**HSNC University Mumbai**

A green and gold logo

Description automatically generated

# Practical Journal

# In

BLOCKCHAIN

**SUBMITTED BY**

# UMANG RAICHURA

**MASTER OF SCIENCE IN INFORMATION TECHNOLOGY**

**PART-II**

**Academic Year 2023-2024**

**KISHINCHAND CHELLARAM COLLEGE,**

**D.W. ROAD, CHURCHGATE, MUMBAI – 400 020.**

## BLOCKCHAIN

SUBMITTED BY

**UMANG RAICHURA**

**2023-2024**



**HSNC UNIVERSITY**

MASTERS OF SCIENCE IN

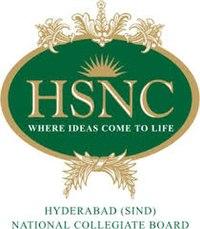
INFORMATION TECHNOLOGY

KISHINCHAND CHELLARAM COLLEGE

D.W.ROAD, CHURCHGATE,MUMBAI–400 020.

**SUBJECT CODE MS-SIT-4P1**

BLOCKCHAIN



**KISHINCHAND CHELLARAM COLLEGE**

CHURCHGATE, MUMBAI – 400 020.

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**M.SC. PART- II**

**CERTIFICATE**

This is to certify that the practical done at **K.C. College** by

**MR/MS. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (**Seat No:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) in partial fulfillment for M.SC. (I.T.) Degree Examination has been found satisfactory. This Practical journal had not been submitted for any other examination and does not form part of any other course undergone by the candidate.

**Signature Signature Signature**

**Lecturer-In-Charge External Examiner Course Coordination**

**Guided By Examined By Certified By**

**College Stamp**

**ACKNOWLEDGMENT**

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of this practical journal.

First and foremost, I am immensely thankful to my instructor Mrs. Yaseera Navrekar, for their guidance, support, and valuable insights throughout the course. Their dedication and commitment to teaching have been instrumental in shaping my understanding of the subject matter.

I am also indebted to my fellow students and lab partners who collaborated with me during the practical sessions. Their shared experiences, discussions, and assistance greatly enriched my learning and made the practical exercises more enjoyable.

I would also like to thank all my professors and all the other non-teaching staff of the M.Sc. IT Department of the college for all the help, support and co-operation that they offered during the course of this semester.

I am deeply grateful to all of you for your involvement, encouragement, and support.

**INDEX**

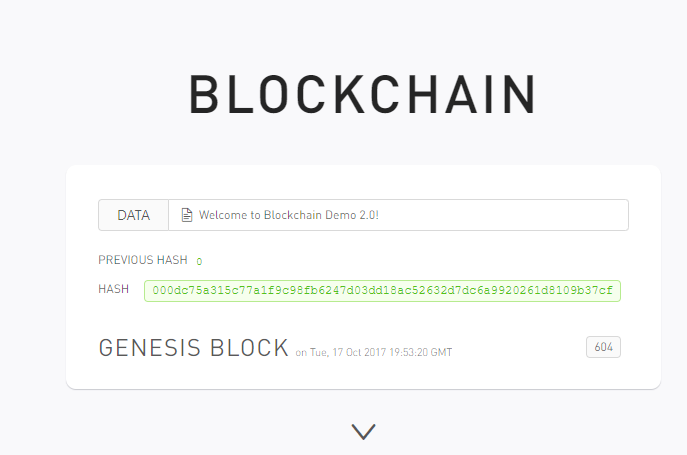
| **Sr. No** | **Topic** | **Date** | **Sign** |
| --- | --- | --- | --- |
| **1** | 1. **Perform Visual demonstration of Blockchain.** 2. **Case Study on any application of Blockchain.** |  |  |
| **2** | **Write the following programs for Blockchain in Python:**   1. A Simple client class that generates 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** | **Write the following programs for Blockchain in Python:**   1. Create multiple transactions and display them. 2. Create a blockchain, a genesis block and execute it. |  |  |
| **4** | **Write the following programs for Blockchain in Python:**   1. Create a mining function and test it. 2. Add blocks to the miner and dump the blockchain. |  |  |
| **5** | **Implement and demonstrate the user of the following in Solidity**   1. Variable 2. Operations 3. Loops 4. Decision Making   v. Strings |  |  |
| **6** | **Implement and demonstrate the user of the following in Solidity**   1. Arrays 2. Enums 3. Mappings 4. Ether Units 5. Special Variables |  |  |

| **7** | **Implement and demonstrate the user of the following in Solidity:**   1. Functions 2. View Functions 3. Pure Functions 4. Fallback Functions 5. Function Overloading 6. Mathematical Functions 7. Cryptographic Functions 8. Generate Random Number |  |  |
| --- | --- | --- | --- |
| **8** | **Implement and demonstrate the user of the following in Solidity:**   1. Contracts 2. Inheritance (Single and Multiple) 3. Constructors 4. Abstract Class 5. Interfaces |  |  |
| **9** | **Implement and demonstrate the user of the following in Solidity:**   1. Libraries 2. Error Handling |  |  |
| **10.** | **Install Hyperledger Fabric and deploy and test it.** |  |  |

**PRACTICAL 1**

1. **AIM:** Perform Visual demonstration of Blockchain.

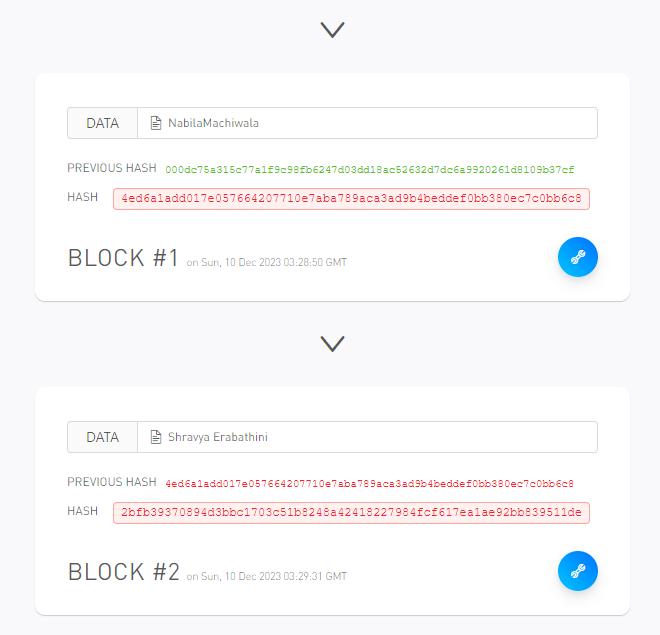
Step 1: Open the given link: https://blockchaindemo.io/



Step 2: Create the block

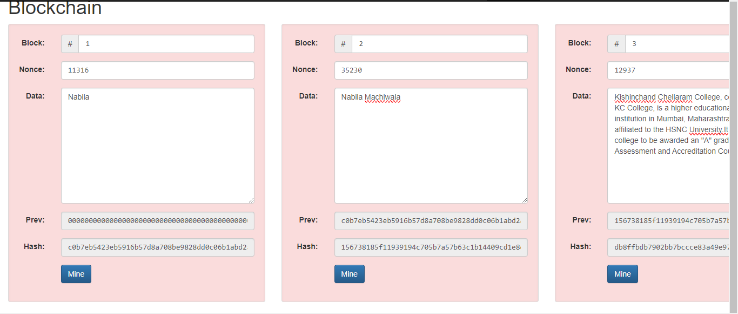


Step 2: Once the block is created we can't modify it

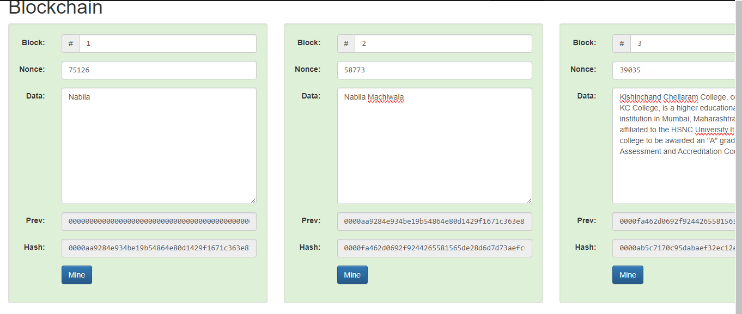


Step 4: Add the Data in blocks (Nonce: hash is generated through the nonce)

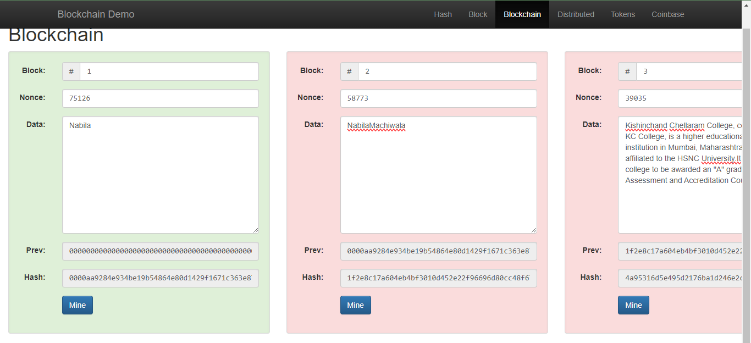
Open the link https://andersbrownworth.com/blockchain/hash



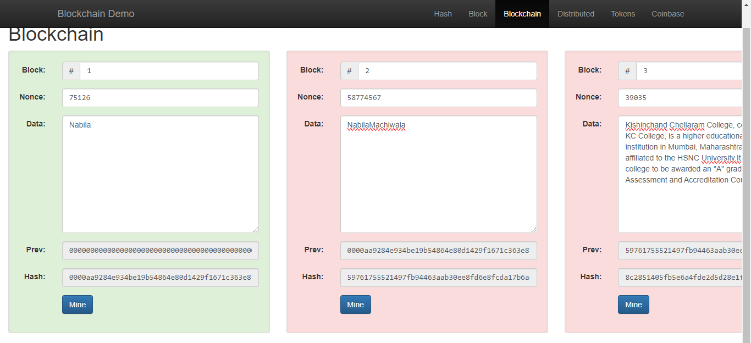
Step 5: The click on mine



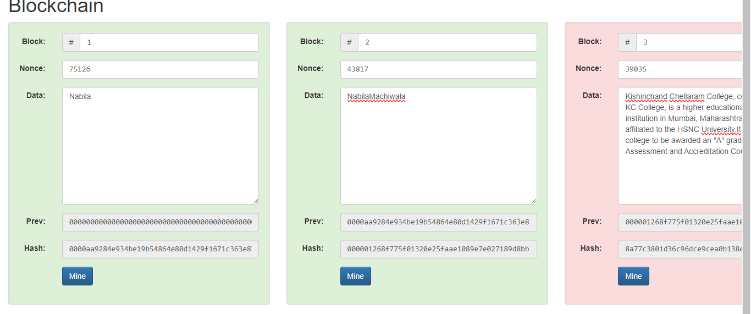
Step 6: Change the block 2 data



Step7 :When we change the block 2 data, the nonce of block 2 will also change

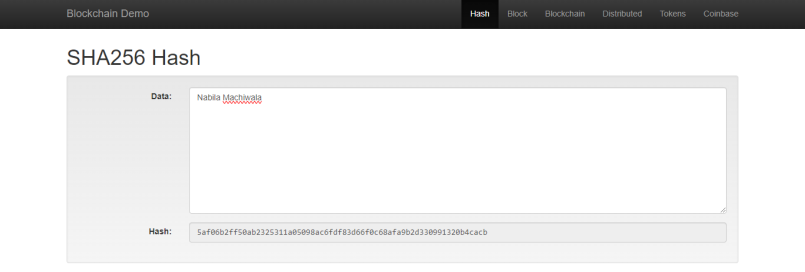


Step 8: Then mine the block

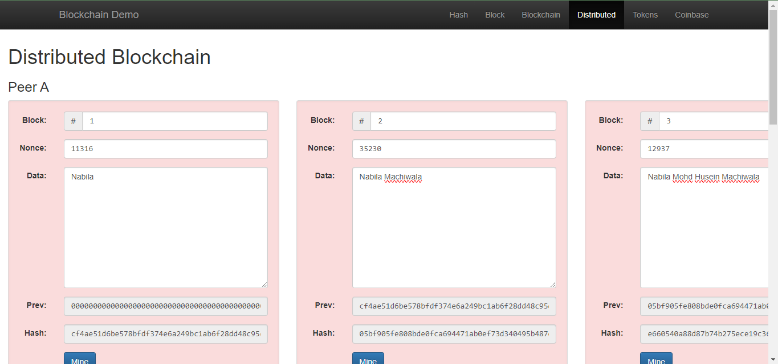


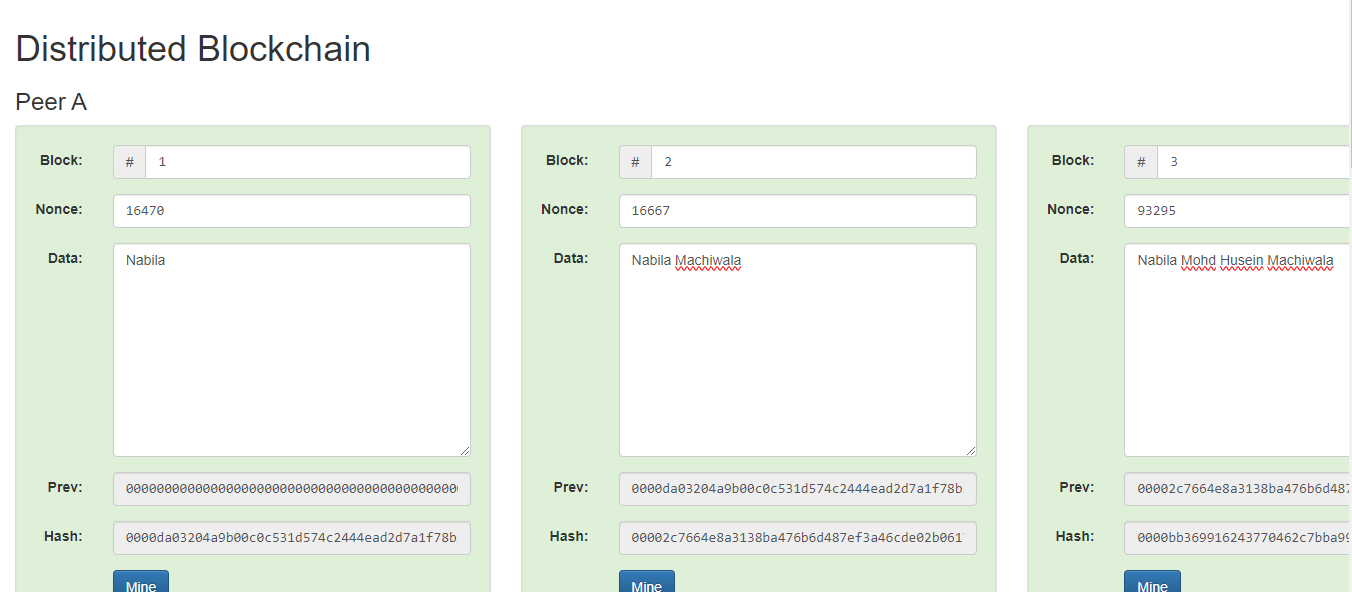
SHA256 Hash (256 bites: 32 bytes)

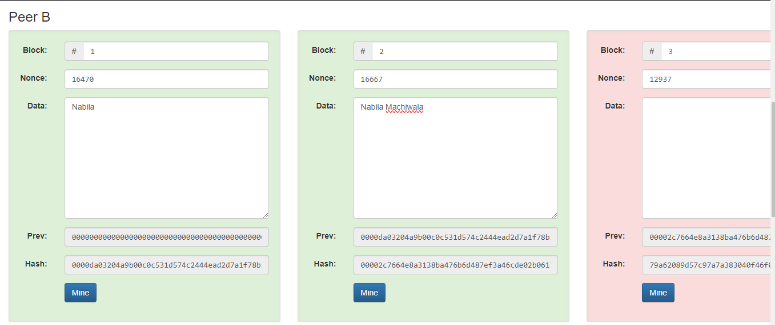
Step 9: When we enter the data the hash will change



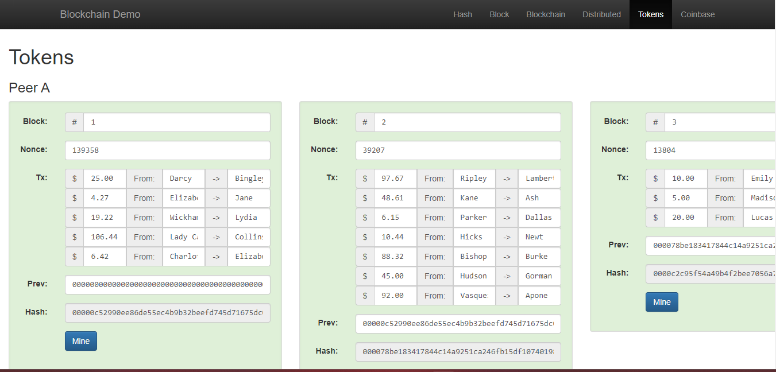
Step 10: Distributed Blockchain

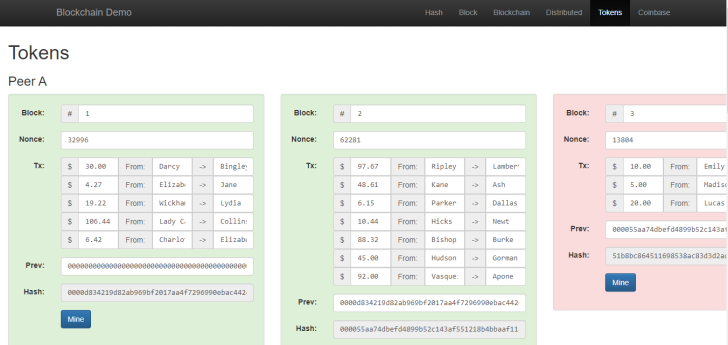




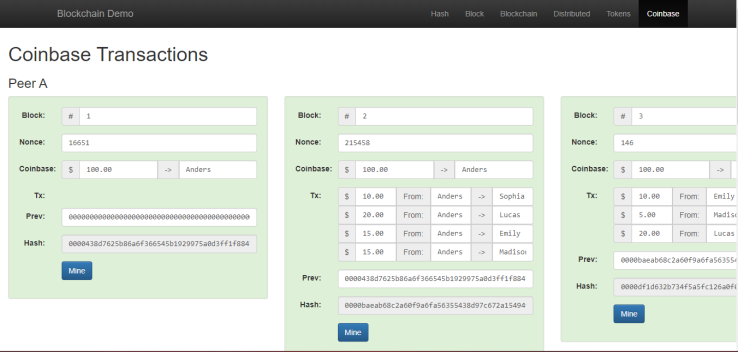


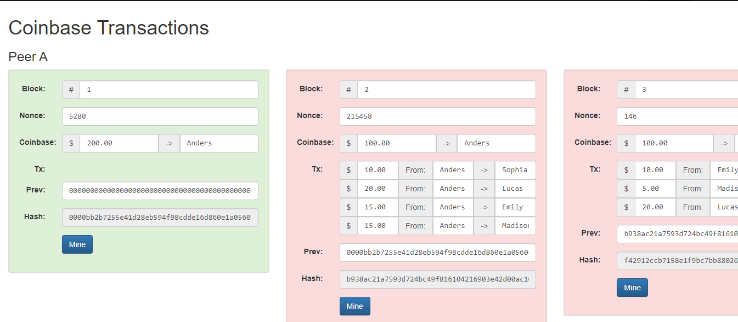
Step 11: Tokens

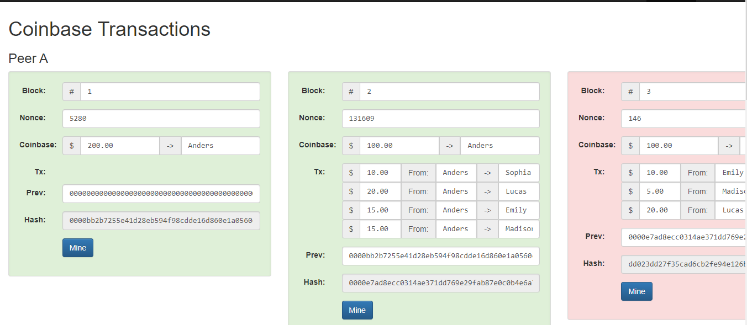




Step 12: Coinbase Transaction







**ii)AIM:** Case Study on any application of Blockchain.

**PRACTICAL 2**

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

i) A Simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it.

**CODE:**

Install the package pycryptodome

from Crypto.PublicKey import RSA

import time

st=time.time()

key=RSA.generate(1024)

#PEM encoding string

p\_key=key.public\_key().export\_key("PEM")

#public and private key are stored in PEM format

priv\_key=key.export\_key("PEM")

print(p\_key)

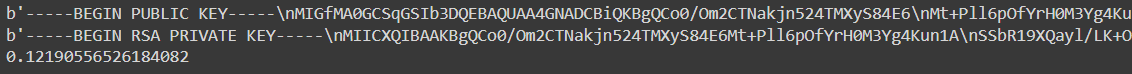
print(priv\_key)

et=time.time()

en=et-st

print(en)

OUTPUT:



1. A transaction class to send and receive money and test it.

**CODE:**

import random

from Crypto.PublicKey import RSA

from Crypto import Random

import binascii

from Crypto.Cipher import PKCS1\_v1\_5

from Crypto.Hash import SHA1

import datetime

from Crypto.Signature import PKCS1\_v1\_5

import collections

from Crypto.Signature import PKCS1\_v1\_5

from collections import OrderedDict

class Client:

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

random=Random.new().read

self.\_private\_key=RSA.generate(1024,random) # private key is used for record keeping

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,receiver,value):

self.sender=sender

self.receiver=receiver

self.value=value

self.time=datetime.datetime.now()

def to\_dict(self):

if self.sender=="Genesis":

identity="Gensis"

else:

identity=self.sender.identity

return collections.OrderedDict({

"sender":identity,

"receiver":self.receiver,

"value":self.value,

"time":self.time

})

def sign\_tran(self):

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

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

h=SHA1.new(str(self.to\_dict()).encode('utf8'))

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

def display\_tran(transaction):

dict=transaction.to\_dict()

print('\nsender,Sender-->\n'+dict['sender'])

print('\nreceiver,Receiver-->\n'+dict['receiver'])

print('\nvalue-->\n'+str(dict['value']))

print('\nstime-->\n'+str(dict['time']))

transaction=[]

Sender=Client()

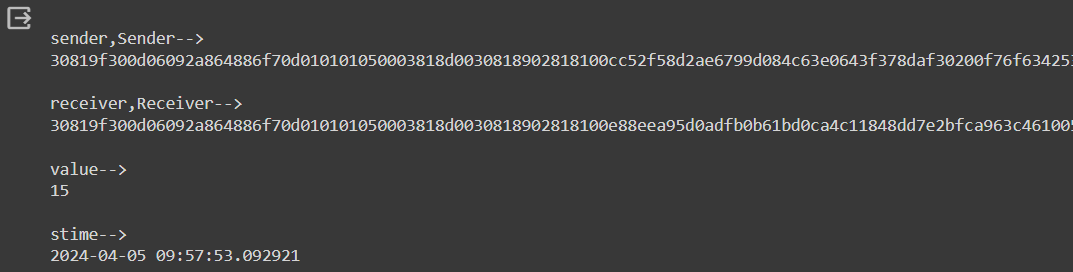
Receiver=Client()

t1=Transaction(Sender,Receiver.identity,15)

t1.sign\_tran()

display\_tran(t1)

**OUTPUT:**

****

**PRACTICAL 3**

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

i)Create multiple transactions and display them.

**CODE:**

import random

from Crypto.PublicKey import RSA

from Crypto import Random

import binascii

from Crypto.Cipher import PKCS1\_v1\_5

from Crypto.Hash import SHA1

import datetime

from Crypto.Signature import PKCS1\_v1\_5

import collections

from Crypto.Signature import PKCS1\_v1\_5

from collections import OrderedDict

class Client:

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

random=Random.new().read

self.\_private\_key=RSA.generate(1024,random) # private key is used for record keeping

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,receiver,value):

self.sender=sender

self.receiver=receiver

self.value=value

self.time=datetime.datetime.now()

def to\_dict(self):

if self.sender=="Genesis":

identity="Gensis"

else:

identity=self.sender.identity

return collections.OrderedDict({

"sender":identity,

"receiver":self.receiver,

"value":self.value,

"time":self.time

})

def sign\_tran(self):

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

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

h=SHA1.new(str(self.to\_dict()).encode('utf8'))

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

def display\_tran(transaction):

dict=transaction.to\_dict()

print('\nsender,Sender-->\n'+dict['sender'])

print('\nreceiver,Receiver-->\n'+dict['receiver'])

print('\nvalue-->\n'+str(dict['value']))

print('\nstime-->\n'+str(dict['time']))

transactions=["Transaction1 1","Transaction 2","Transaction 3"]

Sender=Client()

Receiver=Client()

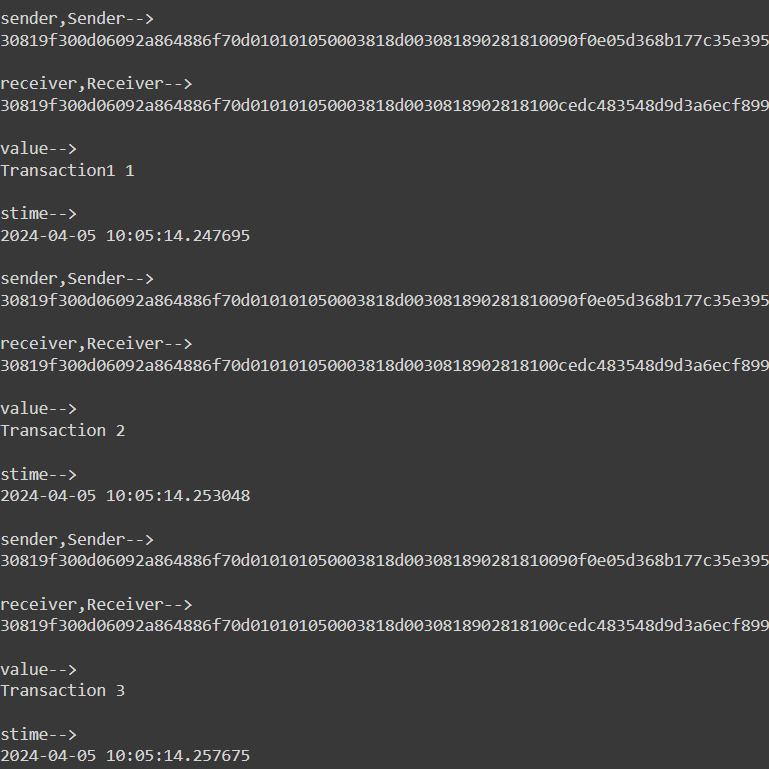
for transaction in transactions:

t=Transaction(Sender,Receiver.identity,transaction)

t.sign\_tran()

display\_tran(t)

**OUTPUT:**

****

**ii)** Create a blockchain, a genesis block and execute it.

**CODE:**

import hashlib

import json

from time import time

class Blockchain:

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

self.chain = []

self.pending\_transactions = []

#create a genesis block

self.add\_block(previous\_hash='1')

def add\_block(self, proof = None, previous\_hash = None):

block = {

'index' : len(self.chain)+1,

'timestamp' : time(),

'transactions': self.pending\_transactions,

'proof' : proof or 0, #Proofof Work

'previous\_hash': previous\_hash or self.hash(self.chain[-1]) if self.chain else None,

}

#Reset the list of pending transaction

self.pending\_transactions = []

self.chain.append(block)

return block

def add\_transaction(self, sender, recipient , amount):

transaction = {

'sender': sender,

'recipient' : recipient,

'amount': amount,

}

self.pending\_transactions.append(transaction)

return self.last\_block['index']+1

@staticmethod

def hash(block):

#hashes a block

block\_string= json.dumps(block, sort\_keys = True).encode()

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

@property

def last\_block(self):

return self.chain[-1]

my\_blockchain = Blockchain()

#Add transactions

my\_blockchain.add\_transaction('A','B',10)

my\_blockchain.add\_transaction('B','C',5)

my\_blockchain.add\_transaction('C','Sender',10)

my\_blockchain.add\_transaction('Sende','Receiver',5)

proof\_of\_work = 123

previous\_hash = my\_blockchain.hash(my\_blockchain.last\_block)

my\_blockchain.add\_block(proof\_of\_work, previous\_hash)

#Dump the blockchain

for block in my\_blockchain.chain:

print(block)

**OUTPUT:**

****

**PRACTICAL 4**

**AIM:** Create multiple transactions and display them.

i) 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

print("prefix",prefix)

for i in range(1000): # attempting to find a valid nonce

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

print("TESTINT ==> "+digest)

if digest.startswith(prefix):

print("aFTER "+str(i)+" ITERITION found nounce "+digest)

return 1

mine("Sender",2)

**OUTPUT:**

****

ii) Add blocks to the miner and dump the blockchain.

**CODE:**

import hashlib

import json

from time import time

class Blockchain:

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

self.chain = []

self.pending\_transactions = []

#create a genesis block

self.add\_block(previous\_hash='1')

def add\_block(self, proof = None, previous\_hash = None):

block = {

'index' : len(self.chain)+1,

'timestamp' : time(),

'transactions': self.pending\_transactions,

'proof' : proof or 0, #Proofof Work

'previous\_hash': previous\_hash or self.hash(self.chain[-1]) if self.chain else None,

}

#Reset the list of pending transaction

self.pending\_transactions = []

self.chain.append(block)

return block

def add\_transaction(self, sender, recipient , amount):

transaction = {

'sender': sender,

'recipient' : recipient,

'amount': amount,

}

self.pending\_transactions.append(transaction)

return self.last\_block['index']+1

@staticmethod

def hash(block):

#hashes a block

block\_string= json.dumps(block, sort\_keys = True).encode()

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

@property

def last\_block(self):

return self.chain[-1]

my\_blockchain = Blockchain()

#Add transactions

my\_blockchain.add\_transaction('A','B',10)

my\_blockchain.add\_transaction('B','C',5)

my\_blockchain.add\_transaction('C','Sender',10)

my\_blockchain.add\_transaction('Sender','Receiver',5)

proof\_of\_work = 123

previous\_hash = my\_blockchain.hash(my\_blockchain.last\_block)

my\_blockchain.add\_block(proof\_of\_work, previous\_hash)

#Dump the blockchain

for block in my\_blockchain.chain:

print(block)

**OUTPUT:**

****

# PRACTICAL 5

# AIM: Implement and demonstrate the user of the following in Solidity

* Variable
* Operations
* Loops
* Decision Making
* Strings

### **Variable**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract variable\_dem{

int256 public sign\_num;

bool public isTrue;

address public wallet\_address;

uint256[3] public fixarray;

string public message;

enum status {Pending, Approved}

status public currentstatus;

constructor()

{

sign\_num=-45;

isTrue=true;

wallet\_address=msg.sender;

fixarray=[3,10,8];

currentstatus=status.Pending;

message="K.C college";

}

function update\_var() public {

sign\_num=-20;

isTrue=false;

wallet\_address=msg.sender;

fixarray [2]=37;

currentstatus=status.Approved;

message="K.C College MSc Part 2”.

}

}

**OUTPUT:**

1. Enter no>>Click call >> click on currentstaus >> click isTrue >> click message >> click sign\_num >> click wallet\_address
2. Enter no>>Click call >> click on update\_var >> click on currentstatus >> click isTrue >> click message >> click sign\_num >> click wallet\_address

A screenshot of a computer

Description automatically generated A screenshot of a chat

Description automatically generated

### **Operations**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract math{

uint256 public num1;

uint256 public num2;

uint256 public result;

function setnum (uint256 \_num1, uint \_num2)public {

num1 = \_num1;

num2 = \_num2;

}

function div() public returns (uint) {

require (num2!=0, "oops that possible to divide");

result=num1/num2;

return result;

}

function add() public returns (uint){

result=num1+num2;

return result;

}

function sub() public returns (uint){

result=num1-num2;

return result;

}

function multiply()public returns (uint){

result=num1\*num2;

return result;

}

}

**OUTPUT:**

Enter numbers >> click on transact >> click on operation >> click on result

Add sub div

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Multiply num 1 & num2

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

### **Loops**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract WhileLoopExample {

uint256 public counter;

function increment (uint256 \_iterations) public {

uint256 i = 0;

while (i < \_iterations) {

counter++;

i++;

}

}

}

**OUTPUT:**

Enter iterations >> click on transact >> click on counter.

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

### **Decision Making**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Decision\_Making{

uint256 public num;

function setNumber(uint256 \_num) public {

num = \_num;

}

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

if(num % 2 == 0) {

return string("The number entered is even.");

}

return string(abi.encodePacked("The number entered is odd."));

}

}

**OUTPUT:**

Enter number >> click on transact >> click on check >> click on num

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

### **String**

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Hello{

string public greet;

constructor(){

greet="Hello, MScIT";

}

function setGreet(string memory \_newgreet) public{

greet=\_newgreet;

}

}

**OUTPUT:**

>> Enter Name ->Click on transact -> Click on Greet

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**PRACTICAL 6**

**Implement and demonstrate the user of the following in solidity**

**a) Arrays**

**b) Enums**

**e) Special Variables**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Variables {

    int256 public signNum;

    bool public isTrue;

    address public walletAddress;

    uint256[3] public fixArray;

    string public message;

    enum STATUS { pending, approved } // return index of enums instead of actual value, starting from 0.

    STATUS public currentStatus;

    constructor() {

        signNum = -45;

        isTrue = true;

        walletAddress = msg.sender; // wallet address

        fixArray = [1, 2, 3];

        currentStatus = STATUS.pending; // return index of emum

        message = "Initial message!";

    }

    function updateVariables() public {

        signNum = -20;

        isTrue = false;

        walletAddress = msg.sender; // wallet address

        fixArray[2] = 4; // update value of 3rd element in an array

        currentStatus = STATUS.approved; // return index of emum

        message = "Updated message!";

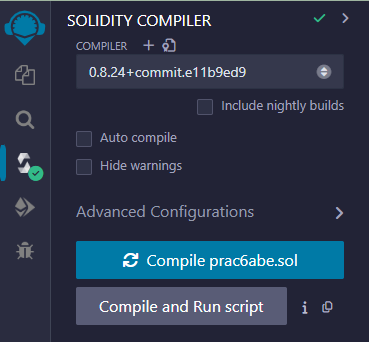
    }

}

**Steps to execute:**

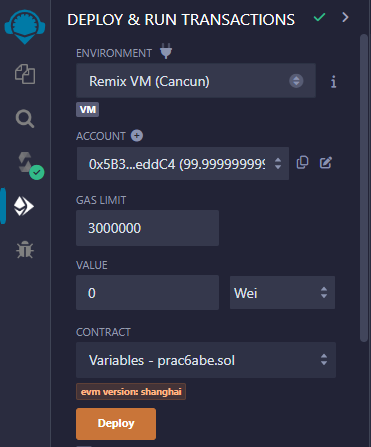
1. Open Remix Ethereum IDE: Go to https://remix.ethereum.org/.
2. Create a new file and write your Solidity code into it.
3. Compile the Contract:

* Click on the "Solidity Compiler" tab on the left side.
* Ensure that the correct compiler version is selected (0.8.24).
* Click on the "Compile" button to compile your contract. Make sure there are no errors in the compilation output.



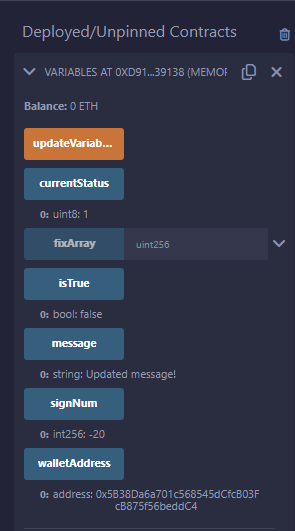
1. Deploy the Contract:

* Switch to the "Deploy & Run Transactions" tab.
* Click on the "Deploy" button next to your contract name. This will deploy your contract onto the Ethereum testnet provided by Remix.



1. Interact with the Contract:

* After deployment, you'll see your contract interface below.
* You can interact with your contract by calling its functions.
* Click on the "updateVariables" button to execute the function.
* After executing the function, you can check the values of the public variables by clicking on it(e.g., signNum, isTrue, walletAddress, fixArray, currentStatus, message) to see if they have been updated according to the logic of your contract.



**c) Ether unit:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract FallbackFunction {

    string public calledFallbackFun;

    fallback() external payable  {

        calledFallbackFun = "I am Fallback Function";

    }

    function getBalance() public view returns(uint){

        return address(this).balance;

    }

    receive() external payable {}

}

contract Sender {

    function transfer() public payable{

        require (msg.value>=2 ether,"Insufficient ether sent");

        (bool sent,)=payable(0xC7B2776E53caAc66eB0725aF2Dd8B1F54EbFdB94).call{value:2 ether}("Transaction Done");

        require(sent, "transaction Failed");

    }

    function getBalance() public view returns (uint) {

        return address(this).balance;

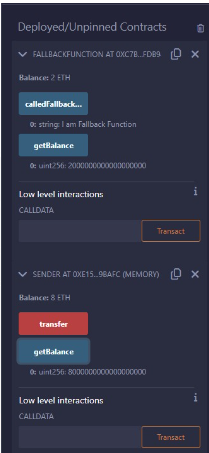
    }

}

**Steps to execute:**

* + 1. Open Remix Ethereum IDE:
* Go to <https://remix.ethereum.org/>.
* Write Solidity code.
  + 1. Compile Contracts:
* Switch to the "Solidity Compiler" tab.
* Make sure the correct compiler version is selected (0.8.24).
* Compile the file.
  + 1. Deploy Contracts:
* Switch to the "Deploy & Run Transactions" tab.
* Deploy both Test and Sender contracts:
* Select Test contract from the dropdown.
* Click on the "Deploy" button to deploy Test contract.
* Repeat the same process for the Sender contract.
  + 1. Interact with the Contracts:
* Select the Sender contract from the dropdown.
* Find the transfer function and specify an amount of ether (at least 2 ether) to send along with the transaction.
* Click on the "transact" button to execute the transfer function.
  + 1. View Output:
* After executing the transfer function, check the balance of both contracts:
* Select the Test contract from the dropdown and click on the getBalance button to see the balance of the Test contract.
* Select the Sender contract from the dropdown and click on the getBalance button to see the balance of the Sender contract.
* Additionally, check the value of calledFallbackfun in the Test contract to see if the fallback function was triggered.

**Output:**



**d) Mapping:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract LedgerBalance{

    mapping(address => uint) balance;

    mapping(address => string) name;

    function updateBalance() public returns(uint){

            balance[msg.sender]=20;

            return balance[msg.sender];

    }

    function senderInfo() public returns(string memory){

        name[msg.sender] = "Sameera";

        return name[msg.sender];

    }

    function printSender() public view returns(address){

        return msg.sender;

    }

}

**Steps to Execute:**

1. Open Remix Ethereum IDE:

* Go to https://remix.ethereum.org/ or use a locally installed version of Remix.

1. Create a New File and write your solidity code in it.
2. Compile the Contract:

* Click on the "Solidity Compiler" tab on the left side.
* Ensure that the correct compiler version is selected (0.8.24).
* Click on the "Compile" button to compile your contract. Make sure there are no errors in the compilation output.

1. Deploy the Contract:

* Switch to the "Deploy & Run Transactions" tab.
* Click on the "Deploy" button next to your contract name. This will deploy your contract onto the Ethereum testnet provided by Remix.

1. Interact with the Contract:

* Execute the updateBalance function:

Click on the "transact" button next to the updateBalance function.

This function will update the balance of the sender to 20 and return the updated balance.

* Execute the senderInfo function:

Click on the "transact" button next to the senderInfo function.

This function will set the name of the sender to "Sameera" and return the name.

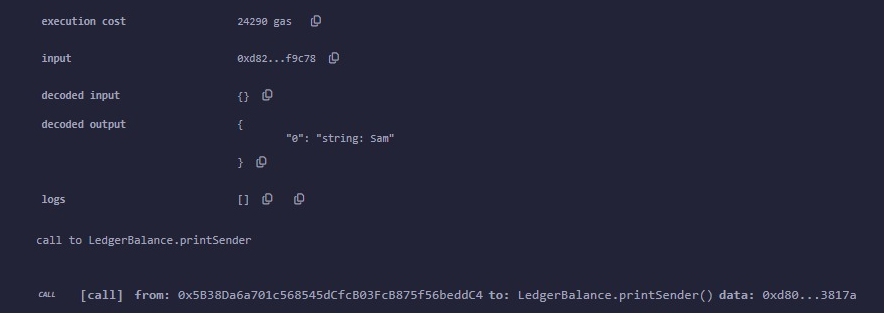
* Execute the printSender function:

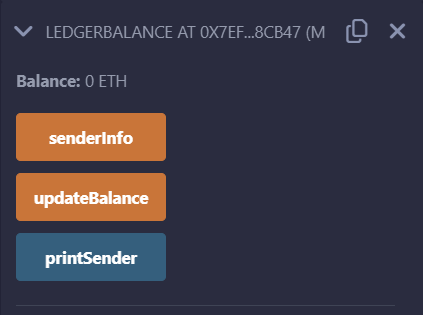
Click on the "call" button next to the printSender function.

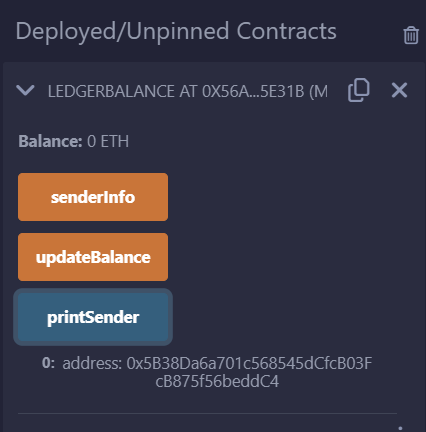
This function will return the address of the sender.

**Output:**

**Before Clicking on senderinfo ,updateBalance**



****

****

### **PRACTICAL 7**

### **Implement and demonstrate the user of the following in Solidity:**

1. Functions
2. View Functions
3. Pure Functions
4. Fallback Functions
5. Function Overloading
6. Mathematical Functions
7. Cryptographic Functions
8. Generate Random Number
9. **Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Function {

uint256 public number;

function setNumber(uint256 \_newNumber) public {

number = \_newNumber;

}

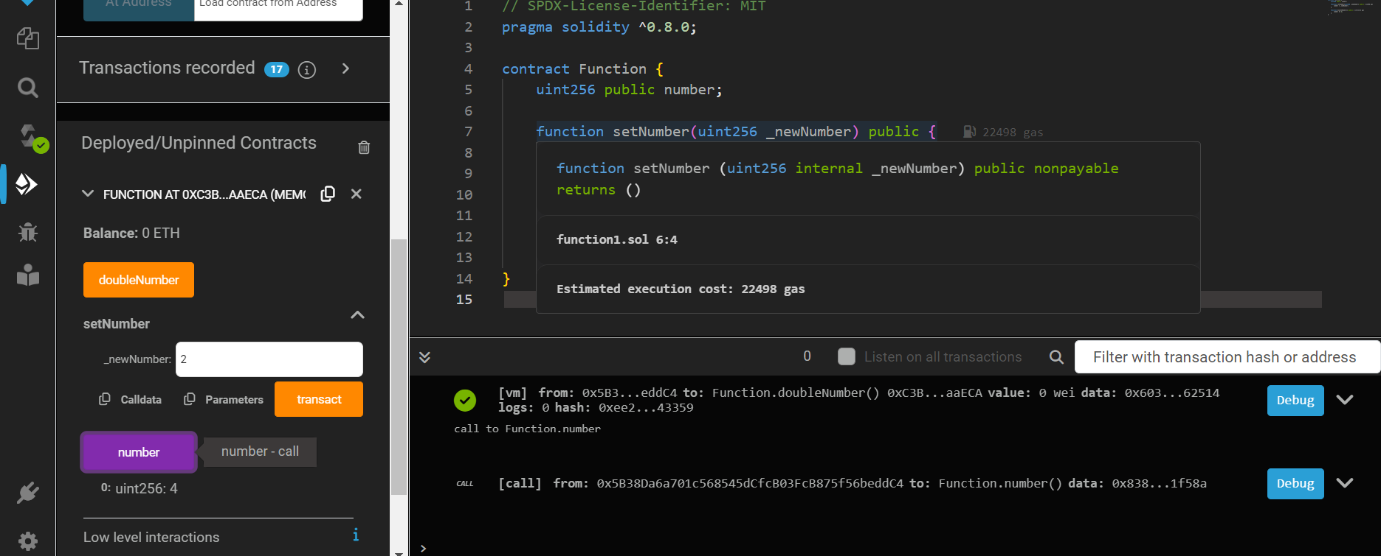
function doubleNumber() public {

number \*= 2;

}

}

**Output:**

****

1. **View Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract HashContract {

bytes32 private message;

function hash1(string memory \_msg) public {

message = keccak256(bytes(\_msg));

}

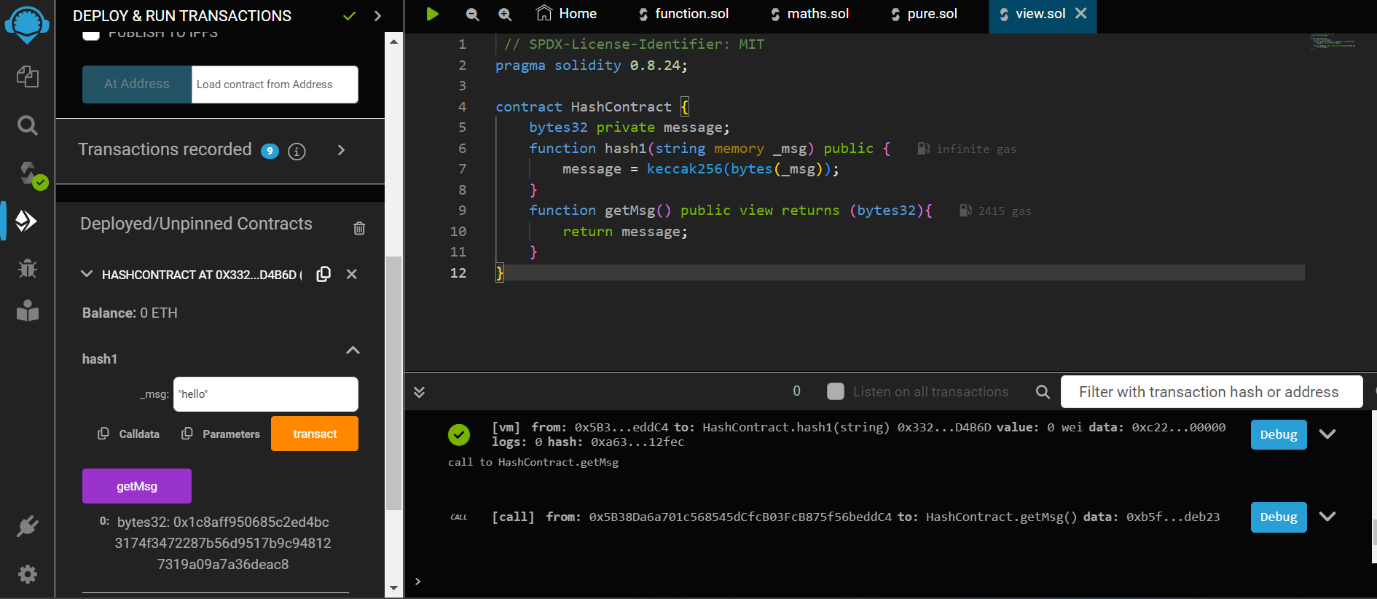
function getMsg() public view returns (bytes32){

return message;

}

}

**Output:**

****

1. **Pure Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Sol{

function testAddMod() public pure returns (uint) {

return addmod(4,5,3);

}

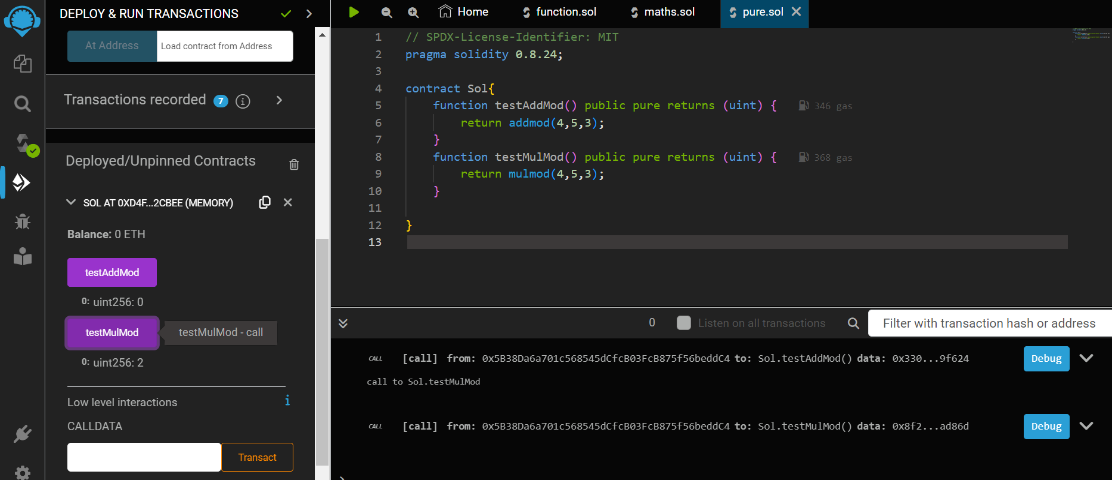
function testMulMod() public pure returns (uint) {

return mulmod(4,5,3);

}

}

**Output:**

****

1. **Fallback Functions:**

**Code: a)**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract FallBackFunction {

// fallback declaration as external always its declared with out name, parameter and does not

return any value.

string public call;

fallback() external payable {

call = "I am a fallback function!";

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

**b)**

//Creating the sender contract

contract Sender

{

function transferEther() public payable

{

require (msg.value>=2 ether,"Insufficient Ether Sent");

(bool sent,) = payable (0xf8e81D47203A594245E36C48e151709F0C19fBe8).call{value: 1

ether}("Transaction completed!");

require(sent, "Transaction failed!");

}

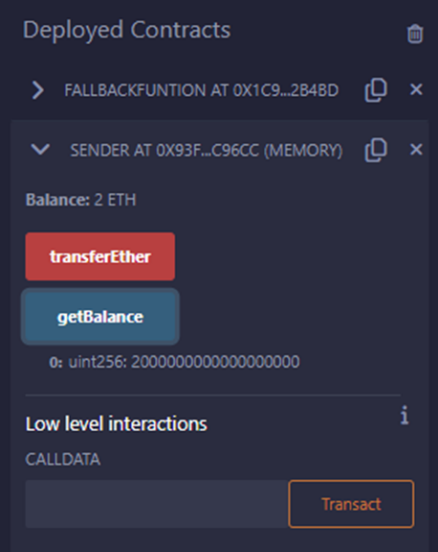
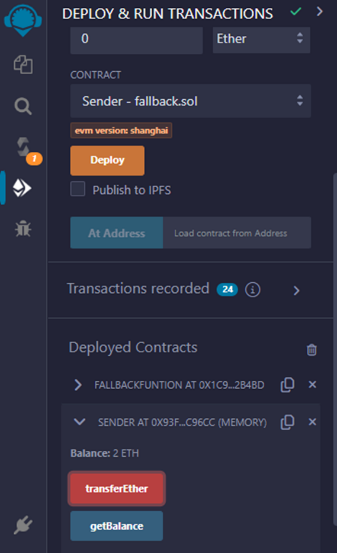
function getBalance() public view returns(uint){

return address(this).balance;

}

}

**Output:**

****

1. **Function Overloading:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Test

{

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

return x+y;

}

function getsum(uint x, uint y, uint z) public pure returns (uint){

return x+y+z;

}

function callTwo() public pure returns (uint){

return getsum(2,8);

}

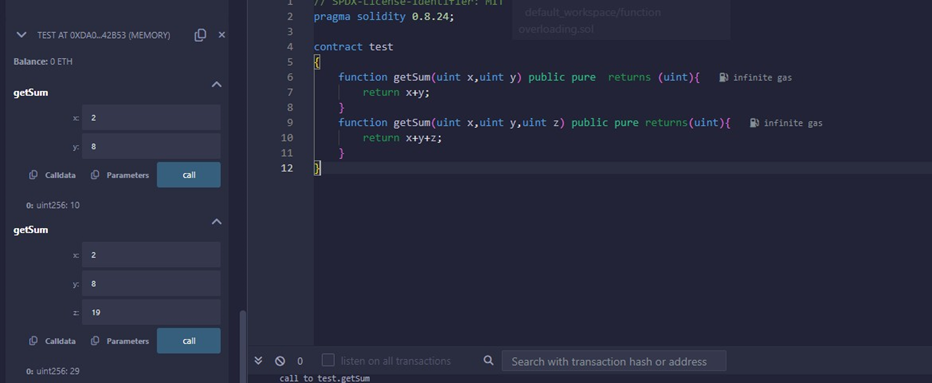
function callthree() public pure returns (uint){

return getsum(2,8,20);

}

}

**Output:**

****

1. **Mathematical Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract PrimeNumber {

function isPrime(uint256 n) public pure returns (string memory) {

for (uint256 i = 2; i < n; i++) {

if (n % i == 0) {

return "Not a prime";

}

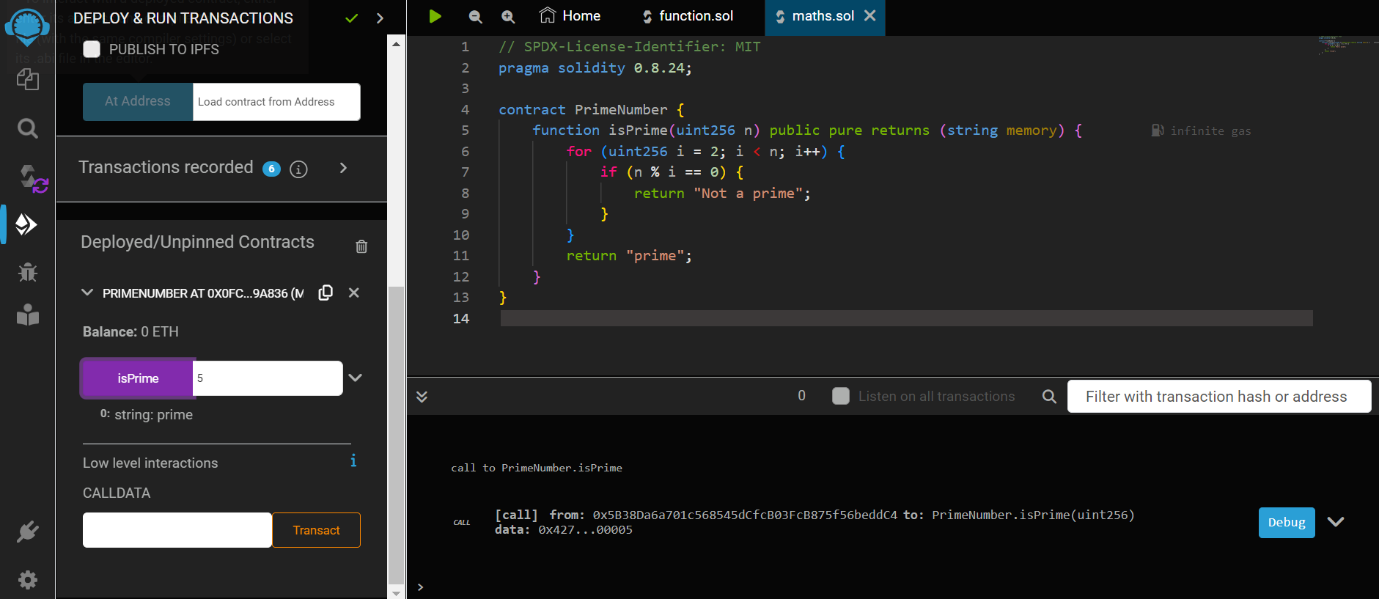
}

return "prime";

}

}

**Output:**

****

1. **Cryptographic Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

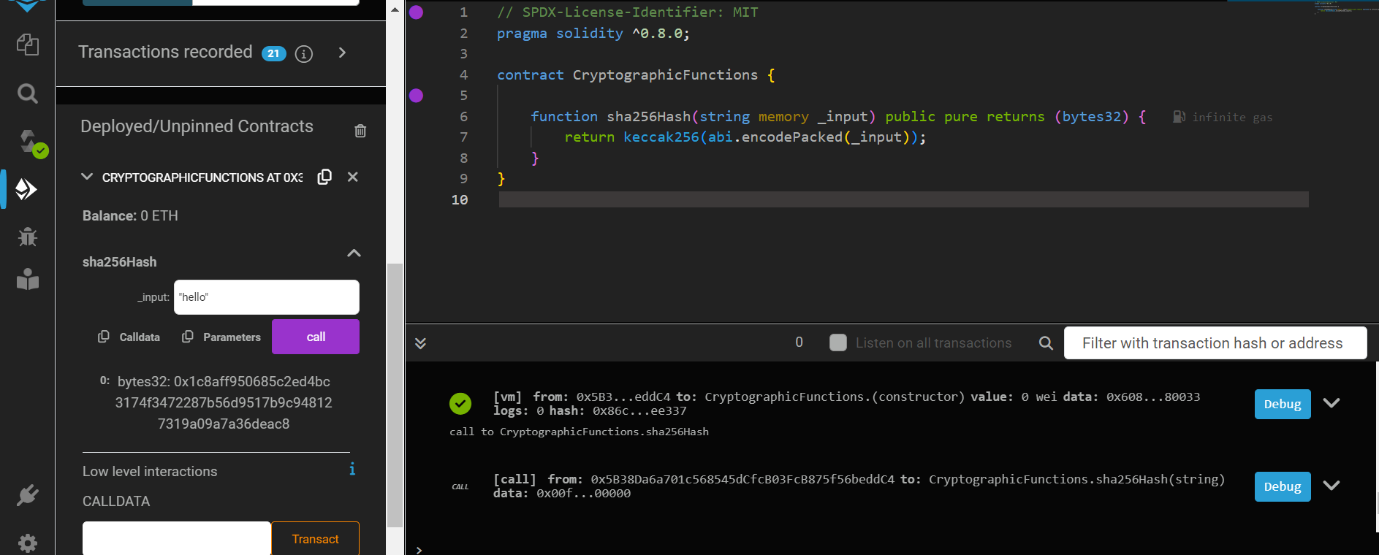
contract CryptographicFunctions {

function sha256Hash(string memory \_input) public pure returns (bytes32) {

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

}

**Output:**

****

1. **Generate Random Number:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract RandomNumber {

uint randomNumber = 0;

function setRandomNumber() public {

randomNumber = uint (keccak256(abi.encodePacked(msg.sender, randomNumber)));

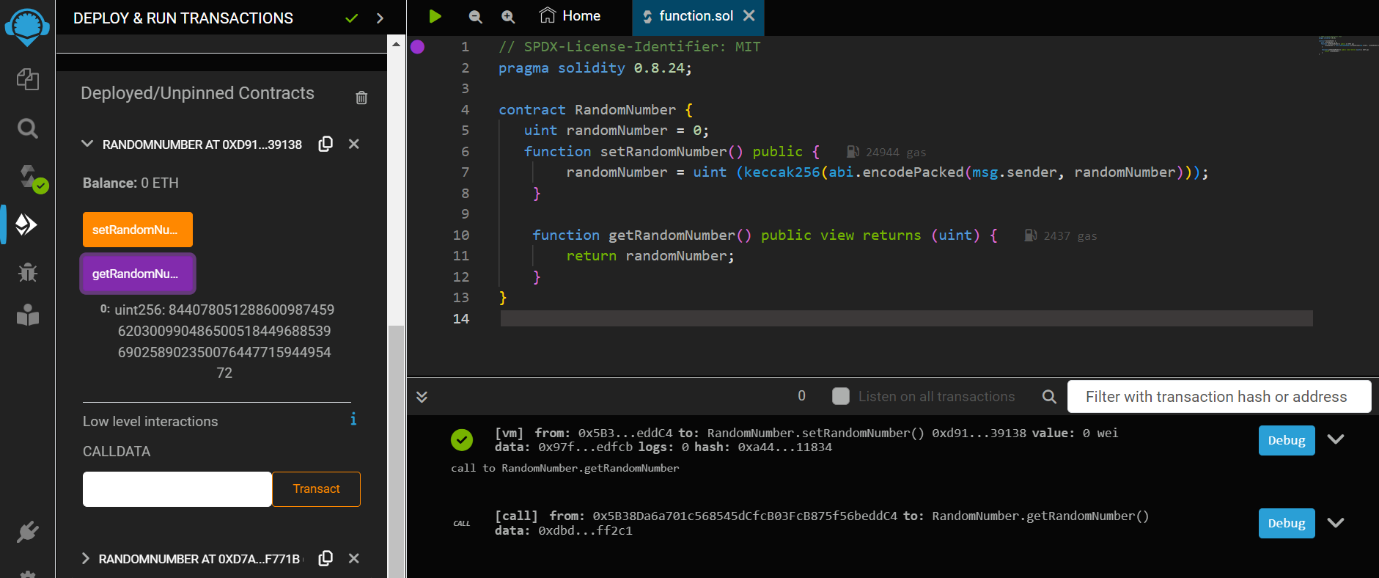
}

function getRandomNumber() public view returns (uint) {

return randomNumber;

}

}

**Output: **

**PRACTICAL 8**

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

1. Contracts
2. Inheritance (Single and Multiple)
3. Constructors
4. Abstract Class
5. Interfaces

**1) Contracts:**

**Code:**

//SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Test{

function TakeLoan(uint256) external payable {

GiveLoan loan = new GiveLoan (350);

}

}

contract GiveLoan{

uint public money;

constructor (uint256 \_amt) public {

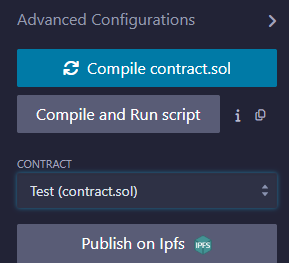
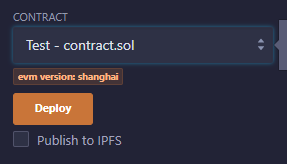
money=\_amt;

}

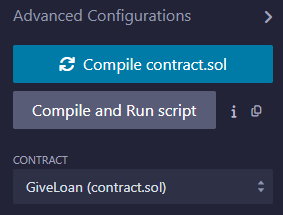
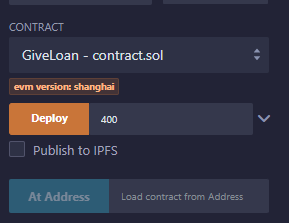
}

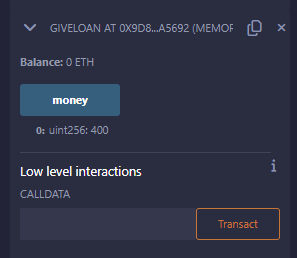
**OUTPUT:**

Compile and deploy test

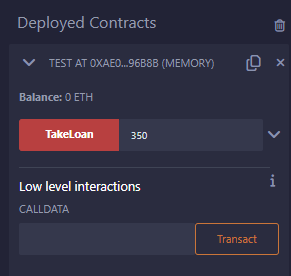
 

Compile and deploy by giving value and click money()



Test.TakeLoan()



2) **Inheritance**

1) Single Inheritance in solidity.

**Code:**

//SPDX-License-Identifier:GPL-3.0

pragma solidity 0.8.24;

contract Parent{

uint internal sum;

function setVal() external{

uint a=50;

uint b=20;

uint c=20;

sum = a+b+c;

}

}

contract child is Parent{

function getVal() external view returns(uint){

return sum;

}

}

contract caller{

child cc = new child();

function Inher() public returns (uint)

{

cc.setVal();

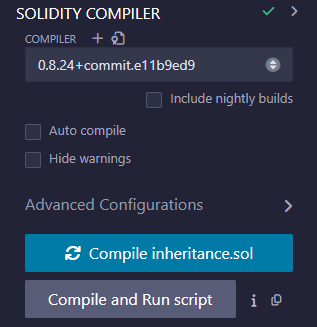
return cc.getVal();

}

}

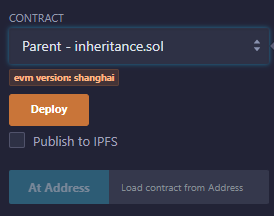
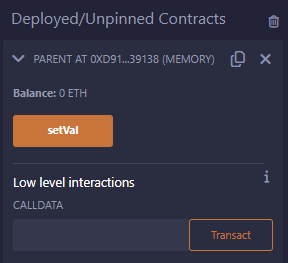
**OUTPUT:**

Compile



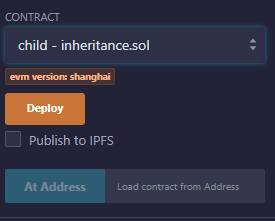
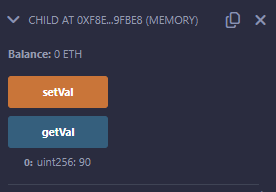
Run:

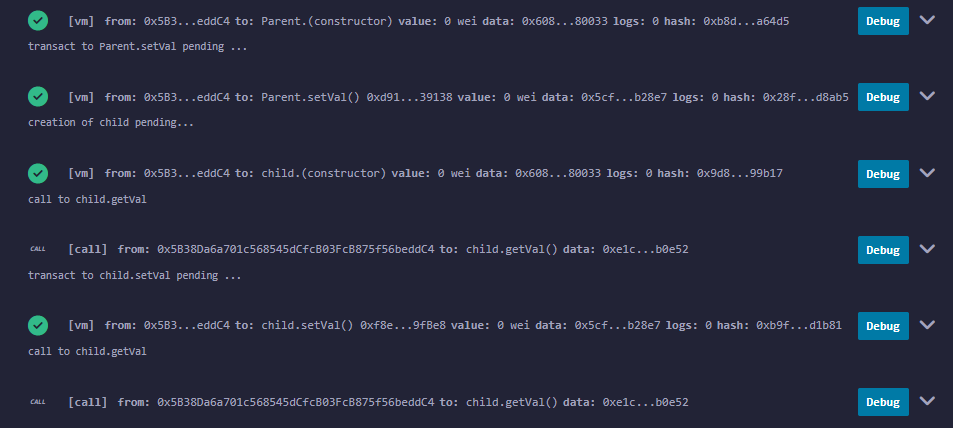
First DEPLOY the parent contract

Deploy the child contract

Select setVal>getVal



**2) Multiple Inheritance in Solidity**

**Code:**

//SPDX-License-Identifier:GPL-3.0

pragma solidity>= 0.8.2<0.9.0;

contract A{

string internal x;

function setA() external{

x = "Multiple inheritance";

}

}

contract B{

uint256 internal pow;

function setB() external{

uint256 a = 20;

uint256 b = 2;

pow = a\*\*b;

}

}

contract C is A,B{

function getStr() external returns (string memory){

return x;

}

function getPow() external returns (uint256){

return pow;

}

}

contract caller{

C contractC = new C();

function testInheritance() public returns(string memory, uint256){

contractC.setA();

contractC.setB();

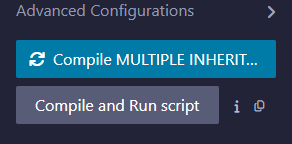
return (contractC.getStr(),contractC.getPow());

}

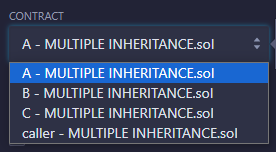
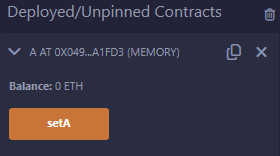
}

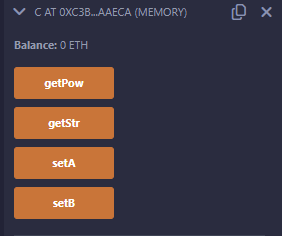
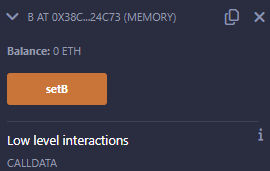
**OUTPUT:**

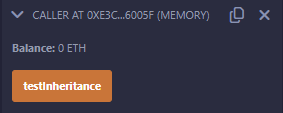
Compile:



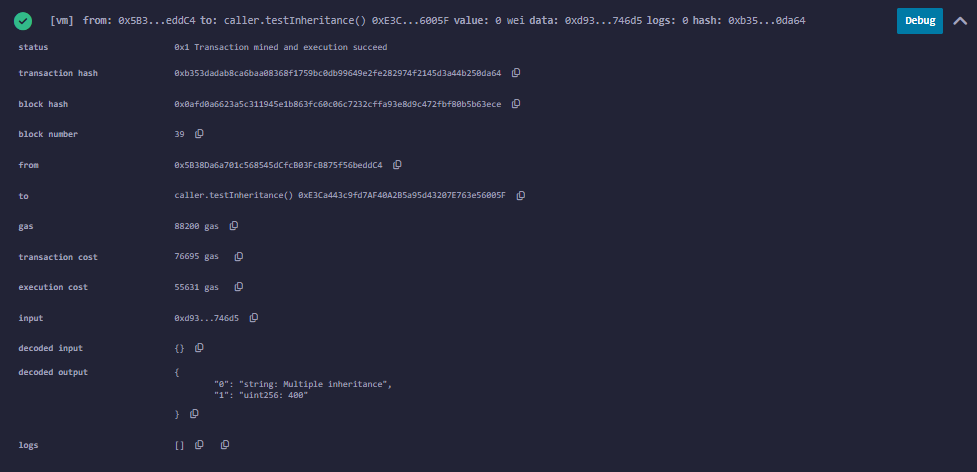
DEPLOY&RUN: DEPLOY all contracts one by one and then deploy.





**Final Output:**



**3) Constructors in solidity**

Step1: create a worksapce

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Hello{

string public greet;

constructor()

{

greet="Hello, M.Sc I.T K.C";

}

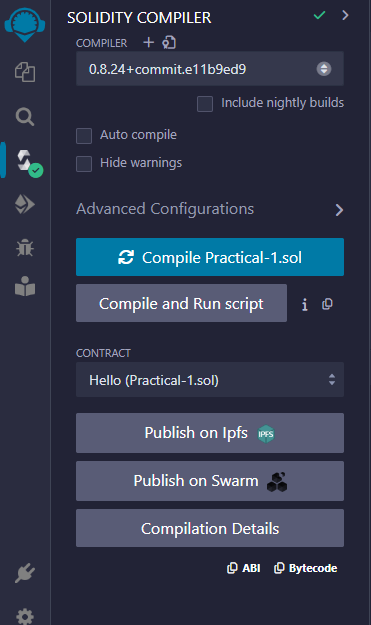
function setGreet (string memory \_newgreet ) public{

greet = \_newgreet;

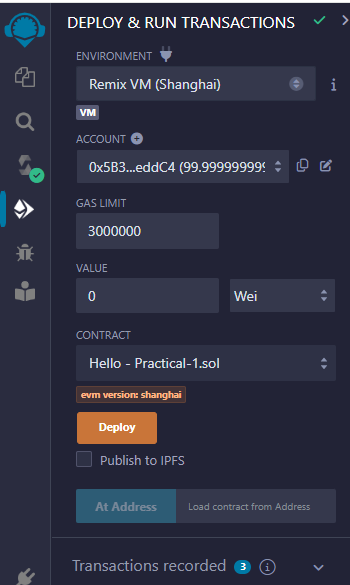
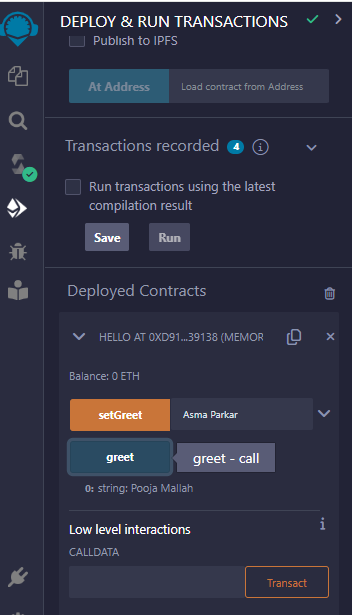
}

}

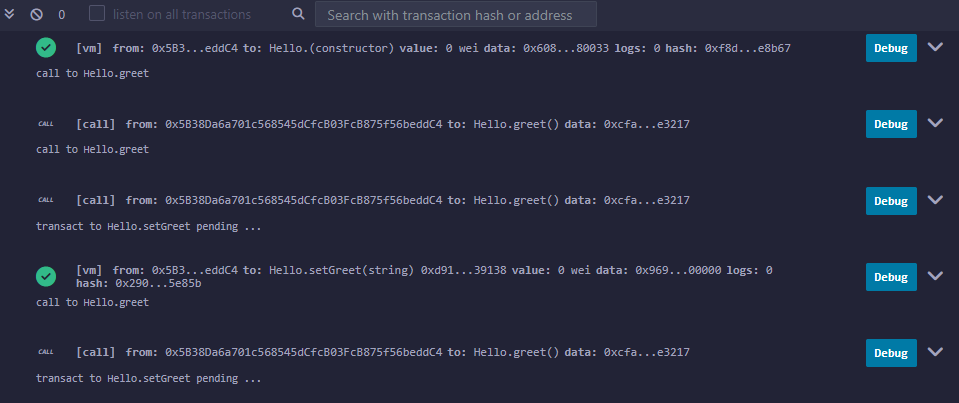
Step 2: Complie the code with solidity option in the left pane of Remix IDE.

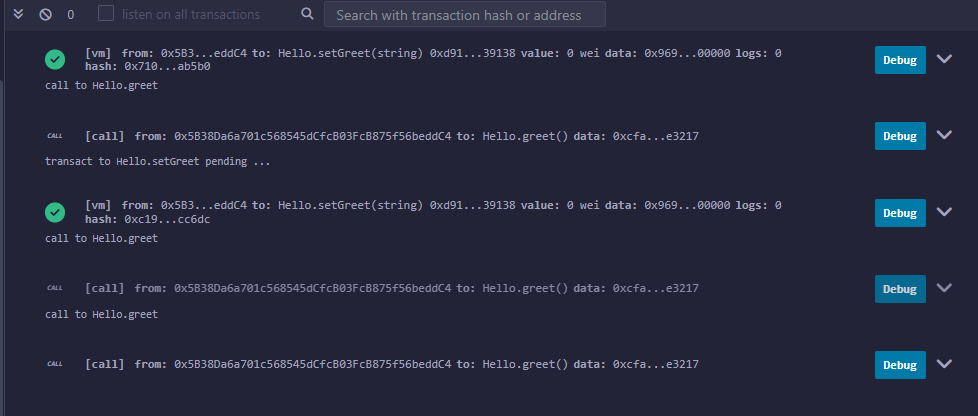


Step 3: Deploy and run transaction option in the left pane and setgreet and then greet

**Output:**





**4) Abstract Class in solidity**

//SPDX-License-Identifier:GPL-3.0

pragma solidity ^0.5.17;

contract A {

function getValue() public view returns (uint256);

}

contract B is A {

function getValue() public view returns (uint256)

{

uint256 x = 10;

uint256 y = 20;

uint256 result = x\*y;

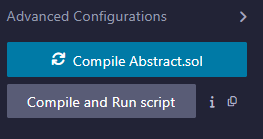
return result;

}

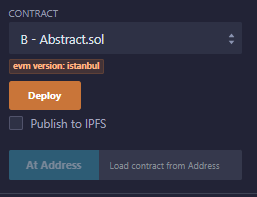
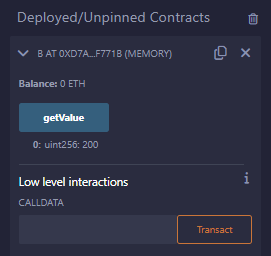
}

Note: Run only contract B.

Compile:



Deploy:

**5) Interfaces in Solidity**

//SPDX-License-Identifier:GPL-3.0

pragma solidity ^0.5.17;

interface A {

function getValue() external view returns (uint256);

}

contract B is A {

function getValue() external view returns (uint256)

{

uint256 x = 10;

uint256 y = 20;

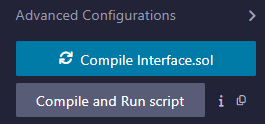
uint256 result = x\*y;

return result;

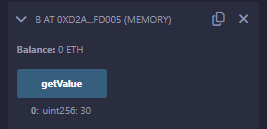
}

}

Compile:



Deploy:



**PRACTICAL 9**

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

1. **Libraries**
2. **Error Handling**

**i) Libraries**

* Reuse certain code
* Stateless entity, never alter rosmodify state of contract
* Reduces Gas burn

1. **Util.sol**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

library Util {

function concat(string memory x,string memory y) public pure returns (string memory) {

return string.concat(x, y);

}

function add(uint x, uint y) public pure returns (uint) {

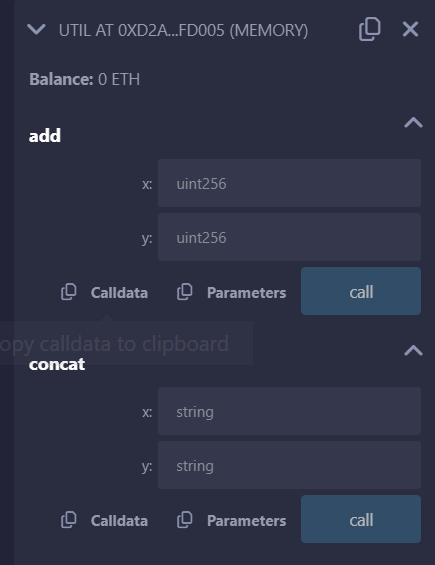
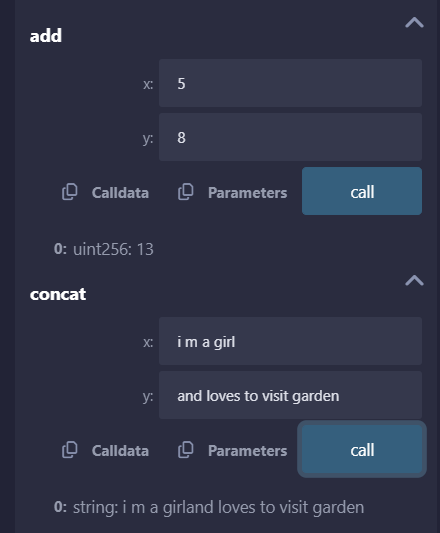
return x + y;

}

}

**Output:**

**Before: After:**

** **

**b) Test.sol**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

import "util.sol";

contract ImportHere {

using Util for uint;

using Util for string;

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

return a.add(b);

}

function concat(string memory x,string memory y) public pure returns (string memory) {

return x.concat(y);

}

}

**Output:**

**A screenshot of a chat

Description automatically generated**

**Error Handling**

1. **Require:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract requireStatement {

function checkInput(uint \_input) public pure returns(string memory){

require(\_input >= 0, "invalid uint8");

require(\_input <= 255, "invalid uint8");

return "Input is Uint8";

}

function odd(uint \_input) public pure returns(bool){

require(\_input % 2 != 0);

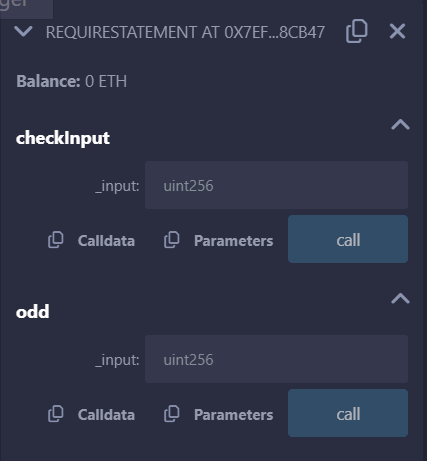
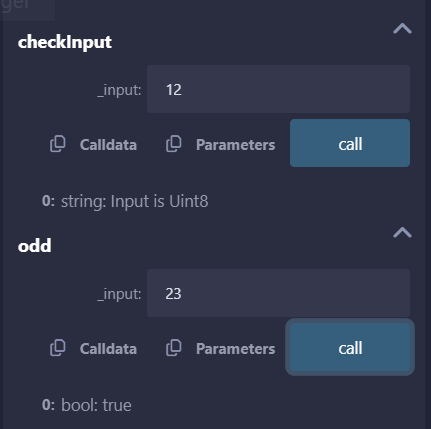
return true;

}

}

**OUTPUT:**

**Before: After entering inputs:**

** **

1. **Assert:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract assertStatement {

bool result;

function checkOverflow(uint \_num1, uint \_num2) public {

uint sum = \_num1 + \_num2;

assert(sum<=255);

result = true;

}

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

if(result == true){

return "No Overflow";

}

else{

return "Overflow exist";

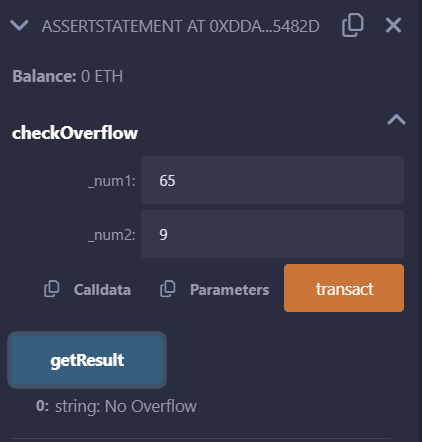
}

}

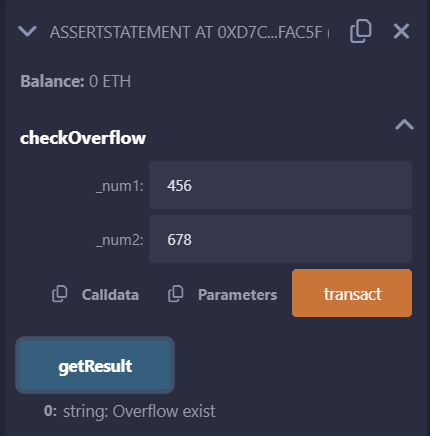
}

**Output:**

**If the number of sums is lesser or equal to 255 it return No Overflow**

****

**If the number of sum is greater than 255 it return Overflow Exist**

****

1. **Revert:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract revertStatement {

function checkOverflow(uint \_num1, uint \_num2) public pure returns(string memory, uint){

uint sum = \_num1 + \_num2;

if(sum < 0 || sum > 255){

return ("Overflow", sum);

revert(" Overflow Exist");

}

else{

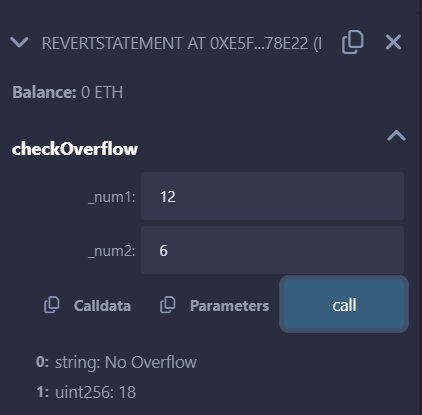
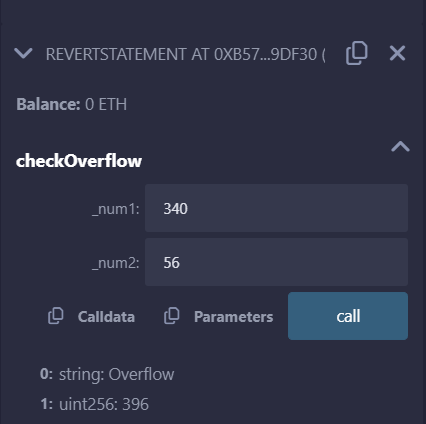
return ("No Overflow", sum);

}

}

}

**Output:**

 ****