Experiment 3

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

Theory:

Blockchain is a distributed and decentralized ledger technology used to record transactions securely across a network of computers (nodes). Each block in a blockchain contains a set of transactions, a timestamp, a cryptographic hash of the previous block, and a proof value generated through a consensus mechanism. The linking of blocks using cryptographic hashes ensures data integrity, immutability, and resistance to tampering.

Cryptocurrency is a digital currency built on blockchain technology, where transactions are verified and added to the blockchain through a process called mining. Mining involves solving a computationally difficult mathematical problem known as Proof of Work (PoW). In PoW, miners compete to find a valid proof by repeatedly hashing data until a hash with specific characteristics (such as leading zeros) is produced. The miner who finds the valid proof first is rewarded with cryptocurrency.

In this experiment, a basic blockchain is implemented using Python and Flask to simulate a cryptocurrency network. Each node maintains its own copy of the blockchain and communicates with other nodes using REST APIs. Transactions are added to a pending list and are included in a new block when mining occurs. SHA-256 hashing is used to secure blocks, and chain validation ensures that the blockchain remains consistent and unaltered. The consensus algorithm follows the longest-chain rule, where nodes replace their chain with the longest valid chain available in the network, ensuring agreement across all nodes.

Implementation:

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

    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
```

```
        }
        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

    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()
            ).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 SETUP ---------------- #
app = Flask(__name__)
node_address = str(uuid4()).replace('-', '')
blockchain = Blockchain()

# ---------------- API ROUTES ---------------- #
@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='Network',
        receiver=node_address,
        amount=1    )
    block = blockchain.create_block(proof, previous_hash)
    response = {
        'message': 'Block mined successfully!',
        'block': block    }
    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():
    valid = blockchain.is_chain_valid(blockchain.chain)
    if valid:
        return jsonify({'message': 'Blockchain is valid'}), 200
    else:
        return jsonify({'message': 'Blockchain is not valid'}), 200

@app.route('/add_transaction', 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 fields', 400
    index = blockchain.add_transaction(
        json_data['sender'],
        json_data['receiver'],
        json_data['amount']
    )
    return jsonify({
        'message': f'Transaction will be added to Block {index}'
    }), 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)
    return jsonify({
        'message': 'All nodes connected',
        'total_nodes': list(blockchain.nodes)
    }), 201

@app.route('/replace_chain', methods=['GET'])
def replace_chain():
    replaced = blockchain.replace_chain()
    if replaced:
        return jsonify({
            'message': 'Chain was replaced',
            'new_chain': blockchain.chain
        }), 200
    else:
        return jsonify({
            'message': 'Chain is already the longest',
            'chain': blockchain.chain
        }), 200

# ---------------- RUN APP ---------------- #
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Node 1

```
PS C:\Users\INFT511-01\Documents\chain> python blockchain.py
>>
 * Serving Flask app 'blockchain'
 * 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
```

Node 2

```
PS C:\Users\INFT511-01\Documents\chain> python blockchain.py
>>
 * Serving Flask app 'blockchain'
 * 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://192.168.34.227:5001
```

Node 3

```
PS C:\Users\INFT511-01\Documents\chain> python blockchain.py
>>
 * Serving Flask app 'blockchain'
 * 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://192.168.34.227:5002
```

Get chain

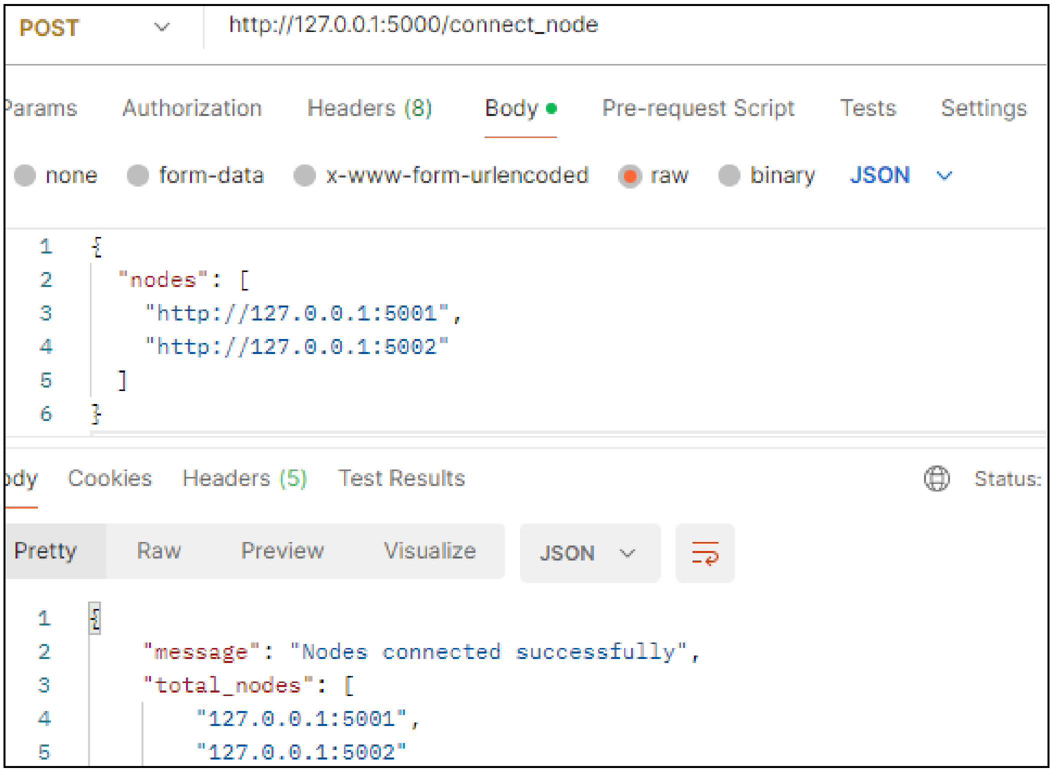
127.0.0.1:5000/get_chain

Apps Docs Gmail Drive Sheets

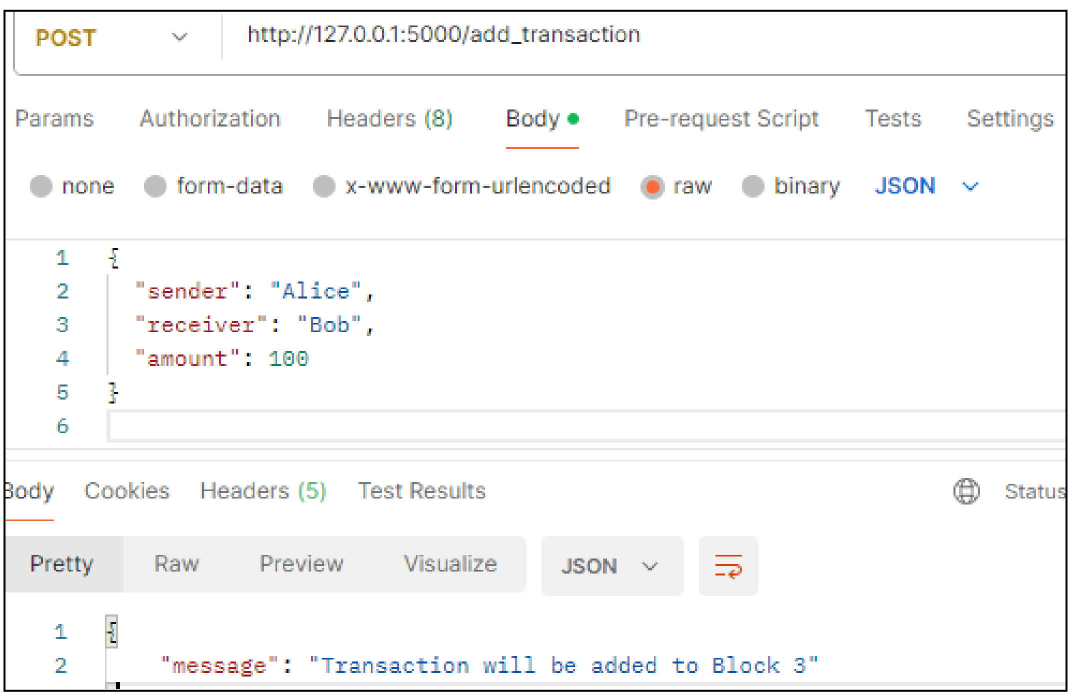
Pretty-print


```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-02-09 12:34:48.291413",
      "transactions": []
    }
  ],
  "length": 1
```


Connect



Transaction



Mine Block

127.0.0.1:5000/mine_block

Apps Docs Gmail Drive Sheets Slides Sheets GitHub

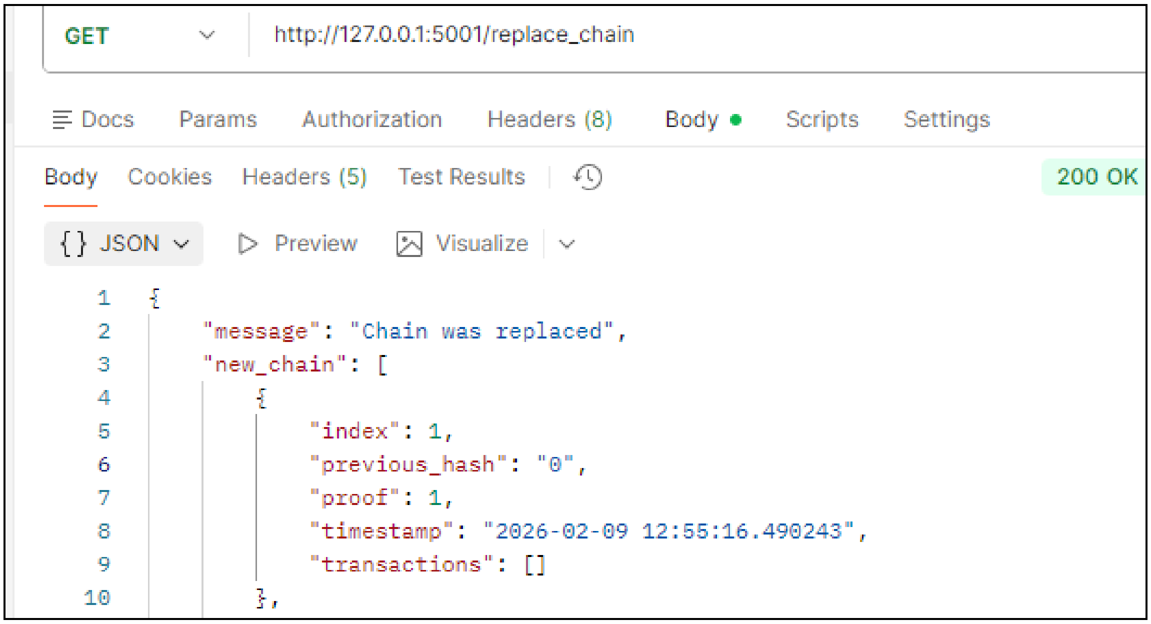
Pretty-print


```
{
  "block": {
    "index": 3,
    "previous_hash": "df5ae6a4b5515b71a4acb754e7324e2cb6f3d03fea46650cf5e1360824d540f1",
    "proof": 45293,
    "timestamp": "2026-02-09 12:46:20.599564",
    "transactions": [
      {
        "amount": 100,
        "receiver": "Bob",
        "sender": "Alice"
      },
      {
        "amount": 1,
        "receiver": "3f290029ead44559a9ffdced529bf8e0",
        "sender": "SYSTEM"
      }
    ]
  },
  "message": "Block mined successfully!"
```


Get Chain to verify


```
    {
      "index": 3,
      "previous_hash": "df5ae6a4b5515b71a4acb754e7324e2cb6f3d03fea46650cf5e1360824d540f1",
      "proof": 45293,
      "timestamp": "2026-02-09 12:46:20.599564",
      "transactions": [
        {
          "amount": 100,
          "receiver": "Bob",
          "sender": "Alice"
        },
        {
          "amount": 1,
          "receiver": "3f290029ead44559a9ffdced529bf8e0",
          "sender": "SYSTEM"
```


Replace Chain


GET
http://127.0.0.1:5001/replace_chain
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Body
Cookies
Headers (5)
Test Results
200 OK
{} JSON
Preview
Visualize
{
"message": "Chain was replaced",
"new_chain": [
{
"index": 1,
"previous_hash": "0",
"proof": 1,
"timestamp": "2026-02-09 12:55:16.490243",
"transactions": []
},

Get chain

127.0.0.1:5000/get_chain

Pretty-print


```
      "proof": 533,
      "timestamp": "2026-02-09 13:12:14.418127",
      "transactions": [
        {
          "amount": 50,
          "receiver": "Khushbu",
          "sender": "Nikita"
        },
        {
          "amount": 120,
          "receiver": "Krishang",
          "sender": "Nikita"
        },
        {
          "amount": 12,
          "receiver": "Doechi",
          "sender": "Bob"
        },
        {
          "amount": 1,
          "receiver": "Nikita",
          "sender": "9ff3cc05279b4d10b332bc36fa0a146c"
        }
      ]
    },
    {
      "index": 3,
      "previous_hash": "499a28635ebf5922c113a76deb4ade53bd84ed0d58327525c0ed58edfaebab9b",
      "proof": 45293,
      "timestamp": "2026-02-09 13:24:55.996815",
      "transactions": [
        {
          "amount": 1,
          "receiver": "Nikita",
          "sender": "9ff3cc05279b4d10b332bc36fa0a146c"
        }
      ]
    }
  ],
  "length": 3
}
```


Conclusion:

In this experiment, a simple cryptocurrency system was successfully created using Python by implementing core blockchain concepts such as block creation, transaction management, mining using Proof of Work, and chain validation. The distributed nature of the blockchain was demonstrated by connecting multiple nodes and achieving consensus through the longest-chain rule. Mining rewarded the node with cryptocurrency, validating the working of a decentralized ledger. Overall, the experiment provided a practical understanding of how blockchain and cryptocurrency systems function in a secure and decentralized environment.