Experiment 3

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

THEORY:

1. Blockchain Introduction

Blockchain is a decentralized and distributed ledger technology used for securely storing transaction records across multiple computers called nodes. Unlike traditional systems that depend on a central authority, every node in a blockchain network maintains an identical copy of the ledger.

Each block in the blockchain consists of:

- A collection of transactions
- The time at which the block was created
- The hash of the preceding block
- A unique cryptographic hash of the current block

Blocks are connected through cryptographic hashes, forming a continuous chain. If any block data is altered, its hash changes, causing a mismatch with subsequent blocks. This makes unauthorized modification easily detectable and ensures transparency, security, and immutability of data.

2. Mining Process

Mining is the mechanism through which new blocks are created and appended to the blockchain.

It includes:

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

In the Proof-of-Work method, miners repeatedly modify a nonce value to produce a hash that satisfies a predefined difficulty condition, such as a specific number of leading zeros.

Once a valid hash is discovered, the block is broadcast to the network and added to the blockchain.

Miners are rewarded with cryptocurrency for their computational contribution.

3. Multi-Node Blockchain Setup

In this experiment, a decentralized blockchain environment is implemented using three independent nodes running on different ports (5001, 5002, and 5003).

Each node:

- Functions independently
- Stores its own copy of the blockchain
- Exchanges information with peer nodes

This configuration demonstrates how blockchain networks operate without a centralized server while maintaining data synchronization across nodes.

4. Consensus Mechanism

To maintain uniformity across the network, a consensus strategy known as the Longest Valid Chain Rule is applied.

If multiple blockchain versions exist, the node adopts the longest chain that is verified to be valid. This approach allows the network to resolve temporary inconsistencies and ensures that all nodes eventually agree on a single, consistent transaction history.

5. Transactions and Mining Incentives

A transaction in a blockchain-based currency system contains:

- Sender’s address
- Recipient’s address

● Amount to be transferred

When a new block is mined, all pending transactions are recorded within it. In addition, a reward transaction is generated to credit the miner with newly created cryptocurrency. This incentive encourages participants to actively support and secure the blockchain network

CODE:

```
import datetime
import hashlib
import json
import requests
from flask import Flask, jsonify, request
from uuid import uuid4
from urllib.parse import urlparse
class SimpleBlockchain:
def __init__(self):
self.ledger = []
self.pending_txns = []
self.network_nodes = set()
self._create_genesis_block()
def _create_genesis_block(self):
self.create_block(proof=1, prev_hash="0")
def create_block(self, proof, prev_hash):
block = {
"block_no": len(self.ledger) + 1,
```

```
"time": str(datetime.datetime.now()),
"proof": proof,
"prev_hash": prev_hash,
"transactions": self.pending_txns
}
self.pending_txns = []
self.ledger.append(block)
return block
def last_block(self):
return self.ledger[-1]
def compute_proof(self, last_proof):
new_proof = 1
while True:
guess = hashlib.sha256(
str(new_proof**2 - last_proof**2).encode()
).hexdigest()
if guess[:4] == "0000":
return new_proof
new_proof += 1
def generate_hash(self, block):
encoded = json.dumps(block, sort_keys=True).encode()
return hashlib.sha256(encoded).hexdigest()
def validate_chain(self, chain):
prev_block = chain[0]
```

```
idx = 1
while idx < len(chain):
curr_block = chain[idx]
if curr_block["prev_hash"] != self.generate_hash(prev_block):
return False
prev_proof = prev_block["proof"]
curr_proof = curr_block["proof"]
check_hash = hashlib.sha256(
str(curr_proof**2 - prev_proof**2).encode()
).hexdigest()
if check_hash[:4] != "0000":
return False
prev_block = curr_block
idx += 1
return True
def add_txn(self, sender, receiver, amount):
self.pending_txns.append({
"from": sender,
"to": receiver,
"value": amount
})
return self.last_block()["block_no"] + 1
def register_node(self, node_url):
parsed = urlparse(node_url)
```

```
self.network_nodes.add(parsed.netloc)

def sync_chain(self):

longest = None

max_len = len(self.ledger)

for node in self.network_nodes:

response = requests.get(f"http://{node}/get_chain")

if response.status_code == 200:

length = response.json()["length"]

chain = response.json()["chain"]

if length > max_len and self.validate_chain(chain):

max_len = length

longest = chain

if longest:

self.ledger = longest

return True

return False

app = Flask(__name__)

node_id = str(uuid4()).replace("-", "")

blockchain = SimpleBlockchain()

@app.route("/mine", methods=["GET"])

def mine():

last_block = blockchain.last_block()

proof = blockchain.compute_proof(last_block["proof"])

prev_hash = blockchain.generate_hash(last_block)
```

```
blockchain.add_txn(
sender=node_id,
receiver="Ria",
amount=1
)
block = blockchain.create_block(proof, prev_hash)
return jsonify({
"status": "Block mined successfully!",
"block": block
}), 200
@app.route("/transaction", methods=["POST"])
def create_transaction():
data = request.get_json()
required = ["sender", "receiver", "amount"]
if not all(k in data for k in required):
return "Invalid transaction data", 400
block_no = blockchain.add_txn(
data["sender"],
data["receiver"],
data["amount"]
)
return jsonify({
"message": f"Transaction will be added to block {block_no}"
}), 201
```

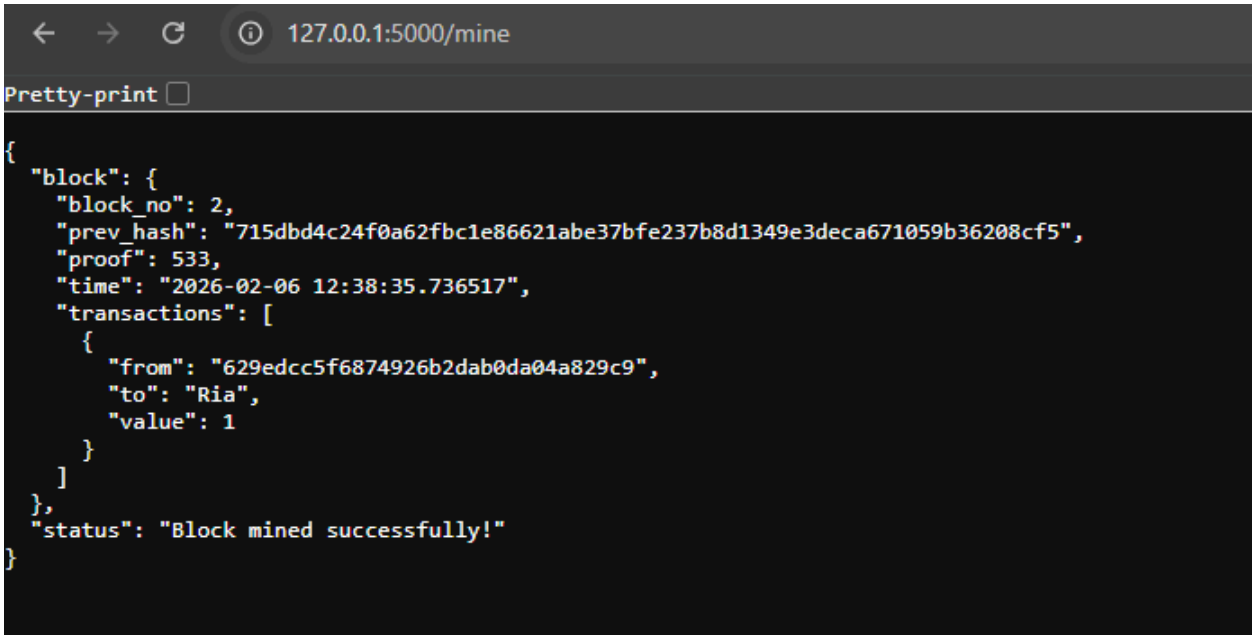
```
@app.route("/connect", methods=["POST"])
def connect_nodes():
data = request.get_json()
nodes = data.get("nodes")
if not nodes:
return "No nodes provided", 400
for node in nodes:
blockchain.register_node(node)
return jsonify({
"message": "Nodes connected successfully",
"nodes": list(blockchain.network_nodes)
}), 201
@app.route("/sync", methods=["GET"])
def sync():
replaced = blockchain.sync_chain()
return jsonify({
"replaced": replaced,
"chain": blockchain.ledger
}), 200
@app.route("/get_chain", methods=["GET"])
def get_chain():
return jsonify({
"length": len(blockchain.ledger),
"chain": blockchain.ledger
```

}), 200

if __name__ == "__main__":

app.run(host="0.0.0.0", port=5000, debug=True)

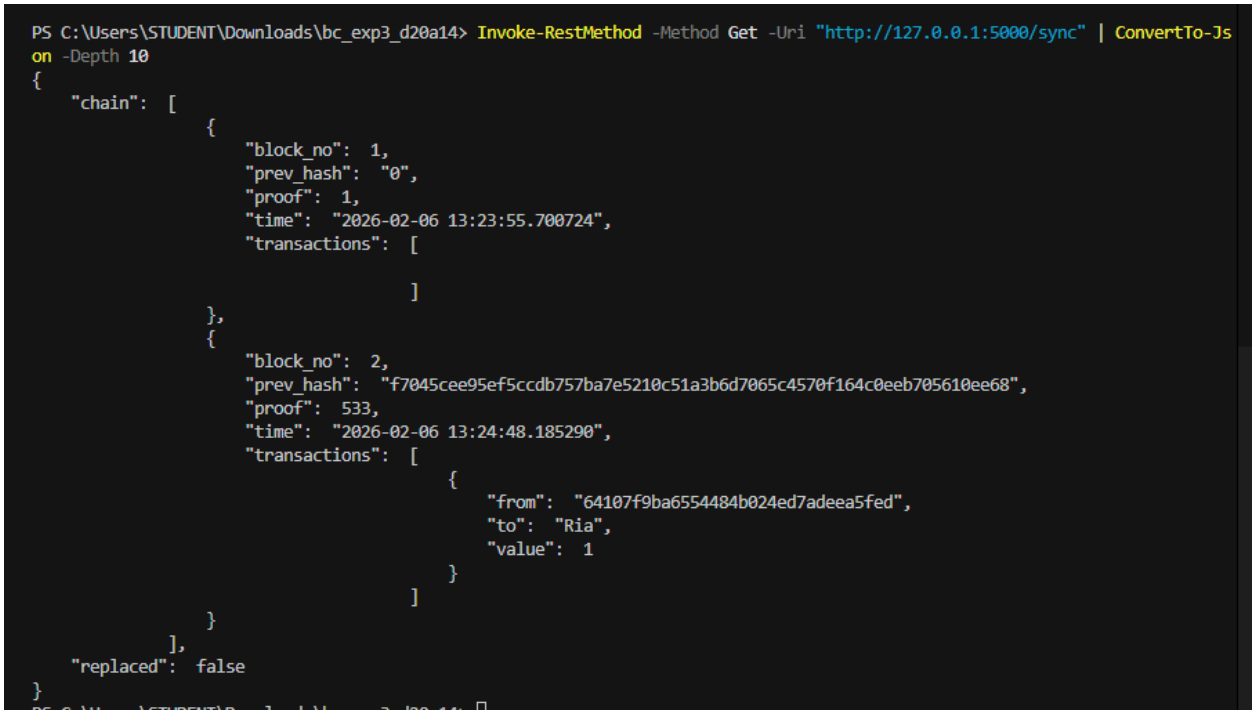
OUTPUT:



```
PS C:\Users\STUDENT\Downloads\bc_exp3_d20a14> Invoke-RestMethod -Method Get -Uri "http://127.0.0.1:5000/get_chain" | Convert
To-Json -Depth 10
{
    "chain":  [
                  {
                      "block_no":  1,
                      "prev_hash":  "0",
                      "proof":  1,
                      "time":  "2026-02-06 12:37:44.036978",
                      "transactions":  [

                                       ]
                  },
                  {
                      "block_no":  2,
                      "prev_hash":  "715dbd4c24f0a62fbc1e86621abe37bfe237b8d1349e3deca671059b36208cf5",
                      "proof":  533,
                      "time":  "2026-02-06 12:38:35.736517",
                      "transactions":  [
                                           {
                                               "from":  "629edcc5f6874926b2dab0da04a829c9",
                                               "to":  "Ria",
                                               "value":  1
                                           }
                                       ]
```

```
PS C:\Users\STUDENT\Downloads\bc_exp3_d20a14> Invoke-RestMethod -Method Post `
>> -Uri "http://127.0.0.1:5000/transaction" `
>> -ContentType "application/json" `
>> -Body '{"sender":"Ria","receiver":"Sneha","amount":50000}'

message
-------
Transaction will be added to block 4
```

```
PS C:\Users\STUDENT\Downloads\bc_exp3_d20a14> Invoke-RestMethod -Method Post `
>> -Uri "http://127.0.0.1:5000/connect" `
>> -ContentType "application/json" `
>> -Body '{"nodes":["http://127.0.0.1:5001"]}'

message                       nodes
-------                       -----
Nodes connected successfully {127.0.0.1:5001}
```



CONCLUSION:

In this experiment, we implemented a simple blockchain network across two devices connected within the same network. Each device ran its own blockchain node on port 5000 with endpoints to add transactions, mine blocks, and replace the chain. Using cURL requests, we tested the functionality by successfully adding transactions, mining them into blocks, and attempting to synchronize both devices using the replace chain mechanism. This allowed us to understand how blockchain ensures decentralization, immutability, and consensus across multiple nodes. Although simplified compared to real-world systems, the experiment demonstrated the core principles of blockchain, including how nodes communicate and maintain a consistent ledger, thereby fulfilling the objective of exploring blockchain in a distributed environment.