I**NDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No** | **Topic** |  | **Date** | **Remark** |
| **1** | **Write the following programs for Blockchain in Python :**   1. A Simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it. 2. A transaction class to send and receive money and   test it. |  |  |  |
| **2** | **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. |  |  |  |
| **3** | **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. |  |  |  |
| **4** | **Implement and demonstrate the user of the following in Solidity :**   1. Variable 2. Operations 3. Loops 4. Decision Making   v. Strings |  |  |  |
| **5** | **Implement and demonstrate the user of the following in Solidity :**   1. Arrays 2. Enums 3. Structs 4. Mappings 5. Conversations 6. Ether Units 7. Special Variables |  |  |  |
| **6** | **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 |  |  |  |
| **7** | **Implement and demonstrate the user of the following in Solidity:**   1. Contracts 2. Inheritance 3. Constructors 4. Abstract Class 5. Interfaces |  |  |  |
| **8** | **Implement and demonstrate the user of the following in Solidity:**   1. Libraries 2. Assembly 3. Events 4. Error Handling |  |  |  |

**PRACTICAL 1**

1. **Create a simple client class that generates the private and public keys by using the built-in python RSA algorithm and test it**

**Code:**

#import random

from Crypto.PublicKey import RSA

from Crypto import Random

import binascii

from Crypto.Cipher import PKCS1\_v1\_5

class Client:

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

random=Random.new().read

self.\_private\_key=RSA.generate(1024,random) #1024->key size

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')

Rifath=Client()

print('Rifath,3--> \n',Rifath.identity)

**Output:**

****

**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 SHA

import datetime

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) #1024->key size

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="Genesis"

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=SHA.new(str(self.to\_dict).encode('utf-8'))

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

def display\_tran(transaction):

dict=transaction.to\_dict()

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

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

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

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

transactions=[]

Rifath=Client()

Sara= Client()

t1=Transaction(

Rifath,

Sara.identity,

15)

t1.sign\_tran()

display\_tran(t1)

**PRACTICAL 2**

1. **Create multiple transactions and display them**

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

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

random = Random.new().read

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

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

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

@property

def identity(self):

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

class Transaction:

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

self.sender = sender

self.recipent = recipent

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender': identity,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(self):

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

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

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

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

def display\_transaction(transaction):

# for transaction in transactions:

dict = transaction.to\_dict()

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

print('-----')

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

print('-----')

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

print('-----')

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

print('-----')

transactions = []

Rifath = Client()

Armeen = Client()

Sara = Client()

t1 = Transaction(

Rifath,

Armeen.identity,

15.0

)

t1.sign\_tran()

transactions.append(t1)

t2 = Transaction(

Armeen,

Sara.identity,

17.0

)

t2.sign\_tran()

transactions.append(t2)

t3 = Transaction(

Sara,

Armeen.identity,

10.0

)

t3.sign\_tran()

transactions.append(t3)

tn = 1

for t in transactions:

print("Transaction: ", tn)

display\_transaction(t)

tn = tn + 1

print('-------------------')

**Output:**

****

**Create a block chain a Genesis block and execute it**.

*Noance: a randomly generated number (unique) used once in cryptography transaction*

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

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

random = Random.new().read

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

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

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

@property

def identity(self):

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

class Transaction:

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

self.sender = sender

self.recipent = recipent

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender': identity,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(self):

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

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

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

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

def display\_transaction(transaction):

# for transaction in transactions:

dict = transaction.to\_dict()

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

print('-----')

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

print('-----')

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

print('-----')

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

print('-----')

def dump\_blockchain(self):

print("Number of blocks in the chain:" + str(len(self)))

for x in range (len(TPCoins)):

block\_temp=TPCoins[x]

print("block#" + str(x))

for transaction in block\_temp.verified\_transaction:

display\_transaction(transaction)

print("...............")

print("====================")

class Block:

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

self.verified\_transaction=[]

self.previous\_block\_hash=""

self.Nonce=""

Rifath = Client()

t0=Transaction(

"Genesis",

Rifath.identity,

500.0

)

block0=Block()

block0.previous\_block\_hash=None

Nonce=None

block0.verified\_transaction.append(t0)

digest=hash(block0)

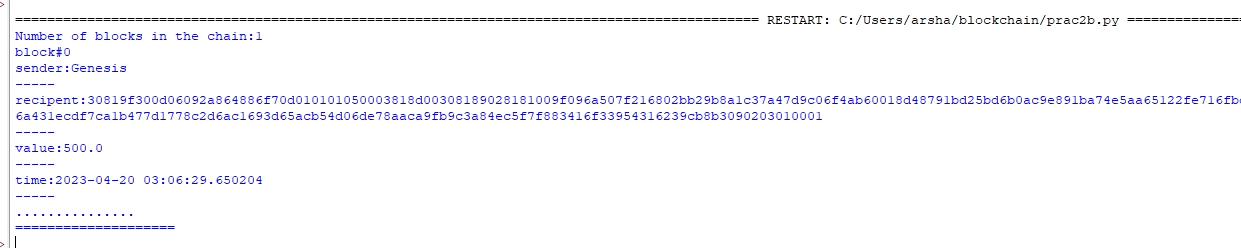
last\_block\_hash = digest

TPCoins=[]

TPCoins.append(block0)

dump\_blockchain(TPCoins)

**Output:**

****

**PRACTICAL 3**

1. **Create a mining function and test it.**

*Miners: verifies the transactions in block chain*

**Code:**

print("Rifath,3")

import hashlib

def sha256(message):

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

def mine(message,difficulty=1):

assert difficulty>=1 #debugging

prefix= '1'\* difficulty #verify diffficulty

print ("prefix",prefix)

for i in range(1000):

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

print("Testing --> " + digest)

if digest.startswith(prefix):

print("After" + str(i) + "iterations found nounce" + digest)

return i

mine("Rifath", 3)

**Output:**

****

1. **Add block to miner and dump the block chain.**

*Miners: verifies the transactions in block chain*

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

import hashlib

print("Rifath,27")

class Client:

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

random = Random.new().read

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

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

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

@property

def identity(self):

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

class Transaction:

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

self.sender = sender

self.recipent = recipent

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender': identity,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(self):

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

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

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

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

def display\_transaction(transaction):

# for transaction in transactions:

dict = transaction.to\_dict()

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

print('-----')

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

print('-----')

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

print('-----')

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

print('-----')

def dump\_blockchain(self):

print("Number of blocks in the chain:" + str(len(self)))

for x in range (len(TPCoins)):

block\_temp=TPCoins[x]

print("block#" + str(x))

for transaction in block\_temp.verified\_transaction:

display\_transaction(transaction)

print("...............")

print("====================")

class Block:

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

self.verified\_transaction=[]

self.previous\_block\_hash=""

self.Nonce=""

def sha256(message):

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

def mine(message,difficulty=1):

assert difficulty>=1 #debugging

prefix= '1'\* difficulty #verify diffficulty

print ("prefix",prefix)

for i in range(1000):

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

print("Testing --> " + digest)

if digest.startswith(prefix):

print("After " + str(i) + "iterations found nounce " + digest)

return i

mine("Rifath", 3)

transactions = []

Rifath = Client()

Armeen = Client()

Sara = Client()

t0=Transaction(

"Genesis",

Rifath.identity,

500.0

)

t1 = Transaction(

Rifath,

Armeen.identity,

15.0

)

t1.sign\_tran()

transactions.append(t1)

t2 = Transaction(

Armeen,

Sara.identity,

17.0

)

t2.sign\_tran()

transactions.append(t2)

t3 = Transaction(

Sara,

Armeen.identity,

10.0

)

#blockchain

TPCoins=[]

block0=Block()

block0.previous\_block\_hash=None

Nonce=None

block0.verified\_transaction.append(t0)

digest=hash(block0)

last\_block\_hash = digest

last\_block\_hash=digest

TPCoins.append(block0)

block1=Block()

block1.previous\_block\_hash=last\_block\_hash

block1.verified\_transaction.append(t1)

block1.verified\_transaction.append(t2)

block1.Nonce=mine(block1,2)

digest=hash(block1)

last\_block\_hash=digest

TPCoins.append(block1)

block2=Block()

block2.previous\_block\_hash=last\_block\_hash

block2.verified\_transaction.append(t3)

Nonce=mine(block2,2)

block2.Nonce=mine(block2,2)

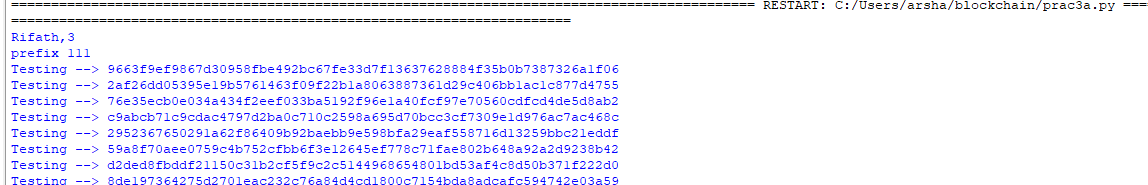
digest=hash(block2)

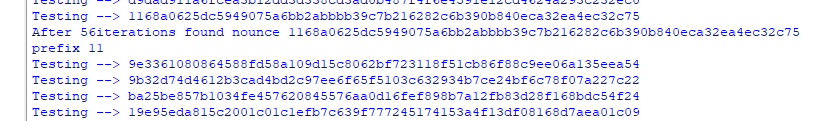
last\_block\_hash=digest

TPCoins.append(block2)

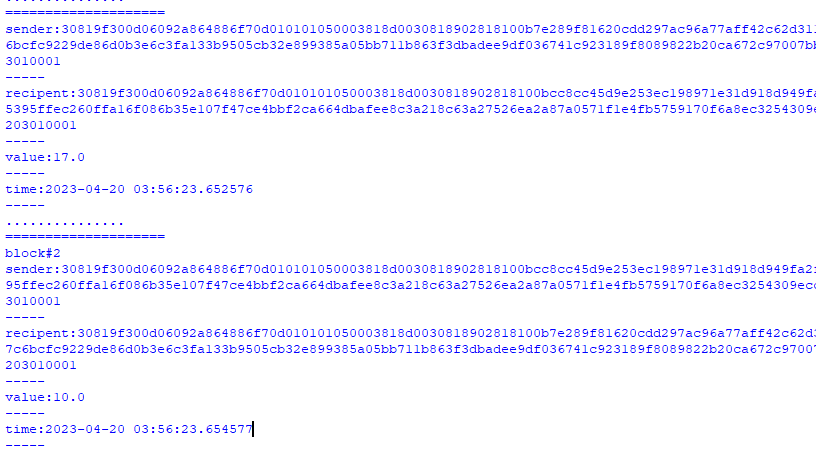
dump\_blockchain(TPCoins)

**Output:**

****





****

**PRACTICAL 4**

1. **Variable**

**Code:**

pragma solidity ^0.8.0;

//RIFATH 3

contract SolidityTest {

    uint storedData; // State variable

    constructor() public{

        storedData=10;

    }

    function getDiv() public view returns(uint){

        uint a=10; // local variable

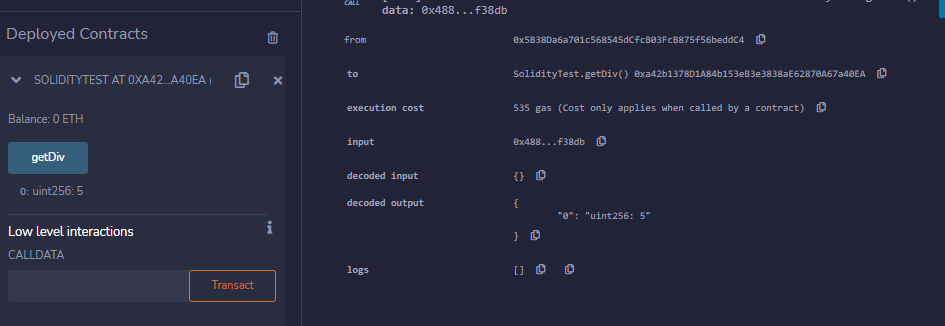
        uint b=2;

        uint result = a / b;

    return result; // accesss the state variable

}}

**Output:**

****

1. **Operations**

**Code:**

pragma solidity ^0.8.0;

//RIFATH 3

contract SolidityTest {

    uint storedData; // State variable

    constructor() public{

        storedData=10;

    }

    function getDiv() public view returns(uint){

        uint a=50; // local variable

        uint b=5;

        uint result = a / b;

    return result; // accesss the state variable

    }

    function getMul() public view returns(uint){

        uint a=50; // local variable

        uint b=5;

        uint result = a \* b;

    return result; // accesss the state variable

    }

    function getSum() public view returns(uint){

        uint a=50; // local variable

        uint b=5;

        uint result = a + b;

    return result; // accesss the state variable

    }

    function getSub() public view returns(uint){

        uint a=50; // local variable

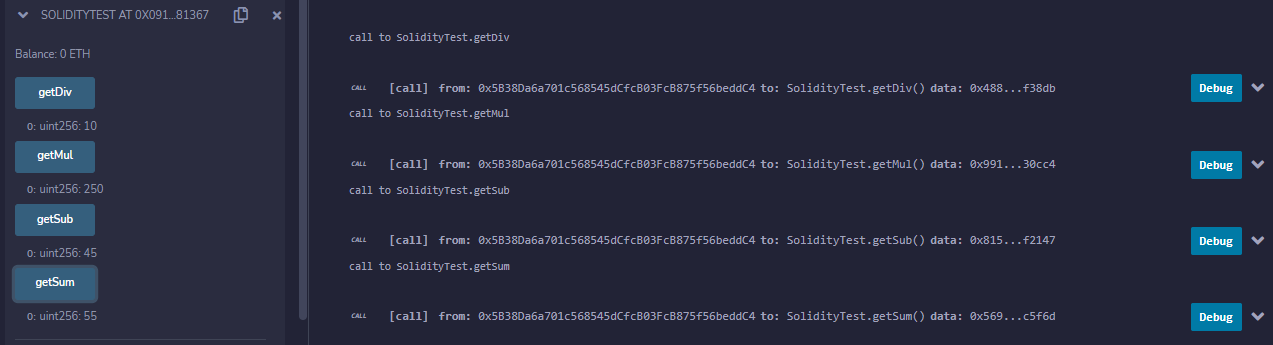
        uint b=5;

        uint result = a - b;

    return result; // accesss the state variable

    }}

**Output:**

****

1. **Loops**
   1. **While**

**Code:**

pragma solidity ^0.8.0;

//rifath 3

contract while1{

uint[] data;

uint8 j=0;

function loop() public returns(uint[] memory)

{

    while (j<10)

    {

        j++;

        data.push(j);

    }

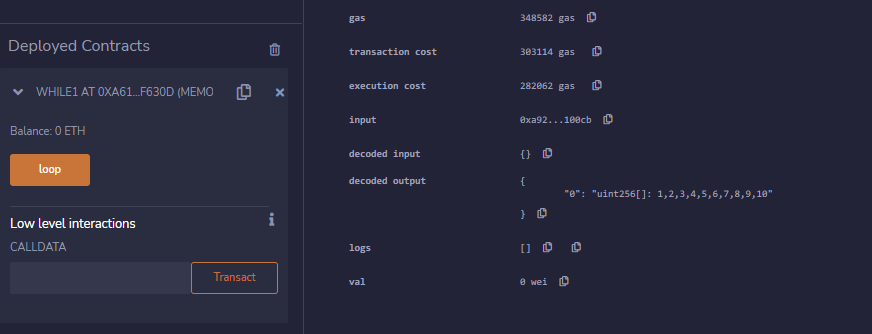
    return data;

}

}

}

**Output:**

****

**Do While**

**Code:**

pragma solidity ^0.8.0;

//rifath 3

contract doWhile1{

uint[] data;

uint8 j=0;

function loop() public returns(uint[] memory)

{

    do

    {

        j++;

        data.push(j);

    }

    while (j<10);

    return data;

}

}

* 1. **For**

**Code:**

pragma solidity ^0.8.0;

contract ForLoop{

    function count() public pure returns(uint256){

        uint256 sum=0;

        for(uint256 i=0;i<=25;i++){

            sum+=i;

        }

        return sum;

   }}

**Output:**

1. **Decision Making**
   1. **If else**

**Code:**

pragma solidity ^0.8.0;

contract Check{

    uint i=100;

    uint j=80;

    function ifElse() public returns(string memory)

    {

        if(i<j)

        {

            return "i is smaller than j";

        }

        else

        {

            return " i is greater than j";

        }   }}

**Output:**

* 1. **If else-if**

Code:

pragma solidity ^0.8.0;

contract Check{

    uint i=100;

    uint j=100;

    function ifElseIf() public returns(string memory)

    {

        if(i<j)

        {

            return "i is smaller than j";}

        else if(i>j)

        {

            return " i is greater than j"; }

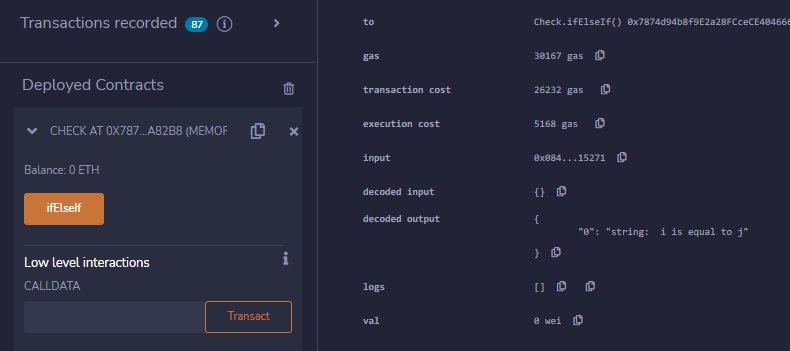
        else

        {

            return " i is equal to j";

       }}}

**Output:**

****

1. **Strings**
   1. **Regular string**

**Code:**

pragma solidity ^0.8.0;

contract SS{

    string str1="M.SC I.T PART 2";

    string str2='K.C COLLEGE, COLABA';

    string str3=new string(20);

    function getstr1() public returns(string memory)

    {

        return str1;   }

    function getstr2() public returns(string memory)

    {

        return str2;  }

    function getstr3() public returns(string memory)

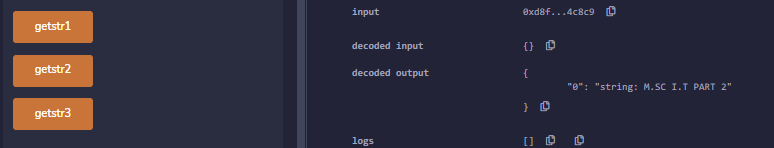
    {

        return str3;

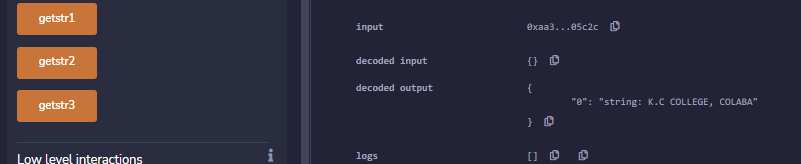
    }}

**Output:**

**String1:**

****

**String2:**

****

**String3:**

* 1. **Concatenate**

**Code:**

pragma solidity >=0.5.0 <0.9.0;

//rifath 3

   contract Demo{

      string public s1 = "RIFATH ";

      string public s2 = "ZAHRAA";

      string public new\_str;

      function concatenate() public {

         new\_str = string(abi.encodePacked(s1, s2));

       }

}

**Output:**

* 1. **Compare**

**Code:**

pragma solidity ^0.8.0;

contract Demo{

    string str1="rifath";

    string str2='rifath”';

    bool public isEqual;

    function cmp() public

    {

        isEqual=keccak256(abi.encodePacked(str1))==keccak256(abi.encodePacked(str2));

    }

}

**Output:**

**PRACTICAL 5**

1. **Arrays**

**Code:**

pragma solidity ^0.5.0;

contract Arrray{

    uint[] nums=[1,2,33,21];

    function getlength() public returns(uint){

        return nums.length;

    }

    function pop() public{

        delete nums[1];

    }

    function push() public returns (uint[] memory){

        nums.push(7);

        return nums;

    }

function push1(uint i) public{

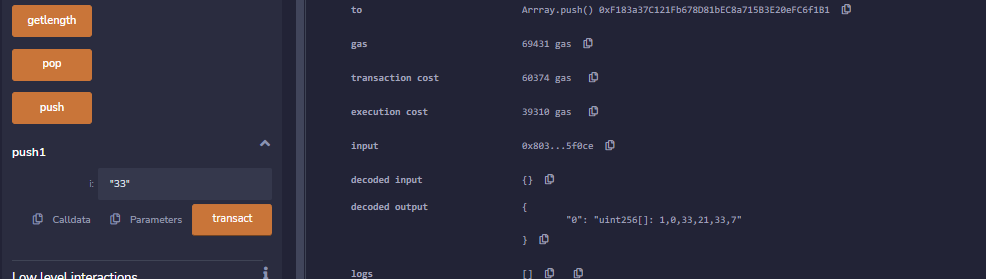
             nums.push(i);

        }

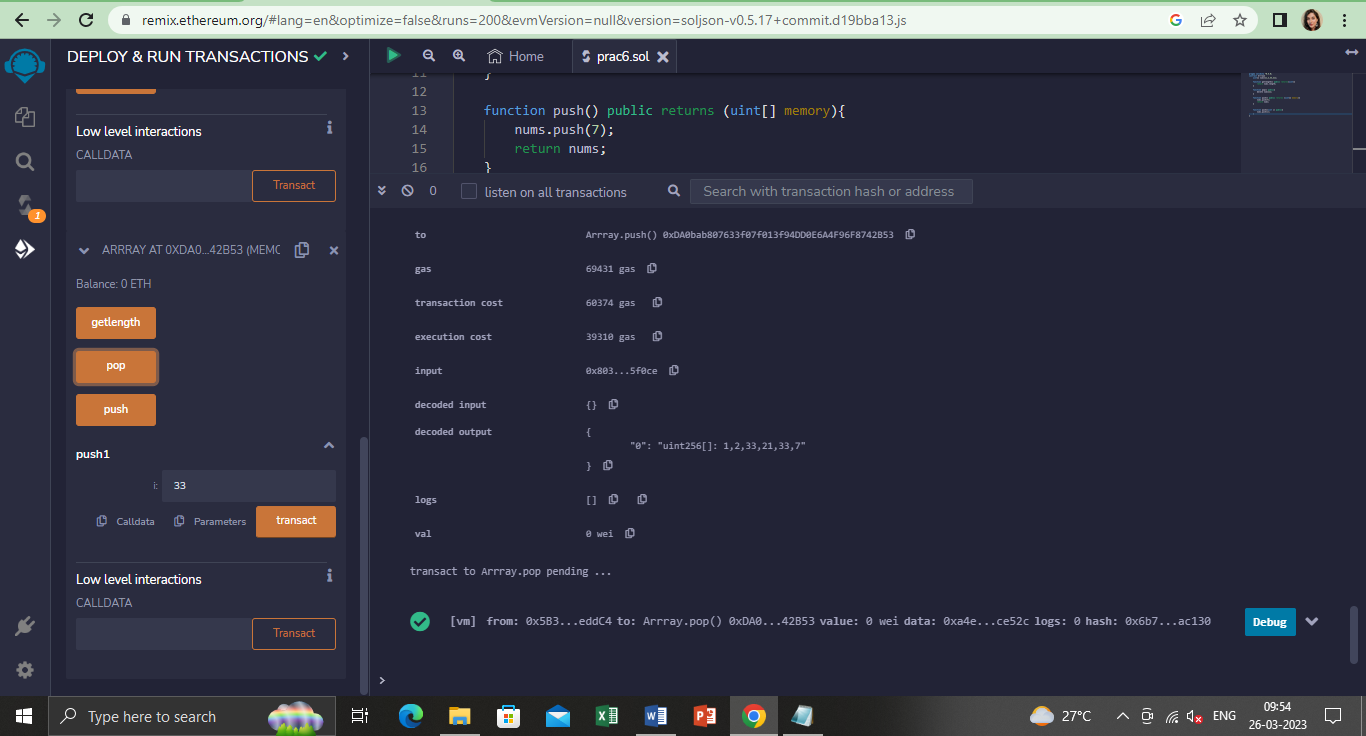
}

**Output:**

**Push:**

****

**Pop:**



**Dynamic:**

****

1. **Struct**

**Code:**

pragma solidity ^0.5.0;

contract test{

    struct Book{

        string title;

        string author;

        string name;

        uint book\_id;

    }

   Book;

   function setBook() public{

       book = Book('SOLIDITY','JOHN','fantasy world',101);

   }

   function getBookId() public view returns(uint){

        return book.book\_id;

   }

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

        return book.name;

   }

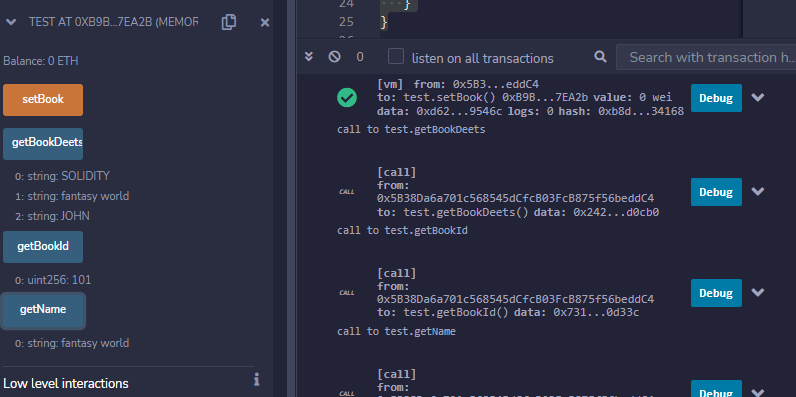
   function getBookDeets() public view returns(string memory,string memory){

       return(book.title,book.name,book.author);

   }

}

**Output:**

****

1. **Enum**

**Code:**

pragma solidity ^0.5.0;

contract Types{

        enum week\_days

        {

            Monday,

            Tuesday,

            Wednesday,

            Thursday,

            Friday,

            Saturday,

            Sunday

        }

        week\_days week;

        week\_days choice;

        week\_days constant default\_value = week\_days.Sunday;

        function set\_value() public{

            choice = week\_days.Thursday;

        }

        function get\_choice() public view returns(week\_days){

            return choice;

        }

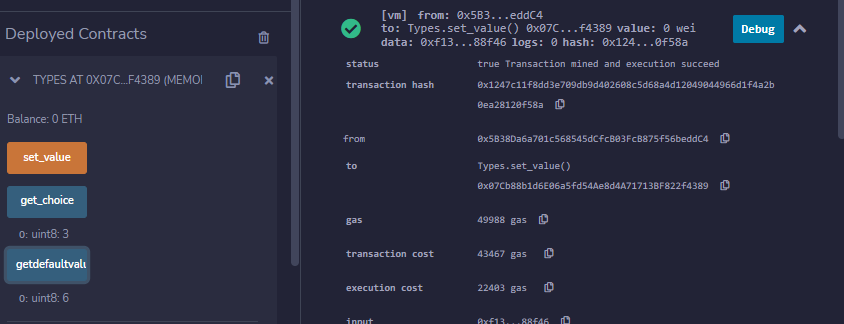
        function getdefaultvalue() public pure returns(week\_days){

            return default\_value;

        }

}

**Output:**

****

1. **Mapping**

**Code:**

pragma solidity ^0.5.0;

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] = "rifath";

        return name[msg.sender];

    }

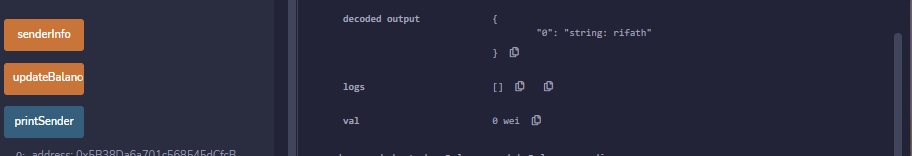
    function printSender() public view returns(address){

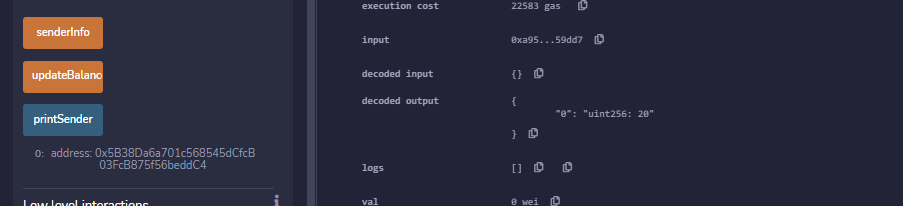
        return msg.sender;

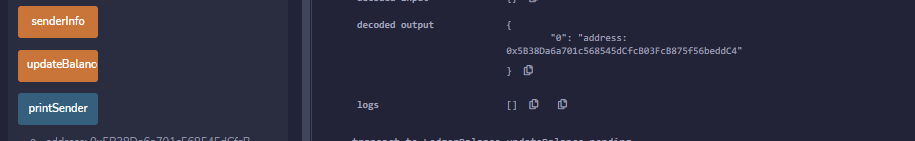
    }

}

**Output:**

****

****

****

**6. Conversions and Ether units**

**keccak256**

**Code**:

pragma solidity >=0.4.0 <0.7.0;

contract EtherUnitsExample {

    uint256 public valueInWei = 1 ether; // 1 ether in Wei

    uint256 public valueInFinney = 1 finney; // 1 finney in Wei

    uint256 public valueInSzabo = 1 szabo; // 1 szabo in Wei

    uint256 public valueInEther = 1 ether; // 1 ether in Wei

    function convert(uint256 \_amount, string memory \_unit) public pure returns (uint256) {

        if (keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("wei"))) {

            return \_amount;

        } else if (keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("finney"))) {

            return \_amount \* 1 finney;

        } else if (keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("szabo"))) {

            return \_amount \* 1 szabo;

        } else if (keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("ether")) || keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("eth"))) {

            return \_amount \* 1 ether;

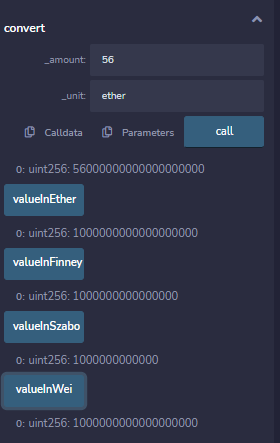
        } else {

            revert("Invalid unit");

        }

    }}

**Output**:



**7. Special variables:**

* 1. **Solidity contract to demonstrate the special variables block.number and blockhash.**

**Code**:

pragma solidity ^0.5.0;

contract prac

{

    uint BNumber;

    bytes32 BHashPresent;

    bytes32 BHashPrevious;

    function PresentHash()

            public returns(bytes32)

    {

        BNumber = block.number;

        BHashPresent =blockhash(BNumber);

        return BHashPresent;

    }

    function PreviousHash()

            public returns(bytes32)

    {

        BNumber = block.number;

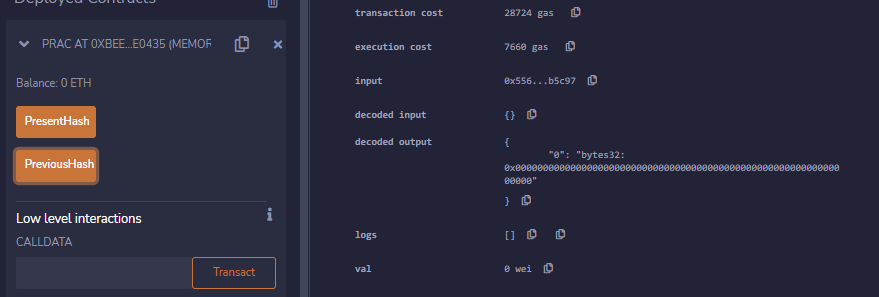
        BHashPrevious = blockhash(BNumber - 1);

        return BHashPrevious;

    }

}

**Output**:



* 1. **Solidity contract to demonstrate msg.sender**

**Code**:

pragma solidity ^0.5.0;

contract LedgerBalance{

    mapping(address => string) name;

    function senderInfo() public returns(string memory){

        name[msg.sender] = "rifath";

        return name[msg.sender];

    }

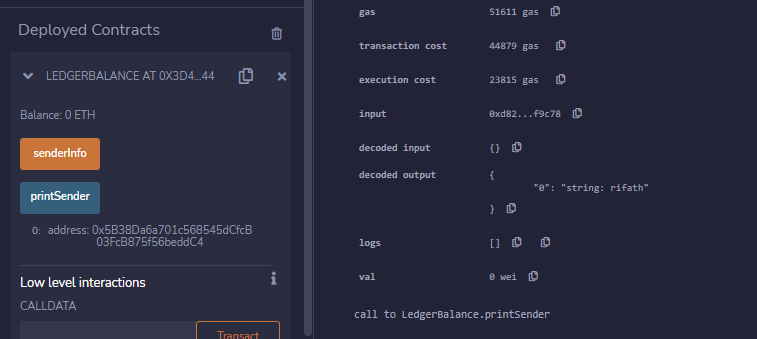
    function printSender() public view returns(address){

        return msg.sender;

    }

}

**Output**:



**PRACTICAL 6**

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

1. **Functions**

**Code:**

pragma solidity ^0.8.0;

contract LedgerBalancee {

    mapping(address => string) name;

    function updateBalance() public returns(string memory) {

        name[msg.sender]="zah";

        return name [msg.sender];

    }

    function printsender() public view returns(address){

        return msg.sender;

    }

}

contract Test {

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

      uint a = 1; // local variable

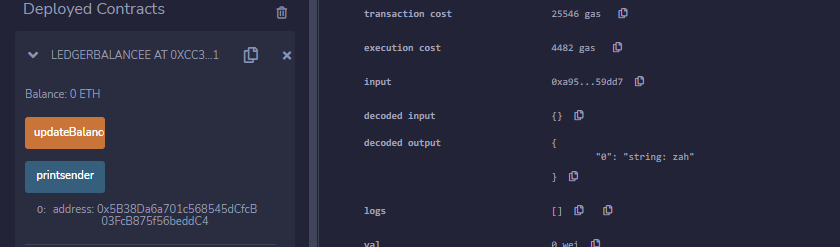
      uint b = 2;

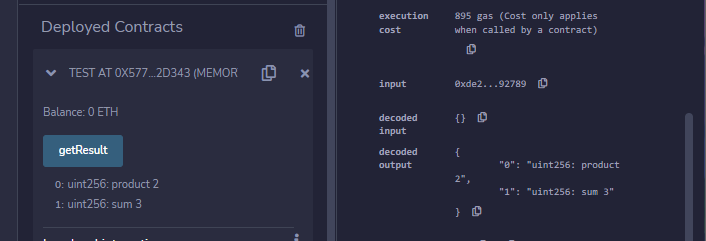
      product = a \* b;

      sum = a + b;

      //return(a\*b, a+b);

   }}

**Output: **

****

1. **.View Functions**

**Code**:

pragma solidity ^0.5.0;

contract Test{

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

        uint a=1;// local variable

        uint b=2;

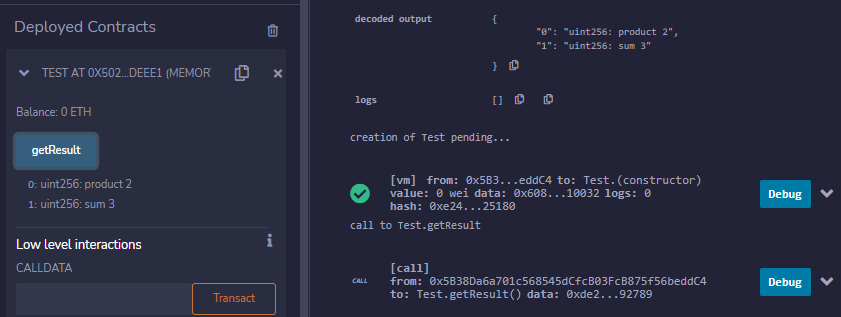
        product=a\*b;

        sum = a+b;

    }

}

**Output**:



1. **.Pure Functions**

**Code**:

pragma solidity ^0.5.0;

contract Test{

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

        uint a = 1; //local variable

        uint b = 2;

        product = a\*b;

        sum = a+b;

    }

}

**Output**:

1. **.Fallback Functions**

**Code**:

pragma solidity ^0.5.12;

contract A {

    uint n;

    function set(uint value) external {

        n=value;

    }

    //fallback function

    function() external payable{

        n=0;

    }

}

contract example{

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

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

        require(success);

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

        return(payableA.send(2 ether));

    }

}

**Output**:

1. **.Function Overloading**

**Code**:

pragma solidity ^0.5.12;

contract Sample{

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

        return a+b;

    }

    function getSum(uint a, uint b, uint c) public pure returns (uint){

        return a+b+c;

    }

    function callSumWithTwoArguments() public pure returns (uint){

        return getSum(4,9);

    }

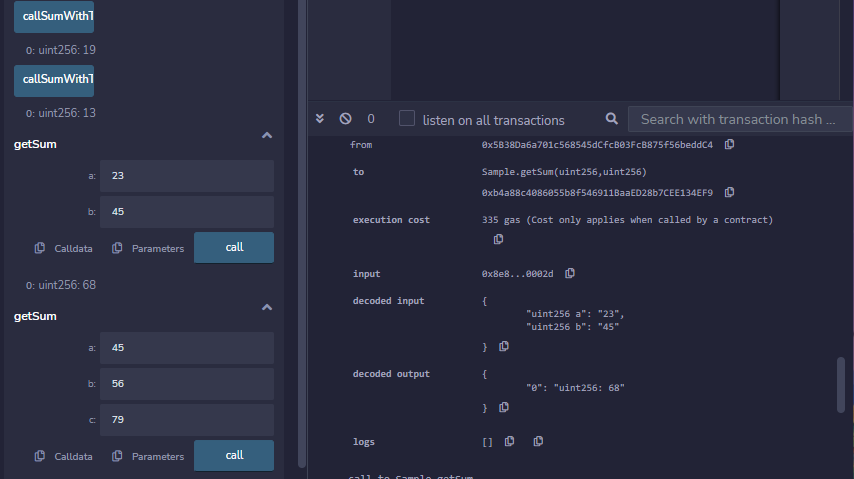
     function callSumWithThreeArguments() public pure returns (uint){

        return getSum(4,9,6);

    }

}

**Output**:



1. **.Mathematical Functions**

**Code**:

pragma solidity ^0.5.0;

contract Sample{

    function callAddMod() public pure returns (uint){

        return addmod(3,4,5);

//3+4 % 5

    }

   function callMulMod() public pure returns (uint){

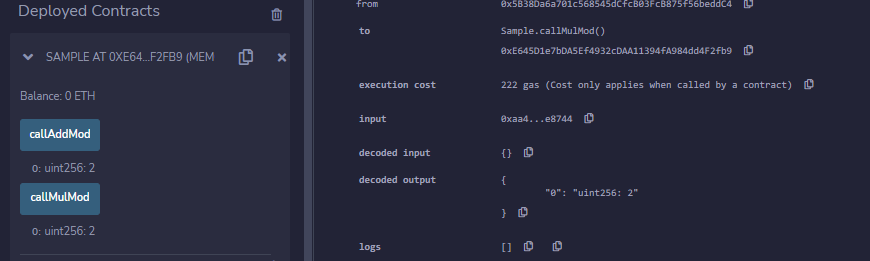
        return mulmod(3,4,5);

    }

//3\*4 % 5

}

**Output**:



1. **.Cryptographic Functions**

**Code**:

pragma solidity ^0.5.12;

contract Test{

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

        return sha256("rifath");

    }

    function callkeccak256() public pure returns(bytes32 result){

        return keccak256("rifath");

    }

}

**Output**:

**PRACTICAL 7**

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

1. **Contracts**
2. **Inheritance**
3. **Constructors**
4. **Abstract class**
5. **Interfaces**
6. **Contracts**

**Code:**

pragma solidity ^0.8.0;

contract Storage

{

uint public setData;

function set(uint x) public{

setData = x;

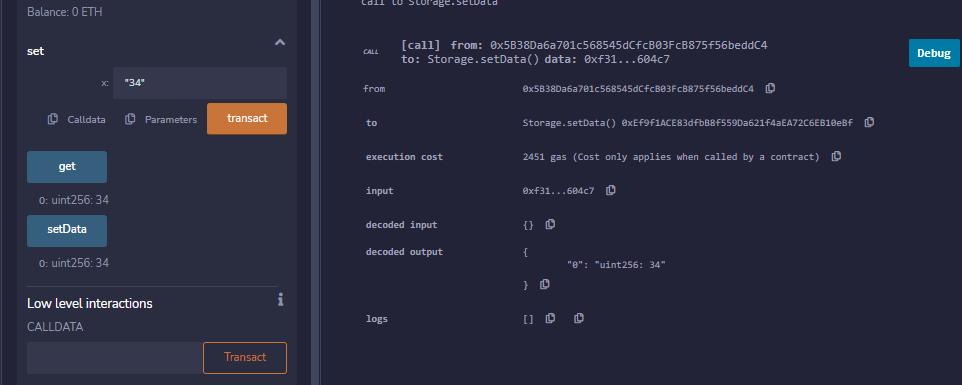
}

function get() public view returns (uint) {

    return setData;

}}

**Output:**

****

1. **Inheritance**
2. **Single Inheritance:**

**Code:**

pragma solidity 0.5.0;

contract parent{

    uint internal sum;

    function setValue() external {

        uint a = 10;

        uint b = 25;

        sum = a + b;

    }

}

contract child is parent{ //defining the child contract

    function getValue(

    ) external view returns(uint) {

        return sum;

    }

}

contract caller {

    child cc = new child();

    function testInheritance(

    ) public returns (uint) {

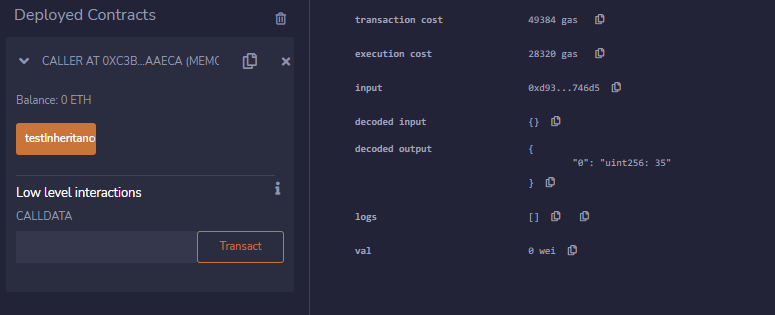
        cc.setValue();

        return cc.getValue();

    }

}

**Output:**

****

1. **Multiple Inheritance:**

**Code:**

pragma solidity ^0.5.0;

contract A {

    string internal x;

    function setA() external {

        x = "Multiple Inheritance";

    }

}

contract B {

    uint256 internal pow;

    function setB() external {

        uint256 a = 2;

        uint256 b = 20;

        pow = a\*\*b;

    }

}

contract C is A, B {

    function getStr() external view returns (string memory)

    {

        return x;

    }

    function getPow() external view returns (uint256)

    {

        return pow;

    }

}

contract caller {

    C contractC = new C();

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

        contractC.setA();

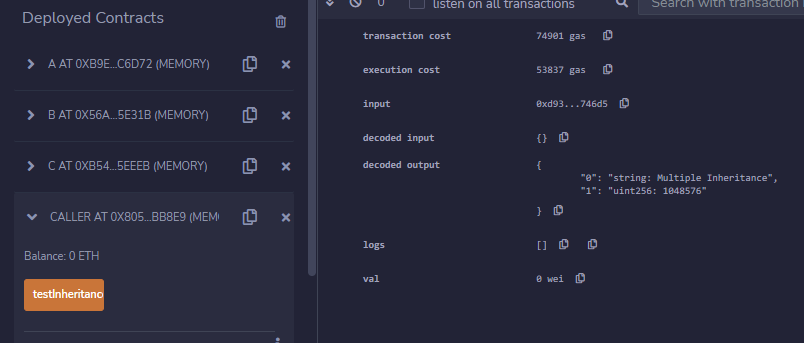
        contractC.setB();

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

    }

}

**Output:**



1. **Multilevel Inheritance:**

**Code:**

pragma solidity ^0.5.0;

contract A {

      uint256 internal x;

    function setX() external {

        x=600;

    }

}

contract B is A {

      uint256 internal y;

    function setY() external {

        y=20-x;

    }

}

contract C is B{

    function getY() external view returns(

      uint){

        return y;

    }

}

contract caller {

    C cc = new C();

    function testInheritance(

    ) public returns (

      uint256) {

        cc.setX();

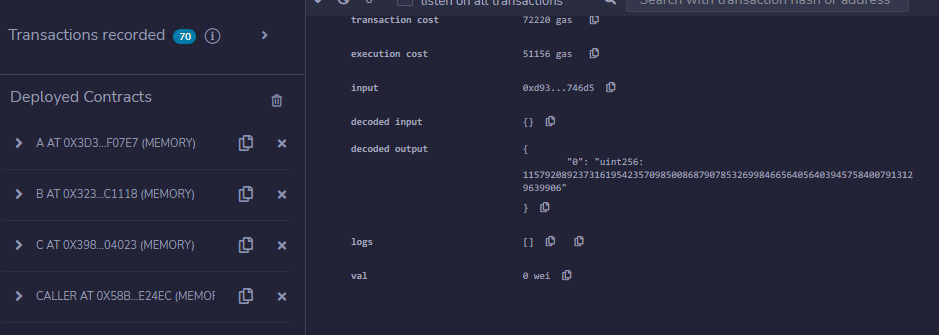
        cc.setY();

        return cc.getY();

    }

}

**Output:**

****

1. **Constructors**

**Code:**

pragma solidity ^0.5.0;

// Creating a contract

contract constructorExample {

// Declaring state variable

string str;

constructor() public {

str = "RIFATH 3, MSC IT P2";

}

// Defining function to

// return the value of 'str'

function getValue(

) public view returns (

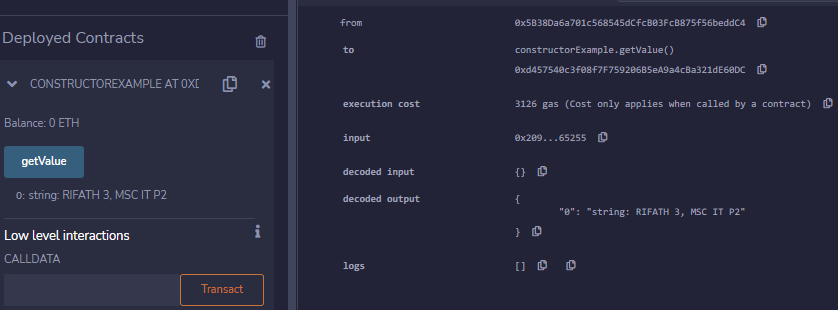
string memory) {

return str;

}

}

**Output:**

****

1. **Abstract class**

**Code:**

pragma solidity ^0.5.0;

contract A {

   function getResult() public view returns(uint);

}

contract B is A {

   function getResult() public view returns(uint) {

      uint a = 100;

      uint b = 201;

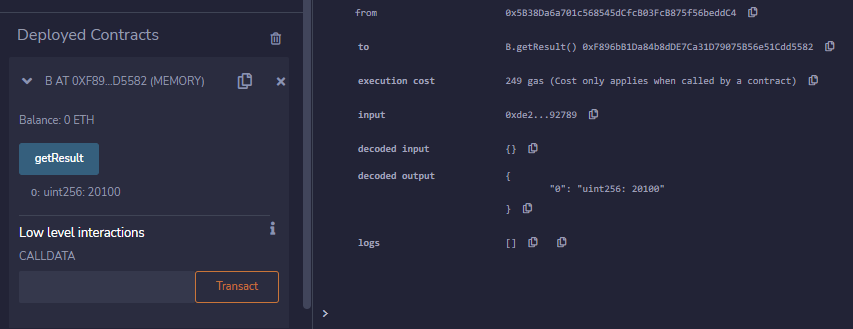
      uint result = a \* b;

      return result;

   }

}

**Output:**

****

1. **Interfaces**

**Code:**

pragma solidity ^0.5.0;

interface Calculator {

   function getResult() external view returns(uint);

}

contract Test is Calculator {

   constructor() public {}

   function getResult() external view returns(uint){

      uint a = 1;

      uint b = 2;

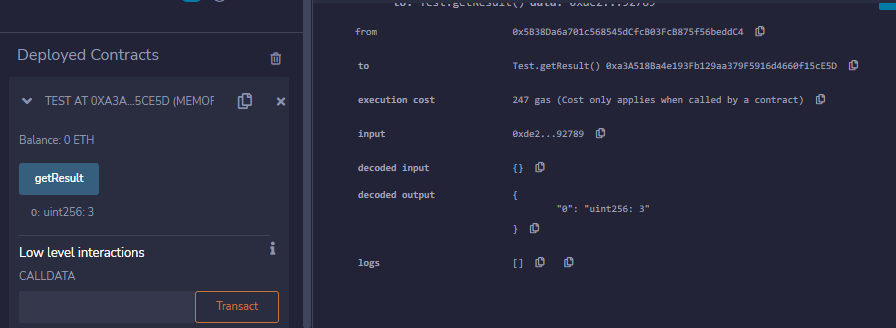
      uint result = a + b;

      return result;

   }

}

**Output:**

****

**PRACTICAL 8**

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

1. **Libraries**

**Code:**

**Libraries.sol:**

pragma solidity ^0.8.0;

import "./MathUtils.sol";

contract calculator{

    using MathUtils for uint;

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

        return a.add(b);

    }

}

**MathUtils.sol:**

pragma solidity ^0.8.0;

library MathUtils{

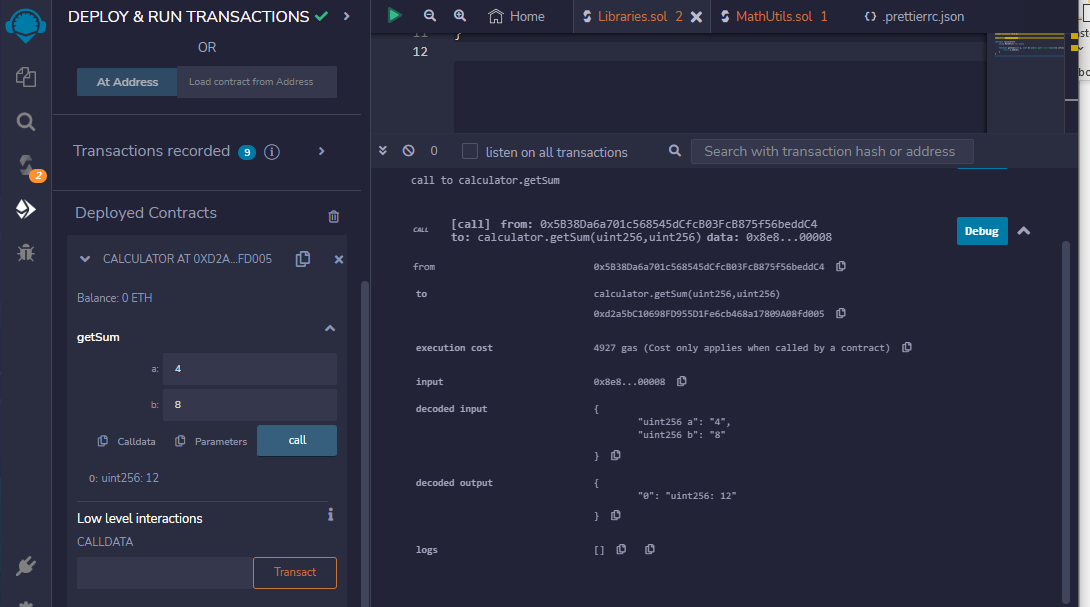
    function add(uint x, uint y) public pure returns(uint){

        return x+y;

    }

}

**Output:**



1. **Assembly**

**Code:**

pragma solidity ^0.5.0;

contract InlineAssembly {

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

        assembly {

            let c := add(a, 56)

            mstore(0x80, c)

            {

                let d := add(sload(c), 22)

                b := d

            }

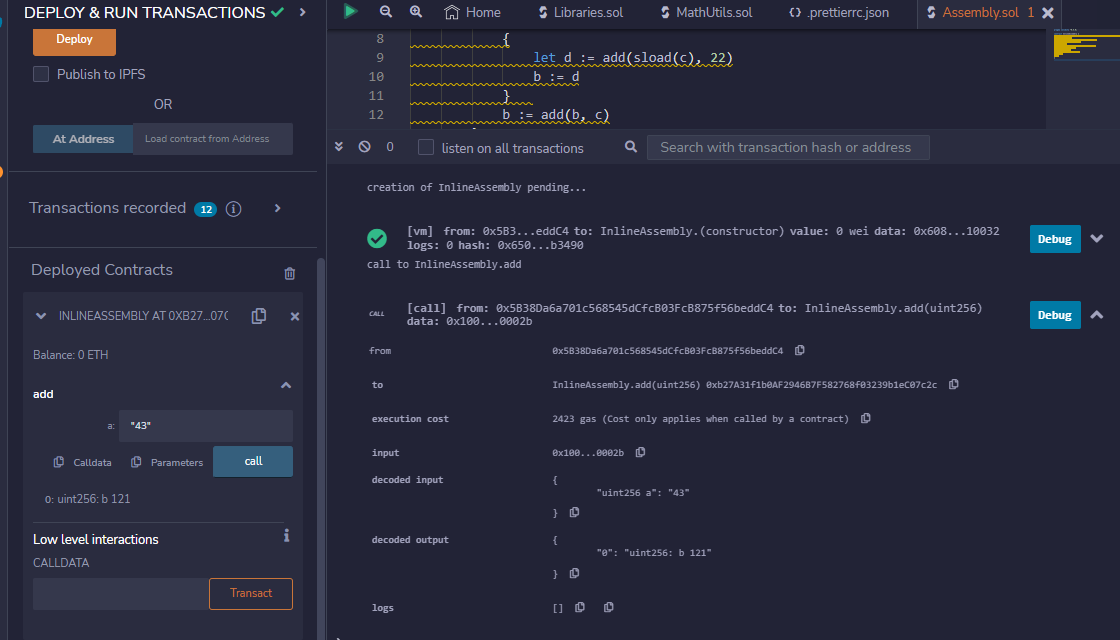
            b := add(b, c)

        }

    }

}

**Output:**

****

1. **Events**

**Code:**

pragma solidity ^0.4.21;

contract eventExample {

    uint256 public value = 0;

    event Increment(address owner);

    function getValue(uint \_a, uint \_b) public { // \_a, \_b is instance variable (used internally only)

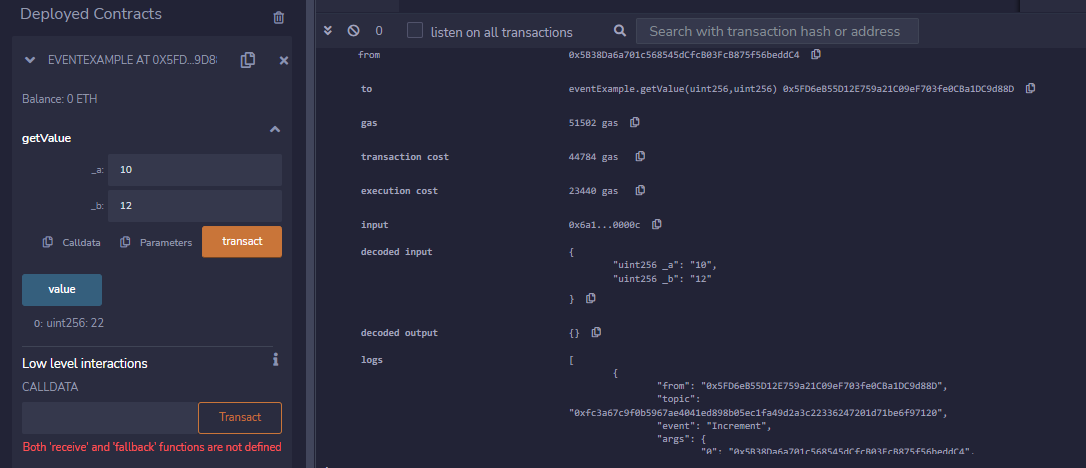
        emit Increment(msg.sender);

        value = \_a + \_b;

    }

}

**Output:**

****

****

1. **Error Handling**
   1. **Require**
   2. **Assert**
   3. **Revert**
2. **Require:**

**Code:**

pragma solidity ^0.5.0;

//RIFATH 3

contract requireStatement {

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

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

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

        return "Input is Uint8";

    }

    function Odd(uint \_input) public view returns(bool){

        require(\_input % 2 != 0);

        return true;

    }

}

**Output:**

1. **Assert:**

**Code:**

pragma solidity ^0.5.0;

//RIFATH 3

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";

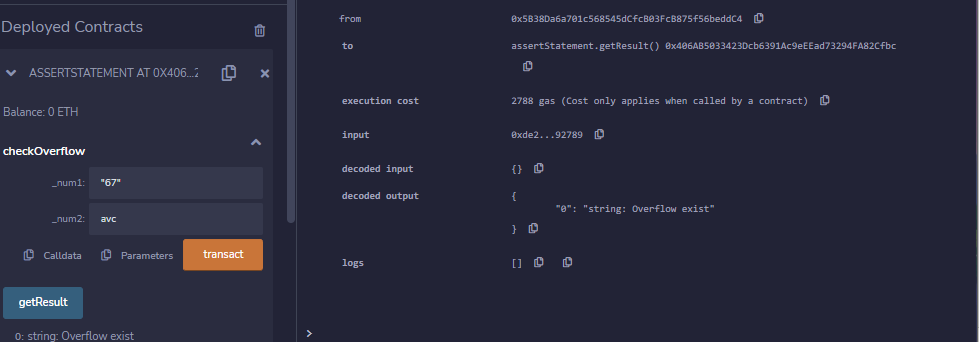
        }

    }

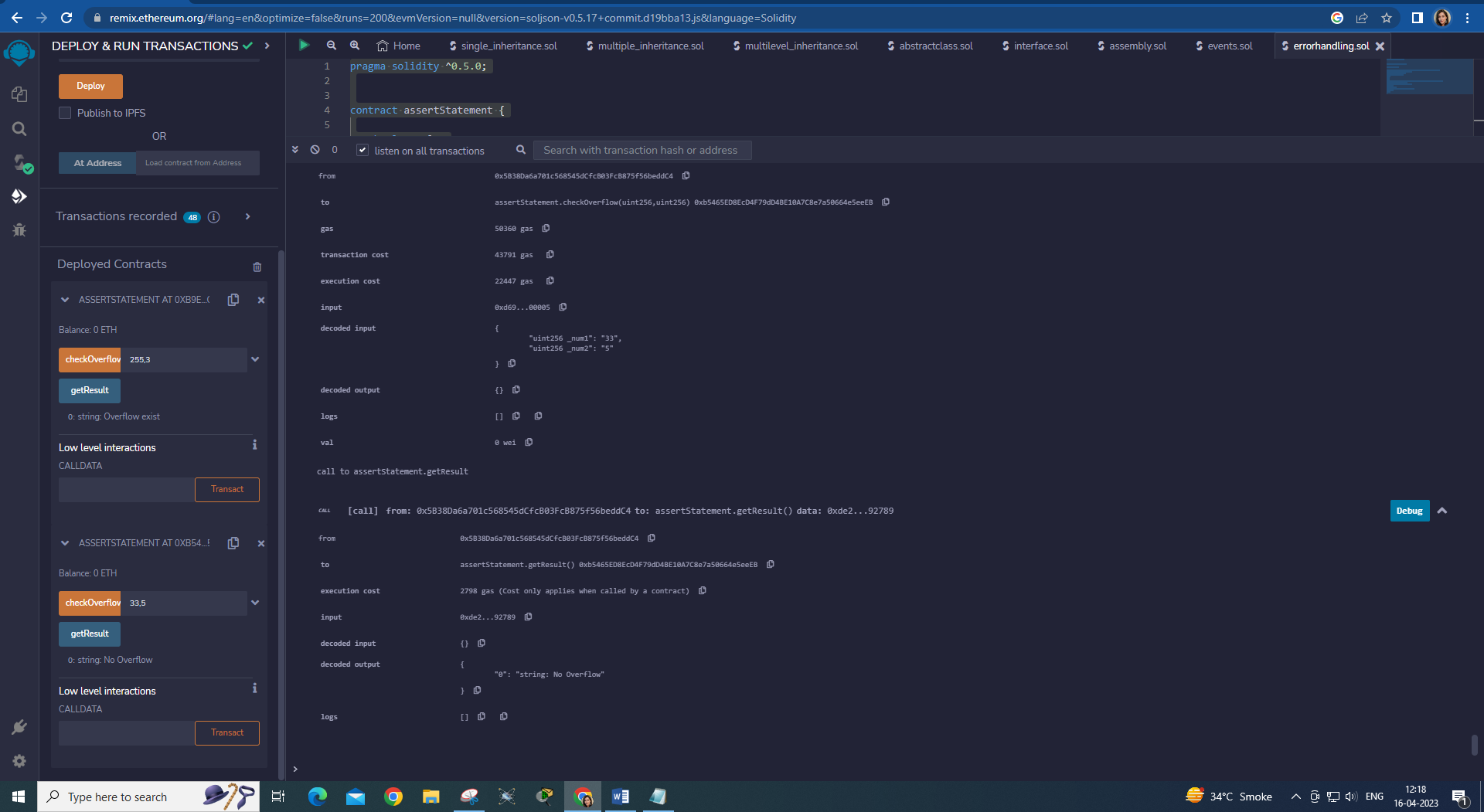
}

**Output:**

**Overflow exists:**



**No Overflow:**



1. **Revert:**

**Code:**

pragma solidity ^0.5.0;

//Rifath 3

contract revertStatement {

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

{

        uint sum = \_num1 + \_num2;

        if(sum < 0 || sum > 255){

            revert(" Overflow Exist");

        }

        else{

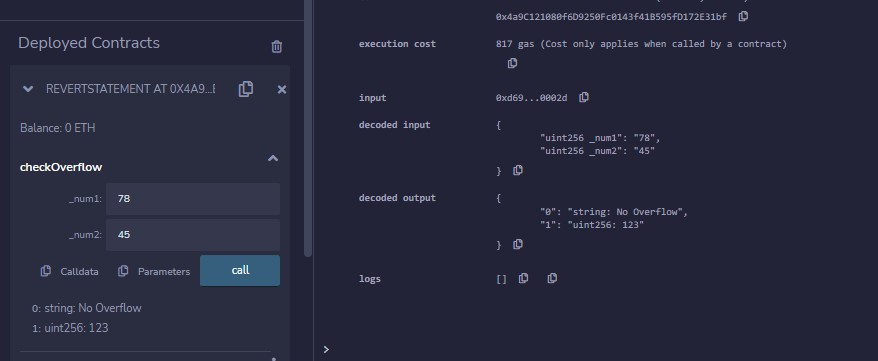
            return ("No Overflow", sum);

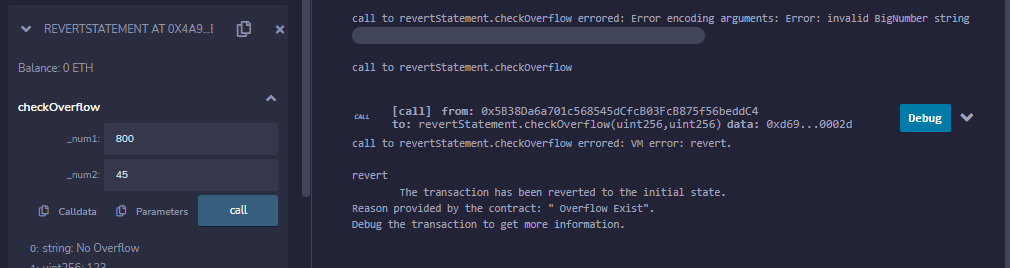
        }

    }

}

**Output:**

****

****