Aim: Create blockchain using python

Theory:

What is Blockchain?

Blockchain is a revolutionary technology that functions as a shared, immutable digital ledger. The name "blockchain" comes from its structure data is organized in blocks, with each new block linked to the one before it, forming a continuous chain.

Each block contains crucial data, such as a list of transactions, a timestamp, and a unique identifier called a cryptographic hash. This hash is generated from the block's contents and the hash of the previous block, ensuring that each block is tightly connected to the one before it.

- Blockchain's linked structure makes data tampering detectable by altering hashes and breaking the chain.
- It acts as a distributed database, storing transactions across the network.
- Each transaction is verified by the majority, ensuring legitimacy.
- This decentralization prevents any single party from manipulating the data.

Blockchain is decentralized and distributed, meaning no single authority controls it. Instead, multiple computers (nodes) on a network each have a copy of the blockchain, keeping the ledger synchronized. This setup ensures that once data, like a transaction, is recorded and confirmed, it becomes immutable almost impossible to alter or delete

What is Block?

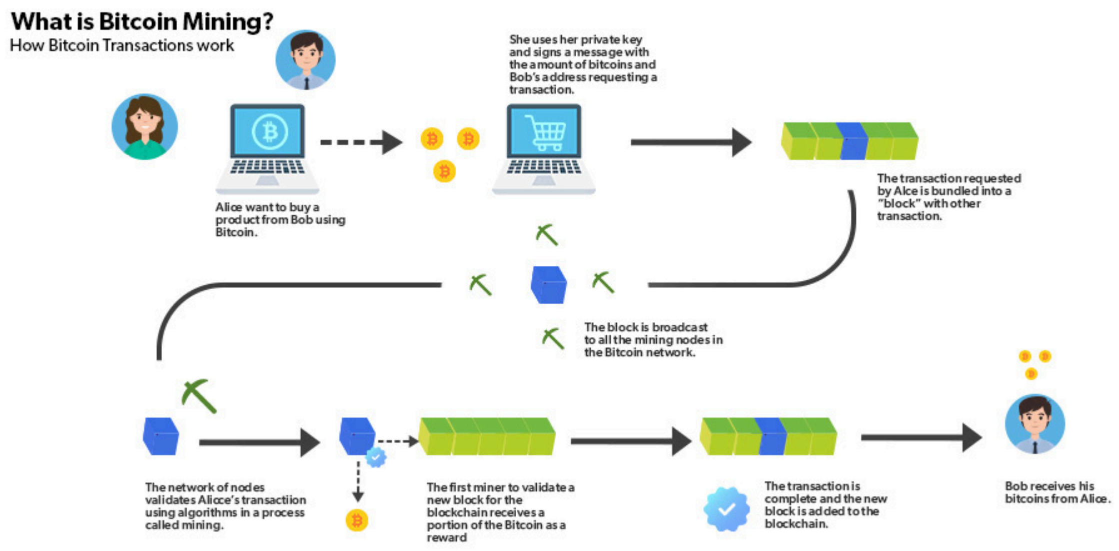
A block in a blockchain network is similar to a link in a chain. In the field of cryptocurrency, blocks are like records that store transactions like a record book, and those are encrypted into a hash tree. There are a huge number of transactions occurring every day in the world. The users need to keep track of those transactions, and they do it with the help of a block structure. The block structure of the blockchain is mentioned in the very first diagram in this article.

Components of block

1. Header: It is used to identify the particular block in the entire blockchain. It handles all blocks in the blockchain. A block header is hashed periodically by miners by changing the nonce value as part of normal mining activity, also Three sets of block metadata are contained in the block header.
2. Previous Block Address/ Hash: It is used to connect the i+1th block to the ith block using the hash. In short, it is a reference to the hash of the previous (parent) block in the chain.
3. Timestamp: It is a system that verifies the data into the block and assigns a time or date of creation for digital documents. The timestamp is a string of characters that uniquely identifies the document or event and indicates when it was created.
4. Nonce: A nonce number which is used only once. It is a central part of the proof of work in the block. It is compared to the live target if it is smaller or equal to the current target. People who mine, test, and eliminate many Nonce per second until they find that Valuable Nonce is valid.
5. Merkel Root: It is a type of data structure frame of different blocks of data. A Merkle Tree stores all the transactions in a block by producing a digital fingerprint of the entire transaction. It allows the users to verify whether a transaction can be included in a block or not.

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

Process of mining



What is Bitcoin Mining?
How Bitcoin Transactions work
She uses her private key and signs a message with the amount of bitcoins and Bob's address requesting a transaction.
Alice want to buy a product from Bob using Bitcoin.
The transaction requested by Alice 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 Alice's transaction 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.

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

Step 1: Collect Transactions

- All pending transactions are collected into a new block.
- Example: Alice sends 50 coins to Bob.

Step 2: Create Block Header

Block header includes:

- Previous hash
- Transactions
- Timestamp
- Nonce (initially 0)

Step 3: Proof of Work (PoW)

- The miner repeatedly changes the nonce to find a hash that satisfies the difficulty requirement.
- Difficulty = number of leading zeroes required in hash (e.g., 4 leading zeroes).

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

- This requires computational work, hence the name Proof-of-Work.

Step 4: Validate and Broadcast
- Once a valid hash is found, the block is broadcast to all nodes in the network.
- Other nodes verify:
  Hash correctness
    1. Nonce validity
    2. Previous hash link

Step 5: Add to Blockchain
- If all nodes validate, the block is added to the chain.
- The miner receives a reward (e.g., cryptocurrency).

How to check validity of blocks in blockchain?
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
```

Step 2: Recalculate Current Hash
- Compute hash of current block’s data and nonce.
- It must match the stored hash in the block.

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

Step 3: Check Proof-of-Work
- Verify that the block hash satisfies the difficulty (e.g., 4 leading zeroes).

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

Step 4: Validate Entire Chain
- Repeat steps 1–3 for all blocks from Genesis to the latest block.
- If any block fails → blockchain is invalid.

Step 5: Validate Genesis Block
- The first block (Genesis block) is hardcoded.
- Previous hash = "0"
- Must not be altered.

Code:
blockchain.py

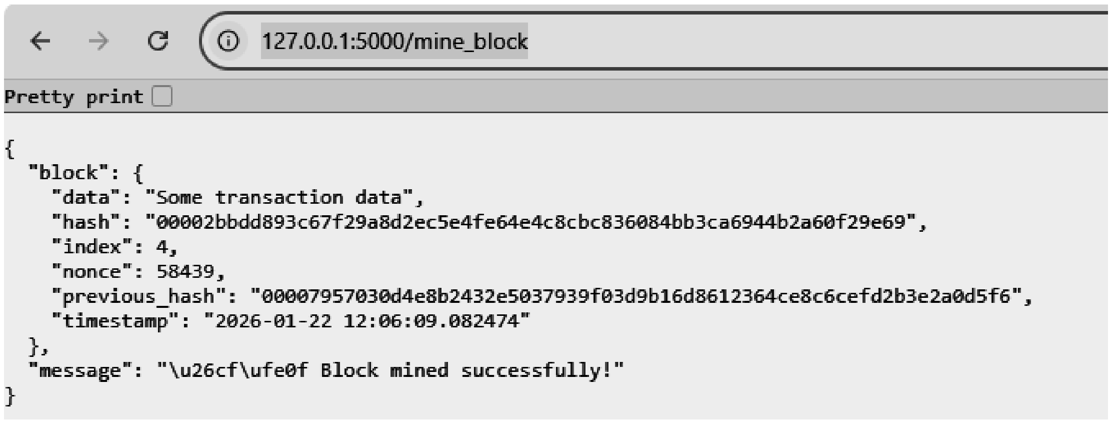
```
import datetime
import hashlib
import json
```

```
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

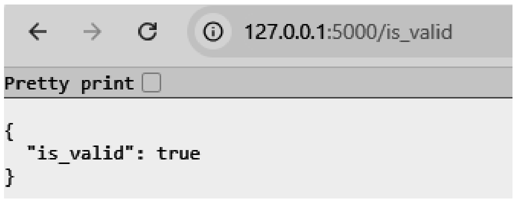
```
from flask import Flask, jsonify
from blockchain import Blockchain

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)
```



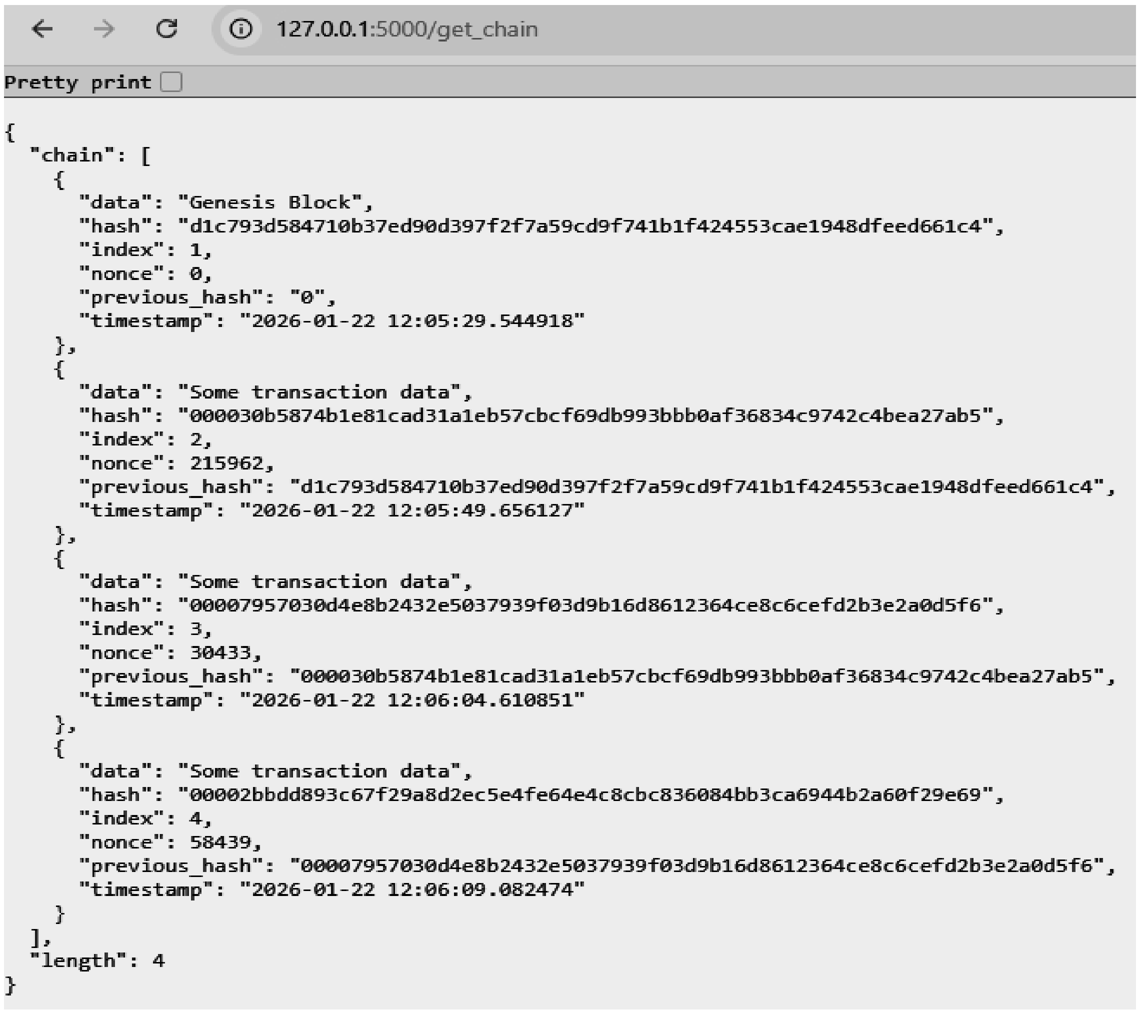
127.0.0.1:5000/mine_block

Pretty print

```
{
  "block": {
    "data": "Some transaction data",
    "hash": "00002bbdd893c67f29a8d2ec5e4fe64e4c8cbc836084bb3ca6944b2a60f29e69",
    "index": 4,
    "nonce": 58439,
    "previous_hash": "00007957030d4e8b2432e5037939f03d9b16d8612364ce8c6cefd2b3e2a0d5f6",
    "timestamp": "2026-01-22 12:06:09.082474"
  },
  "message": "\u26cf\ufe0f Block mined successfully!"
}
```



```
{
  "is_valid": true
}
```



```
{
  "chain": [
    {
      "data": "Genesis Block",
      "hash": "d1c793d584710b37ed90d397f2f7a59cd9f741b1f424553cae1948dfeed661c4",
      "index": 1,
      "nonce": 0,
      "previous_hash": "0",
      "timestamp": "2026-01-22 12:05:29.544918"
    },
    {
      "data": "Some transaction data",
      "hash": "000030b5874b1e81cad31a1eb57cbcf69db993bbb0af36834c9742c4bea27ab5",
      "index": 2,
      "nonce": 215962,
      "previous_hash": "d1c793d584710b37ed90d397f2f7a59cd9f741b1f424553cae1948dfeed661c4",
      "timestamp": "2026-01-22 12:05:49.656127"
    },
    {
      "data": "Some transaction data",
      "hash": "00007957030d4e8b2432e5037939f03d9b16d8612364ce8c6cefd2b3e2a0d5f6",
      "index": 3,
      "nonce": 30433,
      "previous_hash": "000030b5874b1e81cad31a1eb57cbcf69db993bbb0af36834c9742c4bea27ab5",
      "timestamp": "2026-01-22 12:06:04.610851"
    },
    {
      "data": "Some transaction data",
      "hash": "00002bbdd893c67f29a8d2ec5e4fe64e4c8cbc836084bb3ca6944b2a60f29e69",
      "index": 4,
      "nonce": 58439,
      "previous_hash": "00007957030d4e8b2432e5037939f03d9b16d8612364ce8c6cefd2b3e2a0d5f6",
      "timestamp": "2026-01-22 12:06:09.082474"
    }
  ],
  "length": 4
}
```

Conclusion : In this experiment, a basic blockchain was successfully implemented using Python. The blockchain demonstrates core concepts such as block structure, hashing, Proof-of-Work mining, and chain validation. This experiment provides a clear understanding of how blockchain works in a decentralized and secure manner.