Experiment 2

Aim: Create a Blockchain using Python

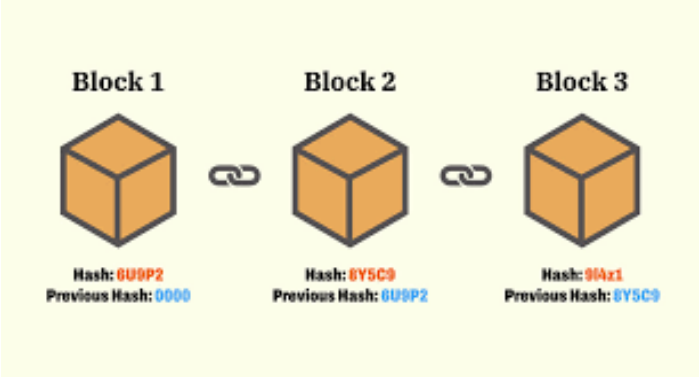
Theory:

1. **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.



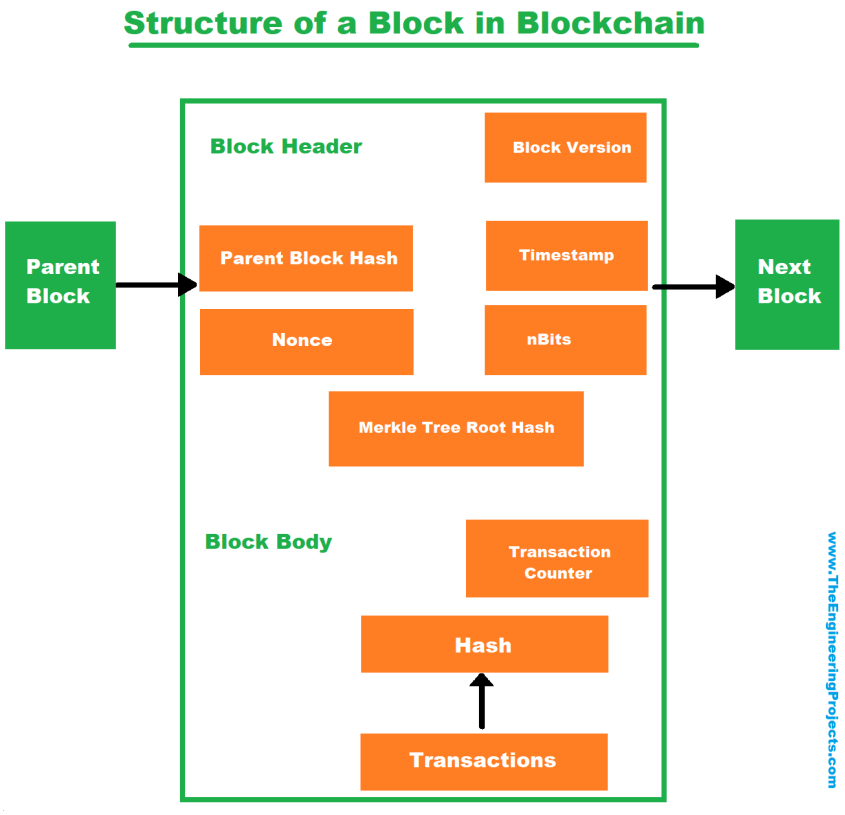
2. **What is a block?**

A block in blockchain is a digital container that securely stores validated transaction data, bundled together with a timestamp and a unique hash linking it to the previous block, forming an unchangeable chain (blockchain) of records. Think of it as a page in a public digital ledger,

holding a batch of verified transactions that, once added, cannot be altered without invalidating subsequent blocks.

Key components of a block:

- **Transactions**: A list of confirmed operations (e.g., payments, data entries).
- **Timestamp**: Records the exact time the block was created.
- **Hash**: A unique digital fingerprint of the block's data, generated by a cryptographic function.
- **Previous Hash:** The hash of the block that came before it, creating the chain link.



3. **Process of mining**

Mining in blockchain is the process of validating, securing, and adding new transaction blocks to a decentralized ledger (blockchain) using Proof-of-Work (PoW). Miners use specialized hardware to solve complex cryptographic puzzles to validate transactions and earn cryptocurrency rewards.

The Process of Mining (Step-by-Step):

- **Transaction Broadcast:** Users initiate transactions, which are broadcasted to the decentralized network.
- **Transaction Validation:** Miners, acting as network nodes, collect, verify, and group these transactions into a candidate block.

- **Proof-of-Work (Puzzle Solving):** Miners race to solve a complex mathematical puzzle (hashing) by finding a valid "nonce." This requires immense computational power.
- **Block Addition:** The first miner to solve the puzzle broadcasts the solution to the network.
- **Verification**: Other nodes verify the solution. Once confirmed, the block is permanently added to the blockchain.
- **Reward**: The successful miner is rewarded with newly created cryptocurrency and transaction fees.

Key Aspects of Mining:

- **Purpose:** To prevent double-spending, secure the network, and ensure decentralization.
- **Difficulty Adjustment**: The network automatically adjusts the puzzle's difficulty to maintain a steady block creation rate (e.g., ~10 minutes for Bitcoin).
- **Mining Pools:** Miners combine their computational power to increase the probability of solving the puzzle, sharing the rewards.

4. **How to check validity of block in blockchain**

To check the validity of a block in a blockchain, a node (computer) in the network performs a series of checks based on the network's specific rules and consensus mechanism. The two most important checks are the cryptographic integrity of the block and the validity of all transactions within it.

Key Verification Steps:

The validation process ensures that the block adheres to all the rules of the blockchain protocol:

- **Cryptographic Linkage Check**: A node verifies that the current block's header contains the correct hash of the previous (parent) block. It then recalculates the current block's hash using all the data within the proposed block (including the previous hash, timestamp, transaction data, etc.) and ensures that the recalculated hash matches the hash recorded in the current block's header. If the data in any previous block was altered, the subsequent hashes would not match, invalidating the entire chain from that point forward.
- **Proof of Work/Stake Validation:** The node verifies that the block meets the criteria set by the network's consensus mechanism:
    1. In a Proof of Work (PoW) system (like Bitcoin), the node checks that the block's hash is below the current network difficulty target, proving that sufficient computational effort was expended to mine it.
    2. In a Proof of Stake (PoS) system, the node verifies that the block was proposed by a valid validator who was selected according to the staking rules and that other validators have attested to its validity.

- **Transaction Validity:** Each and every transaction within the block is individually checked to ensure it complies with network rules.
- **Merkle Root Verification:** The node uses the Merkle root (a cryptographic summary of all transactions in the block) to efficiently and securely verify the integrity and inclusion of all transactions without needing to check every single transaction individually against the entire history.

Code:

1. Importing required libraries

    - datetime → to timestamp each block
    - hashlib → to generate SHA-256 hashes
    - json → to encode blocks as JSON for hashing
    - Flask → to run the blockchain API server

2. Code:

```
import datetime      # To generate timestamps for blocks
import hashlib       # To generate SHA256 hashes
import json          # To convert block data into JSON
from flask import Flask, jsonify    # To create API endpoints

# ------------------------------------------------------------
# Part 1 - Building the Blockchain
# ------------------------------------------------------------

class Blockchain:

    def __init__(self):
        # This list will store the entire chain of blocks
        self.chain = []

        # Create the genesis block (first block)
        # proof = 1 → arbitrary
        # previous_hash = '0' → since no previous block exists
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        """
        Creates a new block and adds it to the chain.
        Each block contains:
        - index
        - timestamp
```

```
    - proof (PoW output)
    - previous block hash
    """
    block = {
        'index': len(self.chain) + 1,              # Position of block in chain
        'timestamp': str(datetime.datetime.now()), # Current timestamp
        'proof': proof,                            # PoW result
        'previous_hash': previous_hash             # Hash of the previous block
    }

    # Add block to the chain
    self.chain.append(block)
    return block

def get_previous_block(self):
    """
    Returns the last block in the chain.
    """
    return self.chain[-1]

def proof_of_work(self, previous_proof):
    """
    Simple Proof of Work (PoW) algorithm:
    - Find a number (new_proof)
    - Such that the SHA256 hash of (new_proof^2 - previous_proof^2)
      starts with '0000'

    This is a computational puzzle to secure the network.
    """
    new_proof = 1
    check_proof = False

    while check_proof is False:
        # Cryptographic puzzle: hash difference of squares
        hash_operation = hashlib.sha256(
            str(new_proof**2 - previous_proof**2).encode()
        ).hexdigest()

        # Check if hash begins with 4 leading zeros
        if hash_operation[:4] == '0000':
            check_proof = True
        else:
            new_proof += 1
```

```
    return new_proof

def hash(self, block):
    """
    Creates a SHA256 hash of a block.
    - Sort keys to ensure consistent hashing
    """
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()

def is_chain_valid(self, chain):
    """
    Validates the blockchain by checking:
    1. The previous_hash field matches the actual hash of the previous block.
    2. The PoW condition (hash starting with '0000') is satisfied for each block.
    """
    previous_block = chain[0]   # Genesis block
    block_index = 1             # Start checking from block 2

    while block_index < len(chain):
        block = chain[block_index]

        # Check previous hash correctness
        if block['previous_hash'] != self.hash(previous_block):
            return False

        # Validate Proof of Work
        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

        # Move to next block
        previous_block = block
        block_index += 1

    return True
```

```
# ------------------------------------------------------------
# Part 2 - Mining our Blockchain (Flask API)
# ------------------------------------------------------------

# Initialize Flask web application
app = Flask(__name__)

@app.route("/")
def home():
    return "Flask app is working!"


# Create an instance of the Blockchain
blockchain = Blockchain()

# ------------------------------------------------------------
# API Endpoint: Mine a new block
# ------------------------------------------------------------
@app.route('/mine_block', methods=['GET'])
def mine_block():
    # Get the previous block
    previous_block = blockchain.get_previous_block()

    # Extract previous proof
    previous_proof = previous_block['proof']

    # Run Proof of Work algorithm
    proof = blockchain.proof_of_work(previous_proof)

    # Get hash of previous block
    previous_hash = blockchain.hash(previous_block)

    # Create the new block
    block = blockchain.create_block(proof, previous_hash)

    # Prepare response
    response = {
        'message': 'Congratulations, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }
    return jsonify(response), 200
```

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

# ------------------------------------------------------------
# API Endpoint: Check if 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

# ------------------------------------------------------------
# Run the Flask app
# ------------------------------------------------------------
app.run(host='0.0.0.0', port=5000)
```

Output:

```
(venv) PS C:\Users\siddi\Desktop\blockchain Lab\lab2> flask run
 * Ignoring a call to 'app.run()' that would block the current 'flask' CLI command.
   Only call 'app.run()' in an 'if __name__ == "__main__"' guard.
 * 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 http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [30/Jan/2026 12:29:10] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:29:12] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:29:12] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:29:15] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 12:29:19] "GET /is_valid HTTP/1.1" 200 -
```

1. **API */mine_block***


```
{
  "index": 2,
  "message": "Mahvish, you just mined a block!",
  "previous_hash": "651ed70718ba2119a874c7a5070452d67e13cb71fb2f0acaabb605f8dff564cf",
  "proof": 533,
  "timestamp": "2026-01-30 12:38:35.498865"
}
```


2. **API */get_chain***


```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 1,
      "timestamp": "2026-01-30 11:55:06.522143"
    },
    {
      "index": 2,
      "previous_hash": "652e4c1da98ce465a1f6df10bab2e07efe4d141a208faa34220a5cc0d5fa2698",
      "proof": 533,
      "timestamp": "2026-01-30 11:55:48.102279"
    },
    {
      "index": 3,
      "previous_hash": "c28963df2a634f62ffdfa12db64c6f57edaec8d2251f24d5f5fd5200c878eb1f",
      "proof": 45293,
      "timestamp": "2026-01-30 12:24:43.884370"
    }
  ],
  "length": 3
}
```


3. **API */is_valid***


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


Conclusion:

In this experiment, a simple blockchain was implemented to understand the fundamental working of blockchain technology. Blocks were created and securely linked using cryptographic hashing and a basic Proof of Work mechanism. The integrity of the blockchain was ensured through validation checks at each block.