|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Write a program to demonstrate RSA and mining [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |
|  | | |

For RSA

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("Pooja Anchewar")

print(p\_key)

print(priv\_key)

et=time.time()

en=et-st

print(en)

For Mining

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("Pooja Anchewar",2)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Demonstrate the running of the blockchain node [40 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Create your own blockchain and demonstrate its use. [40] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Use Remix IDE to develop an deploy solidity code on Ethereum VM. [20 Marks] | | |
| Q.2 Write a program for a simple client class that generates the private and public keys by  using the built-in Python RSA algorithm and test it [10 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

Q2

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("Pooja Anchewar")

print(p\_key)

print(priv\_key)

et=time.time()

en=et-st

print(en)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Write a program for a transaction class to send and receive money and test it. [20 Marks] | | |
| Q.2 Write a program for Blockchain in Python to Create multiple transactions and display  them. [20 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

Q1

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,Pooja-->\n'+dict['sender'])

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

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

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

transaction=[]

Pooja=Client()

Ashwini=Client()

t1=Transaction(Pooja,Ashwini.identity,15)

t1.sign\_tran()

display\_tran(t1)

Q2

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,Pooja-->\n'+dict['sender'])

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

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

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

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

Pooja=Client()

Ashwini=Client()

for transaction in transactions:

t=Transaction(Pooja,Ashwini.identity,transaction)

t.sign\_tran()

display\_tran(t)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Write a program to Create a blockchain, a genesis block and execute it. [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

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('Charlie','Pooja',10)

my\_blockchain.add\_transaction('Sri','Pooja',5)

my\_blockchain.add\_transaction('Alice','Bob',10)

my\_blockchain.add\_transaction('Bob','Charlie',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)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Write a program to Create a mining function and test it [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

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("Pooja Anchewar",2)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Add blocks to the miner and dump the blockchain. [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

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('Charlie','Pooja',10)

my\_blockchain.add\_transaction('Sri','Pooja',5)

my\_blockchain.add\_transaction('Alice','Bob',10)

my\_blockchain.add\_transaction('Bob','Charlie',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)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement and demonstrate the use of the following in Solidity:  Functions, Function Modifiers, View functions, Pure Functions [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**i)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;

}

}

**ii)Function Modifier:**

**Code:**

pragma solidity ^0.8.0;

contract UserInfo {

string public userName;

uint public userAge;

address public owner;

constructor() {

owner = msg.sender;

}

modifier onlyOwner() {

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

\_;

}

function setUserInfo(string memory name, uint age) public {

userName = name;

userAge = age;

}

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

return (userName, userAge);

}

}

**ii)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;

}

}

**iii)Pure Functions:**

// 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);

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement and demonstrate the use of the following in Solidity:  Fallback Function, Function Overloading, Mathematical functions, Cryptographic  functions. [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**iv)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;

}

}

**v)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);

}

}

**vi)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";

}

}

**vii)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));

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement and demonstrate the use of the following in Solidity:  Variable, Operators and Loops[40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

### **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

### **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

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement and demonstrate the use of the following in Solidity: Decision Making, Strings, Arrays [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

### **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 generated5

**5) 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

>> Click on Greet

A screenshot of a computer

Description automatically generated

**Array:**

**Code**

pragma solidity ^0.8.0;

contract SimpleArrayExample {

uint[] public numbers;

function addNumber(uint \_number) public {

numbers.push(\_number);

}

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

return numbers[\_index];

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Implement and demonstrate the use of the following in Solidity:  Libraries, and Error handling [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**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) {

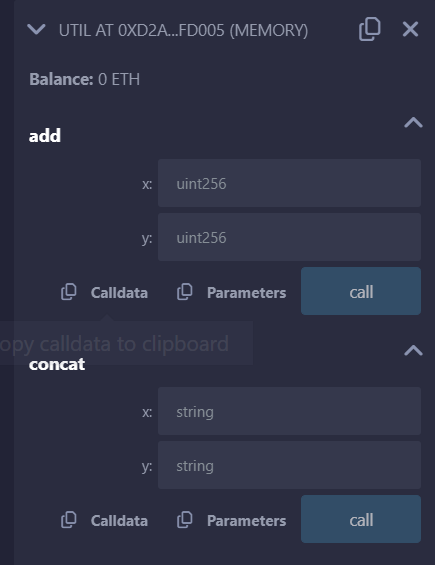
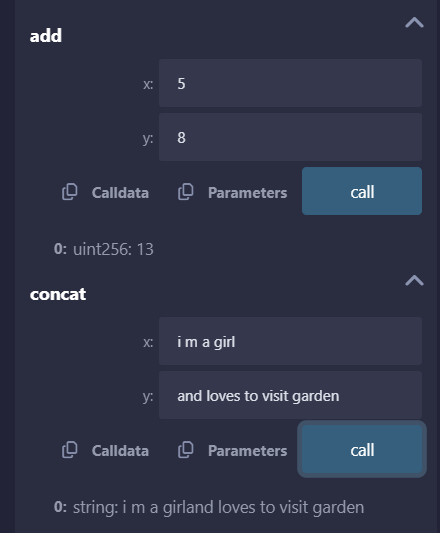
return x + y;

}

}

**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:**

**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);

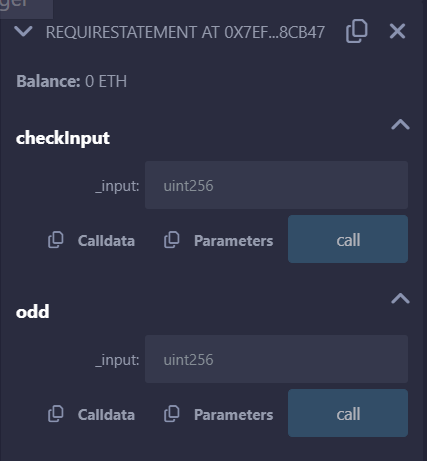
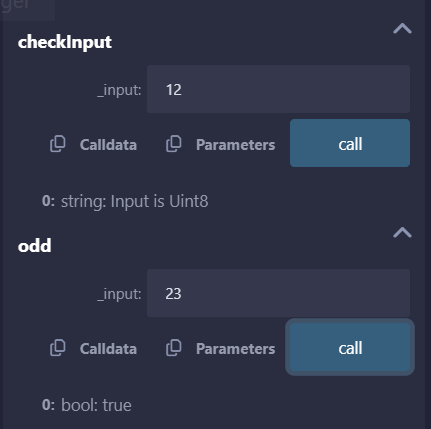
return true;

}

}

**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";

}

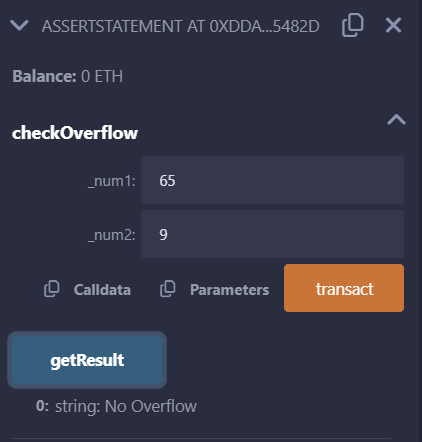
else{

return "Overflow exist";

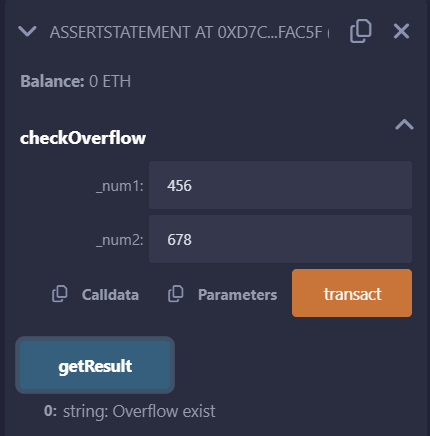
}

}

}

**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{

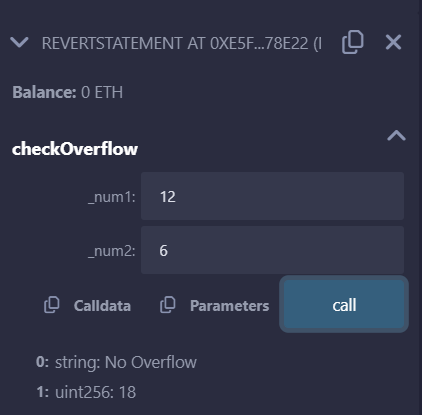
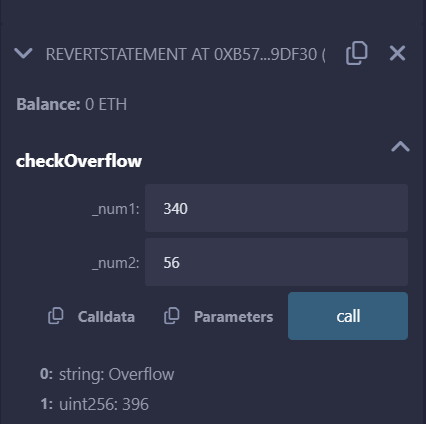
return ("No Overflow", sum);

}

}

}

**Output:**

 ****

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Implement and demonstrate the use of the following in Solidity:  Constructors[20 Marks] | | |
| Q.2 Implement and demonstrate the use of the following in Solidity: Interfaces [20 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

**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";

}

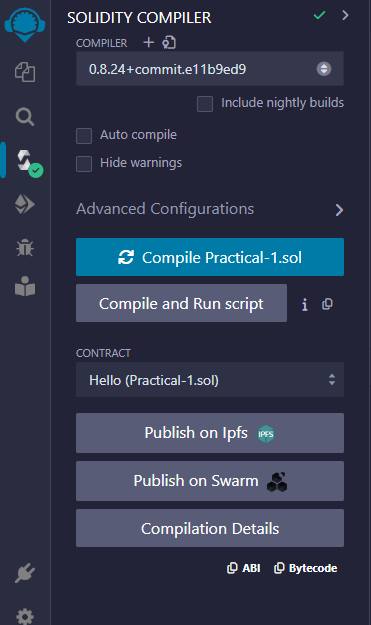
function setGreet (string memory \_newgreet ) public{

greet = \_newgreet;

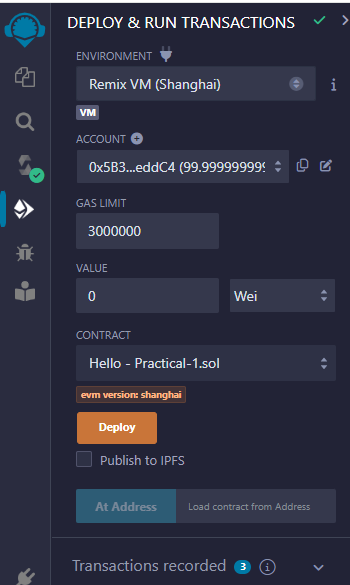
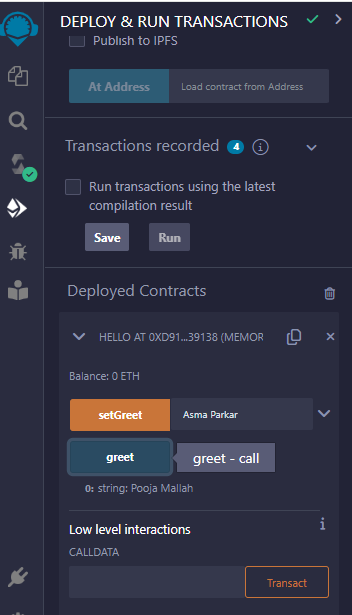
}

}

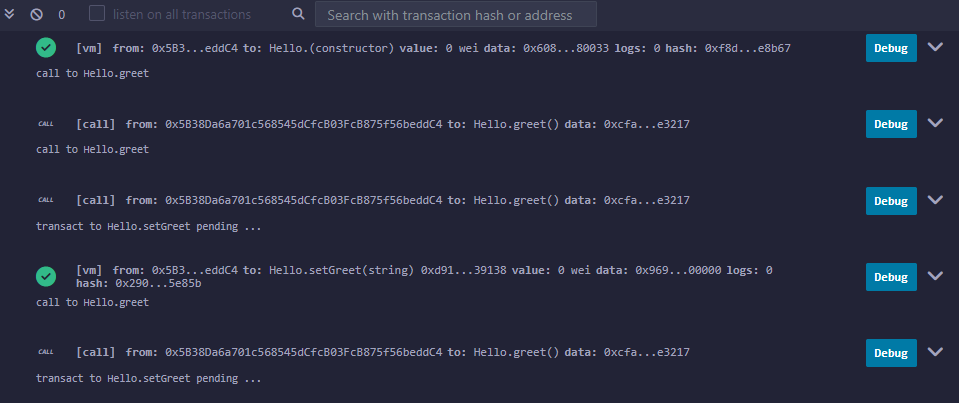
Step 2: Complie the code with solidity option in the left pane of Remix IDE.

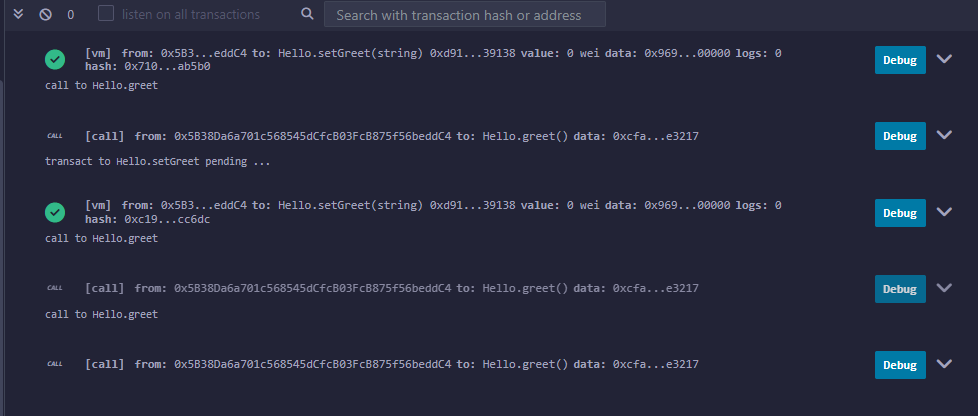


Step 3: Deploy and run transaction option in the left pane and setgreet and then greet

**Output:**





**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;

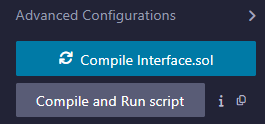
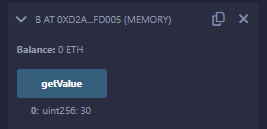
uint256 result = x\*y;

return result;

}

}

Compile: Deploy:

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Implement and demonstrate the use of the following in Solidity:  Contracts, Inheritance [20 Marks] | | |
| Q.2 Implement and demonstrate the use of the following in Solidity: Abstract Contracts [20 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

**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 {

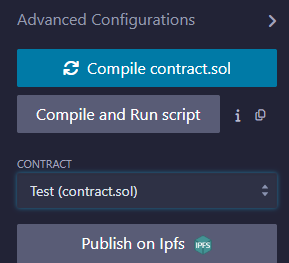
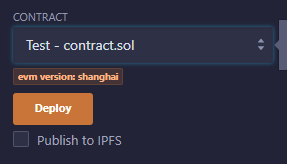
money=\_amt;

}

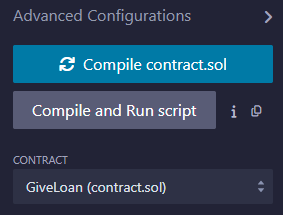
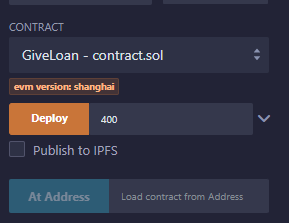
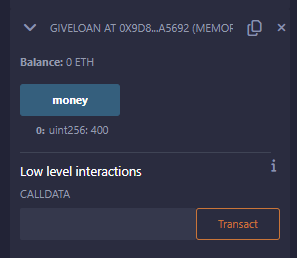
}

**OUTPUT:**

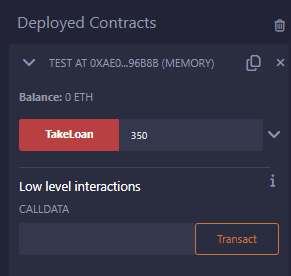
Compile and deploy test

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();

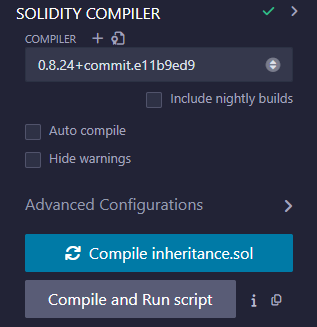
return cc.getVal();

}

}

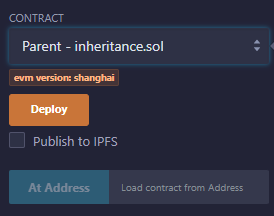
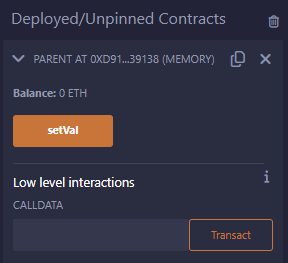
**OUTPUT:**

Compile



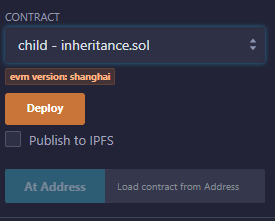
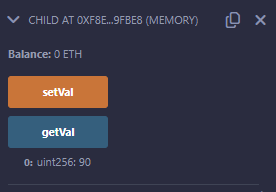
Run:

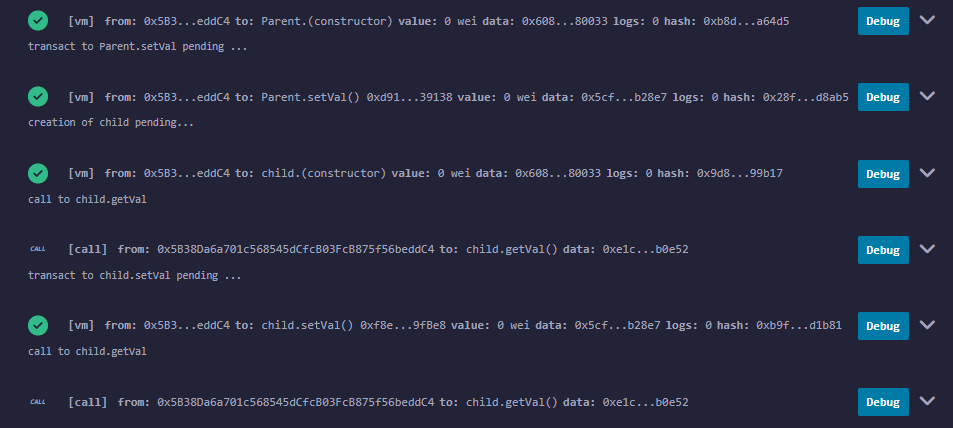
First DEPLOY the parent contract

Deploy the child contract

Select setVal>getVal



**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;

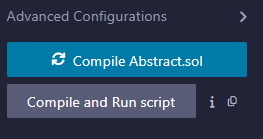
return result;

}

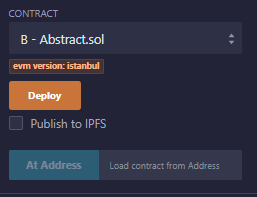
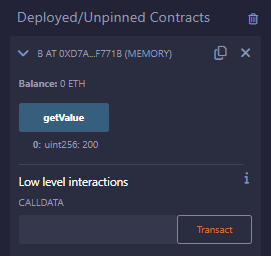
}

Note: Run only contract B.

Compile:



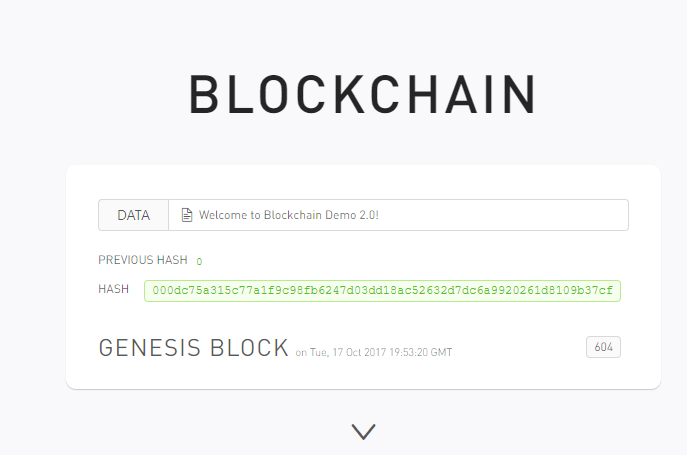
Deploy:

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 **Perform Visual demonstration of Blockchain.** [20 Marks] | | |
| Q.2 Demonstrate any solidity deployment [20 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

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

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

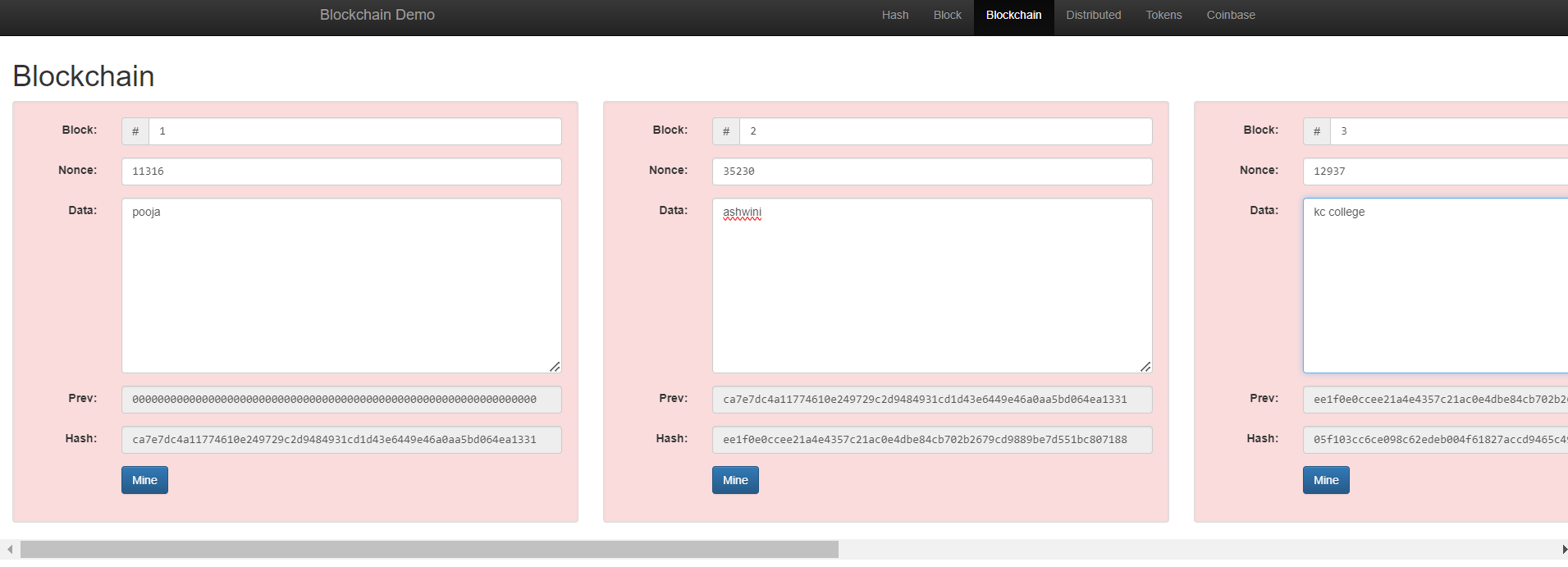


Step 2: Create the block

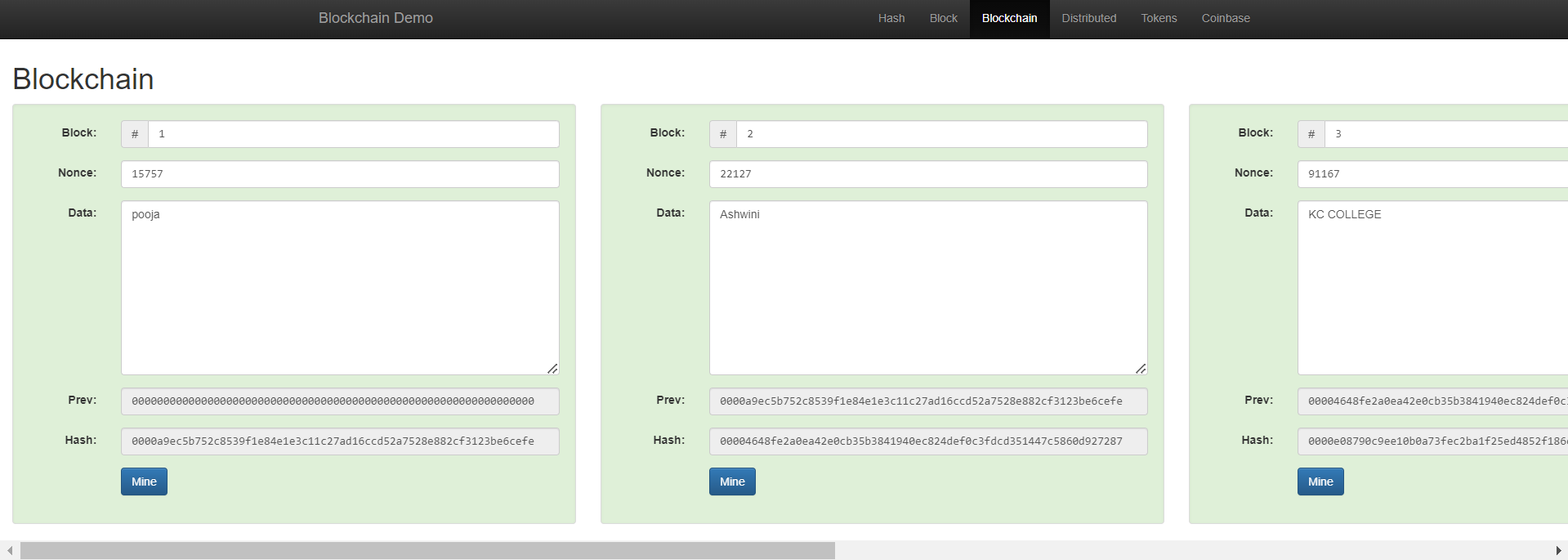


Step 3: Add the Data in blocks (Nonce : hash is generated through the nonce)

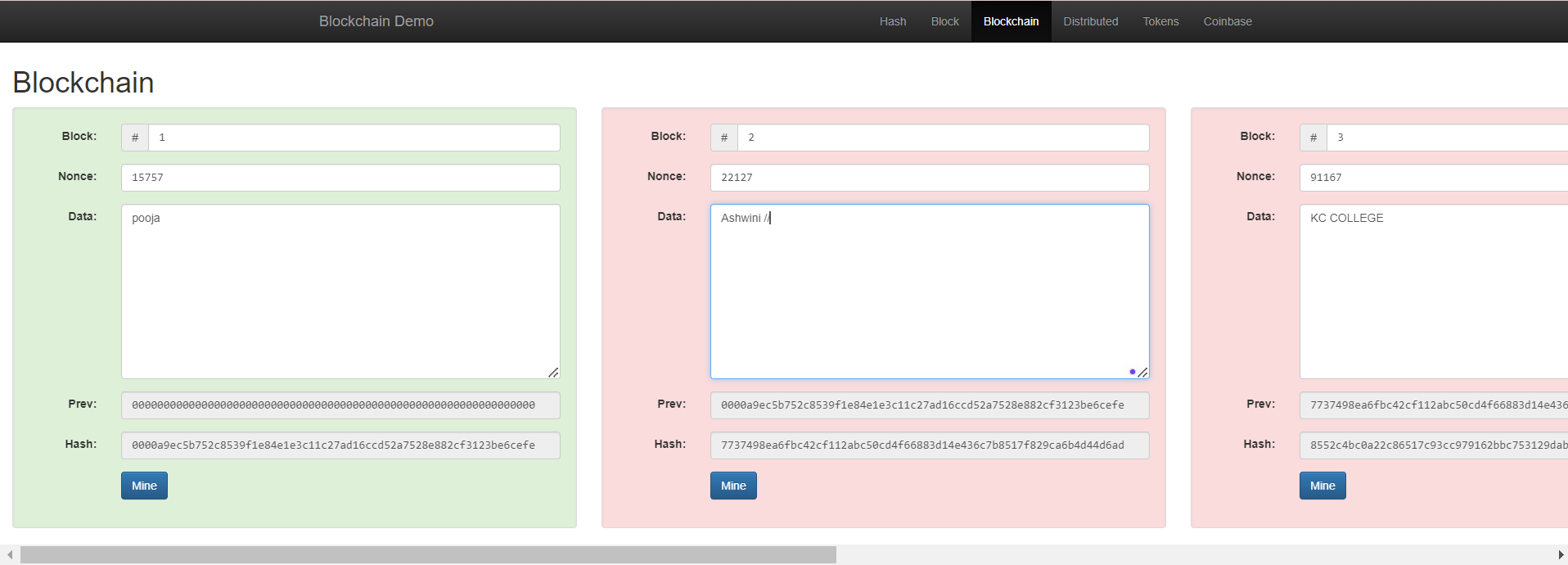
Open the link <https://andersbrownworth.com/blockchain/hash>



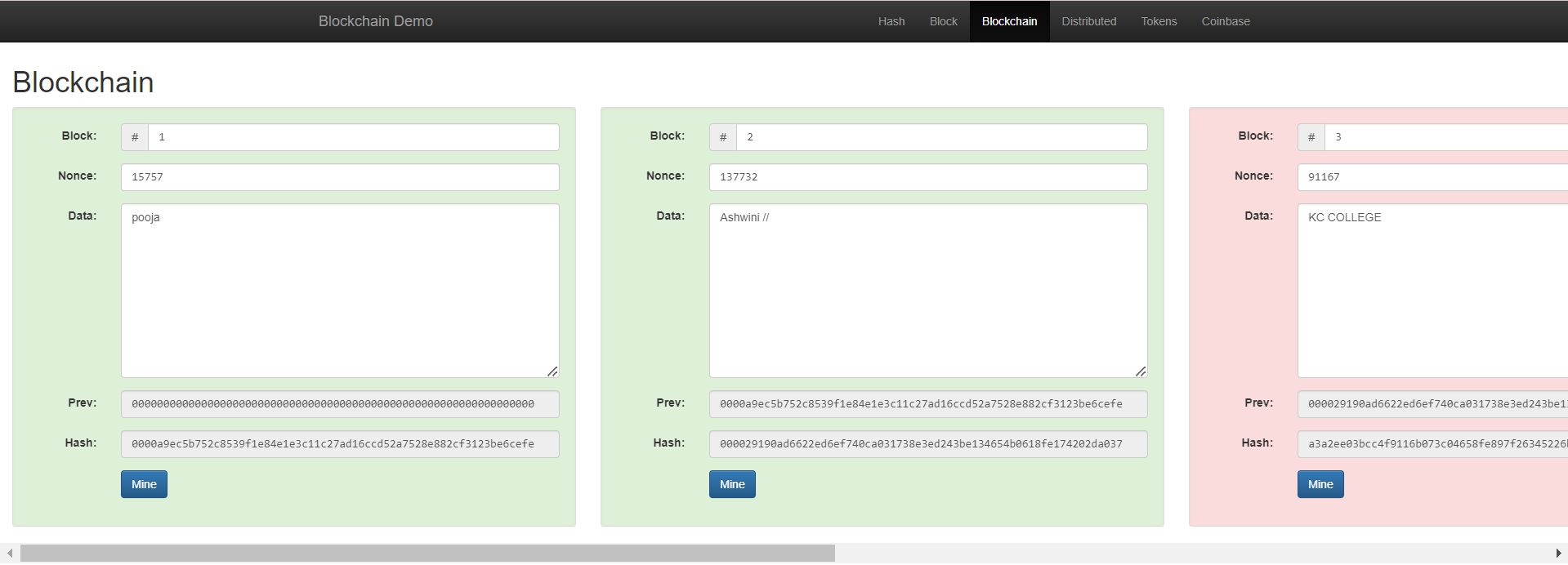
Step 4: Then click on mine



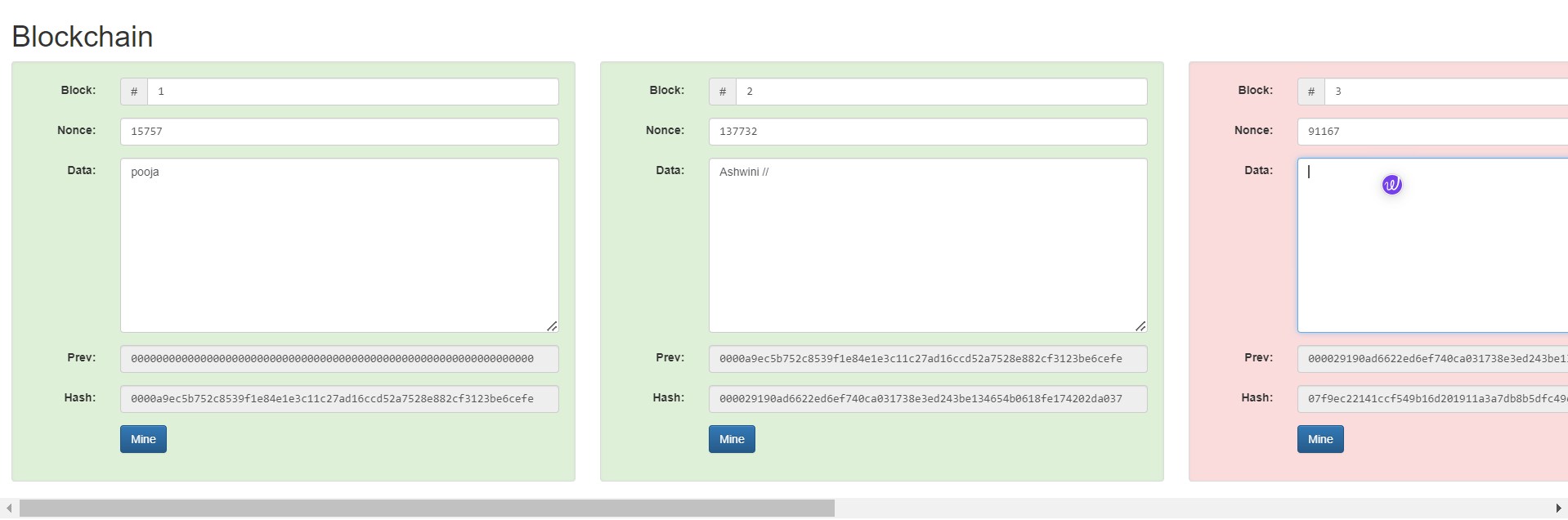
Step 5: Change the block 2 data



Step6 :When we change the block 2 data, the nonce of block 2 will also change



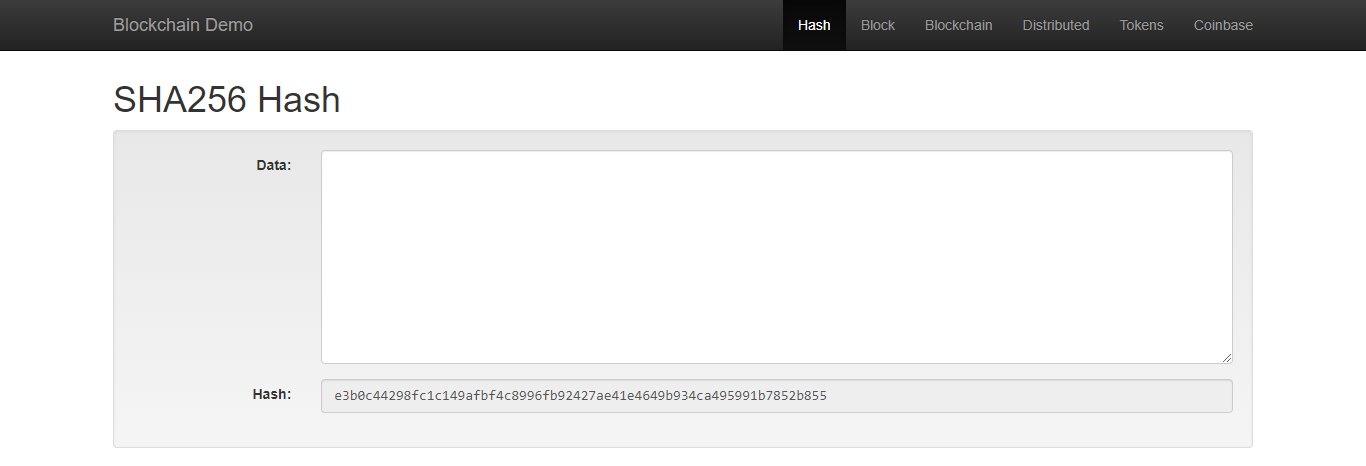
Step 7: Then mine the block



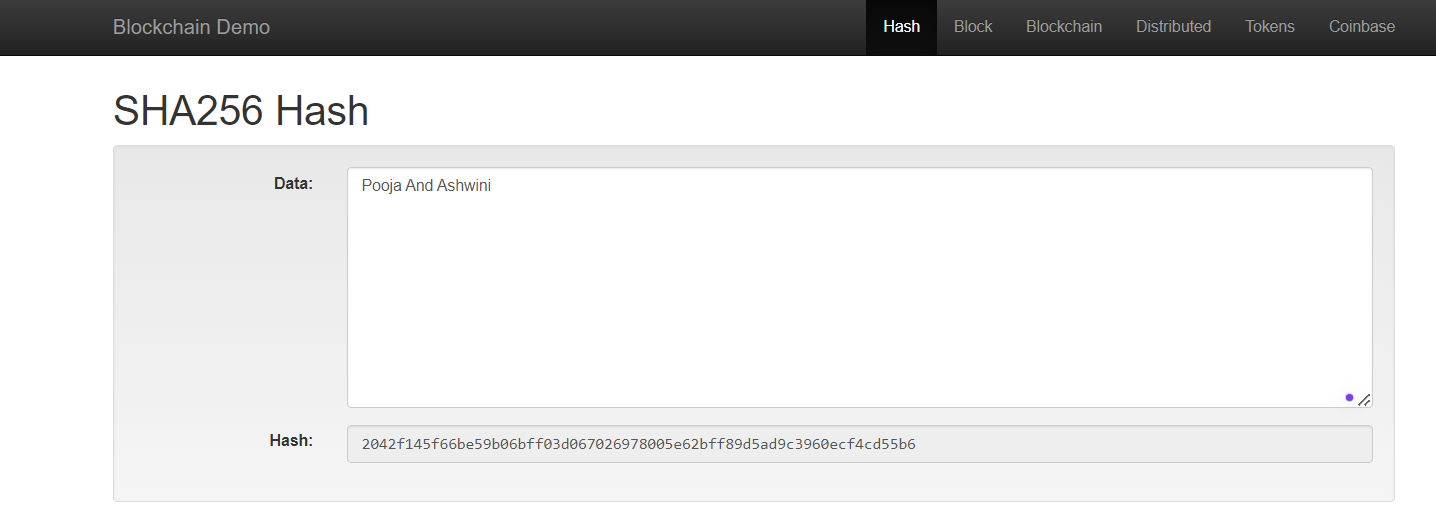
SHA256 Hash

256 bites: 32 bytes

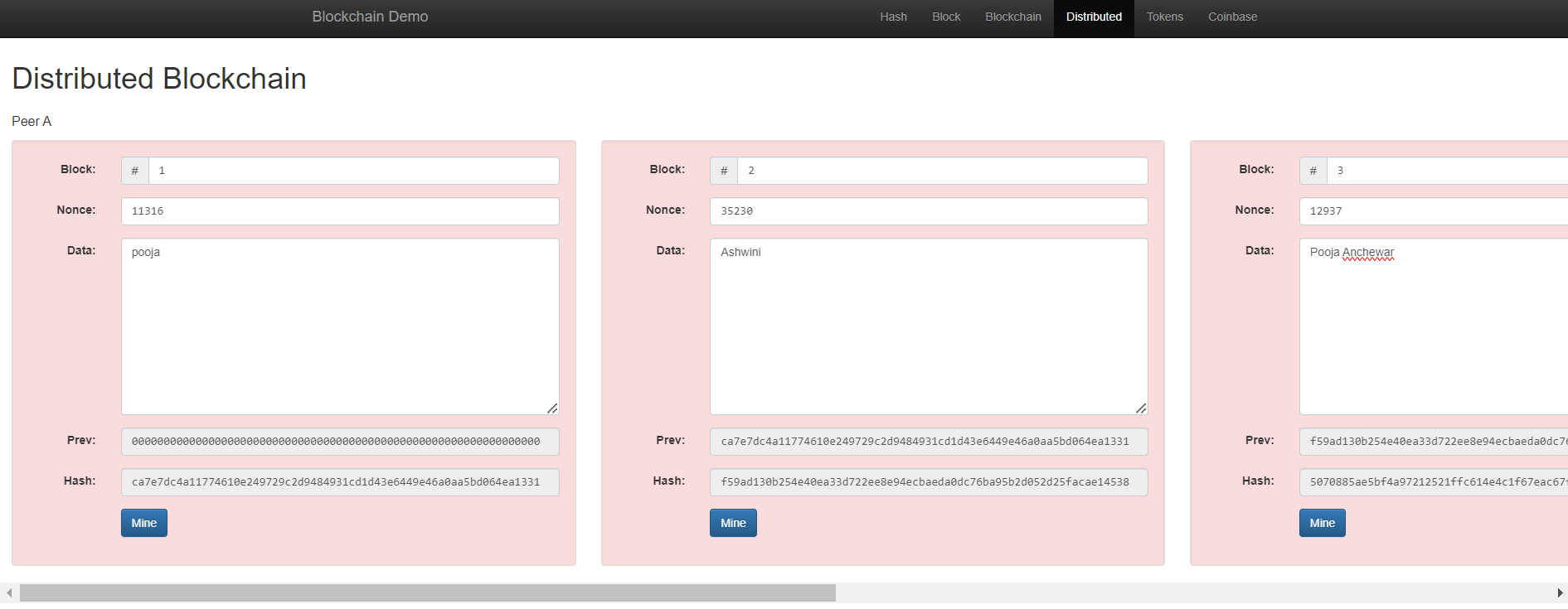
Step 8:Go to the HASH Tab

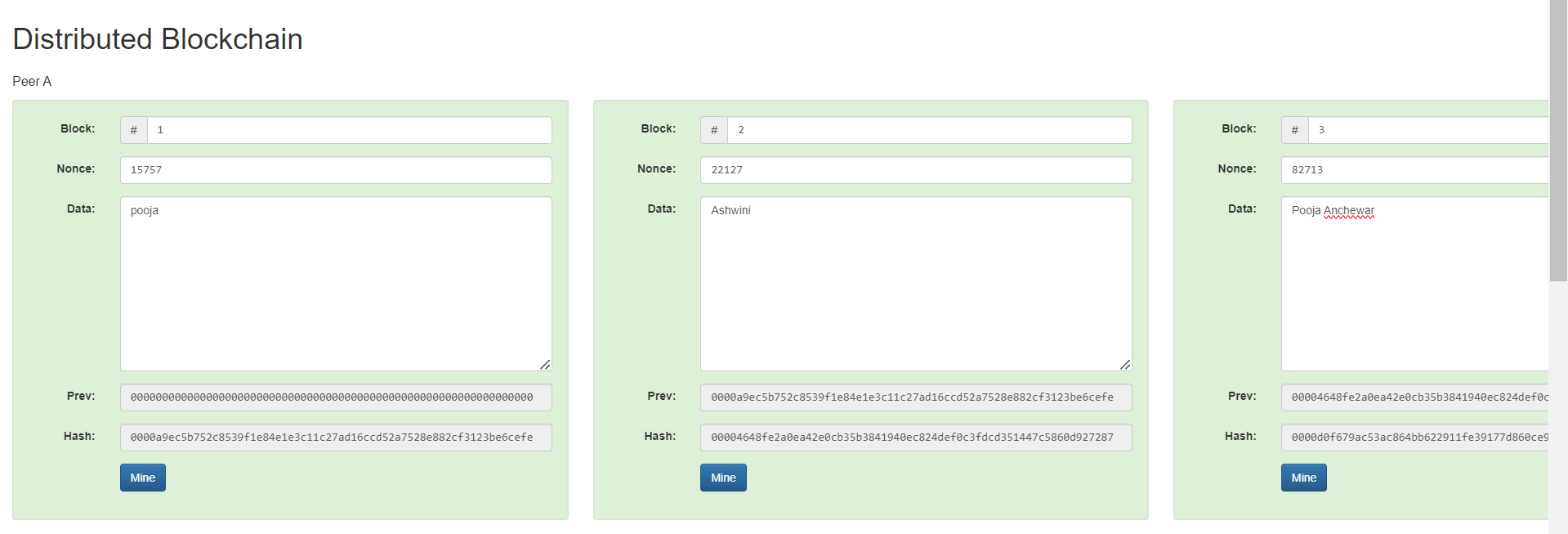


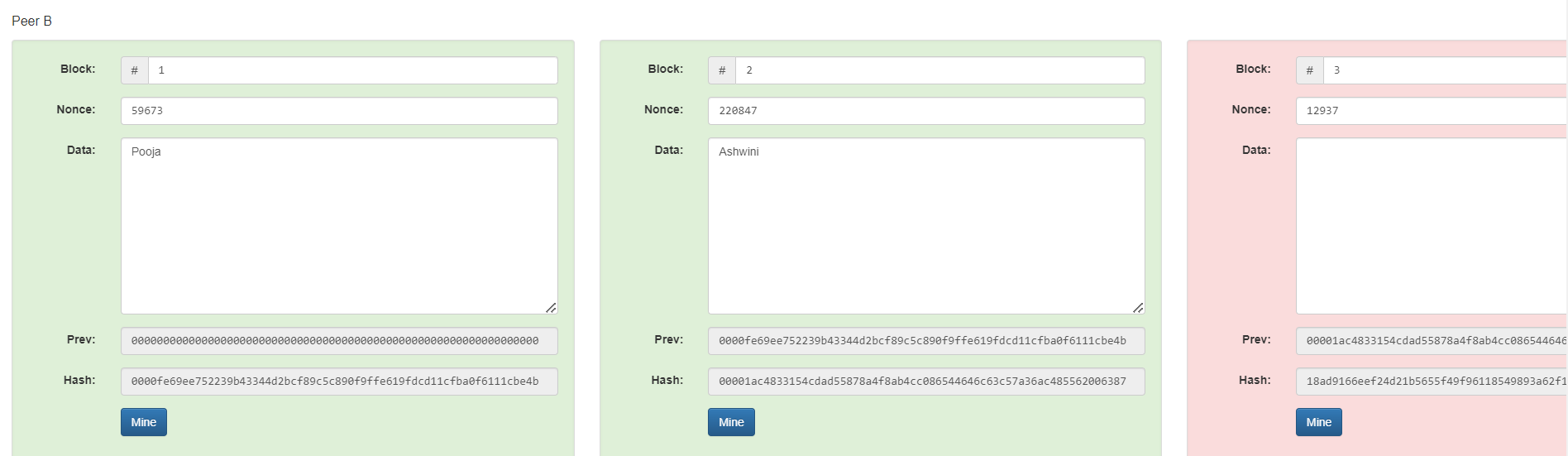
Step 9: When we enter the data the hash will change



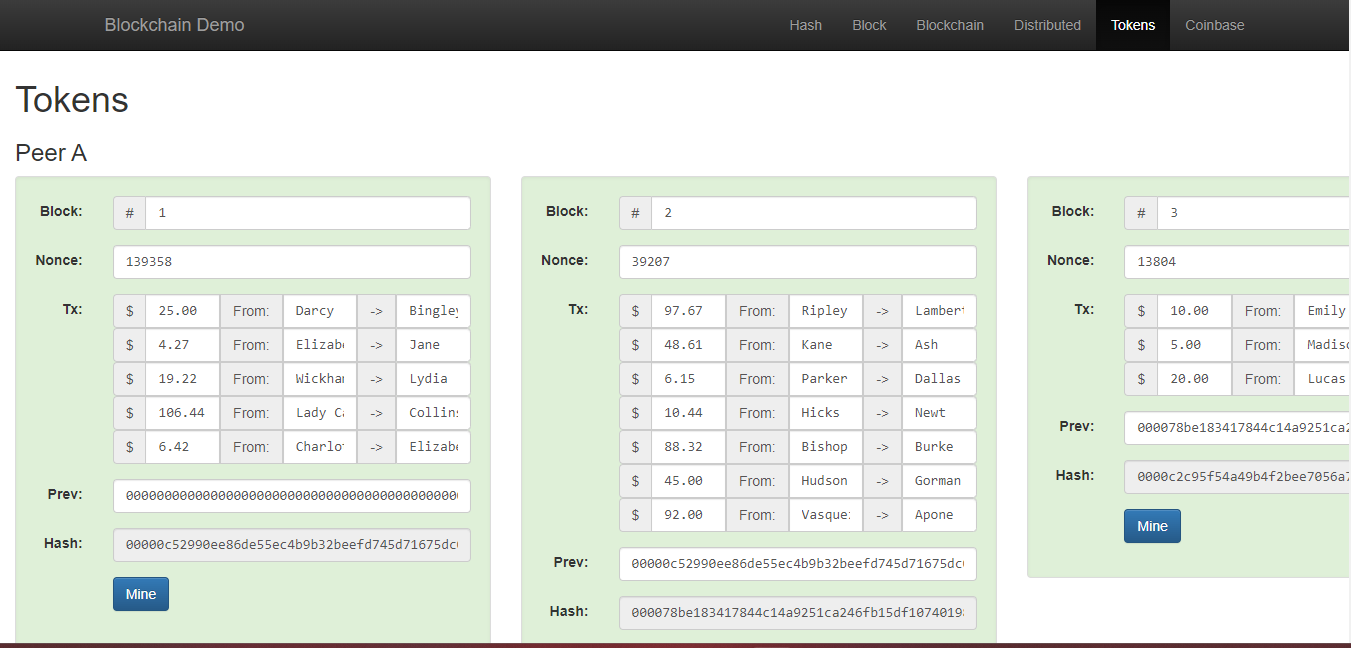
Step 10: Distributed Blockchain

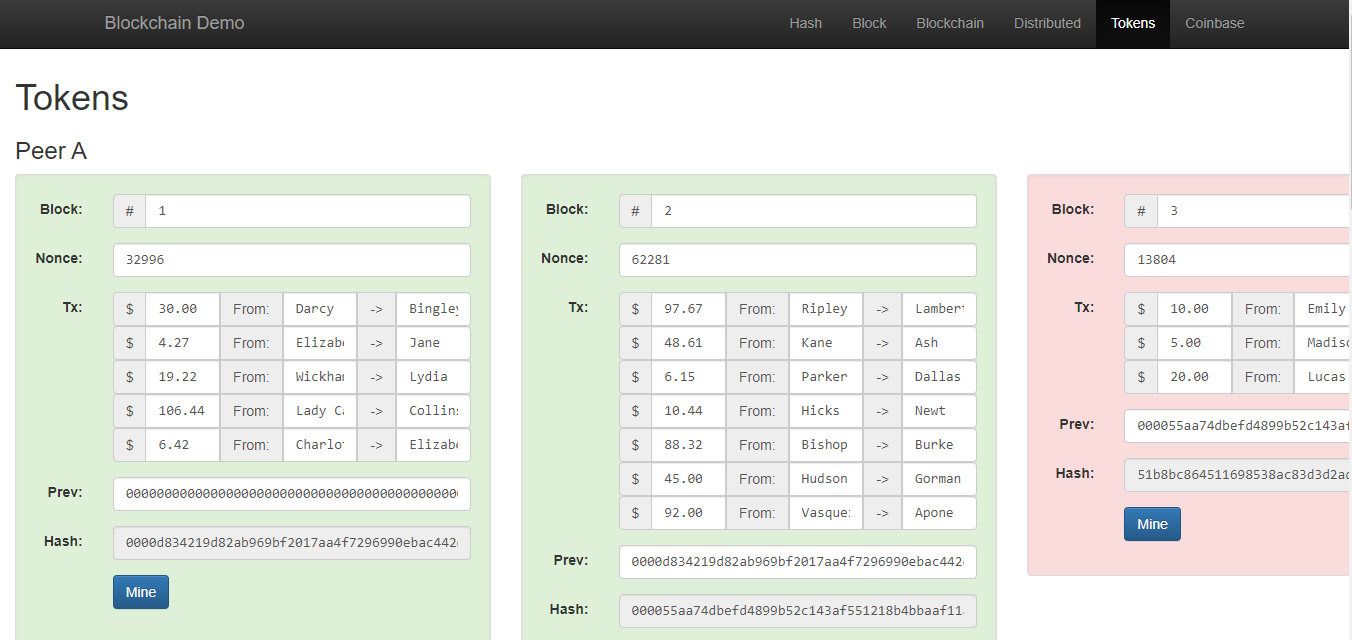


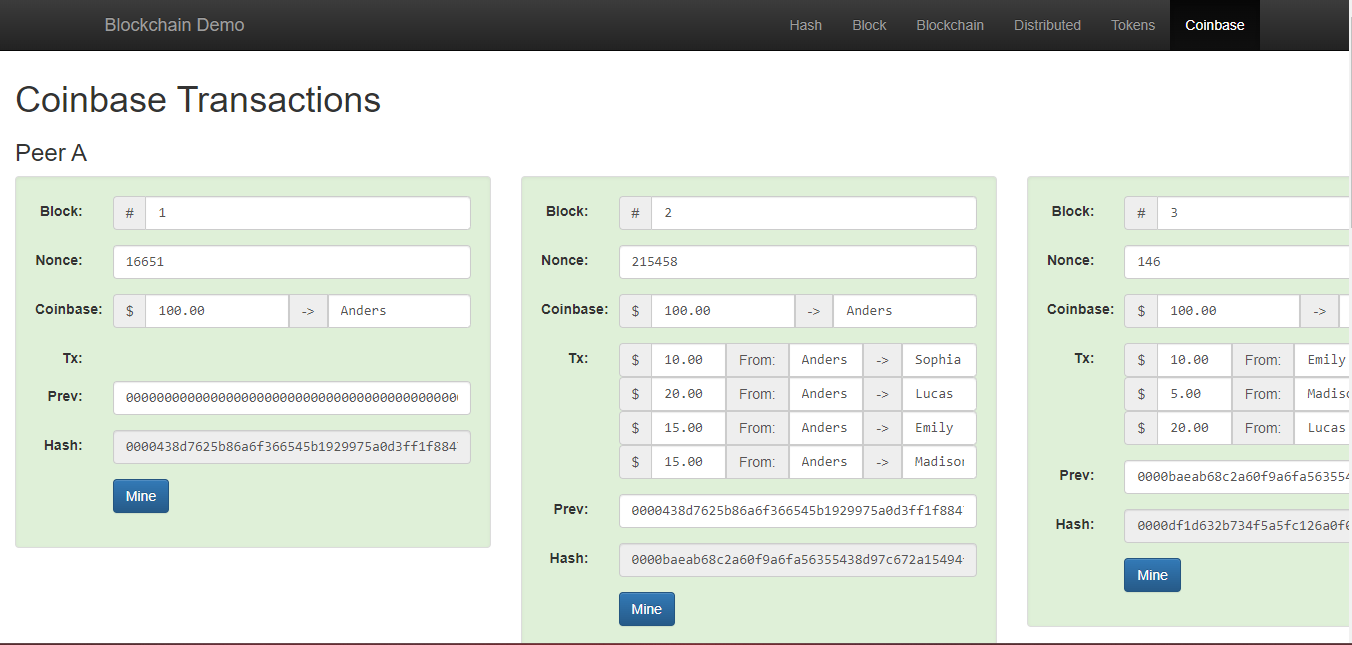


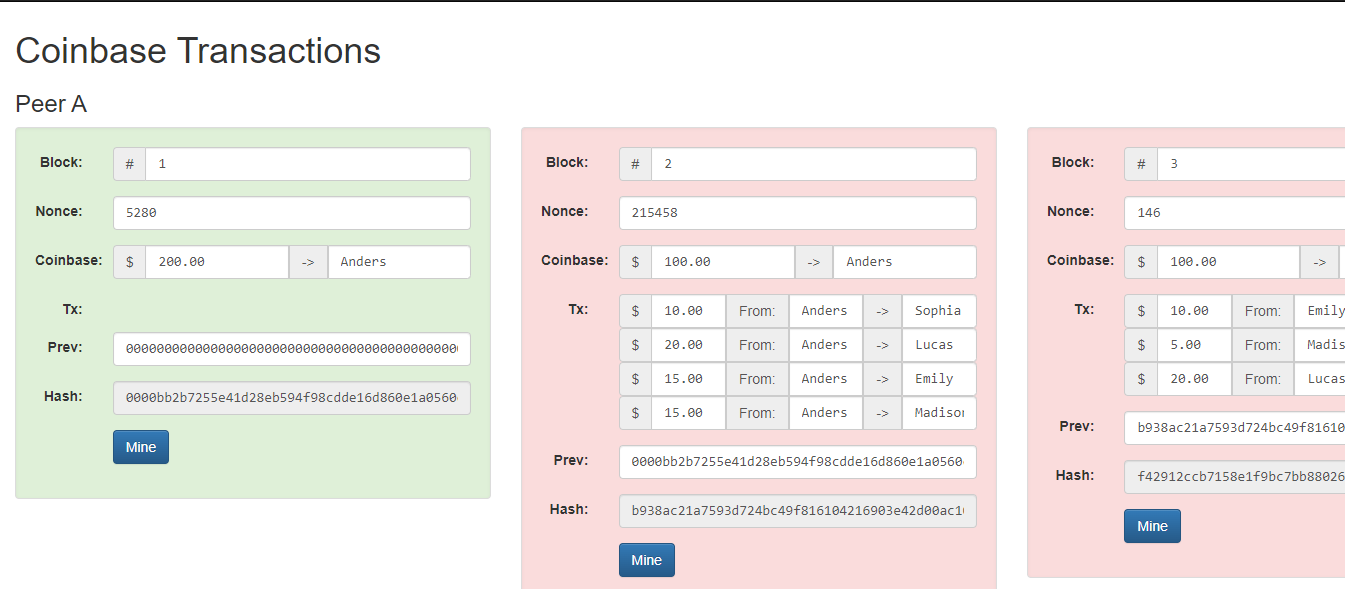


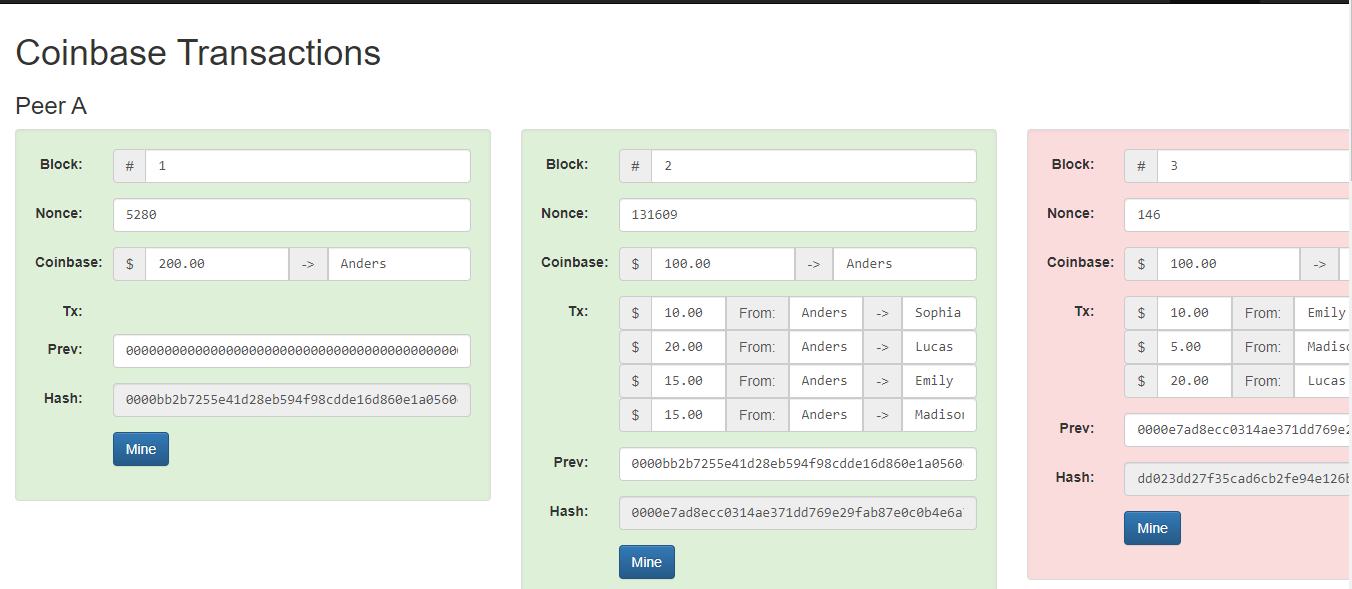
Step 11: Tokens











Q2

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement solidity code to generate random number [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**viii)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)));

}

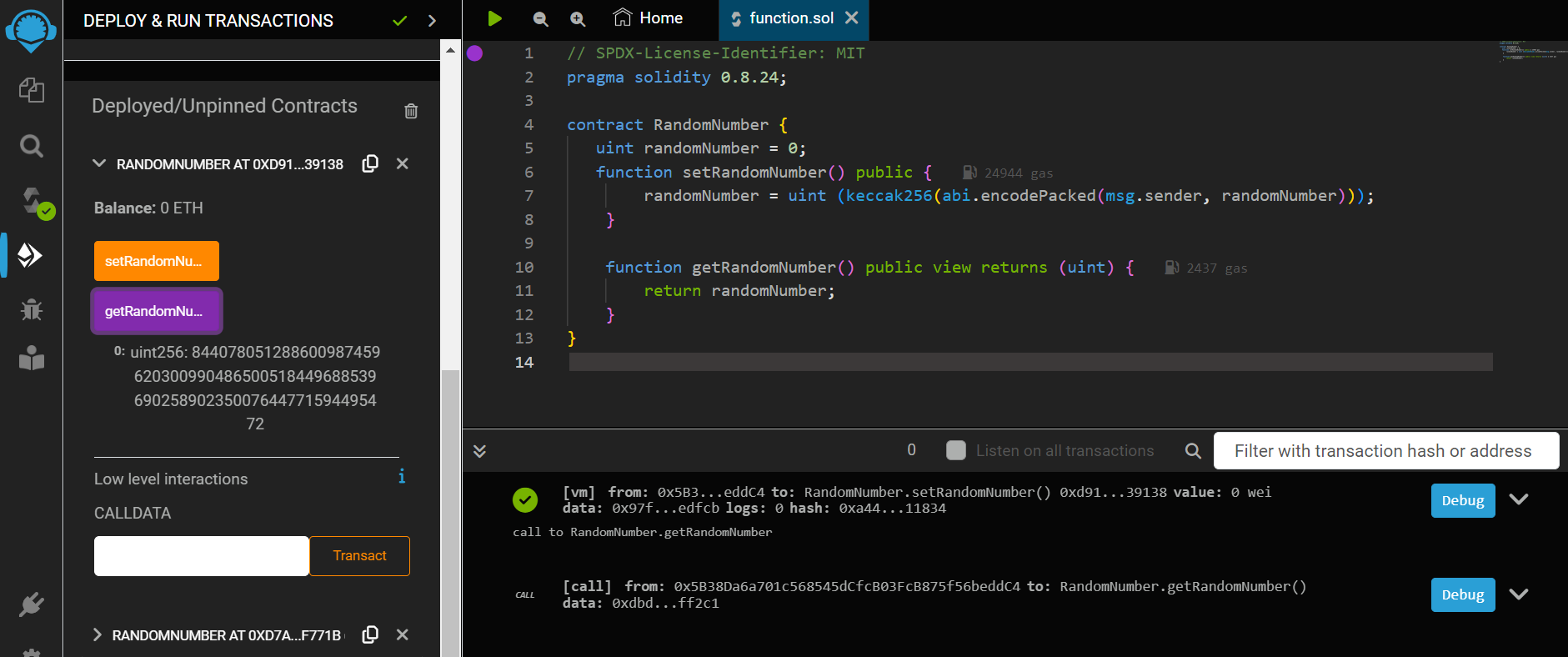
function getRandomNumber() public view returns (uint) {

return randomNumber;

}

}

**Output:**

****

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate Even odd in solidity and deploy it[40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

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."));

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate Pure Function and Mathematical Function [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**iii)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);

}

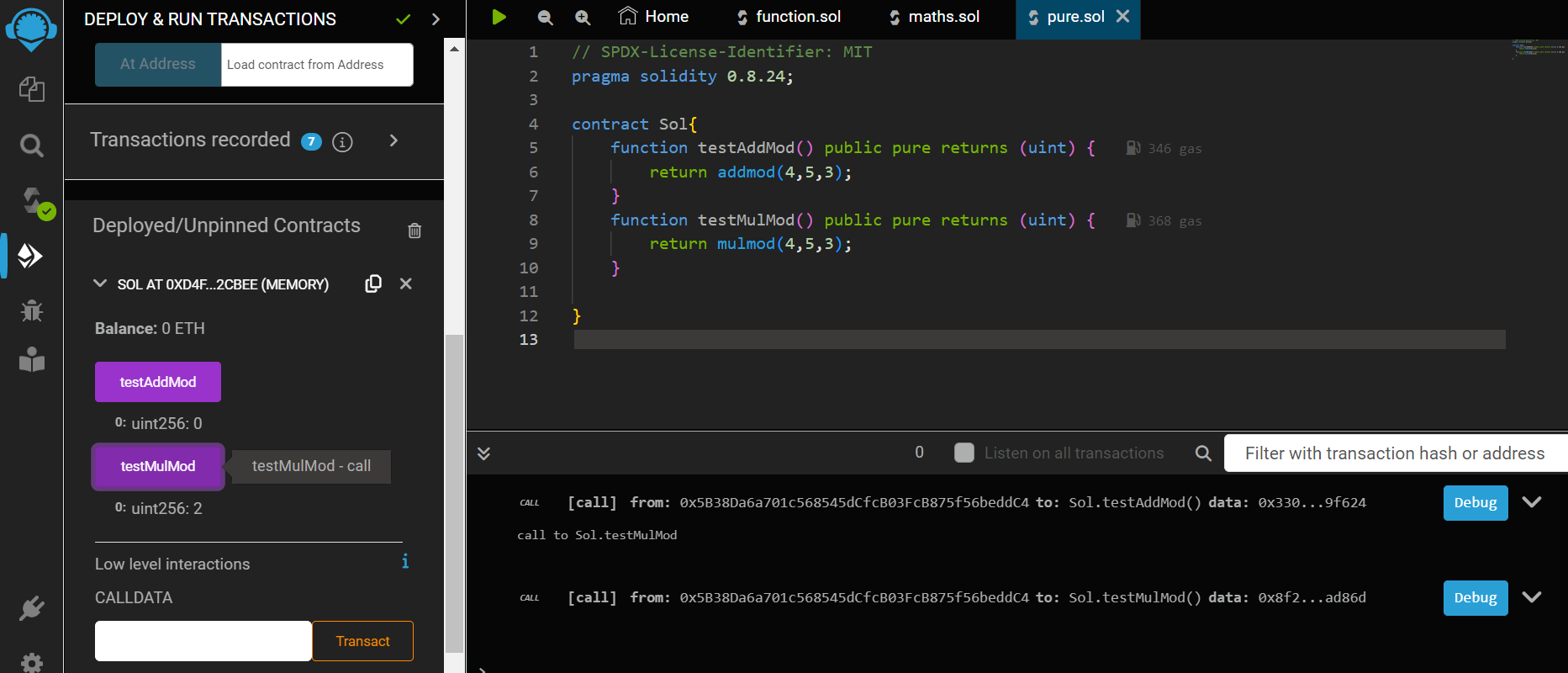
function testMulMod() public pure returns (uint) {

return mulmod(4,5,3);

}

}

**Output:**

****

**vi)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";

}

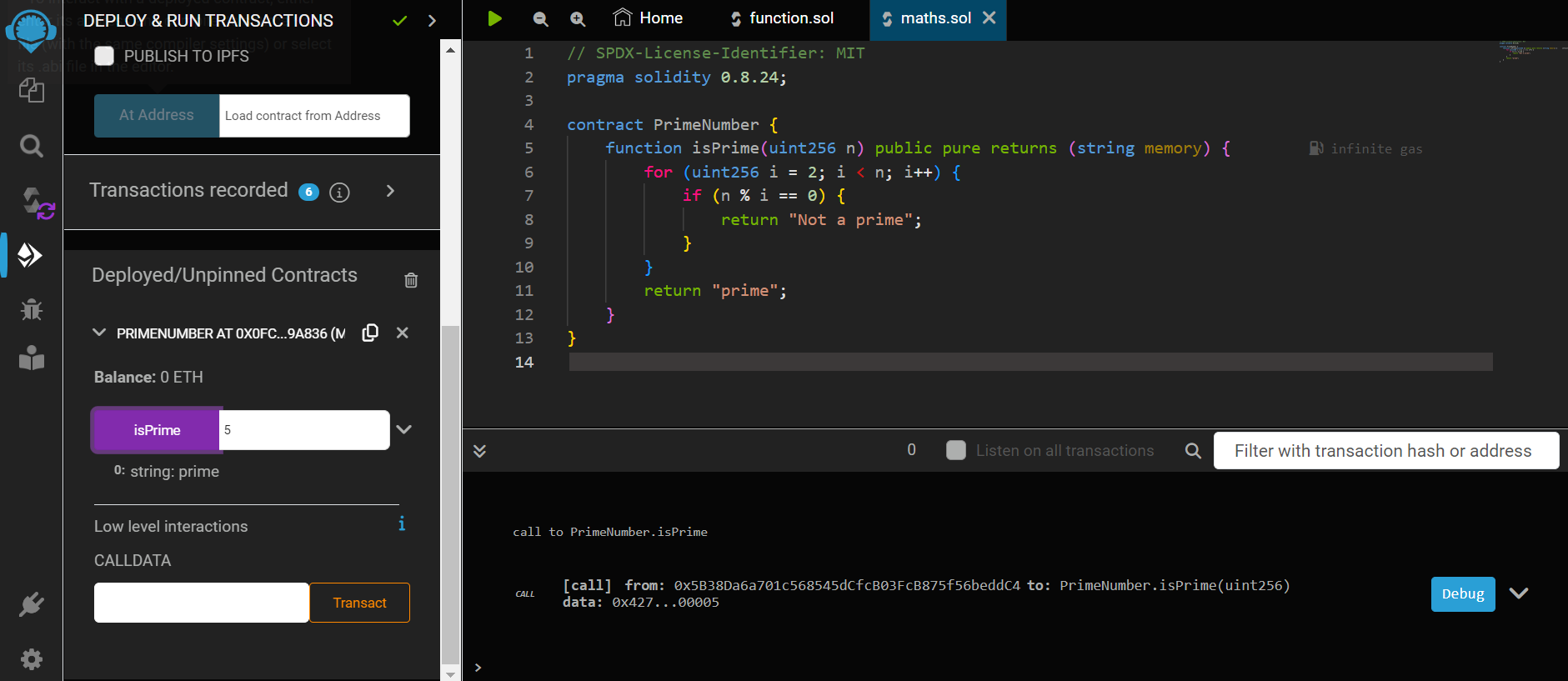
}

return "prime";

}

}

**Output:**

****

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Demonstrate any solidity code for multiple contract[30 Marks] | | |
| Q.2 Implement Prime Number in Solidity [10 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

Q1

**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 {

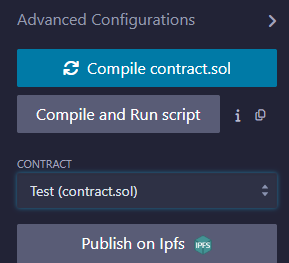
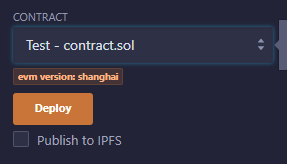
money=\_amt;

}

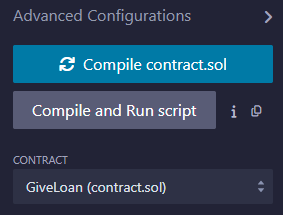
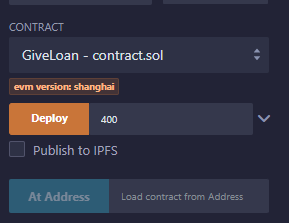
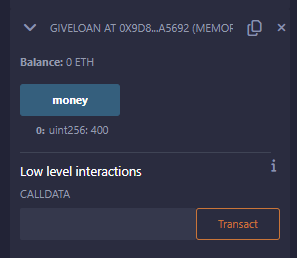
}

**OUTPUT:**

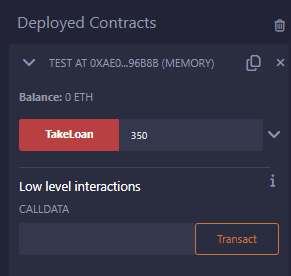
Compile and deploy test

Compile and deploy by giving value and click money()

Test.TakeLoan()



Q2

**vi)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";

}

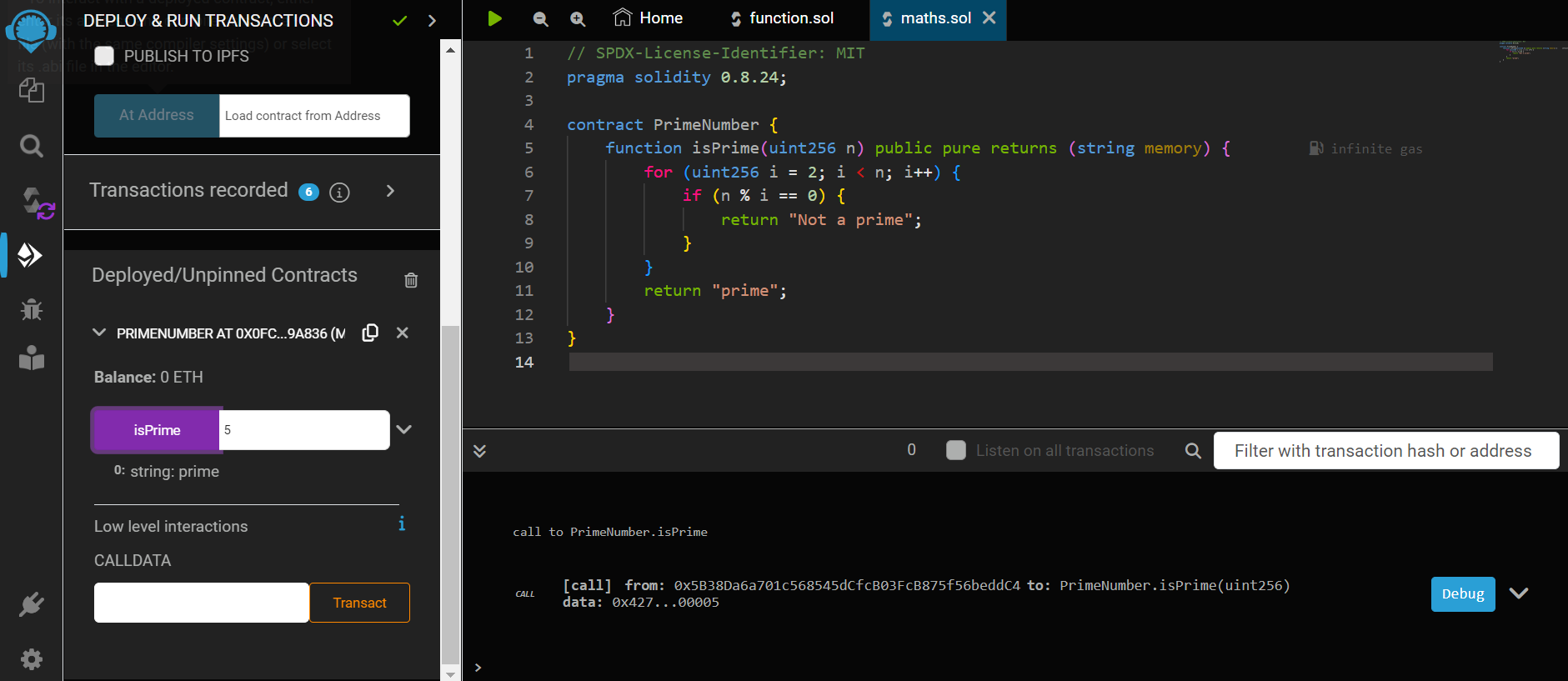
}

return "prime";

}

}

**Output:**

****

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1 Demonstrate solidity variable [30 Marks] | | |
| Q.2 Demonstrate solidity deployment for division[10 Marks] | | |
| Q.3 Journal [5 Marks] | | |
| Q.4 Viva [5 Marks] | | |

### **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”.

}

}

Q2)

// 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;

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Implement Genesis Block and dump on Blockchain[40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**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('Charlie','Pooja',10)

my\_blockchain.add\_transaction('Sri','Pooja',5)

my\_blockchain.add\_transaction('Alice','Bob',10)

my\_blockchain.add\_transaction('Bob','Charlie',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)

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate solidity to divide the number[40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

// 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;

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate solidity code to implement Fallback function and Mapping[40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**iii) Mappings**

**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];

}

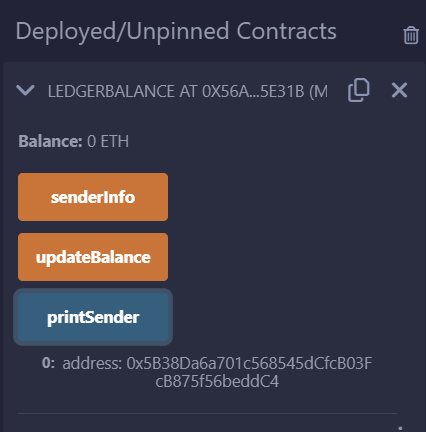
function printSender() public view returns(address){

return msg.sender;

}

}

**Output:**

****

**iv)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!");

}

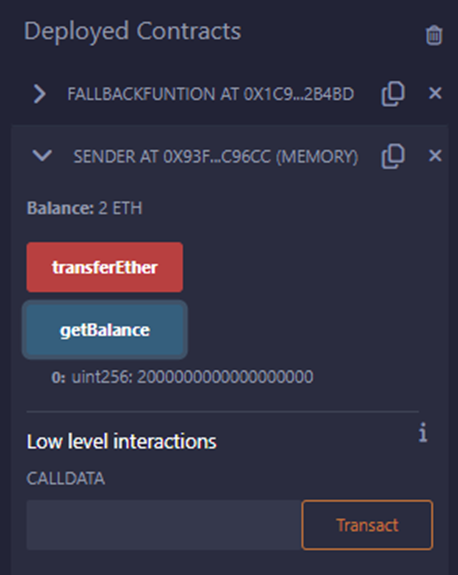
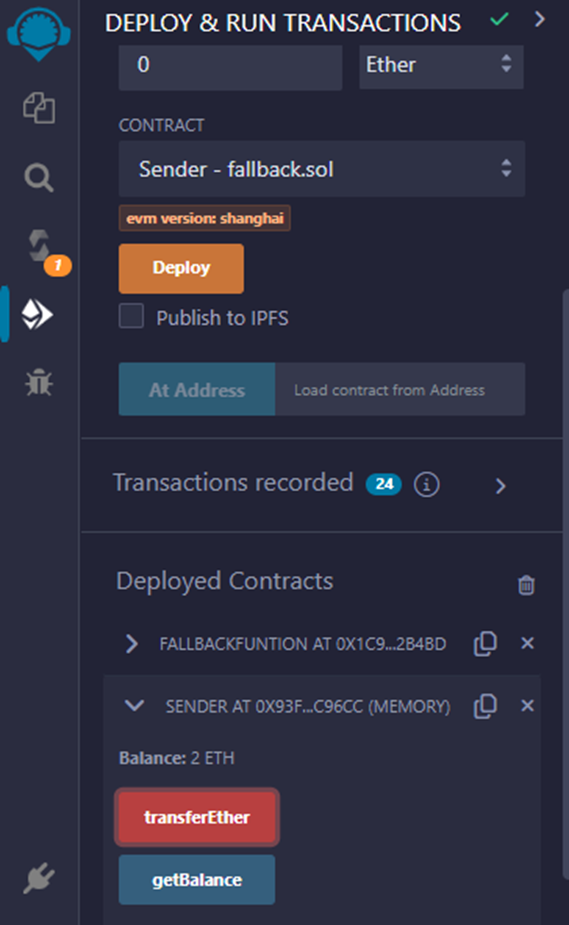
function getBalance() public view returns(uint){

return address(this).balance;

}

}

**Output:**

****

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate solidity code to implement Cryptographic Function and Single Inheritance [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**vii)Cryptographic Functions:**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

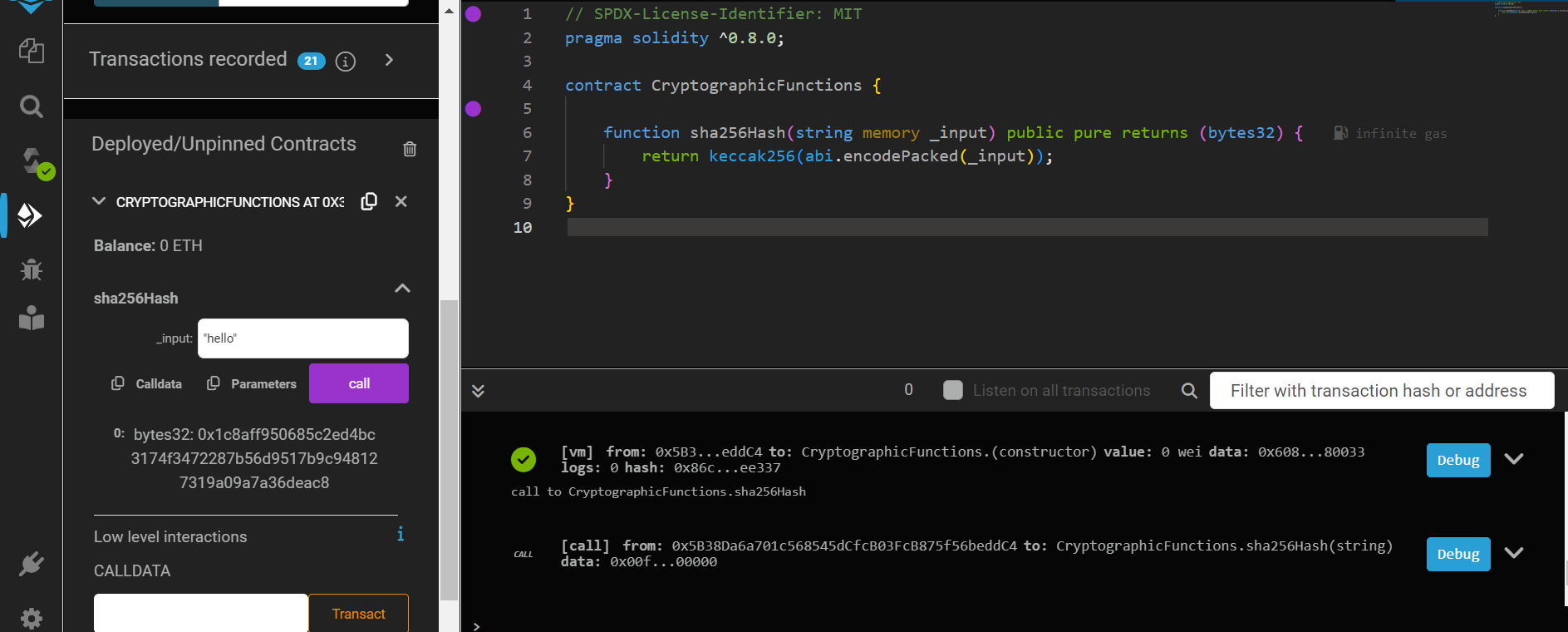
contract CryptographicFunctions {

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

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

}

**Output:**

****

1) 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();

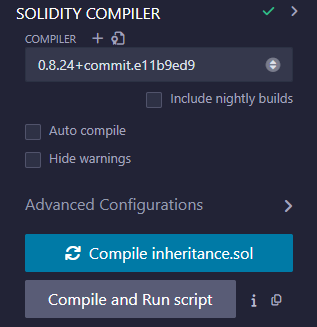
return cc.getVal();

}

}

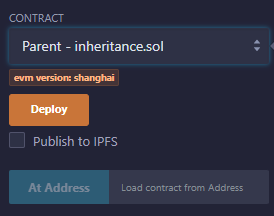
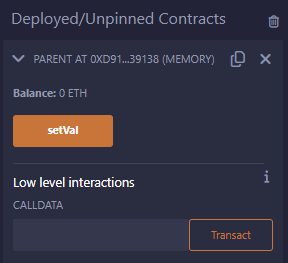
**OUTPUT:**

Compile



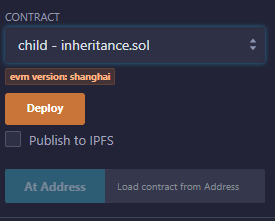
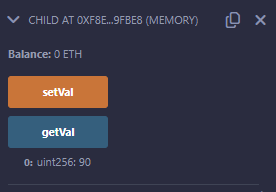
Run:

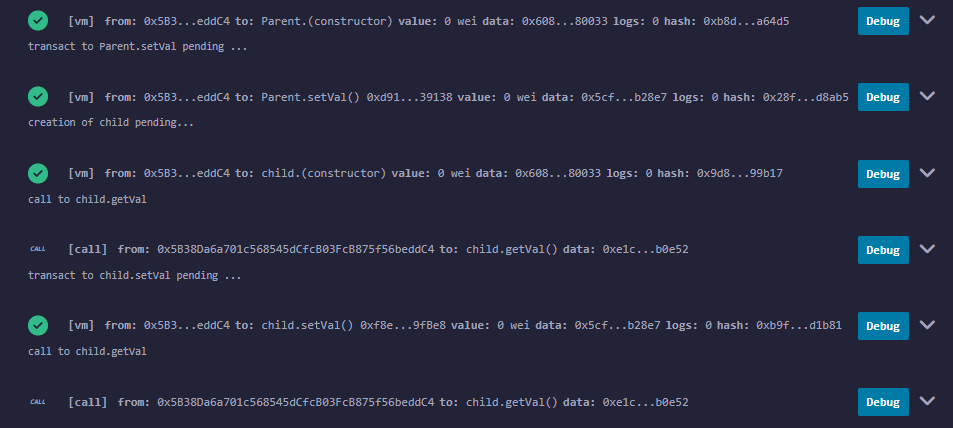
First DEPLOY the parent contract

Deploy the child contract

Select setVal>getVal



|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate solidity code to implement Enum and Special Variables [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

// 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!";

}

}

|  |  |  |
| --- | --- | --- |
|  | **M.Sc. (Information Technology) Part-II SEM-IV**  **Practical Examination**  **Subject: BlockChain**  **Student Seat No: \_\_\_\_\_\_\_\_\_\_\_\_ Total Marks 50** |  |
| Q.1. Demonstrate solidity code to implement Function Overloading and Mathematical Function [40 Marks] | | |
| Q.2 Journal [5 Marks] | | |
| Q.3 Viva [5 Marks] | | |

**vi)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";

}

}

**v)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);

}

}