Vedant Sanap
D20A 50

Blockchain 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 ledger** that stores information in a series of linked blocks.

Each block contains:

- Transaction data
- Timestamp
- Previous block's hash
- Its own unique hash (digital fingerprint)

Once data is recorded in a blockchain, it becomes **immutable** because altering one block would require recalculating all subsequent blocks.

2. Mining

Mining is the process of:

1. Collecting pending transactions into a block.
2. Performing a computational puzzle (Proof-of-Work) to find a valid hash.
3. Adding the new block to the blockchain.
   Broadcasting it to all connected peers.

Miners are rewarded with cryptocurrency for successfully mining a block.

3. Multi-Node Blockchain Network

In this lab, we simulate **three independent blockchain nodes** (5001, 5002, 5003).

Each node:

- Runs on a separate port.
- Maintains its own copy of the blockchain.
- Can connect with peers to share and validate blocks.

4. Consensus Mechanism

We use the **Longest Chain Rule**:

- If multiple versions of the chain exist, the **longest valid chain** is chosen.
- This ensures all nodes agree on a single transaction history.

5. Transactions & Mining Reward

Each transaction has:

- Sender
- Receiver
- Amount

When mining a block:

- Pending transactions are added to the block.
- A **reward transaction** is added automatically to pay the miner.

6. Chain Replacement

When /replace_chain is called:

1. Node requests chains from peers.
2. If it finds a longer and valid chain, it replaces its own.
3. This keeps the blockchain consistent across all nodes.

7. Cryptographic Hashing

Hashing is a fundamental security feature of blockchain technology.
A **hash** is a fixed-length output generated from input data using a cryptographic hash function such as SHA-256.

In a blockchain:

- Each block contains the hash of the previous block.
- Any small change in block data changes its hash completely.
- This creates a secure chain where blocks are cryptographically linked.

Because of hashing, tampering with one block breaks the entire chain, making the system highly secure and tamper-resistant.

8. Proof-of-Work (PoW) Algorithm

Proof-of-Work is a consensus technique that prevents misuse of the blockchain network.
In PoW:

- Miners must find a hash that satisfies a specific difficulty condition (e.g., starts with "0000").

- This requires trying many random values (nonces) until the correct hash is found.

- The first miner to solve the puzzle gets the right to add the block.

PoW ensures that adding a block requires real computational effort, which protects the network from spam and fraud.

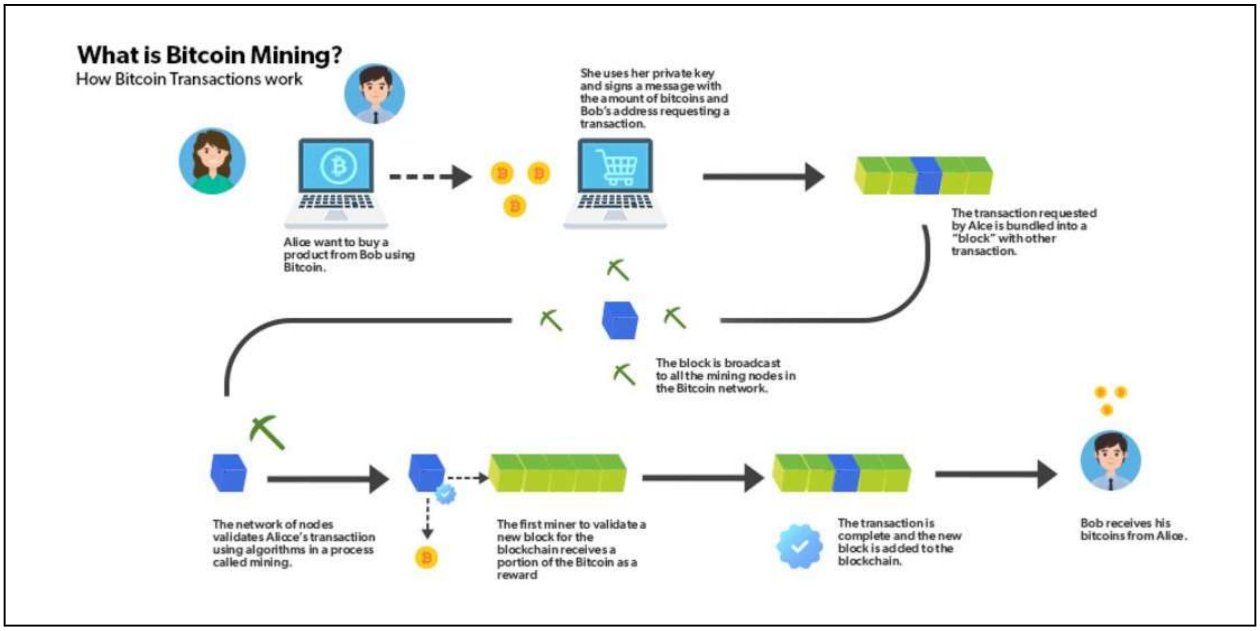
9. Block Structure

Each block in the blockchain consists of the following fields:

- **Index:** Position of the block in the chain
- **Timestamp:** Time of creation
- **Data:** List of verified transactions
- **Nonce:** Value used in PoW
- **Previous Hash:** Hash of the previous block
- **Hash:** Cryptographic hash of the current block

This structure ensures data integrity and traceability.

Mining:



Alice want to buy a product from Bob using Bitcoin.

She uses her private key and signs a message with the amount of bitcoins and Bob's address requesting a transaction.

The transaction requested by Alce is bundled into a "block" with other transaction.

The block is broadcast to all the mining nodes in the Bitcoin network.

The network of nodes validates Alicce's transactiion using algorithms in a process called mining.

The first miner to validate a new block for the blockchain receives a portion of the Bitcoin as a reward

The transaction is complete and the new block is added to the blockchain.

Bob receives his bitcoins from Alice.

Code:

```
import datetime
import hashlib
import json
from flask import Flask, jsonify, request
import requests
from         uuid         import         uuid4
# Generate a unique id that is in hex
from      urllib.parse      import      urlparse
# To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:

  def  __init__(self):
    self.chain = []
                self.transactions   =   []
#  Adding  transactions  before  they  are
added to a block
            self.create_block(proof  =  1,
previous_hash = '0')
                  self.nodes   =   set()
#  Set  is  used  as  there  is  no  order  to  be
maintained  as  the  nodes  can  be  from  all
around the globe


          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}
```

```
#   Adding   transactions   to   make   the
blockchain a cryptocurrency
                self.transactions   =   []
#  The  list  of  transacction  should  become
empty after they are added to a block
    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 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

  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

     # This method will add the trqnsaction to
the list of trnsactions
              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
# It will return the block index to which the
transaction should be added

          #  This  function  will  add  the  node
containing  an  address  to  the  set  of  nodes
created in init function
    def add_node(self, address):
                   parsed_url  =  urlparse(address)
#  urlparse  will  parse  the  url  from  the
address
                  self.nodes.add(parsed_url.netloc)
# Add is used and not append as it's a set.
Netloc will only return '127.0.0.1:5000'

      # Consensus Protocol. This function will
replace all the shorter chain with the longer
chain in all the nodes on the network
    def replace_chain(self):
                                network   =   self.nodes
# network variable is the set of nodes all
around the globe
                               longest_chain   =   None
# It will hold the longest chain when we
scan the network
                      max_length  =  len(self.chain)
# This will hold the length of the chain held
by the node that runs this function
        for node in network:
                                           response    =
requests.get(f'http://{node}/get_chain')
# Use get chain method already created to
get the length of the chain
            if   response.status_code   ==   200:
                   length = response.json()['length']
# Extract the length of the chain from
get_chain fiunction
                chain = response.json()['chain']
                          if length > max_length and
self.is_chain_valid(chain):          # We check
if the length is bigger and if the chain is valid
then


                                   max_length  =  length
# We update the max length
                              longest_chain  =  chain
# We update the longest chain
                                      if     longest_chain:
# If longest_chain is not none that means it
was replaced
                          self.chain  =  longest_chain
# Replace the chain of the current node with
the longest chain
```

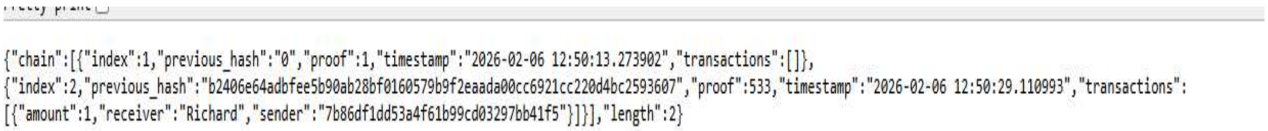
```
          return True
                                        return False
# Return false if current chain is the longest
one

# Part 2 - Mining our Blockchain

# Creating a Web App
app = Flask(__name__)

# Creating an address for the node on Port
5000. We will create some other nodes as
well on different ports
node_address = str(uuid4()).replace('-', '')
#

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@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 =
node_address, receiver = 'Richard', amount
= 1) # Hadcoins to mine the block (A
Reward). So the node gives 1 hadcoin to
Abcde for mining the block
      block = blockchain.create_block(proof,
previous_hash)
    response = {'message': 'Congratulations,
you just mined a block!',
            'index': block['index'],
            'timestamp': block['timestamp'],
            'proof': block['proof'],
                                'previous_hash':
block['previous_hash'],
                                  'transactions':
block['transactions']}
   return jsonify(response), 200

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

# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
                                is_valid =
blockchain.is_chain_valid(blockchain.chain)
   if is_valid:
        response = {'message': 'All good. The
Blockchain is valid.'}
   else:
         response = {'message': 'Houston, we
have a problem. The Blockchain is not
valid.'}
   return jsonify(response), 200

# Adding a new transaction to the
Blockchain
@app.route('/add_transaction', methods =
['POST'])                    # Post method
as we have to pass something to get
something in return
def add_transaction():
   json = request.get_json()
 # This will get the json file from postman. In
```

```
Postman we will create a json file in which
we will #pass the values for the keys in the
json file
     transaction_keys = ['sender', 'receiver',
'amount']
         if not all(key in json for key in
transaction_keys):
 # Checking if all keys are available in json
     return 'Some elements of the transaction
are missing', 400
                                   index =
blockchain.add_transaction(json['sender'],
json['receiver'], json['amount'])
    response = {'message': f'This transaction
will be added to Block {index}'}
   return jsonify(response), 201
   # Code 201 for creation

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes
@app.route('/connect_node', methods =
['POST'])                          # POST
request to register the new nodes from the
json file
def connect_node():
   json = request.get_json()
               nodes = json.get('nodes')
# Get the nodes from json file
   if nodes is None:
      return "No node", 400
   for node in nodes:
      blockchain.add_node(node)
   response = {'message': 'All the nodes are
now connected. The Hadcoin Blockchain
now contains the following nodes:',
                                'total_nodes':
list(blockchain.nodes)}
   return jsonify(response), 201

# Replacing the chain by the longest chain if
needed
@app.route('/replace_chain', methods =
['GET'])
def replace_chain():
                      is_chain_replaced =
blockchain.replace_chain()
   if is_chain_replaced:
         response = {'message': 'The nodes
had different chains so the chain was
replaced by the longest one.',
              'new_chain': blockchain.chain}
   else:
       response = {'message': 'All good. The
chain is the largest one.',
                               'actual_chain':
blockchain.chain}
   return jsonify(response), 200

# Running the app
app.run(host = '0.0.0.0', port = 5000)
```

Output:

1. /get_chain (GET Request)

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

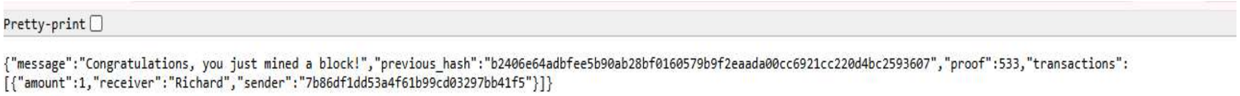


Pretty-print □

```
{"chain":[{"index":1,"previous_hash":"0","proof":1,"timestamp":"2026-02-06 12:50:13.273902","transactions":[]},
{"index":2,"previous_hash":"b2406e64adbfee5b90ab28bf0160579b9f2eaada00cc6921cc220d4bc2593607","proof":533,"timestamp":"2026-02-06 12:50:29.110993","transactions":
[{"amount":1,"receiver":"Richard","sender":"7b86df1dd53a4f61b99cd03297bb41f5"}]}],"length":2}
```

2. /mine_block (GET Request)

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

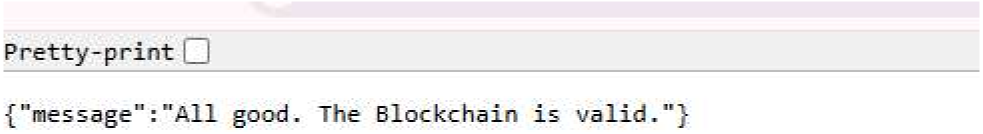


Pretty-print □

```
{"message":"Congratulations, you just mined a block!","previous_hash":"b2406e64adbfee5b90ab28bf0160579b9f2eaada00cc6921cc220d4bc2593607","proof":533,"transactions":
[{"amount":1,"receiver":"Richard","sender":"7b86df1dd53a4f61b99cd03297bb41f5"}]}
```

3. /is_valid (GET Request)

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

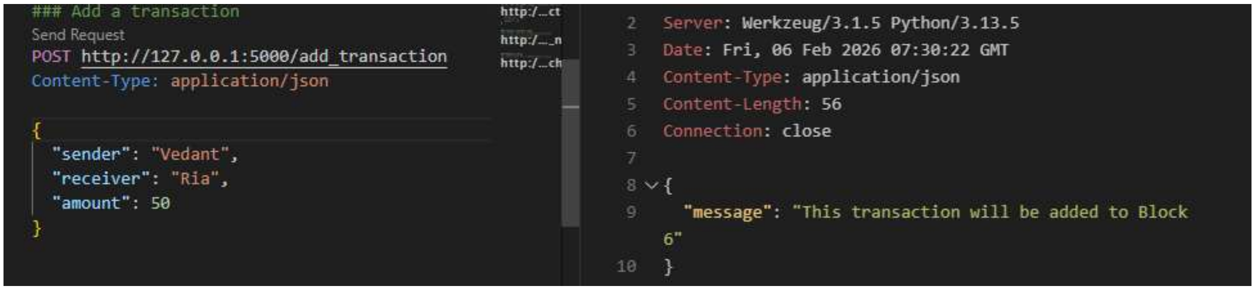


Pretty-print □

```
{"message":"All good. The Blockchain is valid."}
```

4. /add_transaction (POST Request)

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

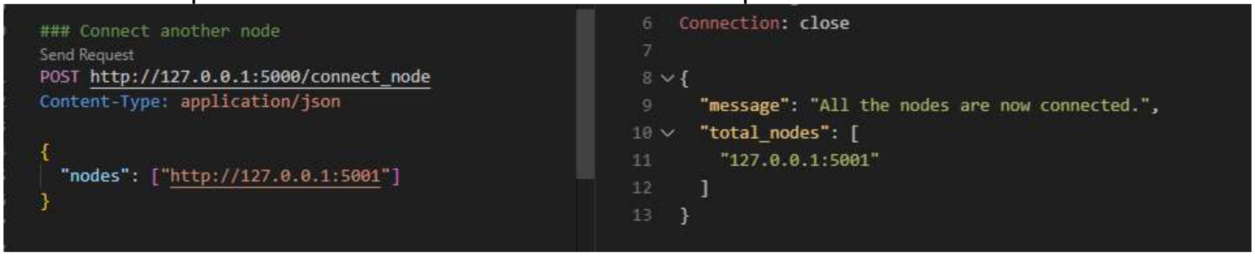


```
### Add a transaction
Send Request
POST http://127.0.0.1:5000/add_transaction
Content-Type: application/json

{
  "sender": "Vedant",
  "receiver": "Ria",
  "amount": 50
}
```

```
Server: Werkzeug/3.1.5 Python/3.13.5
Date: Fri, 06 Feb 2026 07:30:22 GMT
Content-Type: application/json
Content-Length: 56
Connection: close

{
  "message": "This transaction will be added to Block
6"
}
```

5. /connect_node (POST Request)

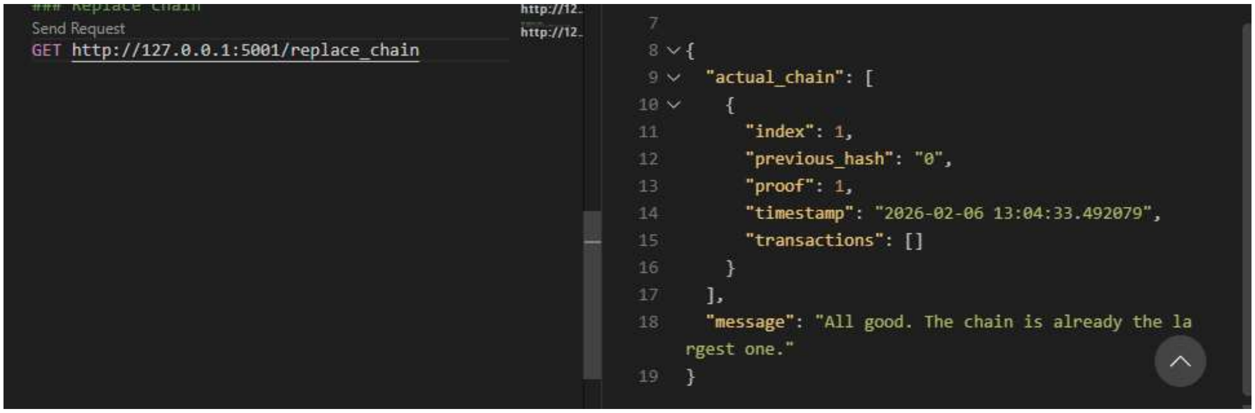
This POST request connects the current node with other peer nodes in the blockchain network.



```
### Connect another node
Send Request
POST http://127.0.0.1:5000/connect_node
Content-Type: application/json

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

```
Connection: close

{
  "message": "All the nodes are now connected.",
  "total_nodes": [
    "127.0.0.1:5001"
  ]
}
```

6. **/replace_chain (GET Request)**
This GET request checks all connected peer nodes and replaces the current blockchain with the longest valid chain, ensuring network-wide consistency.



```
### Replace chain
Send Request
GET http://127.0.0.1:5001/replace_chain
```

```
{
  "actual_chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-02-06 13:04:33.492079",
      "transactions": []
    }
  ],
  "message": "All good. The chain is already the la
rgest one."
}
```

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.