**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no.** | **Practicals** | **Date** | **Sign** |
| **1** | **Write the following programs for Blockchain in Python.**   1. **A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.** 2. **A transaction class to send and receive money and test it.** 3. **Create multiple transactions and display them.** 4. **Create a blockchain, a genesis block and execute it.** 5. **Create a mining function and test it.** 6. **Create a mining function and test it.** |  |  |
| **2** | **Implement and demonstrate the use of the following in Solidity.**   1. **Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables.** 2. **Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions.** |  |  |
| **3** | **Implement and demonstrate the use of the following in Solidity.**   1. **Withdrawal Pattern, Restricted Access.** 2. **Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.** |  |  |
| **4** | **Implement and demonstrate the use of the following in Solidity.**   * **Libraries, Assembly, Events, Error handling.** |  |  |
| **5** | **Write a program to demonstrate mining of Ether.** |  |  |
| **6** | **Demonstrate the running of the blockchain node.** |  |  |
| **7** | **Demonstrate the use of Bitcoin Core API.** |  |  |
| **8** | **Create your own blockchain and demonstrate its use.** |  |  |

**Practical:**

**Implement and demonstrate the use of the following in Solidity:**

**Practical A:**

**Aim: Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables**

**1. Variable:**

**Code:**

pragma solidity ^0.8.0;

contract ForVariableExample {

// Function to double a given number

function double(uint x) public pure returns (uint) {

uint result = 0;

// For loop with variable declaration for i

for (uint i = 0; i < 2; i++) {

result += x;

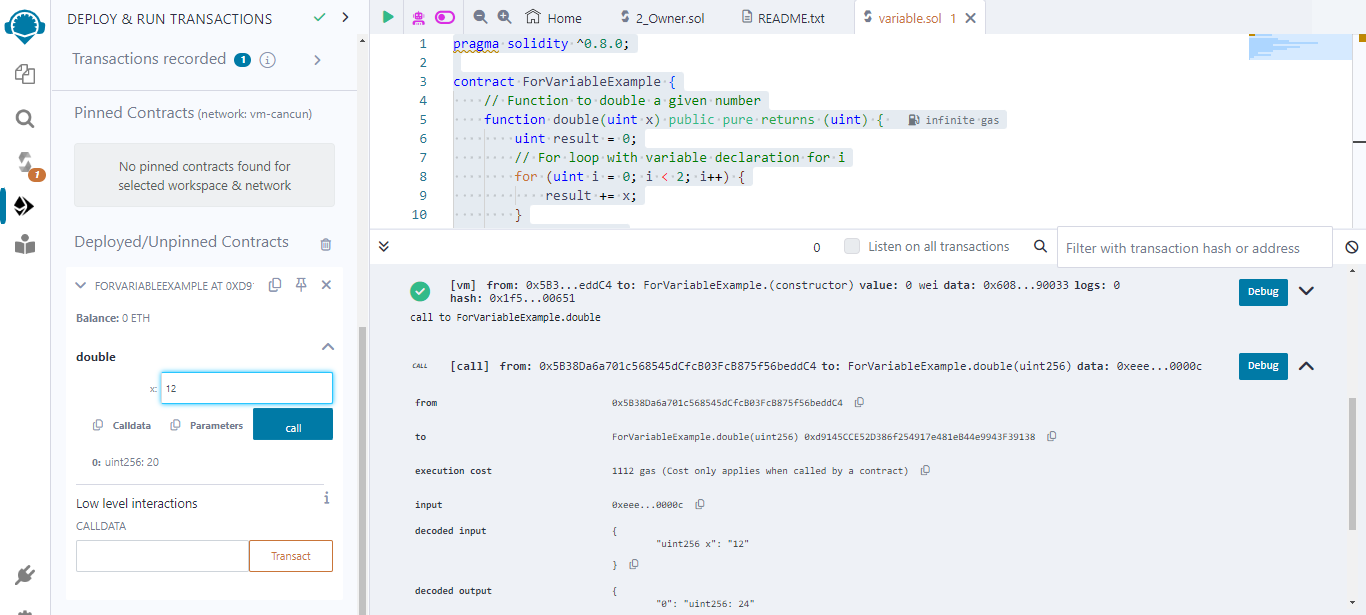
}

return result;

}

}

**Output:-**

****

**2. Operators:**

**Arithmetic Operator:**

**Code:**

pragma solidity ^0.8.0;

contract ArithmeticExample {

function basicArithmetic(uint a, uint b) public pure returns (uint sum, uint difference, uint product, uint quotient, uint remainder) {

sum = a + b; // Addition

difference = a - b; // Subtraction

product = a \* b; // Multiplication

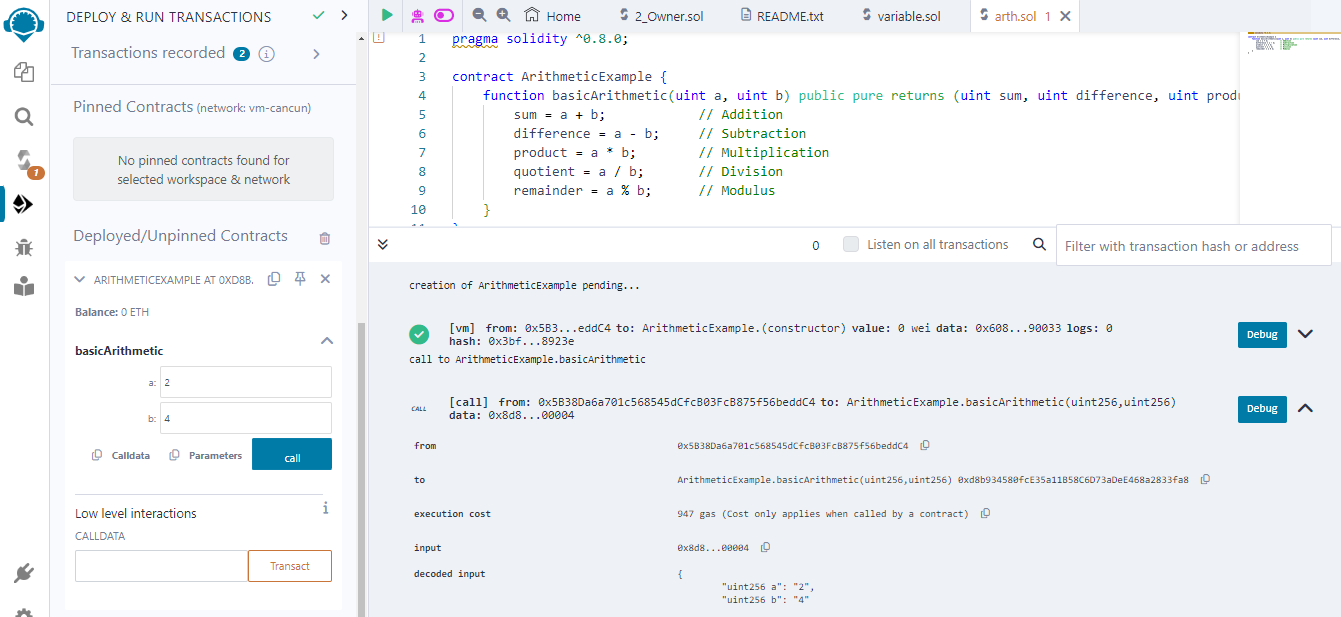
quotient = a / b; // Division

remainder = a % b; // Modulus

}

}

**Output:**

****

**Relational Operator:**

**Code:**

pragma solidity ^0.8.0;

contract RelationalOperatorExample {

function compareValues(uint a, uint b) public pure returns (bool equal, bool notEqual, bool greater, bool less, bool greaterOrEqual, bool lessOrEqual) {

equal = (a == b); // Equal

notEqual = (a != b); // Not Equal

greater = (a > b); // Greater Than

less = (a < b); // Less Than

greaterOrEqual = (a >= b); // Greater Than or Equal To

lessOrEqual = (a <= b); // Less Than or Equal To

}

}

**Output:**

****

**logical operator:**

**Code:**

pragma solidity ^0.8.0;

contract LogicalOperatorExample {

function logicalOperations(bool a, bool b) public pure returns (bool andResult, bool orResult, bool notA, bool notB) {

andResult = a && b; // Logical AND

orResult = a || b; // Logical OR

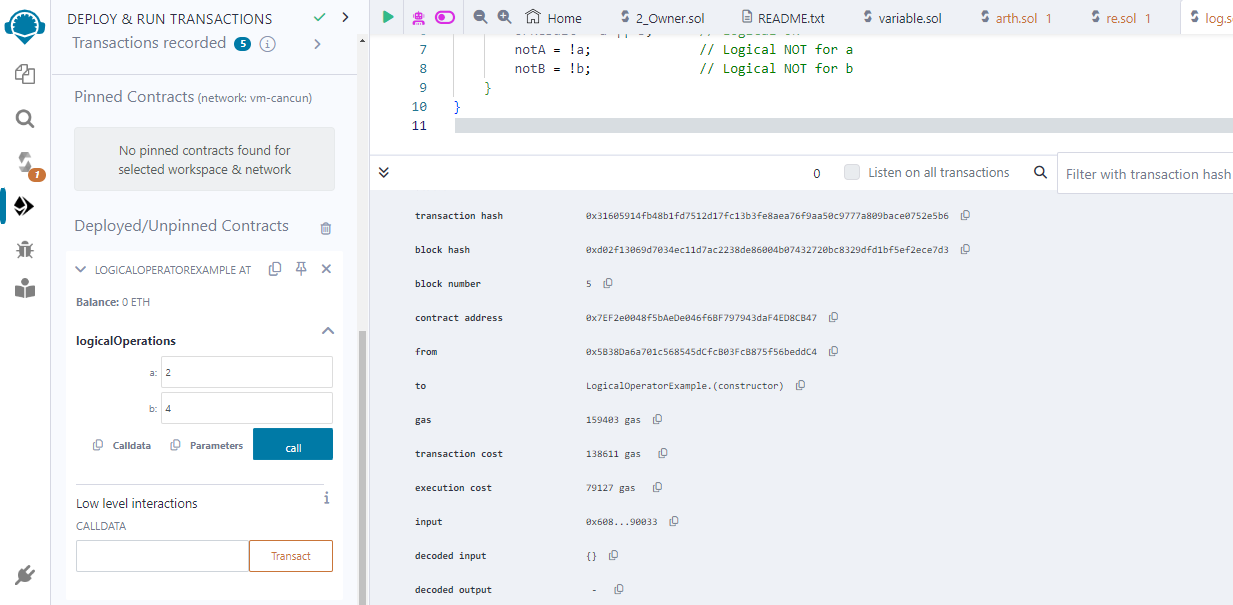
notA = !a; // Logical NOT for a

notB = !b; // Logical NOT for b

}

}

**Output:**

****

**bitwise operators:**

**Code:**

pragma solidity ^0.8.0;

contract BitwiseOperatorExample {

function bitwiseOperations(uint a, uint b) public pure returns (uint andResult, uint orResult, uint xorResult, uint notA, uint leftShift, uint rightShift) {

andResult = a & b; // Bitwise AND

orResult = a | b; // Bitwise OR

xorResult = a ^ b; // Bitwise XOR

notA = ~a; // Bitwise NOT for a

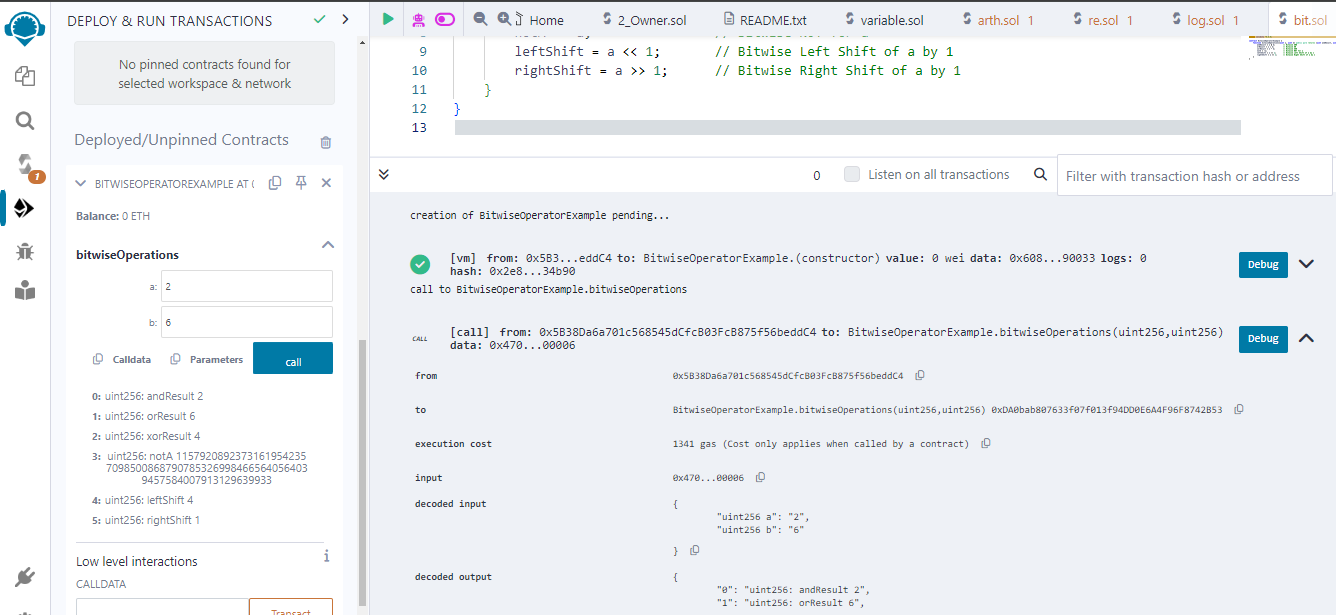
leftShift = a << 1; // Bitwise Left Shift of a by 1

rightShift = a >> 1; // Bitwise Right Shift of a by 1

}

}

**Output:**

****

**assignment operator:**

**Code:**

pragma solidity ^0.8.0;

contract AssignmentOperatorExample {

uint public a;

function assignmentOperations(uint value) public {

a = value; // Simple Assignment

a += value; // Addition Assignment

a -= value; // Subtraction Assignment

a \*= value; // Multiplication Assignment

a /= value; // Division Assignment

a %= value; // Modulus Assignment

a &= value; // Bitwise AND Assignment

a |= value; // Bitwise OR Assignment

a ^= value; // Bitwise XOR Assignment

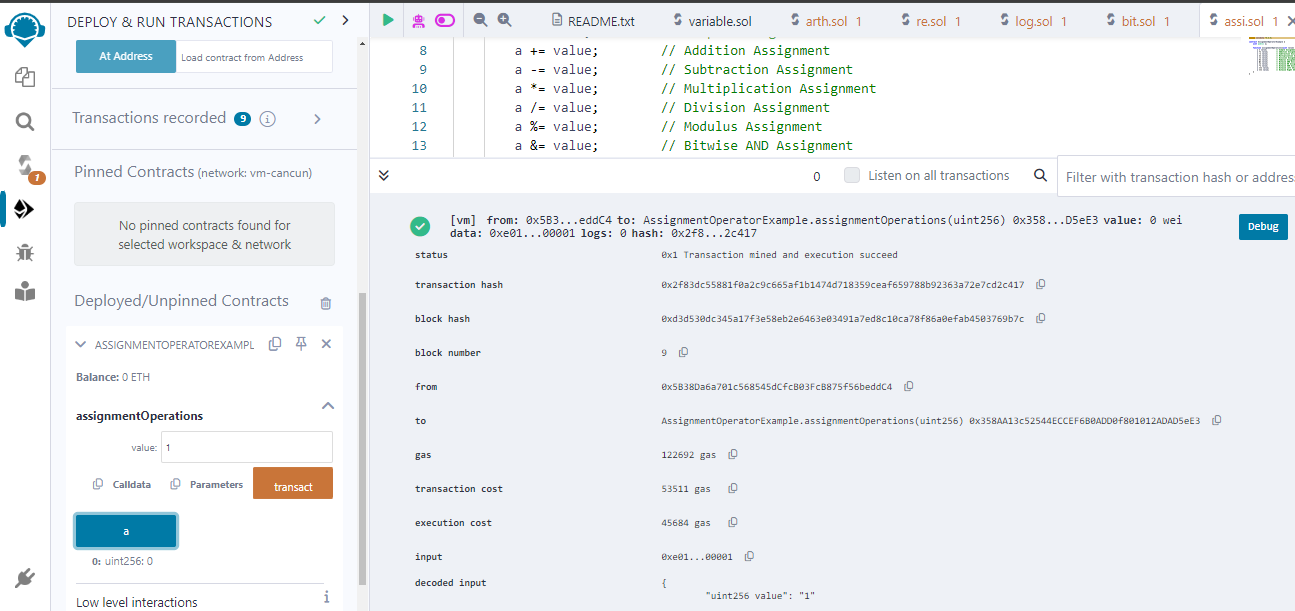
a <<= value; // Bitwise Left Shift Assignment

a >>= value; // Bitwise Right Shift Assignment

}

}

**Output:**

****

**3. Loops:**

**for loop:**

**Code:**

pragma solidity ^0.8.0;

contract ForLoopExample {

uint[] public numbers;

// Function to populate the numbers array

function populateNumbers(uint count) public {

for (uint i = 0; i < count; i++) {

numbers.push(i);

}

}

// Function to calculate the sum of the numbers array

function sumNumbers() public view returns (uint) {

uint sum = 0;

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

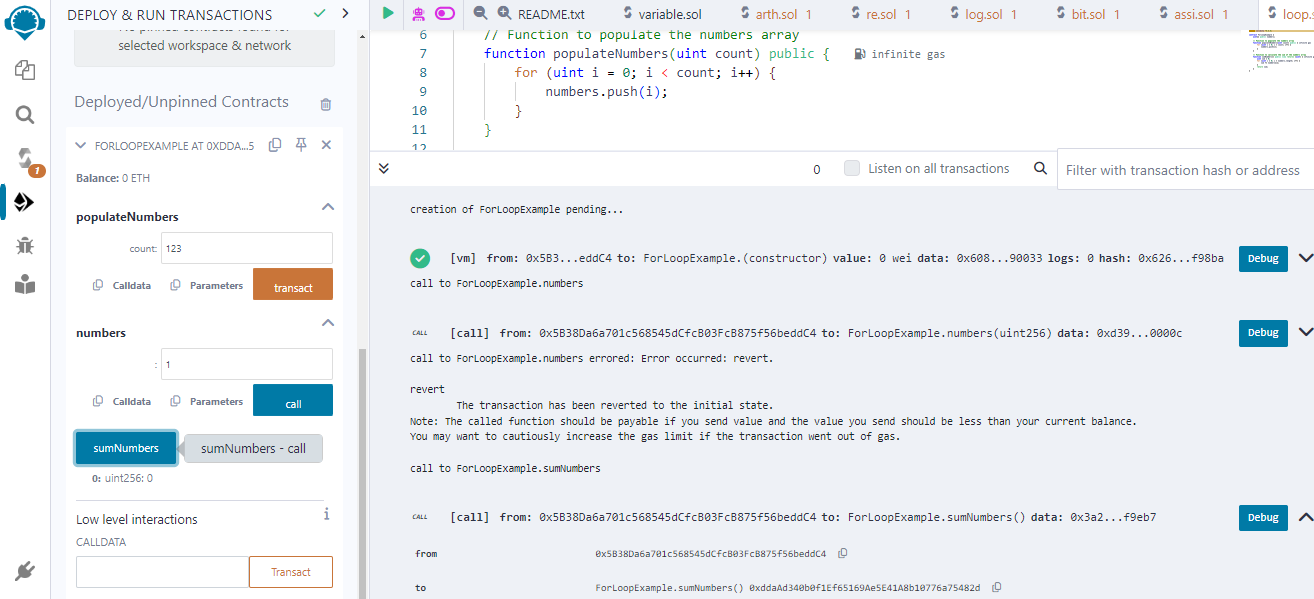
sum += numbers[i];

}

return sum;

}}

**Output:**

****

**while loop:**

**Code:**

pragma solidity ^0.8.0;

contract WhileLoopExample {

uint public counter;

// Function to simulate a while loop

function simulateWhileLoop(uint limit) public {

counter = 0;

for (; counter < limit; ) {

// Do something here

counter++;

} }}

**Output:**

****

**do while loop:**

**Code:**

pragma solidity ^0.8.0;

contract DoWhileLoopExample {

uint public counter;

// Function to simulate a do-while loop

function simulateDoWhileLoop(uint limit) public {

counter = 0;

do {

// Do something here

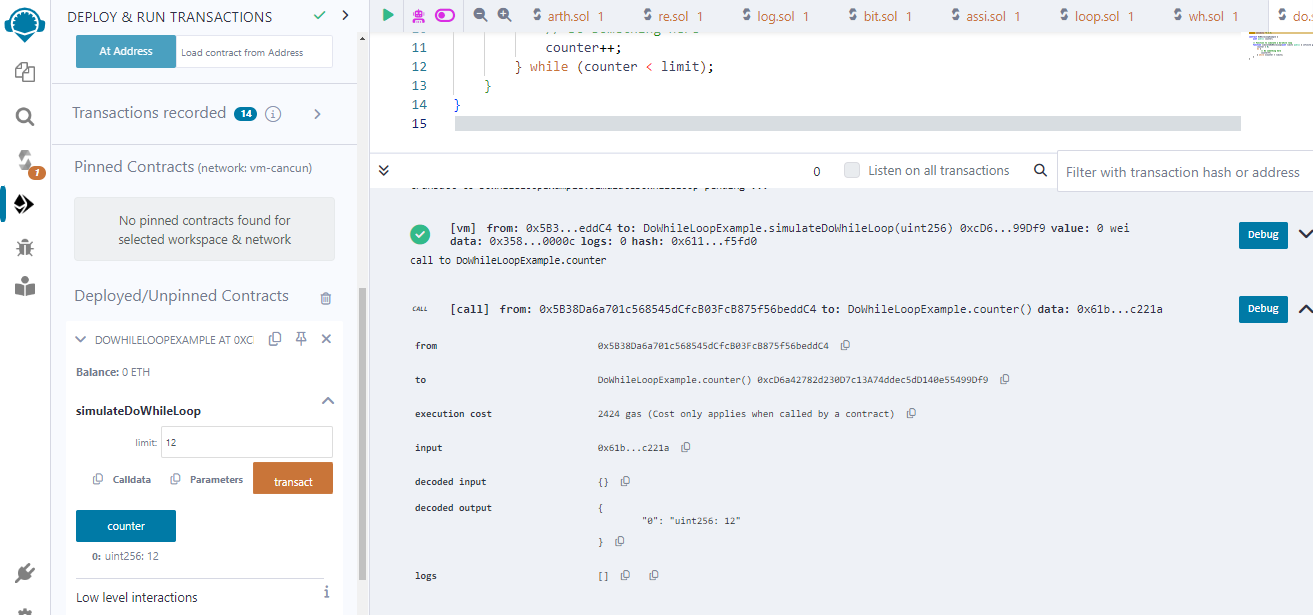
counter++;

} while (counter < limit);

}

}

**Output:**

****

**4. Decision Making:**

**Code:**

**if else:**

pragma solidity ^0.8.0;

contract IfElseExample {

uint public number;

// Function to set number based on condition

function setNumber(uint value) public {

if (value > 10) {

number = value;

} else {

number = 0;

} }

// Function to get a message based on number

function getMessage() public view returns (string memory) {

if (number > 0) {

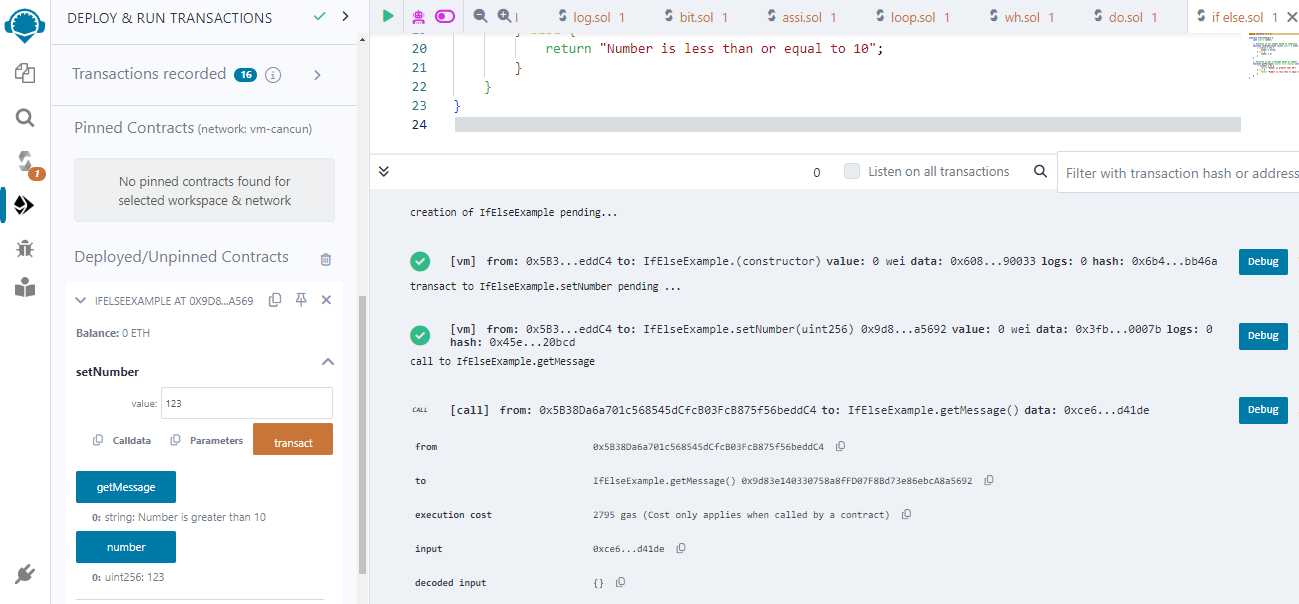
return "Number is greater than 10";

} else {

return "Number is less than or equal to 10";

}}}

**Output:**

****

**if else if else:**

**Code:**

pragma solidity ^0.8.0;

contract IfElseIfElseExample {

uint public number;

// Function to set number based on condition

function setNumber(uint value) public {

if (value > 10) {

number = value;

} else if (value == 10) {

number = value \* 2;

} else {

number = 0;

} }

function getMessage() public view returns (string memory) {

if (number > 10) {

return "Number is greater than 10";

} else if (number == 10) {

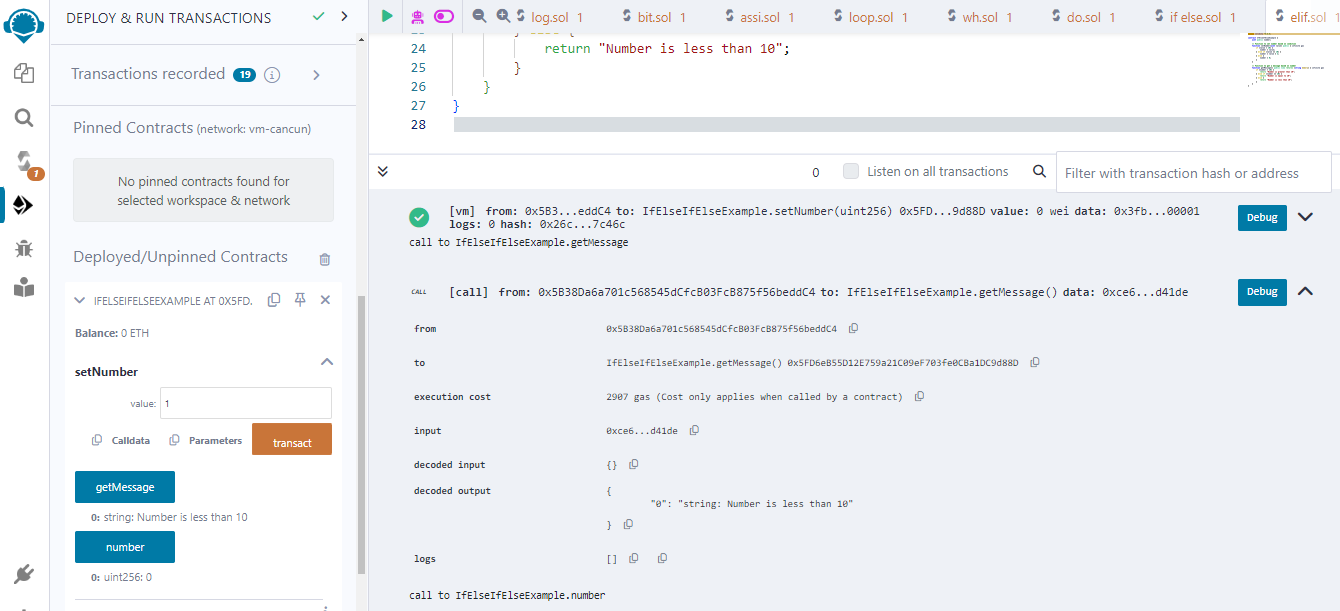
return "Number is equal to 10";

} else {

return "Number is less than 10";

} }}

**Output:**

****

**if statement:**

**Code:**

pragma solidity ^0.8.0;

contract IfStatementExample {

uint public number;

// Function to set number based on condition

function setNumber(uint value) public {

if (value > 10) {

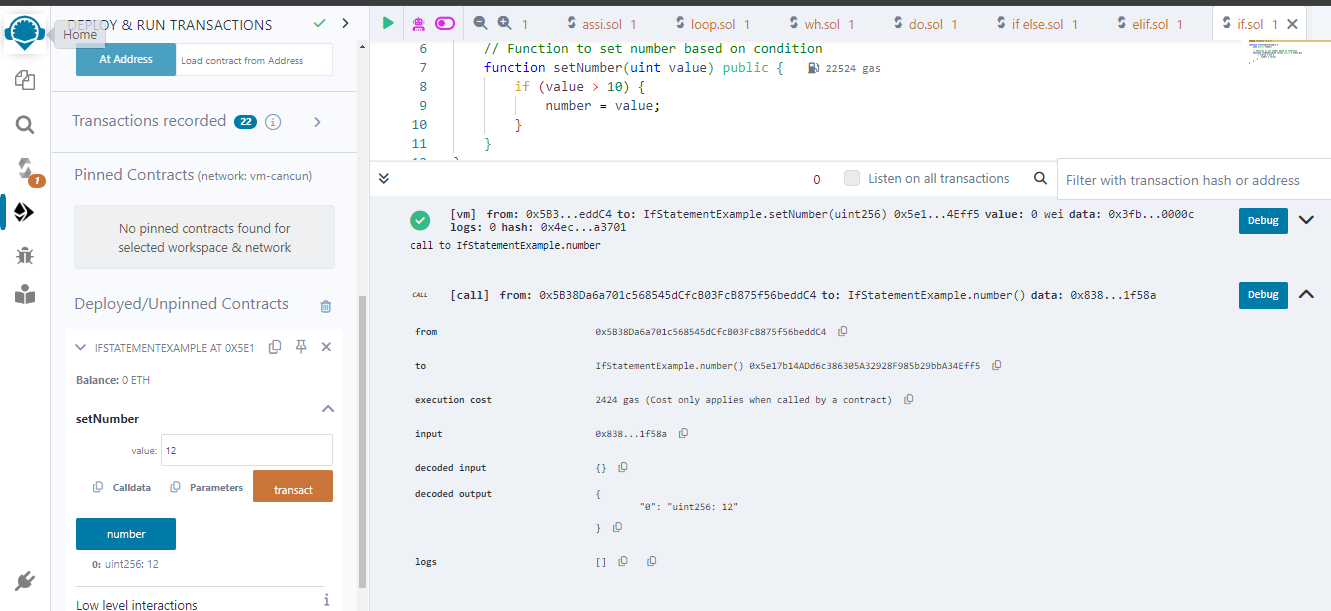
number = value;

}

}

}

**Output:**

****

**5. Strings:**

**Code:**

pragma solidity ^0.8.0;

contract StringExample {

string public myString;

// Function to set a string

function setString(string memory \_value) public {

myString = \_value;

}

// Function to get the length of the string

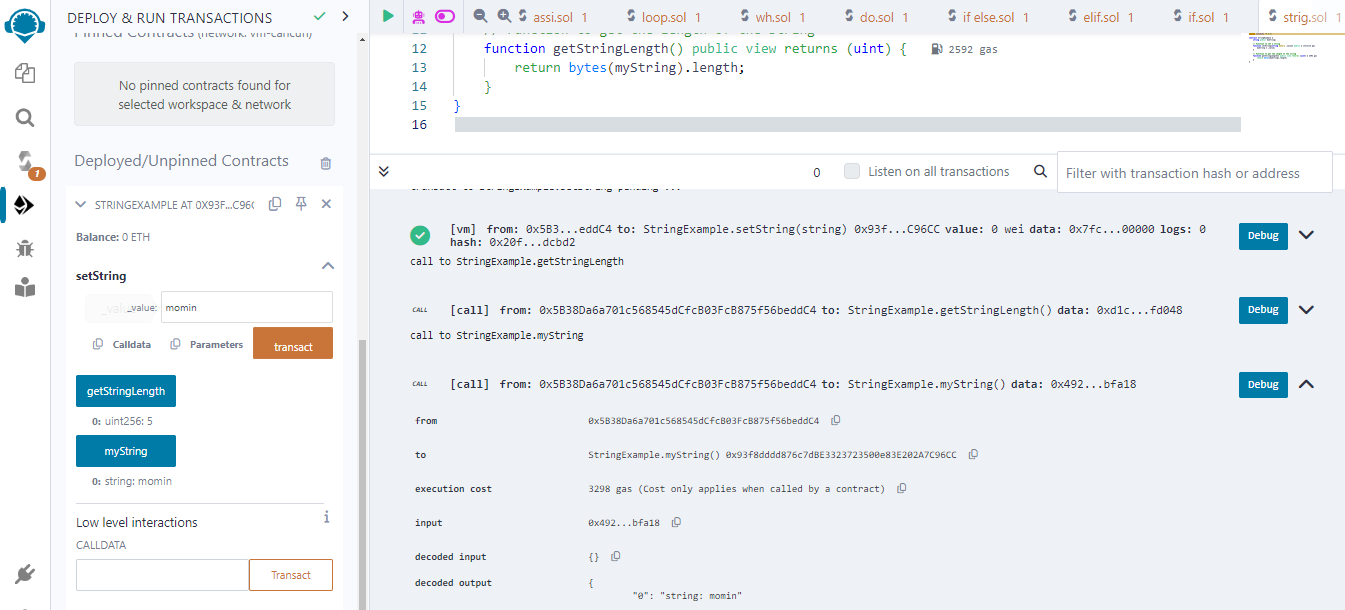
function getStringLength() public view returns (uint) {

return bytes(myString).length;

}

}

**Output:**

****

**6. Array:**

**Code:**

pragma solidity ^0.8.0;

contract ArrayExample {

// Fixed-size array

uint[5] public fixedArray;

// Dynamic array

uint[] public dynamicArray;

// Function to initialize fixed-size array

function initializeFixedArray() public {

fixedArray = [1, 2, 3, 4, 5];

}

// Function to add elements to dynamic array

function addToDynamicArray(uint \_value) public {

dynamicArray.push(\_value);

}

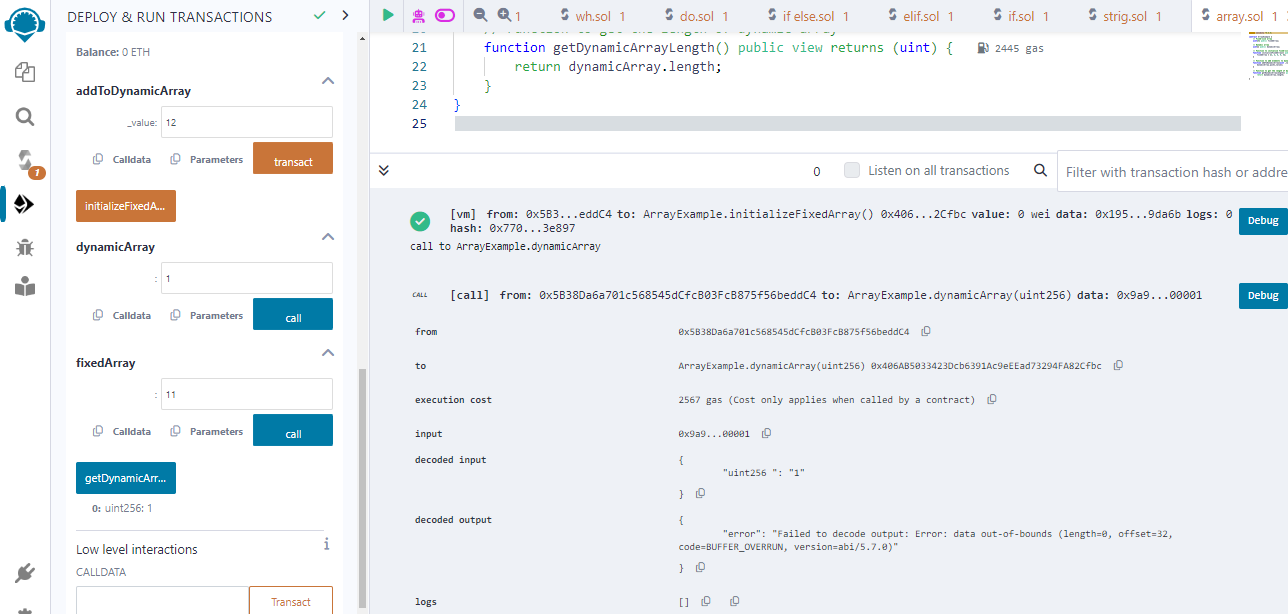
// Function to get the length of dynamic array

function getDynamicArrayLength() public view returns (uint) {

return dynamicArray.length;

}}

**Output:**

****

**7. Enums:**

**Code:**

pragma solidity ^0.8.0;

contract EnumExample {

// Enum to represent different colors

enum Color { Red, Green, Blue }

// State variable of type Color

Color public chosenColor;

// Function to set the chosen color

function setChosenColor(Color \_color) public {

chosenColor = \_color;

}

// Function to get the chosen color

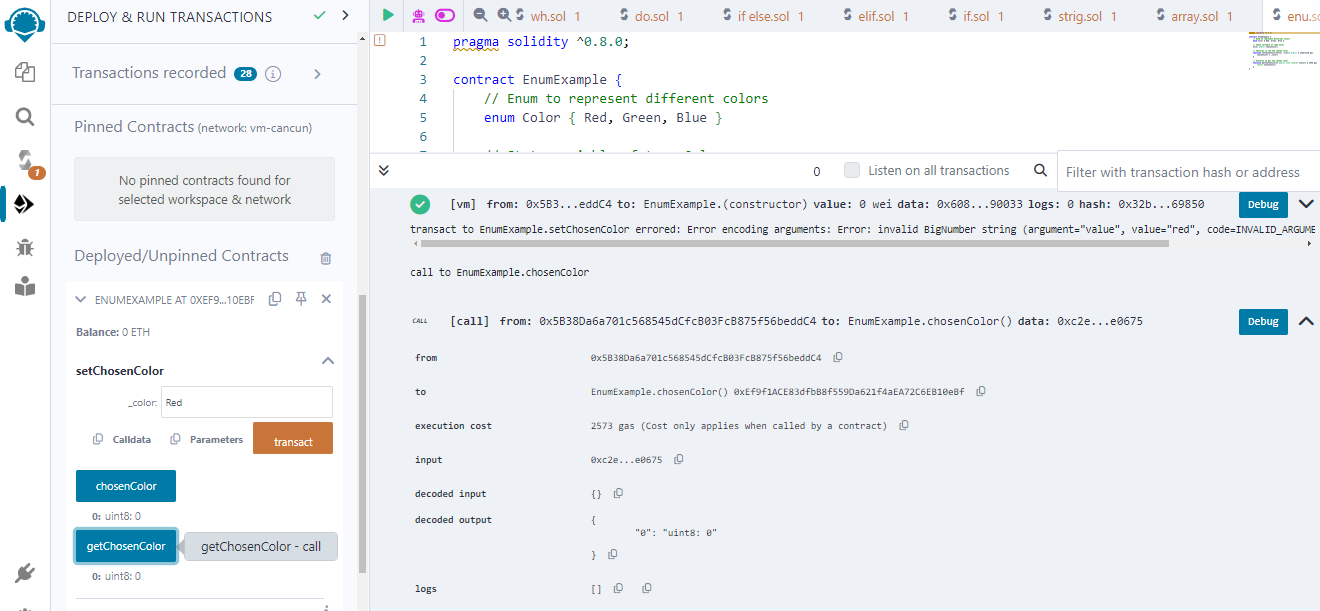
function getChosenColor() public view returns (Color) {

return chosenColor;

}

}

**Output:**

****

**8. structs:**

**Code:**

pragma solidity ^0.8.0;

contract StructExample {

// Struct to represent a Person

struct Person {

string name;

uint age;

}

// State variable of type Person

Person public myPerson;

// Function to set the values of the Person struct

function setPerson(string memory \_name, uint \_age) public {

myPerson = Person(\_name, \_age);

}

// Function to get the values of the Person struct

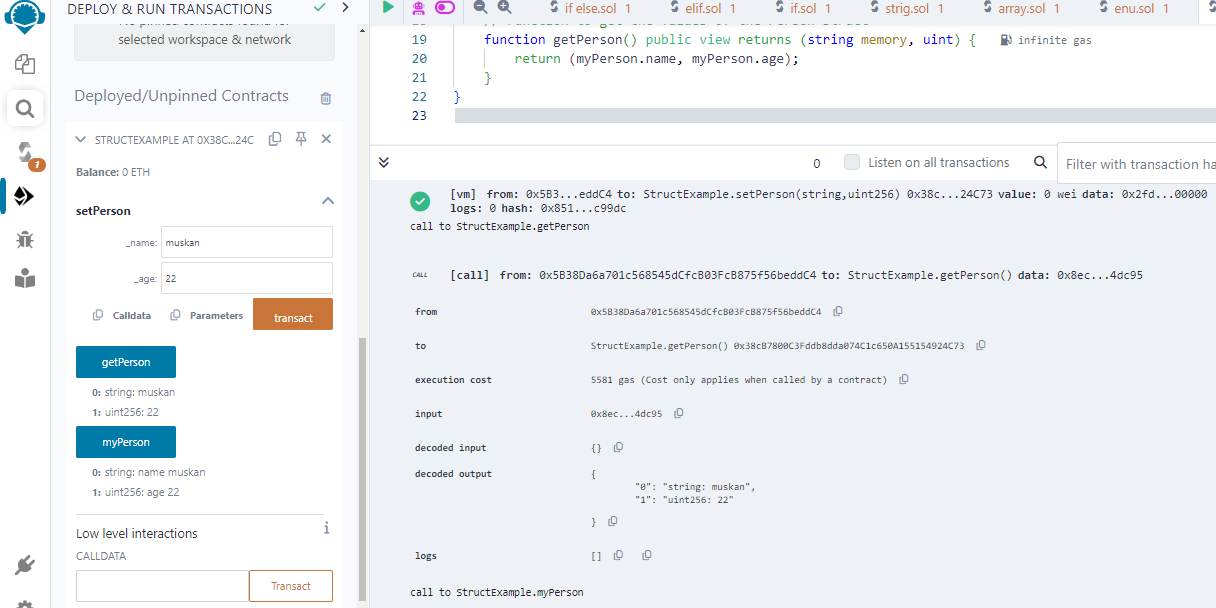
function getPerson() public view returns (string memory, uint) {

return (myPerson.name, myPerson.age);

}

}

**Output:**

****

**9. mapping:**

**Code:**

pragma solidity ^0.8.0;

contract MappingExample {

// Mapping to store the balance of each address

mapping(address => uint) public balances;

// Function to set the balance of an address

function setBalance(address \_address, uint \_balance) public {

balances[\_address] = \_balance;

}

// Function to get the balance of an address

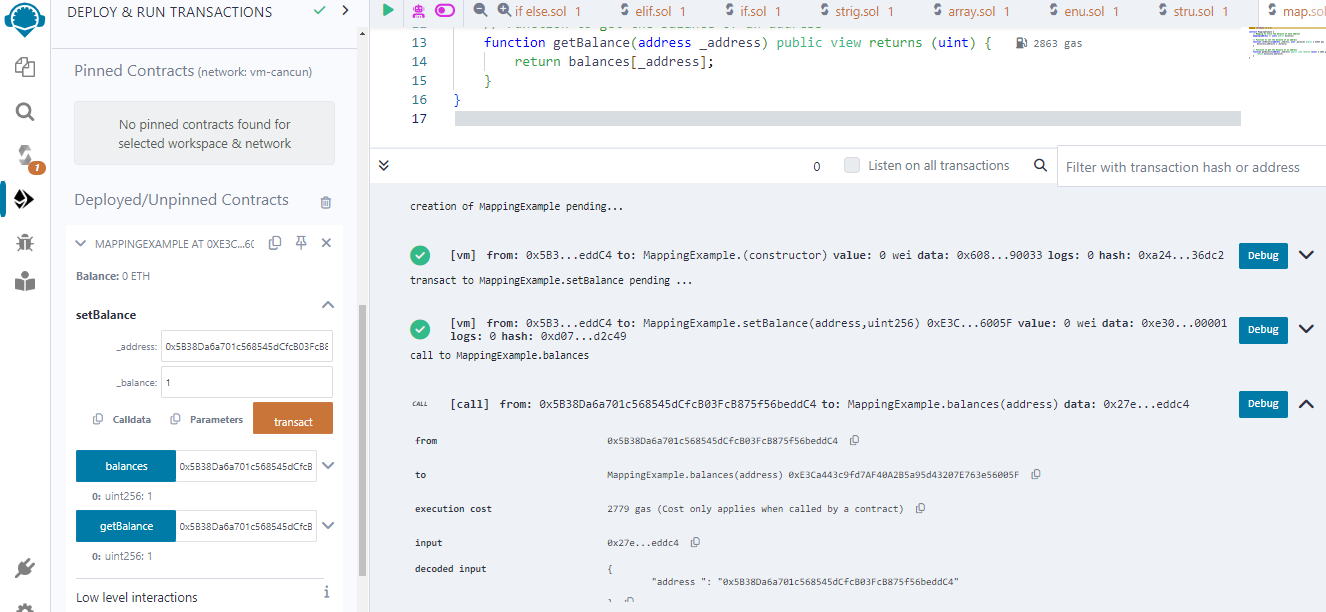
function getBalance(address \_address) public view returns (uint) {

return balances[\_address];

}

}

**Output:**

****

**10. conversion:**

**Code:**

pragma solidity ^0.8.0;

contract ConversionExample {

// Function to convert an integer to a string

function intToString(uint256 \_number) public pure returns (string memory) {

if (\_number == 0) {

return "0";

}

uint256 temp = \_number;

uint256 digits;

while (temp != 0) {

digits++;

temp /= 10;

}

bytes memory buffer = new bytes(digits);

while (\_number != 0) {

buffer[--digits] = bytes1(uint8(48 + \_number % 10));

\_number /= 10;

}

return string(buffer);

}}

**Output:**

**A screenshot of a computer

Description automatically generated**

**11. ether :**

**Code:**

pragma solidity ^0.8.0;

contract EtherExample {

// Function to receive ether

receive() external payable {}

// Function to send ether

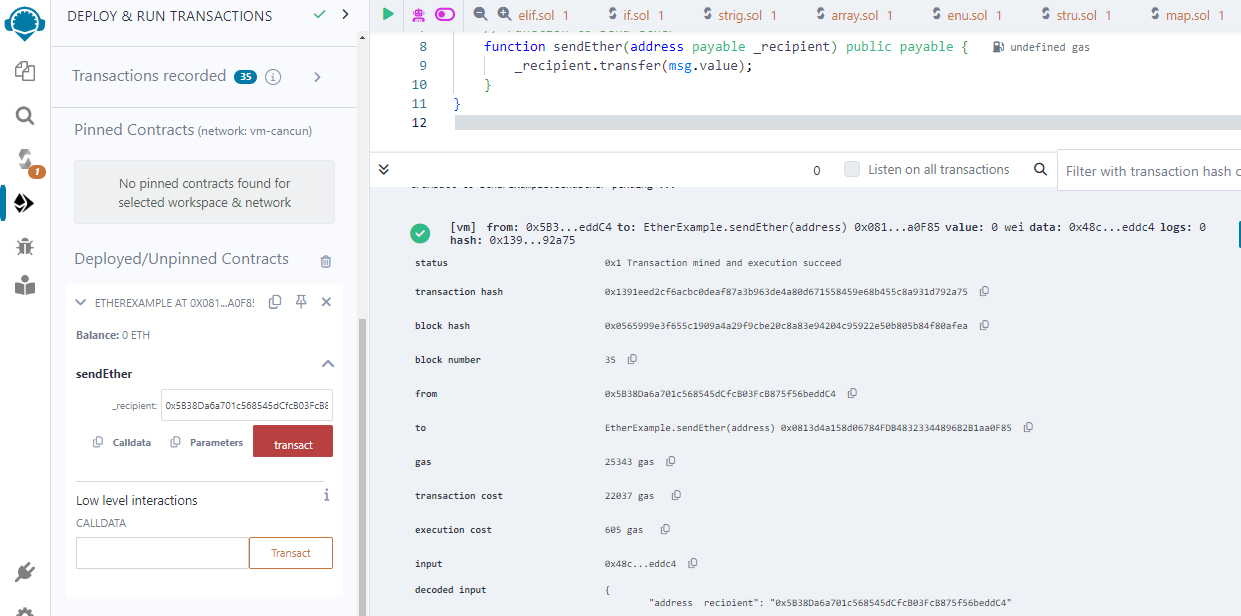
function sendEther(address payable \_recipient) public payable {

\_recipient.transfer(msg.value);

}

}

**Output:**

****

**12.** **special variable:**

**Code:**

pragma solidity ^0.8.0;

contract SimpleContract {

address public owner;

// Constructor to set the owner of the contract

constructor() {

owner = msg.sender;

}

// Function to check if the caller is the owner

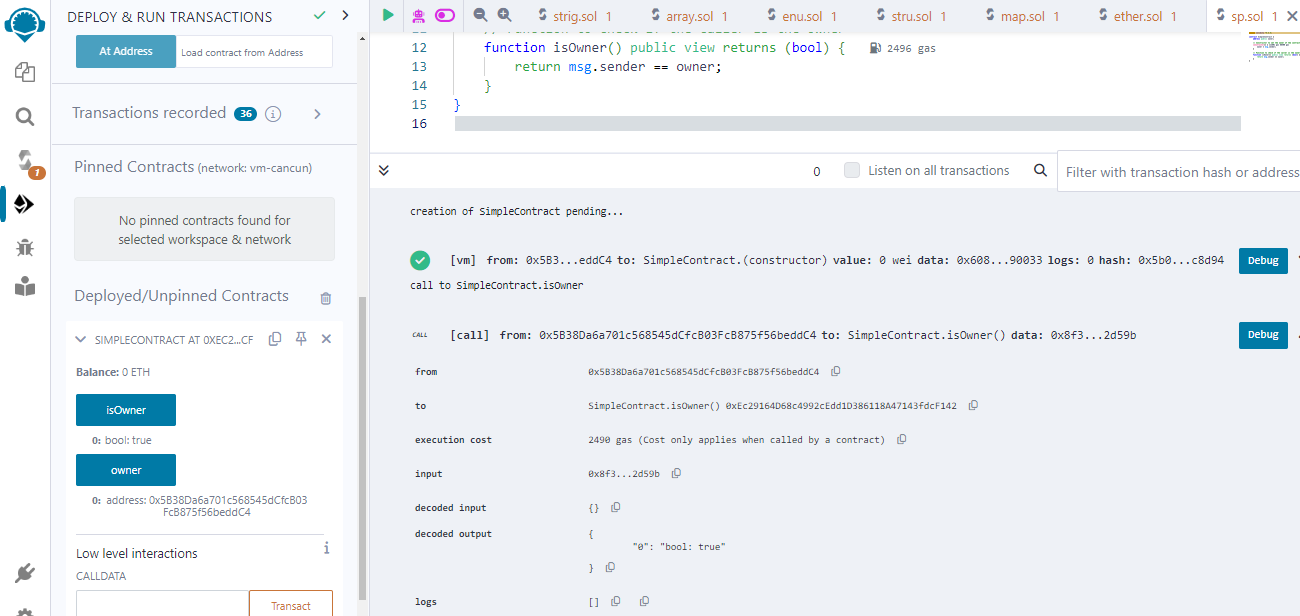
function isOwner() public view returns (bool) {

return msg.sender == owner;

}

}

**Output:**

****

**b) Functions, Function Modifiers, View functions, Pure Functions,**

**Fallback Function, Function Overloading, Mathematical functions,**

**Cryptographic functions.**

**Function:**

**Code:**

pragma solidity ^0.8.0;

contract FunctionExample {

// State variable

uint public data;

// Function to set data

function setData(uint \_data) public {

data = \_data;

}

// Function to get data

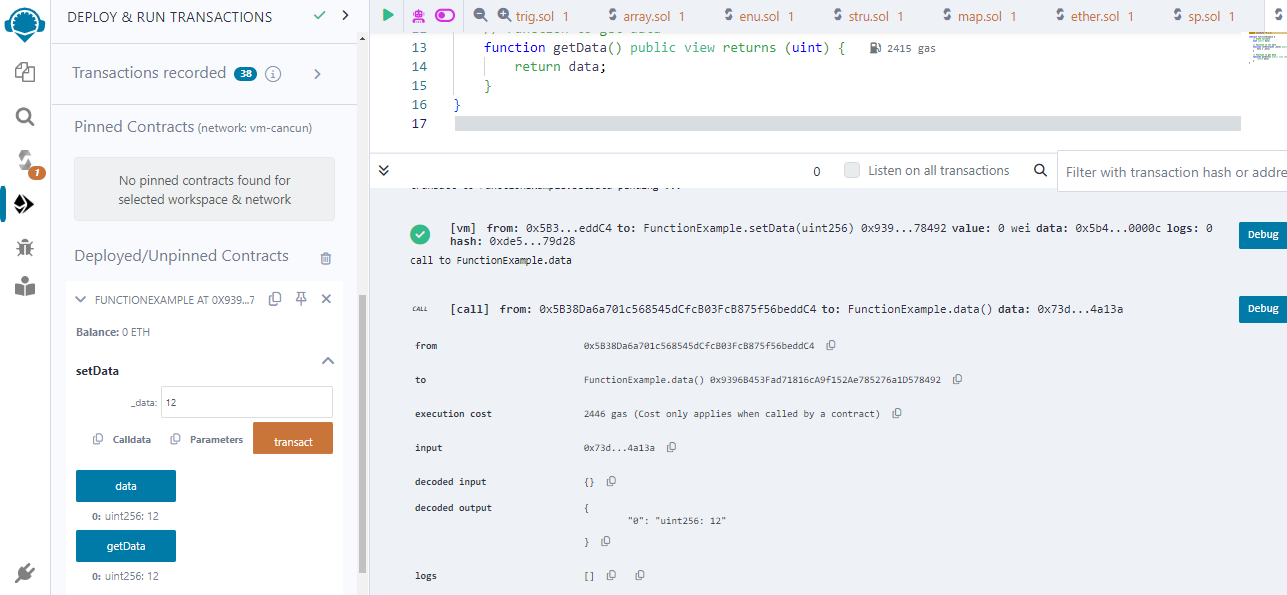
function getData() public view returns (uint) {

return data;

}

}

**Output:**

****

**Functions Modifiers:**

**Code:**

pragma solidity ^0.8.0;

contract SimpleModifierExample {

address public owner;

uint public data;

// Modifier to check if the caller is the owner

modifier onlyOwner() {

require(msg.sender == owner, "Only owner can call this function");

\_; // Continue executing the function body

}

// Constructor to set the owner of the contract

constructor() {

owner = msg.sender;

}

// Function to set data, accessible only by the owner

function setData(uint \_data) public onlyOwner {

data = \_data;

}

// Function to get data

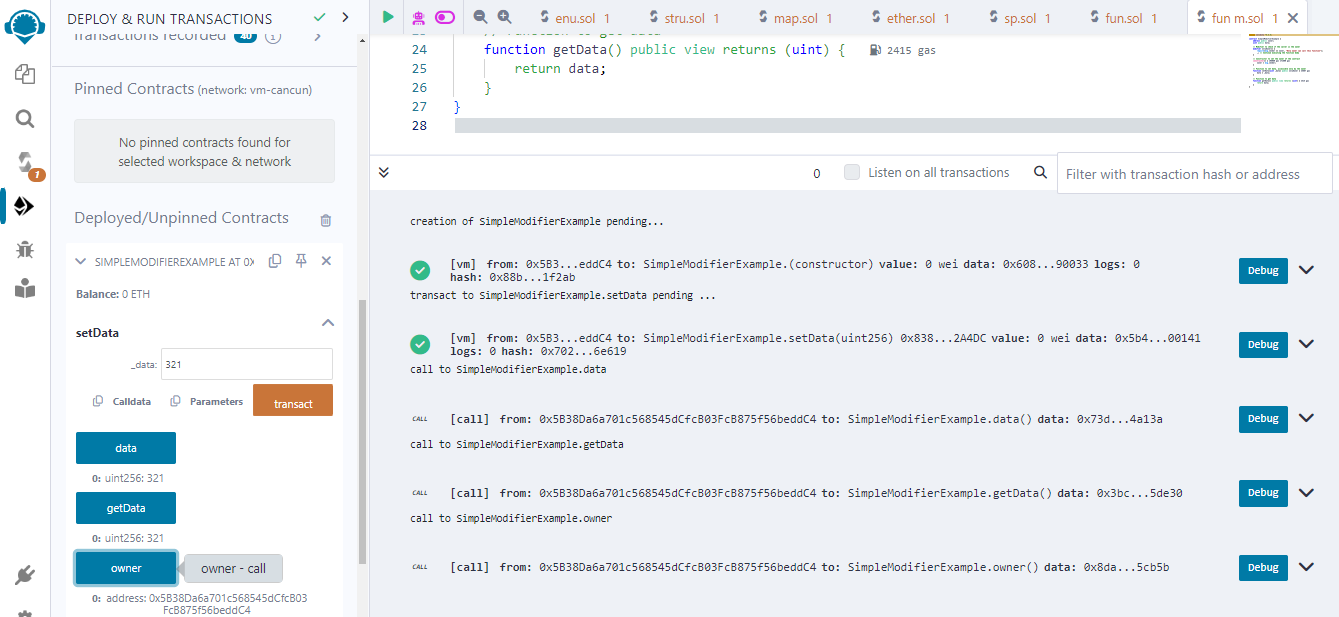
function getData() public view returns (uint) {

return data;

}

}

**Output:**

****

**View function:**

**Code:**

pragma solidity ^0.8.0;

contract ViewFunctionExample {

uint public data;

// Function to set data

function setData(uint \_data) public {

data = \_data;

}

// View function to get data

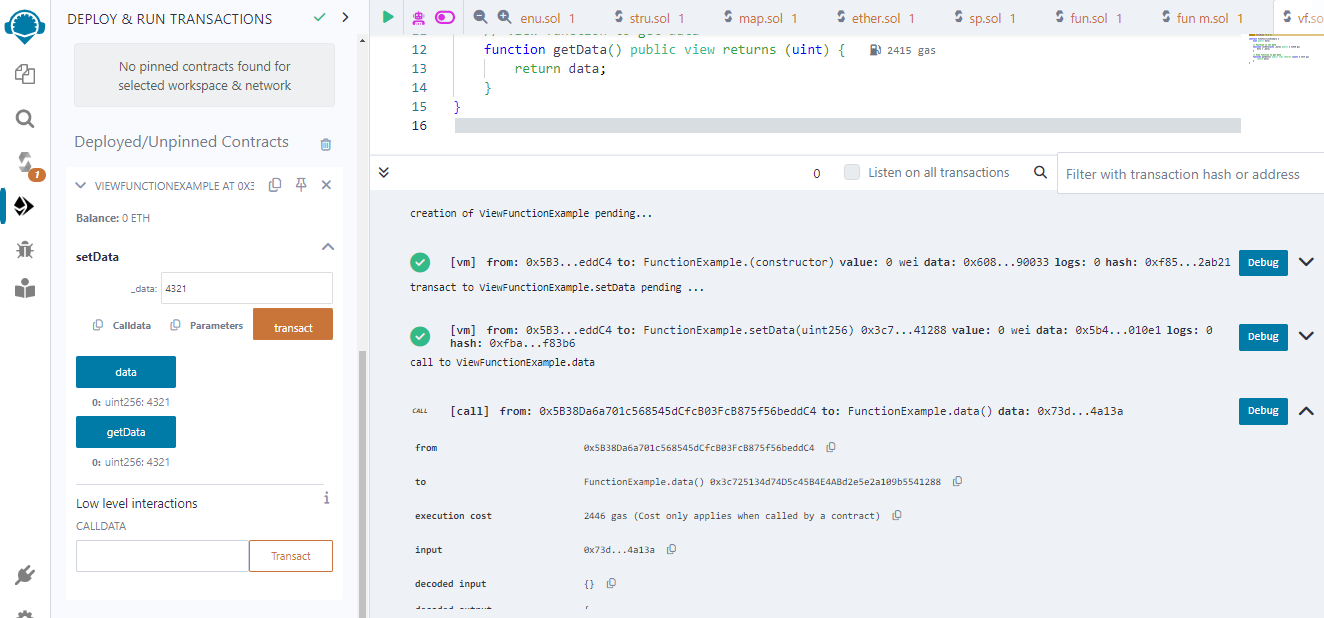
function getData() public view returns (uint) {

return data;

}

}

**Output:**

****

**Pure function:**

**Code:**

pragma solidity ^0.8.0;

contract PureFunctionExample {

// Pure function to add two numbers

function add(uint a, uint b) public pure returns (uint) {

return a + b;

}

// Pure function to multiply two numbers

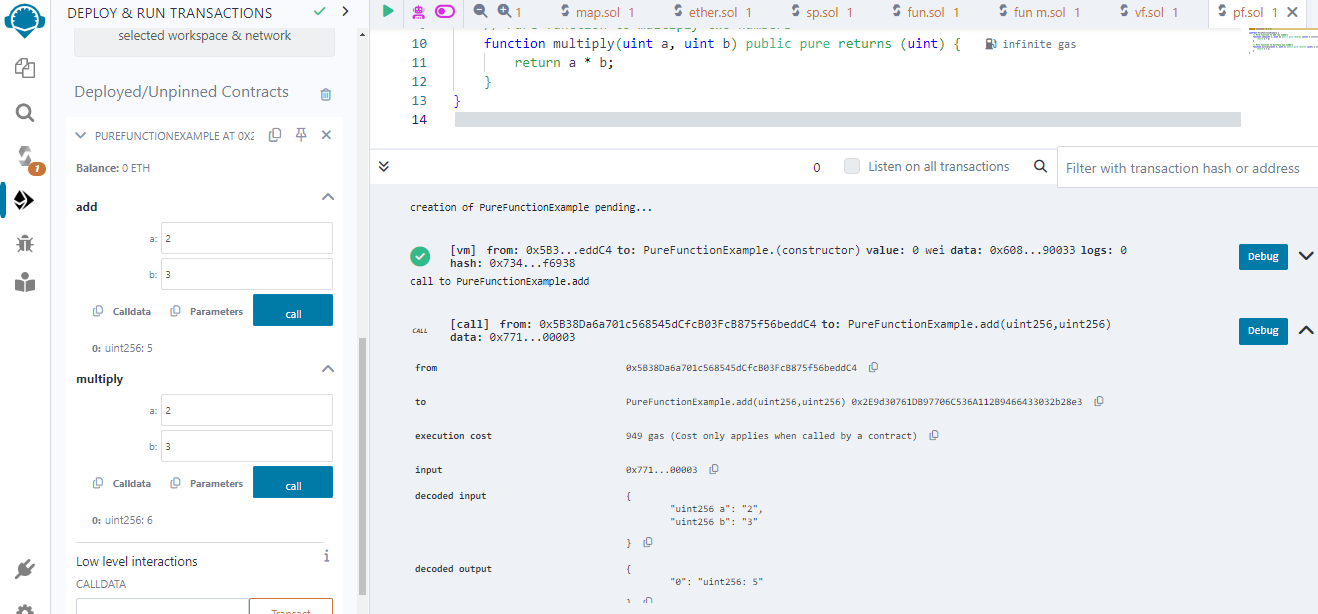
function multiply(uint a, uint b) public pure returns (uint) {

return a \* b;

}

}

**Output:**

****

**Fallback function:**

**Code:**

pragma solidity ^0.8.0;

contract A {

uint256 public n;

function set(uint256 value) external {

n = value;

}

fallback() external payable {

n = 0;

}

}

contract example {

function callA(A a) public returns (bool) {

(bool success, ) = address(a).call(abi.encodeWithSignature("set(uint256)", 2 ether));

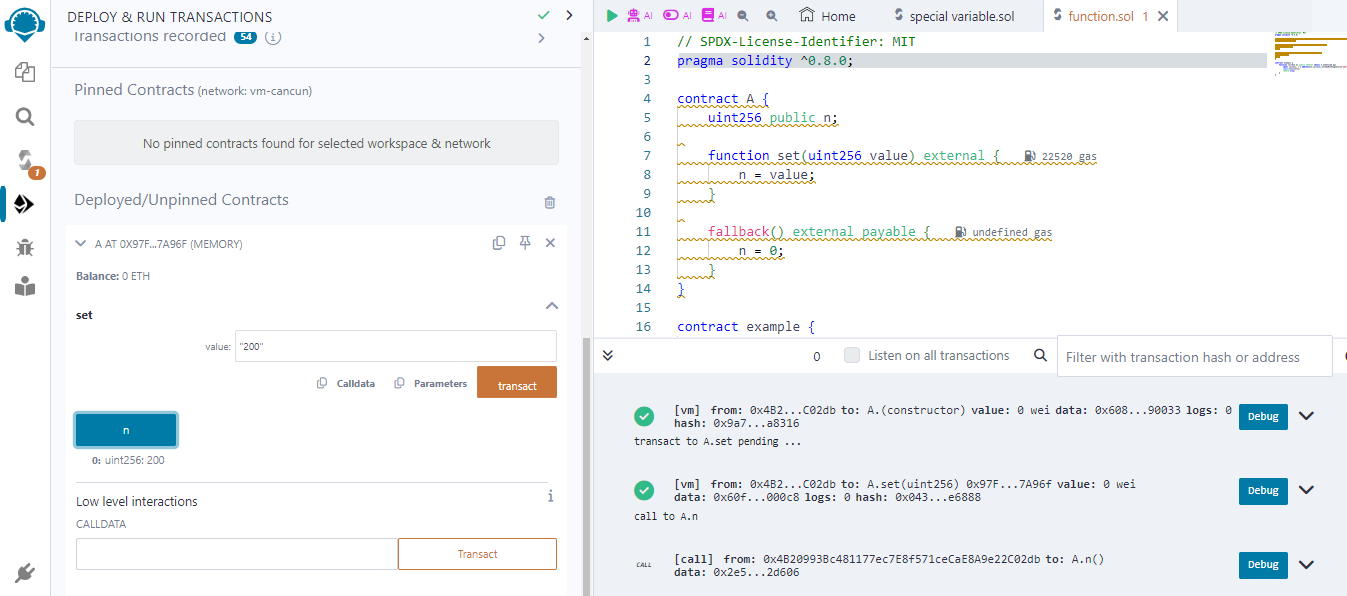
require(success);

return true;

}

}

**Output:**

****

**Function Overloading:**

**Code:**

pragma solidity ^0.8.0;

contract FunctionOverloadingExample {

function process(uint \_value) public pure returns (uint) {

return \_value \* 2;

}

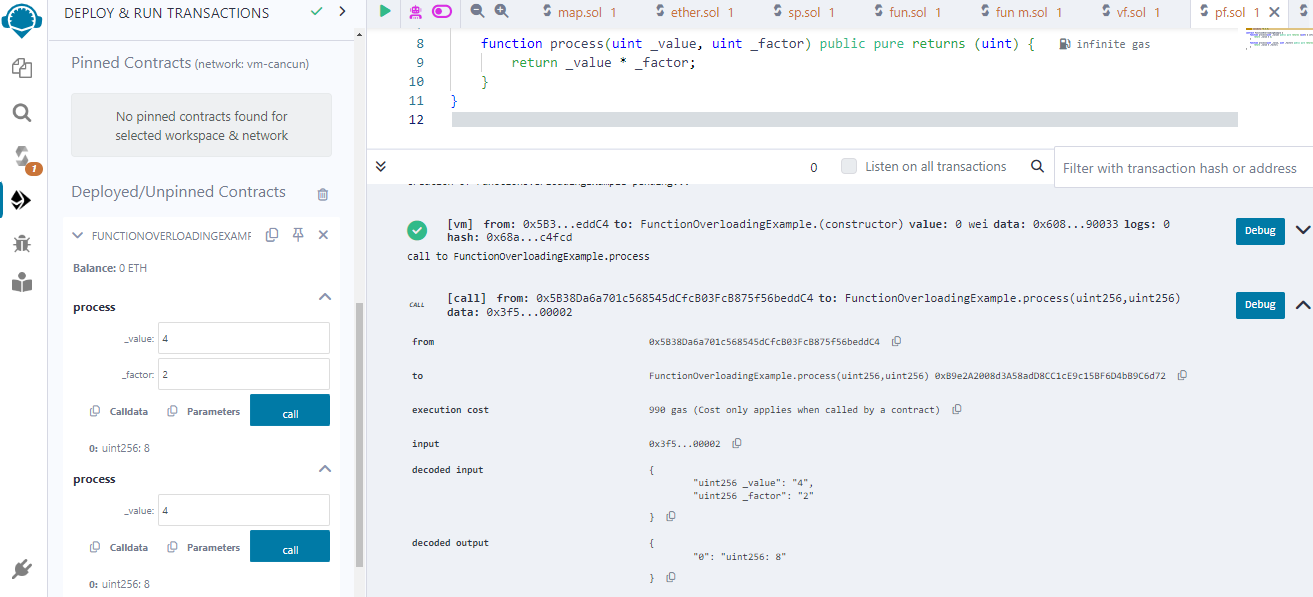
function process(uint \_value, uint \_factor) public pure returns (uint) {

return \_value \* \_factor;

}

}

**Output:**

****

**Mathematical Function:**

**Code:**

pragma solidity ^0.8.0;

contract Test {

function CallAddMod() public pure returns(uint) {

return addmod(7, 3, 3);

}

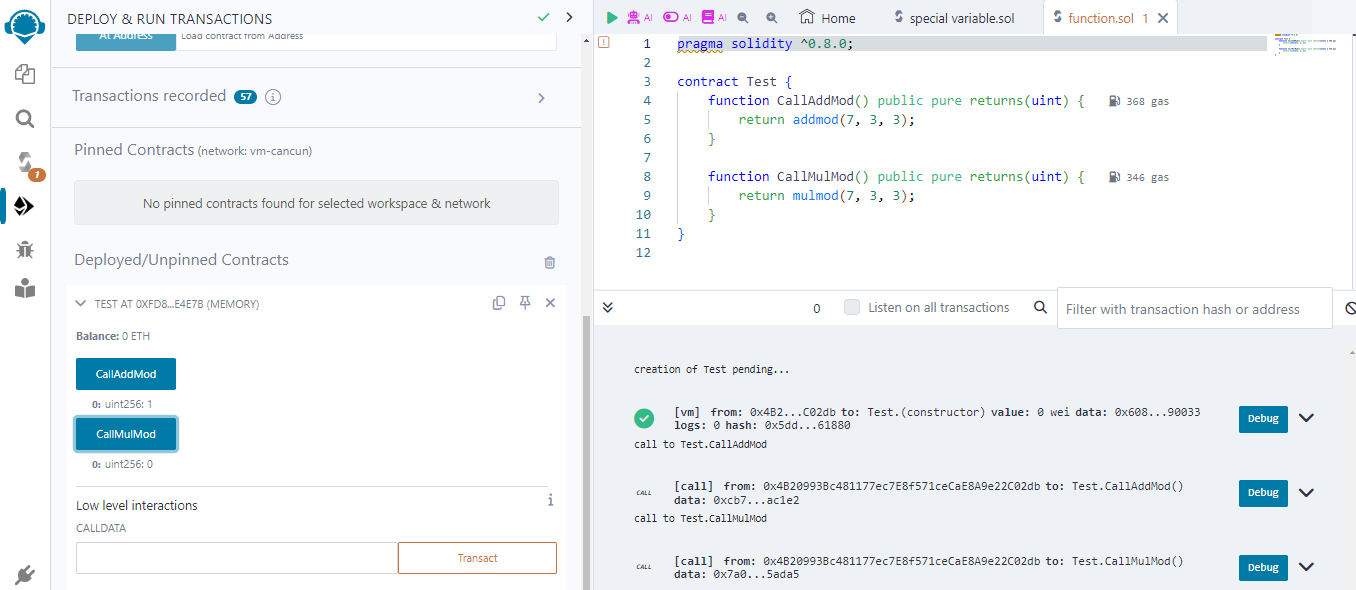
function CallMulMod() public pure returns(uint) {

return mulmod(7, 3, 3);

}

}

**Output:**

****

**Cryptographic function:**

**Code:**

pragma solidity ^0.8.0;

contract CryptographicFunctions {

// Function to hash a string using keccak256

function hashString(string memory \_input) public pure returns (bytes32) {

return keccak256(abi.encodePacked(\_input));

}

// Function to generate a random number based on the current block timestamp and caller's address

function generateRandomNumber() public view returns (uint) {

return uint(keccak256(abi.encode(block.timestamp, msg.sender))) % 100;

}

// Function to verify a digital signature

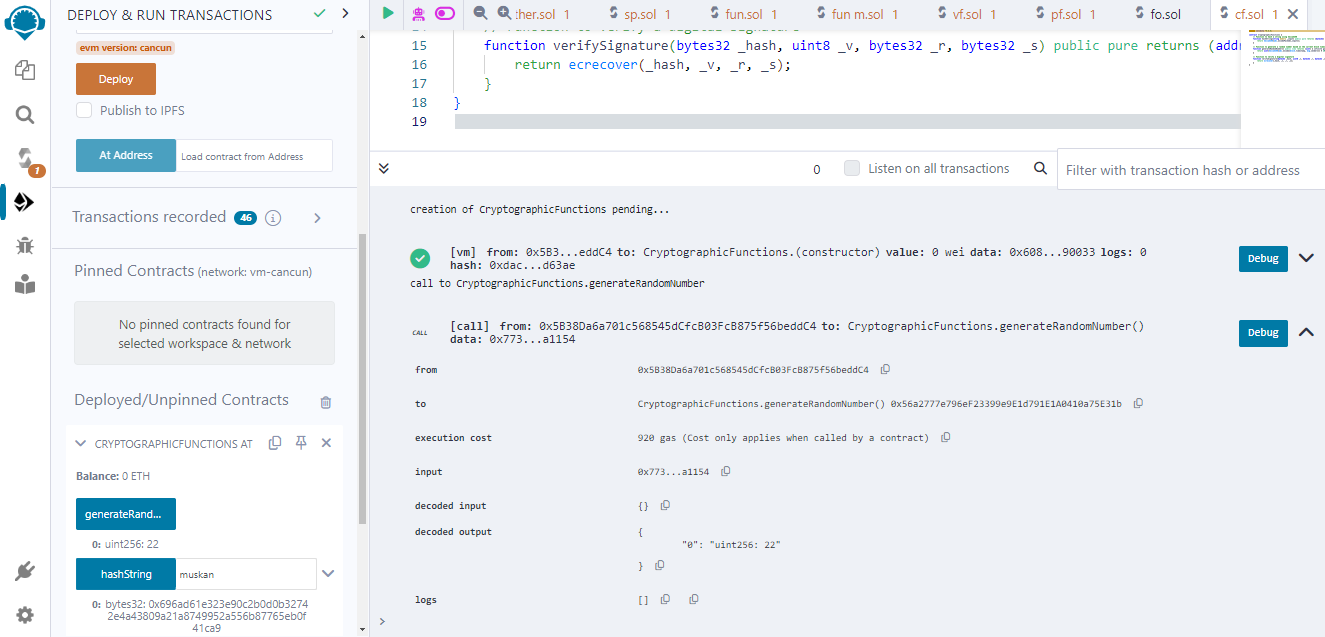
function verifySignature(bytes32 \_hash, uint8 \_v, bytes32 \_r, bytes32 \_s) public pure returns (address) {

return ecrecover(\_hash, \_v, \_r, \_s);

}

}

**Output:**

****

**Practical:**

**Aim: Implement and demonstrate the use of the following in**

**Solidity:**

**a) Withdrawal Pattern, Restricted Access.**

**Withdrawal Pattern:**

**Code:**

pragma solidity ^0.8.0;

contract WithdrawalPattern {

mapping(address => uint) public balances;

// Function to deposit funds into the contract

function deposit() public payable {

balances[msg.sender] += msg.value;

}

// Function to withdraw funds

function withdraw(uint \_amount) public {

require(balances[msg.sender] >= \_amount, "Insufficient balance");

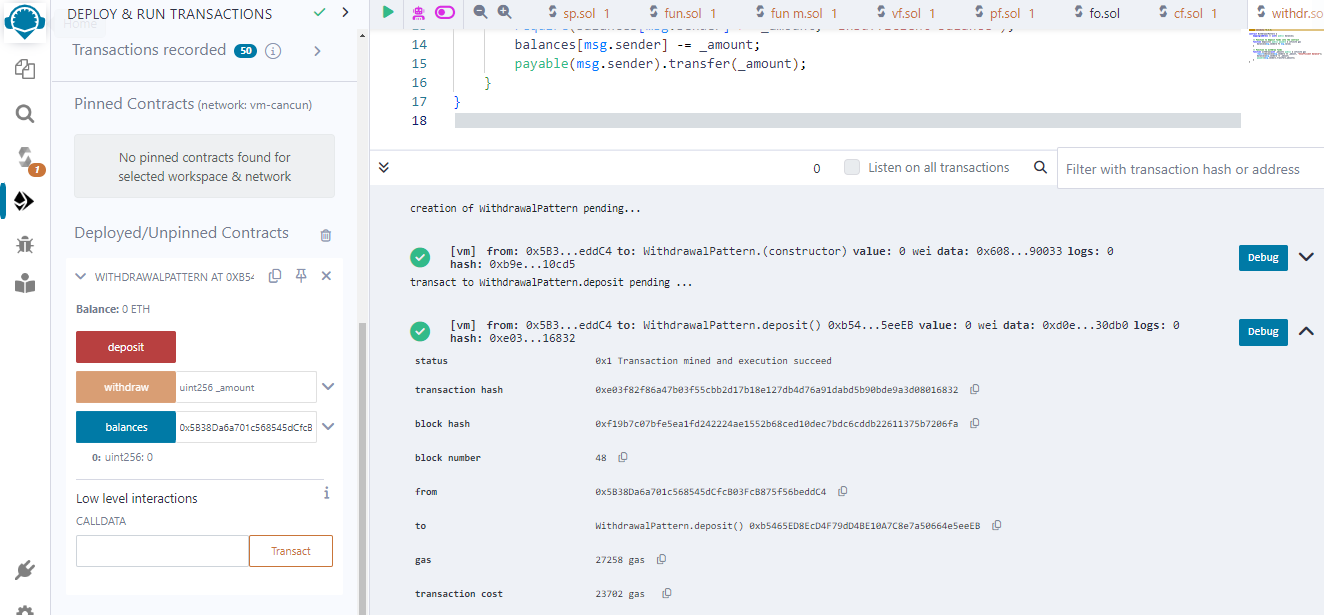
balances[msg.sender] -= \_amount;

payable(msg.sender).transfer(\_amount);

}

}

**Output:**

****

**Restricted Access:**

**Code:**

pragma solidity ^0.8.0;

contract RestrictedAccessUsingModifiers {

address public owner;

// Modifier to allow only the owner to call a function

modifier onlyOwner() {

require(msg.sender == owner, "Caller is not the owner");

\_; // Continue executing the function

}

// Constructor sets the contract owner

constructor() {

owner = msg.sender;

}

// Function restricted to the owner

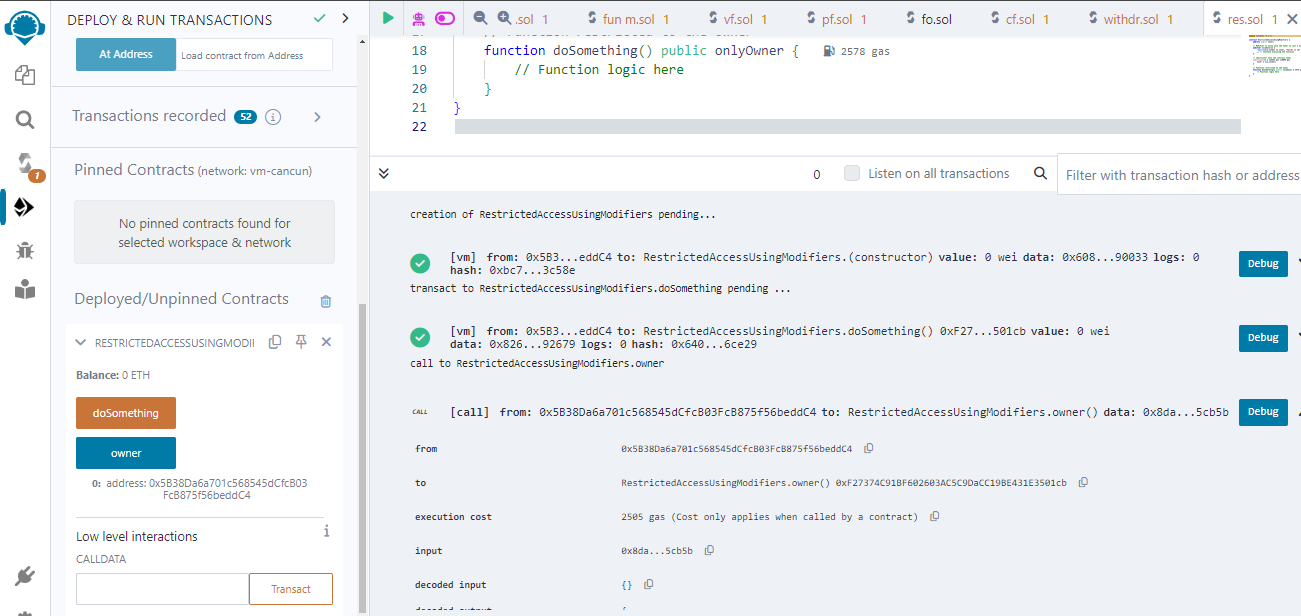
function doSomething() public onlyOwner {

// Function logic here

}

}

**Output:**

****

**b) Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.**

**Contracts:**

**Code:**

pragma solidity ^0.8.0;

contract SimpleContract {

// State variables

uint public data;

// Constructor

constructor() {

data = 0; // Initialize data to 0

}

// Function to set data

function setData(uint \_data) public {

data = \_data;

}

// Function to get data

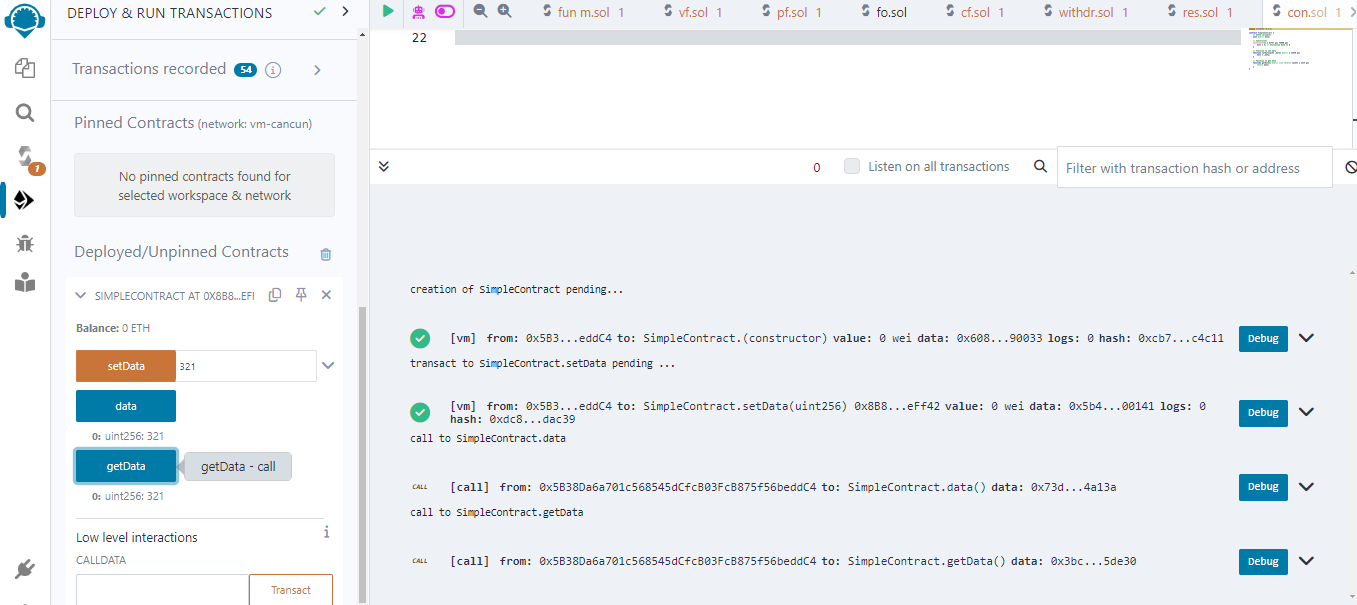
function getData() public view returns (uint) {

return data;

}

}

**Output:**

****

**Inheritance:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Parent contract

contract Parent {

uint256 public parentData;

constructor(uint256 \_parentData) {

parentData = \_parentData;

}

function parentFunction() public pure returns(string memory) {

return "This is from parent contract";

}

}

// Child contract inheriting from Parent

contract Child is Parent {

uint256 public childData;

constructor(uint256 \_parentData, uint256 \_childData) Parent(\_parentData) {

childData = \_childData;

}

function childFunction() public pure returns(string memory) {

return "This is from child contract";

}

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Constructors:**

**Code:**

pragma solidity ^0.8.0;

contract ConstructorExample {

uint public data;

// Constructor to initialize the contract

constructor() {

data = 100; // Initialize data to 100 when the contract is deployed

}

// Function to set data

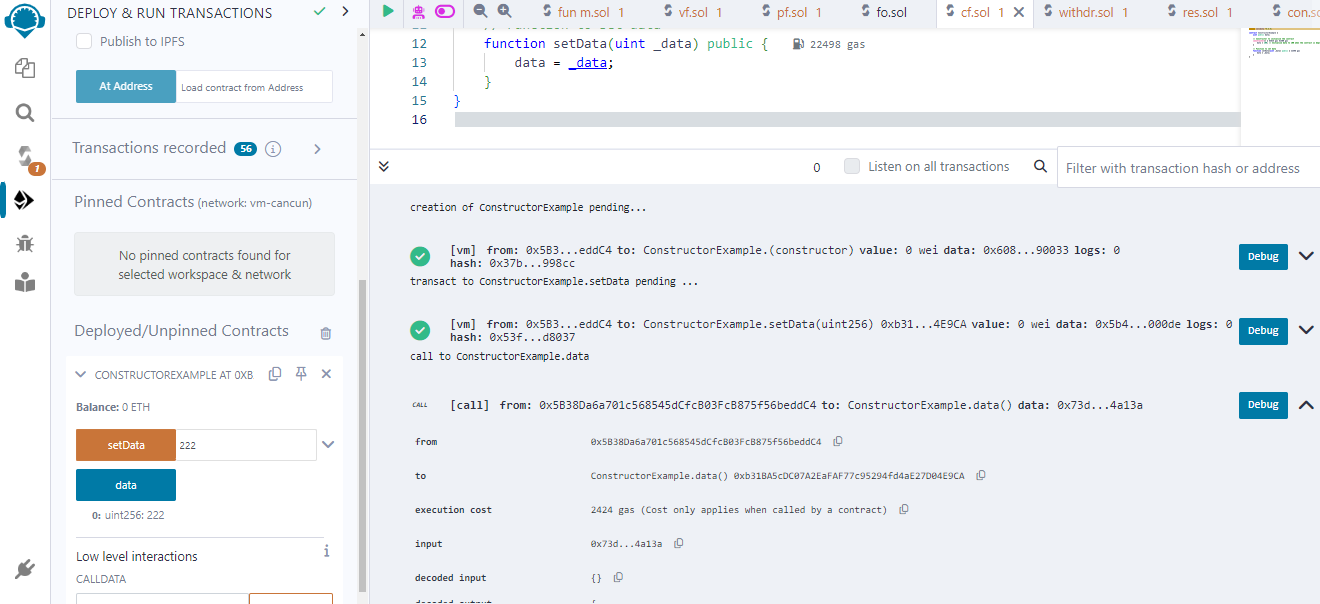
function setData(uint \_data) public {

data = \_data;

}

}

**Output:**

****

**Abstract Contracts:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Abstract contract

abstract contract Animal {

function makeSound() public virtual returns (string memory);

}

// Concrete contract implementing the abstract contract

contract Dog is Animal {

function makeSound() public pure override returns (string memory) {

return "Woof!";

}

}

// Concrete contract implementing the abstract contract

contract Cat is Animal {

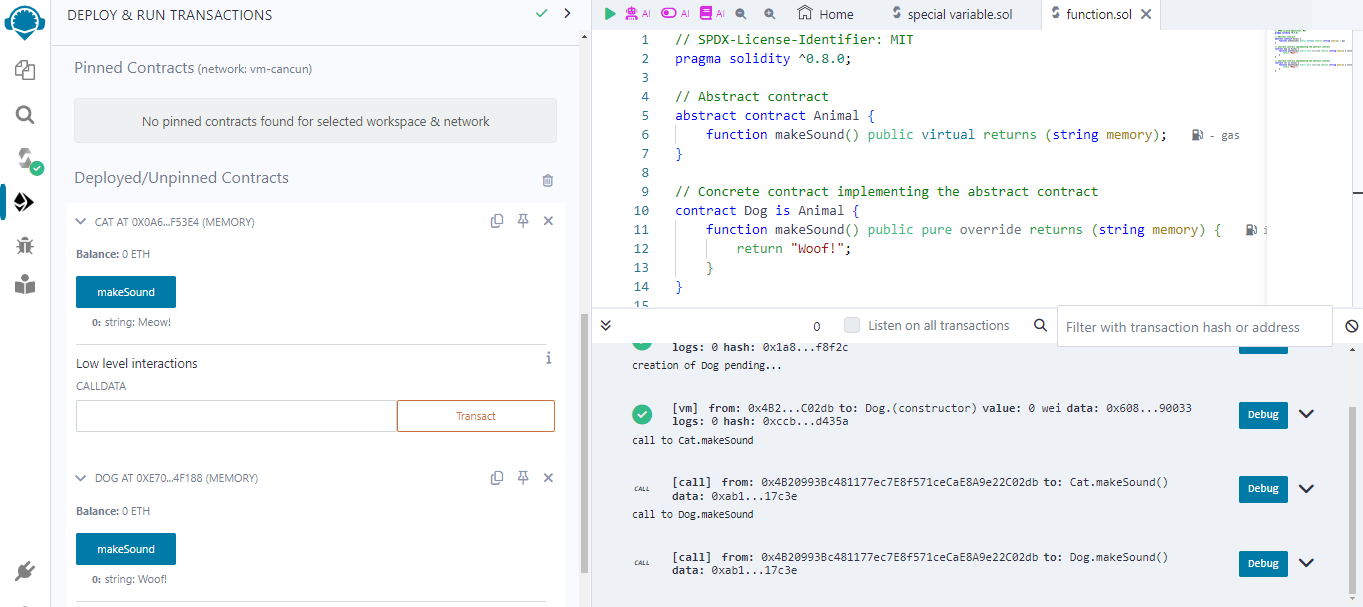
function makeSound() public pure override returns (string memory) {

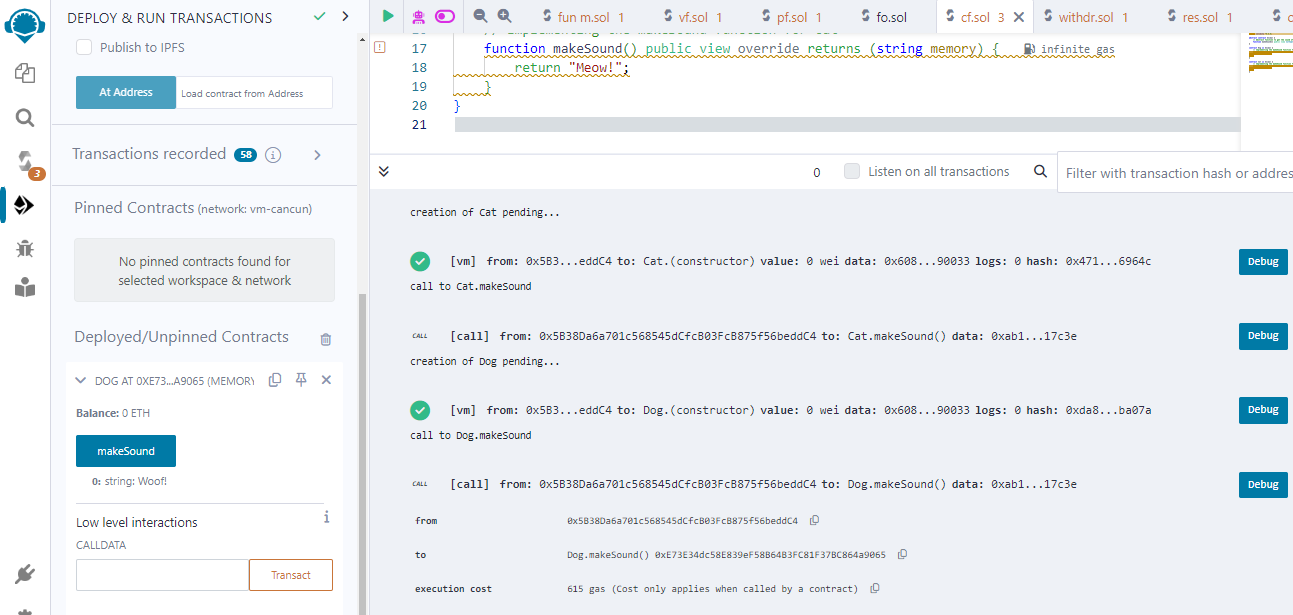
return "Meow!";

}

}

**Output:**

****

****

**Interfaces:**

**Code:**

pragma solidity ^0.8.0;

interface Token {

// Function to transfer tokens

function transfer(address recipient, uint amount) external returns (bool);

}

contract MyContract {

Token public token;

// Constructor to initialize the token contract

constructor(address \_tokenAddress) {

token = Token(\_tokenAddress);

}

// Function to transfer tokens using the interface

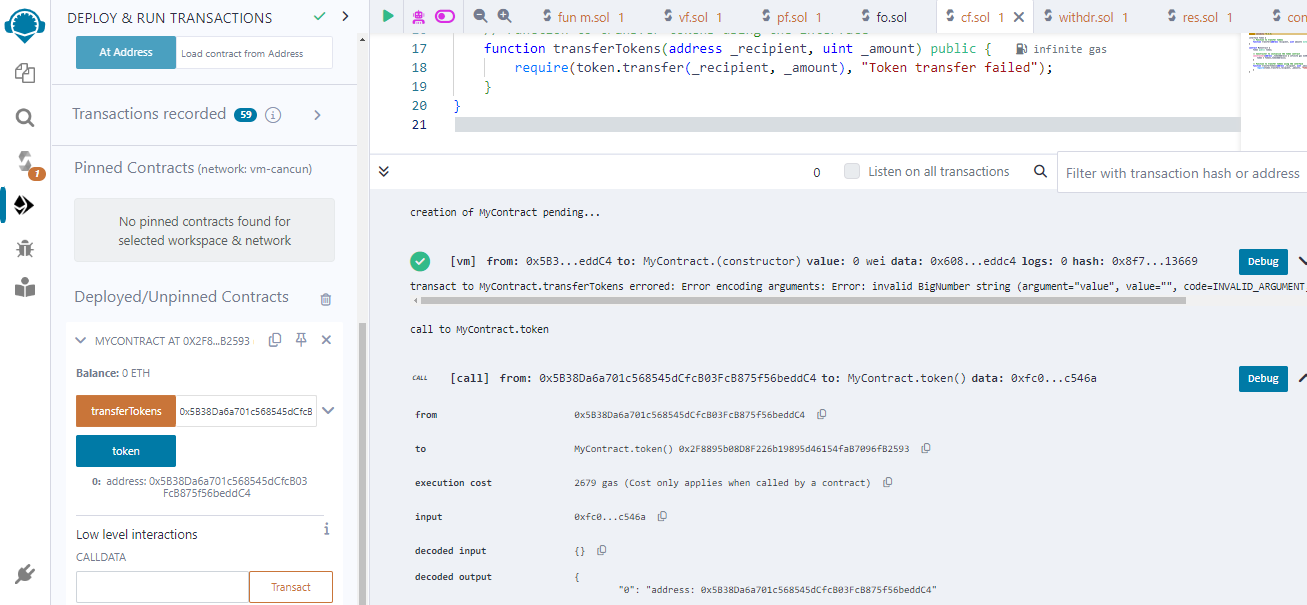
function transferTokens(address \_recipient, uint \_amount) public {

require(token.transfer(\_recipient, \_amount), "Token transfer failed");

}

}

**Output:**

****

**Practical:**

**Aim: Implement and demonstrate the use of the following in Solidity\_**

**Libraries, Assembly, Events, Error handling.**

**Libraries:**

**Code:**

pragma solidity ^0.8.0;

library MathLibrary {

function add(uint256 a, uint256 b) internal pure returns (uint256) {

return a + b;

}

function subtract(uint256 a, uint256 b) internal pure returns (uint256) {

return a - b;

}}

contract Calculator {

using MathLibrary for uint256;

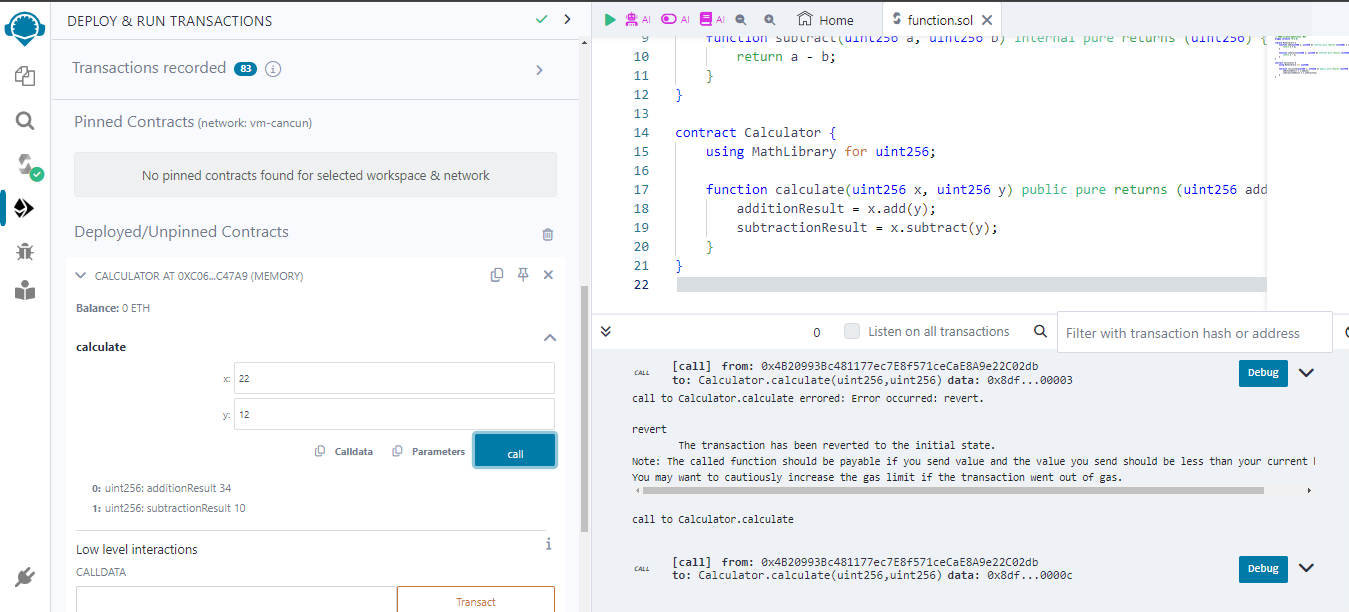
function calculate(uint256 x, uint256 y) public pure returns (uint256 additionResult, uint256 subtractionResult) {

additionResult = x.add(y);

subtractionResult = x.subtract(y);

}}

**Output:**

****

**Assembly:**

**Code:**

pragma solidity ^0.8.0;

contract SimpleArrayExample {

// Dynamic array of unsigned integers

uint[] public numbers;

// Function to add a number to the array

function addNumber(uint \_number) public {

numbers.push(\_number);

}

// Function to get the length of the array

function getLength() public view returns (uint) {

return numbers.length;

}

// Function to get the element at a specific index of the array

function getElement(uint \_index) public view returns (uint) {

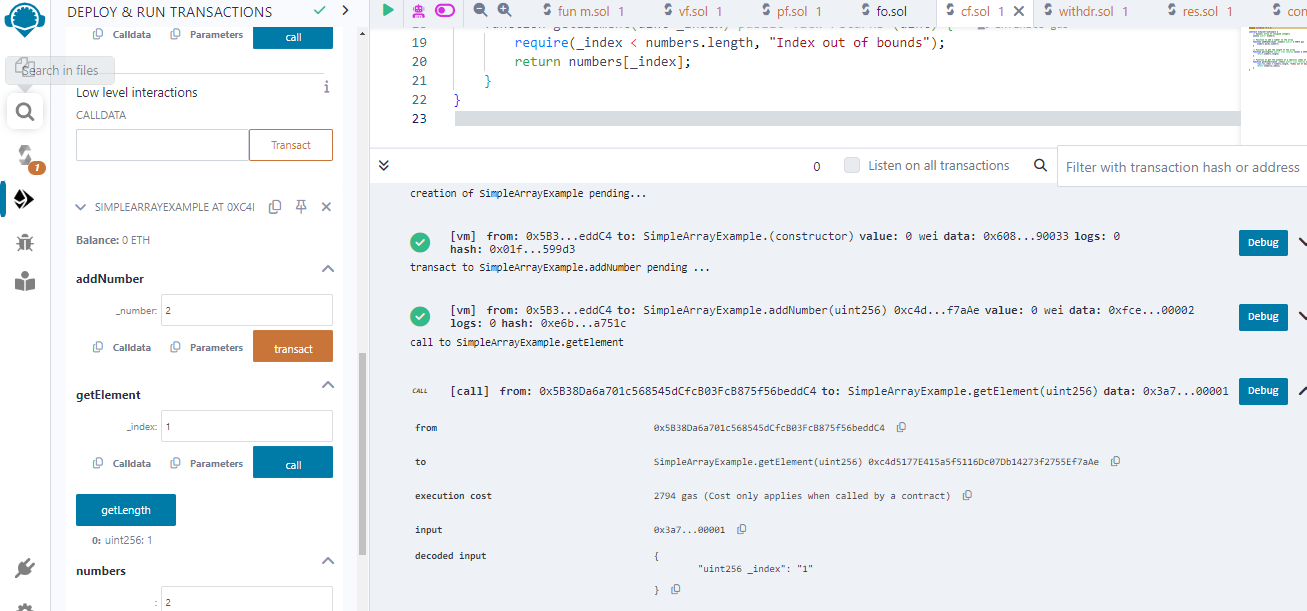
require(\_index < numbers.length, "Index out of bounds");

return numbers[\_index];

}

}

**Output:**

****

**Events:**

**Code:**

pragma solidity ^0.8.0;

contract EventExample {

// Define an event named `ValueChanged`

event ValueChanged(address indexed \_sender, uint \_newValue);

uint public value;

// Function to set the value and emit an event

function setValue(uint \_newValue) public {

value = \_newValue;

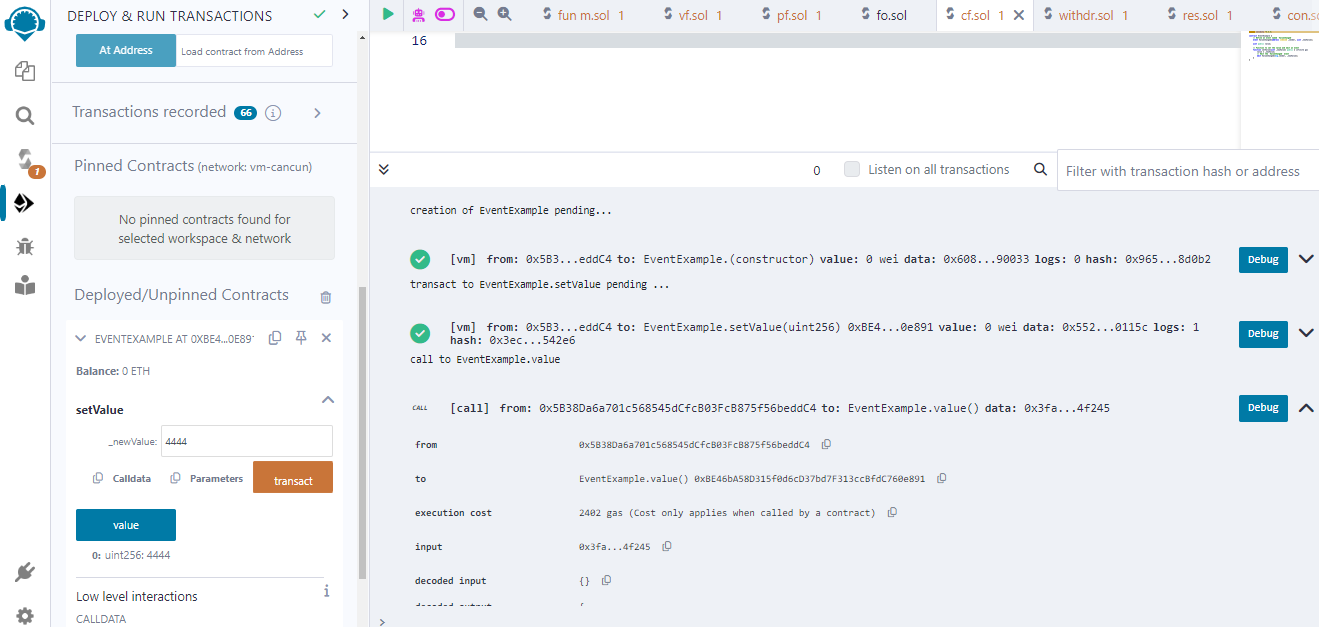
// Emit the `ValueChanged` event

emit ValueChanged(msg.sender, \_newValue);

}

}

**Output:**

****

**Error handling:**

**Code:**

pragma solidity ^0.8.0;

contract ErrorDemo {

function getSum(uint256 a, uint256 b) public pure returns (uint256) {

uint256 sum = a + b;

require(sum < 255, "Sum exceeds maximum value");

return sum;

}

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Practical:**

**Aim: Write a program to demonstrate mining of Ether.**

**Code:**

**code remix idle:**

**Step 1: open a notepad and paste the below code and save with .js Extension.**

const { Web3 } = require("web3");

const web3 = new Web3("http://127.0.0.1:7545");

async function mine() {

const accounts = await web3.eth.getAccounts();

const coinbaseacc1 = accounts[0];

const coinbaseacc2 = accounts[1];

console.log('Mining ether on Ganache with coinbase address: ${coinbaseacc1}');

while (true) {

try {

await web3.eth.sendTransaction({

from: coinbaseacc1,

to: coinbaseacc2,

value: web3.utils.toWei("0.05", "ether"),

});

console.log("Mined a new block!");

} catch (err) {

console.error(err);

}

}

}

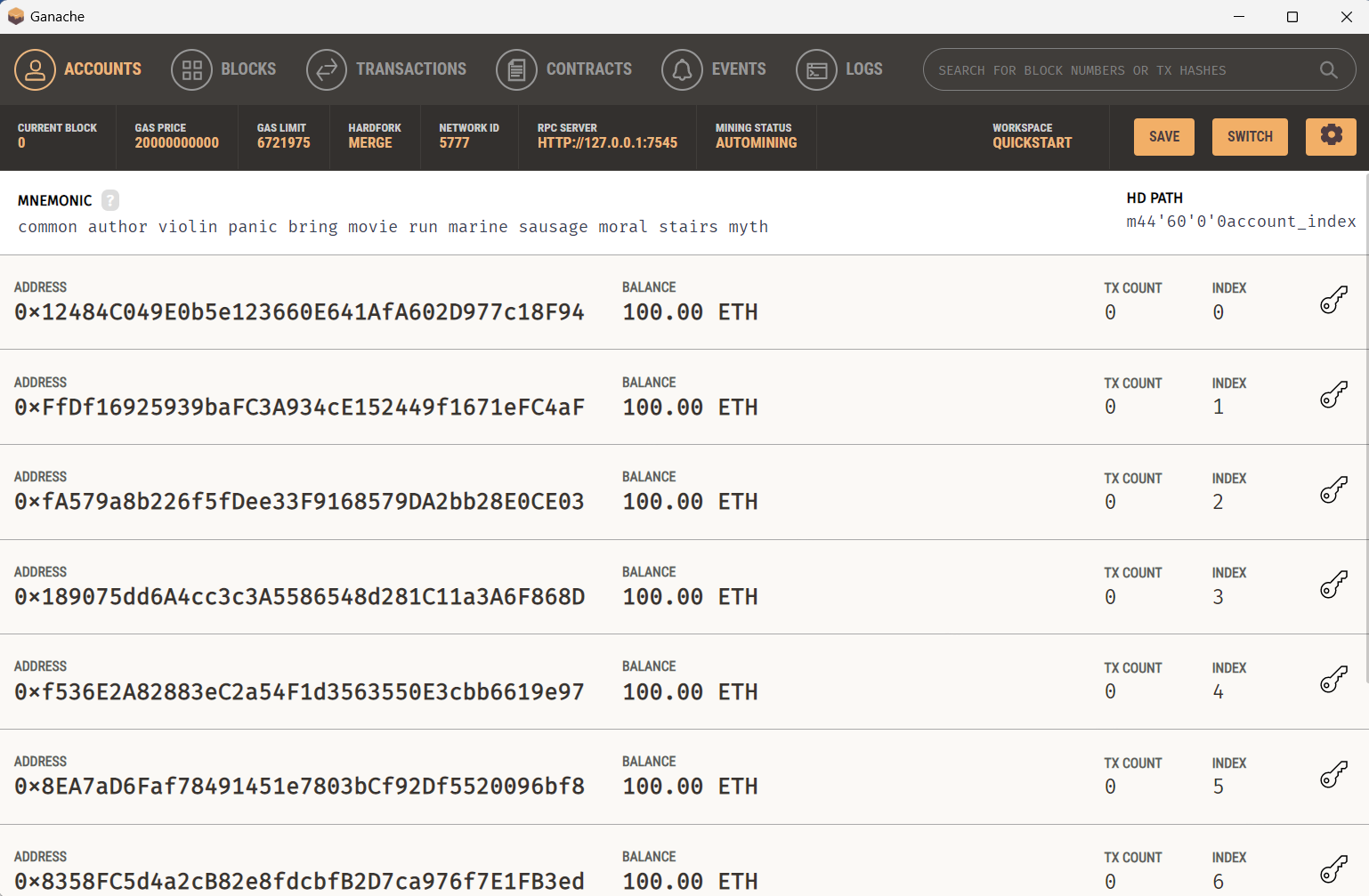
**Step 2: Install required package as in this practical required of “web3”**

* **npm install web3**

**A screenshot of a computer

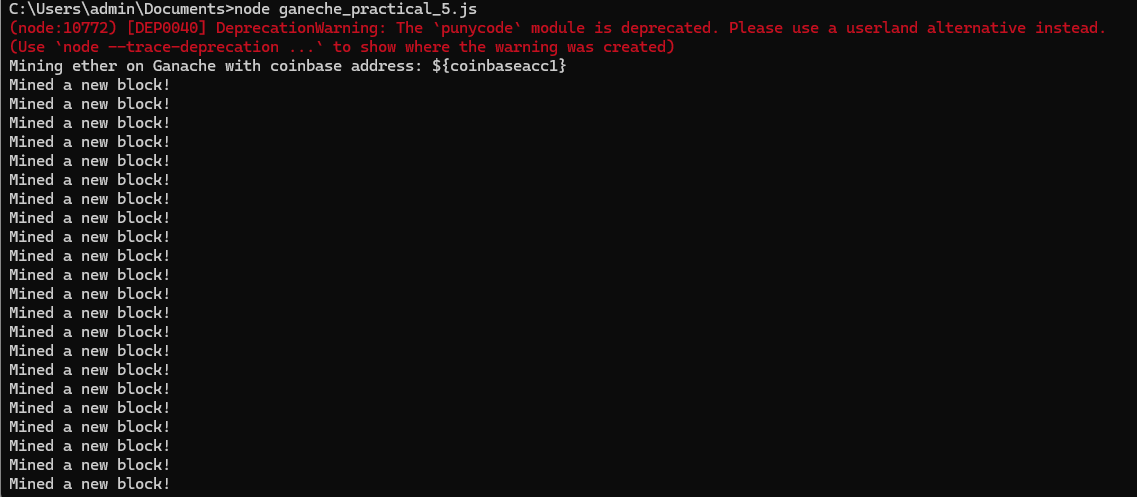
Description automatically generated**

**Step 3: Install Ganache application for this program after that open then you will get interface as below image.**

****

**Step 4: Follow the commandto run the program**

**Command: node ganeche\_practical\_5.js**

****

**Practical:**

**Aim: Demonstrate the running of the blockchain node.**

**Code:**

import hashlib

import time

class Block:

def \_\_init\_\_(self, index, timestamp, data, previous\_hash):

self.index = index

self.timestamp = timestamp

self.data = data

self.previous\_hash = previous\_hash

self.hash = self.calculate\_hash()

def calculate\_hash(self):

block\_string = str(self.index) + str(self.timestamp) + str(self.data) + str(self.previous\_hash)

return hashlib.sha256(block\_string.encode()).hexdigest()

class Blockchain:

def \_\_init\_\_(self):

self.chain = [self.create\_genesis\_block()]

def create\_genesis\_block(self):

return Block(0, int(time.time()), "Genesis Block", "0")

def add\_block(self, new\_block):

new\_block.previous\_hash = self.chain[-1].hash

new\_block.hash = new\_block.calculate\_hash()

self.chain.append(new\_block)

blockchain = Blockchain()

# Create a new block

block = Block(1, int(time.time()), "Transaction Data 1", blockchain.chain[-1].hash)

# Add the new block to the blockchain

blockchain.add\_block(block)

# Print the contents of the blockchain

for block in blockchain.chain:

print("Block #{}".format(block.index))

print("Timestamp: {}".format(block.timestamp))

print("Data: {}".format(block.data))

print("Hash: {}\n".format(block.hash))

**Output:**

**A screenshot of a computer

Description automatically generated**

**Practical:**

**Aim: Demonstrate the use of Bitcoin Core API.**

**For performing this code we need to download packages name as “bitcoin” as commented below.**

**Code:**

**#pip install bitcoin.**

**#** **pip install bitcoinlib.**

from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet6')

key1 = w.get\_key()

print('Wallet Address:',key1.address)

w.scan()

print(w.info())

**Output:**

****

**Practical:**

**Aim: Create your own blockchain and demonstrate its use.**

Create a javascript folder with the following code in any folder of your choice.

**Code:**

const SHA256 = require("crypto-js/sha256");

class Block {

constructor(index, timestamp, data, previousHash = "") {

this.index = index;

this.timestamp = timestamp;

this.data = data;

this.previousHash = previousHash;

this.hash = this.calculateHash();

}

calculateHash() {

return SHA256(

this.index +

this.previousHash +

this.timestamp +

JSON.stringify(this.data)

).toString();

}

}

class Blockchain {

constructor() {

this.chain = [this.createGenesisBlock()];

}

createGenesisBlock() {

return new Block(0, "21/04/2023", "Genesis Block", "0");

}

getLatestBlock() {

return this.chain[this.chain.length - 1];

}

addBlock(newBlock) {

newBlock.previousHash = this.getLatestBlock().hash;

newBlock.hash = newBlock.calculateHash();

this.chain.push(newBlock);

}

isChainValid() {

for (let i = 1; i < this.chain.length; i++) {

const currentBlock = this.chain[i];

const previousBlock = this.chain[i - 1];

if (currentBlock.hash !== currentBlock.calculateHash()) {

return false;

}

if (currentBlock.previousHash !== previousBlock.hash) {

return false;

}

}

return true;

}

}

// Blockchain Implementation

let myCoin = new Blockchain();

myCoin.addBlock(new Block(1, "22/04/2023", { amount: 4 }));

myCoin.addBlock(new Block(2, "22/04/2023", { amount: 8 }));

console.log('Is blockchain valid? ' + myCoin.isChainValid());

console.log(JSON.stringify(myCoin, null, 4));

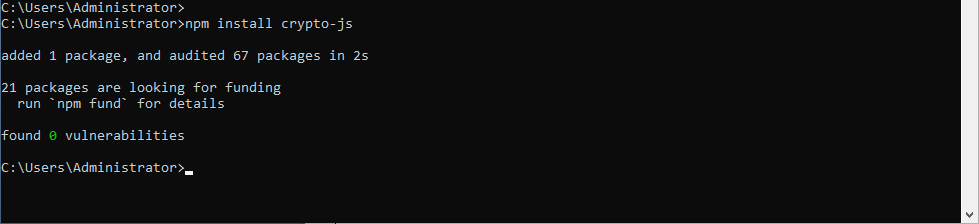
**Output:**

**Step 1->**Make sure you have installed nodejs in your system

****

**Step 2->**We need crypto –js node module to make our own blockchain. So

install it as following

****

**Step 3->**Run the above code in command line using command: node aunty\_practical\_7.js(in this case my file name is aunty\_practical\_7.js you replace the file name with your file name.)

