

```
HTTP/1.1 201 CREATED
connection: close
content-length: 56
content-type: application/json
date: Fri, 06 Feb 2026 07:05:51 GMT
server: Werkzeug/3.1.5 Python/3.12.4

{"message":"This transaction will be added to Block 3"}
```

## 7. Mine Block in node 1

Body 16 Headers 5 Raw 8

200 (OK) 142 ms 0.34 kb

HTTP Message

```
HTTP/1.1 200 OK
connection: close
content-length: 352
content-type: application/json
date: Fri, 06 Feb 2026 07:07:04 GMT
server: Werkzeug/3.1.5 Python/3.12.4

{"index":3,"message":"Congratulations, you just mined a block!","previous_hash":"0219ff41e19a4fd94d129c4e8e0de309bd8e15c56fa9297464c2t"}
```

```

Body 16 Headers 5 Raw 8
200 (OK) 142 ms 0.34 kb
{"index":3,"message":"Congratulations, you just mined a block!","previous_hash":"0219ff41e19a4fd94d129c4e8e0de309bd8e15c56fa9297464c2b421afedaaa3","proof":45293,"timestamp":"2026-02-06 12:37:04.827780","transactions":[{"amount":508,"receiver":"shreya","sender":"mavhish"}, {"amount":1,"receiver":"Richard","sender":"51d2d30e4a7e4fa3bb549dbaf3734a99"}]}

```

## 8. Get chain in node 2

Name: mahvish

http://127.0.0.1:5002/get\_chain

GET

Params Body 6 Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

```

Body 10 Headers 5 Raw 8
200 (OK) 10 ms 0.12 kb
Body
{
  "chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 13:07:27.567458", "transactions": []}], "length": 1
}

```

## 9. Replace chain in node 2

Name: mahvish

http://127.0.0.1:5002/replace\_chain

POST

Params Body 6 Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

```

Body 38 Headers 5 Raw 8
200 (OK) 30 ms 0.78 kb
HTTP Message
HTTP/1.1 200 OK
connection: close
content-length: 801
content-type: application/json
date: Fri, 06 Feb 2026 07:43:19 GMT
server: Werkzeug/3.1.5 Python/3.12.4

{"message":"The nodes had different chains so the chain was replaced by the longest one.", "new_chain":[{"index":1,"previous_hash": "0"}]}

```

```
{
  "message": "The nodes had different chains so the chain was replaced by the longest one.",
  "new_chain": [
    {
      "index": 1,
      "previous_hash": "0"
    }
  ]
}
```

```

    "previous_hash": "0",
    "proof": 1,
    "timestamp": "2026-02-06 13:07:21.960872",
    "transactions": []
}, {
    "index": 2,
    "previous_hash": "f3a3c29af20a2e4513ed4a027e6bacf88620980461d6978709ff950f7c590181",
    "proof": 533,
    "timestamp": "2026-02-06 13:10:33.363423",
    "transactions": [
        {
            "amount": 58,
            "receiver": "shreya",
            "sender": "mahvish"
        }, {
            "amount": 1,
            "receiver": "Richard",
            "sender": "ea968f3f9622438da3b08568c81d25f6"
        }
    ]
}, {
    "index": 3,
    "previous_hash": "72e02cc82100b8aaa43cd4b7ebb1904f70af1d06c8fe7665c596c8180d7c8511",
    "proof": 45293,
    "timestamp": "2026-02-06 13:11:07.244174",
    "transactions": [
        {
            "amount": 508,
            "receiver": "shreya",
            "sender": "mahvish"
        }, {
            "amount": 1,
            "receiver": "Richard",
            "sender": "ea968f3f9622438da3b08568c81d25f6"
        }
    ]
}

```

10. Get chain in node 3

Name: mahvish

http://127.0.0.1:5003/get\_chain

Params [Body](#) 6 Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

Body 10 Headers 5 [Raw](#) 8 200 (OK) 9 ms 0.12 kb

HTTP Message

```
HTTP/1.1 200 OK
connection: close
content-length: 124
content-type: application/json
date: Fri, 06 Feb 2026 07:44:42 GMT
server: Werkzeug/3.1.3 Python/3.12.4

{"chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 13:07:34.588301", "transactions": []}], "length": 1}
```

## 11. Replace chain node 3

Name: mahvish

http://127.0.0.1:5003/replace\_chain

Params [Body](#) 6 Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

Body 38 Headers 5 [Raw](#) 8 200 (OK) 29 ms 0.78 kb

HTTP Message

```
HTTP/1.1 200 OK
connection: close
content-length: 801
content-type: application/json
date: Fri, 06 Feb 2026 07:45:29 GMT
server: Werkzeug/3.1.3 Python/3.12.4

{"message": "The nodes had different chains so the chain was replaced by the longest one.", "new_chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 13:07:34.588301", "transactions": []}], "length": 1}
```

## 12. Get chain in node 3

Name: mahvish

http://127.0.0.1:5003/get\_chain

Params [Body](#) 6 Auth Headers 4 Raw 5

None  JSON  Form (url-encoded)  XML  Custom

Body 38 Headers 5 Raw 8

HTTP Message

```
HTTP/1.1 200 OK
connection: close
content-length: 719
content-type: application/json
date: Fri, 06 Feb 2026 07:46:00 GMT
server: Werkzeug/3.1.3 Python/3.12.4

{"chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 13:07:21.960872", "transactions": []}, {"index": 2, "previous_hash": "f3a3c29af20a2e4513ed4a027e6bacf88620980461d6978709ff950f7c590181", "proof": 533, "timestamp": "2026-02-06 13:10:33.363423", "transactions": [{"amount": 58, "receiver": "shreya", "sender": "mahvish"}, {"amount": 1, "receiver": "Richard", "sender": "ea968f3f9622438da3b08568c81d25f6"}]}, {"index": 3, "previous_hash": "72e02cc82100b8aaa43cd4b7ebb1904f70af1d06c8fe7665c596c8180d7c8511", "proof": 45293, "timestamp": "2026-02-06 13:11:07.244174", "transactions": [{"amount": 508, "receiver": "shreya", "sender": "mahvish"}]}]
```

```
{  
    "chain": [  
        {  
            "index": 1,  
            "previous_hash": "0",  
            "proof": 1,  
            "timestamp": "2026-02-06 13:07:21.960872",  
            "transactions": []  
        }, {  
            "index": 2,  
            "previous_hash":  
                "f3a3c29af20a2e4513ed4a027e6bacf88620980461d6978709ff950f7c590181",  
            "proof": 533,  
            "timestamp": "2026-02-06 13:10:33.363423",  
            "transactions": [  
                {  
                    "amount": 58,  
                    "receiver": "shreya",  
                    "sender": "mahvish"  
                }, {  
                    "amount": 1,  
                    "receiver": "Richard",  
                    "sender": "ea968f3f9622438da3b08568c81d25f6"  
                }]  
        }, {  
            "index": 3,  
            "previous_hash":  
                "72e02cc82100b8aaa43cd4b7ebb1904f70af1d06c8fe7665c596c8180d7c8511",  
            "proof": 45293,  
            "timestamp": "2026-02-06 13:11:07.244174",  
            "transactions": [  
                {  
                    "amount": 508,  
                    "receiver": "shreya",  
                    "sender": "mahvish"  
                }]  
        }  
    ]  
}
```

```

        "amount": 1,
        "receiver": "Richard",
        "sender": "ea968f3f9622438da3b08568c81d25f6"
    }]
},
"length": 3
}

```

### 13. Getchain in node 2 and node 1

The screenshot shows two separate API requests made using a browser-based testing interface.

**Request 1 (Node 2):**

- Name: mahvish
- URL: [http://127.0.0.1:5002/get\\_chain](http://127.0.0.1:5002/get_chain)
- Method: GET
- Params: Body 6, Headers 4, Raw 5
- Body (Raw):

```

HTTP/1.1 200 OK
connection: close
content-length: 962
content-type: application/json
date: Fri, 06 Feb 2026 07:17:52 GMT
server: Werkzeug/3.1.5 Python/3.12.4

{"chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 12:24:50.854836", "transactions": []}, {"index": 2, "previous_ha

```

**Request 2 (Node 1):**

- Name: mahvish
- URL: [http://127.0.0.1:5001/get\\_chain](http://127.0.0.1:5001/get_chain)
- Method: GET
- Params: Body 6, Headers 4, Raw 5
- Body (Raw):

```

HTTP/1.1 200 OK
connection: close
content-length: 962
content-type: application/json
date: Fri, 06 Feb 2026 07:16:57 GMT
server: Werkzeug/3.1.5 Python/3.12.4

{"chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-06 12:24:50.854836", "transactions": []}, {"index": 2, "previous_ha

```

### 14. Replace chain node 3

The screenshot shows a single API request made using a browser-based testing interface.

Name: mahvish

URL: [http://127.0.0.1:5003/replace\\_chain](http://127.0.0.1:5003/replace_chain)

Method: GET

Params: Body 6, Auth, Headers 4, Raw 5

Body 38 Headers 5 Raw 8 200 (OK) 26 ms 0.75 kb

Body

```

        "receiver": "shreya",
        "sender": "mahvish"
    }, {
        "amount": 1,
        "receiver": "Richard",
        "sender": "ea968f3f9622438da3b08568c81d25f6"
    }]
},
"message": "All good. The chain is the largest one."
}

```

### Terminal screenshot:

```
(venv) PS C:\Users\siddi\Desktop\blockchain Lab\lab2> python hadcoin_node_5001.py
* Serving Flask app 'hadcoin_node_5001'
* 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://192.168.179.167:5001
Press CTRL+C to quit
127.0.0.1 - - [06/Feb/2026 12:25:09] "POST /connect_node HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2026 12:31:03] "POST /add_transaction HTTP/1.1" 400 -
127.0.0.1 - - [06/Feb/2026 12:31:43] "POST /add_transaction HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2026 12:32:57] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:35:51] "POST /add_transaction HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2026 12:37:04] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:38:21] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:39:56] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:45:15] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:46:20] "GET /replace_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:46:57] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:47:40] "GET /get_chain HTTP/1.1" 200 -

```

```
❖ (venv) PS C:\Users\siddi\Desktop\blockchain Lab\lab2> python hadcoin_node_5002.py
* Serving Flask app 'hadcoin_node_5002'
* 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://127.0.0.1:5002
* Running on http://192.168.179.167:5002
Press CTRL+C to quit
127.0.0.1 - - [06/Feb/2026 12:27:02] "POST /connect_node HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2026 12:38:21] "GET /replace_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:39:56] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:43:04] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:46:20] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:47:40] "GET /replace_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:47:52] "GET /get_chain HTTP/1.1" 200 -

```

```
PS C:\Users\siddi\Desktop\blockchain Lab\lab2> python hadcoin_node_5003.py
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://127.0.0.1:5003
* Running on http://192.168.179.167:5003
Press CTRL+C to quit
127.0.0.1 - - [06/Feb/2026 12:28:16] "POST /connect_node HTTP/1.1" 201 -
127.0.0.1 - - [06/Feb/2026 12:38:21] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:39:56] "GET /replace_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:41:13] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:41:45] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:46:20] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [06/Feb/2026 12:47:40] "GET /get_chain HTTP/1.1" 200 -

```

**Conclusion:**

We developed a cryptocurrency using Python and implemented mining in a simulated three-node blockchain network. Each node maintained its own ledger and synchronized with peers using the Longest Chain Rule to ensure consistency. Mining was performed through Proof-of-Work, securely adding transactions and rewarding miners. This demonstrated key blockchain concepts, including decentralized consensus, transaction validation, and network synchronization.