

Name: Pranav Pol

Class : D20A

Roll no: 44

Batch: C

EXPERIMENT 2

AIM: Create a Blockchain using Python.

THEORY:

1. What is Blockchain?

Blockchain is a **decentralized, distributed digital ledger** that securely records transactions across multiple computers (nodes). Transactions are grouped into blocks, and each block is linked to the previous one using **cryptographic hashes**, forming a continuous chain.

Once data is recorded and validated, it becomes **immutable**, meaning it is extremely difficult to alter or delete.

Key Features of Blockchain:

- **Decentralization:** No single authority controls the blockchain; it is maintained by a network of nodes.
- **Transparency:** Transactions are visible to all authorized participants.
- **Immutability:** Any attempt to modify data changes the block's hash and breaks the chain.
- **Security:** Transactions are verified by majority consensus.
- **Distributed Ledger:** Every node maintains a copy of the blockchain.

2. What is a Block in Blockchain?

A **block** is a fundamental unit of a blockchain, similar to a link in a chain. It stores a group of verified transactions along with metadata and is securely connected to the previous block using cryptographic hashing.

Blocks help manage and track the massive number of transactions occurring globally by organizing them into a secure and structured format.

What are the Components of a Block

A block consists of the following components:

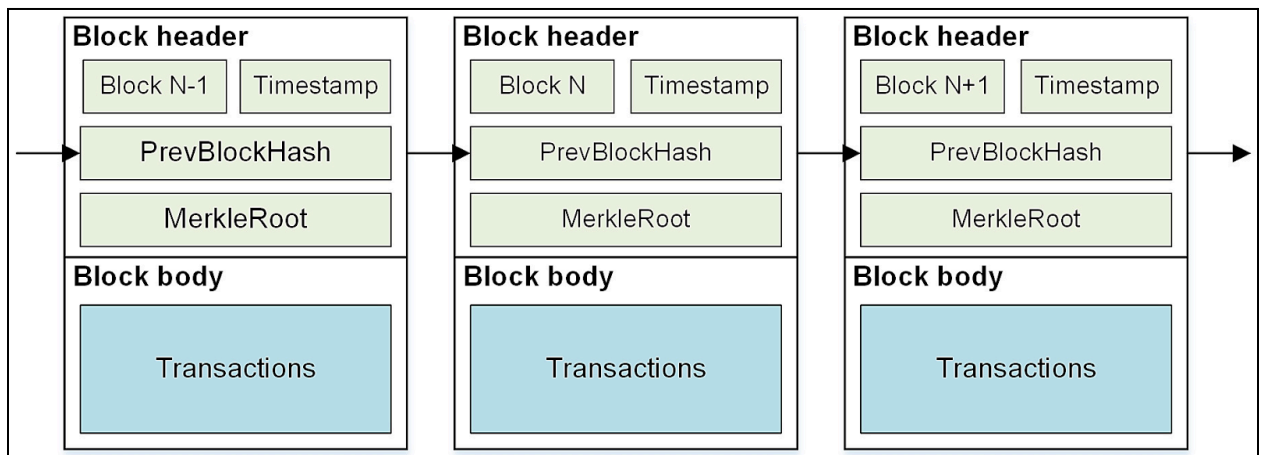
1. Block Header

Identifies the block within the blockchain. It contains essential metadata and is repeatedly hashed by miners during mining by changing the nonce value.

2. Previous Block Hash

A reference to the hash of the previous block, linking blocks together and ensuring chain integrity.

3. **Timestamp**
Records the exact date and time when the block was created, ensuring chronological order.
4. **Nonce**
A number used only once. Miners modify the nonce during Proof-of-Work to generate a valid hash that meets the difficulty target.
5. **Merkle Root**
A cryptographic hash representing all transactions in the block. It allows efficient and secure verification of transactions.



Example of a Block Structure

Block Number: 105678

Block Header:

- Previous Block Hash: 000000000000000000a3f2c9e7b1d45c8fa9a1e7b6d9c2f0a4e3b1c8d9f
- Merkle Root: 4f3c2a1b9e8d7c6b5a4f3e2d1c9b8a7f6e5d4c3b2a1f9e8d7c6b5a4
- Timestamp: 2026-01-29 14:32:10 UTC
- Nonce: 845932
- Block Hash: 00000000000000000005d6e8f3c2a9b1e7d4f6c8a2e9b5d3f1c7...

Transactions:

1. Alice → Bob : 0.5 BTC
2. Charlie → Dave : 1.2 BTC
3. Eve → Frank : 0.3 BTC

3. What is the Process of Mining in Blockchain?

Mining is the process of **adding a new block to the blockchain** using the Proof-of-Work (PoW) mechanism.

Steps in Mining:

Step 1: Collect Transactions

- Pending transactions are gathered into a block.
- Example: Alice sends 50 coins to Bob.

Step 2: Create Block Header

- Includes previous hash, transaction data, timestamp, and nonce.

Step 3: Proof of Work (PoW)

- Miners repeatedly change the nonce to generate a hash.
- The hash must satisfy the difficulty condition (e.g., leading zeroes).

SHA-256(block data + nonce) → 0000ab34cd...

Step 4: Validate and Broadcast

- The valid block is broadcast to the network.
- Other nodes verify the hash, nonce, and previous hash link.

Step 5: Add to Blockchain

- Once verified, the block is added permanently.
- The miner receives a cryptocurrency reward.

4. How is the Validity of a Block Checked in Blockchain

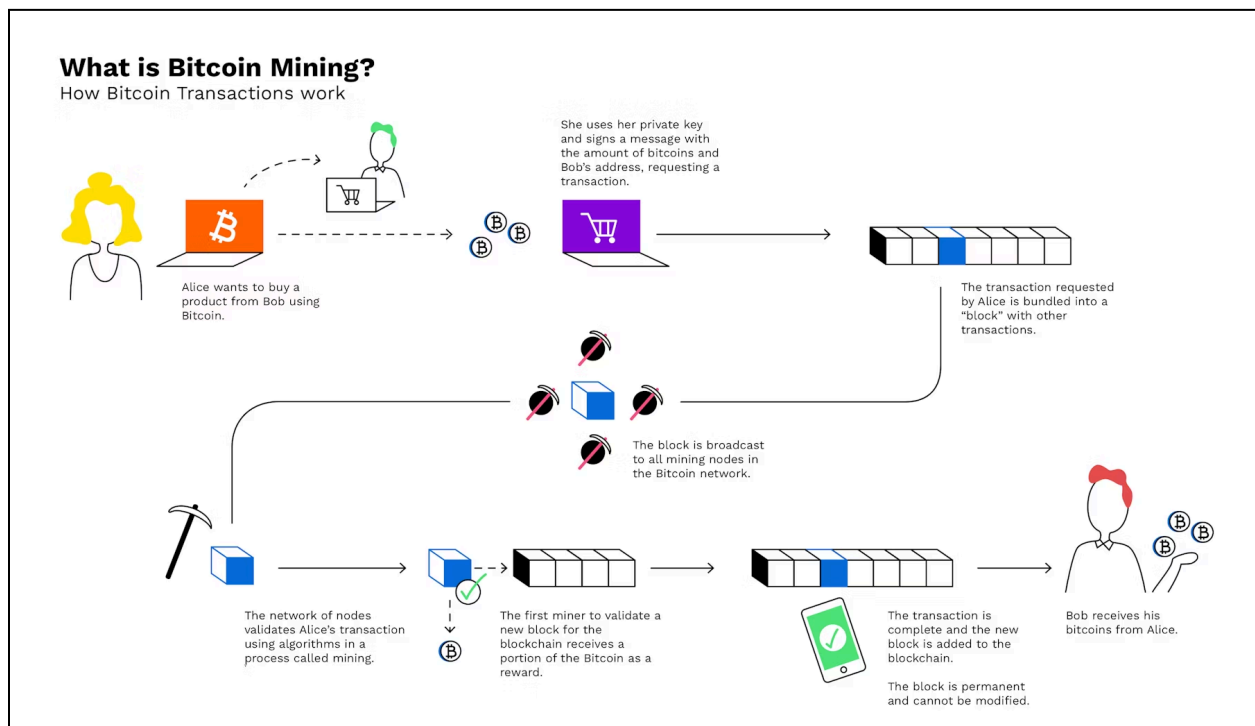
A block is validated through multiple verification steps:

1. **Verify Block Hash**
 - Recalculate the hash and compare it with the stored hash.
2. **Check Proof-of-Work**
 - Ensure the hash meets the difficulty requirement (leading zeroes).
3. **Validate Previous Block Hash**
 - Confirm the previous hash matches the last block's hash.
4. **Verify Transactions**
 - Check transaction authenticity, digital signatures, and prevent double spending.
5. **Verify Merkle Root**

- Recalculate and compare the Merkle root from transactions.
6. **Check Timestamp**
 - Timestamp must be valid and not in the future.
 7. **Check Block Size and Format**
 - Block must follow protocol-defined size and structure.
 8. **Validate Genesis Block**
 - The first block is hardcoded with previous hash = “0” and must remain unchanged.

Final Result:

- All checks pass → Block is valid
- Any check fails → Block is rejected



CODE:

```
import datetime
import hashlib
import json
from flask import Flask, jsonify
```

```
class Blockchain:
```

```
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(0, '0')
```

```
    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,                # Golden nonce
            'transactions': self.transactions,
            'previous_hash': previous_hash
        }
        self.transactions = []            # Clear transactions
        self.chain.append(block)
        return block
```

```
    def get_previous_block(self):
        return self.chain[-1]
```

```
    # Transactions created internally
```

```
    def create_transaction(self):
        self.transactions.append({
            'sender': 'SYSTEM',
            'receiver': 'MINER',
            'amount': 10
        })
```

```
    # Proof of Work: hash must start with "000"
```

```
    def proof_of_work(self, previous_hash, transactions):
        nonce = 0
        while True:
            data = {
                'nonce': nonce,
                'transactions': transactions,
                'previous_hash': previous_hash
            }
            hash_val = hashlib.sha256(
                json.dumps(data, sort_keys=True).encode()
            ).hexdigest()
            if hash_val[:3] == '000':
```

```

        return nonce
        nonce += 1

def hash(self, block):
    return hashlib.sha256(
        json.dumps(block, sort_keys=True).encode()
    ).hexdigest()

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

        if curr['previous_hash'] != self.hash(prev):
            return False

    data = {
        'nonce': curr['proof'],
        'transactions': curr['transactions'],
        'previous_hash': curr['previous_hash']
    }
    hash_val = hashlib.sha256(
        json.dumps(data, sort_keys=True).encode()
    ).hexdigest()

    if hash_val[:3] != '000':
        return False

    return True

# =====
# Flask App
# =====

app = Flask(__name__)
blockchain = Blockchain()

@app.route('/')
def home():
    return "Blockchain API is running"

@app.route('/add_transaction', methods=['GET'])
def add_transaction():
    blockchain.create_transaction()

```

```

    return jsonify({'message': 'Transaction added internally'}), 200

@app.route('/mine_block', methods=['GET'])
def mine_block():
    if len(blockchain.transactions) == 0:
        return jsonify({'message': 'No transactions to mine'}), 400

    prev_block = blockchain.get_previous_block()
    prev_hash = blockchain.hash(prev_block)

    proof = blockchain.proof_of_work(prev_hash, blockchain.transactions)
    block = blockchain.create_block(proof, prev_hash)

    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({
        'valid': blockchain.is_chain_valid()
    }), 200

# =====
# Run App
# =====

if __name__ == '__main__':
    app.run(debug=True)

```

OUTPUT:

← ↻ ⓘ 127.0.0.1:5000/add_transaction

Pretty-print ☐

```
{
  "message": "Transaction added internally"
}
```

← ↻ ⓘ 127.0.0.1:5000/mine_block

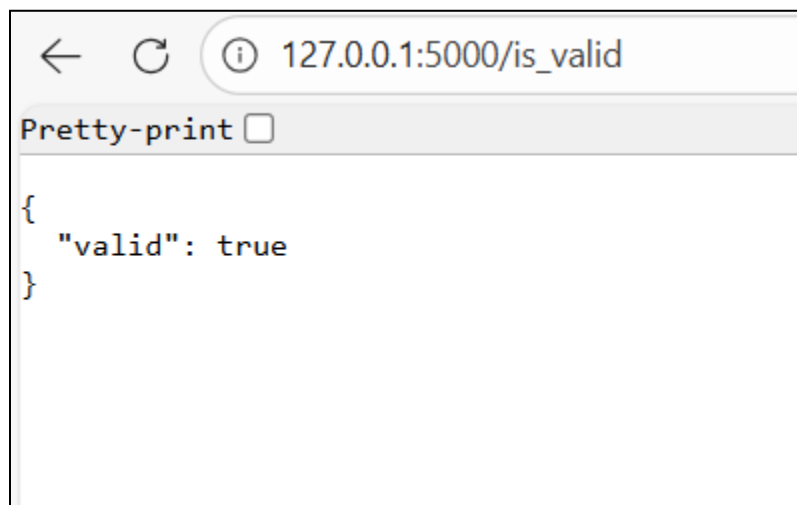
Pretty-print ☐

```
{
  "block": {
    "index": 2,
    "previous_hash": "207828c62cd49627f6edb41c7eebbb51914eb892cb0b2c5ba71296c54ad228aa",
    "proof": 14602,
    "timestamp": "2026-02-03 03:06:07.797595",
    "transactions": [
      {
        "amount": 10,
        "receiver": "MINER",
        "sender": "SYSTEM"
      }
    ]
  },
  "message": "Block mined successfully!"
}
```



```
← 127.0.0.1:5000/get_chain
Pretty-print ☐

{
  "chain": [
    {
      "index": 1,
      "previous_hash": "0",
      "proof": 0,
      "timestamp": "2026-02-03 03:04:38.829887",
      "transactions": []
    },
    {
      "index": 2,
      "previous_hash": "207828c62cd49627f6edb41c7eebbb51914eb892cb0b2c5ba71296c54ad228aa",
      "proof": 14602,
      "timestamp": "2026-02-03 03:06:07.797595",
      "transactions": [
        {
          "amount": 10,
          "receiver": "MINER",
          "sender": "SYSTEM"
        }
      ]
    }
  ],
  "length": 2
}
```



```
← 127.0.0.1:5000/is_valid
Pretty-print ☐

{
  "valid": true
}
```

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. By linking blocks through cryptographic hashes and verifying them using difficulty rules, the system ensures data integrity and immutability. This experiment provides a clear understanding of how blockchain works in a decentralized and secure manner.

