EXPERIMENT NO.2

Name: Vaishnal Mali
Class:D20A
Roll No:32
Batch:B

Aim: Create a Blockchain using Python

1. What is Blockchain

Blockchain is a revolutionary technology that functions as a shared, immutable digital ledger, often described as a decentralized database or distributed ledger technology. It records transactions across a network of computers in a way that ensures security, transparency, and resistance to tampering. The term "blockchain" derives from its structure: data is organized into blocks, each linked to the previous one through cryptographic hashes, forming a continuous chain. This decentralized nature eliminates the need for intermediaries, allowing multiple parties to perform transactions without third-party intervention.

Key characteristics include:

- **Decentralization**: No single entity controls the network; it's maintained by nodes (computers) that validate and record transactions.
- **Immutability**: Once data is added, it cannot be altered without consensus from the network, making it tamper-proof.
- **Transparency**: All participants can view the ledger, but privacy is maintained through cryptography.
- **Security**: Uses cryptographic techniques to protect data.

Blockchain stores various types of information, but it's most commonly used for transactions in cryptocurrencies like Bitcoin. It operates on principles such as consensus mechanisms to agree on the state of the ledger. Unlike traditional databases that structure data in tables, blockchain uses blocks chained together, ensuring that changes to one block would require altering all subsequent blocks, which is computationally infeasible

2. What is a Block?

A block is the fundamental building block of a blockchain, acting as a container for data such as transactions. It serves as a record in the digital ledger, similar to a page in a book, where each block holds a complete record of transactions or other information. Blocks

are linked together in a chain, with each new block referencing the previous one via a cryptographic hash, creating an unbreakable sequence.

In essence, a block is a node in the blockchain network that stores encrypted data, including a list of transactions, a timestamp, and a unique identifier (hash). The first block in any blockchain is called the Genesis Block, which has no previous block reference and is hardcoded into the protocol. This structure ensures the integrity of the entire chain, as altering any block would invalidate the hashes of all following blocks

3. Components of a Block

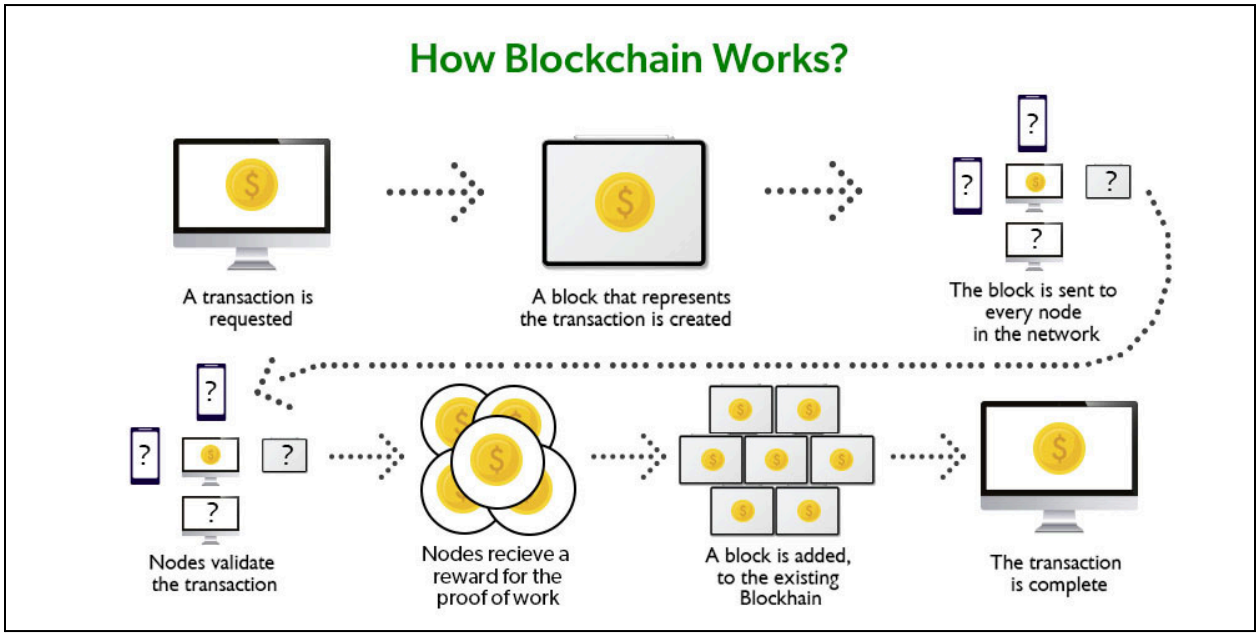
A block in blockchain consists of two main parts: the block header and the block body (containing transactions). The header is crucial for identification and linking, while the body holds the actual data.

Key components include:

Component	Description
Block Header	Identifies the block and includes metadata like version, timestamp, Merkle root, difficulty target, nonce, and previous block hash.
Version	A 4-byte field tracking software or protocol upgrades.
Previous Block Hash	A 32-byte reference to the hash of the prior block, ensuring chain continuity.
Merkle Root	A 32-byte hash of all transactions in the block, derived from a Merkle tree for efficient verification.
Timestamp	A 4-byte record of the block's approximate creation time.

Difficulty Target	A 4-byte value setting the mining difficulty.
Nonce	A 4-byte number used in proof-of-work mining to find a valid hash.
Transactions	The list of verified transactions stored in the block body.

These elements work together to maintain security and enable validation. The header alone is often sufficient for light clients to verify blocks without the full data



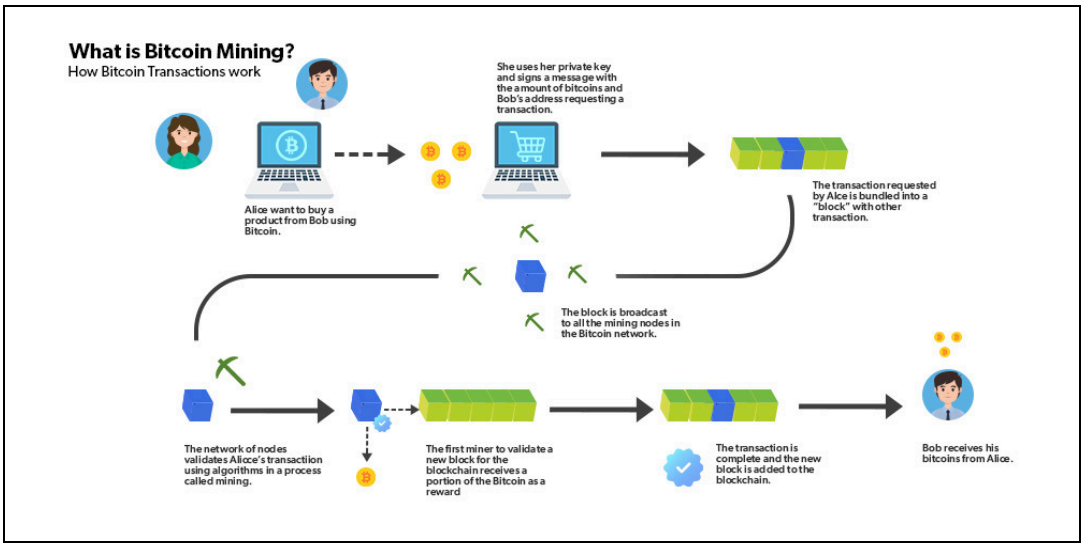
4. Process of mining

Mining is the process of validating transactions and adding new blocks to the blockchain. It is primarily associated with **Proof-of-Work (PoW)** consensus mechanisms (e.g., Bitcoin). Miners compete to solve a complex cryptographic puzzle, and the winner gets to add the block and receive a reward.

Key steps in the mining process:

1. **Transaction Collection** Unconfirmed transactions are collected from the network and stored in a **mempool** (memory pool). Miners select transactions (usually prioritizing those with higher fees) to include in a candidate block. A special **coinbase transaction** is added, which rewards the miner with new coins + transaction fees.
2. **Block Header Preparation** The miner constructs the block header, which includes:
    - Previous block hash
    - Merkle root (summary hash of all transactions)
    - Timestamp
    - Difficulty target
    - Nonce (a variable number starting from 0)
3. **Solving the Cryptographic Puzzle (Proof-of-Work)** Miners repeatedly hash the block header (using SHA-256 in Bitcoin) while changing the nonce value. The goal is to find a hash that is **less than or equal to the current difficulty target** (a very small number, often requiring the hash to start with many leading zeros). This is computationally intensive and probabilistic — miners try billions/trillions of nonces per second.
4. **Block Found** The first miner to find a valid nonce (producing a qualifying hash) has "solved" the puzzle. They broadcast the completed block to the network.
5. **Network Acceptance & Reward** Other nodes verify the block. If valid, it is added to their copy of the blockchain. The successful miner receives the **block reward** (e.g., currently 3.125 BTC as of recent halvings) + transaction fees. Difficulty adjusts periodically to keep average block time stable (~10 minutes in Bitcoin).

Mining secures the network by making it extremely expensive to alter past blocks (would require re-mining all subsequent blocks) and prevents double-spending.



5. Checking the validity of blocks in blockchain

Once a mined block is broadcast, every full node independently verifies it before accepting it into their blockchain copy. This decentralized validation is what makes blockchain trustless and secure.

Key validity checks:

1. **Block Structure & Syntax**
    - Correct format and size
    - Valid version number
    - Timestamp is reasonable (not too far in future or before previous block)
2. **Previous Block Hash Match** The "previous block hash" field must exactly match the hash of the most recent block already in the node's chain.
3. **Proof-of-Work Validation** Recompute the block hash → check if it is ≤ current difficulty target (i.e., has enough leading zeros). This confirms the miner did the required work.
4. **Merkle Root Verification** Rebuild the Merkle tree from all transactions in the block body → verify the computed Merkle root matches the one in the block header.
5. **Transaction Validity**
    - Each transaction has valid signatures
    - Inputs are unspent (no double-spending)
    - `Input amounts ≥ output amounts + fees`

- Transactions follow protocol rules

6. **Consensus Rules Compliance** Block follows all network rules (max block size, valid coinbase, etc.).

If **any** check fails, the block is rejected and not added to the chain. Nodes continue waiting for a valid block. The longest valid chain (most cumulative work) is considered the true chain.

Light clients (SPV wallets) use simplified checks — they rely on **Merkle proofs** to confirm a transaction exists in a block without downloading everything.

Implementation:-
Installing Flask

```
PS C:\Users\Student\Downloads\BCEx2> pip install flask
Collecting flask
  Downloading flask-3.1.2-py3-none-any.whl.metadata (3.2 kB)
Collecting blinker>=1.9.0 (from flask)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.3.1-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2.0 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting markupsafe>=2.1.1 (from flask)
```

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):
```

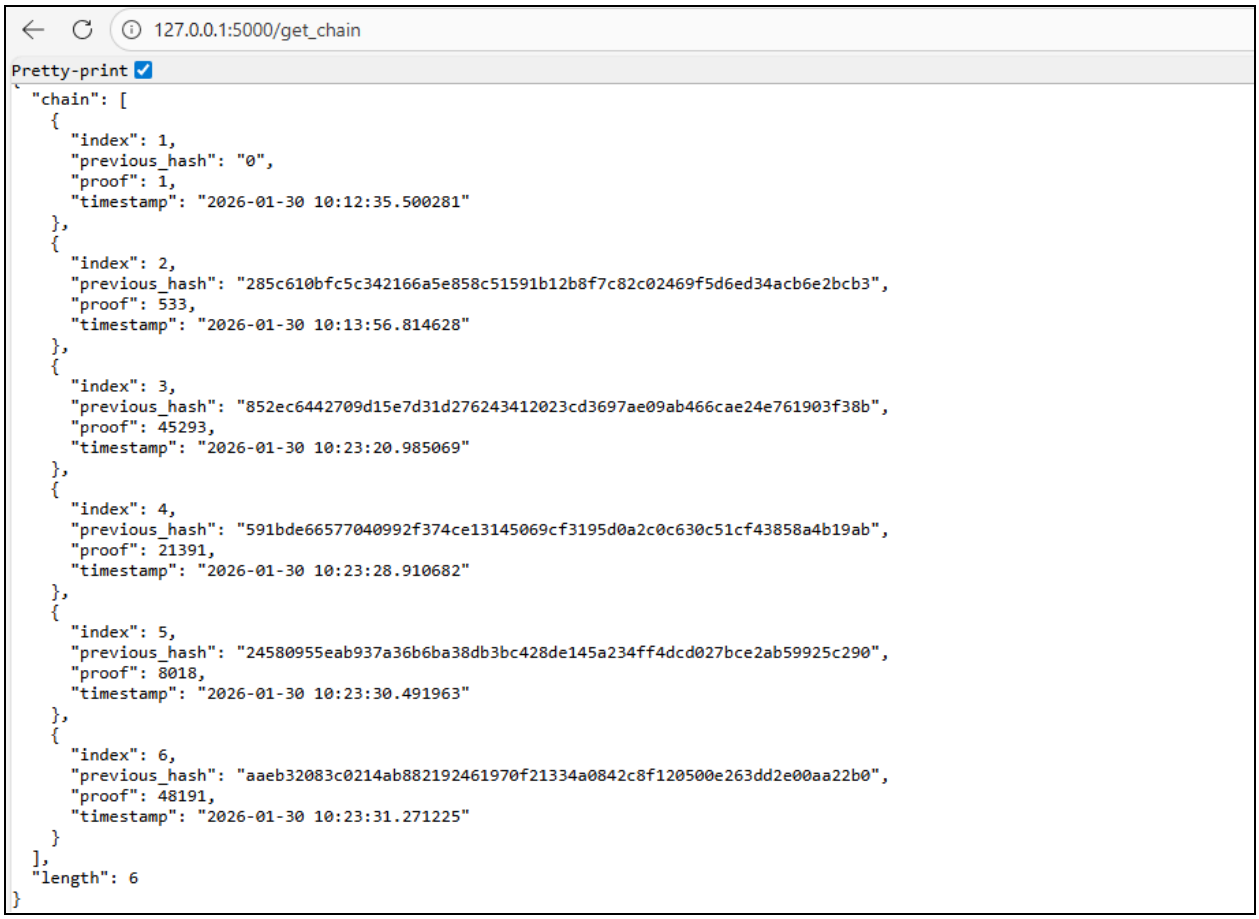
```
    genesis_block = {
        'index': 1,
        'timestamp': str(datetime.datetime.now()),
        'data': 'Genesis Block',
        'nonce': 0,
        'previous_hash': '0'
    }
    genesis_block['hash'] = self.calculate_hash(genesis_block)
    self.chain.append(genesis_block)

def calculate_hash(self, block):
    block_copy = block.copy()
    block_copy.pop('hash', None)  # Remove hash before recalculating

    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)

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

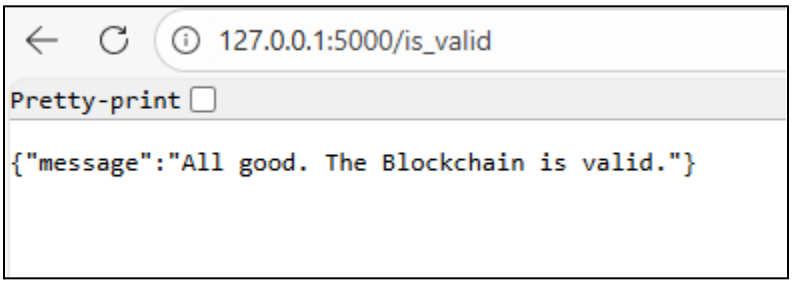
```
            nonce += 1

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

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

            # Recalculate and verify hash
            recalculated_hash = self.calculate_hash(current)
            if recalculated_hash != current['hash']:
                return False

            # Check difficulty rule
            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)
```





Conclusion:-

In this experiment, a simple blockchain was successfully built using Python. It clearly illustrated the essential concepts of blockchain, such as block structure, cryptographic hashing, Proof-of-Work mining, and chain validation.Through hash-linked blocks and strict difficulty-based verification rules, the system guarantees data integrity, immutability, and strong resistance to tampering. Even a minor change in any block breaks the entire chain, perfectly showcasing the decentralized and secure foundation of blockchain technology.