

BlockChain EXPERIMENT 3

Name- Ronit Santwani

Class-D20A

Roll no- 63

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

Theory:

1. Blockchain Overview

A blockchain is a distributed, decentralized digital ledger used to record transactions securely across multiple computers (nodes). Instead of relying on a central authority, every node in the network maintains its own copy of the blockchain.

Each block in the blockchain contains:

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

The blocks are linked together using hashes, forming a chain. Any attempt to modify data in a block changes its hash, which breaks the chain and makes tampering easily detectable. This property ensures data integrity, transparency, and immutability.

2. Mining

Mining is the process by which new blocks are added to the blockchain. It involves:

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

In Proof-of-Work, miners repeatedly change a nonce value to generate a hash that satisfies the network's difficulty requirement (such as leading zeroes). Once a valid hash is found, the block is added to the blockchain and shared with other nodes. As an incentive, miners receive a cryptocurrency reward for their computational effort.

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 peer nodes to share newly mined blocks

This setup demonstrates how blockchain achieves decentralization and synchronization without a central server.

4. Consensus Mechanism

To ensure consistency across all nodes, the Longest Chain Rule is used as the consensus mechanism.

When different versions of the blockchain exist, the node accepts the longest valid chain as the correct one. This rule ensures that all nodes eventually agree on a single transaction history, even if temporary differences arise.

5. Transactions and Mining Reward

Transactions in the cryptocurrency system include:

- Sender address
- Receiver address
- Transaction amount

When a block is mined, all pending transactions are added to the block. Additionally, a special transaction is created to reward the miner with newly generated cryptocurrency. This reward mechanism motivates miners to maintain and secure the network.

6. Chain Replacement

The chain replacement process ensures network consistency. When the /replace_chain endpoint is triggered:

- A node requests blockchain data from its peers
- Validates the received chains
- Replaces its own chain if a longer and valid chain is found.

This mechanism helps resolve conflicts and keeps all nodes synchronized.

Code:

```
# Module 2 - Create a Cryptocurrency

# ===== #
HadCoin Blockchain Node
```

```
# =====
import datetime import hashlib import
json from flask import Flask, jsonify,
request import requests from uuid import
uuid4 from urllib.parse import urlparse
```

```
# =====
```

```

# Blockchain Class
# =====
class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
        self.nodes = set()
        self.create_block(proof=1,
                          previous_hash='0')

    # -----
    -- # Create Block
    # -----
    def create_block(self,
                     proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp':
                str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash,
            'transactions': self.transactions.copy()
        }

        # ⚠️ IMPORTANT FIX: Clear
        transactions after adding to block
        self.transactions = []

        self.chain.append(block)
        return block

    # -----
    # Get Previous Block # --
    ----- def
get_previous_block(self):
    return self.chain[-1]

    # -----
    - # Proof of Work
    # ----- def
    proof_of_work(self, previous_proof):
        new_proof = 1      check_proof = False

        while check_proof is False:
            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

    # -----
    -- # Hash Block
    # ----- def hash(self,
block):
        encoded_block = json.dumps(block,
sort_keys=True).encode()
        return
        hashlib.sha256(encoded_block).hexdigest()
()

    # -----
    -- # Check Chain
Validity
    # ----- 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=      def replace_chain(self):      previous_block['proof']
network = self.nodes proof = block['proof']           longest_chain =
None          max_length = len(self.chain)           hash_operation =
hashlib.sha256(           str(proof**2) -           for node in network:
previous_proof**2).encode()           response =           requests.get(fhttp://{node}/get_chain')

if hash_operation[:4] != '0000':           if response.status_code == 200:
    return False           length = response.json()['length']           chain =
response.json()['chain']           previous_block = block
block_index += 1           if length > max_length     and self.is_chain_valid(chain):
return True   max_length = length           longest_chain = chain
# ----- # Add
Transaction # ----- def      if longest_chain:      add_transaction(self,
sender, receiver, self.chain = longest_chain
amount):           return True
self.transactions.append({
'sender': sender,           return False
'receiver': receiver,           # =====
'amount': amount
})      # Flask App # =====
previous_block =
self.get_previous_block()           app = Flask(__name__) return
previous_block['index'] + 1           node_address = str(uuid4()).replace('-', '')
# -----
- # Add Node      blockchain = Blockchain()
# ----- def      add_node(self, address):      # ---
----- parsed_url = urlparse(address)      # Mine Block
self.nodes.add(parsed_url.netloc)      #
# ----- @app.route('/mine_block',
methods=['GET']) def mine_block():
# -----
- # Replace Chain
# ----- previous_block =
previous_proof=      blockchain.get_previous_block()
previous_block['proof'] proof=      blockchain.proof_of_work(previous_pro
o f)      previous_hash =
blockchain.hash(previous_block)

```

```

# Reward transaction
blockchain.add_transaction(
    sender=node_address,
    receiver='Soham',
    amount=1
)

block =
blockchain.create_block(proof,
previous_hash)

response = {
'message': 'Block mined
successfully!',
'block': block
}

return jsonify(response), 200

# -----
-- # Get Chain
# -----
@app.route('/get_chain',
methods=['GET']) def get_chain():
response = { 'chain':
blockchain.chain,
'length': len(blockchain.chain)
}
return jsonify(response), 200

# -----
-- # Check Validity
# -----
@app.route('/is_valid', methods=['GET'])
def is_valid(): is_valid =
blockchain.is_chain_valid(blockchain.ch
a
in)

if is_valid:
    response = {'message': 'Blockchain
is valid.'} else:
        response = {'message': 'Blockchain
is NOT valid.'}

return jsonify(response), 200

# -----
-- # Add Transaction
# -----
@app.route('/add_transacti
on', methods=['POST']) def add_transaction():
    json_data = request.get_json()
    required_fields = ['sender', 'receiver',
'amount']

    if not all(field in json_data for field in required_fields):
        return 'Missing values', 400

    index = blockchain.add_transaction(
        json_data['sender'],
        json_data['receiver'],
        json_data['amount']
    )

    response = {
        'message': f'Transaction will be
added to Block {index}'
    }

    return jsonify(response), 201

# -----
# Connect Nodes
# -----
@app.route('/connect_no de',
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 nodes connected successfully!',
        'total_nodes': list(blockchain.nodes)
    }

    return jsonify(response), 201

# -----
Replace Chain
# -----
@app.route('/replace_chain',
methods=['GET'])
def replace_chain():
    is_replaced = blockchain.replace_chain()

    if is_replaced:
        response = {
            'message': 'Chain was replaced.',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Current chain is longest.',
            'actual_chain': blockchain.chain
        }

    return jsonify(response), 200

#
=====
# Run App
# =====

```

Output:

1)/get_chain

This GET request retrieves the complete blockchain from the node, showing all mined blocks and stored transactions.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'RONIT SANTWANI's Workspace' at the top, followed by 'Collections', 'Environments', 'History', and 'Flows'. Below these are 'Files' and a 'BETA' button. The main area has tabs for 'New' and 'Import' at the top. A search bar says 'Search collections' with a plus sign. Under 'My Collection', there are two items: 'GET Get data' and 'POST Post data'. The 'GET Get data' item is selected. At the top right, it says 'GET http://127.0.0.1:5001/get_chain'. Below that is a 'Send' button. The 'Params' tab is selected under the request details. In the 'Query Params' table, there are columns for 'Key' and 'Value'. The body section shows a response with a status of '200 OK' and a timestamp of '2026-02-17 19:49:51.660174'. The JSON response is as follows:

```
1  {"chain": [{"index": 1, "previous_hash": "0", "proof": 1, "timestamp": "2026-02-17 19:49:51.660174", "transactions": []}], "length": 1}
```

2)/mine_block

This GET request performs the mining process using Proof-of-Work and adds a new block to the blockchain.

RONIT SANTWANI's Workspace

New Import

GET Get data | GET http://127.0.0.1:5001/mine_block | + | No environment

Collections

Environments

History

Flows

Files BETA

My Collection

GET Get data

POST Post data

HTTP http://127.0.0.1:5001/mine_block

GET http://127.0.0.1:5001/mine_block

Docs Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body

200 OK 130 ms 464 B

{ } JSON Preview Visualize

```

1 {
2   "block": {
3     "index": 2,
4     "previous_hash": "08237b31300fd66c447e8e9303408c62843b9262b5abb0d7261ef
5       f23a3c0801",
6     "proof": 533,
7     "timestamp": "2026-02-17 19:56:11.715021",
8     "transactions": [
9       {
10         "amount": 1,
11         "receiver": "Ronit Santwani",
12         "sender": "dfa3ed1691b94e369e0272178effcd9a"
13       }
14     ],
15     "message": "Block mined successfully!"
16   }
}

```

3)/is_valid

This GET request checks whether the current blockchain is valid and untampered.

RONIT SANTWANI's Workspace

New Import

GET Get data | GET http://127.0.0.1:5001/is_valid | + | No environment

Collections

Environments

History

Flows

Files BETA

My Collection

GET Get data

POST Post data

HTTP http://127.0.0.1:5001/is_valid

GET http://127.0.0.1:5001/is_valid

Docs Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body

200 OK 8 ms 200 B

{ } JSON Preview Visualize

```

1 {
2   "message": "Blockchain is valid."
3 }

```

4)/add_transaction

This POST request submits a new transaction (sender, receiver, amount) to be added to the next mined block.

The screenshot shows the Postman application interface. On the left sidebar, under 'Collections', there is a section for 'My Collection' containing a 'POST Post data' endpoint. The main workspace shows a POST request to 'http://127.0.0.1:5001/add_transaction'. The request body is set to 'JSON' and contains the following JSON payload:

```
1 {
2   "sender": "Ronit",
3   "receiver": "Ronit Santwani",
4   "amount": 10
5 }
```

Below the request, the response status is '201 CREATED' with a response time of '28 ms' and a response size of '221 B'. The response body is also in JSON format, showing a success message:

```
1 {
2   "message": "Transaction will be added to Block 3"
3 }
```

5)/connect_node

This POST request connects the current node with other peer nodes in the blockchain network.

The screenshot shows the Postman application interface. On the left sidebar, under 'Collections', there is a section for 'My Collection' containing a 'POST Post data' endpoint. The main workspace shows a POST request to 'http://127.0.0.1:5001/connect_node'. The request body is set to 'JSON' and contains the following JSON payload:

```
1 {
2   "nodes": ["http://127.0.0.1:5002"]
3 }
```

Below the request, the response status is '201 CREATED' with a response time of '9 ms' and a response size of '218 B'. The response body is also in JSON format, showing a success message:

```
1 {
2   "message": "All nodes connected successfully!"
3 }
```

6)/replace_chain

This GET request checks all connected peer nodes and replaces the current blockchain with the longest valid chain, ensuring network-wide consistency.

The screenshot shows the Postman interface with a collection named "My Collection". A GET request is made to "http://127.0.0.1:5001/replace_chain". The response is a 200 OK status with a response time of 8 ms and a body size of 337 B. The response body is a JSON object:

```
1  {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "0",
6       "proof": 1,
7       "timestamp": "2026-02-17 20:08:18.459118",
8       "transactions": []
9     }
10  ],
11  "message": "Chain checked",
12  "student_name": "Ronit Santwani"
13 }
```

Conclusion:

In this experiment, a basic cryptocurrency system was designed and implemented using Python by applying fundamental blockchain principles such as block generation, cryptographic hashing, Proof-of-Work mining, transaction processing, and decentralized networking. A Flask-based framework was used to enable communication between multiple nodes, with each node maintaining its own copy of the blockchain ledger.

The mining process required solving the Proof-of-Work challenge, after which new blocks were added to the chain and miners received rewards. The system also supported transaction validation and inclusion of verified transactions into newly mined blocks. Through this implementation, the experiment offered hands-on insight into the functioning of blockchain technology, demonstrating how consensus, decentralization, and mining together enable the secure operation of cryptocurrencies in distributed environments.