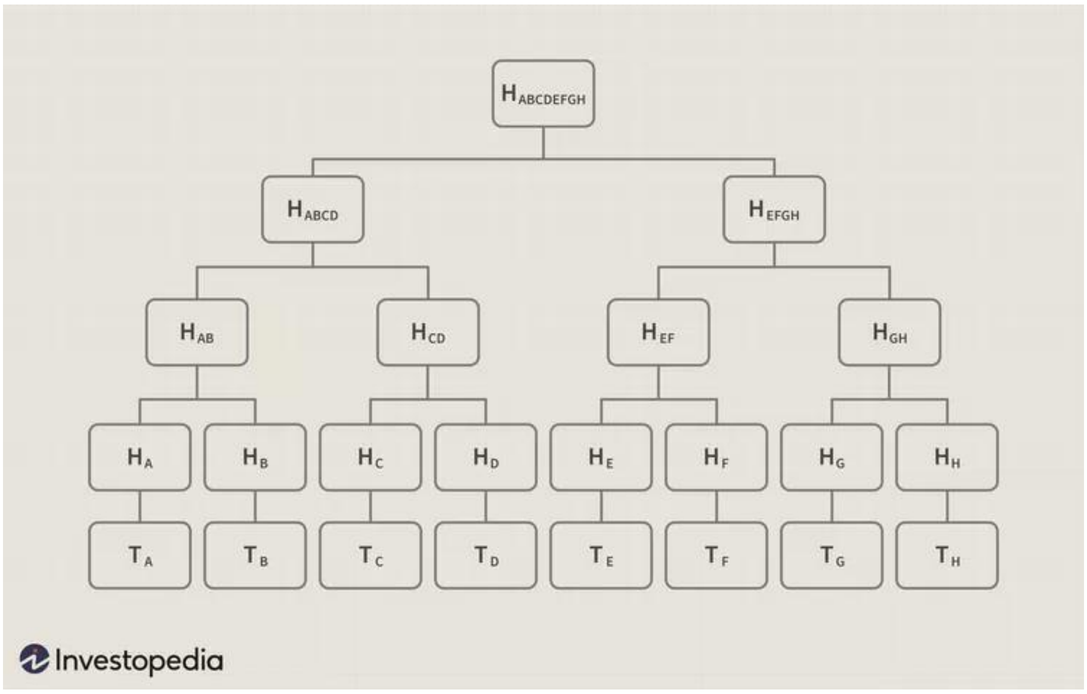
Blockchain Technology Lab

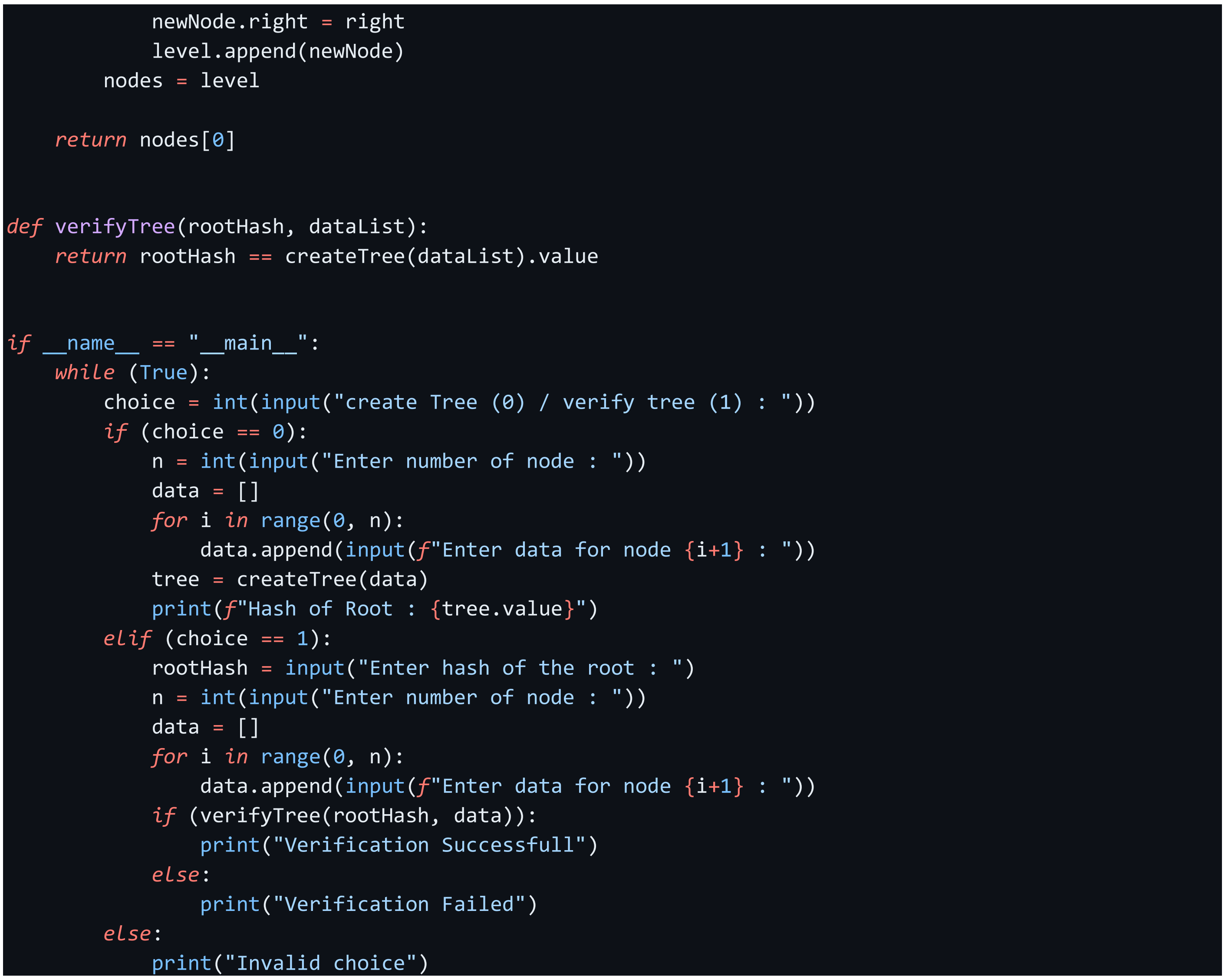
Lab – 5

Aim : Implement Merkle Tree to learn data structure used in blockchain.



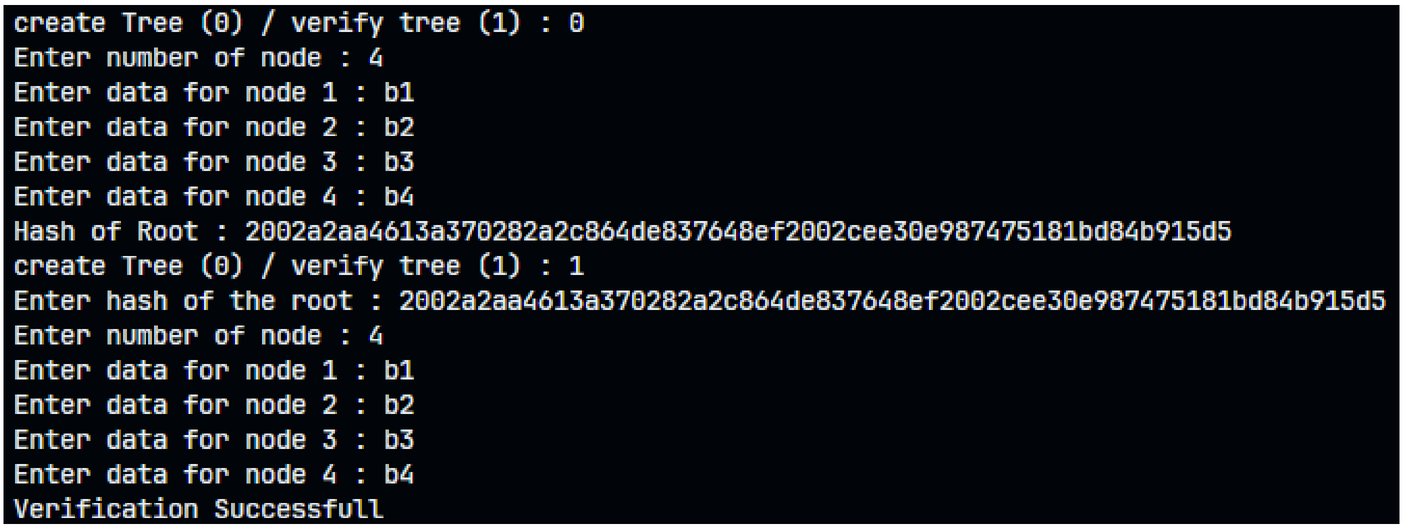
Code:

```python
import hashlib


class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


def hashData(data):
    return hashlib.sha256(data.encode('utf-8')).hexdigest()


def createTree(dataList):
    if len(dataList) % 2 != 0:
        dataList.append(dataList[-1])

    nodes = [TreeNode(hashData(data)) for data in dataList]

    while (len(nodes) > 1):
        level = []
        for i in range(0, len(nodes), 2):
            left = nodes[i]
            right = nodes[i+1]
            contcatedData = hashData(left.value+right.value)
            newNode = TreeNode(contcatedData)
            newNode.left = left
```



```python
            newNode.right = right
            level.append(newNode)
        nodes = level

    return nodes[0]


def verifyTree(rootHash, dataList):
    return rootHash == createTree(dataList).value


if __name__ == "__main__":
    while (True):
        choice = int(input("create Tree (0) / verify tree (1) : "))
        if (choice == 0):
            n = int(input("Enter number of node : "))
            data = []
            for i in range(0, n):
                data.append(input(f"Enter data for node {i+1} : "))
            tree = createTree(data)
            print(f"Hash of Root : {tree.value}")
        elif (choice == 1):
            rootHash = input("Enter hash of the root : ")
            n = int(input("Enter number of node : "))
            data = []
            for i in range(0, n):
                data.append(input(f"Enter data for node {i+1} : "))
            if (verifyTree(rootHash, data)):
                print("Verification Successfull")
            else:
                print("Verification Failed")
        else:
            print("Invalid choice")
```

Output:



```
create Tree (0) / verify tree (1) : 0
Enter number of node : 4
Enter data for node 1 : b1
Enter data for node 2 : b2
Enter data for node 3 : b3
Enter data for node 4 : b4
Hash of Root : 2002a2aa4613a370282a2c864de837648ef2002cee30e987475181bd84b915d5
create Tree (0) / verify tree (1) : 1
Enter hash of the root : 2002a2aa4613a370282a2c864de837648ef2002cee30e987475181bd84b915d5
Enter number of node : 4
Enter data for node 1 : b1
Enter data for node 2 : b2
Enter data for node 3 : b3
Enter data for node 4 : b4
Verification Successfull
```