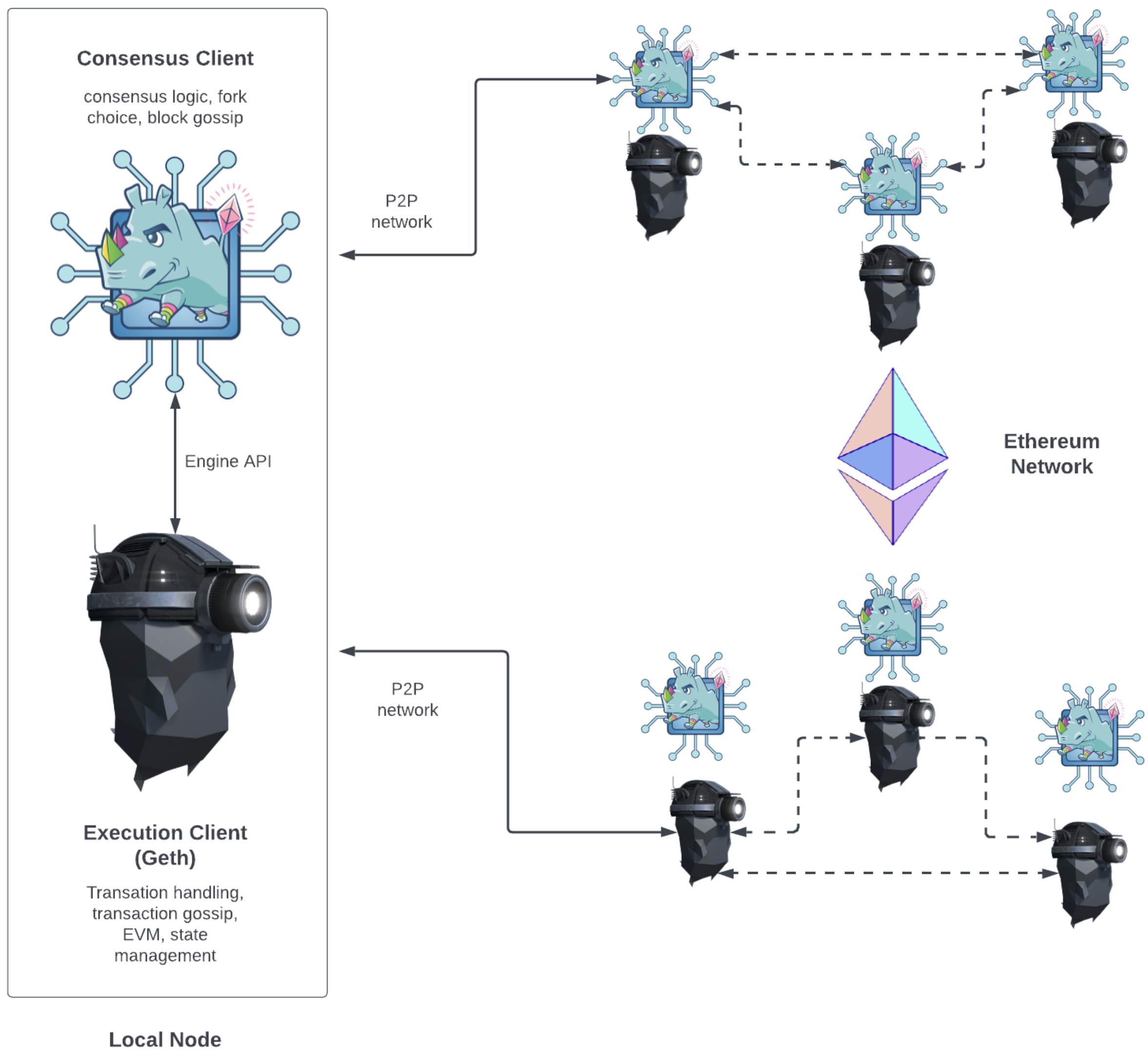
Name: Prasad Jawale **Class**: D16AD **Roll**: 20

Blockchain Lab 1

Aim: To install and set up an Ethereum network to create a private Ethereum blockchain for development and testing purposes.

Theory: Ethereum node is composed of two clients: an execution client and a consensus client. Geth is an execution client. Originally, an execution client alone was enough to run a full Ethereum node. However, ever since Ethereum turned off proof-of-work and implemented proof-of-stake, Geth has needed to be coupled to another piece of software called a “consensus client” in order to keep track of the Ethereum blockchain.The execution client (Geth) is responsible for transaction handling, transaction gossip, state management and supporting the Ethereum Virtual Machine EVM. However, Geth is not responsible for block building, block gossiping or handling consensus logic. These are in the remit of the consensus client.The relationship between the two Ethereum clients is shown in the schematic below. The two clients each connect to their own respective peer-to-peer (P2P) networks. This is because the execution clients gossip transactions over their P2P network enabling them to manage their local transaction pool. The consensus clients gossip blocks over their P2P network, enabling consensus and chain growth.

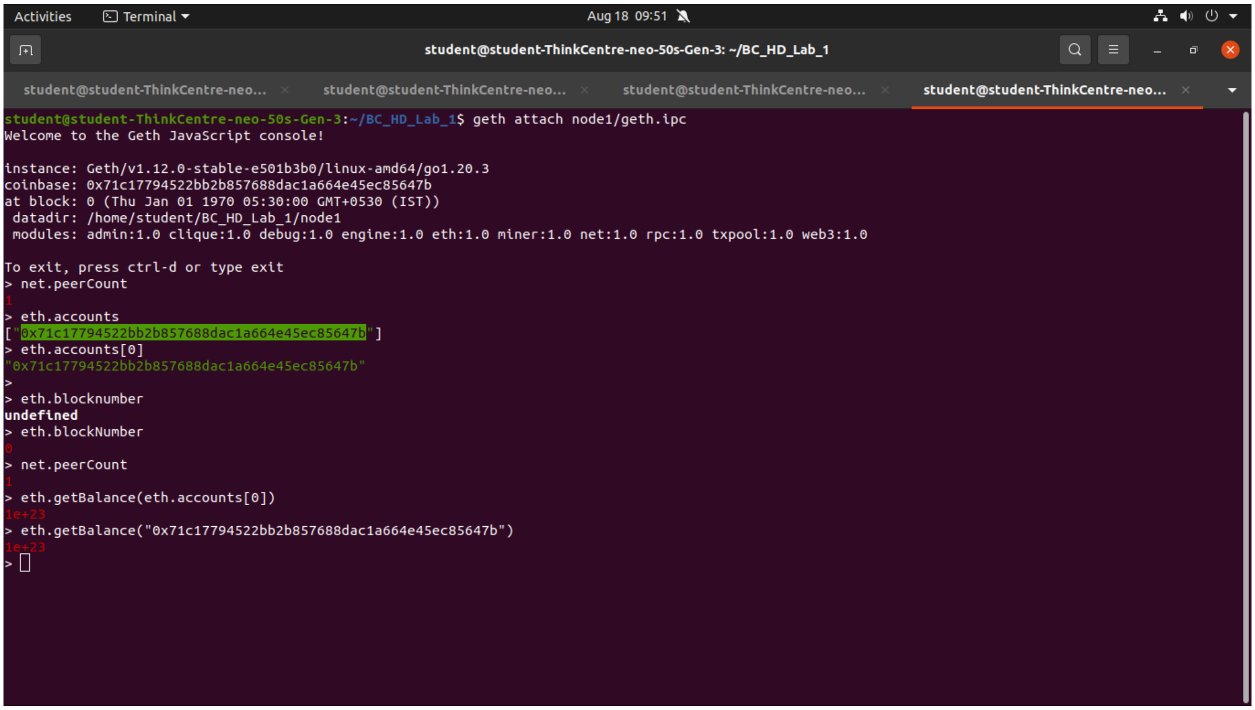
Node-architecture:



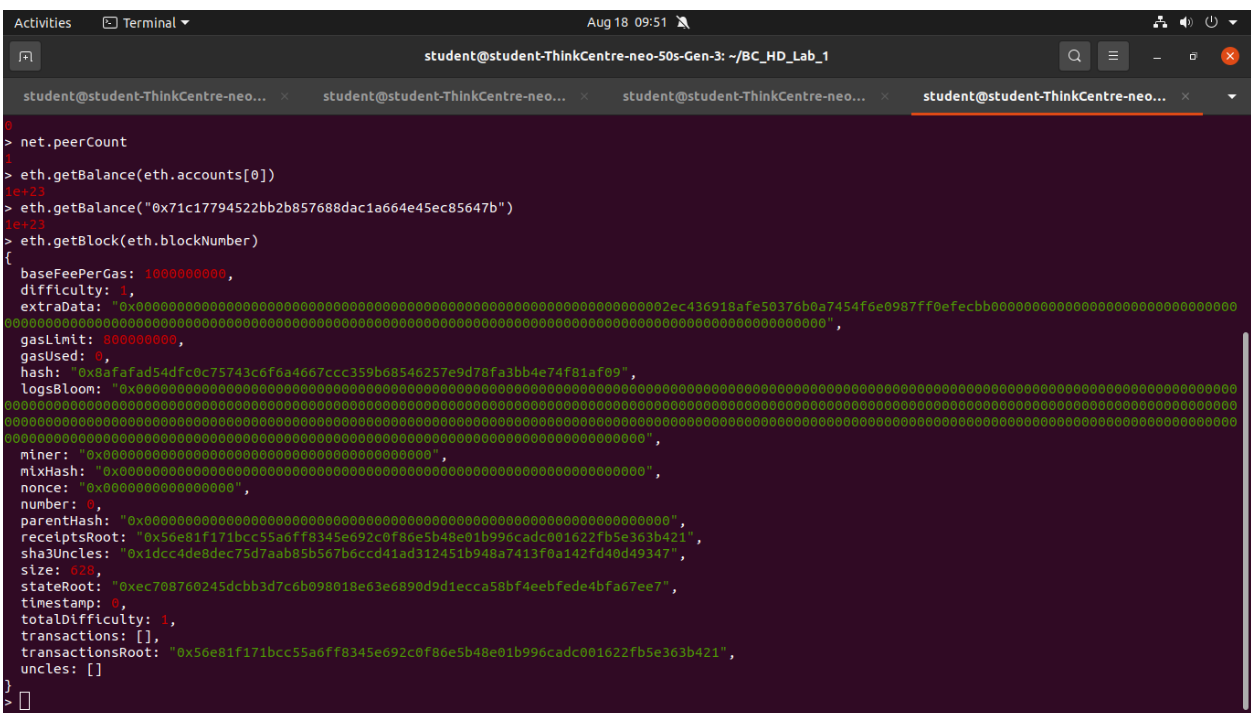
For this two-client structure to work, consensus clients must be able to pass bundles of transactions to Geth to be executed. Executing the transactions locally is how the client validates that the transactions do not violate any Ethereum rules and that the proposed update to Ethereum's state is correct. Likewise, when the node is selected to be a block producer the consensus client must be able to request bundles of transactions from Geth to include in the new block. This inter-client communication is handled by a local RPC connection using the engine API.

Program:

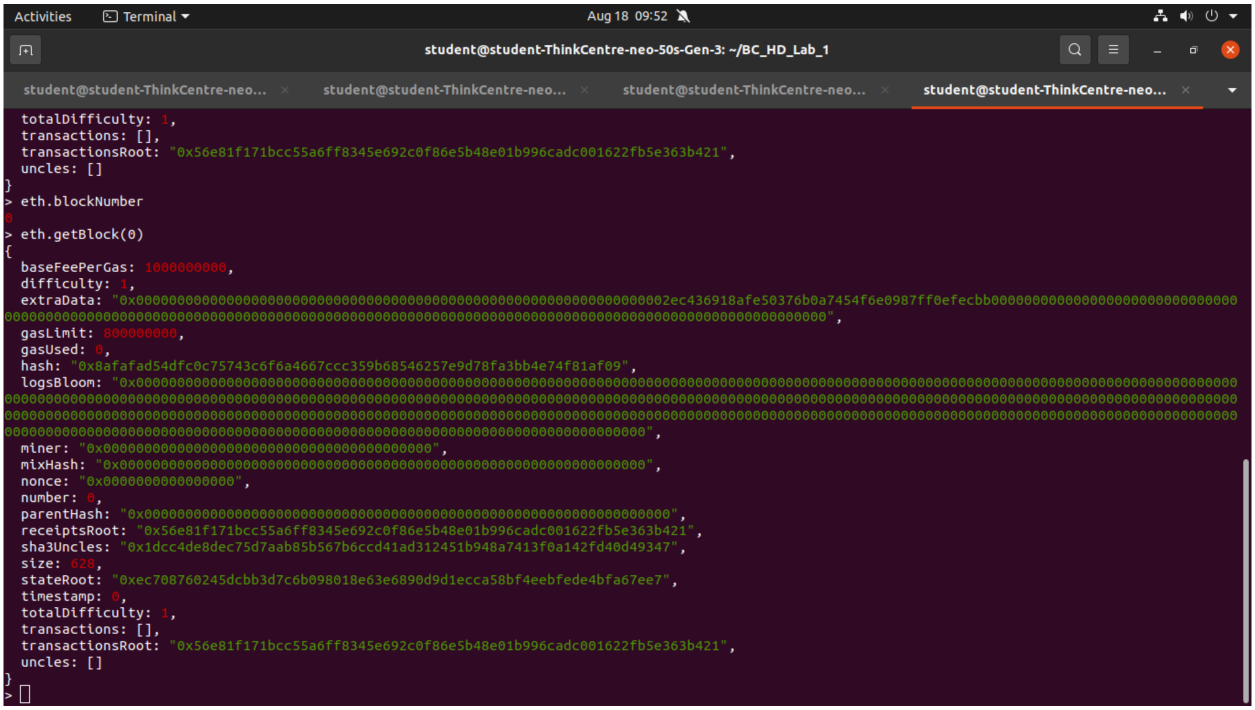
1. Geth commands

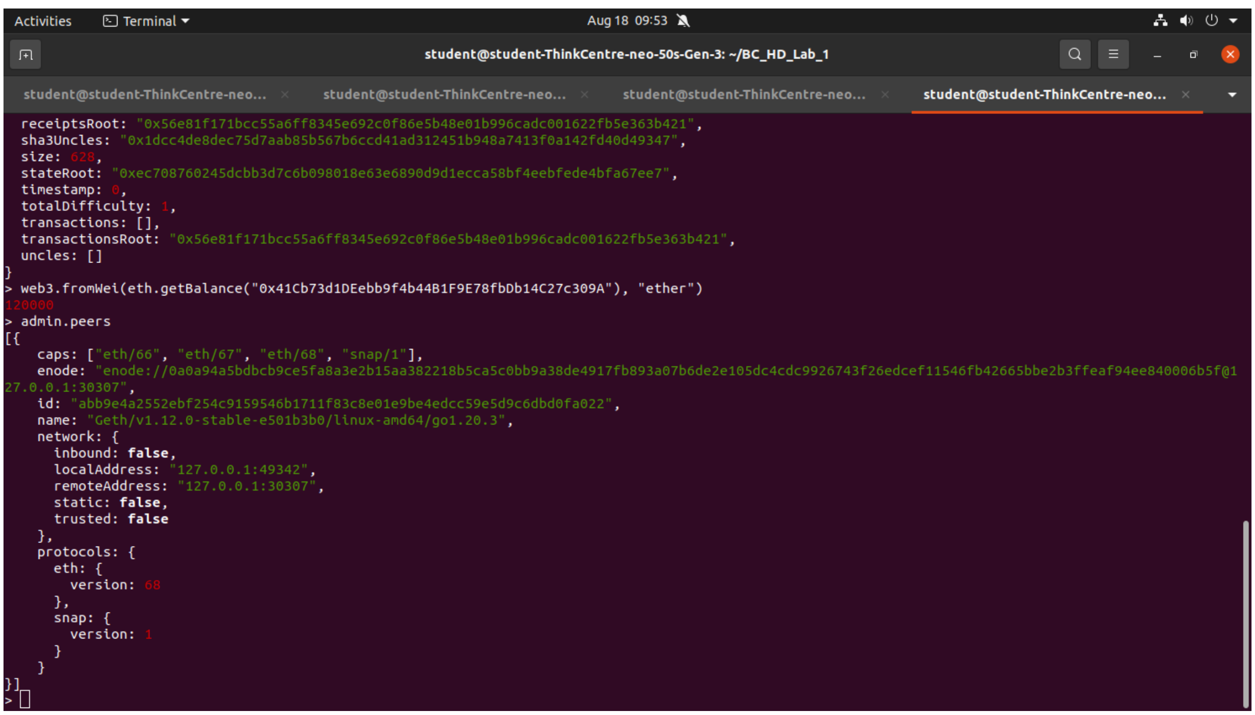



```
student@student-ThinkCentre-neo-50s-Gen-3:~/BC_HD_Lab_1$ geth attach node1/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.12.0-stable-e501b3b0/linux-amd64/go1.20.3
coinbase: 0x71c17794522bb2b857688dac1a664e45ec85647b
at block: 0 (Thu Jan 01 1970 05:30:00 GMT+0530 (IST))
 datadir: /home/student/BC_HD_Lab_1/node1
 modules: admin:1.0 clique:1.0 debug:1.0 engine:1.0 eth:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> net.peerCount
1
> eth.accounts
["0x71c17794522bb2b857688dac1a664e45ec85647b"]
> eth.accounts[0]
"0x71c17794522bb2b857688dac1a664e45ec85647b"
>
> eth.blocknumber
undefined
> eth.blockNumber
0
> net.peerCount
1
> eth.getBalance(eth.accounts[0])
1e+23
> eth.getBalance("0x71c17794522bb2b857688dac1a664e45ec85647b")
1e+23
> 
```

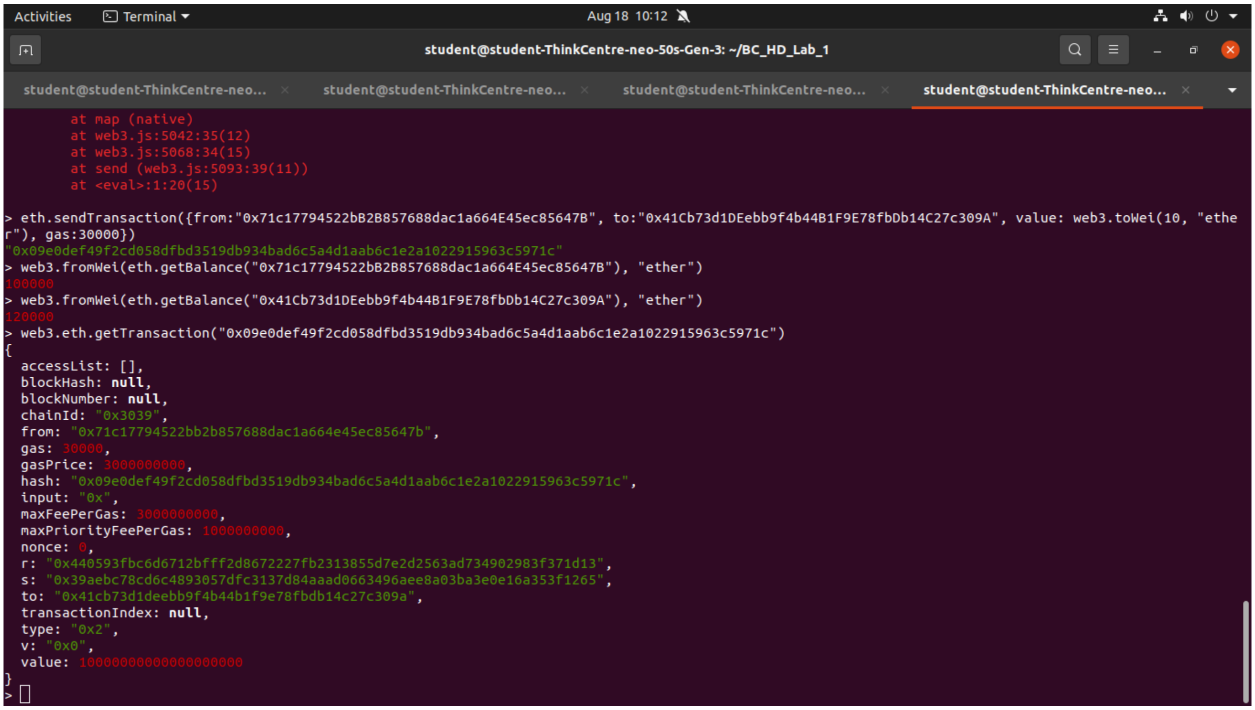



```
0
> net.peerCount
1
> eth.getBalance(eth.accounts[0])
1e+23
> eth.getBalance("0x71c17794522bb2b857688dac1a664e45ec85647b")
1e+23
> eth.getBlock(eth.blockNumber)
{
  baseFeePerGas: 1000000000,
  difficulty: 1,
  extraData: "0x00000000000000000000000000000000000000000000000000000000000000002ec436918afe50376b0a7454f6e0987ff0efecbb0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  gasLimit: 800000000,
  gasUsed: 0,
  hash: "0x8afafad54dfc0c75743c6f6a4667ccc359b68546257e9d78fa3bb4e74f81af09",
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  miner: "0x0000000000000000000000000000000000000000",
  mixHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  nonce: "0x0000000000000000",
  number: 0,
  parentHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 628,
  stateRoot: "0xec708760245dcbb3d7c6b098018e63e6890d9d1ecca58bf4eebfede4bfa67ee7",
  timestamp: 0,
  totalDifficulty: 1,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  uncles: []
}
> 
```

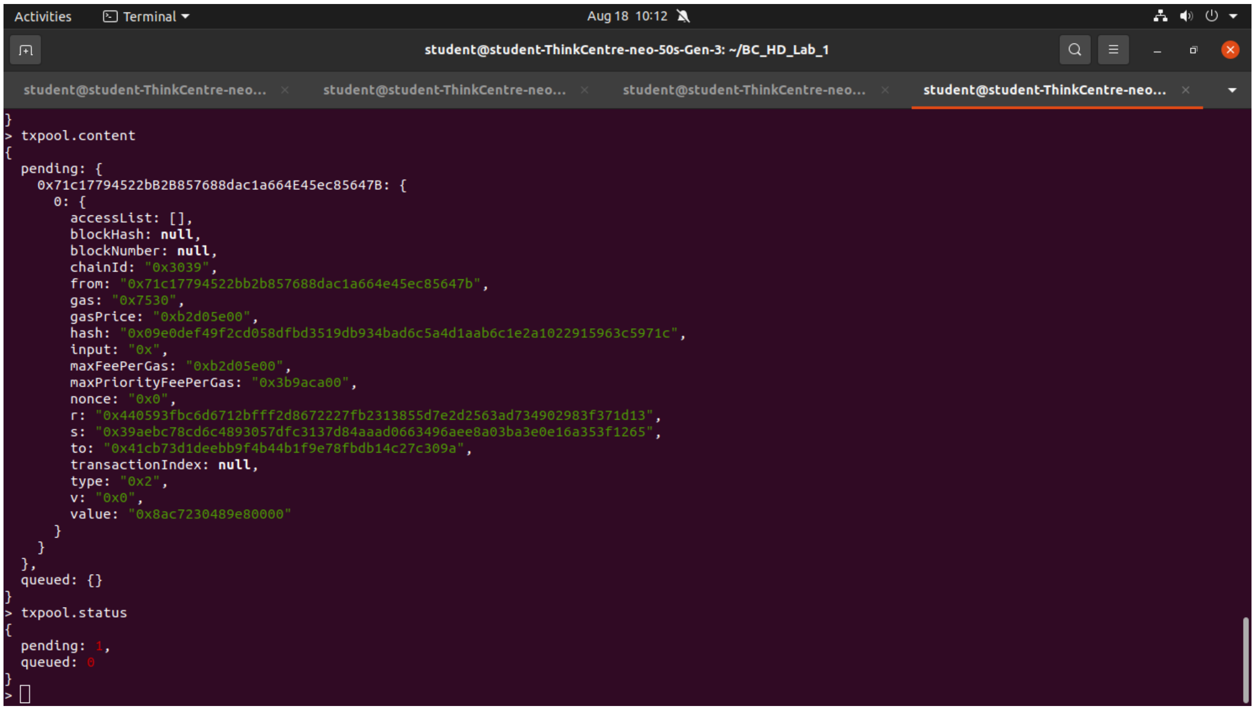

Activities
Terminal
Aug 18 09:52
student@student-ThinkCentre-neo-50s-Gen-3: ~/BC_HD_Lab_1
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
  totalDifficulty: 1,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  uncles: []
}
> eth.blockNumber
0
> eth.getBlock(0)
{
  baseFeePerGas: 1000000000,
  difficulty: 1,
  extraData: "0x00000000000000000000000000000000000000000000000000000000000000002ec436918afe50376b0a7454f6e0987ff0efecbb00000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  gasLimit: 800000000,
  gasUsed: 0,
  hash: "0x8afafad54dfc0c75743c6f6a4667ccc359b68546257e9d78fa3bb4e74f81af09",
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000",
  miner: "0x0000000000000000000000000000000000000000",
  mixHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  nonce: "0x0000000000000000",
  number: 0,
  parentHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 628,
  stateRoot: "0xec708760245dcbb3d7c6b098018e63e6890d9d1ecca58bf4eebfede4bfa67ee7",
  timestamp: 0,
  totalDifficulty: 1,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  uncles: []
}
>

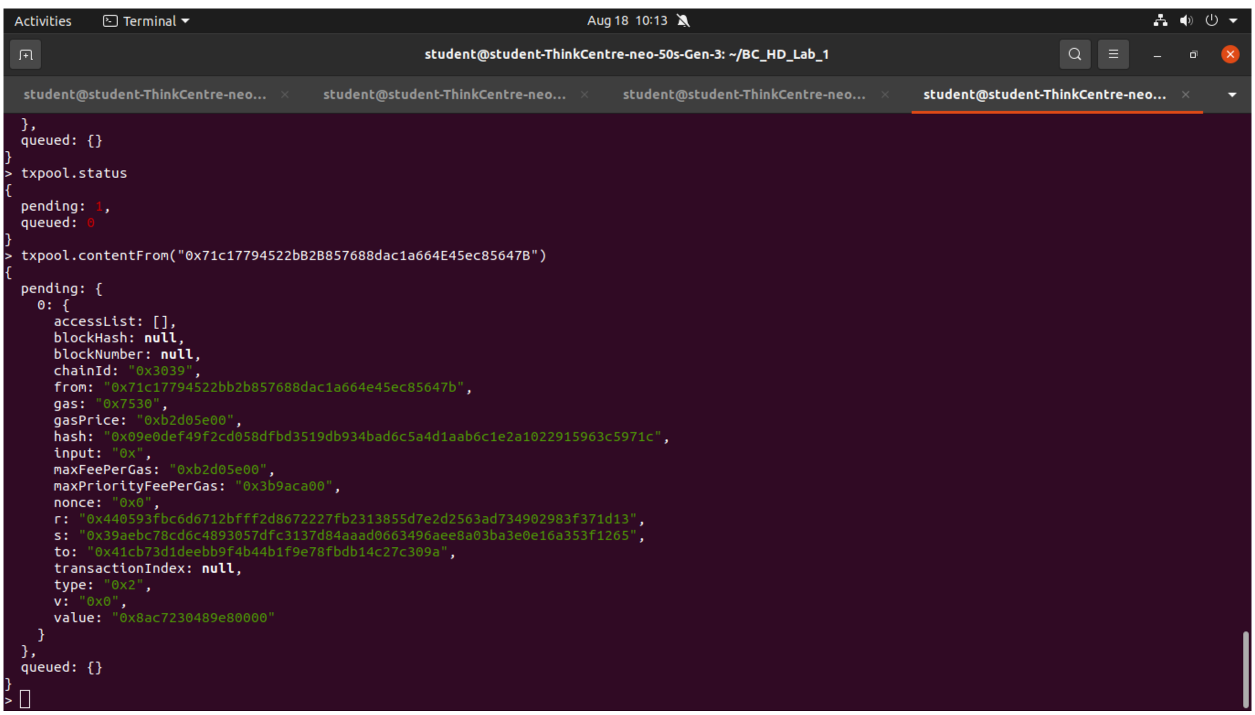

Activities
Terminal
Aug 18 09:53
student@student-ThinkCentre-neo-50s-Gen-3: ~/BC_HD_Lab_1
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
student@student-ThinkCentre-neo...
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 628,
  stateRoot: "0xec708760245dcbb3d7c6b098018e63e6890d9d1ecca58bf4eebfede4bfa67ee7",
  timestamp: 0,
  totalDifficulty: 1,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
  uncles: []
}
> web3.fromWei(eth.getBalance("0x41Cb73d1DEebb9f4b44B1F9E78fbDb14C27c309A"), "ether")
120000
> admin.peers
[{
    caps: ["eth/66", "eth/67", "eth/68", "snap/1"],
    enode: "enode://0a0a94a5bdbcb9ce5fa8a3e2b15aa382218b5ca5c0bb9a38de4917fb893a07b6de2e105dc4cdc9926743f26edcef11546fb42665bbe2b3ffeaf94ee840006b5f@1
27.0.0.1:30307",
    id: "abb9e4a2552ebf254c9159546b1711f83c8e01e9be4edcc59e5d9c6dbd0fa022",
    name: "Geth/v1.12.0-stable-e501b3b0/linux-amd64/go1.20.3",
    network: {
      inbound: false,
      localAddress: "127.0.0.1:49342",
      remoteAddress: "127.0.0.1:30307",
      static: false,
      trusted: false
    },
    protocols: {
      eth: {
        version: 68
      },
      snap: {
        version: 1
      }
    }
}]
>

2. Transaction from Node1 to Node2



3. Content of transaction pool





Conclusion : Hence we have successfully implemented the Ethereum network for testing and development purpose .

Name: Prasad Jawale **Class**: D16AD **Roll**: 20

Blockchain Lab 2

Aim: To create a blockchain using python

Theory: Creating a blockchain from scratch using Python involves implementing the core principles and components of a blockchain system. Here's a theoretical overview of the key elements you'll need to understand when building a basic blockchain:

- Blocks and Chain: A blockchain is a distributed ledger composed of a chain of blocks. Each block contains a set of transactions and a reference (usually a hash) to the previous block in the chain, forming a linked list. The linking ensures the chronological order of transactions and adds security through immutability.
- Transactions: Transactions represent the data being stored on the blockchain. In a cryptocurrency blockchain like Bitcoin, transactions include sender and receiver addresses, transaction amount, digital signatures, and other relevant information. For non-cryptocurrency blockchains, the content of transactions can vary widely.
- Hash Functions: Cryptographic hash functions like SHA-256 are used to create unique, fixed-size hashes for blocks and transactions. Hashes are essential for data integrity and linking blocks together. Any change in the block's content results in a completely different hash.
- Mining and Proof of Work: Mining is the process by which new blocks are added to the blockchain. Miners (computers or nodes) compete to solve a computationally intensive mathematical puzzle, often called Proof of Work (PoW), to create a new block. The first miner to solve the puzzle broadcasts the new block to the network, and other nodes verify its validity.

- Consensus Mechanism: Blockchain networks use a consensus mechanism to agree on the validity of transactions and blocks. PoW, Proof of Stake (PoS), and Delegated Proof of Stake (DPoS) are common consensus mechanisms. PoW ensures that miners put in computational work to validate transactions and create new blocks.
- Decentralization: One of the core principles of a blockchain is decentralization, meaning that no single entity or authority has control over the network. Nodes (computers) in the network maintain copies of the blockchain and collectively validate transactions and maintain the ledger.
- Security: Security is paramount in blockchain. Cryptographic techniques are used for secure transactions and block validation. Private and public keys, digital signatures, and encryption are used to protect data.
- Peer-to-Peer Networking: Blockchain networks are typically decentralized and rely on peer-to-peer networking protocols. Nodes communicate with each other to propagate transactions and blocks, ensuring network integrity.
- Smart Contracts (Optional): Some blockchains, like Ethereum, support smart contracts, which are self-executing code that runs on the blockchain. Smart contracts enable programmable and automated transactions and are written in languages like Solidity.
- Testing and Validation: Extensive testing and validation are necessary to ensure that the blockchain operates correctly and securely. This includes testing the consensus mechanism, validating transactions, and checking the overall network health.

Creating a blockchain from scratch can be a complex and time-consuming task, but it provides valuable insights into blockchain technology. You can start by implementing a basic blockchain with the above principles in mind and gradually add more features and complexity as you become more comfortable with the concepts. There are also open-source blockchain libraries and frameworks available in Python that can help streamline the development process.

Program:

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

def build_merkle_tree(transactions):
    if len(transactions) == 0:
        return None

    if len(transactions) == 1:
        return transactions[0]

    # Recursive construction of the Merkle Tree
    while len(transactions) > 1:
        if len(transactions) % 2 != 0:
            transactions.append(transactions[-1])

        new_transactions = []
        for i in range(0, len(transactions), 2):
            combined = transactions[i] + transactions[i+1]
            hash_combined = hashlib.sha256(
                    combined.encode('utf-8')).hexdigest()
            new_transactions.append(hash_combined)

        transactions = new_transactions

    return transactions[0]


class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
        self.mine_genesis_block()


    def create_block(self, previous_hash, merkle_root, block_transactions):
        block = {'index': len(self.chain) + 1,
                 'timestamp': str(datetime.datetime.now()),
```

```
            'proof': 1,
            'previous_hash': previous_hash,
            'merkle_root': merkle_root,
            'transactions': block_transactions
           }
        return block


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

    def remove_transactions(self):
        self.transactions = []

    def proof_of_work(self, block):
        block['proof'] = 1
        while True:
            encoded_block = json.dumps(block, sort_keys = True).encode()
            hash_operation = hashlib.sha256(encoded_block).hexdigest()
            if hash_operation.startswith("0000"):
                self.chain.append(block)
                self.remove_transactions()
                return block['proof']
            block['proof'] += 1


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


    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            hash_operation = hashlib.sha256(json.dumps(block, sort_keys =
True).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
```

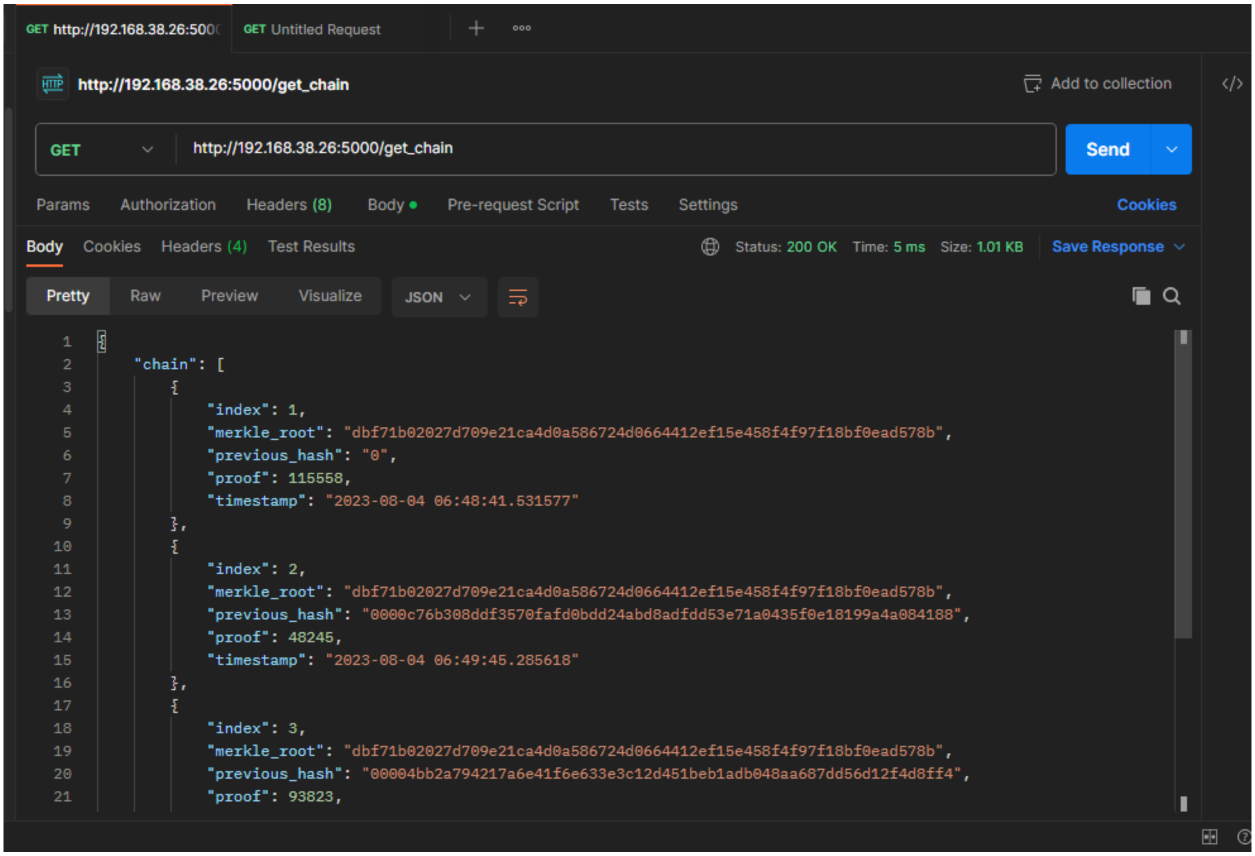
```
            previous_block = block
            block_index += 1
        return True

    def mine_genesis_block(self):
        merkle_root = build_merkle_tree(self.transactions)
        genesis_block = self.create_block(previous_hash='0', merkle_root=merkle_root,
block_transactions=self.transactions)
        self.proof_of_work(genesis_block)

# Creating a Web App
app = Flask(__name__)

# Creating a Blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    previous_hash = blockchain.hash(previous_block)

    merkle_root = build_merkle_tree(blockchain.transactions)
    block = blockchain.create_block(previous_hash, merkle_root,blockchain.transactions)

    blockchain.proof_of_work(block)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'merkle_root': block['merkle_root'],
                'transactions':block['transactions']}
    return jsonify(response), 200

# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200
```

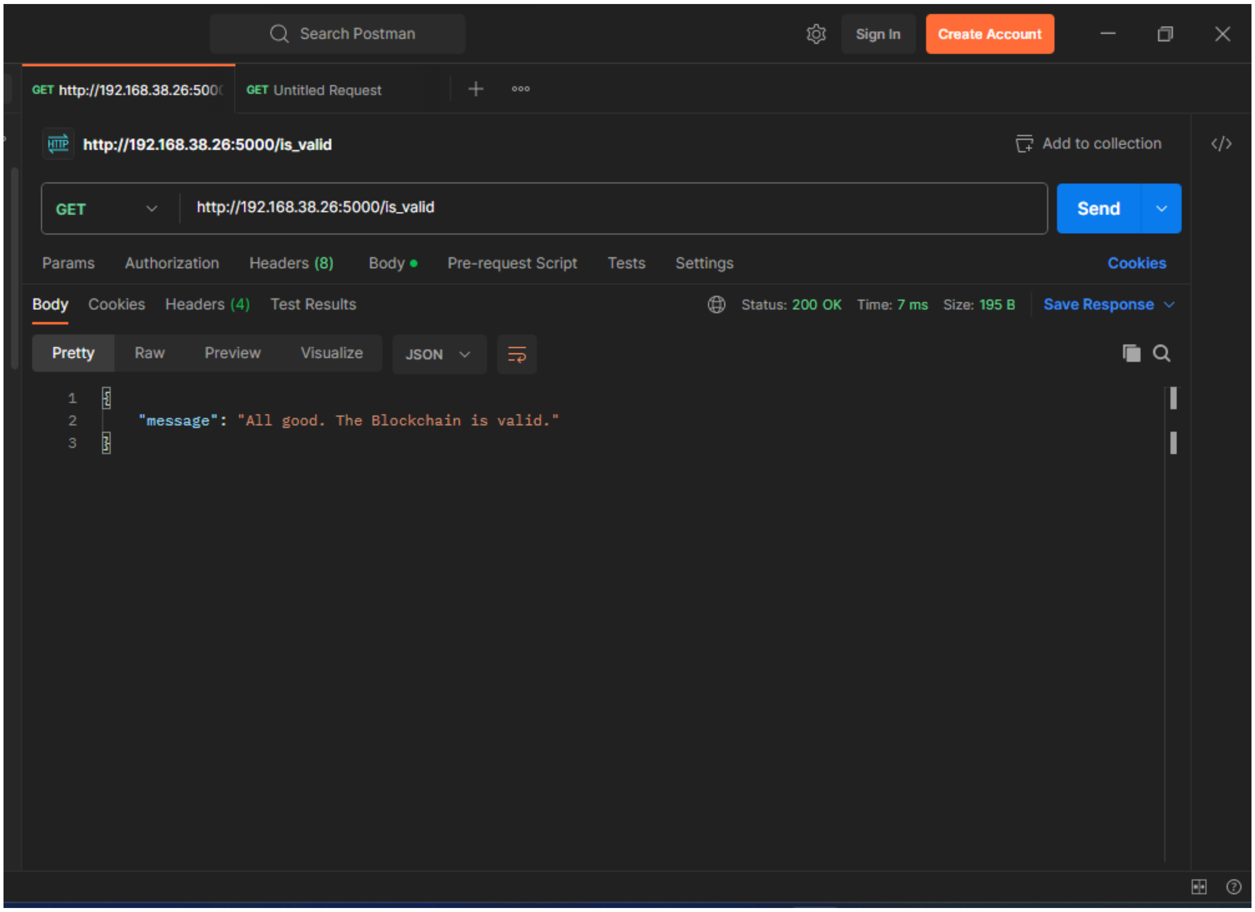
```
# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}
    return jsonify(response), 200

@app.route('/add_transaction', methods = ['POST'])
def add_transaction():
    data = request.get_json()
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in data for key in transaction_keys):
        return 'Some elements of the transaction are missing', 400

    sender = data['sender']
    receiver = data['receiver']
    amount = data['amount']
    current_transaction = f'{sender} -> {receiver}: ${amount}'
    blockchain.transactions.append(current_transaction)

    return jsonify('Transaction added'), 200

# Running the app
app.run(host = '0.0.0.0', port = 5000)
```

Starting flask server

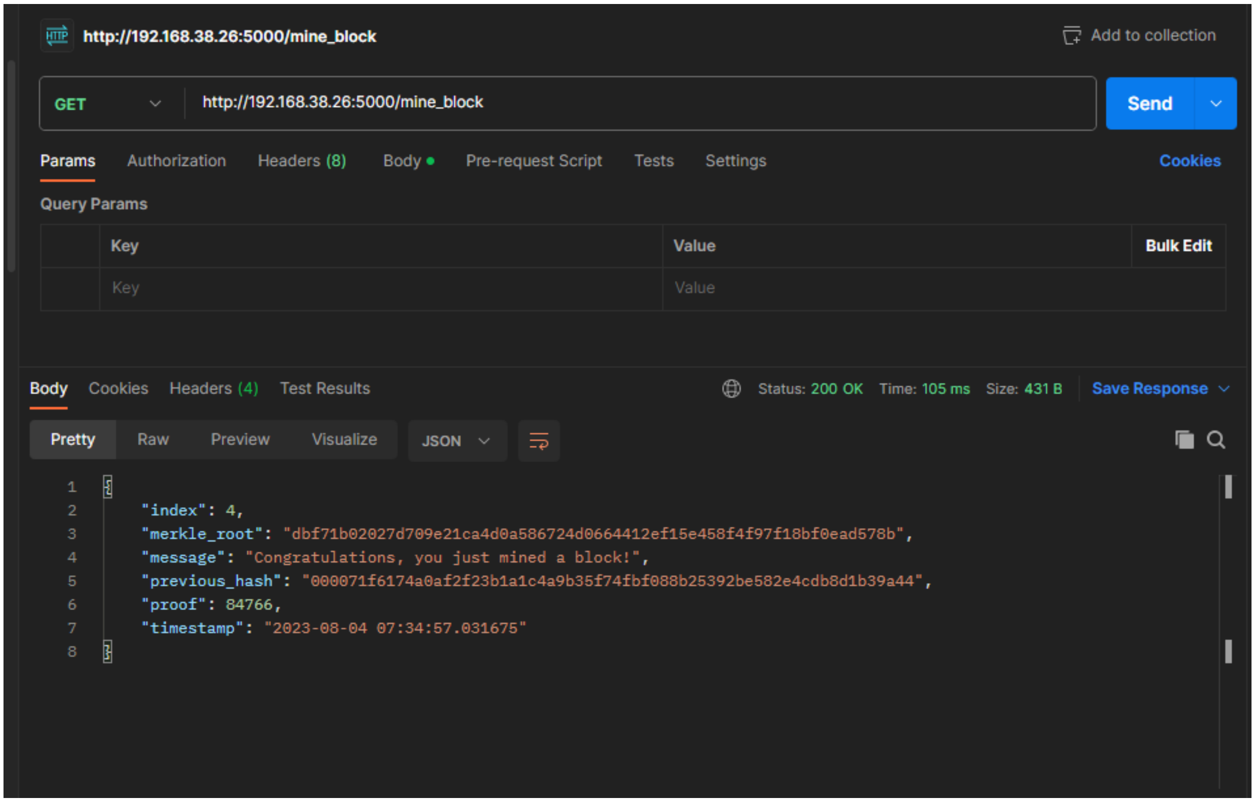
```
C:\Users\Student\Desktop\D17A_28_BC>python BCLab2.py
 * Serving Flask app 'BCLab2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.38.26:5000
Press CTRL+C to quit
```

HTTP Requests on Postman


GET http://192.168.38.26:500(
GET Untitled Request
http://192.168.38.26:5000/get_chain
Add to collection
GET
http://192.168.38.26:5000/get_chain
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 5 ms
Size: 1.01 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"chain": [
{
"index": 1,
"merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
"previous_hash": "0",
"proof": 115558,
"timestamp": "2023-08-04 06:48:41.531577"
},
{
"index": 2,
"merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
"previous_hash": "0000c76b308ddf3570fafd0bdd24abd8adfdd53e71a0435f0e18199a4a084188",
"proof": 48245,
"timestamp": "2023-08-04 06:49:45.285618"
},
{
"index": 3,
"merkle_root": "dbf71b02027d709e21ca4d0a586724d0664412ef15e458f4f97f18bf0ead578b",
"previous_hash": "00004bb2a794217a6e41f6e633e3c12d451beb1adb048aa687dd56d12f4d8ff4",
"proof": 93823,



Search Postman
Sign In
Create Account
GET http://192.168.38.26:500(
GET Untitled Request
http://192.168.38.26:5000/is_valid
Add to collection
GET
http://192.168.38.26:5000/is_valid
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 7 ms
Size: 195 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "All good. The Blockchain is valid."
}



Conclusion: Blockchain has been implemented in Python

Name: Prasad Jawale **Class**: D16AD **Roll**: 20

Blockchain Lab 3

Aim: Study on Solidity Programming for creating Smart Contracts

Theory: Solidity is a high-level programming language specifically designed for creating smart contracts on blockchain platforms like Ethereum. Smart contracts are self-executing contracts with the terms and conditions of the agreement directly written into code. Solidity plays a pivotal role in enabling the automation and decentralization of various applications, such as decentralized finance (DeFi), non-fungible tokens (NFTs), and supply chain management, by ensuring the trustworthiness and transparency of these applications through the blockchain. To understand Solidity, one must grasp its core concepts.

One fundamental concept in Solidity is the Ethereum Virtual Machine (EVM), which executes the smart contract code. Smart contracts operate in a trustless environment, meaning they execute autonomously without relying on a central authority. Solidity includes data types, variables, and control structures to facilitate the creation of these self-executing contracts. It also supports inheritance, allowing developers to build on existing smart contracts to create more complex and feature-rich applications.

Program:

Assignment 1:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Counter {

   uint public count;

   // Function to get the current count

   function get() public view returns (uint) {

       return count;

   }

   // Function to increment count by 1

   function inc() public {

       count += 1;

   }

   // Function to decrement count by 1

   function dec() public {

       count -= 1;

   }

}
```

Assignment 2

```
// SPDX-License-Identifier: MIT

// compiler version must be greater than or equal to 0.8.3 and less than
0.9.0
```

```
pragma solidity ^0.8.3;


contract MyContract {
    string public name = "Alice";
}
```

Assignment 3

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;


contract Primitives {
    bool public boo = true;


    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available
        uint8   ranges from 0 to 2 ** 8 - 1
        uint16  ranges from 0 to 2 ** 16 - 1
        ...
        uint256 ranges from 0 to 2 ** 256 - 1
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256
```

```
/*
Negative numbers are allowed for int types.
Like uint, different ranges are available from int8 to int256
*/
int8 public i8 = -1;
int public i256 = 456;
int public i = -123; // int is same as int256

address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

// Default values
// Unassigned variables have a default value
bool public defaultBoo; // false
uint public defaultUint; // 0
int public defaultInt; // 0
address public defaultAddr; //
0x0000000000000000000000000000000000000000
// New values
address public newAddr = 0x0000000000000000000000000000000000000000;
int public neg = -12;
uint8 public newU = 0;
}
```

Assignment 4

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Variables {
   // State variables are stored on the blockchain.
   string public text = "Hello";
   uint public num = 123;
   uint public blockNumber;

   function doSomething() public {
       // Local variables are not saved to the blockchain.
       uint i = 456;

       // Here are some global variables
       uint timestamp = block.timestamp; // Current block timestamp
       address sender = msg.sender; // address of the caller
       blockNumber = block.number;
   }
}
```

Assignment 5

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract SimpleStorage {
```

```
    // State variable to store a number
    uint public num;
    bool public b = true;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {
        return num;
    }

    function get_b() public view returns (bool) {
        return b;
    }
}
```

Assignment 6

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract ViewAndPure {
    uint public x = 1;

    // Promise not to modify the state.
    function addToX(uint y) public view returns (uint) {
        return x + y;
    }

    // Promise not to modify or read from the state.
    function add(uint i, uint j) public pure returns (uint) {
        return i + j;
    }

    function addToX2(uint y) public {
        x = x+ y;
    }
}
```

Assignment 7

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract FunctionModifier {
    // We will use these variables to demonstrate how to use
    // modifiers.
    address public owner;
    uint public x = 10;
    bool public locked;

    constructor() {
        // Set the transaction sender as the owner of the contract.
        owner = msg.sender;
    }

    // Modifier to check that the caller is the owner of
    // the contract.
    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        // Underscore is a special character only used inside
        // a function modifier and it tells Solidity to
        // execute the rest of the code.
        _;
    }

    // Modifiers can take inputs. This modifier checks that the
    // address passed in is not the zero address.
    modifier validAddress(address _addr) {
        require(_addr != address(0), "Not valid address");
        _;
    }

    function changeOwner(address _newOwner) public onlyOwner
validAddress(_newOwner) {
        owner = _newOwner;
    }
```

```
    // Modifiers can be called before and / or after a function.
    // This modifier prevents a function from being called while
    // it is still executing.
    modifier noReentrancy() {
        require(!locked, "No reentrancy");

        locked = true;
        _;
        locked = false;
    }

    modifier biggerThan0(uint y) {
        require(y > 0, "Not bigger than x");
        _;
    }

    modifier increaseXbyY(uint y) {
        _;
        x = x + y;
    }

    function increaseX(uint y) public onlyOwner biggerThan0(y)
increaseXbyY(y){
    }

    function decrement(uint i) public noReentrancy {
        x -= i;

        if (i > 1) {
            decrement(i - 1);
        }
    }

}
```

Assignment 8

```
// SPDX-License-Identifier: MIT
```

```solidity
pragma solidity ^0.8.3;

contract Function {
   // Functions can return multiple values.
   function returnMany()
       public
       pure
       returns (
           uint,
           bool,
           uint
       )
   {
       return (1, true, 2);
   }

   // Return values can be named.
   function named()
       public
       pure
       returns (
           uint x,
           bool b,
           uint y
       )
   {
       return (1, true, 2);
   }

   // Return values can be assigned to their name.
   // In this case the return statement can be omitted.
   function assigned()
       public
       pure
       returns (
           uint x,
           bool b,
           uint y
       )
   {
```

```
        x = 1;
        b = true;
        y = 2;
    }

    // Use destructing assignment when calling another
    // function that returns multiple values.
    function destructingAssigments()
        public
        pure
        returns (
            uint,
            bool,
            uint,
            uint,
            uint
        )
    {
        (uint i, bool b, uint j) = returnMany();

        // Values can be left out.
        (uint x, , uint y) = (4, 5, 6);

        return (i, b, j, x, y);
    }

    // Cannot use map for neither input nor output

    // Can use array for input
    function arrayInput(uint[] memory _arr) public {}

    // Can use array for output
    uint[] public arr;

    function arrayOutput() public view returns (uint[] memory) {
        return arr;
    }

    function returnTwo() public pure  returns (
            int x,
```

```
            bool b
        ){
         x = -2;
        b = true;
    }
}
```

Assignment 9

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Base {
    // Private function can only be called
    // - inside this contract
    // Contracts that inherit this contract cannot call this function.
    function privateFunc() private pure returns (string memory) {
        return "private function called";
    }

    function testPrivateFunc() public pure returns (string memory) {
        return privateFunc();
    }

    // Internal function can be called
    // - inside this contract
    // - inside contracts that inherit this contract
    function internalFunc() internal pure returns (string memory) {
        return "internal function called";
    }

    function testInternalFunc() public pure virtual returns (string memory)
{
        return internalFunc();
    }

    // Public functions can be called
    // - inside this contract
```

```
    // - inside contracts that inherit this contract
    // - by other contracts and accounts
    function publicFunc() public pure returns (string memory) {
        return "public function called";
    }

    // External functions can only be called
    // - by other contracts and accounts
    function externalFunc() external pure returns (string memory) {
        return "external function called";
    }

    // This function will not compile since we're trying to call
    // an external function here.
    // function testExternalFunc() public pure returns (string memory) {
    //     return externalFunc();
    // }

    // State variables
    string private privateVar = "my private variable";
    string internal internalVar = "my internal variable";
    string public publicVar = "my public variable";
    // State variables cannot be external so this code won't compile.
    // string external externalVar = "my external variable";
}

contract Child is Base {
    // Inherited contracts do not have access to private functions
    // and state variables.
    // function testPrivateFunc() public pure returns (string memory) {
    //     return privateFunc();
    // }

    // Internal function call be called inside child contracts.
    function testInternalFunc() public pure override returns (string
memory) {
        return internalFunc();
    }
```

```
    function testInternalVar() public view returns (string memory, string
memory) {
        return (internalVar, publicVar);
    }
}
```

Assignment 10

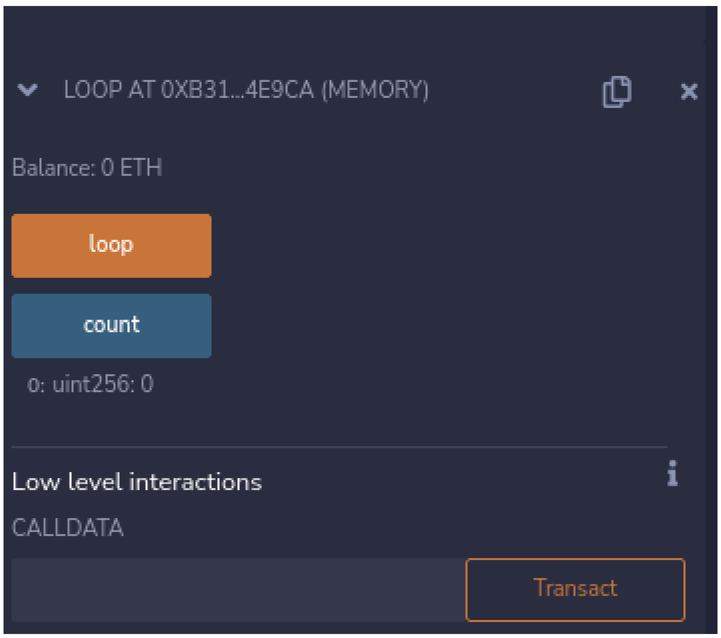
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract IfElse {
    function foo(uint x) public pure returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

    function ternary(uint _x) public pure returns (uint) {
        // if (_x < 10) {
        //     return 1;
        // }
        // return 2;

        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }

    function evenCheck(uint _num) public pure returns(bool){
        if(_num % 2 == 0){
            return true;
        }else{
            return false;
        }
    }
```

```
}
```

Assignment 11

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Loop {
   uint public count;
   function loop() public{
       // for loop
       for (uint i = 0; i < 10; i++) {
           if (i == 5) {
               // Skip to next iteration with continue
               continue;
           }
           if (i == 5) {
               // Exit loop with break
               break;
           }
           count++;
       }

       // while loop
       uint j;
```

```
        while (j < 10) {
            j++;
        }
    }
}
```



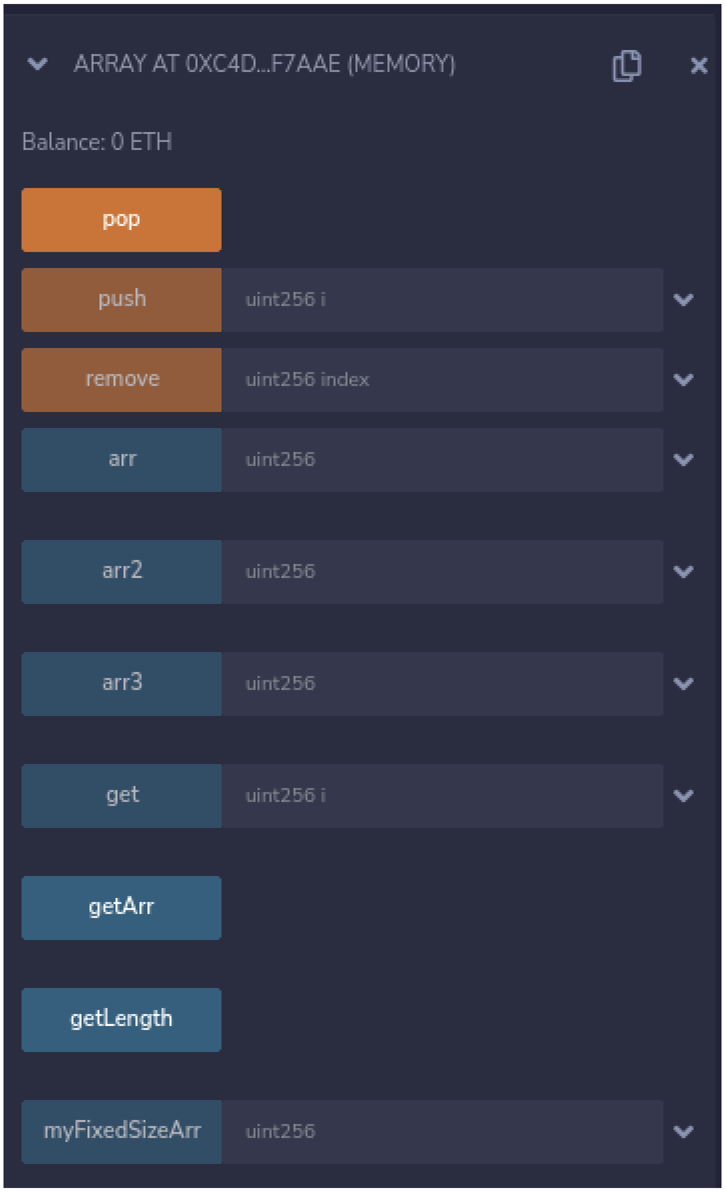
Assignment 12

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Array {
    // Several ways to initialize an array
    uint[] public arr;
    uint[] public arr2 = [1, 2, 3];
```

```
uint[3] public arr3 = [0, 1, 2];
// Fixed sized array, all elements initialize to 0
uint[10] public myFixedSizeArr;

function get(uint i) public view returns (uint) {
    return arr[i];
}

// Solidity can return the entire array.
// But this function should be avoided for
// arrays that can grow indefinitely in length.
function getArr() public view returns (uint[3] memory) {
    return arr3;
}

function push(uint i) public {
    // Append to array
    // This will increase the array length by 1.
    arr.push(i);
}

function pop() public {
    // Remove last element from array
    // This will decrease the array length by 1
    arr.pop();
```

```
    }

    function getLength() public view returns (uint) {
        return arr.length;
    }

    function remove(uint index) public {
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case 0
        delete arr[index];
    }
}

contract CompactArray {
    uint[] public arr;

    // Deleting an element creates a gap in the array.
    // One trick to keep the array compact is to
    // move the last element into the place to delete.
    function remove(uint index) public {
        // Move the last element into the place to delete
        arr[index] = arr[arr.length - 1];
        // Remove the last element
        arr.pop();
```

```
    }

    function test() public {
        arr.push(1);
        arr.push(2);
        arr.push(3);
        arr.push(4);
        // [1, 2, 3, 4]

        remove(1);
        // [1, 4, 3]

        remove(2);
        // [1, 4]
    }
}
```

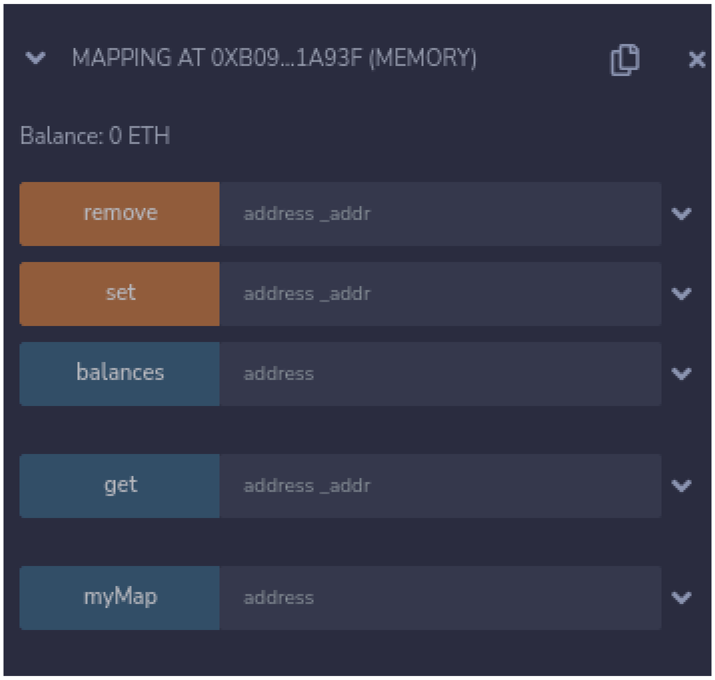


Assignment 13

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;


contract Mapping {

    // Mapping from address to uint

    mapping(address => uint) public myMap;
```

```
    mapping(address => uint) public balances;


    function get(address _addr) public view returns (uint) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return balances[_addr];
    }


    function set(address _addr) public {
        // Update the value at this address
        balances[_addr] = _addr.balance;
    }


    function remove(address _addr) public {
        // Reset the value to the default value.
        delete balances[_addr];
    }
}


contract NestedMapping {
    // Nested mapping (mapping from address to another mapping)
    mapping(address => mapping(uint => bool)) public nested;


    function get(address _addr1, uint _i) public view returns (bool) {
        // You can get values from a nested mapping
```

```
        // even when it is not initialized
        return nested[_addr1][_i];
    }

    function set(
        address _addr1,
        uint _i,
        bool _boo
    ) public {
        nested[_addr1][_i] = _boo;
    }

    function remove(address _addr1, uint _i) public {
        delete nested[_addr1][_i];
    }
}
```

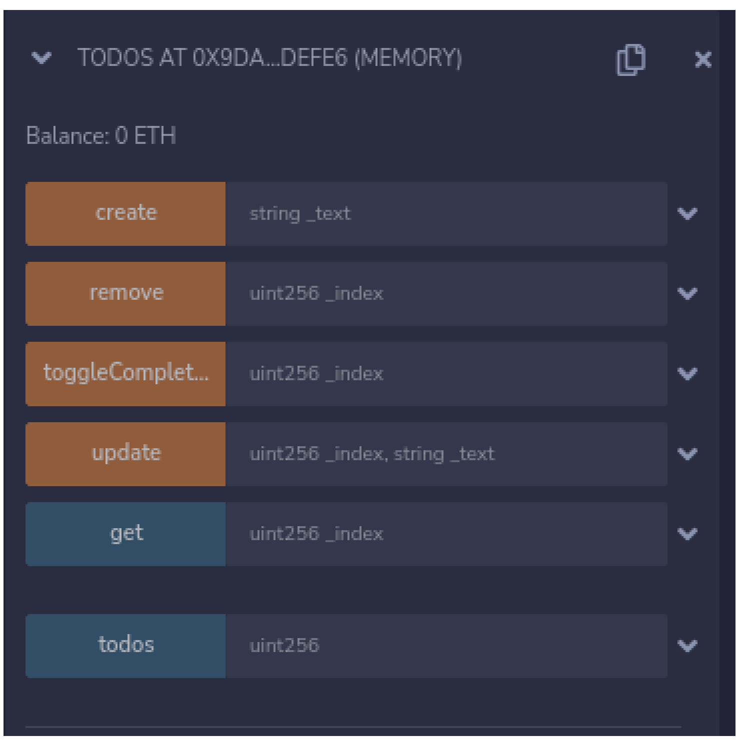


Assignment 14

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;


contract Todos {
   struct Todo {
       string text;
       bool completed;
   }


   // An array of 'Todo' structs
   Todo[] public todos;


   function create(string memory _text) public {
```

```
        // 3 ways to initialize a struct

        // - calling it like a function

        todos.push(Todo(_text, false));


        // key value mapping

        todos.push(Todo({text: _text, completed: false}));


        // initialize an empty struct and then update it

        Todo memory todo;

        todo.text = _text;

        // todo.completed initialized to false


        todos.push(todo);

    }


    // Solidity automatically created a getter for 'todos' so

    // you don't actually need this function.

    function get(uint _index) public view returns (string memory text, bool
completed) {

        Todo storage todo = todos[_index];

        return (todo.text, todo.completed);

    }


    // update text

    function update(uint _index, string memory _text) public {
```

```
        Todo storage todo = todos[_index];
        todo.text = _text;
    }

    function remove(uint _index) public {
        delete todos[_index];
    }

    // update completed
    function toggleCompleted(uint _index) public {
        Todo storage todo = todos[_index];
        todo.completed = !todo.completed;
    }
}
```

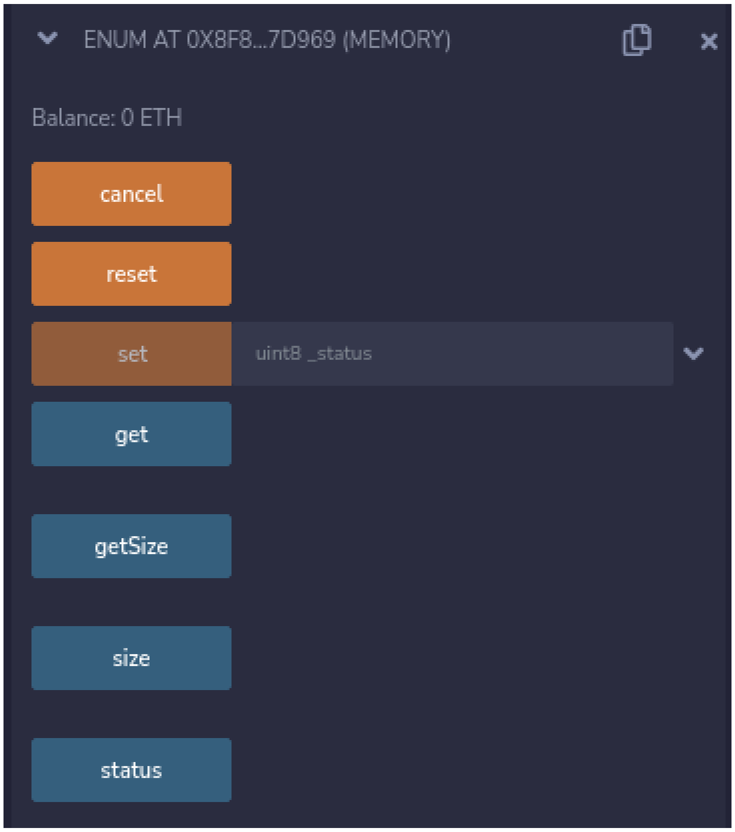


Assignment 15

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Enum {
   // Enum representing shipping status
   enum Status {
       Pending,
       Shipped,
       Accepted,
       Rejected,
       Canceled
   }
   enum Size {
   S,
   M,
   L
   }

   // Default value is the first element listed in
   // definition of the type, in this case "Pending"
   Status public status;
   Size public size;
   // Returns uint
   // Pending  - 0
```

```
// Shipped  - 1

// Accepted - 2

// Rejected - 3

// Canceled - 4

function get() public view returns (Status) {

    return status;

}


function getSize() public view returns (Size) {

    return size;

}

// Update status by passing uint into input

function set(Status _status) public {

    status = _status;

}


// You can update to a specific enum like this

function cancel() public {

    status = Status.Canceled;

}


// delete resets the enum to its first value, 0

function reset() public {

    delete status;

}
```

```
}
```

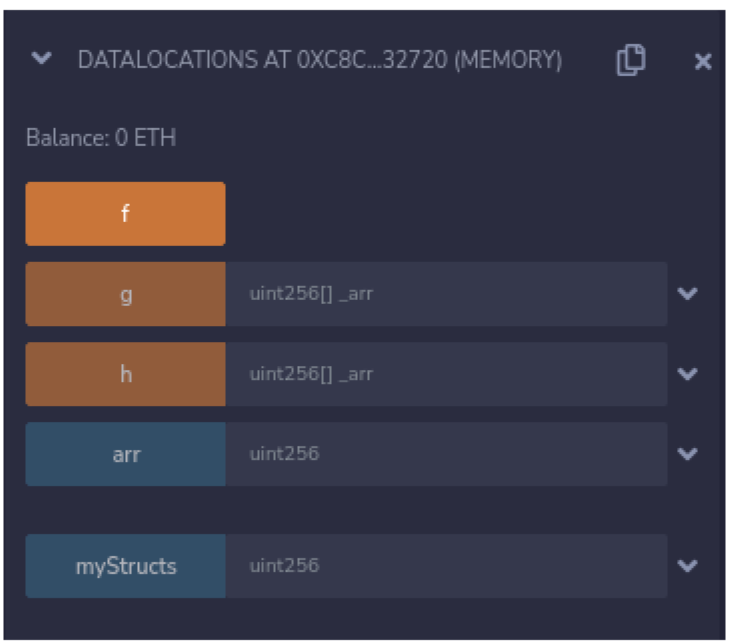


Assignment 16

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;


contract DataLocations {
   uint[] public arr;
   mapping(uint => address) map;
   struct MyStruct {
       uint foo;
   }
   mapping(uint => MyStruct) public myStructs;
```

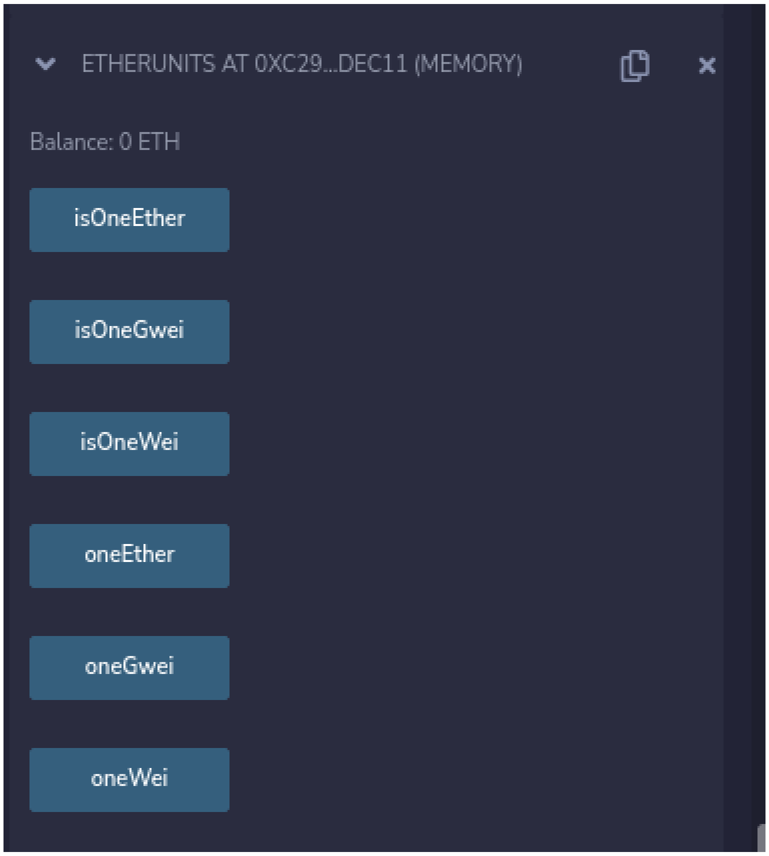
```
function f() public returns (MyStruct memory, MyStruct memory, MyStruct memory){
    // call _f with state variables
    _f(arr, map, myStructs[1]);
    // get a struct from a mapping
    MyStruct storage myStruct = myStructs[1];
    myStruct.foo = 4;
    // create a struct in memory
    MyStruct memory myMemStruct = MyStruct(0);
    MyStruct memory myMemStruct2 = myMemStruct;
    myMemStruct2.foo = 1;

    MyStruct memory myMemStruct3 = myStruct;
    myMemStruct3.foo = 3;
    return (myStruct, myMemStruct2, myMemStruct3);
}

function _f(
    uint[] storage _arr,
    mapping(uint => address) storage _map,
    MyStruct storage _myStruct
) internal {
    // do something with storage variables
}
```

```
    // You can return memory variables
    function g(uint[] memory _arr) public returns (uint[] memory) {
        // do something with memory array
        _arr[0] = 1;
    }

    function h(uint[] calldata _arr) external {
        // do something with calldata array
        // _arr[0] = 1;
    }
}
```



Assignment 17

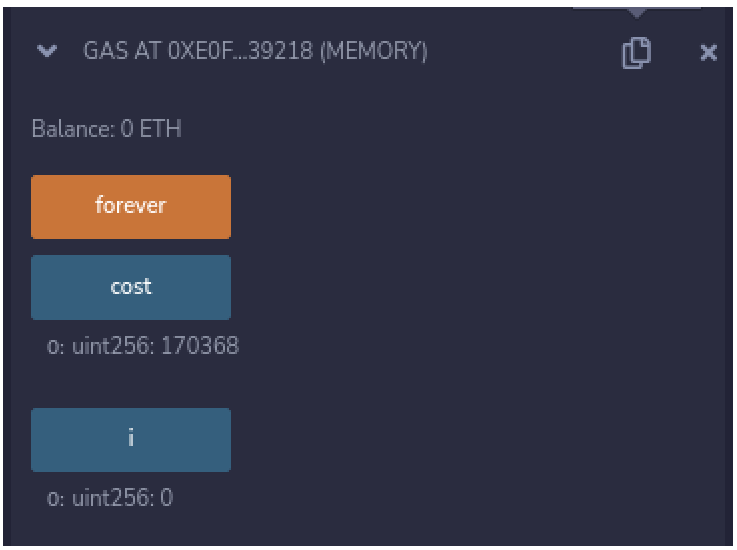
```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;


contract EtherUnits {

   uint public oneWei = 1 wei;

   // 1 wei is equal to 1

   bool public isOneWei = 1 wei == 1;

   uint public oneGwei = 1 gwei;

   bool public isOneGwei = 1 ether == 1e9;

   uint public oneEther = 1 ether;

   // 1 ether is equal to 10^18 wei

   bool public isOneEther = 1 ether == 1e18;

}
```



Assignment 18

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Gas {
   uint public i = 0;
   uint public cost= 170367;

   // Using up all of the gas that you send causes your transaction to
fail.
   // State changes are undone.
```

```
    // Gas spent are not refunded.

    function forever() public {

        // Here we run a loop until all of the gas are spent

        // and the transaction fails

        while (true) {

            i += 1;

        }

    }

}
```



Assignment 19

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;


contract z {

    /*

    Which function is called, fallback() or receive()?
```

```
          send Ether
              |
        msg.data is empty?
             / \
           yes  no
           /     \
receive() exists?  fallback()
        /   \
       yes   no
       /      \
   receive()   fallback()
   */

   // Function to receive Ether. msg.data must be empty
   receive() external payable {}

   // Fallback function is called when msg.data is not empty
   fallback() external payable {}

   function getBalance() public view returns (uint) {
       return address(this).balance;
   }
}
```

```
contract SendEther {
   function sendViaTransfer(address payable _to) public payable {
       // This function is no longer recommended for sending Ether.
       _to.transfer(msg.value);
   }

   function sendViaSend(address payable _to) public payable {
       // Send returns a boolean value indicating success or failure.
       // This function is not recommended for sending Ether.
       bool sent = _to.send(msg.value);
       require(sent, "Failed to send Ether");
   }

   function sendViaCall(address payable _to) public payable {
       // Call returns a boolean value indicating success or failure.
       // This is the current recommended method to use.
       (bool sent, bytes memory data) = _to.call{value: msg.value}("");
       require(sent, "Failed to send Ether");
   }
}

contract Charity {
   address public owner;

   constructor() {
```

```
        owner = msg.sender;

    }


    function donate() public payable {}


    function withdraw() public {

        uint amount = address(this).balance;


        (bool sent, bytes memory data) = owner.call{value: amount}("");

        require(sent, "Failed to send Ether");

    }

}
```

Conclusion: Solidity programming has been implemented on Remix IDE.