Ronak Katariya
D20A / 29

AIM: Create a Blockchain using Python

Theory:

1. Introduction to Blockchain

Blockchain is a revolutionary digital technology that works as a **shared, immutable ledger** for recording transactions. The term *blockchain* comes from its structure, where data is stored in **blocks**, and each block is securely linked to the previous one, forming a **chain of blocks**.

Each block contains transaction data, a timestamp, and a cryptographic hash. Because every block is connected using cryptographic hashes, any attempt to change data in one block will alter its hash and break the entire chain, making tampering **easy to detect**.

Blockchain operates as a **distributed and decentralized system**, meaning there is no central authority controlling it. Instead, multiple computers called **nodes** maintain identical copies of the blockchain, ensuring transparency, security, and trust.

Once a transaction is verified and added to the blockchain, it becomes **almost impossible to alter or delete**, ensuring data integrity.

2. What is a Block?

A **block** is a fundamental unit of a blockchain. It acts like a **record page** in a digital ledger, storing transaction details in a secure and encrypted manner.

In cryptocurrency systems, blocks store multiple transactions that occur over a certain period. These transactions are organized using cryptographic techniques and combined into a structure called a **Merkle Tree**, ensuring efficient verification and data integrity.

Each block is linked to the previous block, forming a continuous and secure chain.

3. Components of a Block

A block consists of two main parts: **Block Header** and **Block Body**.

3.1 Block Header

The block header uniquely identifies a block in the blockchain and contains essential metadata:

- **Previous Block Hash**
  Stores the cryptographic hash of the previous block, linking blocks together and maintaining chain integrity.
- **Timestamp**
  Records the exact date and time when the block was created.

- **Nonce**
  A number used only once, crucial in the Proof-of-Work mechanism. Miners continuously change

the nonce to find a valid hash.

- **Merkle Root**
  A single hash representing all transactions in the block, generated using a Merkle Tree. It allows efficient transaction verification.

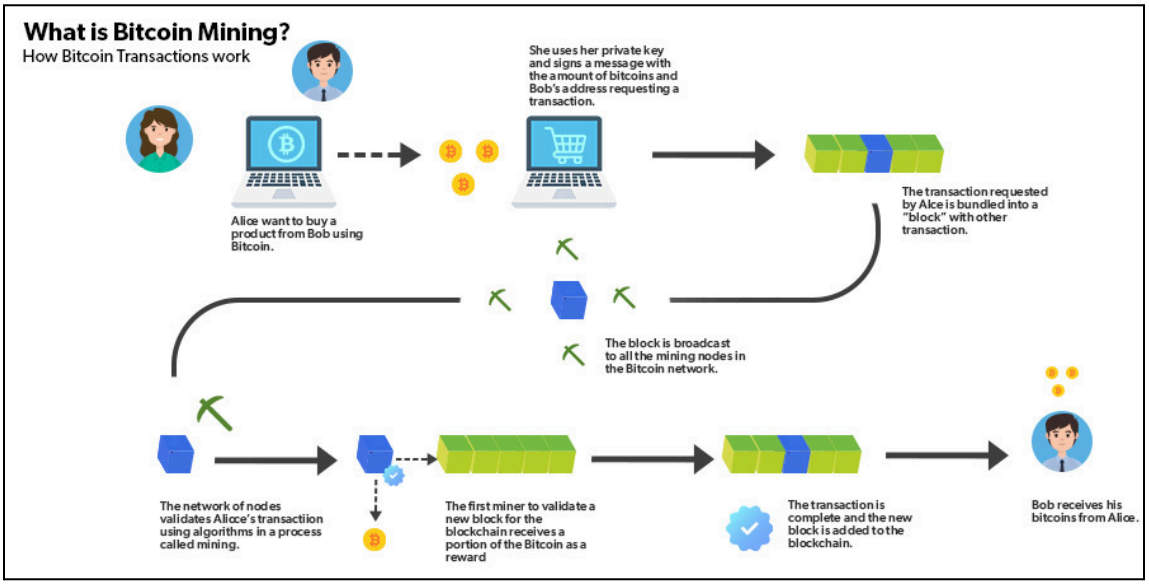
3.2 Block Body

The block body contains the **actual transaction data**, such as sender, receiver, and amount.

Example Block Structure

```
Block {
   Index: 2
   Timestamp: 2026-01-22 10:05:00
   Data: {"Sender":"Alice", "Receiver":"Bob", "Amount":50}
   Nonce: 102345
   Previous Hash: "0000a1b2c3d4e5..."
   Hash: "0000f3b7d6c1e8..."
}
```

4. Process of Mining



Mining is the process of adding a new block to the blockchain. It ensures network security and data integrity using the **Proof-of-Work (PoW)** consensus mechanism.

Step 1: Collect Transactions

Pending transactions from the network are gathered into a new block.

- Example: Alice sends 50 coins to Bob.

Ronak Katariya
D20A / 29

Step 2: Create Block Header

The block header is prepared containing:

- Previous block hash
- Transaction data
- Timestamp
- Nonce (initially set to 0)

Step 3: Proof of Work (PoW)

Miners repeatedly change the nonce and calculate the hash until it meets the difficulty requirement.

- **Difficulty**: Number of leading zeros required in the hash
- Example condition:

```
SHA256(block_data + nonce) → hash starts with "0000"
```

This process requires high computational effort, hence called **Proof-of-Work**.

Step 4: Validation and Broadcasting

Once a valid hash is found, the new block is broadcast to the network. Other nodes verify:

- Hash correctness
- Nonce validity
- Previous hash linkage

Step 5: Adding Block to Blockchain

If verified, the block is added to the blockchain, and the miner receives a **reward** (cryptocurrency).

5. Validity Checking of Blocks in Blockchain

To ensure the blockchain remains secure and trustworthy, blocks are continuously validated.

Step 1: Validate Previous Hash

Each block's previous_hash must match the hash of the previous block.

```
if Block[n].previous_hash != Hash(Block[n-1]):
    Invalid
```

Ronak Katariya
D20A / 29

Step 2: Recalculate Current Hash

The hash of the block is recalculated using its data and nonce.

```
if SHA256(Block[n].data + Block[n].nonce) != Block[n].hash:
   Invalid
```

Step 3: Check Proof-of-Work

Ensure the hash meets the difficulty condition.

```
if not Block[n].hash.startswith("0000"):
   Invalid
```

Step 4: Validate Entire Chain

All blocks from the **Genesis block** to the latest block are checked.

- If any block fails validation → the blockchain is invalid.

Step 5: Validate Genesis Block

The **Genesis block** is the first block in the blockchain.

- Previous hash must be "0"
- It is hardcoded and must never be altered

Code:

blockchain.py

```
import datetime
import hashlib
import json


class Blockchain:
    def __init__(self):
        self.chain = []
        self.difficulty = 4  # 4 leading zeroes
        self.create_genesis_block()

    def create_genesis_block(self):
        genesis_block = {
            'index': 1,
            'timestamp': str(datetime.datetime.now()),
            'data': 'Genesis Block',
```

```
            'nonce': 0,
            'previous_hash': '0'
        }
        genesis_block['hash'] = self.calculate_hash(genesis_block)
        self.chain.append(genesis_block)

    def calculate_hash(self, block):
        block_copy = block.copy()
        block_copy.pop('hash', None)
        encoded = json.dumps(block_copy, sort_keys=True).encode()
        return hashlib.sha256(encoded).hexdigest()

    def mine_block(self, data):
        previous_block = self.chain[-1]
        nonce = 0

        while True:
            block = {
                'index': len(self.chain) + 1,
                'timestamp': str(datetime.datetime.now()),
                'data': data,
                'nonce': nonce,
                'previous_hash': previous_block['hash']
            }

            block_hash = self.calculate_hash(block)

            if block_hash.startswith('0' * self.difficulty):
                block['hash'] = block_hash
                self.chain.append(block)
                return block

            nonce += 1

    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i - 1]

            if current['previous_hash'] != previous['hash']:
                return False

            recalculated_hash = self.calculate_hash(current)
            if recalculated_hash != current['hash']:
```

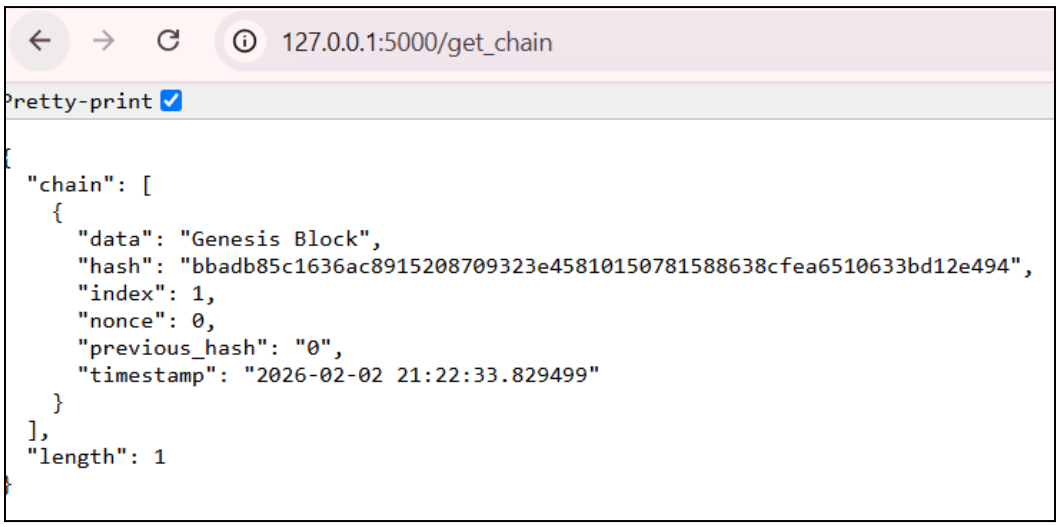
```
            return False

        if not current['hash'].startswith('0' * self.difficulty):
            return False

    return True
```

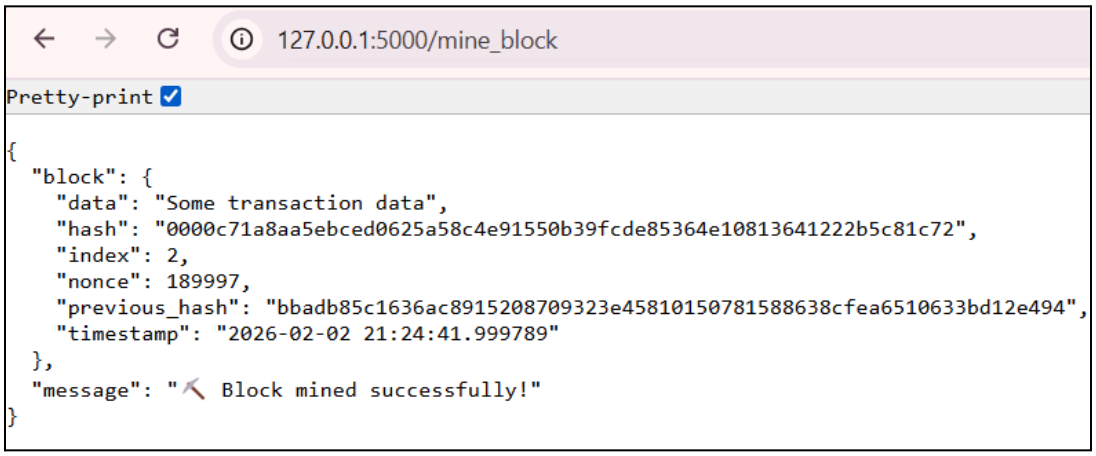
app.py

```
from flask import Flask, jsonify
from blockchain import Blockchain

app = Flask(__name__)
blockchain = Blockchain()


@app.route('/mine_block', methods=['GET'])
def mine_block():
    block = blockchain.mine_block("Some transaction data")

    return jsonify({
        'message': '⛏️ Block mined successfully!',
        'block': block
    }), 200


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


@app.route('/is_valid', methods=['GET'])
def is_valid():
    return jsonify({
        'is_valid': blockchain.is_chain_valid()
    }), 200


if __name__ == '__main__':
    print(" Mining blockchain with 4 leading zeroes...")
    app.run(debug=True, use_reloader=False)
```

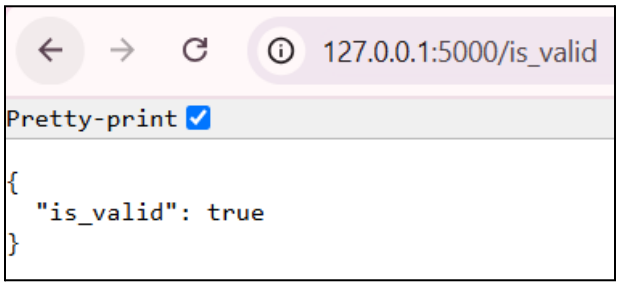
Ronak Katariya
D20A / 29

```
(.venv) PS C:\Users\272241\Desktop\Ronak\Sem8\BC\BC2\.venv> pip install flask
  Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting colorama
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: markupsafe, itsdangerous, colorama, blinker, werkzeug, jinja2
Successfully installed blinker-1.9.0 click-8.3.1 colorama-0.4.6 flask-3.1.2 itsdangerous-2.2
e-3.0.3 werkzeug-3.1.5

[notice] A new release of pip is available: 23.0.1 -> 26.0
```





Ronak Katariya
D20A / 29




```
(.venv) PS C:\Users\272241\Desktop\Ronak\Sem8\BC\BC2\.venv> python .\app.py
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [02/Feb/2026 21:22:49] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [02/Feb/2026 21:22:49] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [02/Feb/2026 21:23:03] "GET /get_mine HTTP/1.1" 404 -
127.0.0.1 - - [02/Feb/2026 21:23:44] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [02/Feb/2026 21:24:41] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [02/Feb/2026 21:25:21] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [02/Feb/2026 21:25:51] "GET /is_valid HTTP/1.1" 200 -
```

Conclusion:

Blockchain technology provides a **secure, transparent, and decentralized** method for storing digital data. By using cryptographic hashing, distributed consensus, and Proof-of-Work, blockchain ensures data integrity, prevents fraud, and eliminates the need for a central authority. This makes blockchain a powerful foundation for cryptocurrencies, smart contracts, and many modern digital applications.