**PRACTICAL 1**

**AIM:** Write the following programs for Blockchain in Python:

**[A]** A simple client class that generates the private and public keys by using the builtin Python RSA algorithm and test it.

**CODE:**

import binascii

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Random

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode("ascii")

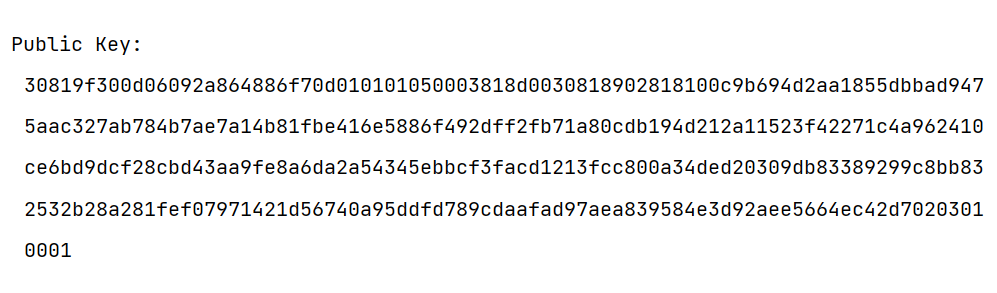
# Create an instance of the Client class

UDIT = Client()

# Print the public key (identity) of the client

print("\nPublic Key:", UDIT.identity)

**OUTPUT:**



**[B]** A transaction class to send and receive money and test it.

**CODE:**

import binascii

import collections

import datetime

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Random

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode("ascii")

class Transaction:

def \_\_init\_\_(self, sender, recipient, value):

self.sender = sender

self.recipient = recipient

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

identity = "Genesis" if self.sender == "Genesis" else self.sender.identity

return collections.OrderedDict(

{

"sender": identity,

"recipient": self.recipient,

"value": self.value,

"time": self.time,

}

)

def sign\_transaction(self):

private\_key = self.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode("utf8"))

return binascii.hexlify(signer.sign(h)).decode("ascii")

UDIT = Client()

UGC = Client()

t = Transaction(UDIT, UGC.identity, 5.0)

print("\nTransaction Recipient:\n", t.recipient) # print("\nTransaction Sender:\n", t.sender) print("\nTransaction Value:\n", t.value)

signature = t.sign\_transaction()

print("\nSignature:\n", signature)

**OUTPUT:**



**[C]** Create multiple transactions and display them.

**CODE:**

import binascii

import collections

import datetime

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Random

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode("ascii")

class Transaction:

def \_\_init\_\_(self, sender, recipient, value):

self.sender = sender

self.recipient = recipient

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

identity = "Genesis" if self.sender == "Genesis" else self.sender.identity

return collections.OrderedDict(

{

"sender": identity,

"recipient": self.recipient,

"value": self.value,

"time": self.time,

}

)

def sign\_transaction(self):

private\_key = self.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode("utf8"))

return binascii.hexlify(signer.sign(h)).decode("ascii")

def display\_transaction(transaction):

# for transaction in transactions:

dict = transaction.to\_dict()

print("sender: " + dict['sender'])

print('-----')

print("recipient: " + dict['recipient'])

print('-----')

print("value: " + str(dict['value']))

print('-----')

print("time: " + str(dict['time']))

print('-----')

UDIT = Client()

UGC = Client()

AICTE = Client()

MU = Client()

t1 = Transaction(UDIT, UGC.identity, 15.0)

t1.sign\_transaction()

transactions = [t1]

t2 = Transaction(UDIT, AICTE.identity, 6.0)

t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction(UGC, MU.identity, 2.0)

t3.sign\_transaction()

transactions.append(t3)

t4 = Transaction(AICTE, UGC.identity, 4.0)

t4.sign\_transaction()

transactions.append(t4)

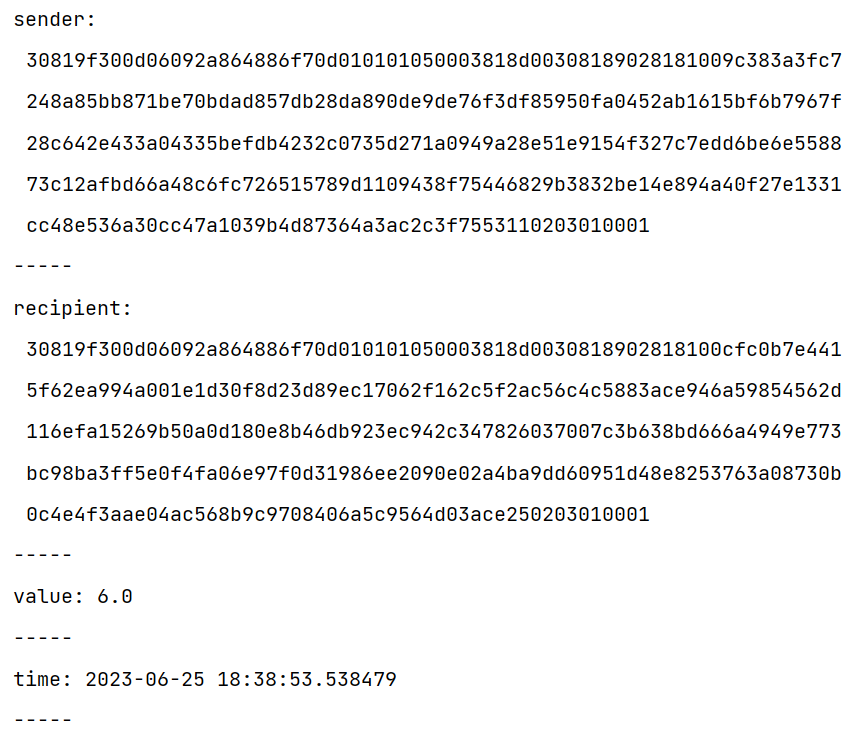
for transaction in transactions:

Transaction.display\_transaction(transaction)

print(" ")

**OUTPUT:**









**[D]** Create a blockchain, a genesis block and execute it.

**CODE:**

import binascii

import collections

import datetime

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Random

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode("ascii")

class Transaction:

def \_\_init\_\_(self, sender, recipient, value):

self.sender = sender

self.recipient = recipient

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

identity = "Genesis" if self.sender == "Genesis" else self.sender.identity

return collections.OrderedDict(

{

"sender": identity,

"recipient": self.recipient,

"value": self.value,

"time": self.time,

}

)

def sign\_transaction(self):

private\_key = self.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode("utf8"))

return binascii.hexlify(signer.sign(h)).decode("ascii")

def display\_transaction(transaction):

# for transaction in transactions:

dict = transaction.to\_dict()

print("sender: " + dict['sender'])

print('-----')

print("recipient: " + dict['recipient'])

print('-----')

print("value: " + str(dict['value']))

print('-----')

print("time: " + str(dict['time']))

print('-----')

class Block:

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

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

self.client = client

def dump\_blockchain(blocks):

print(f"\nNumber of blocks in the chain: {len(blocks)}")

for i, block in enumerate(blocks):

print(f"block # {i}")

for transaction in block.verified\_transactions:

Transaction.display\_transaction(transaction)

print(" ")

print(" ")

UDIT = Client()

t0 = Transaction("Genesis", UDIT.identity, 500.0)

block0 = Block(UDIT)

block0.previous\_block\_hash = ""

NONCE = None

block0.verified\_transactions.append(t0)

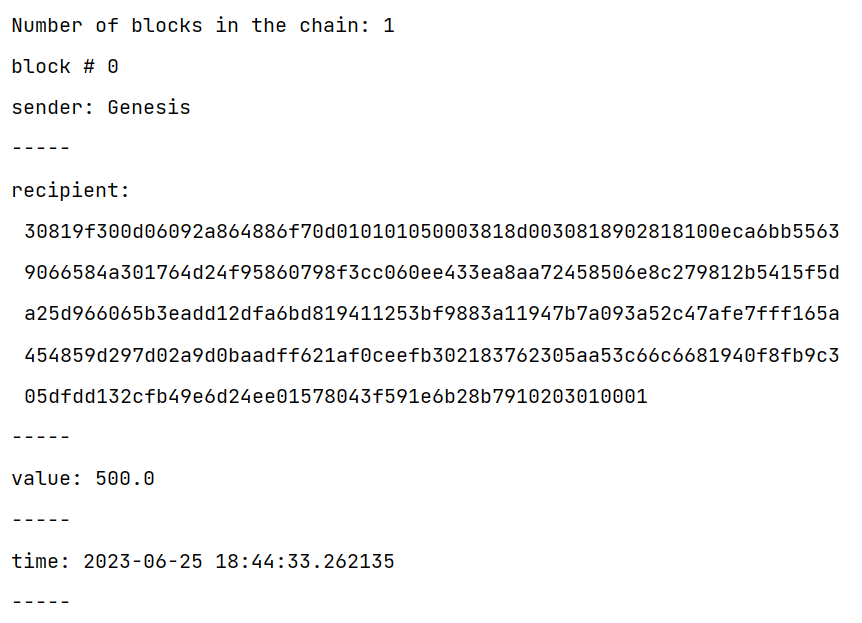
digest = hash(block0)

last\_block\_hash = digest

TPCoins = [block0]

dump\_blockchain(TPCoins)

**OUTPUT:**



**[E]** Create a mining function and test it.

**CODE:**

import hashlib

def sha256(message):

return hashlib.sha256(message.encode("ascii")).hexdigest()

def mine(message, difficulty=1):

assert difficulty >= 1

prefix = "1" \* difficulty

for i in range(1000):

digest = sha256(str(hash(message)) + str(i))

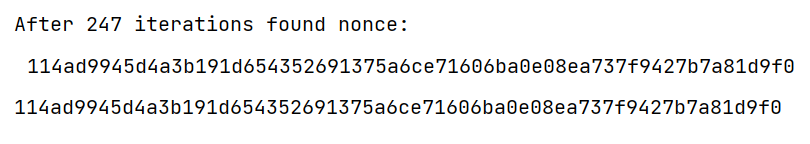
if digest.startswith(prefix):

print(f"After {str(i)} iterations found nonce: {digest}")

return digest

print(mine("test message", 2))

**OUTPUT:**



**[F]** Add blocks to the miner and dump the blockchain.

**CODE:**

import datetime

import hashlib

class Block:

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

self.timestamp = datetime.datetime.now(datetime.timezone.utc)

self.data = data

self.previous\_hash = previous\_hash

self.hash = self.calc\_hash()

def calc\_hash(self):

sha = hashlib.sha256()

hash\_str = self.data.encode("utf-8")

sha.update(hash\_str)

return sha.hexdigest()

blockchain = [Block("First block", "0")]

blockchain.append(Block("Second block", blockchain[0].hash))

blockchain.append(Block("Third block", blockchain[1].hash))

# Dumping the blockchain

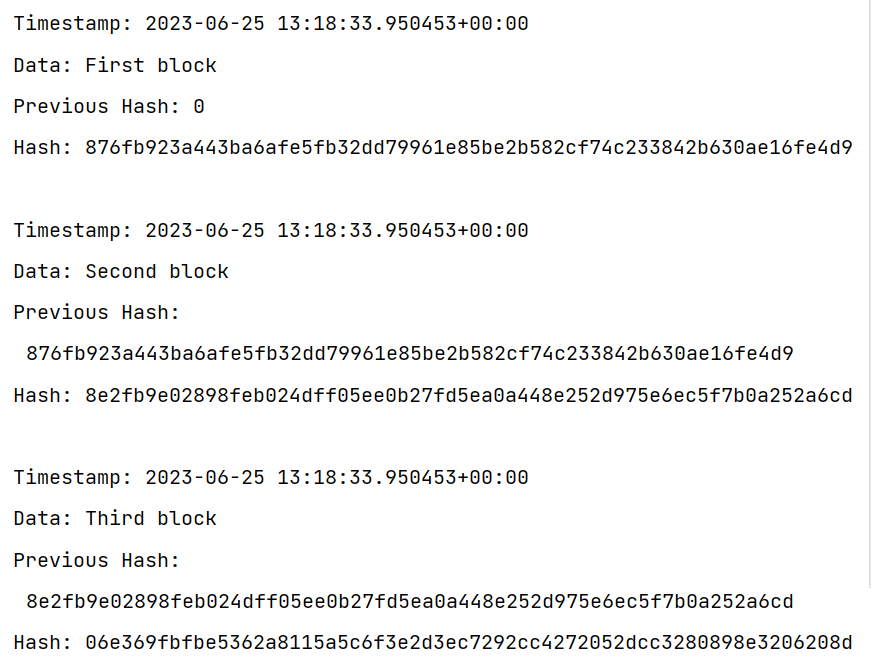
for block in blockchain:

print(

f"Timestamp: {block.timestamp}\nData: {block.data}\nPrevious Hash: {block.previous\_hash}\nHash: {block.hash}\n"

)

**OUTPUT:**



**PRACTICAL 2**

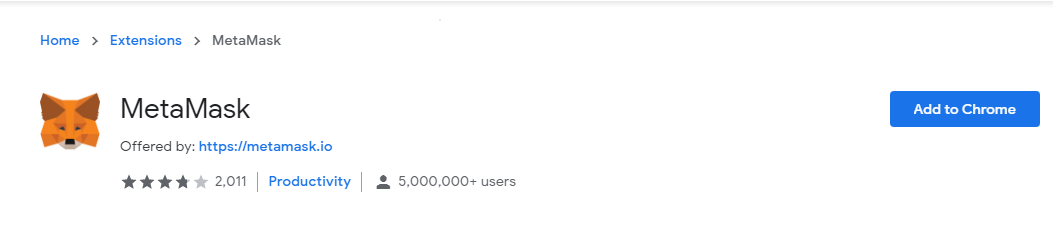
**AIM:** Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.

**Step 1:** Go to [Chrome Web Store Extensions Section](https://chrome.google.com/webstore/category/extensions).

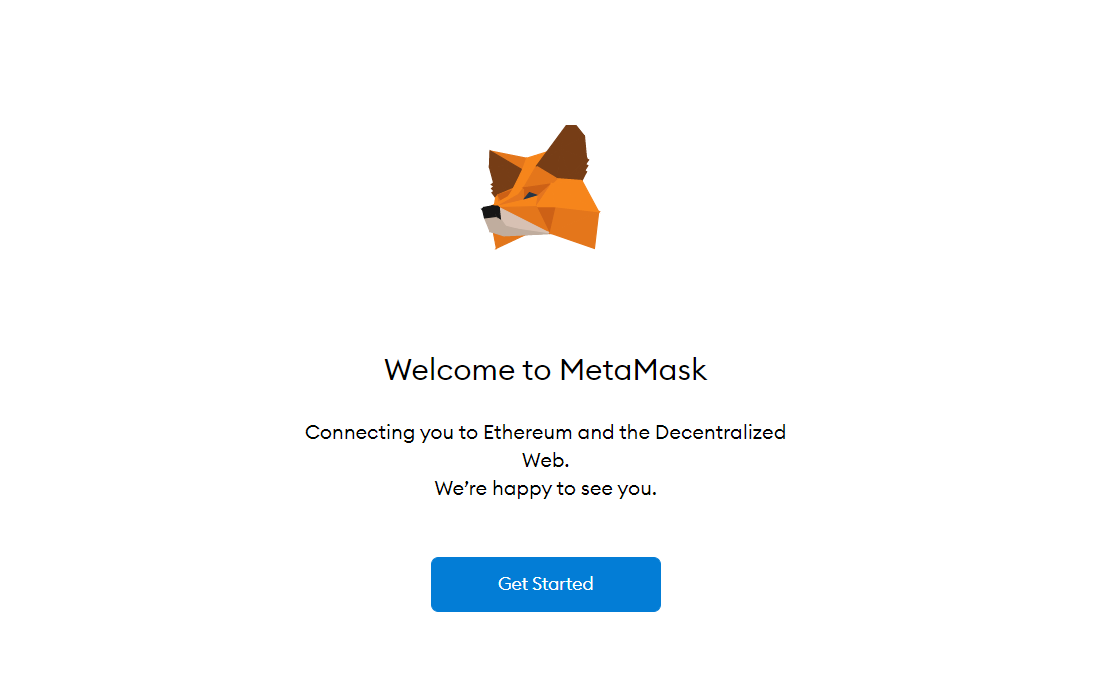
**Step 2:** Search MetaMask.

**Step 3:** Check the number of downloads to make sure that the legitimate MetaMask is being installed, as hackers might try to make clones of it.

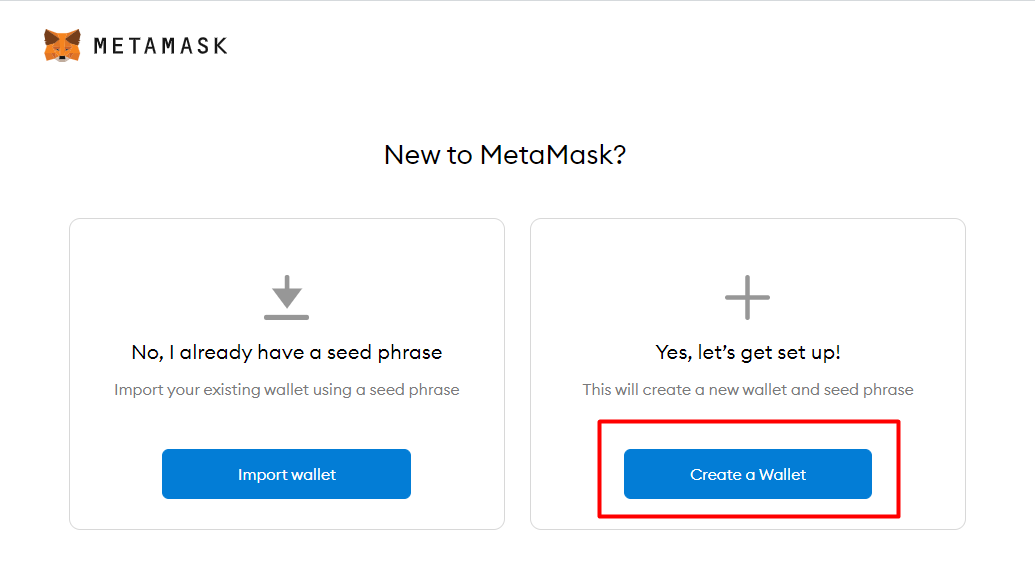
**Step 4:** Click the Add to Chrome button.



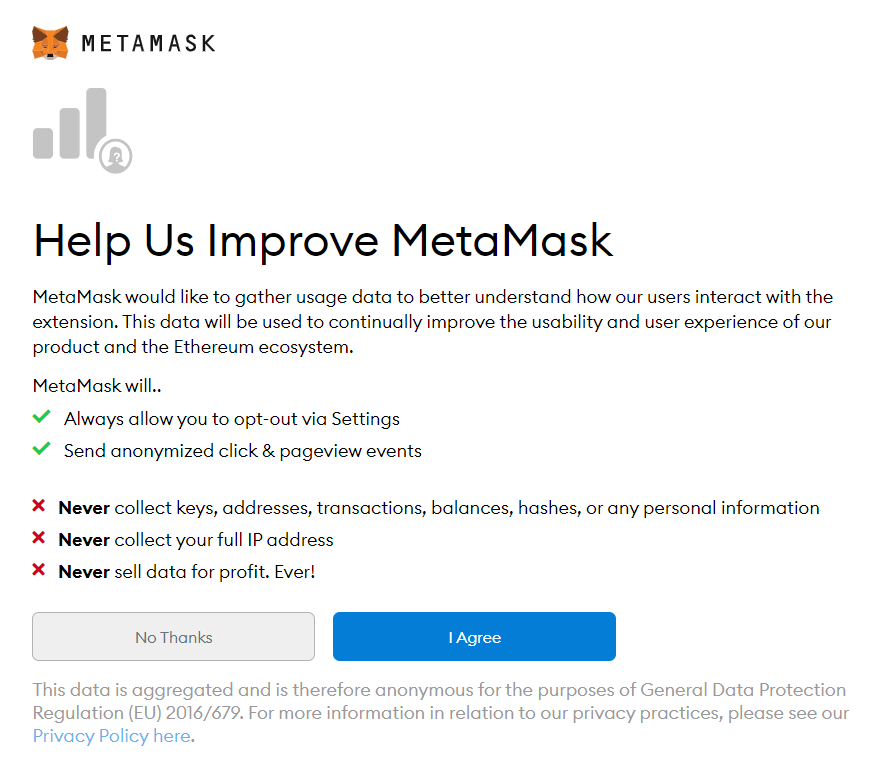
**Step 5:** Once installation is complete this page will be displayed. Click on the Get Started button.



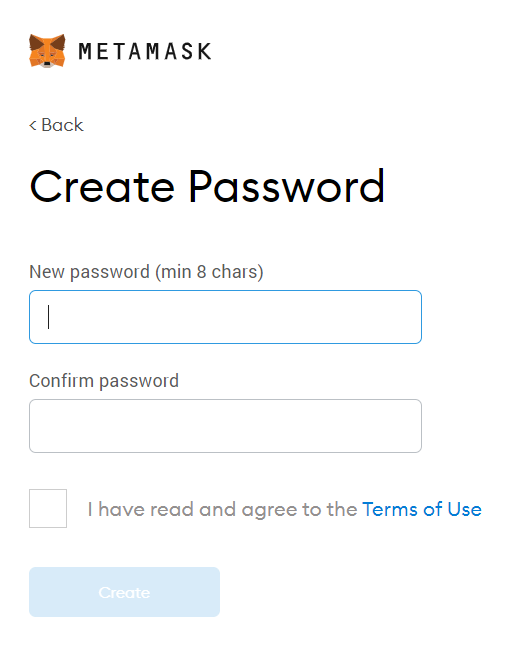
**Step 6:** This is the first time creating a wallet, so click the Create a Wallet button. If there is already a wallet then import the already created using the Import Wallet button.



**Step 7:** Click I Agree button to allow data to be collected to help improve MetaMask or else click the No Thanks button. The wallet can still be created even if the user will click on the No thanks Button.

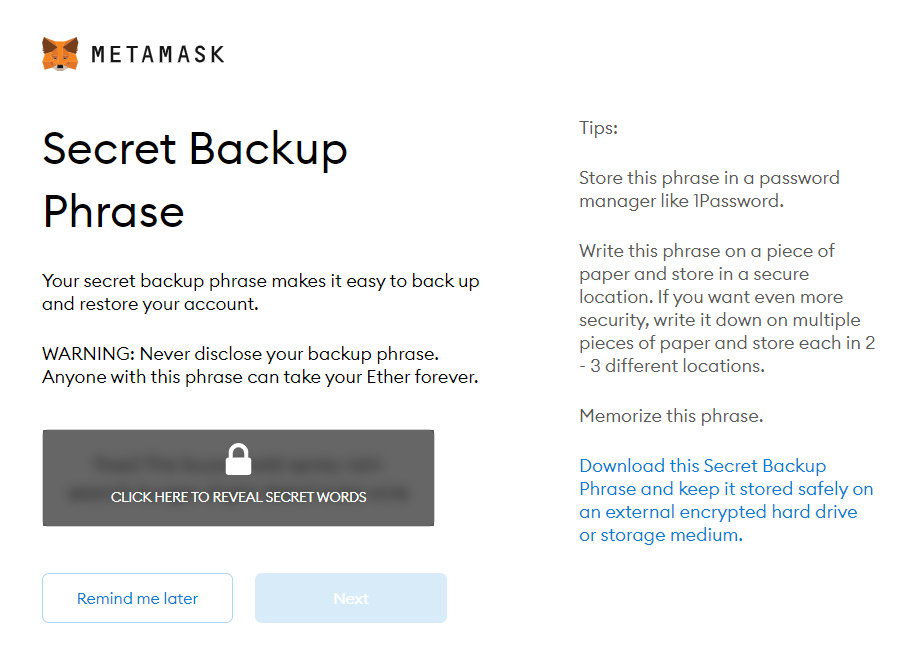


**Step 8:** Create a password for your wallet. This password is to be entered every time the browser is launched and wants to use MetaMask. A new password needs to be created if chrome is uninstalled or if there is a switching of browsers. In that case, go through the Import Wallet button. This is because MetaMask stores the keys in the browser. Agree to Terms of Use.

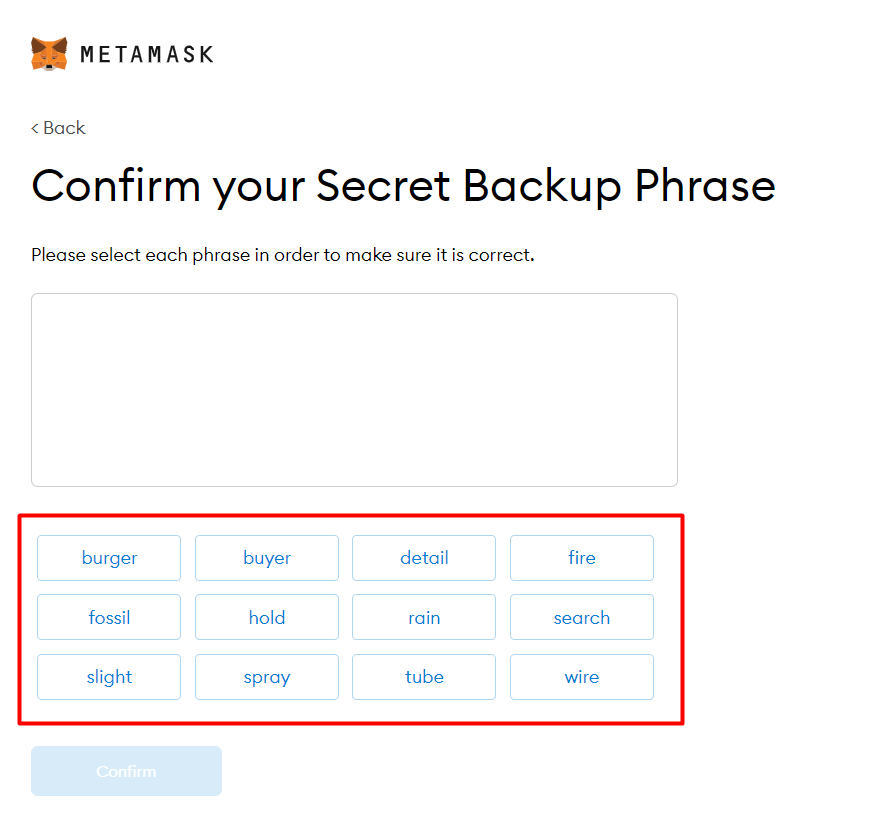


**Step 9:** Click on the dark area which says Click here to reveal secret words to get your secret phrase.

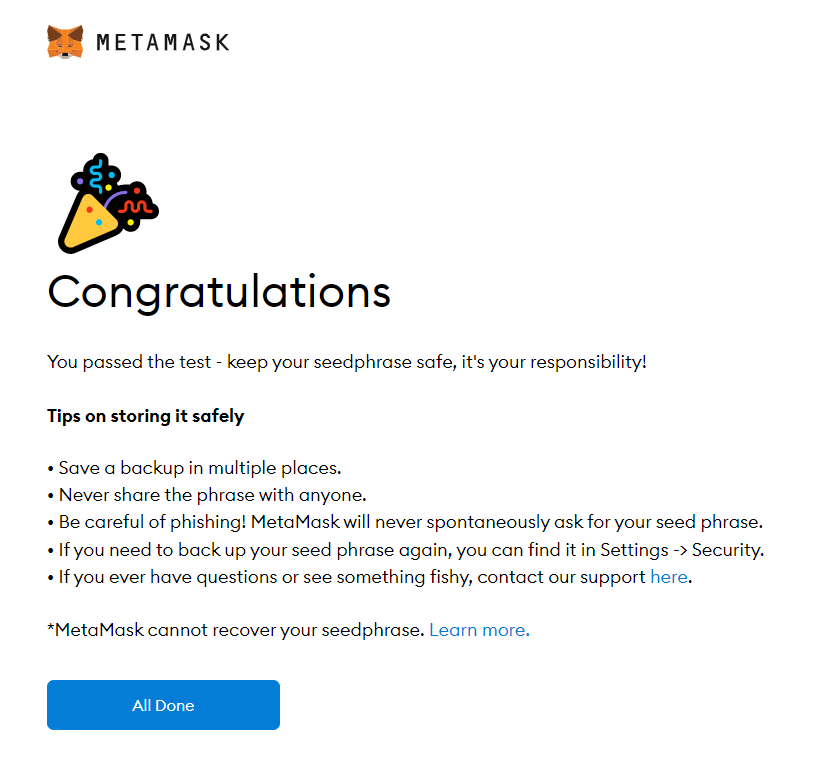
**Step 10:** This is the most important step. Back up your secret phrase properly. Do not store your secret phrase on your computer. Please read everything on this screen until you understand it completely before proceeding. The secret phrase is the only way to access your wallet if you forget your password. Once done click the Next button.



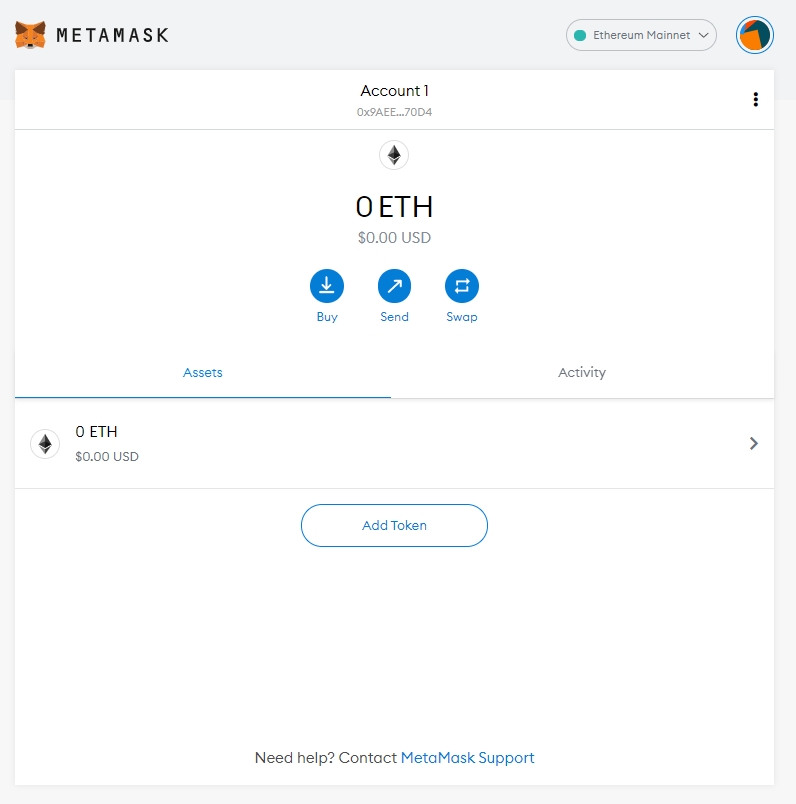
**Step 11:** Click the buttons respective to the order of the words in your seed phrase. In other words, type the seed phrase using the button on the screen. If done correctly the Confirm button should turn blue.



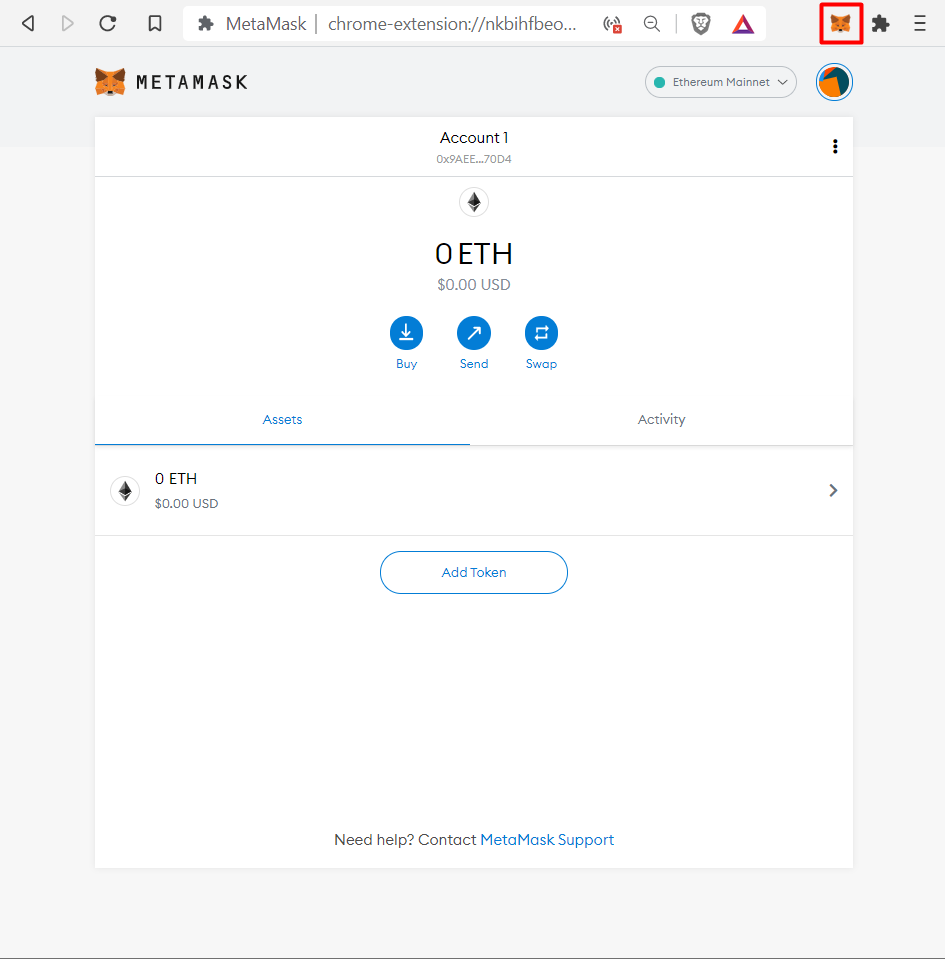
**Step 12:** Click the Confirm button. Please follow the tips mentioned.



**Step 13:** One can see the balance and copy the address of the account by clicking on the Account 1 area.



**Step 14:** One can access MetaMask in the browser by clicking the Foxface icon on the top right. If the Foxface icon is not visible, then click on the puzzle piece icon right next to it.

****

**PRACTICAL 3**

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

**[A] Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs,**

**Mappings, Conversions, Ether Units, Special Variables.**

**[I] Variable**

**[i] State Variable**

**CODE:**

//Solidity program to demonstrate state variables

pragma solidity ^0.5.0;

// Creating a contract

contract var\_Test

{

// Declaring a state variable

uint8 public state\_var;

// Defining a constructor

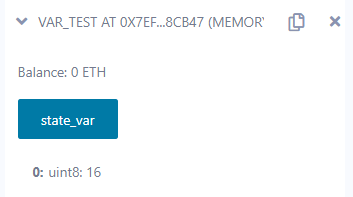
constructor() public {

state\_var = 16;

}

}

**OUTPUT:**



**[ii] Local Variable**

**CODE:**

//Solidity program to demonstrate Local variables

pragma solidity ^0.5.0;

// Creating a contract

contract local\_var\_Test

{

// Defining function to show the declaration and

// scope of Local variables

function acsess\_local\_variable() public pure returns(uint) {

// Initializing Local variables

uint a = 10;

uint b = 40;

uint sum = a + b;

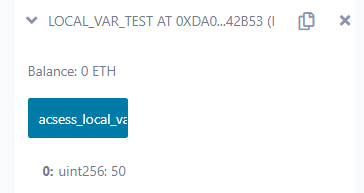
// Access the Local variable

return sum;

}

}

**OUTPUT:**



**[iii] Global Variable**

**CODE:**

//Solidity program to show Global variables

pragma solidity ^0.5.0;

// Creating a contract

contract globalTest

{

// Defining a variable

address public admin;

// Creating a constructor to

// use Global variable

constructor() public

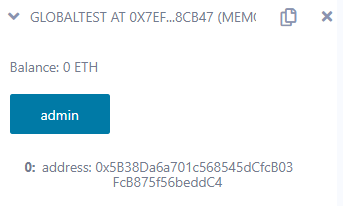
{

admin = msg.sender;

}

}

**OUTPUT:**



**[II] Operators**

**CODE:**

pragma solidity ^0.4.0;

contract Operators {

uint256 result = 0;

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

return a + b;

}

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

return a - b;

}

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

return a / b;

}

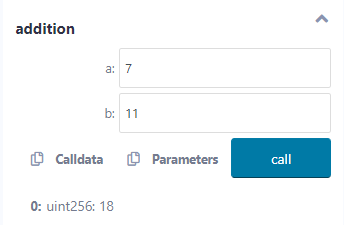
function multiply(uint256 a, uint256 b) public pure returns (uint256) {

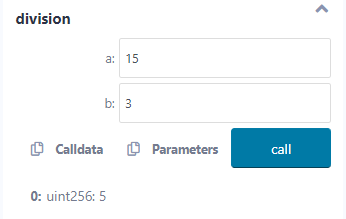
return a \* b;

}

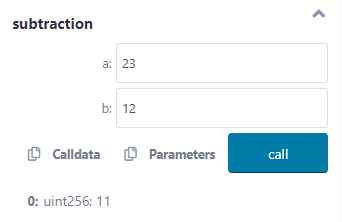
}

**OUTPUT:**









**[III] Loops**

**CODE:**

pragma solidity ^0.5.0;

contract LoopingTest {

uint256 storedData;

constructor() public {

storedData = 10;

}

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

uint256 a = 10;

uint256 b = 2;

uint256 result = a + b;

return integerToString(result);

}

function integerToString(uint256 \_i) internal pure returns (string memory) {

if (\_i == 0) {

return "0";

}

uint256 j = 0;

uint256 len;

for (j = \_i; j != 0; j /= 10) {

//for loop example

len++;

}

bytes memory bstr = new bytes(len);

uint256 k = len - 1;

while (\_i != 0) {

bstr[k--] = bytes1(uint8(48 + (\_i % 10)));

\_i /= 10;

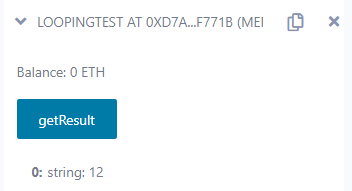
}

return string(bstr); //access local variable

}

}

**OUTPUT:**



**[IV] Decision Making**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.1;

// Creating a contract

contract Types {

// Declaring state variables

uint256 i = 10;

string result;

function decision\_making() public payable returns (string memory) {

if (i < 10) {

result = "less than 10";

}

else if (i == 10) {

result = "equal to 10";

}

else {

result = "greater than 10";

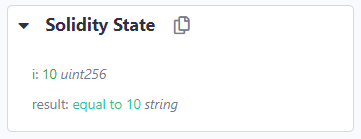
}

return result;

}

}

**OUTPUT:**



**[V] Strings**

**[i] Double Quotes**

**CODE:**

pragma solidity ^0.5.0;

contract testString

{

function test\_string() public pure returns (string memory)

{

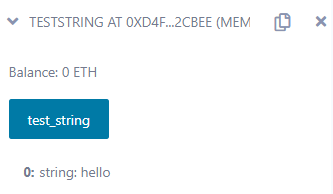
string memory a = "hello";

return a;

}

}

**OUTPUT:**



**[ii] Single Quotes**

**CODE:**

pragma solidity ^0.5.2;

contract stringTest

{

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

uint a = 25;

uint b = 25;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure returns (string memory){

if (\_i == 0)

{

return "0";

}

uint j = \_i;

uint len;

while (j != 0)

{

len++;

j/= 10;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0)

{

bstr[k-- ] = byte(uint8(48 + \_i % 10));

\_i /= 10;

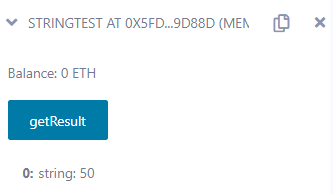
}

return string(bstr);

}

}

**OUTPUT:**



**[VI] Arrays**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity 0.7.0;

contract Arrays {

function initArray() public pure returns (uint256) {

uint128[3] memory array = [1, 2, uint128(3)];

return array[0];

}

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

uint128[] memory array = new uint128[](a);

uint128 val = 5;

for (uint128 j = 0; j < a; j++) {

array[j] = j \* val;

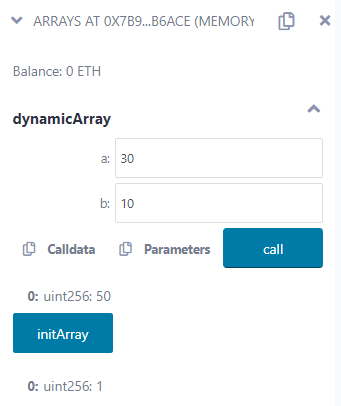
}

return array[b];

}

}

**OUTPUT:**



**[VII] Enums**

**CODE:**

pragma solidity ^0.5.0;

contract enumTest {

enum FreshJuiceSize {

SMALL,

MEDIUM,

LARGE

}

FreshJuiceSize choice;

FreshJuiceSize constant defaultChoice = FreshJuiceSize.MEDIUM;

function setLarge() public {

choice = FreshJuiceSize.LARGE;

}

function getChoice() public view returns (FreshJuiceSize) {

return choice;

}

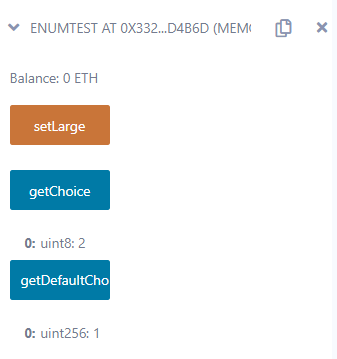
function getDefaultChoice() public pure returns (uint256) {

return uint256(defaultChoice);

}

}

**OUTPUT:**



**[VIII] Structs**

**STEPS:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.1;

contract test {

struct Book {

string title;

string author;

uint book\_id;

}

Book book;

function setBook() public {

book = Book('Learn Java', 'TP', 1);

}

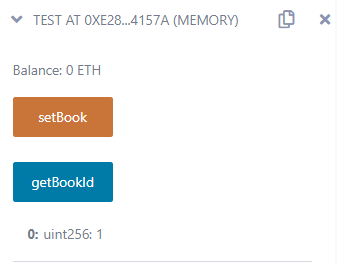
function getBookId() public view returns (uint) {

return book.book\_id;

}

}

**OUTPUT:**



**[IX] Mappings**

**CODE:**

pragma solidity ^0.5.0;

contract LedgerBalance

{

mapping(address => uint) public balances;

function updateBalance(uint newBalance) public

{

balances[msg.sender] = newBalance;

}

}

contract Updater

{

function updateBalance() public returns (uint)

{

LedgerBalance ledgerBalance = new LedgerBalance();

ledgerBalance.updateBalance(10);

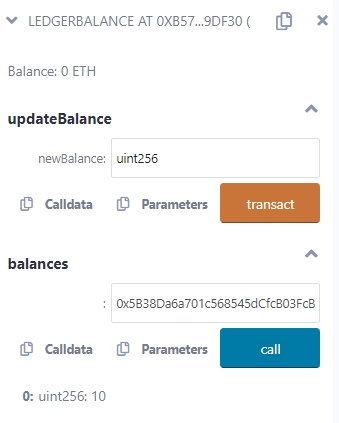
return ledgerBalance.balances(address(this));

}

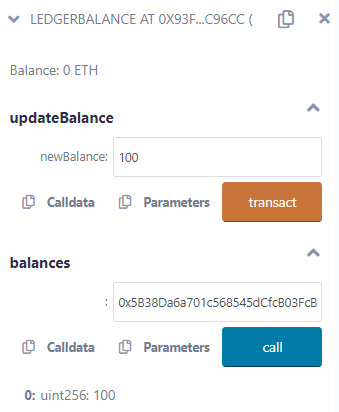
}

**OUTPUT:**

With Original Balance: 10



With Updated Balance: 100



**[X] Conversions**

**CODE:**

pragma solidity ^0.4.0;

contract Conversions {

function intToUint(int8 a) public pure returns (uint256) {

uint256 b = uint256(a);

return b;

}

function uint32ToUint16(uint32 a) public pure returns (uint16) {

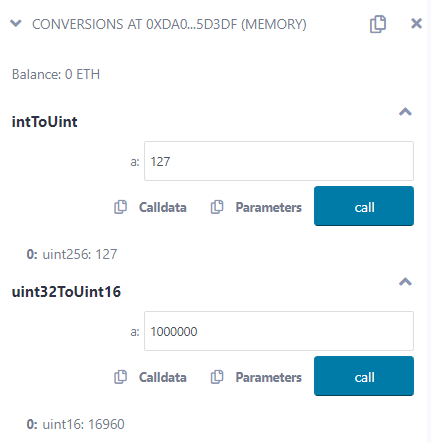
uint16 b = uint16(a);

return b;

}

}

**OUTPUT:**



**[XI] Ether Units**

**CODE:**

pragma solidity ^0.4.6;

contract SimpleStorage {

uint256 storedData = 2;

function set(uint256 x) public {

storedData = x;

/\*

Ether Units

Wei

Finney

Szabo

Ether

\*/

if (2000000000000000000 == 2 ether) {

storedData = 2;

}

else {

storedData = 3;

}

/\*

Time Units

seconds

minutes

hours

days

weeks

month

years

\*/

if (120 seconds == 2 minutes) {

storedData = 6;

}

else {

storedData = 9;

}

}

function get() constant public returns (uint256 retVal)

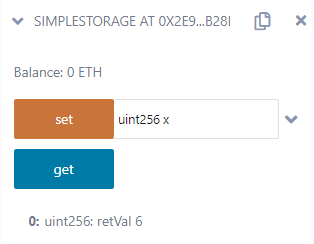
{

return storedData;

}

}

**OUTPUT:**



**[XII] Special Variables**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.1;

contract LedgerBalance {

mapping(address => uint256) public balances;

function updateBalance(uint256 newBalance) public {

balances[msg.sender] = newBalance;

}

}

contract Updater {

function updateBalance() public returns (uint256) {

LedgerBalance ledgerBalance = new LedgerBalance();

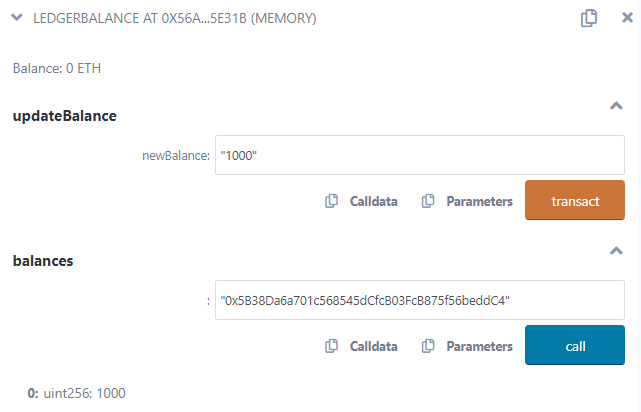
ledgerBalance.updateBalance(10);

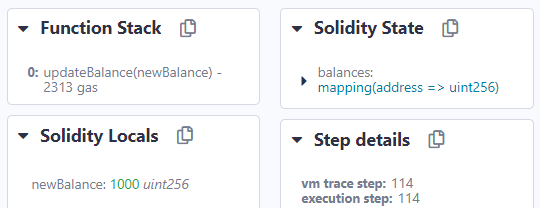
return ledgerBalance.balances(address(this));

}

}

**OUTPUT:**





**[B] Functions, Function Modifiers, View Functions, Pure Functions, Fallback Function,**

**Function Overloading, Mathematical functions, Cryptographic functions.**

**[I] Functions**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Test {

function return\_example()

public

pure

returns (

uint256,

uint256,

uint256,

string memory

)

{

uint256 num1 = 10;

uint256 num2 = 16;

uint256 sum = num1 + num2;

uint256 prod = num1 \* num2;

uint256 diff = num2 - num1;

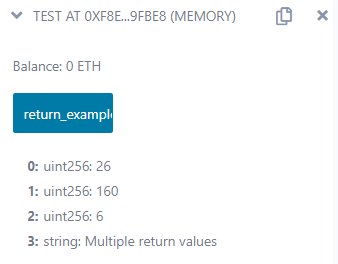
string memory message = "Multiple return values";

return (sum, prod, diff, message);

}

}

**OUTPUT:**



**[II] Function Modifiers**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract ExampleContract {

address public owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;

uint256 public counter;

modifier onlyowner() {

require(msg.sender == owner, "Only the contract owner can call");

\_;

}

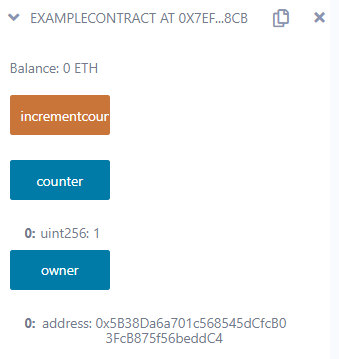
function incrementcounter() public onlyowner {

counter++;

}

}

**OUTPUT:**



**[III] View Functions**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract view\_demo {

uint256 num1 = 2;

uint256 num2 = 4;

function getResult() public view returns (uint256 product, uint256 sum) {

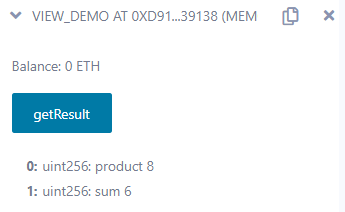
product = num1 \* num2;

sum = num1 + num2;

}

}

**OUTPUT:**



**[IV] Pure Functions**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract pure\_demo {

function getResult() public pure returns (uint256 product, uint256 sum) {

uint256 num1 = 2;

uint256 num2 = 4;

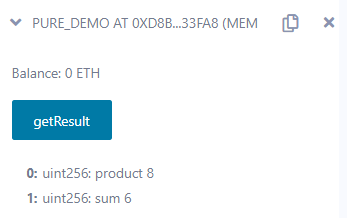
product = num1 \* num2;

sum = num1 + num2;

}

}

**OUTPUT:**



**[V] Fallback Function**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract A {

uint256 n;

function set(uint256 value) external {

n = value;

}

function() external payable {

n = 0;

}

}

contract example {

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

(bool success, ) = address(a).call(abi.encodeWithSignature("setter()"));

require(success);

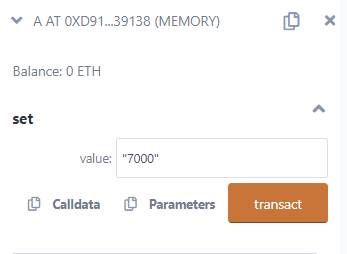
address payable payableA = address(uint160(address(a)));

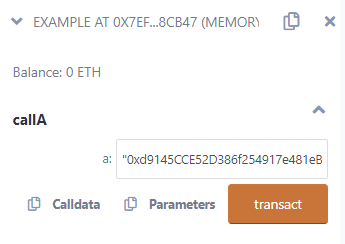
return (payableA.send(2 ether));

}

}

**OUTPUT:**





**[VI] Function Overloading**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract OverloadingExample {

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

return a + b;

}

function add(string memory a, string memory b)

public

pure

returns (string memory)

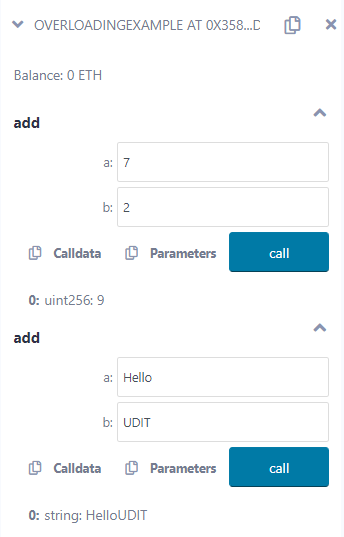
{

return string(abi.encodePacked(a, b));

}

}

**OUTPUT:**



**[VII] Mathematical Functions**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Test{ function CallAddMod() public pure returns(uint){

return addmod(7,3,3);

}

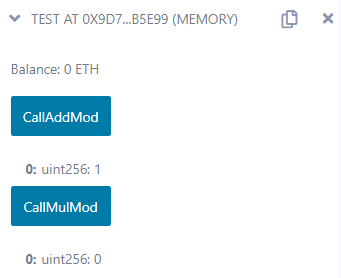
function CallMulMod() public pure returns(uint){

return mulmod(7,3,3);

}

}

**OUTPUT:**



**[VIII] Cryptographic Functions**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Test{ function callKeccak256() public pure returns(bytes32 result){

return keccak256("BLOCKCHAIN");

}

function callsha256() public pure returns(bytes32 result){

return sha256("BLOCKCHAIN");

}

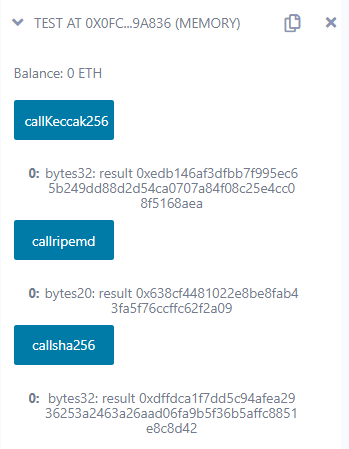
function callripemd() public pure returns (bytes20 result){

return ripemd160("BLOCKCHAIN");

}

}

**OUTPUT:**



**PRACTICAL 4**

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

**[A] Withdrawal Pattern, Restricted Access.**

**[I] Withdrawal Pattern**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract WithdrawalPattern {

address public owner;

uint256 public lockedbalance;

uint256 public withdrawablebalance;

constructor() {

owner = msg.sender;

}

modifier onlyowner() {

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

\_;

}

function deposit(uint256 amount) public payable {

require(amount > 0, "Amount must be greater than zero");

lockedbalance += amount;

}

function withdraw(uint256 amount) public payable onlyowner {

require(

amount <= withdrawablebalance,

"Insufficient withdrawable balance"

);

withdrawablebalance -= amount;

payable(msg.sender).transfer(amount);

}

function unlock(uint256 amount) public onlyowner {

require(amount <= lockedbalance, "Insufficient locked balance");

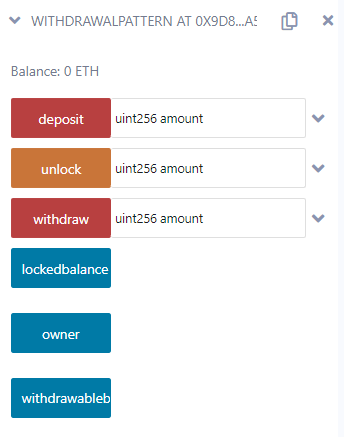
lockedbalance -= amount;

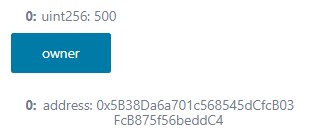
withdrawablebalance += amount;

}

}

**OUTPUT:**

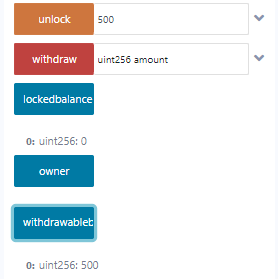
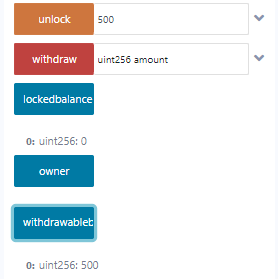
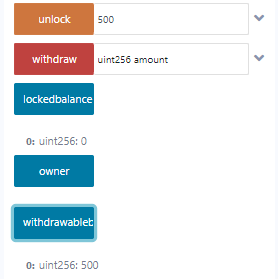
**Step 1:** Click on owner to create the owner object.

**Step 2:** Enter an amount and click on deposit.

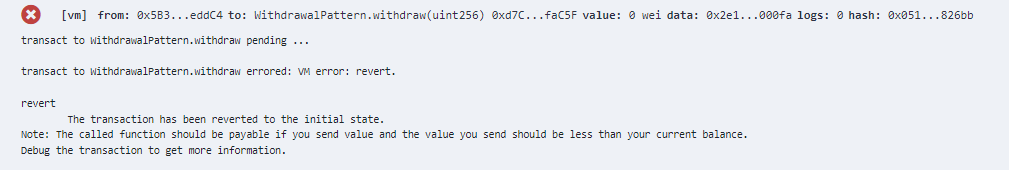
**Step 3:** Click on locked balance button to display the locked amount in the account.

**Step 4:** Click on withdrawable balance button.

**Step 5:** Click on unlock button and enter any amount to transfer amount to withdrawable balance. Check locked balance and withdrawable balance.

**Step 6:** Enter any amount you want to withdraw and click the withdraw button. You should get an error and the transaction should be reverted.



****

**[II] Restricted Access**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract AccessRestriction {

address public owner = msg.sender;

uint public lastOwnerChange = now;

modifier onlyBy(address \_account) {

require(msg.sender == \_account);

\_;

}

modifier onlyAfter(uint \_time) {

require(now >= \_time);

\_;

}

modifier costs(uint \_amount) {

require(msg.value >= \_amount);

\_;

if (msg.value > \_amount) {

msg.sender.transfer(msg.value - \_amount);

}

}

function changeOwner(address \_newOwner) public onlyBy(owner) {

owner = \_newOwner;

}

function buyContract() public payable onlyAfter(lastOwnerChange + 4 weeks) costs(1 ether) {

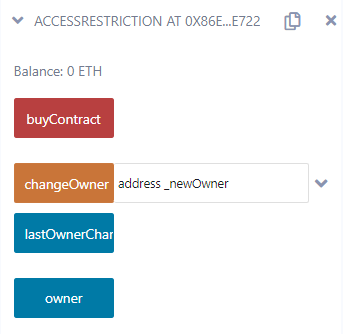
owner = msg.sender;

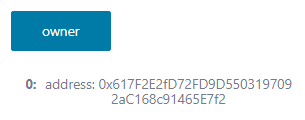
lastOwnerChange = now;

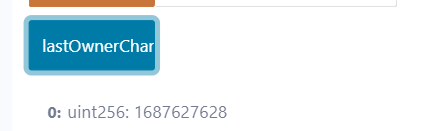
}

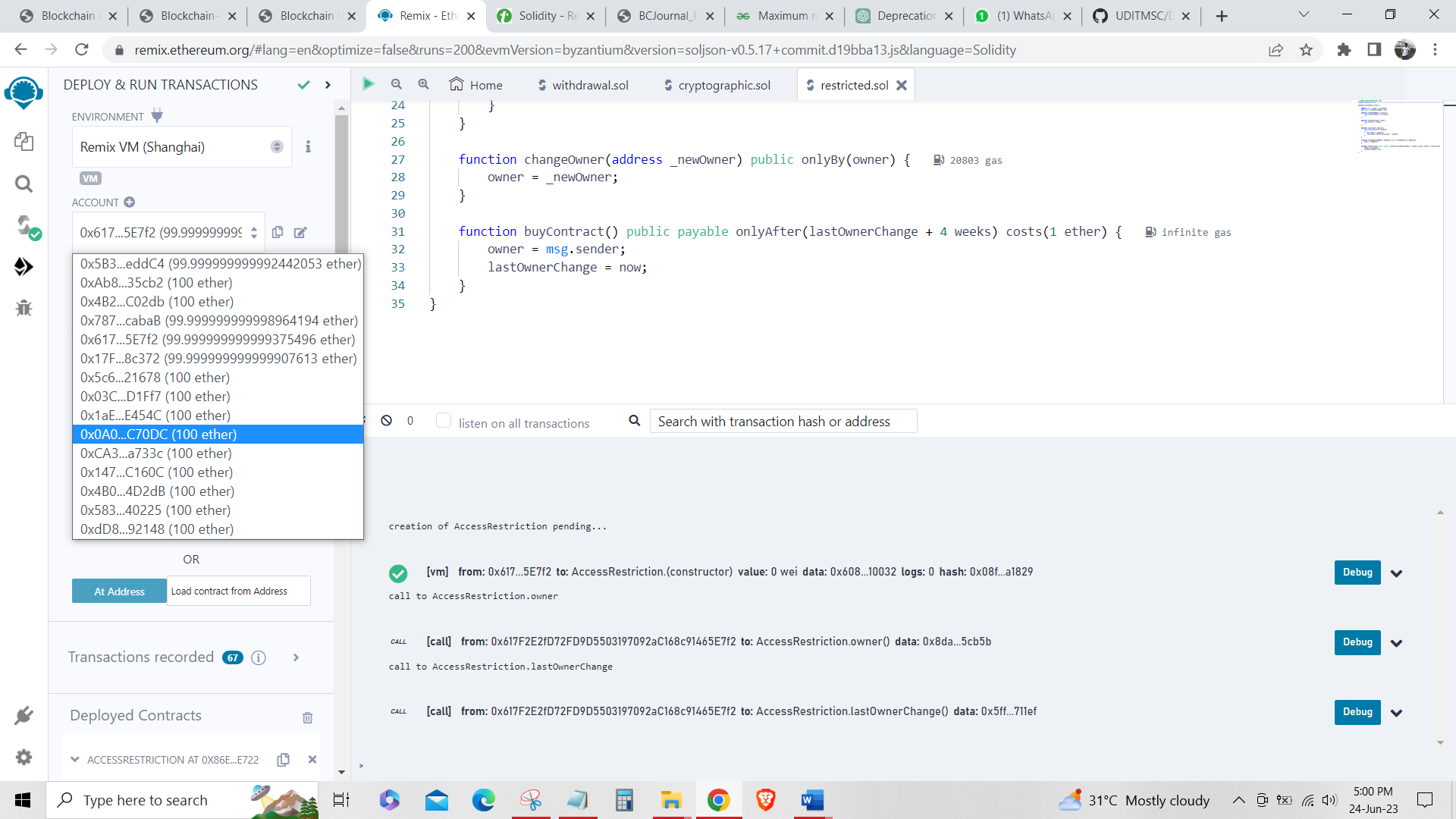
}

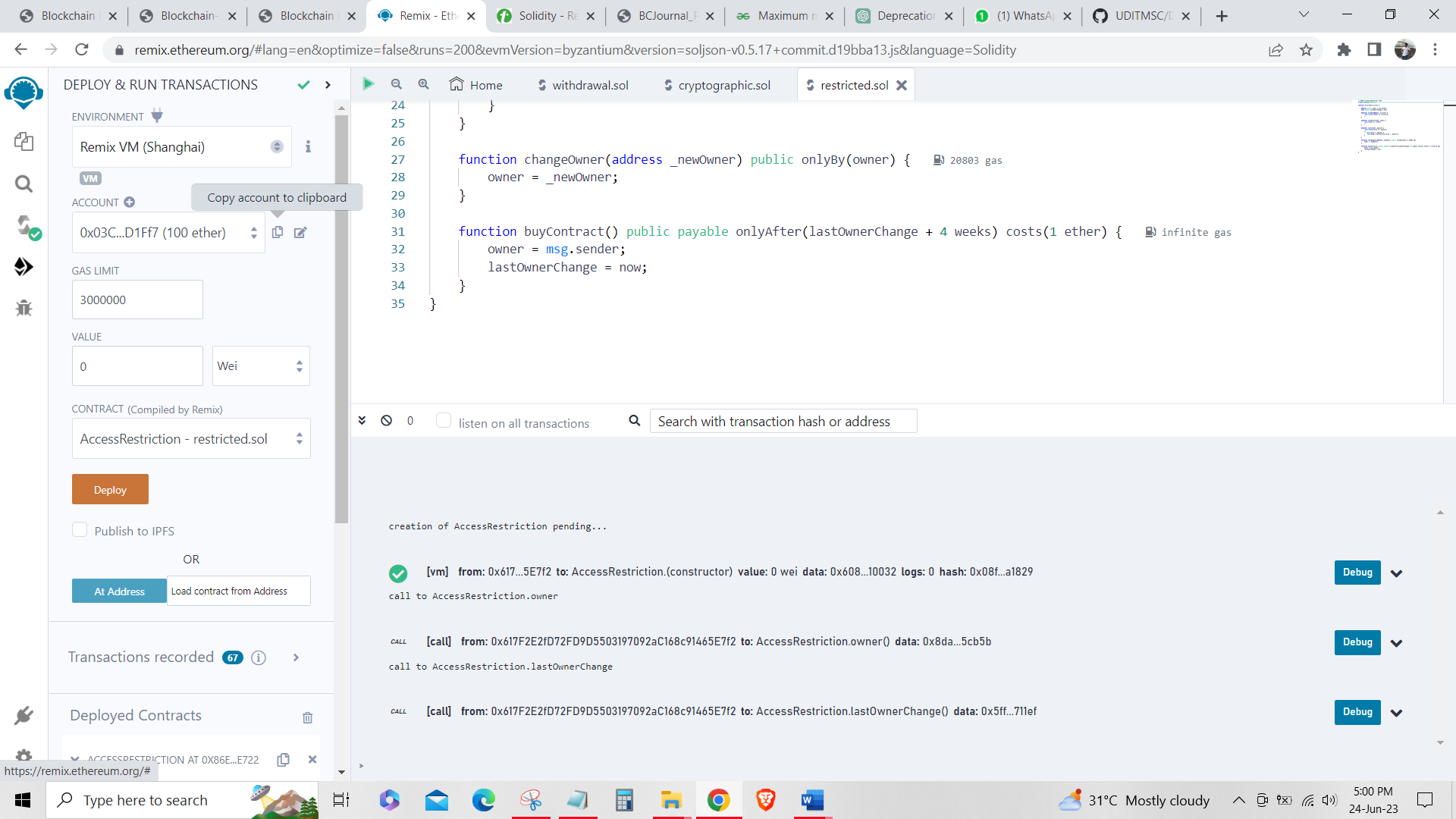
**OUTPUT:**

**Step 1:** Click on owner to create the owner object.

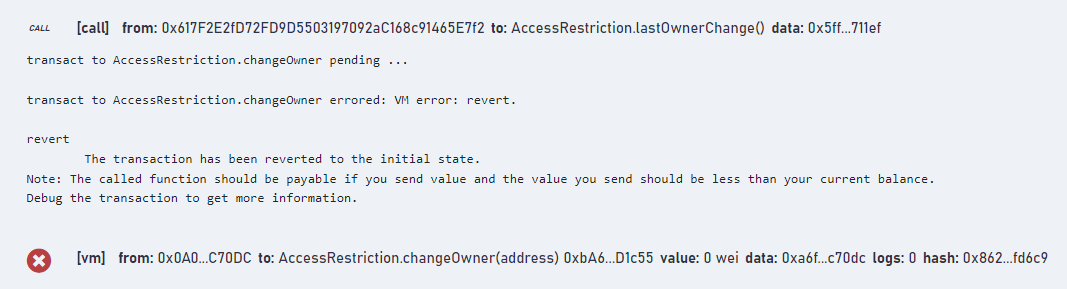
**Step 2:** Click on lastOwnerChange button.

**Step 3:** Change the address of the account from Account dropdown in Deploy tab of Remix IDE.

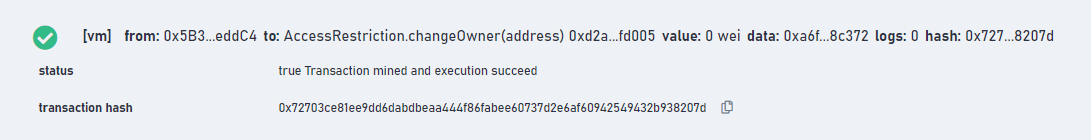
**Step 4:** Copy and paste the address in changeOwner input and click on changeOwner.



**Step 5:** You should get an error as following.



**Step 6:** Now, change back to the actual address of the account and click on changeOwner.





**[B] Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.**

**[I] Contracts**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Contract\_demo {

string message = "Hello";

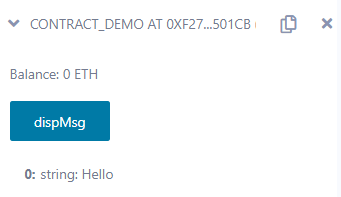
function dispMsg() public view returns (string memory) {

return message;

}

}

**OUTPUT:**



**[II] Inheritance**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Parent {

uint256 internal sum;

function setValue() external {

uint256 a = 10;

uint256 b = 20;

sum = a + b;

}

}

contract child is Parent {

function getValue() external view returns (uint256) {

return sum;

}

}

contract caller {

child cc = new child();

function testInheritance() public returns (uint256) {

cc.setValue();

return cc.getValue();

}

function show\_value() public view returns (uint256) {

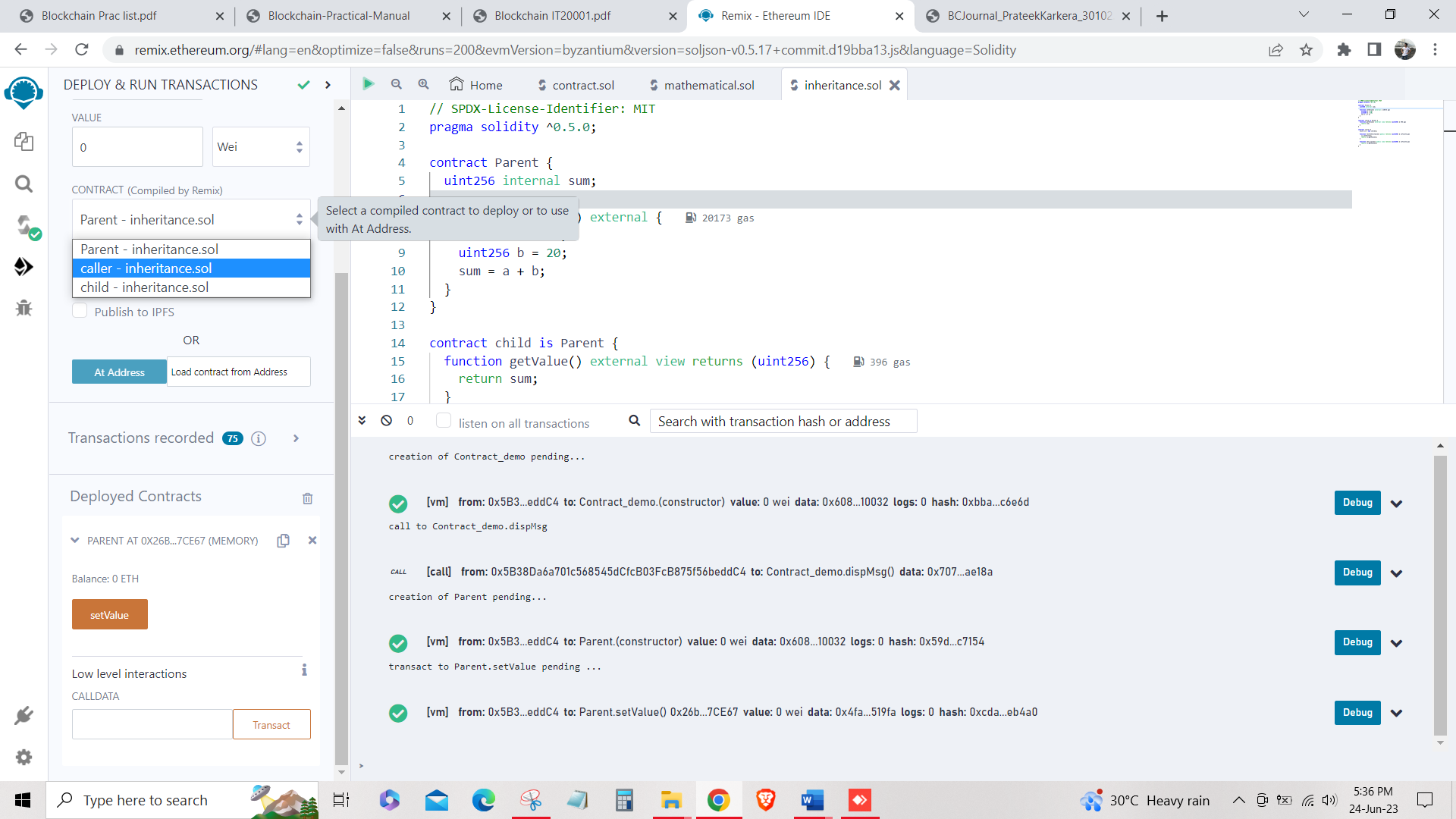
return cc.getValue();

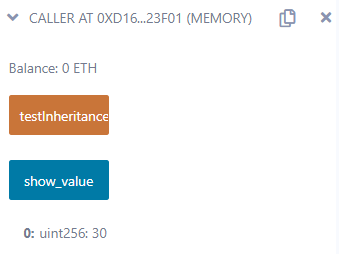
}

}

**OUTPUT:**

**Step 1:** Select caller contract and deploy.

**Step 2:** Click test Inheritance and then click on show\_value to view value.



**[III] Constructors**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

// Creating a contract

contract constructorExample {

string str;

constructor() public {

str = "UDIT";

}

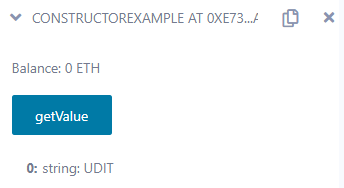
function getValue() public view returns (string memory) {

return str;

}

}

**OUTPUT:**



**[IV] Abstract Contracts**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract Calculator {

function getResult() external view returns (uint256);

}

contract Test is Calculator {

constructor() public {}

function getResult() external view returns (uint256) {

uint256 a = 1;

uint256 b = 2;

uint256 result = a + b;

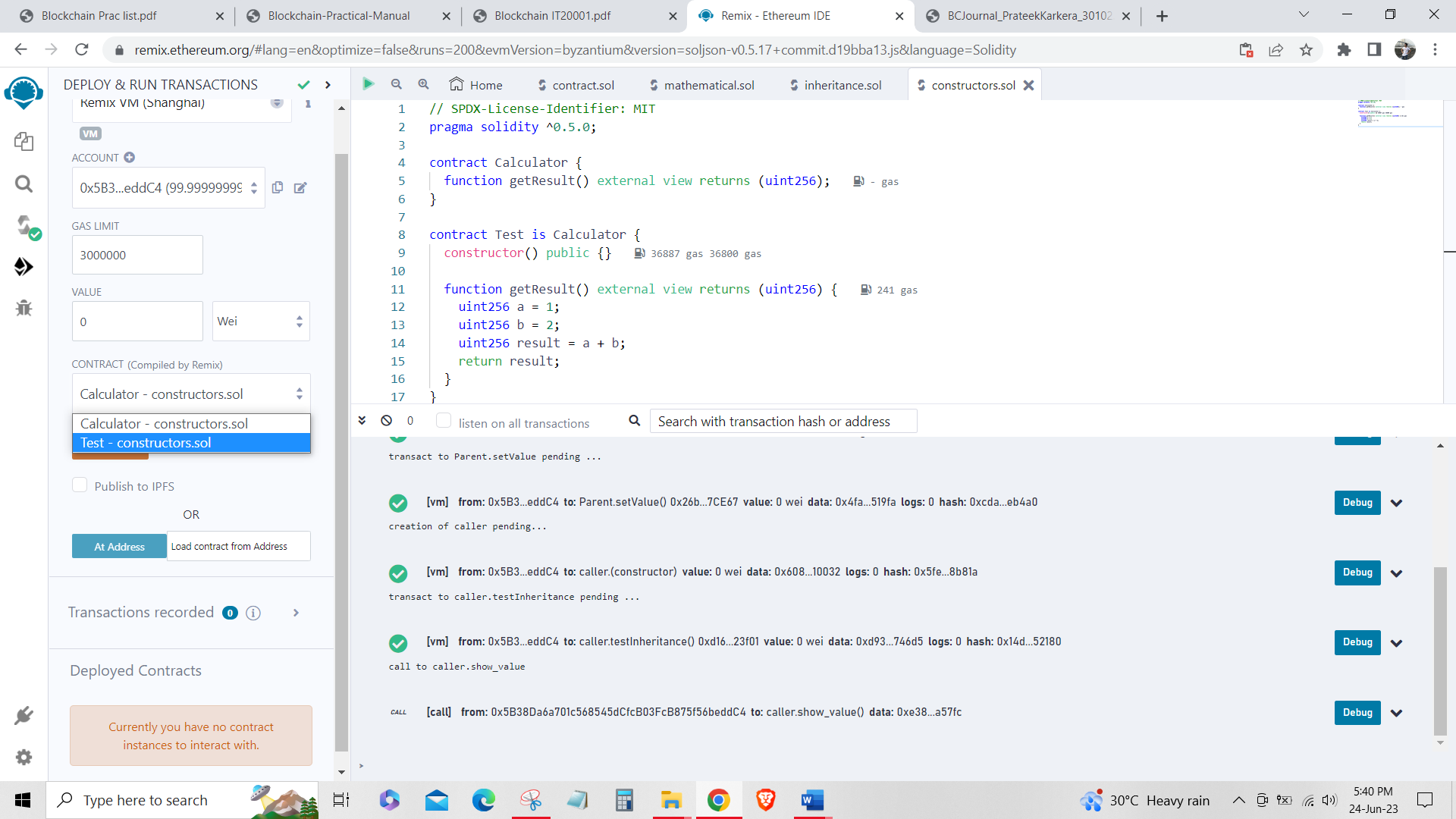
return result;

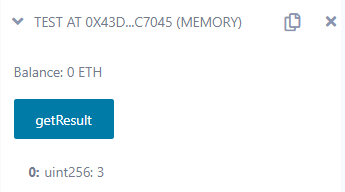
}

}

**OUTPUT:**

**Step 1:** Select Test contract and deploy.

**Step 2:** Click on getResult to get sum of a+b.



**[V] Interfaces**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

interface Calculator {

function getResult() external view returns(uint);

}

contract Test is Calculator {

constructor() public {}

function getResult() external view returns(uint){

uint a = 1;

uint b = 2;

uint result = a + b;

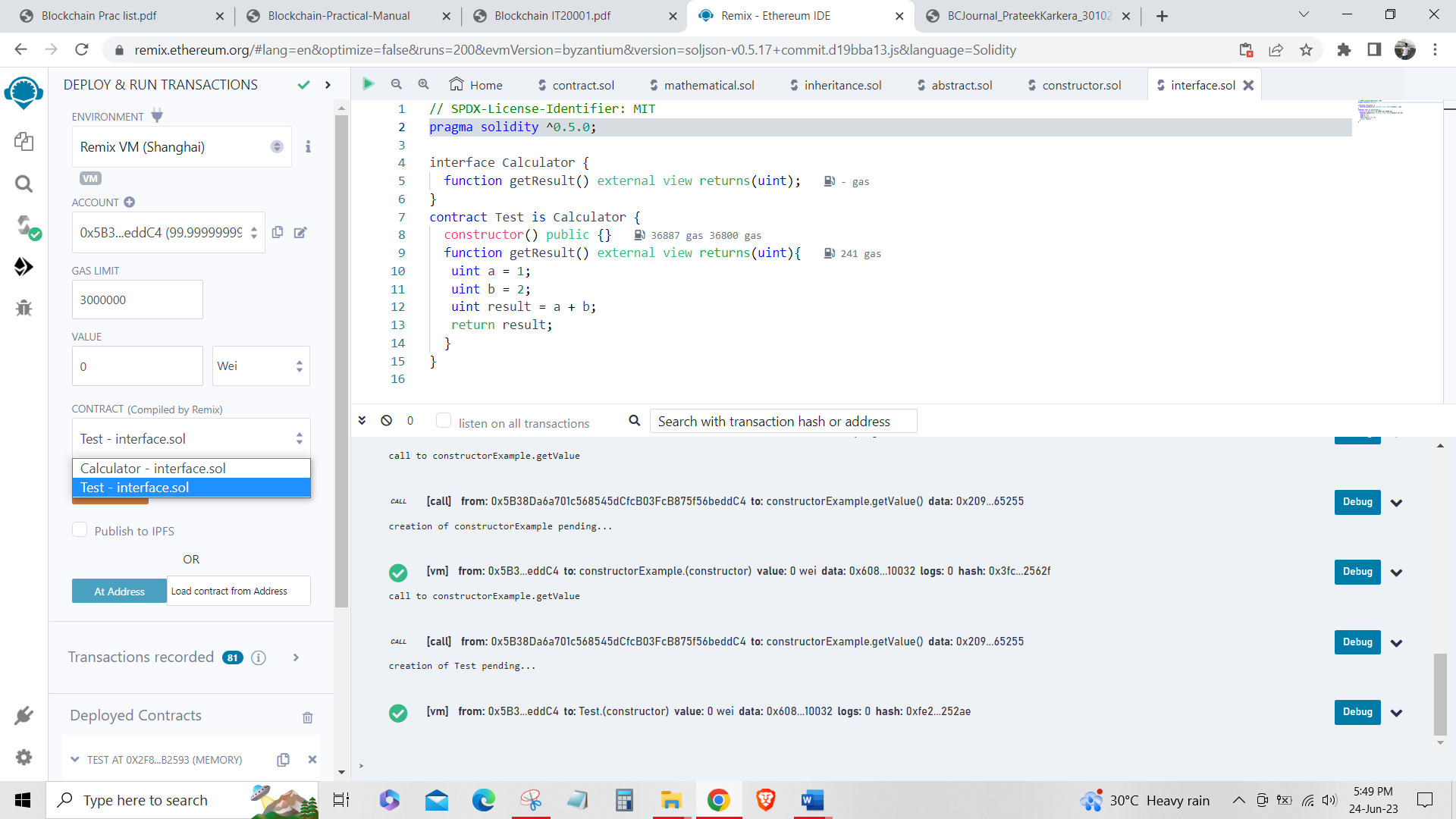
return result;

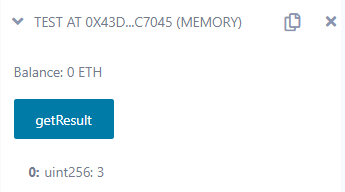
}

}

**OUTPUT:**

**Step 1:** Select Test interface contract and deploy.

**Step 2:** Click on getResult to get sum of a+b.



**[C] Libraries, Assembly, Events, Error handling.**

**[I] Libraries**

**CODE:**

Create library myLib.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

library myMathLib {

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

return a + b;

}

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

return a\*\*b;

}

}

Using the library myLib.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "myLib.sol";

contract UseLib {

function getsum(uint256 x, uint256 y) public pure returns (uint256) {

return myMathLib.sum(x, y);

}

function getexponent(uint256 x, uint256 y) public pure returns (uint256) {

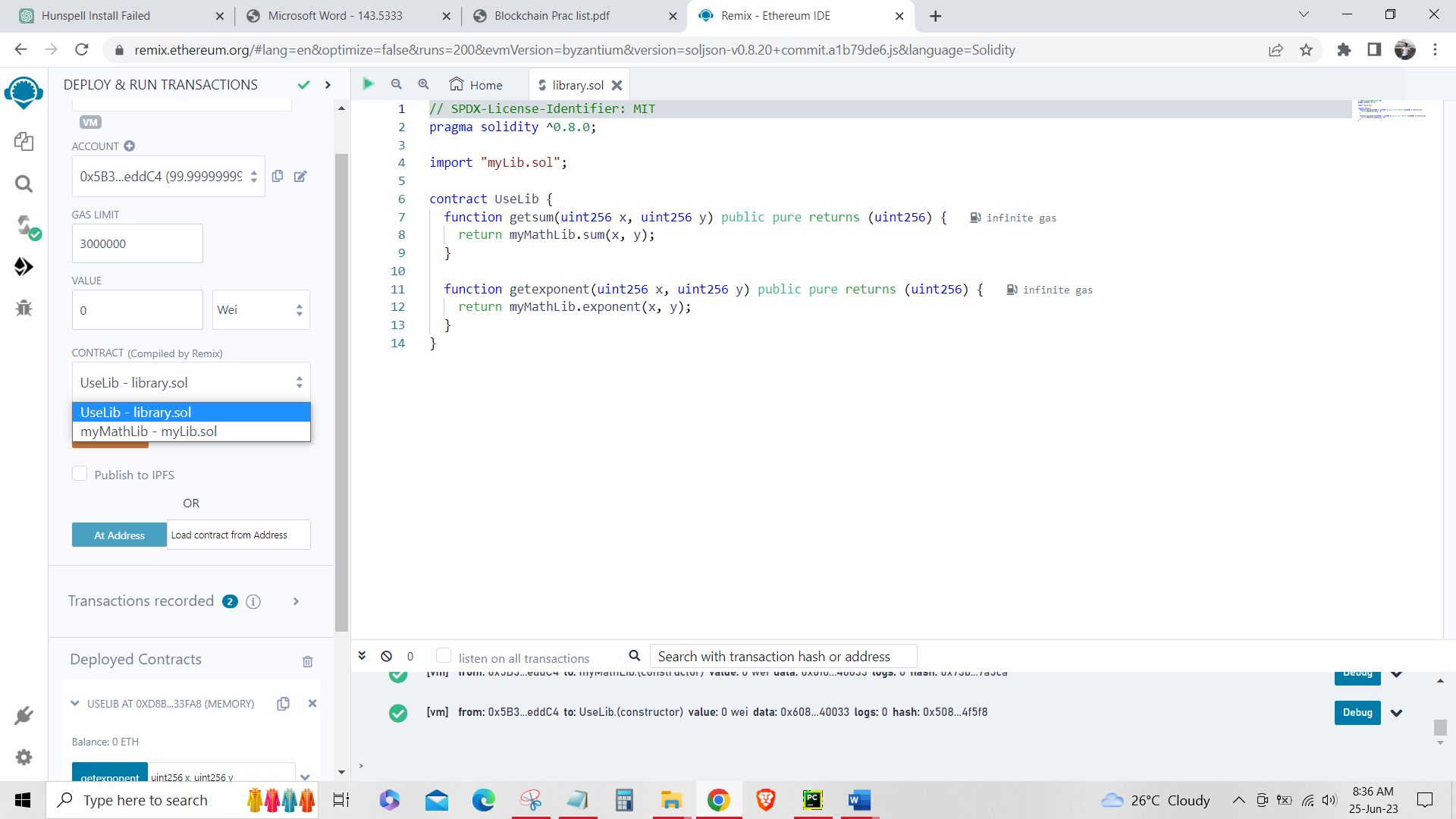
return myMathLib.exponent(x, y);

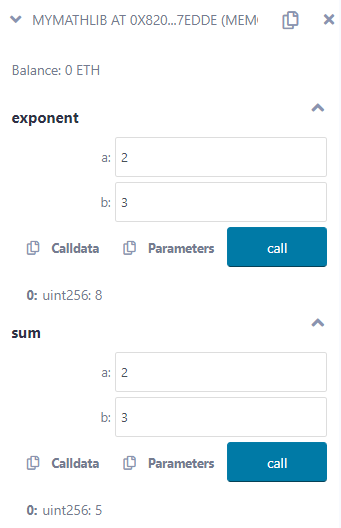
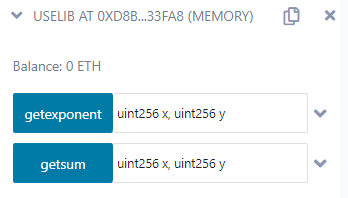
}

}

**OUTPUT:**

**Step 1:** Change contract to UseLib and deploy.

**Step 2:** Input values to both getexponent and getsum functions and get their values as below.



**[II] Assembly**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.16 <0.9.0;

contract InlineAssembly {

// Defining function

function add(uint256 a) public view returns (uint256 b) {

assembly {

let c := add(a, 16)

mstore(0x80, c)

{

let d := add(sload(c), 12)

b := d

}

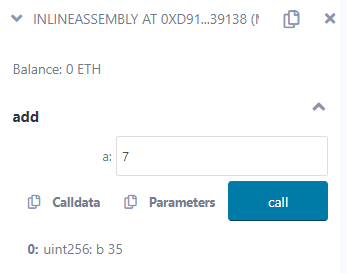
b := add(b, c)

}

}

}

**OUTPUT:**



**[III] Events**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

// Creating a contract

contract eventExample {

// Declaring state variables

uint256 public value = 0;

// Declaring an event

event Increment(address owner);

// Defining a function for logging event

function getValue(uint256 \_a, uint256 \_b) public {

emit Increment(msg.sender);

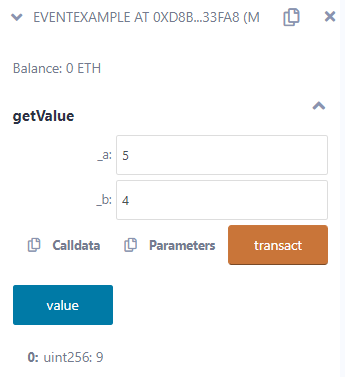
value = \_a + \_b;

}

}

**OUTPUT:**

**Step 1:** Input values to getValue function and get the result by clicking on the value button.



**Step 2:** In the terminal, check for logs.



**[IV] Error handling**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

contract ErrorDemo {

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

uint256 sum = a + b;

require(sum < 255, "Invalid");

assert(sum<255);

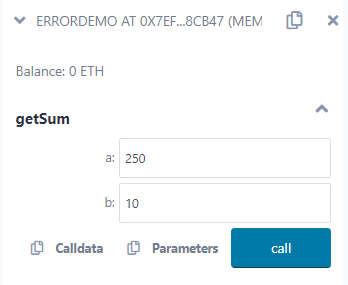
return sum;

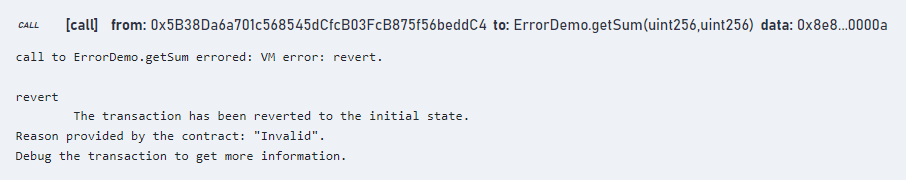
}

}

**OUTPUT:**

**Step 1:** Input values to the getSum function.

**Step 2:** Check terminal panel.



**PRACTICAL 5**

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

**CODE:**

from hashlib import sha256

import time

MAX\_NONCE = 100000000000

def hashGenerator(text):

return sha256(text.encode("ascii")).hexdigest()

def mine(block\_number, transactions, previous\_hash, prefix\_zeros):

prefix\_str = '0'\*prefix\_zeros

for nonce in range(MAX\_NONCE):

text = str(block\_number) + transactions + previous\_hash + str(nonce)

new\_hash = hashGenerator(text)

if new\_hash.startswith(prefix\_str):

print(f"Successfully mined Ethers with nonce value : {nonce}")

return new\_hash

raise BaseException(f"Couldn't find correct hash after trying {MAX\_NONCE} times")

if \_\_name\_\_ == '\_\_main\_\_':

transactions = '''

Prateek->Hajra->77,

Kunal->Soham->18

'''

difficulty = 4

start = time.time()

print("Ether mining started.")

new\_hash = mine(5,transactions,'0000000xa036944e29568d0cff17edbe038f81208fecf9a66be9a2b8321c6ec7', difficulty)

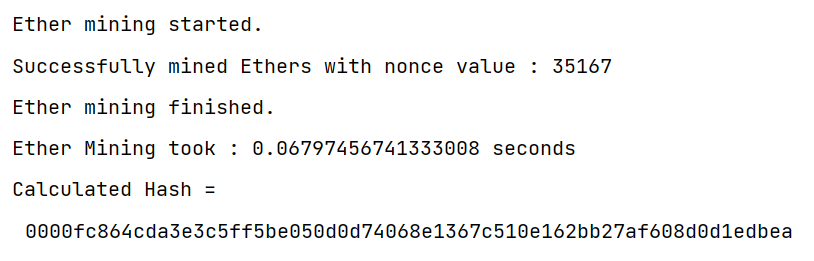
total\_time = str((time.time() - start))

print(f"Ether mining finished.")

print(f"Ether Mining took : {total\_time} seconds")

print(f"Calculated Hash = {new\_hash}")

**OUTPUT:**



**PRACTICAL 6**

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

**CODE:**

from hashlib import sha256

def hashGenerator(text):

return sha256(text.encode("ascii")).hexdigest()

class Block:

def \_\_init\_\_(self,data,hash,prev\_hash):

self.data=data

self.hash=hash

self.prev\_hash=prev\_hash

class Blockchain:

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

hashLast=hashGenerator('gen\_last')

hashStart=hashGenerator('gen\_hash')

genesis=Block('gen-data',hashStart,hashLast)

self.chain=[genesis]

def add\_block(self,data):

prev\_hash=self.chain[-1].hash

hash=hashGenerator(data+prev\_hash)

block=Block(data,hash,prev\_hash)

self.chain.append(block)

bc=Blockchain()

bc.add\_block('1')

bc.add\_block('2')

bc.add\_block('3')

for block in bc.chain:

print(block.\_\_dict\_\_)

**OUTPUT:**

