Aryan Dangat D20A-19

Experiment No. 2

Aim:- Create blockchain using python

What is Blockchain?

Blockchain is a modern digital technology used to store information in a **secure, transparent, and tamper-proof way**. It works like a **shared online record book (ledger)** that keeps track of transactions.

The term **blockchain** comes from how the data is stored:

- Information is saved inside **blocks**
- Each block is connected to the **previous block**
- Together they form a **chain of blocks**

Every block contains:

- Transaction details
- Time of creation (timestamp)
- A special security code called a **cryptographic hash**

This hash links one block to another. If someone tries to change the data, the hash changes and the chain breaks, making tampering easy to detect.

Key Features of Blockchain

- Distributed across many computers (nodes)
- No single authority controls it (decentralized)
- Transactions are verified by the majority of nodes
- Once data is added, it cannot be changed (immutable)
- Highly secure and transparent

In simple words, blockchain is a **decentralized database that safely records transactions across multiple systems**

What is a Block?

A **block** is a basic unit of storage in a blockchain.
Think of it like **one page in a ledger book** that stores transaction records.

Many transactions happen daily, and blocks help organize and store these transactions securely. Once a block is full, it gets added to the blockchain and linked with other blocks.

Components of a Block

Block Header

The block header contains important information that uniquely identifies the block. It helps maintain the order and connection between blocks.

Previous Block Hash

This stores the hash of the previous block.
It links the current block to the earlier one, forming the **chain structure** of the blockchain.

Timestamp

Records the exact **date and time** when the block was created.
It proves when the transactions were added.

Nonce

Nonce means **"number used only once."**

- Used during mining (Proof of Work)
- Miners keep changing the nonce to find a valid hash
- Helps secure the block and validate it

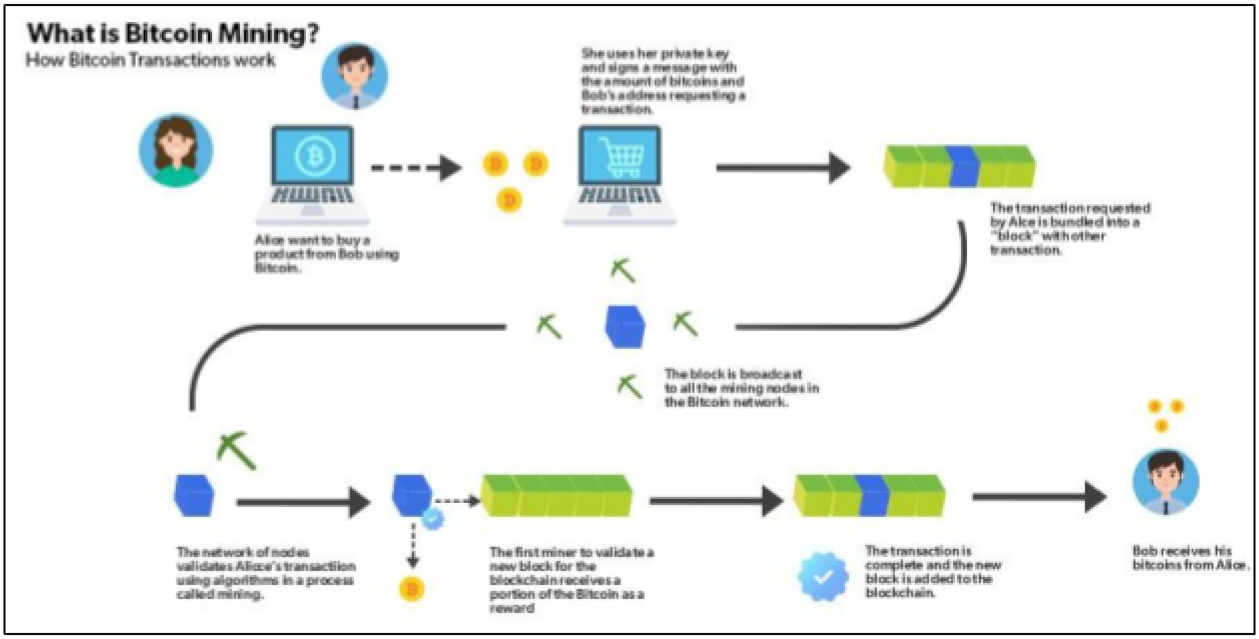
Merkle Root

The Merkle Root is created using a **Merkle Tree**.

- Combines all transactions into one hash
- Acts like a digital fingerprint
- Allows quick and secure verification of transactions

```
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 Alce 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:
    1. Hash correctness
    2. Nonce validity
    3. 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).

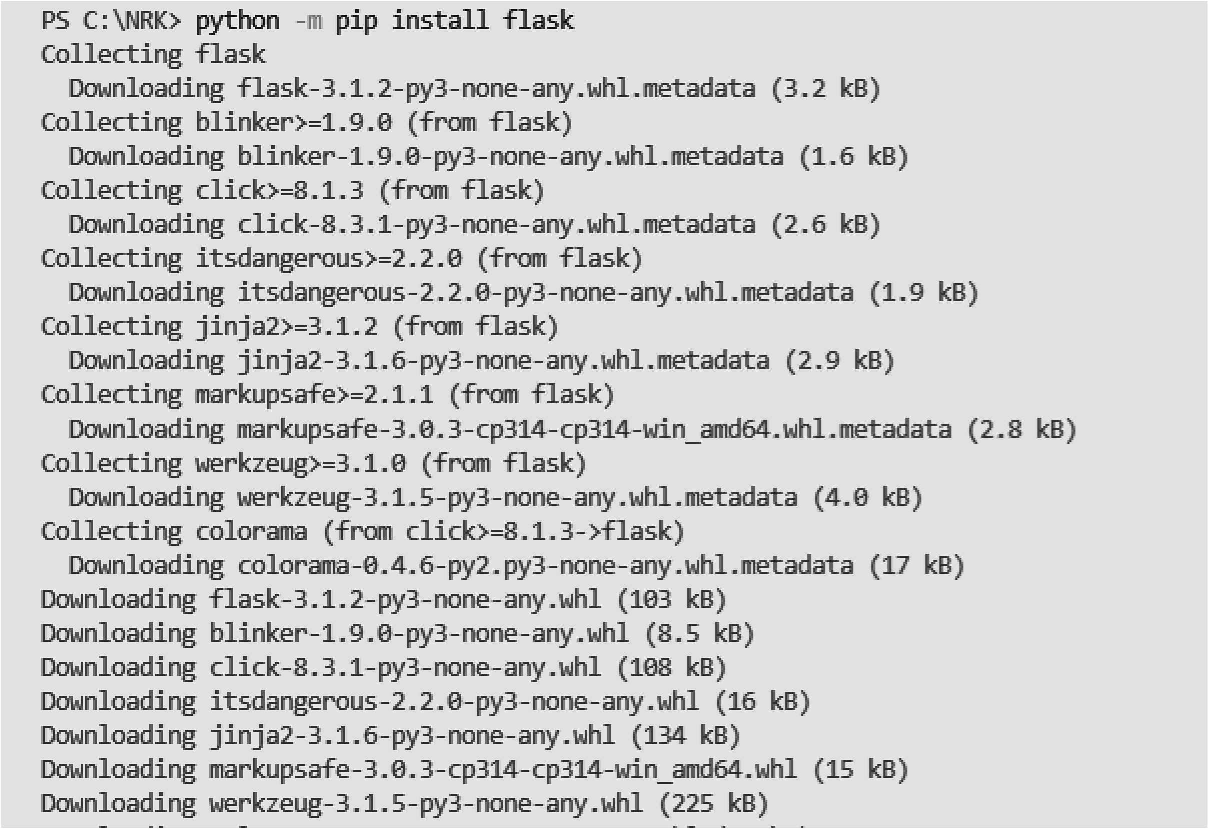
```
if not Block[n].hash.startswith("0000"):
  Invalid
```

Step 4: Validate Entire Chain

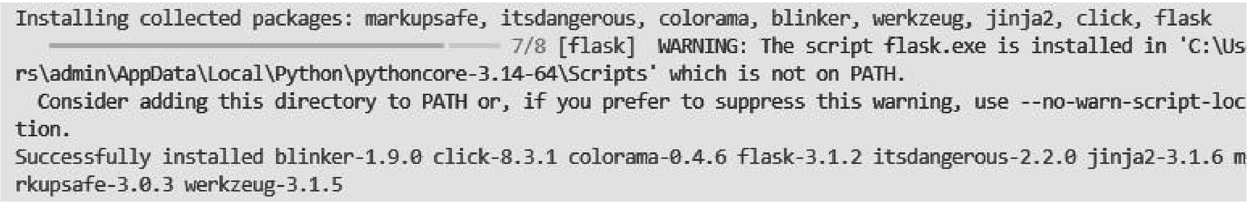
- Repeat steps 1–3 for all blocks from Genesis to the latest block.
- If any block fails → blockchain is invalid.

Step 5: Validate Genesis Block

- The first block (Genesis block) is hardcoded.
- Previous hash = "0"
- Must not be altered.



```
PS C:\NRK> python -m 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)
  Downloading markupsafe-3.0.3-cp314-cp314-win_amd64.whl.metadata (2.8 kB)
Collecting werkzeug>=3.1.0 (from flask)
  Downloading werkzeug-3.1.5-py3-none-any.whl.metadata (4.0 kB)
Collecting colorama (from click>=8.1.3->flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Downloading flask-3.1.2-py3-none-any.whl (103 kB)
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading click-8.3.1-py3-none-any.whl (108 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
Downloading markupsafe-3.0.3-cp314-cp314-win_amd64.whl (15 kB)
Downloading werkzeug-3.1.5-py3-none-any.whl (225 kB)
```



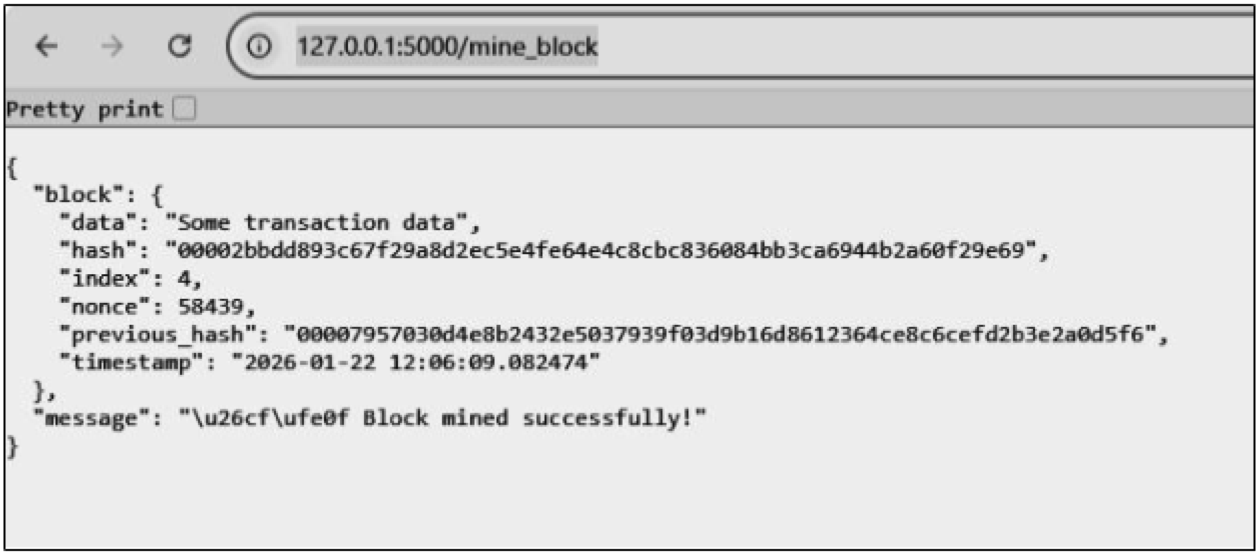
```
Installing collected packages: markupsafe, itsdangerous, colorama, blinker, werkzeug, jinja2, click, flask
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╺━━━ 7/8 [flask]  WARNING: The script flask.exe is installed in 'C:\Us
rs\admin\AppData\Local\Python\pythoncore-3.14-64\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-loc
tion.
Successfully installed blinker-1.9.0 click-8.3.1 colorama-0.4.6 flask-3.1.2 itsdangerous-2.2.0 jinja2-3.1.6 m
rkupsafe-3.0.3 werkzeug-3.1.5
```

blockchain.py

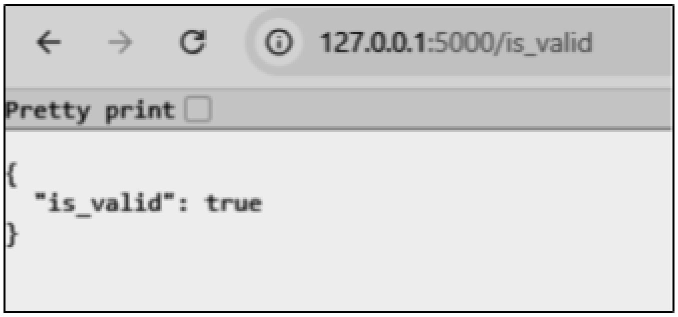
```
import datetime
import hashlib
import json


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

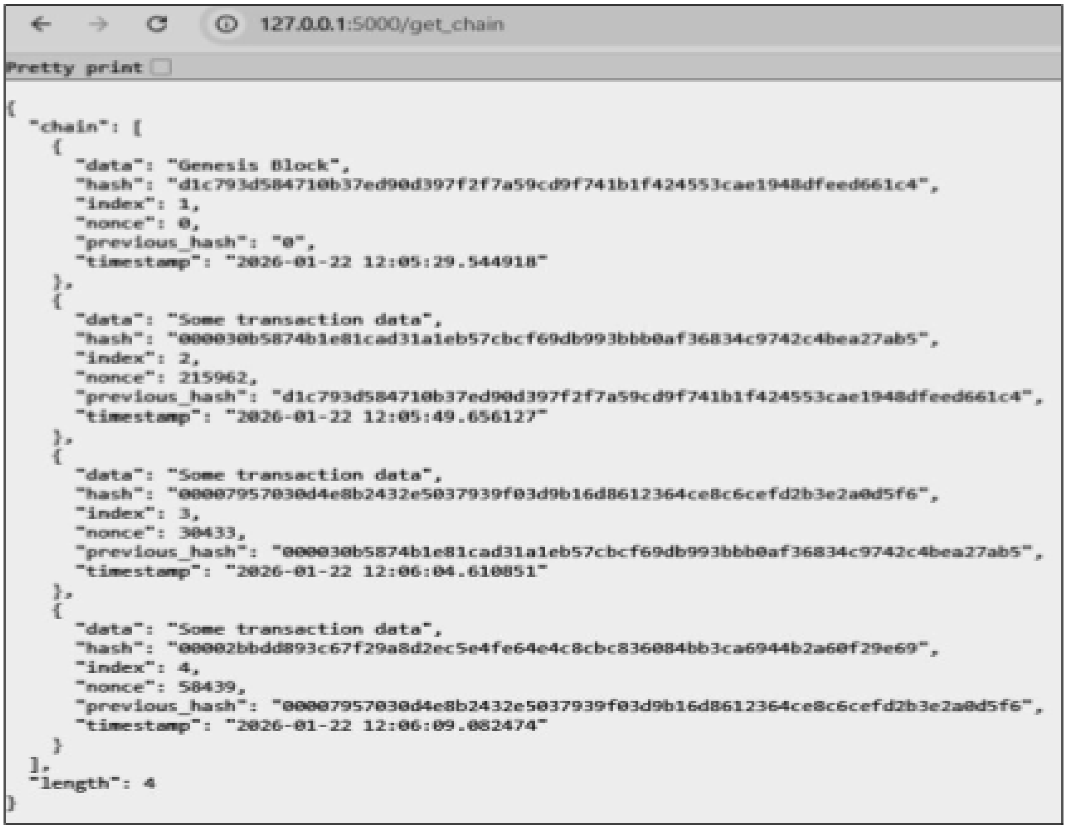
```
from flask import Flask, jsonify
from blockchain import Blockchain

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()
    }), 20

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



```
{
  "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
}
```