`Pranav Titambe, D20A, 60`

# Blockchain Lab

**Aim** - Write a Python program to understand SHA and Cryptography in Blockchain, Merkle root tree hash.

**Theory -**

**What is Cryptography?**

Cryptography is the science of **protecting information** by converting it into an unreadable format so that only authorized users can access it.

It ensures:

- **Confidentiality** – Only intended users can read data
- **Integrity** – Data is not altered
- **Authentication** – Identity verification
- **Non-repudiation** – Sender cannot deny sending data

**Types of Cryptography**

1. **Symmetric Key Cryptography**
    - Same key for encryption and decryption
    - Example: AES, DES

2. **Asymmetric Key Cryptography**
    - Uses public and private keys
    - Example: RSA, ECC

3. **Hash Functions**
    - One-way function
    - Produces fixed-length output
    - Example: SHA-256, MD5

**What is a Merkle Tree?**

A **Merkle Tree** is a **binary tree data structure** in which:

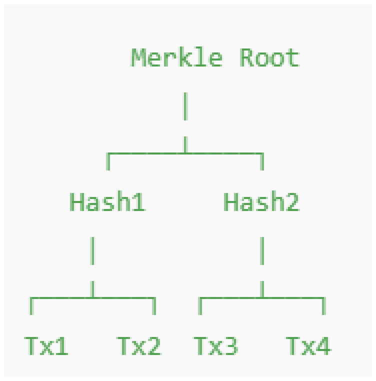
- Each **leaf node** contains a hash of transaction data
- Each **non-leaf node** contains a hash of its child nodes

- The **topmost hash** is called the **Merkle Root**

Merkle Trees are used to **verify data integrity efficiently**.

**Merkle Tree Structure**

- **Leaf Nodes:** Hash of individual transactions
- **Intermediate Nodes:** Hash of combined child hashes
- **Root Node:** Final hash (Merkle Root)



**What is Merkle Root?**

The **Merkle Root** is:

- A **single hash value**
- Represents **all transactions** in a block
- Stored in the **block header** in blockchain

**Importance**

- If **any transaction changes**, the Merkle Root changes
- Ensures **tamper detection**

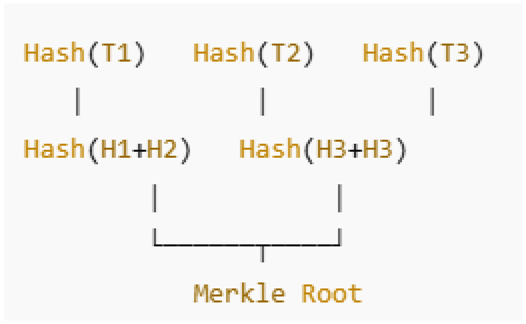
**Working of Merkle Tree (Step-by-Step)**

1. Transactions are collected
2. Each transaction is hashed
3. Hashes are paired and combined
4. Combined hashes are hashed again

5. If odd number of hashes → last hash is duplicated
6. Process continues until one hash remains
7. Final hash is the **Merkle Root**

**Example**

Transactions: T1, T2, T3



```
Hash(T1)   Hash(T2)   Hash(T3)
   |          |          |
Hash(H1+H2)   Hash(H3+H3)
     |            |
     └─────┬──────┘
           |
       Merkle Root
```

**Benefits of Merkle Tree**

1. **Efficient Verification**
   - Only small data needed to verify transactions
2. **Data Integrity**
   - Any data change alters the Merkle Root
3. **Reduced Storage**
   - No need to store full transaction list
4. **Fast Synchronization**
   - Useful in distributed systems
5. **Scalable**
   - Handles millions of transactions efficiently.

**Use of Merkle Tree in Blockchain**

Merkle Trees are used in blockchain to:

1. **Store transactions efficiently**
2. **Verify transactions quickly**
3. **Ensure block integrity**
4. **Enable light nodes (SPV)**
5. **Detect tampering**

```
Pranav Titambe, D20A, 60
```

**In Bitcoin**

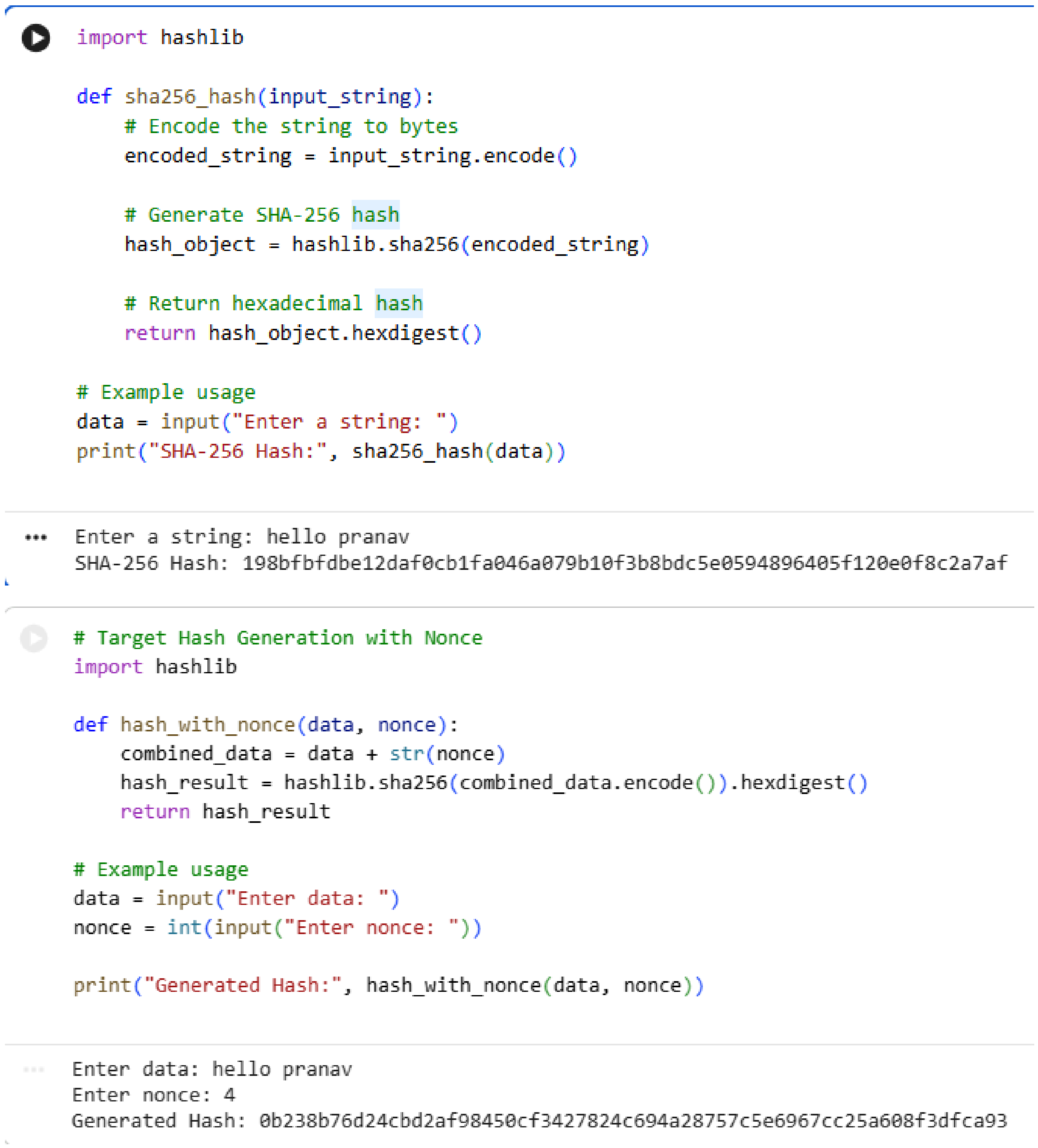
- Transactions → Merkle Tree
- Merkle Root stored in block header
- Any transaction modification invalidates the block

**Use Cases of Merkle Tree**

1. **Blockchain & Cryptocurrencies**
   - Bitcoin, Ethereum
2. **Distributed Systems**
   - Data synchronization
3. **Version Control Systems**
   - Git uses Merkle Trees
4. **Database Verification**
   - Checking data integrity
5. **Peer-to-Peer Networks**
   - Efficient data verification
6. **Cloud Storage**
   - Secure data validation

**Code & Output -**

Pranav Titambe, D20A, 60



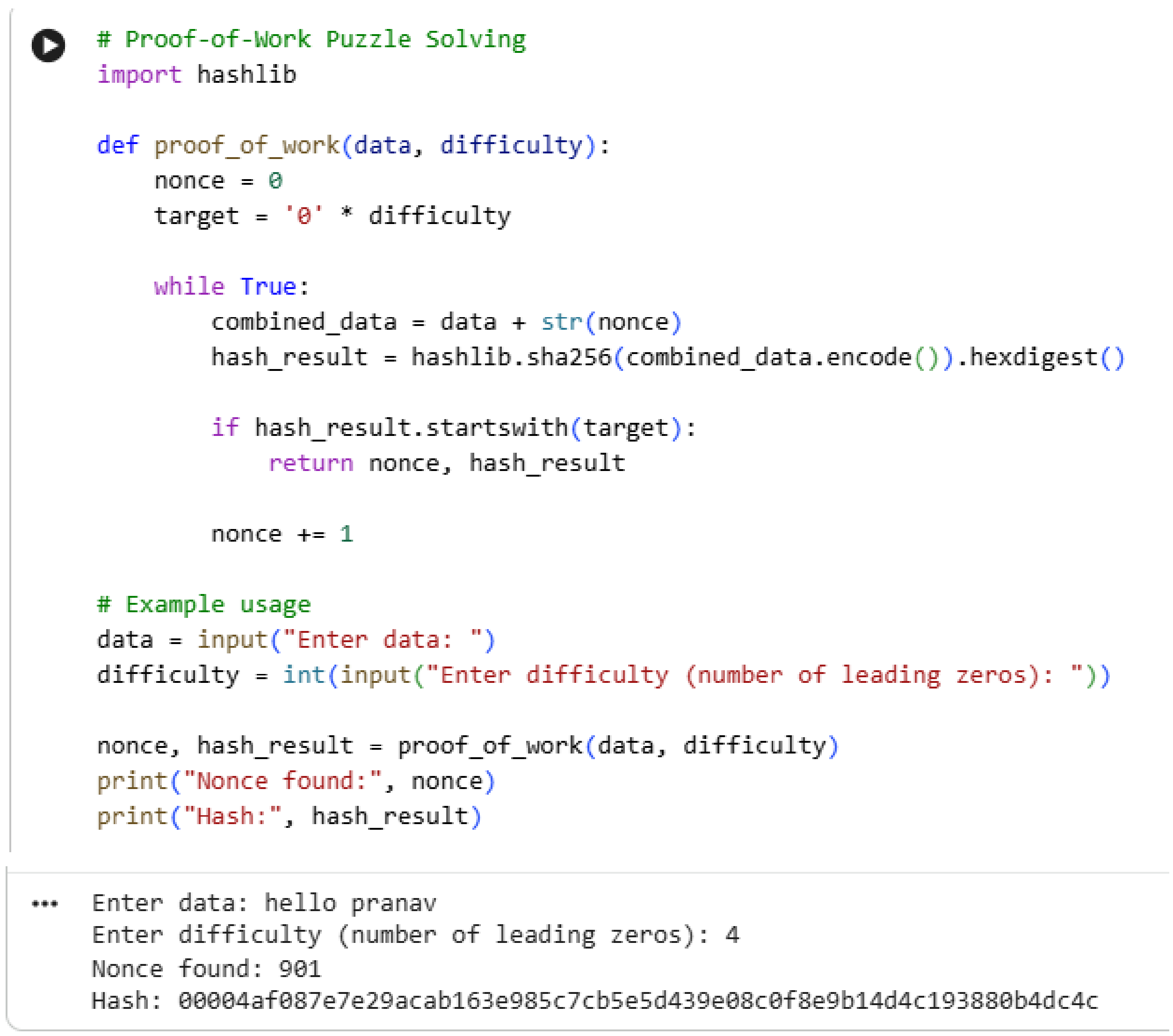
```
import hashlib

def sha256_hash(input_string):
    # Encode the string to bytes
    encoded_string = input_string.encode()

    # Generate SHA-256 hash
    hash_object = hashlib.sha256(encoded_string)

    # Return hexadecimal hash
    return hash_object.hexdigest()

# Example usage
data = input("Enter a string: ")
print("SHA-256 Hash:", sha256_hash(data))
```

```
Enter a string: hello pranav
SHA-256 Hash: 198bfbfdbe12daf0cb1fa046a079b10f3b8bdc5e0594896405f120e0f8c2a7af
```

```
# Target Hash Generation with Nonce
import hashlib

def hash_with_nonce(data, nonce):
    combined_data = data + str(nonce)
    hash_result = hashlib.sha256(combined_data.encode()).hexdigest()
    return hash_result

# Example usage
data = input("Enter data: ")
nonce = int(input("Enter nonce: "))

print("Generated Hash:", hash_with_nonce(data, nonce))
```

```
Enter data: hello pranav
Enter nonce: 4
Generated Hash: 0b238b76d24cbd2af98450cf3427824c694a28757c5e6967cc25a608f3dfca93
```

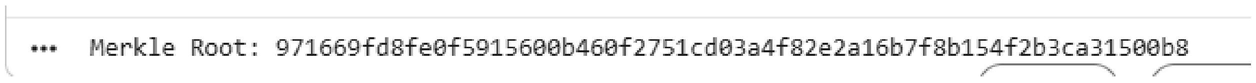
Pranav Titambe, D20A, 60



```python
# Proof-of-Work Puzzle Solving
import hashlib

def proof_of_work(data, difficulty):
    nonce = 0
    target = '0' * difficulty

    while True:
        combined_data = data + str(nonce)
        hash_result = hashlib.sha256(combined_data.encode()).hexdigest()

        if hash_result.startswith(target):
            return nonce, hash_result

        nonce += 1

# Example usage
data = input("Enter data: ")
difficulty = int(input("Enter difficulty (number of leading zeros): "))

nonce, hash_result = proof_of_work(data, difficulty)
print("Nonce found:", nonce)
print("Hash:", hash_result)
```

```
Enter data: hello pranav
Enter difficulty (number of leading zeros): 4
Nonce found: 901
Hash: 00004af087e7e29acab163e985c7cb5e5d439e08c0f8e9b14d4c193880b4dc4c
```

```python
# Merkle Tree Construction

import hashlib

def sha256(data):
    return hashlib.sha256(data.encode()).hexdigest()

def merkle_root(transactions):
    # Step 1: Hash all transactions
    current_level = [sha256(tx) for tx in transactions]

    # Step 2: Build tree until one hash remains
    while len(current_level) > 1:
        next_level = []

        # If odd number of hashes, duplicate last one
        if len(current_level) % 2 != 0:
            current_level.append(current_level[-1])

        # Hash pairs
        for i in range(0, len(current_level), 2):
            combined_hash = current_level[i] + current_level[i + 1]
            next_level.append(sha256(combined_hash))

        current_level = next_level

    return current_level[0]

# Example usage
transactions = [
    "Alice pays Bob 10 BTC",
    "Bob pays Charlie 5 BTC",
    "Charlie pays Dave 2 BTC",
    "Dave pays Eve 1 BTC"
]
print("Merkle Root:", merkle_root(transactions))
```



```
Merkle Root: 971669fd8fe0f5915600b460f2751cd03a4f82e2a16b7f8b154f2b3ca31500b8
```

**Conclusion -** Merkle Trees are a core cryptographic structure that provide security, integrity, and efficiency in blockchain and distributed systems.