**Blockchain Lab Experiment 3**

**Shashwat Tripathi**

**D20A Roll No: 64**

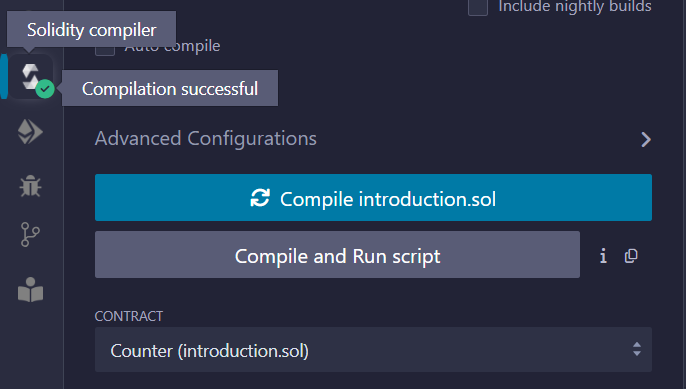
**Aim:** Study on Solidity Programming for creating Smart Contracts

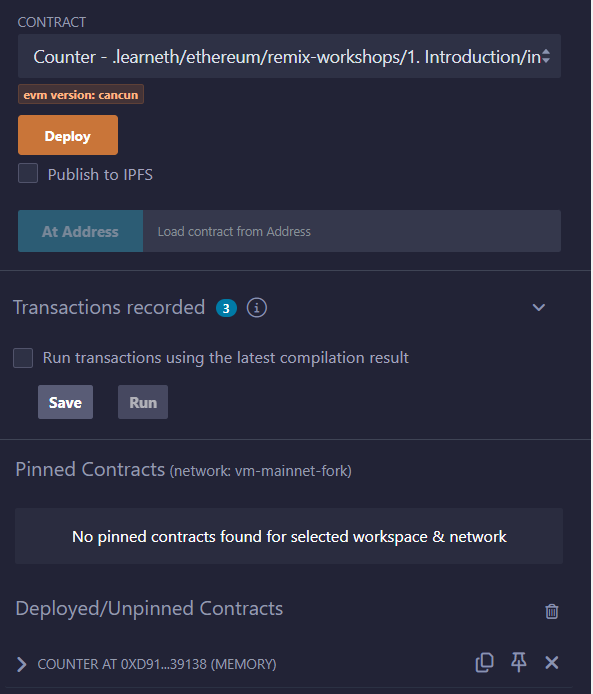
**Practical:**

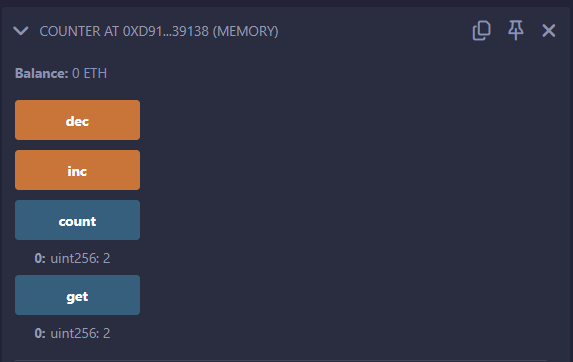
**Step - 1 : Hands on Solidity Programming Assignments for creating Smart Contracts**

**Step - 1 : Introduction**

1. Compile this contract.
2. Deploy it to the Remix VM.
3. Interact with your contract.







**Step - 2 : Basic Syntax**

// SPDX-License-Identifier: MIT

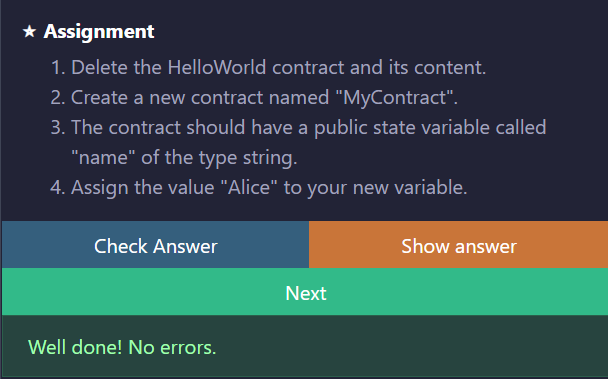
// compiler version must be greater than or equal to 0.8.3 and less than 0.9.0

pragma solidity ^0.8.3;

contract MyContract {

string public name = "Alice";

}



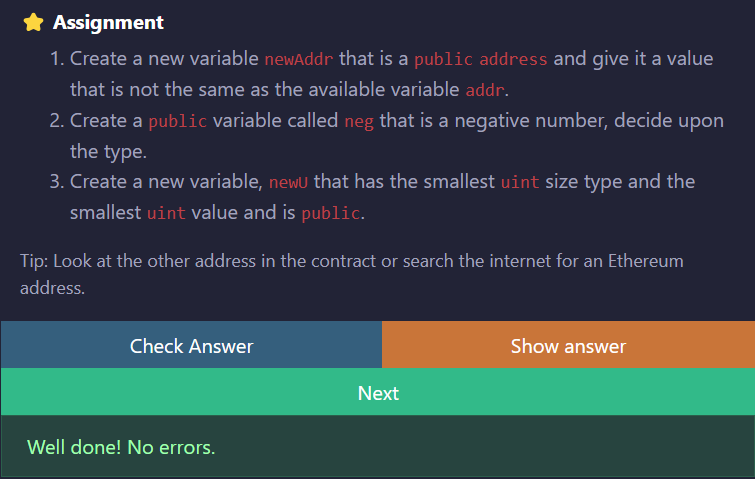
**Step - 3: Primitive Data Types**

// New values

address public newAddr = 0x11fac33125aDbA1419BE146191c84E202caa5407;

int public neg = -30;

uint8 public newU = 0;



**Step - 4 : Variables**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Variables {

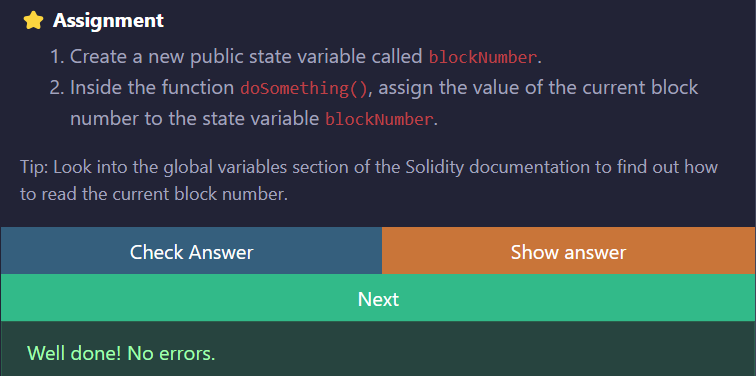
uint256 public blockNumber;

function doSomething() public {

blockNumber = block.number;

}

}



**Step - 5**

**5.1: Functions - Reading and Writing to a State Variable**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract SimpleStorage {

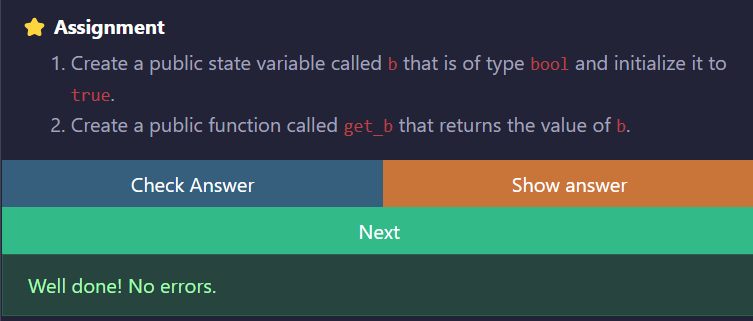
bool public b = true;

function get\_b() public returns (bool) {

return b;

}

}



**5.2 Functions - View and Pure**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract ViewAndPure {

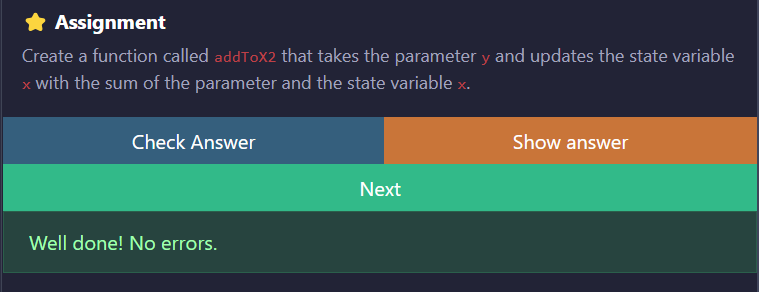
uint public x = 1;

function addToX2(uint y) public {

x = x + y;

}

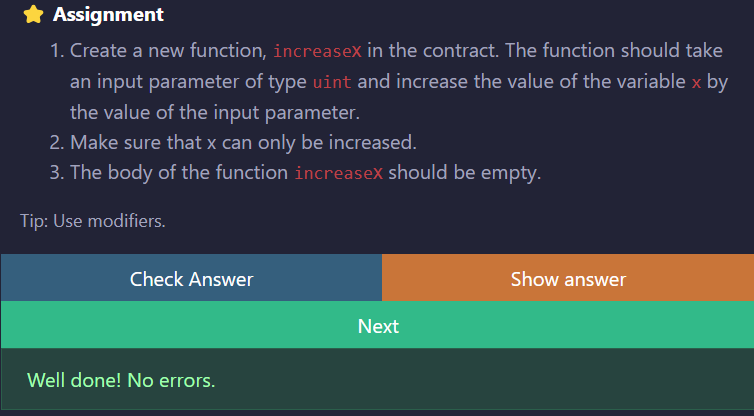
}



**5.3 Functions - Modifiers and Constructors**

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

}



**5.4 Functions - Inputs and Outputs**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Function {

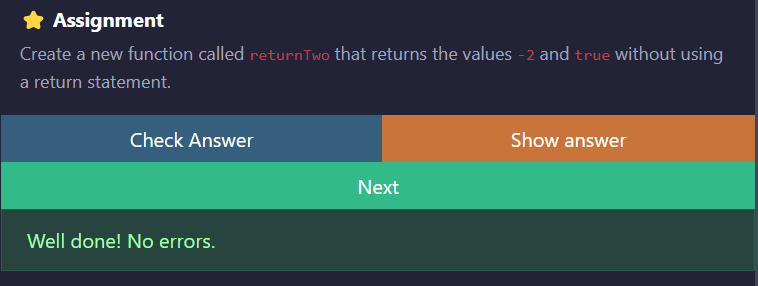
function returnTwo() public pure returns (int i, bool flag) {

i = -2;

flag = true;

}

}



**Step - 6: Visibility**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Base {

string private privateVar = "my private variable";

string internal internalVar = "my internal variable";

string public publicVar = "my public variable";

}

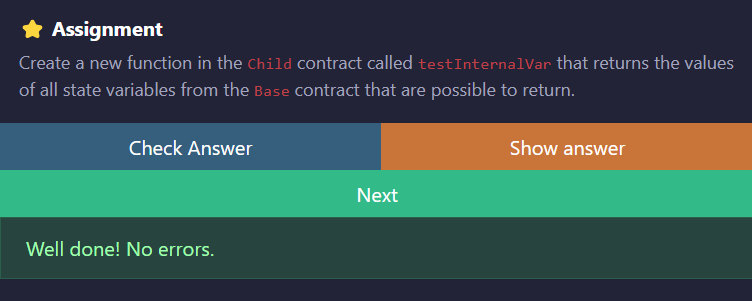
contract Child is Base {

function testInternalVar() public view returns (string memory, string memory) {

return (internalVar, publicVar);

}

}



**Step - 7:  
7.1 Control Flow - If/Else**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

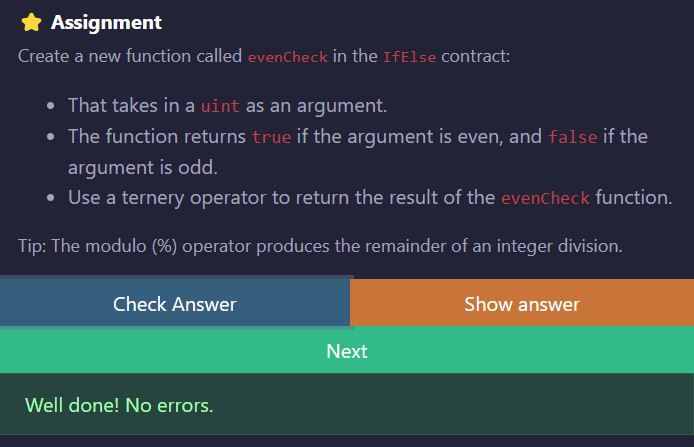
contract IfElse {

function evenCheck(uint num) public pure returns (bool) {

return (num % 2) == 0 ? true : false;

}

}



**7.2 Control Flow - Loops**

// 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;

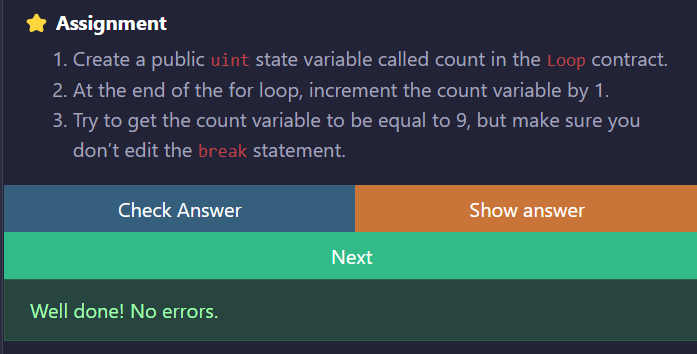
while (j < 10) {

j++;

}

}

}



**Step - 8:**

**8.1 Data Structures - Arrays**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Array {

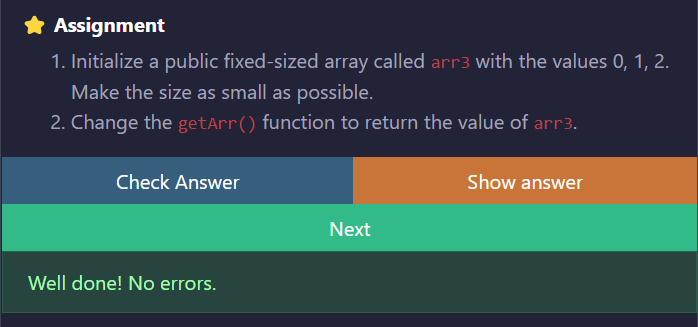
uint[3] public arr3 = [0, 1, 2];

function getArr() public view returns (uint[3] memory) {

return arr3;

}

}



**8.2 Data Structures - Mappings**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Mapping {

// Mapping from address to uint

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;

}

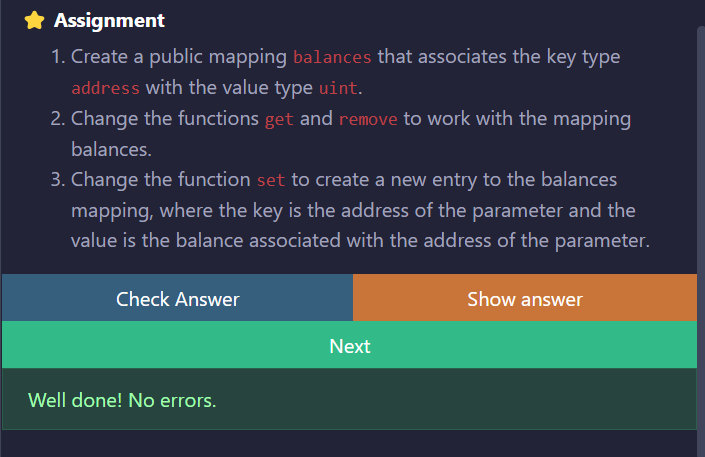
function remove(address \_addr) public {

// Reset the value to the default value.

delete balances[\_addr];

}

}

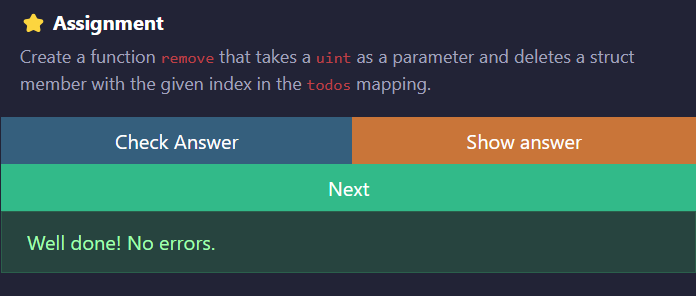


**8.3 Data Structures - Structs**

function remove(uint index) public {

delete todos[index];

}



**8.4 Data Structures - Enums**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Enum {

enum Size {

S,

M,

L

}

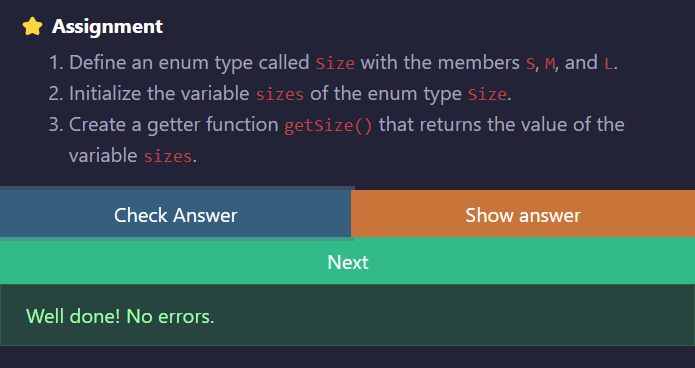
Size public sizes;

function getSize() public view returns (Size) {

return sizes;

}

}



**9. Data Locations**

// 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;

}

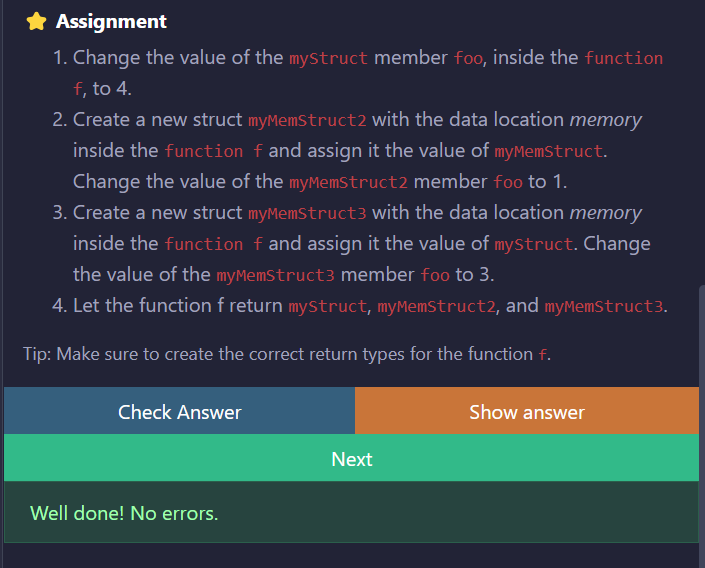
function h(uint[] calldata \_arr) external {

// do something with calldata array

// \_arr[0] = 1;

}

}



**Step - 10:**

**10.1 Transactions - Ether and Wei**

// SPDX-License-Identifier: MIT

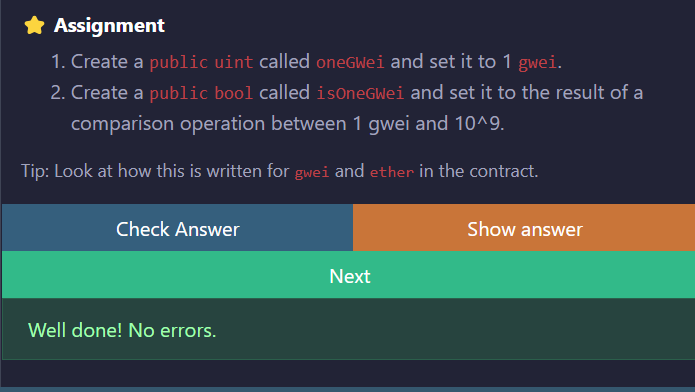
pragma solidity ^0.8.3;

contract EtherUnits {

uint public oneGwei = 1 gwei;

bool public isOneGwei = 1 gwei == 1e9;

}



**10.2 Transactions - Gas and Gas Price**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Gas {

uint public i = 0;

uint public cost = 170367;

function forever() public {

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

// and the transaction fails

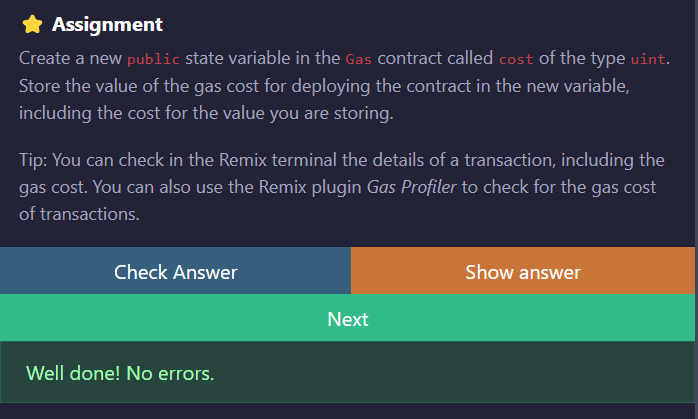
while (true) {

i += 1;

}

}

}



**10.3 Transactions - Sending Ether**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract Charity {

address public owner;

constructor() {

owner = msg.sender;

}

function donate() public payable {}

function withdraw() public {

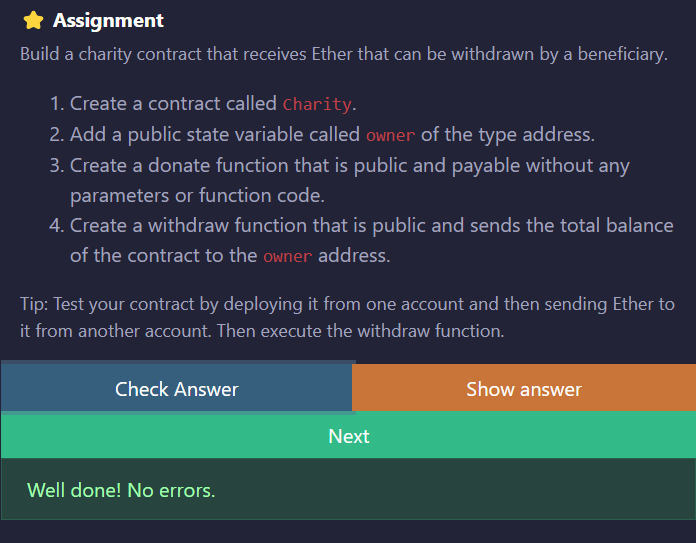
uint amount = address(this).balance;

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

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

}

}



**Step - 2 : Deploying a Voting/Ballot Smart Contract**

**Code:**

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Ballot {

struct Voter {

uint weight; // weight is accumulated by delegation

bool voted; // if true, that person already voted

address delegate; // person delegated to

uint vote; // index of the voted proposal

}

struct Proposal {

// If you can limit the length to a certain number of bytes,

// always use one of bytes1 to string because they are much cheaper

string name; // short name (up to 32 bytes)

uint voteCount; // number of accumulated votes

}

address public chairperson;

mapping(address => Voter) public voters;

Proposal[] public proposals;

constructor(string[] memory proposalNames) {

chairperson = msg.sender;

voters[chairperson].weight = 1;

for (uint i = 0; i < proposalNames.length; i++) {

// 'Proposal({...})' creates a temporary

// Proposal object and 'proposals.push(...)'

// appends it to the end of 'proposals'.

proposals.push(Proposal({

name: proposalNames[i],

voteCount: 0

}));

}

}

function giveRightToVote(address voter) public {

require(

msg.sender == chairperson,

"Only chairperson can give right to vote."

);

require(

!voters[voter].voted,

"The voter already voted."

);

require(voters[voter].weight == 0);

voters[voter].weight = 1;

}

function delegate(address to) public {

Voter storage sender = voters[msg.sender];

require(!sender.voted, "You already voted.");

require(to != msg.sender, "Self-delegation is disallowed.");

while (voters[to].delegate != address(0)) {

to = voters[to].delegate;

// We found a loop in the delegation, not allowed.

require(to != msg.sender, "Found loop in delegation.");

}

sender.voted = true;

sender.delegate = to;

Voter storage delegate\_ = voters[to];

if (delegate\_.voted) {

// If the delegate already voted,

// directly add to the number of votes

proposals[delegate\_.vote].voteCount += sender.weight;

} else {

// If the delegate did not vote yet,

// add to her weight.

delegate\_.weight += sender.weight;

}

}

function vote(uint proposal) public {

Voter storage sender = voters[msg.sender];

require(sender.weight != 0, "Has no right to vote");

require(!sender.voted, "Already voted.");

sender.voted = true;

sender.vote = proposal;

// If 'proposal' is out of the range of the array,

// this will throw automatically and revert all

// changes.

proposals[proposal].voteCount += sender.weight;

}

function winningProposal() public view

returns (uint winningProposal\_)

{

uint winningVoteCount = 0;

for (uint p = 0; p < proposals.length; p++) {

if (proposals[p].voteCount > winningVoteCount) {

winningVoteCount = proposals[p].voteCount;

winningProposal\_ = p;

}

}

}

function winnerName() public view

returns (string memory winnerName\_)

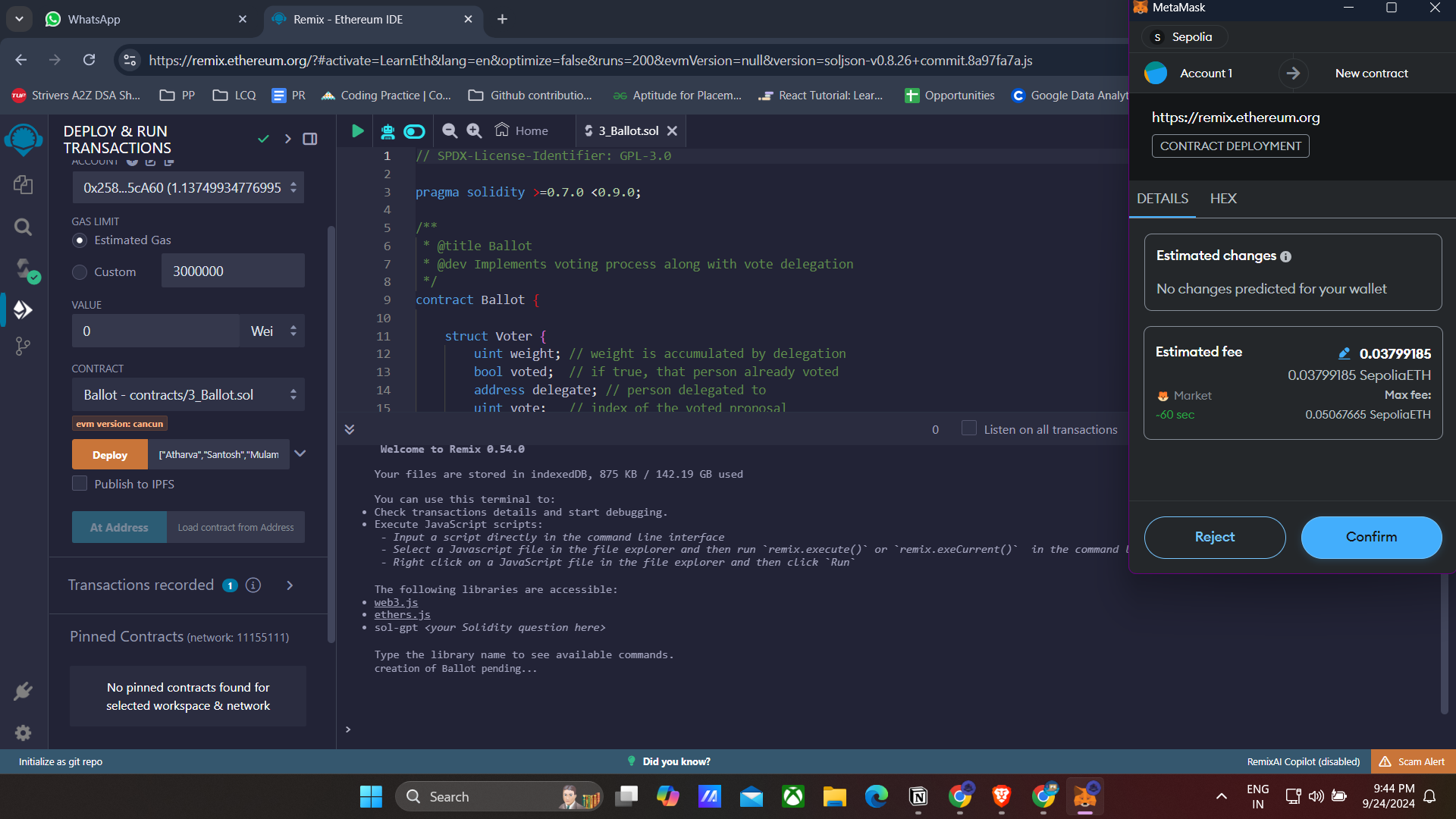
{

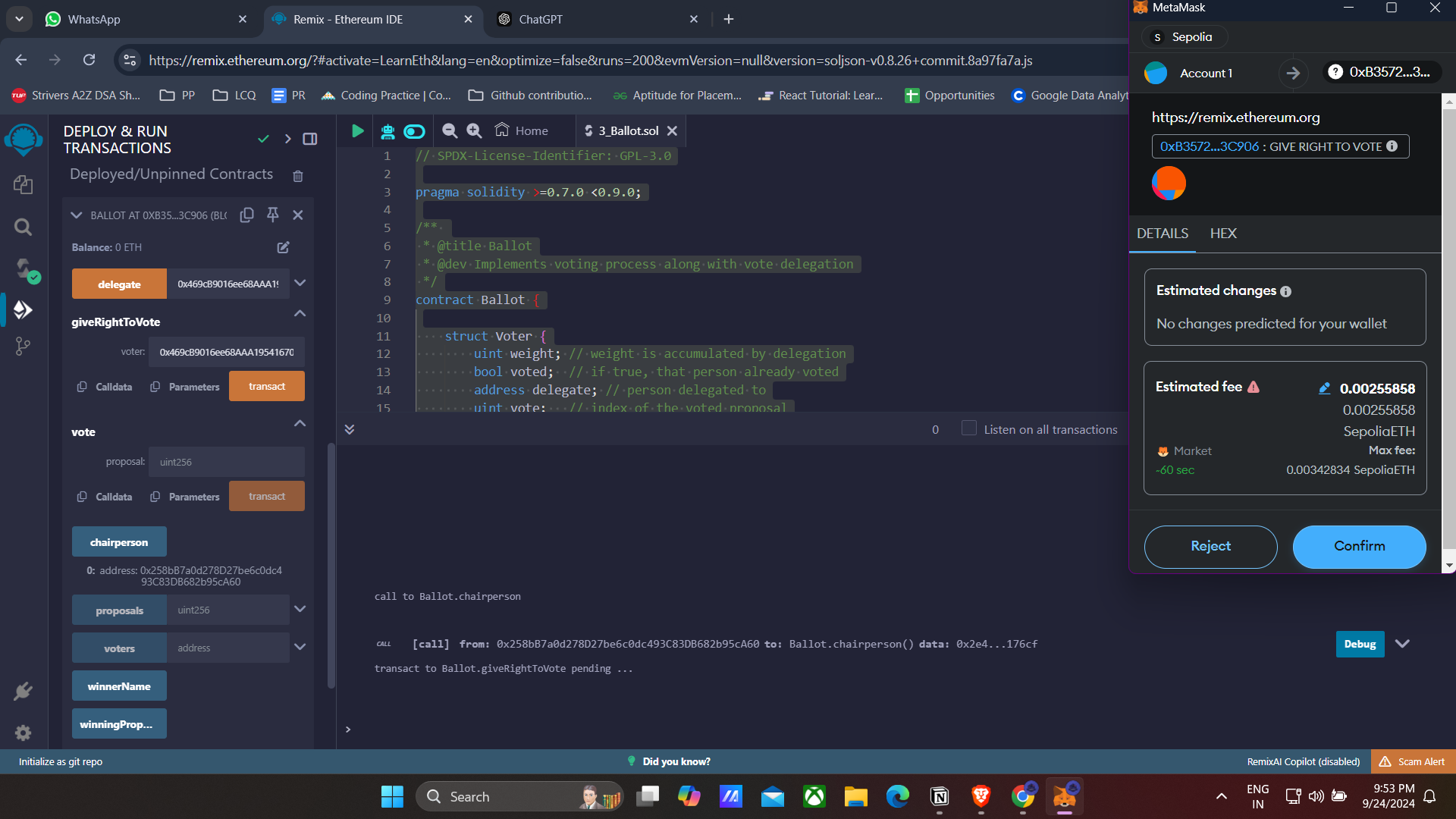
winnerName\_ = proposals[winningProposal()].name;

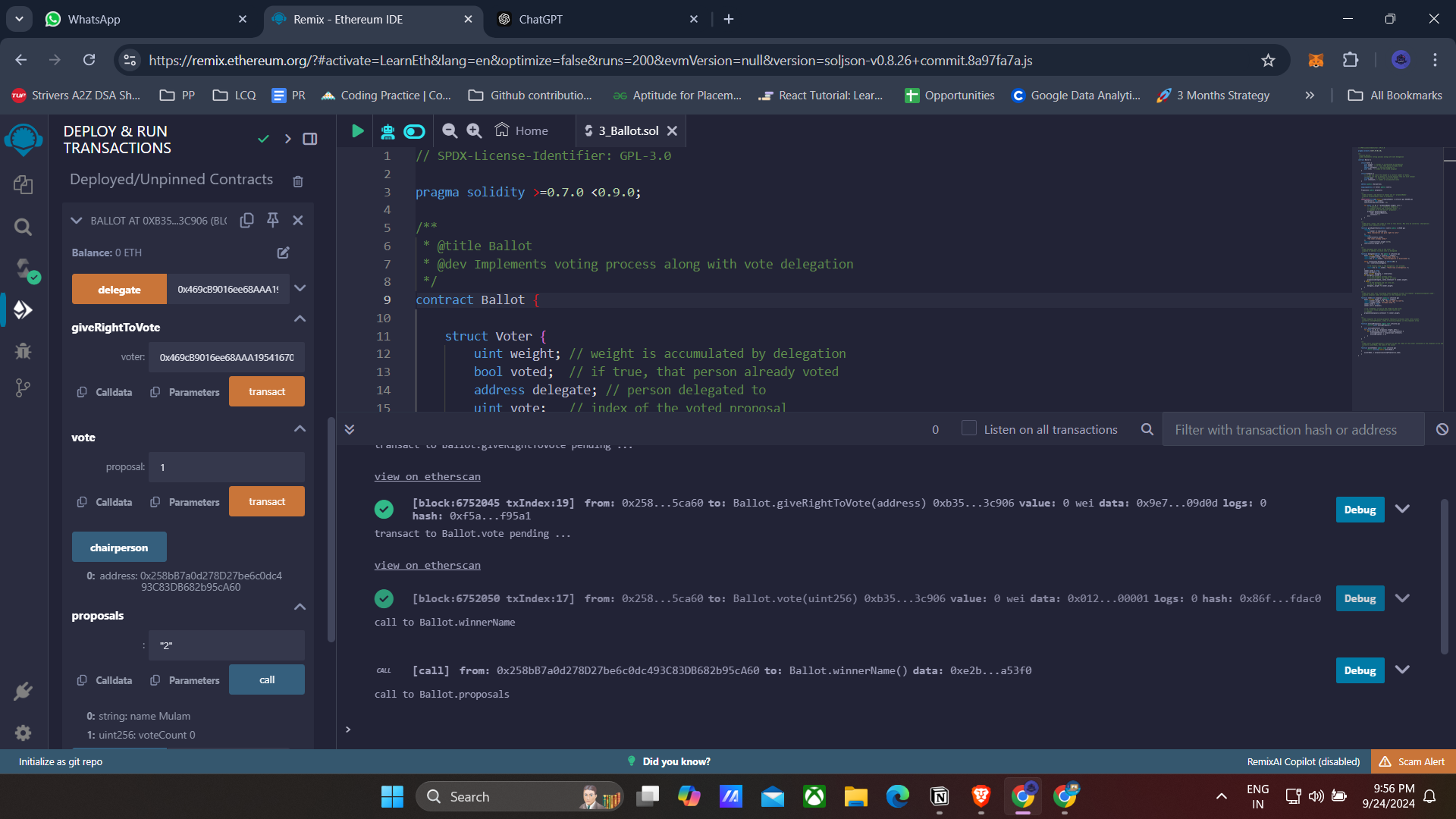
}

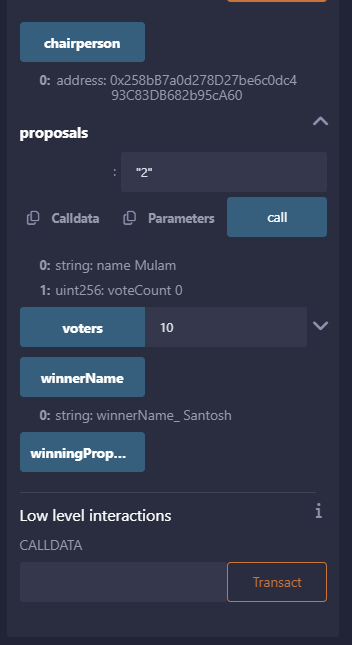
}

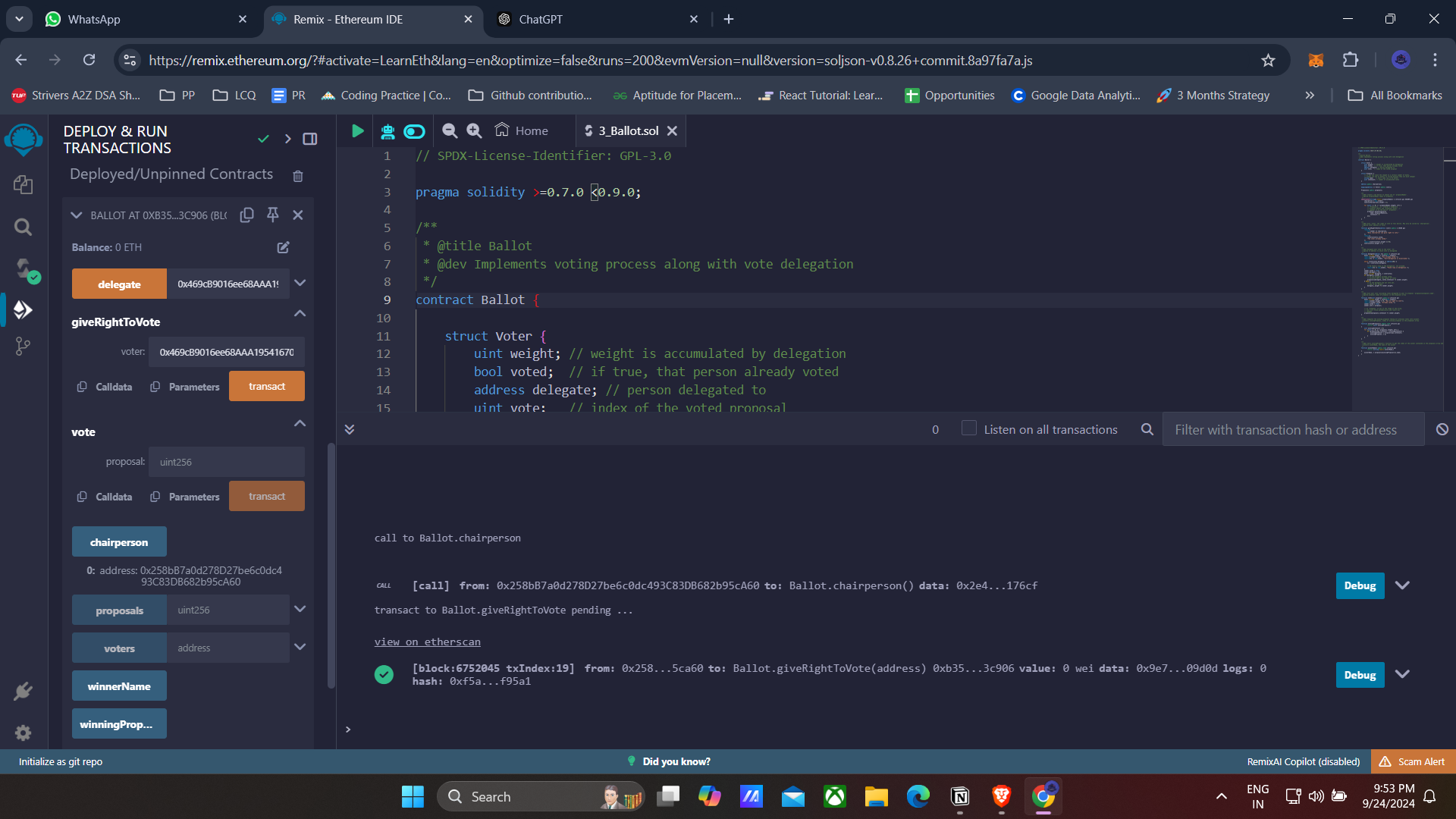
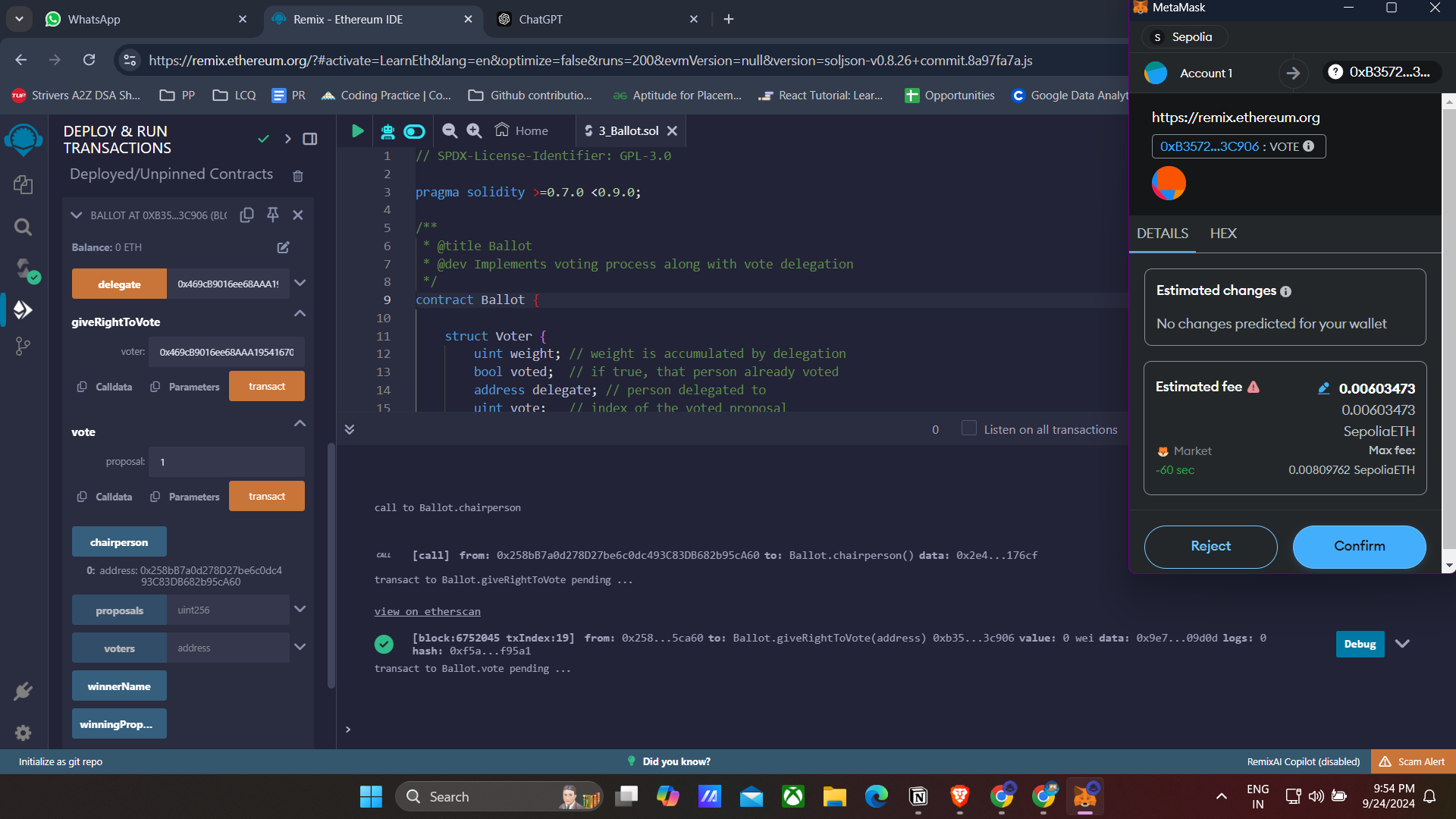
**Output:**

****

****

****

****

****

**Conclusion:** Thus, we have studied solidity programming language for smart contracts.