

**Name of the Experiment : Hello Solidity – Writing First Smart Contract**

**Objective/Aim :**

**The aim is to create and deploy a basic Solidity smart contract that demonstrates how data can be stored and retrieved on the Ethereum blockchain.**

**Apparatus/Software Used**:

** MetaMask Wallet (Testnet enabled)**

** Remix IDE**

**Theory/Concept:**

* **Solidity is a high-level, contract-oriented language for programming smart contracts to execute on the Ethereum Virtual Machine (EVM). A smart contract is an automated program on the blockchain that executes once specific conditions are fulfilled.**
* **A decentralized platform, Ethereum enables one to create smart contracts that can deal with digital assets, perform logic, and facilitate decentralized applications (DApps).**
* **The most basic Solidity program is a "Hello World" type of contract, which generally saves and loads a message or a value to/from the blockchain. This illustrates how blockchain saves data transparently anSolidity is a high-level, contract-oriented programming language used for writing smart contracts that run on the Ethereum Virtual Machine (EVM). A smart contract is a self-executing program stored on the blockchain that executes automatically when certain conditions are met.**
* **Ethereum provides a decentralized environment where smart contracts can be created to manage digital assets, execute logic, and enable decentralized applications (DApps).**
* **The simplest example of a Solidity program is a "Hello World" style contract, which usually stores and retrieves a message or a value from the blockchain. This demonstrates how blockchain stores data in a transparent and immutable way.**

**Procedure:**

Deploying Your First Solidity Smart Contract on Remix IDE

1. Compilation Stage
   * Remix automatically converts your Solidity code into bytecode and ABI — no extra setup needed.
   * The compiler ensures your syntax is valid and matches the chosen version.
   * If there’s even a small syntax error or version mismatch, the build process halts immediately.
2. Connecting for Deployment
   * MetaMask serves as the gateway between Remix and the blockchain network (testnet or mainnet).
   * When you click “Deploy,” MetaMask will request your approval, displaying:  
     • The wallet address initiating the deployment  
     • Estimated gas consumption and cost  
     • The network you’re connected to
   * This is where you realize — deploying a contract means broadcasting a blockchain transaction, not just “saving” it somewhere.
3. Transaction Execution
   * Once confirmed, you receive a transaction hash, which can be tracked on a blockchain explorer (e.g., Etherscan or Sepolia Explorer).
   * The network processes it, and your contract receives a permanent blockchain address.
   * A gas fee is charged from your MetaMask wallet — even on testnets, where costs are simulated.
4. Using the Deployed Contract
   * Remix provides a ready-to-use interface for interacting with your contract functions.
   * Any state-changing action will prompt another MetaMask confirmation.
   * You can observe variable updates, function results, and emitted events directly from the Remix interface.

**Steps to Create and Deploy Your First Solidity Contract in Remix**

**Step 1: Launch Remix IDE**

* **Open your browser and go to:** [**https://remix.ethereum.org**](https://remix.ethereum.org)

**Step 2: Start a New Contract File**

* **On the left panel, open the File Explorer.**
* **Navigate to the contracts folder.**
* **Click the "+" icon to create a fresh .sol file (e.g., SimpleStorage.sol).**

**Step 3: Write Your Solidity Code**

***pragma solidity ^0.8.0;***

***contract SimpleStorage {***

***uint public storedData;***

***constructor(uint initialValue) {***

***storedData = initialValue;***

***}***

***function set(uint newValue) public {***

***storedData = newValue;***

***}***

***function get() public view returns (uint) {***

***return storedData;***

***}***

***}***

**Step 4: Compile the Smart Contract**

* **Go to the Solidity Compiler tab.**
* **Choose the correct compiler version (0.8.x) and click Compile SimpleStorage.sol.**

**Step 5: Deploy Your Contract**

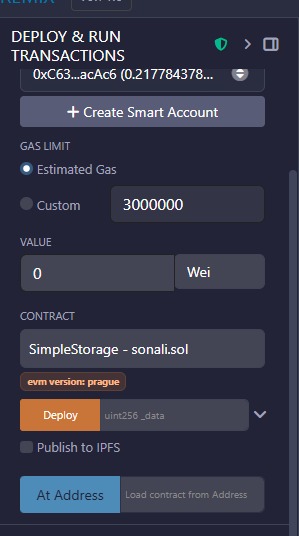
* **Click Deploy and confirm the transaction in MetaMask.**

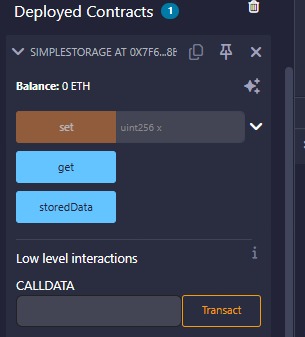
**Step 6: Work with Your Contract**

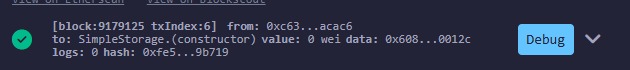
* **Once deployed, the contract will appear under Deployed Contracts in Remix.**
* **You can now:  
  • Click get() → View the stored number.  
  • Click set(200) → Update the stored number.  
  • Use storedData() → Automatically retrieve the current value.**

A screenshot of a computer program

AI-generated content may be incorrect.

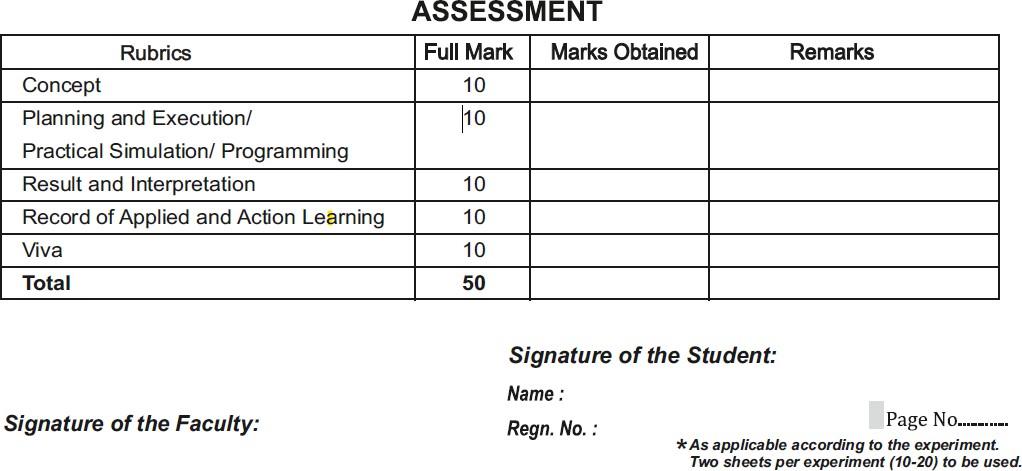






**Observation:**

* Once a smart contract is on the blockchain, its code cannot be altered — fixing errors means deploying a new version.
* The amount of gas required is influenced by how complex your constructor is and how much storage your contract uses.
* Even the most basic contracts leave a permanent entry on the blockchain and incur gas costs.
* MetaMask serves not only as a crypto wallet but also as the tool that signs and sends your blockchain transactions.

****