Blockchain Laboratory
Lab Manual-203105409

Department of Information technology

Parul Institute of Engineering and Technology

Faculty of Engineering and Technology

Parul University

Session 2025-26

Published by:
Department of Information technology
Faculty of Engineering and Technology
Parul University

- **Laboratory Manual is for Internal Circulation only**

EXPERIMENT NO. 1 - Demonstrate Blockchain characteristics [DIT].

Objective(s):

To familiarize students with the fundamental characteristics of blockchain technology, including decentralization, immutability, transparency, and consensus, by using blockchain simulation tools and practical demonstrations.

Outcome:

Students will be able to Explain and demonstrate the core characteristics of blockchain technology.

Problem Statement:

- Show how decentralization works in a peer-to-peer setup.
- Illustrate immutability by recording transactions that cannot be altered.

Background Study:

1. Blockchain Technology
2. Key Characteristics of Blockchain
3. Applications of Blockchain

Theory :

1. Decentralization

- Data is not stored on a single server or location.
- Maintained by a **distributed network** of nodes.
- Eliminates the need for central authority (like banks or governments).

2. Immutability

- Once data is written to the blockchain, **it cannot be changed** or deleted.
- Ensures **data integrity** and **prevents tampering**.

3. Transparency

- All transactions are **visible to all participants** (in public blockchains like Ethereum and Bitcoin).
- Enhances **trust** and **accountability**.

4. Distributed Ledger Technology (DLT)

- Every participant has access to the **same copy** of the ledger.
- Updates are reflected across all nodes simultaneously.

5. Security

- Uses **cryptographic algorithms** (like SHA-256) to secure data.
- Each block is linked to the previous one, making it extremely hard to alter.

Question Bank:

1. What are the core characteristics of blockchain technology?
2. How does decentralization improve security and reliability?
3. Explain the concept of immutability in blockchain with an example.
4. What is a consensus mechanism? Give an example.
5. How does transparency benefit blockchain applications in the real world?
6. What tools can be used to simulate blockchain characteristics?

EXPERIMENT NO. 2 - Introduction to ETHEREUM tools and Solidity.

Objective(s):

To introduce students to Ethereum blockchain development tools and the Solidity programming language for creating and deploying smart contracts.

Outcome:

Students will be able to Understand the Ethereum platform and its architecture. Write, compile, and deploy smart contracts using Solidity.

Problem Statement:

Develop and deploy a simple smart contract using Solidity on the Ethereum blockchain using Remix IDE.
The contract should:

- Accept and store a user's name and age.
- Allow updating and retrieving stored values.
- Be deployed and tested using a local or test Ethereum network.

Background Study:

- Ethereum Overview
- Smart Contracts
- Solidity Programming Language
- Ethereum Tools

Theory :

- **Ethereum :-** Ethereum is an open-source, decentralized blockchain platform that enables developers to build and deploy smart contracts and decentralized applications.
  - Launched in 2015 by **Vitalik Buterin**.
  - It supports its own cryptocurrency called **Ether (ETH)**.
  - Unlike Bitcoin (only used for payments), Ethereum is programmable, allowing complex applications to be built on top of its blockchain.

- **Solidity :-** Solidity is a high-level programming language used to write smart contracts on the Ethereum blockchain.
  - Similar to JavaScript in syntax.
  - Used to define rules, logic, and data handling within Ethereum contracts.
  - Compiles to **EVM (Ethereum Virtual Machine)** bytecode to run on the blockchain.

- **Smart Contract :-** A smart contract is a self-executing contract with the agreement terms written directly in code.
  - Stored and run on the **Ethereum blockchain**.
  - Once deployed, it **automatically executes** when certain conditions are met.
  - **Immutable** (cannot be changed) and **trustless** (no need for a third party).

Solidity Tools :

1. **Remix IDE :** Write, compile, test, and deploy smart contracts in the browser.
2. **Hardhat :** Local development, testing, and deployment of smart contracts.
3. **Truffle Suite :** Build, compile, test, and deploy Solidity contracts.
4. **Ganache :** Run a personal blockchain for development & testing.
5. **MetaMask :** Interact with dApps and smart contracts from the browser.

Application :

1. **Decentralized Finance (DeFi) :** Solidity powers financial applications without intermediaries like banks.
2. **Non-Fungible Tokens (NFTs) :** Solidity is used to create unique digital assets.
3. **Real Estate and Legal Contracts :** Solidity automates ownership**,** lease agreements, and property transfers.
4. **Voting Systems :** Solidity enables transparent and tamper**-**proof elections.
5. **Supply Chain Management :** Track products through each step of the supply chain.

Question Bank:

1. What is Ethereum and how is it different from Bitcoin?
2. What are smart contracts? How do they work in Ethereum?
3. What is Solidity? Mention its key features.
4. Explain the purpose of Remix IDE in Ethereum development.
5. What is Meta Mask? How does it help in deploying contracts?
6. What are the common data types used in Solidity?
7. Explain the use of `memory` and `storage` keywords in Solidity.
8. What are events and modifiers in Solidity?

EXPERIMENT NO. 3 - Deploy a smart contract for printing "Hello World" using Java Script VM, Injected Web3 and Web3Provider using Metamask and Ganache.

Objective(s):

To introduce students to smart contract deployment using different environments in Remix IDE and Ethereum development tools like MetaMask and Ganache.

Outcome:

Students will be able to:

- Write and deploy a basic smart contract on the Ethereum blockchain.
- Understand the use of JavaScript VM, Injected Web3 (MetaMask), and Web3 Provider (Ganache).
- Interact with deployed contracts using a connected wallet or local blockchain.

Problem Statement:

Write and deploy a simple smart contract that prints "Hello World" using Remix IDE in three different environments: JavaScript VM, Injected Web3 (MetaMask), and Web3 Provider (Ganache).

Background Study:

- Remix IDE:
- MetaMask:
- Ganache:
- Smart Contracts:

Ethereum and Solidity:

Ethereum:

- Ethereum is an open-source, blockchain-based platform that enables developers to build anddeploy decentralized applications (dApps). It extends the idea of Bitcoin's blockchai (which is mainly for digital money) by adding a programmable layer where you can write smart con tracts — self-executing contracts with the terms directly written in code.

Key Points About Ethereum:

- Blockchain: Like Bitcoin, Ethereum uses a blockchain — a public, distributed ledgerthat records transactions in secutamper-proof way.
- Smart Contracts: Ethereum introduced smart contracts, which are programs thatautomatically execute when certain conditions are met. These contracts run exactly as programmed without any possibility of downtime or censorship.
- Ether (ETH): Ethereum has its own cryptocurrency called Ether. ETH is used to pay for transaction fees and computational services on the Ethereum network.
- Decentralized Applications (dApps): Developers can build apps that run onEthereum's blockchain rather than on centralized servers. These dApps are resistantto censorship and downtime.
- Ethereum Virtual Machine (EVM): A runtime environment that executes smart contracts. It allows code written in languages like Solidity to run on the blockchain.

Solidity:

- Solidity is a high-level, object-oriented programming language specificallydesigned for writing smart contracts on the Ethereum blockchain.

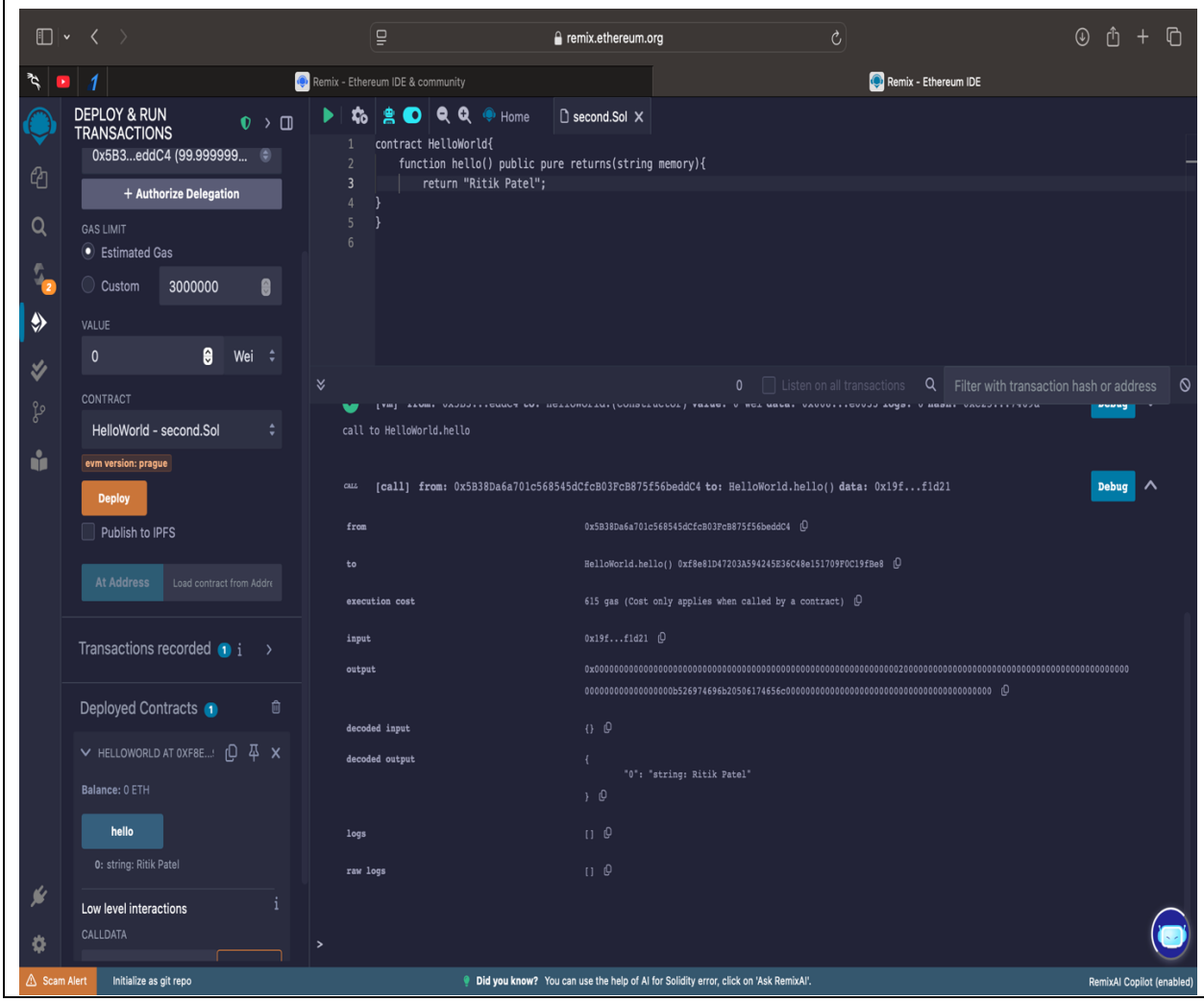
What is Smart Contract:

- A smart contract is a self-executing piece of code that runs on the blockchain andautomatically enforces and executes the terms of an agreement without humanintervention.
- For example: A Solidity smart contract could automatically transfercryptocurrency between two users after certain conditions are met (like escrow,voting, auctions, etc). Why Solidity is Important for Ethereum?
- It defines the logic and rules of decentralized applications(DApps). It is used to build DeFi apps, NFT projects, DAOs, and more.
- Contracts written in Solidity are compiled into Ethereum Virtual Machine (EVM) bytecode, which is then deployed on the Ethereum network.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity^0.8.0;

contract HelloWorld{
    function hello() public pure returns(stringmemory){
        return "Ritik Patel";
  }
}
```

Output (Student Work Area):



Question Bank:

1. What is the role of MetaMask in Ethereum development?
2. What is the difference between JavaScript VM, Injected Web3, and Web3 Provider?
3. Why do we use Ganache in smart contract development?
4. Write the Solidity code to return “Hello World” using a function.
5. What are the advantages of deploying on a local blockchain before the mainnet?
6. Explain the gas fee observed when deploying a smart contract with MetaMask.
7. What is the significance of the `pure` keyword in Solidity?

EXPERIMENT NO. 4 - Deploy a smart contract for arithmetic operations using Java Script VM, Injected Web3 and Web3Provider using Metamask and Ganache.

Objective(s):

To enable students to write and deploy a smart contract that performs basic arithmetic operations using three different Ethereum environments in Remix IDE—**JavaScript VM**, **Injected Web3 (MetaMask)**, and **Web3 Provider (Ganache)**.

Outcome:

Students will be able to:

- Write Solidity smart contracts for performing arithmetic operations.
- Deploy smart contracts using different Ethereum environments.
- Interact with deployed contracts using MetaMask and Ganache.

Problem Statement:

Create and deploy a smart contract that performs arithmetic operations such as addition, subtraction, multiplication, and division. Deploy it using:

1. JavaScript VM
2. Injected Web3 (MetaMask)
3. Web3 Provider (Ganache)

The contract should:

- Accept two numbers.
- Perform the four arithmetic operations.
- Return the results using view functions.

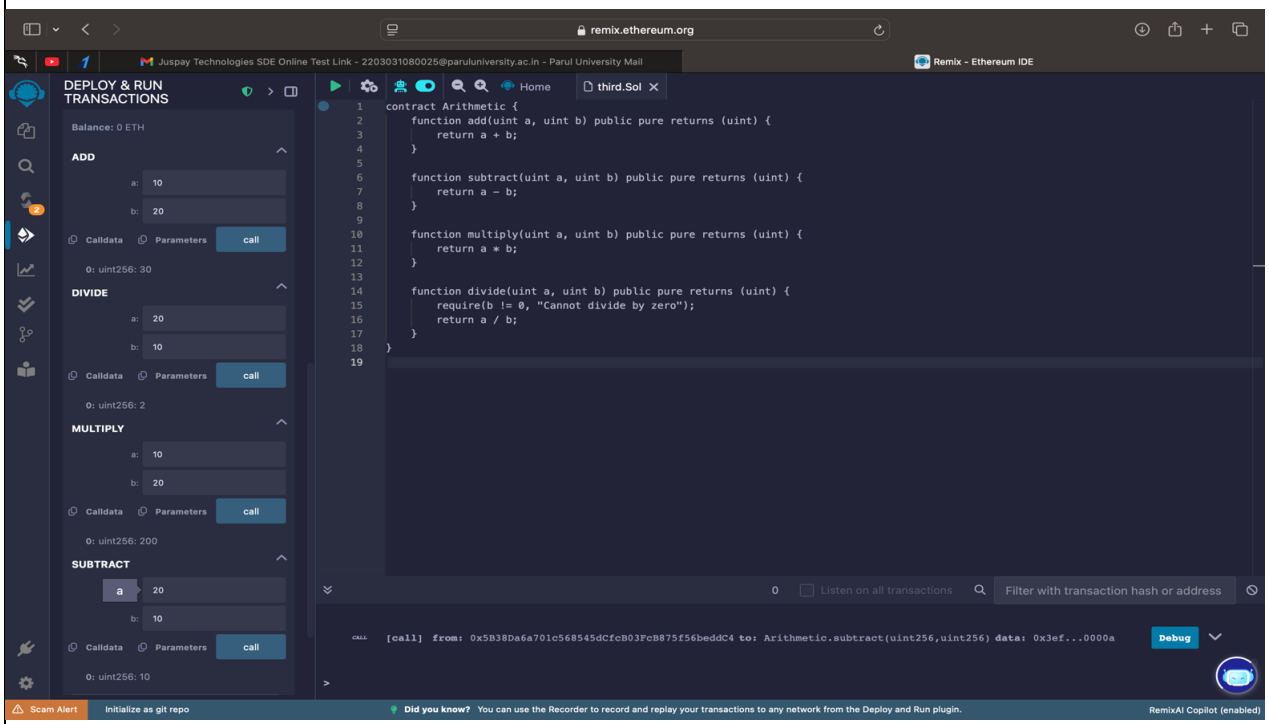
Background Study:

- Ethereum & Smart Contracts:
- Solidity:
- Deployment Environments:
- MetaMask & Ganache:

Code (Student Work Area):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Arithmetic {
    function add(uint a, uint b) public pure returns (uint) {
        return a + b;
    }

    function subtract(uint a, uint b) public pure returns (uint) {
        return a - b;
    }

    function multiply(uint a, uint b) public pure returns (uint) {
        return a * b;
    }

    function divide(uint a, uint b) public pure returns (uint) {
        require(b != 0, "Cannot divide by zero");
        return a / b;
    }
}
```

Output(Student Work Area):-



Question Bank:

1. How does JavaScript VM differ from Injected Web3 and Web3 Provider?
2. Explain the purpose of MetaMask in smart contract deployment.
3. What is Ganache, and how is it useful in local Ethereum development?
4. Why is the require keyword used in the division function?
5. How are gas fees calculated during contract deployment?
6. What are the advantages of deploying on a testnet before deploying on the mainnet?
7. Describe the steps to interact with a deployed smart contract on Remix.
8. What are pure functions in Solidity, and why are they used here?

EXPERIMENT NO. 5- Deploy a smart contract for FINDING LARGEST NUMBER OUT OF THREE NUMBERS using Java Script VM, Injected Web3 and Web3Provider using Metamask and Ganache.

Objective(s):

To develop and deploy a Solidity smart contract that finds the largest of three numbers using Remix IDE and deploy it using: JavaScript VM (in-browser blockchain), Injected Web3 (MetaMask), Web3 Provider (Ganache local blockchain).

Outcome:

Students will be able to:

- Write a Solidity smart contract to compare numerical values.
- Deploy smart contracts using different Ethereum environments.
- Understand how to interact with smart contracts through wallets like MetaMask and local tools like Ganache.

Problem Statement:

Write a smart contract in Solidity that takes three unsigned integers and returns the largest one. Deploy and test the contract using:

- JavaScript VM (local simulated blockchain)
- Injected Web3 (via MetaMask)
- Web3 Provider (via Ganache)

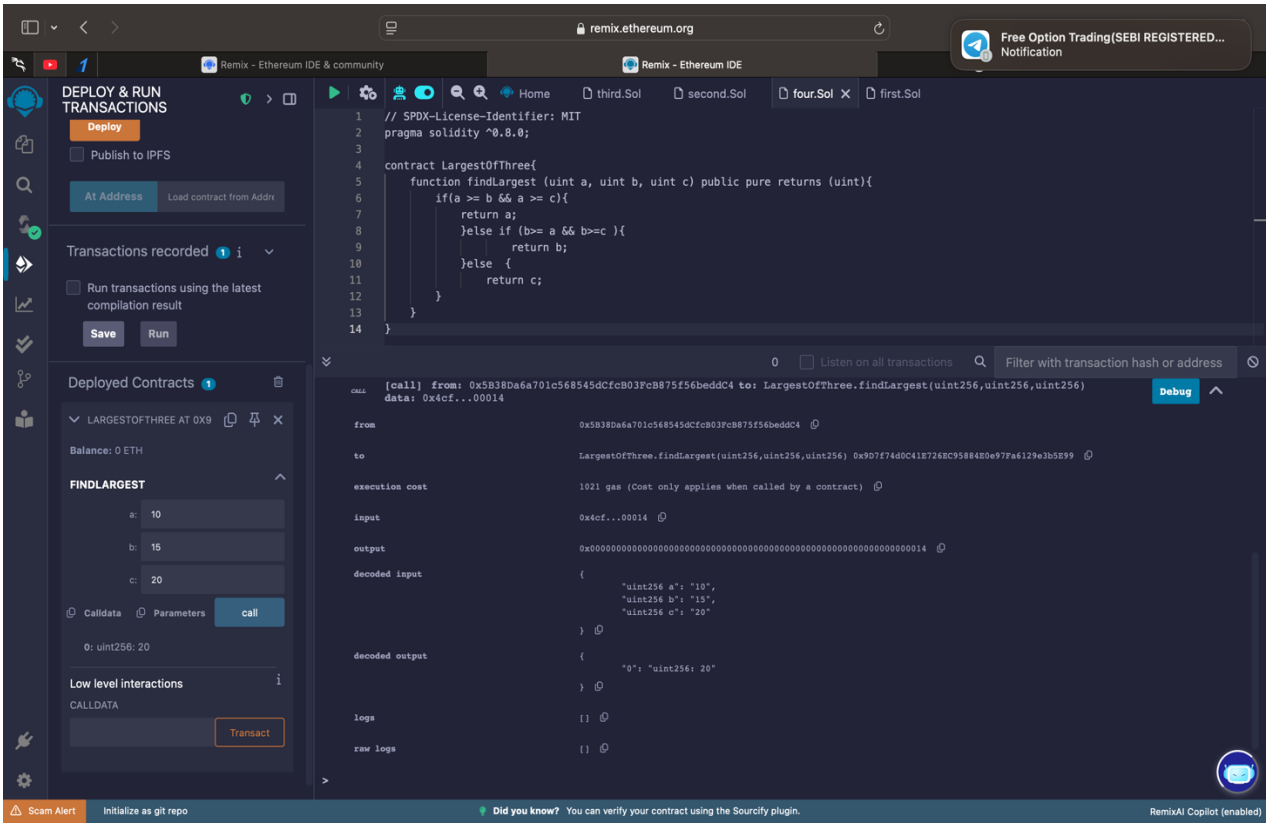
Background Study:

1. Solidity Basics.
2. Smart Contract Deployment Environments:
3. MetaMask
4. Ganache:

Code (Student Work Area):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LargestOfThree{
   function findLargest (uint a, uint b, uint c) public pure returns (uint){
      if(a >= b && a >= c){
         return a;
         }else if (b>= a && b>=c ){
               return b;
         }else  {
            return c;
      }
   }
}
```

Output (Student Work Area):



Question Bank:

1. What is the purpose of using multiple deployment environments?
2. How does the findLargest() function determine the maximum number?
3. What are the advantages of using MetaMask with Remix IDE?
4. How is local testing with Ganache different from testnet deployment?
5. Why do we use pure in the function declaration?
6. Can this contract be modified to return the position/index of the largest number? How?
7. What are potential gas costs in each deployment environment?
8. How does MetaMask sign and approve a transaction?

EXPERIMENT NO. 6 - Create a Smart Contract for a banking application in solidity which allows users to do the following: Mint money into your account Withdraw money from your account Send money from your account to smart contract address Check balance
After a contract is created, deploy the contract on Ethereum Testnet network.

Objective(s):

 To develop a Solidity smart contract that allows users to mint (deposit), withdraw, send money to the contract, and check their balance.

 To deploy the contract on different Ethereum environments: JavaScript VM, Injected Web3 (MetaMask), and Web3 Provider (Ganache).

Outcome:

Students will be able to:

- Write and understand Solidity smart contracts for basic banking operations.
- Deploy smart contracts using Remix IDE on multiple Ethereum environments.
- Use MetaMask and Ganache to test and interact with deployed contracts.
- Perform deposit, withdrawal, and internal transfers within the smart contract.

Problem Statement:

Create and deploy a smart contract that simulates a simple banking system allowing users to:

- Mint money (deposit ETH) into their account.
- Withdraw money from their account.
- Send money to the smart contract's balance.
- Check their account balance.

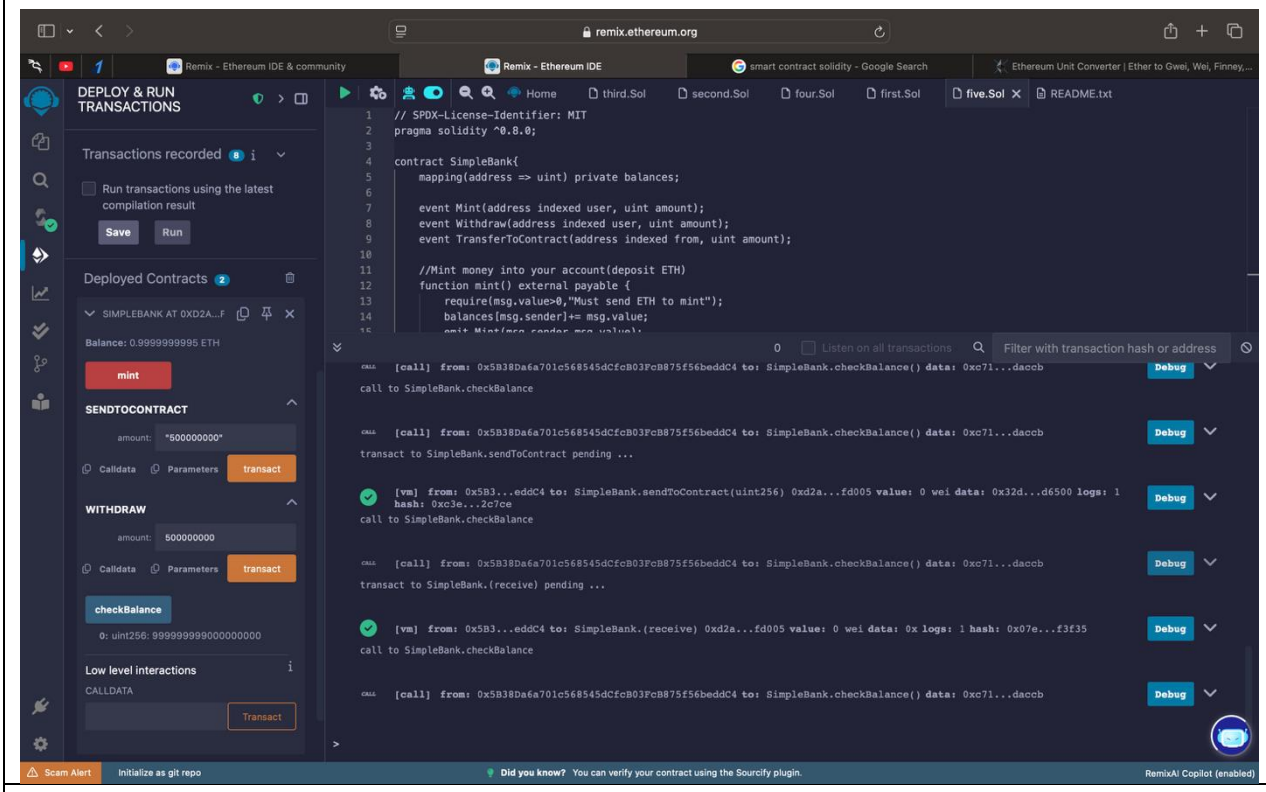
Background Study:

1. Smart Contracts & Solidity:
2. Ethereum Wallets & Tools:
3. Deploying & Interacting with Contracts:
4. Handling ETH in Contracts:

Code (Student Work Area):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleBank{
    mapping(address => uint) private balances;

    event Mint(address indexed user, uint amount);
    event Withdraw(address indexed user, uint amount);
    event TransferToContract(address indexed from, uint amount);

    //Mint money into your account(deposit ETH)
    function mint() external payable {
        require(msg.value>0,"Must send ETH to mint");
        balances[msg.sender]+= msg.value;
        emit Mint(msg.sender,msg.value);
    }

    //Withdraw money from your account
    function withdraw(uint amount) external {
        require(balances[msg.sender]>=amount,"Insufficient balance");
        balances[msg.sender]-=amount;
        payable(msg.sender).transfer(amount);
        emit Withdraw(msg.sender,amount);
    }

    //Send money from your account to contract balance
    function sendToContract(uint amount) external {
        require(balances[msg.sender]>=amount,"Insufficient balance");
        balances[msg.sender]-=amount;
        emit TransferToContract(msg.sender,amount);
    }

    //Check caller's balance
    function checkBalance() external view returns(uint) {
        return balances[msg.sender];
    }

    //FallBack function to receive ETH sent directly to contract
    receive() external payable {
        balances[msg.sender]+=msg.value;
        emit Mint(msg.sender, msg.value);
    }
}
```

Output (Student Work Area):



Question Bank:

1. What is the purpose of the mint() function in this contract?
2. How does the contract track user balances internally?
3. Explain the use of the receive() function in Solidity.
4. What happens if a user tries to withdraw more ETH than their balance?
5. How does the sendToContract() function differ from a direct ETH transfer?
6. Describe the role of events (Mint, Withdraw, TransferToContract) in this contract.
7. What precautions are needed to secure contracts handling ETH?
8. How does deployment on a testnet using MetaMask differ from local Ganache deployment?

EXPERIMENT NO. 7- How to build a smart contract that lets user book rooms and pay for them with cryptocurrency.

Objective(s):

- To create a Solidity smart contract that allows users to book rooms by paying with ETH.
- To implement room availability tracking and payment management.
- To deploy and test the contract on Ethereum testnets using Remix and MetaMask.

Outcome:

- Students will be able to:

- Write Solidity contracts for handling bookings and payments.
- Manage state such as room availability and user bookings.
- Accept and handle ETH payments securely.
- Deploy and interact with the contract on Ethereum test networks.

Problem Statement:

- Design and deploy a smart contract that:

- Keeps track of rooms and their availability.
- Allows users to book a room by sending ETH equal to the room price.
- Records bookings associated with user addresses.
- Prevents double booking of the same room for overlapping time periods.
- Allows the contract owner to withdraw collected funds.

Background Study

- Ethereum and Smart Contracts: Using Solidity for contract logic.
- Payable Functions: Accepting ETH payments with `payable` keyword.
- State Management: Tracking room statuses and user bookings.
- Security Considerations: Handling reentrancy and proper payment transfers.
- Events: Logging booking activities for transparency.

Code (Student Work Area):

Output (Student Work Area):

Question Bank:

1. How does the contract check room availability?
2. What is the significance of the `payable` keyword?
3. How can overlapping bookings be prevented?
4. How does the contract owner withdraw collected funds safely?
5. What improvements can be made for variable pricing or multiple rooms?
6. How can date/time be handled more precisely in smart contracts?
7. What are the security risks of handling ETH payments in contracts?

EXPERIMENT NO. 8 - Deploy a smart contract using MyEtherWallet (MEW).

Objective(s):

• To understand how to deploy a smart contract on the Ethereum blockchain using **MyEtherWallet (MEW)**. And to interact with smart contracts using MEW's contract interface.

Outcome:

- Students will be able to:

- Compile a smart contract and extract its bytecode and ABI.
- Deploy the contract using MEW via a wallet like MetaMask or a hardware wallet.
- Interact with the deployed contract from MEW's interface.

Problem Statement:

Deploy a basic smart contract (e.g., "Hello World" or simple storage) on the Ethereum blockchain using MyEtherWallet. The contract should:

- Store a value on-chain.
- Retrieve the stored value.
  Deployment must be done manually by pasting the **compiled bytecode** and **ABI** into MEW.

Background Study:

- MyEtherWallet (MEW):
- Solidity and Smart Contract Compilation:
- Ethereum Wallets:
- Ethereum Gas and Transactions:

Code (Student Work Area):

Output (Student Work Area):

Question Bank:

1. What is the role of ABI and bytecode in smart contract deployment?
2. How does MEW differ from Remix in terms of contract deployment?
3. What are the security considerations while deploying contracts from MEW?
4. How can you interact with deployed contracts via MEW?
5. What is the significance of gas and gas limits during deployment?
6. Why is MetaMask commonly used in conjunction with MEW?

EXPERIMENT NO. 9 - Deploy the smart contract for RAFFLE DRAW GAME.

Objective(s):

- To design and deploy a smart contract for a decentralized raffle game.
- To enable users to enter the raffle by paying ETH.
- To randomly select a winner from the list of participants.

Outcome:

- Students will be able to:

- Write a Solidity smart contract implementing basic randomness (pseudo-random).
- Manage participants and prize logic in a decentralized manner.
- Deploy and interact with the raffle contract using JavaScript VM, Injected Web3 (MetaMask), and Ganache.

Problem Statement:

- Design and deploy a Raffle Draw smart contract where:

- Participants enter the raffle by paying a fixed amount of ETH.
- The contract collects ETH from participants.
- Only the owner can pick a winner.
- The winner receives all the ETH collected in the contract.
- Ensure the contract allows multiple participants and resets after each draw.

Background Study:

- Smart Contracts in Solidity
- Handling ETH Transfers
- Randomness in Smart Contracts
- Access Control
- Event Logging

Code (Student Work Area):

Output (Student Work Area):

Question Bank:

1. How does the smart contract ensure only paid entries are accepted?
2. Explain how the winner is selected in the `pickWinner()` function.
3. Why is this randomness approach not secure for production?
4. What happens after a winner is selected?
5. How could you improve the randomness in this contract?
6. How would you modify the contract to allow multiple winners?
7. What are the benefits and risks of decentralized raffle games?

EXPERIMENT NO. 10 - Deploy Smart Contract for E-Voting.

Objective(s):

- To implement a secure, transparent, and decentralized E-Voting system using Solidity.
- To allow only registered voters to vote and ensure one vote per person.
- To deploy and interact with the contract on Ethereum test networks using Remix and MetaMask.

Outcome:

- Students will be able to:

- Design and deploy a smart contract for E-Voting.
- Understand voter registration, candidate management, and vote casting logic.
- Handle access control and data integrity in smart contracts.
- Deploy and interact with the contract using JavaScript VM, Injected Web3, and Web3Provider with MetaMask and Ganache.

Problem Statement:

- Create a smart contract for an E-Voting system where:

- The election organizer (owner) registers candidates and voters.
- Registered voters can vote once for a registered candidate.
- After voting ends, the owner can announce the winner.
- The system prevents double voting and unauthorized access.

Background Study:

- Smart Contracts with Solidity
- Access Control with `onlyOwner`
- Data Structures in Solidity
- Events

Code (Student Work Area):

Output (Student Work Area):

Question Bank:

1. How does the contract ensure one person, one vote?
2. What role does `onlyOwner` play in this contract?
3. How are candidates and voters stored and accessed?
4. What improvements can be made to secure the voting process further?
5. Can this voting system be made anonymous?
6. Why is randomness not used in this contract?
7. What are the limitations of using pseudo-randomness in smart contracts?