**Practical 1 Write the following programs for Blockchain in Python:**

Aim : (I) A simple client class that generates the private and public keys by using the built in Python RSA algorithm and tests it .

**Input:**

import hashlib import random import string import json import binascii

import numpy as np import pandas as pd import pylab as pl import logging import datetime import collections

pip install pycryptodome import Crypto

import Crypto.Random

from Crypto.Hash import SHA from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

import binascii class Client:

def init (self):

random = Crypto.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)

**Output:**

****

**Aim: (II) A transaction class to send and receive money and test it . Input:**

import datetime import collections import binascii

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5 from Crypto.Hash import SHA

class Client:

def init (self):

# Generate a public/private key pair key = RSA.generate(2048) self.\_private\_key = key self.\_public\_key = key.publickey()

def identity(self):

# Return the public key in hexadecimal format

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

class Transaction:

def init (self, sender, recipient, value): self.sender = sender

self.recipient = recipient self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

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

else:

identity = self.sender.identity()

return collections.OrderedDict({ 'sender': identity,

'recipient': self.recipient, 'value': self.value, 'time': self.time

})

def sign\_transaction(self):

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

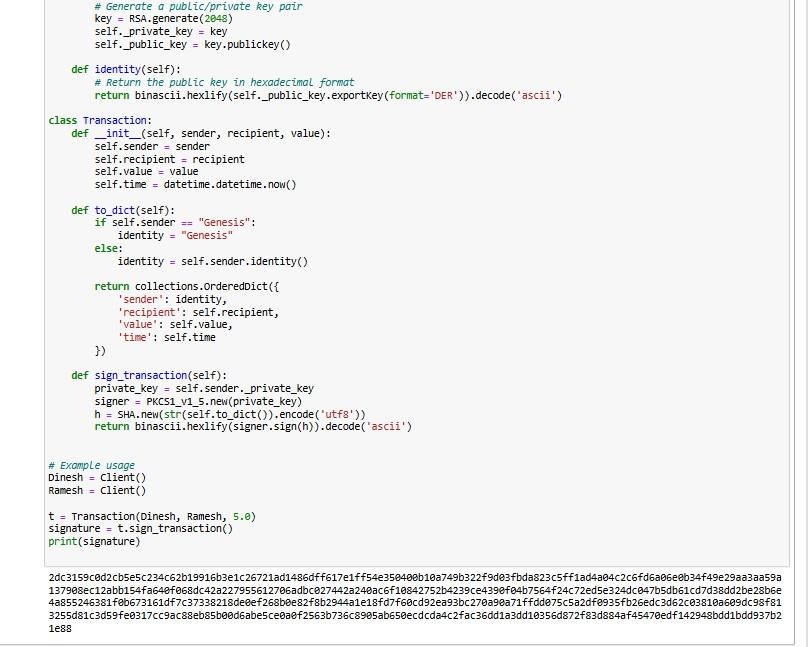
h = SHA.new(str(self.to\_dict()).encode('utf8')) return binascii.hexlify(signer.sign(h)).decode('ascii')

# Example usage Dinesh = Client() Ramesh = Client()

t = Transaction(Dinesh, Ramesh, 5.0) signature = t.sign\_transaction() print(signature)

output:

-



**Practical 2**

**Write the following programs for Blockchain in Python : Aim:(I).Create multiple transactions and display them**.

# Function to display the transaction details def display\_transaction(transaction):

transaction\_dict = transaction.to\_dict() print(f"Sender: {transaction\_dict['sender']}") print(f"Recipient: {transaction\_dict['recipient']}") print(f"Value: {transaction\_dict['value']}") print(f"Time: {transaction\_dict['time']}") print(f"Signature: {transaction.sign\_transaction()}") print(" ")

# Main usage example transactions = []

Dinesh = Client() Ramesh = Client() Seema = Client() Vijay = Client()

# Creating and signing transactions

t1 = Transaction(Dinesh, Ramesh.identity(), 15.0) t1.sign\_transaction()

transactions.append(t1)

t2 = Transaction(Dinesh, Seema.identity(), 6.0) t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction(Ramesh, Vijay.identity(), 2.0) t3.sign\_transaction()

transactions.append(t3)

t4 = Transaction(Seema, Ramesh.identity(), 4.0) t4.sign\_transaction()

transactions.append(t4)

t5 = Transaction(Vijay, Seema.identity(), 7.0) t5.sign\_transaction()

transactions.append(t5)

t6 = Transaction(Ramesh, Seema.identity(), 3.0) t6.sign\_transaction()

transactions.append(t6)

t7 = Transaction(Seema, Dinesh.identity(), 8.0) t7.sign\_transaction()

transactions.append(t7)

t8 = Transaction(Seema, Ramesh.identity(), 1.0) t8.sign\_transaction()

transactions.append(t8)

t9 = Transaction(Vijay, Dinesh.identity(), 5.0) t9.sign\_transaction()

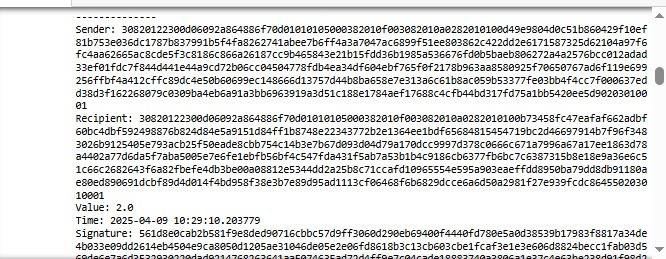
transactions.append(t9)

t10 = Transaction(Vijay, Ramesh.identity(), 3.0) t10.sign\_transaction()

transactions.append(t10)

# Display all transactions

for transaction in transactions: display\_transaction(transaction) print(' ')



**Aim:(II). Create a blockchain, a genesis block and execute it .**

# Block and blockchain definitions TPCoins = []

# Genesis block transaction

t0 = Transaction("Genesis", Dinesh.identity(), 500.0) block0 = Block()

# Genesis block previous block hash is "0" block0.previous\_block\_hash = "0" block0.verified\_transactions.append(t0)

# Calculate hash for the first block (Genesis block)

block0.Nonce = "0000" # Example Nonce (can be set to an actual mined value later) digest = block0.calculate\_hash()

last\_block\_hash = digest

# Add the first block to the blockchain TPCoins.append(block0)

# Function to dump the blockchain def dump\_blockchain(chain):

print(f"Number of blocks in the chain: {len(chain)}") for x in range(len(chain)):

block\_temp = chain[x] print(f"Block # {x}")

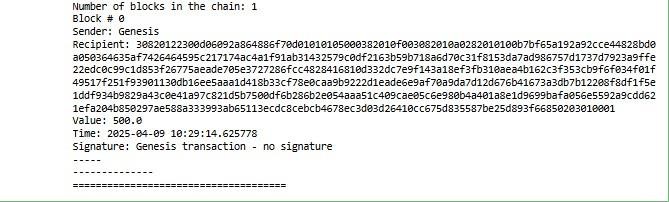
for transaction in block\_temp.verified\_transactions: display\_transaction(transaction)

print(' ')

print('=====================================')

# Dump the blockchain dump\_blockchain(TPCoins)

output:



**Practical 3**

**Write the following programs for Blockchain in Python : Aim: (I). Create a mining function and test it .**

**(II). Add blocks to the miner and dump the blockchain .**

import hashlib import datetime

# Define the Transaction class class Transaction:

def init (self, sender, recipient, value): self.sender = sender

self.recipient = recipient self.value = value

self.time = datetime.datetime.now()

def to\_dict(self): return {

'sender': self.sender, 'recipient': self.recipient, 'value': self.value, 'time': self.time

}

# Define the Block class class Block:

def init (self): self.verified\_transactions = [] self.previous\_block\_hash = "" self.Nonce = ""

self.timestamp = datetime.datetime.now() self.block\_hash = ""

def calculate\_hash(self):

# Concatenate the block's data (transactions, previous hash, nonce, timestamp) and hash

it

block\_data = str(self.verified\_transactions) + str(self.previous\_block\_hash) +

str(self.Nonce) + str(self.timestamp)

return hashlib.sha256(block\_data.encode('utf-8')).hexdigest()

# Define the mining function def sha256(message):

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

def mine(message, difficulty=1): assert difficulty >= 1

prefix = '0' \* difficulty # '0' prefix to simulate proof of work for i in range(1000): # Limit attempts to avoid infinite loop

digest = sha256(str(message) + str(i)) if digest.startswith(prefix):

print(f"After {i} iterations, found nonce: {digest}") return digest # Return the hash when found

return None # Return None if no valid hash is found

# Initialize variables for the blockchain and transactions last\_transaction\_index = 0

last\_block\_hash = "0" # Genesis block has no previous hash TPCoins = [] # Blockchain (list of blocks)

transactions = [

Transaction("Genesis", "Dinesh", 500),

Transaction("Dinesh", "Ramesh", 15),

Transaction("Dinesh", "Seema", 6),

Transaction("Ramesh", "Vijay", 2),

Transaction("Seema", "Ramesh", 4)

]

# Miner 1 adds a block block = Block()

while last\_transaction\_index < len(transactions): # Ensure not going out of range temp\_transaction = transactions[last\_transaction\_index] block.verified\_transactions.append(temp\_transaction)

last\_transaction\_index += 1

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine("Block 1", 2) # Mine the block

block.block\_hash = block.calculate\_hash() # Calculate the block's hash after mining

TPCoins.append(block) # Add the mined block to the blockchain last\_block\_hash = block.block\_hash # Update the last block's hash

# Miner 2 adds a block block = Block()

while last\_transaction\_index < len(transactions): # Ensure not going out of range temp\_transaction = transactions[last\_transaction\_index] block.verified\_transactions.append(temp\_transaction)

last\_transaction\_index += 1

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine("Block 2", 2) # Mine the block

block.block\_hash = block.calculate\_hash() # Calculate the block's hash after mining

TPCoins.append(block) # Add the mined block to the blockchain last\_block\_hash = block.block\_hash # Update the last block's hash

# Miner 3 adds a block block = Block()

while last\_transaction\_index < len(transactions): # Ensure not going out of range temp\_transaction = transactions[last\_transaction\_index]

block.verified\_transactions.append(temp\_transaction) last\_transaction\_index += 1

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine("Block 3", 2) # Mine the block

block.block\_hash = block.calculate\_hash() # Calculate the block's hash after mining

TPCoins.append(block) # Add the mined block to the blockchain last\_block\_hash = block.block\_hash # Update the last block's hash

# Function to display the blockchain def dump\_blockchain():

print(f"Number of blocks in the chain: {len(TPCoins)}") for idx, block in enumerate(TPCoins):

print(f"Block #{idx}")

print(f"Block Hash: {block.block\_hash}") for transaction in block.verified\_transactions:

print(transaction.to\_dict()) print("=====================================")

# Display the entire blockchain dump\_blockchain()

output:



**Practical 4**

**Aim: Implement and demonstrate the use of the following in Solidity : (I)Variable**

1. **Operators**
2. **Decision Making**
3. **Strings (V)Loops**
4. **Variable**

pragma solidity ^0.5.0; contract SolidityTest {

uint storedData; // State variable constructor() public { storedData = 10;

}

function getResult() public view returns(uint){ uint a = 1; // local variable

uint b = 2;

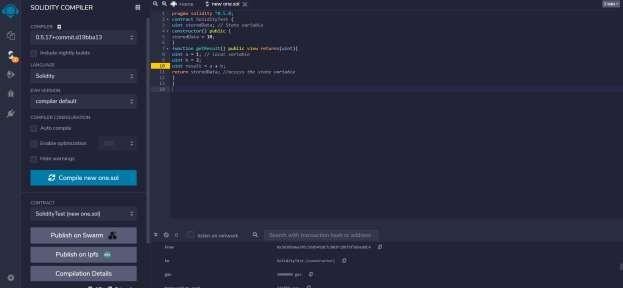
uint result = a + b;

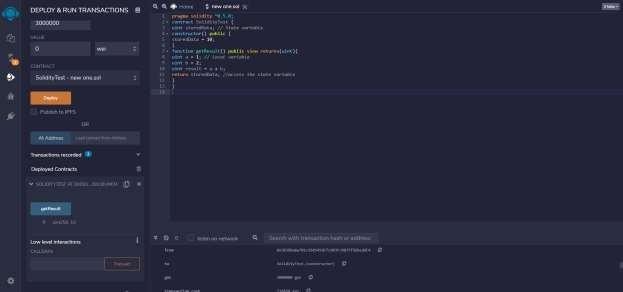
return storedData; //access the state variable

}

}

**Output:**

****

****

// **Solidity program to demonstrate state variables**

pragma solidity ^0.5.0;

// Creating a contract contract Solidity\_var\_Test {

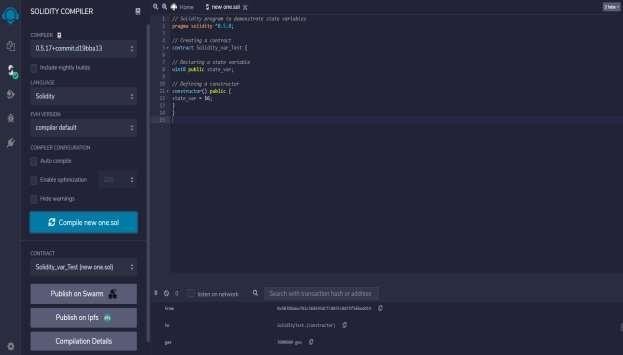
// Declaring a state variable uint8 public state\_var;

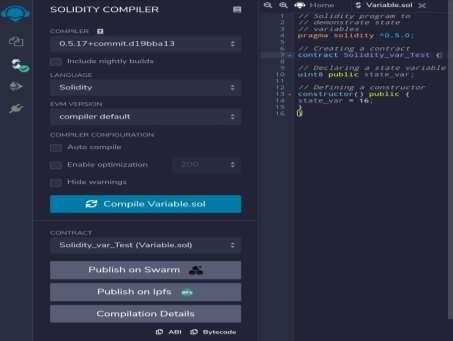
// Defining a constructor constructor() public { state\_var = 16;

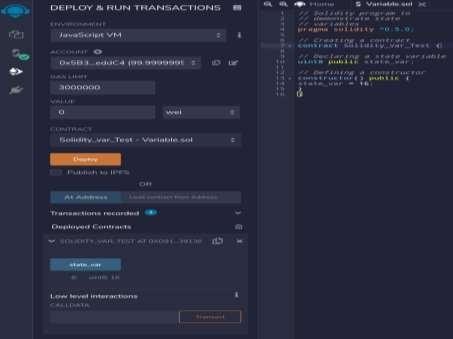
}

}

**Output:**

****



****

// **Solidity program to show Global variables**

pragma solidity ^0.5.0;

// Creating a contract contract Test {

// Defining a variable address public admin;

// Creating a constructor to

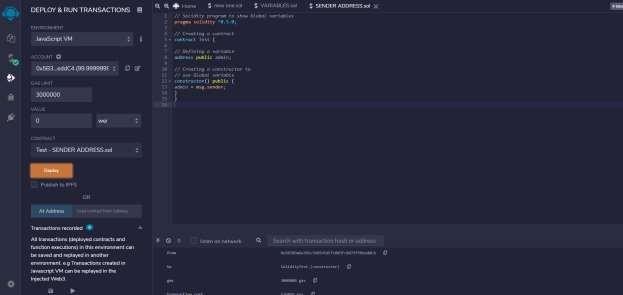
// use Global variable constructor() public { admin = msg.sender;

}

}

**Output:**

****



1. **Operators**

// Solidity contract to demonstrate Arithematic Operator pragma solidity ^0.5.0;

// Creating a contract contract SolidityTest {

// Initializing variables uint16 public a = 20; uint16 public b = 10;

// Initializing a variable with sum uint public sum = a + b;

// Initializing a variable with the difference uint public diff = a - b;

// Initializing a variable with product uint public mul = a \* b;

// Initializing a variable with quotient uint public div = a / b;

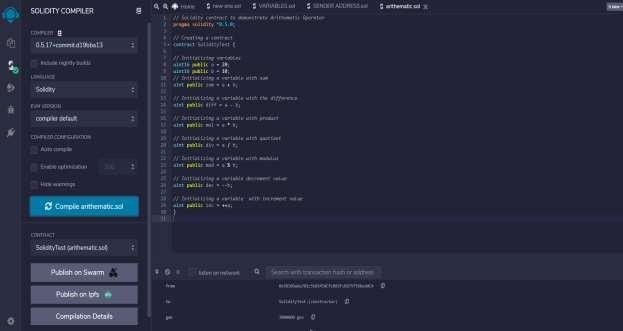
// Initializing a variable with modulus uint public mod = a % b;

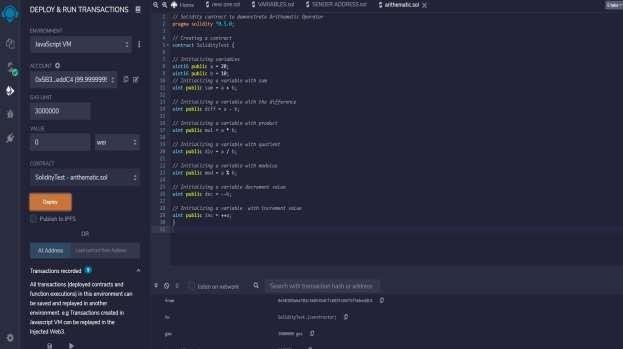
// Initializing a variable decrement value uint public dec = --b;

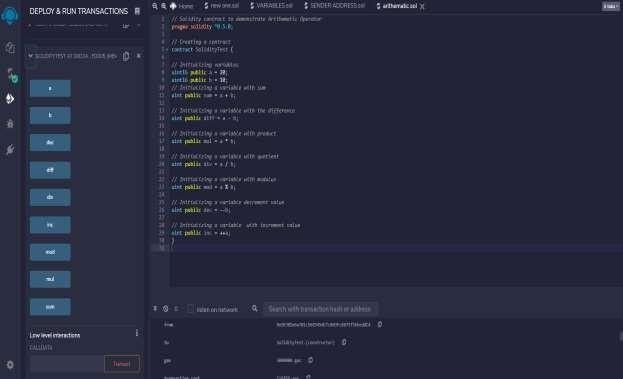
// Initializing a variable with increment value uint public inc = ++a;

}

**Output:**

****

****



1. **Decision Making**

// **Solidity program to demonstrate the use of 'if statement'**

pragma solidity ^0.5.0;

// Creating a contract contract Types {

// Declaring state variable uint i = 10;

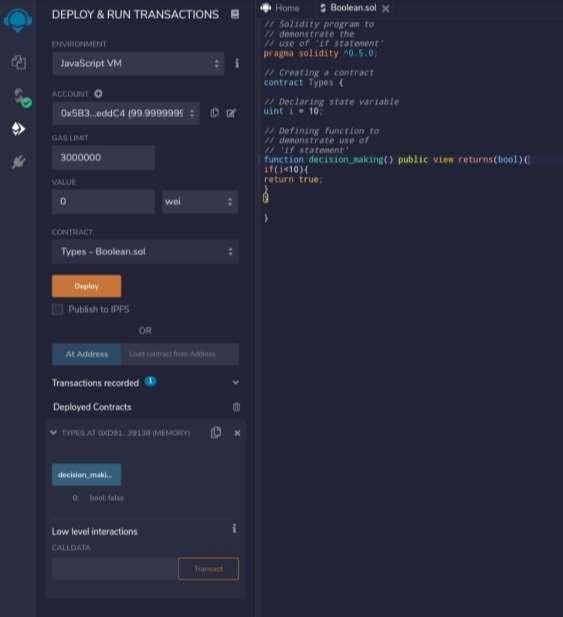
// Defining function to demonstrate use of 'if statement' function decision\_making() public view returns(bool){ if(i<10){

return true;

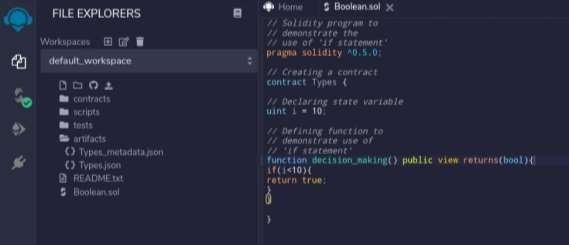
}

}

}



**Output:**

****

// **Solidity program to demonstrate the use of 'if...else' statement**

pragma solidity ^0.5.0;

// Creating a contract contract Types {

// Declaring state variables uint i = 10;

bool even;

// Defining function to

// demonstrate the use of

// 'if...else statement'

function decision\_making() public { if(i%2 == 0){

even = true;

}

else{

even = false;

}

}

function getresult() public view returns(bool)

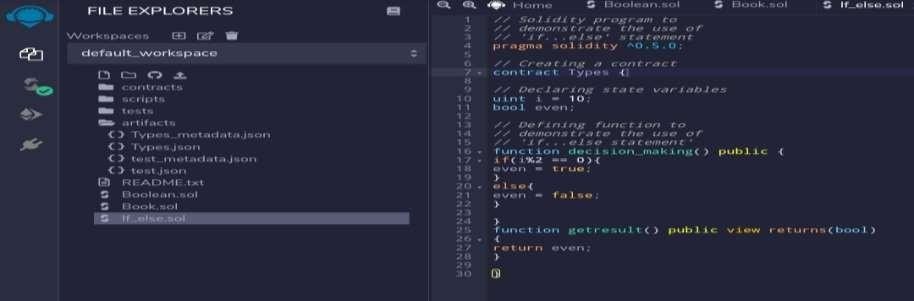
{

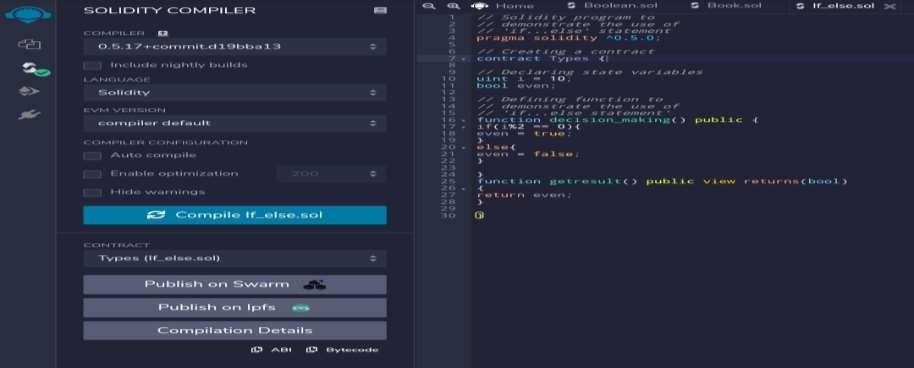
return even;

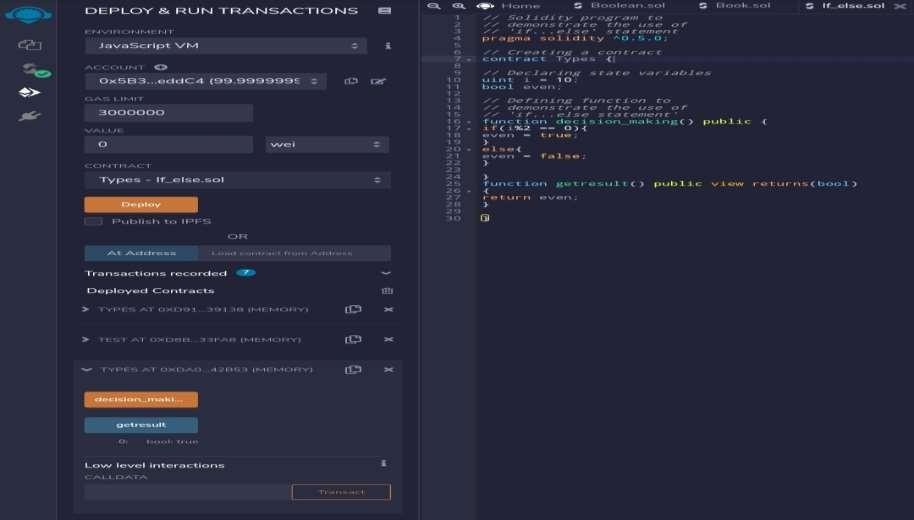
}

}

**Output:**

****

****



**(IV) Strings**

// Solidity program to demonstrate

// how to create a contract pragma solidity ^0.4.23;

// Creating a contract contract Test {

// Declaring variable string str;

// Defining a constructor constructor(string str\_in){ str = str\_in;

}

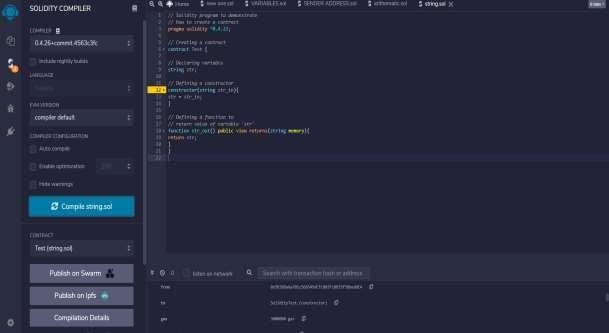
// Defining a function to

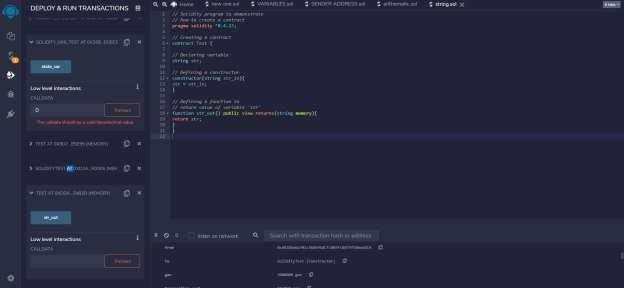
// return value of variable 'str'

function str\_out() public view returns(string memory){ return str;

}

}

**Output**

****

**Practical 5**

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

1. **Arrays**
2. **Enums**
3. **Structs**
4. **Mappings**
5. **Conversations**
6. **Ether Units**
7. **Special Variables**
8. **Arrays**

pragma solidity^0.8.2;

contract types { uint[6] data1; int[5] data ;

function array\_example() public returns (int[5]memory, uint[6]memory){ data = [int(50), -63,77,-28,90];

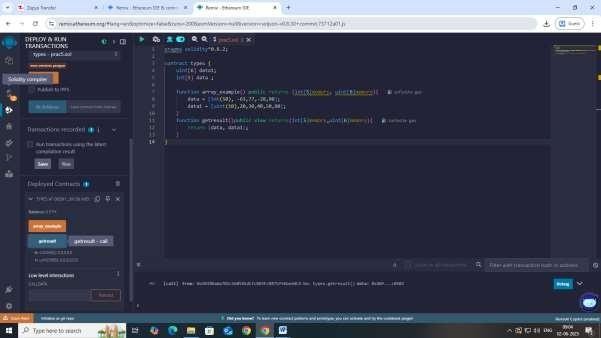
data1 = [uint(10),20,30,40,50,60];

}

function getresult()public view returns(int[5]memory,uint[6]memory){ return (data, data1);

}

}

**Output:**

1. **Enuma**

// Solidity program to demonstrate

// how to use 'enumerator' pragma solidity ^0.8.2;

// Creating a contract contract Types {

// Creating an enumerator enum week\_days

{

Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

}

// Declaring variables of

// type enumerator week\_days week;

week\_days choice;

// Setting a default value week\_days constant default\_value

= week\_days.Sunday;

// Defining a function to

// set value of choice function set\_value() public {

choice = week\_days.Thursday;

}

// Defining a function to

// return value of choice function get\_choice(

) public view returns (week\_days) { return choice;

}

// Defining function to

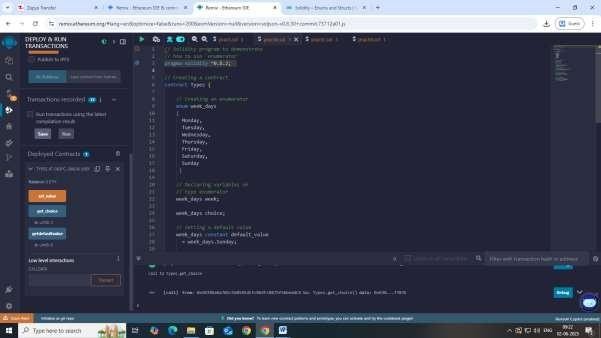
// return default value function getdefaultvalue(

) public pure returns(week\_days) { return default\_value;

}

}

**Output:**

****

1. **Structs**

pragma solidity ^0.8.2; contract test{

struct Book { string title; string author; uint book\_id;

}

Book book;

function setBook() public {

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

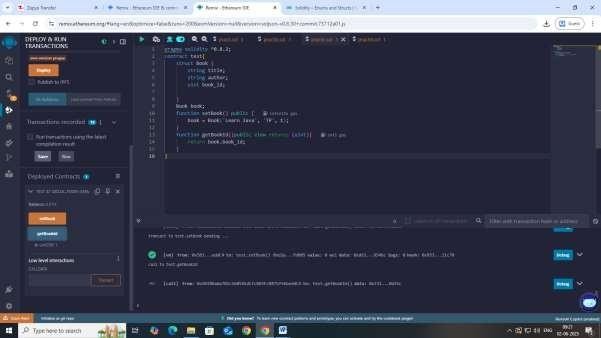
}

function getBookId()public view returns (uint){ return book.book\_id;

}

}

**Output:**

****

1. **Mappings**

pragma solidity ^0.8.2; contract LedgerBalance {

mapping(address => uint) balance;

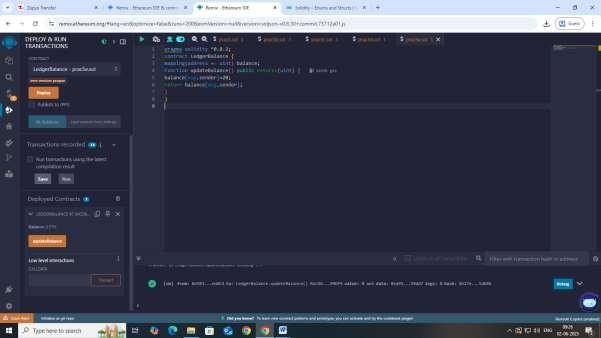
function updateBalance() public returns(uint) { balance[msg.sender]=20;

return balance[msg.sender];

}

}

**Output:**

****

**5e B mapping**:

pragma solidity ^0.8.2; contract LedgerBalance {

mapping(address => string) name;

function updateBalance() public returns(string memory){ name[msg.sender] = "Dip";

return name[msg.sender];

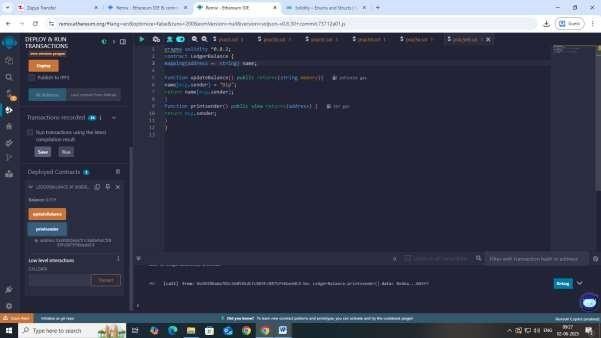
}

function printsender() public view returns(address) { return msg.sender;

}

}

**Output:**

****

1. **Conversations**

// Solidity program to demonstrate

// explicit conversion pragma solidity ^0.8.2;

contract ExplicitConversion

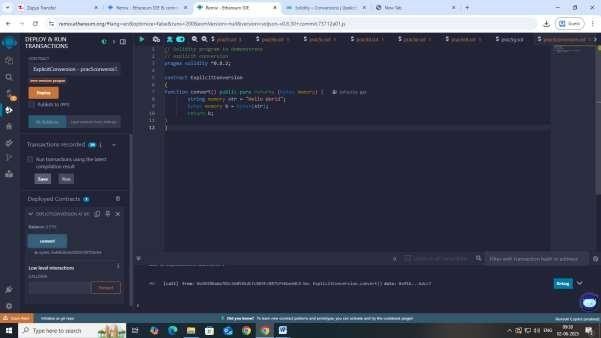
{

function convert() public pure returns (bytes memory) { string memory str = "Hello World";

bytes memory b = bytes(str); return b;

}

}



1. **Ether Units**

// SPDX-License-Identifier: MIT pragma solidity ^0.8.2;

contract EtherUnits {

uint256 public weiValue = 1 ether;

uint256 public gweiValue = 1000000 gwei; uint256 public etherValue = 1 ether;

function getWei() public view returns (uint256) { return weiValue;

}

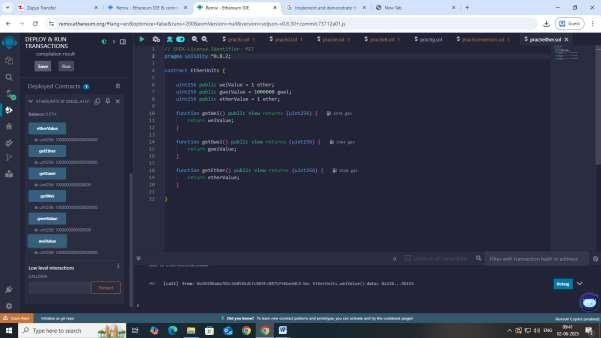
function getGwei() public view returns (uint256) { return gweiValue;

}

function getEther() public view returns (uint256) { return etherValue;

}

}



1. **Special Variables**

// SPDX-License-Identifier: GPL-3.0 pragma solidity ^0.8.2;

/// @title A contract for demonstrate block.number and blockhash

/// @author Jitendra Kumar

/// @notice For now, this contract just show how to return hash value of specific block contract GeeksForGeeks

{

// Declaring state variables

// BlockNumber uint BNumber;

// Hash of current block bytes32 BHashPresent;

// Hash of Previous Block bytes32 BHashPrevious;

// Defining a function to

// return hash value of

// the current block function PresentHash()

public returns(bytes32)

{

BNumber = block.number; BHashPresent =blockhash(BNumber); return BHashPresent;

}

// Defining a function to

// return the hash value of

// the previous block function PreviousHash()

public returns(bytes32)

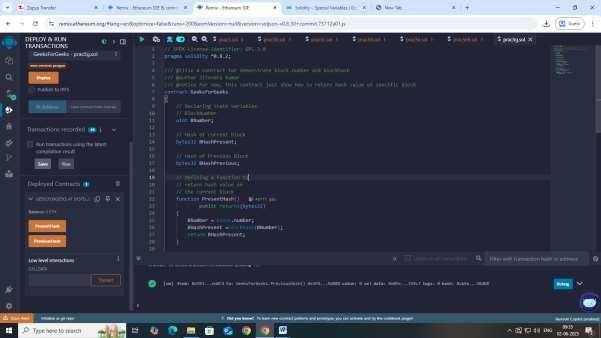
{

BNumber = block.number;

BHashPrevious = blockhash(BNumber - 1); return BHashPrevious;

}

}



**Practical 6**

**Aim: Implement and demonstrate the use 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**

**I). Functions**

pragma solidity ^0.8.2; contract SolidityTest {

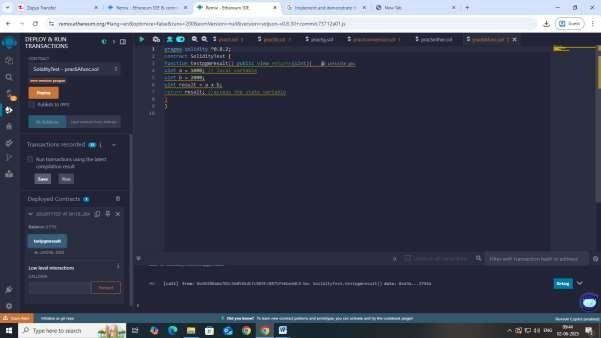
function testpgmresult() public view returns(uint){ uint a = 1000; // local variable

uint b = 2000; uint result = a + b;

return result; //access the state variable

}

}



1. **View Functions**

pragma solidity ^0.8.2; contract Test {

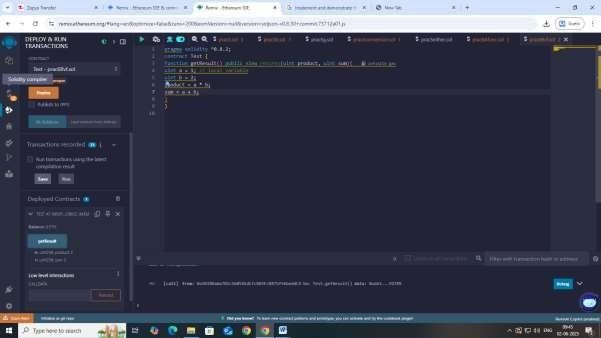
function getResult() public view returns(uint product, uint sum){ uint a = 1; // local variable

uint b = 2; product = a \* b; sum = a + b;

}

}

**Output:**

****

1. **Pure Functions**

// SPDX-License-Identifier: GPL-3.0 pragma solidity ^0.8.2;

/// @title A contract for demonstrating pure functions

/// @notice For now, this contract defining pure function to calculate product and sum of two numbers

contract Test {

function getResult(

) public pure returns( uint product, uint sum){ uint num1 = 2;

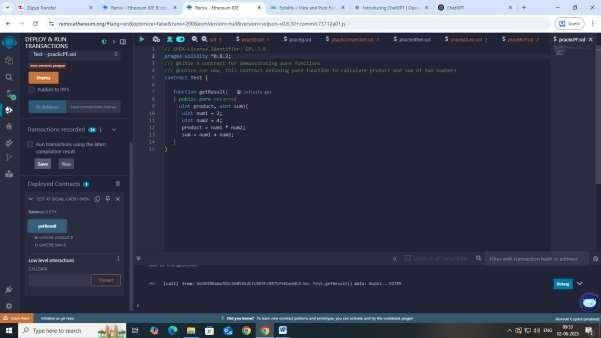
uint num2 = 4;

product = num1 \* num2; sum = num1 + num2;

}

}

**Output:**

****

1. **Fallback Functions**

// SPDX-License-Identifier: GPL-3.0 pragma solidity ^0.8.2;

/// @title A contract for demonstrate fallback function

/// @author Jitendra Kumar

/// @notice For now, this contract just show how to fallback function receive all the ether contract GeeksForGeeks

{

string public calledFallbackFun;

// This fallback function

// will keep all the Ether fallback() external payable{

calledFallbackFun="Fallback function is executed!";

}

function getBalance() public view returns (uint) { return address(this).balance;

}

}

// Creating the sender contract contract Sender

{

function transferEther() public payable

{

//paste the deployed contract address of GeeksForGeeks smart contract here (bool sent, ) =

payable(0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee).call{value: 2 ether}("Transaction Completed!");

require(sent, "Transaction Failed!");

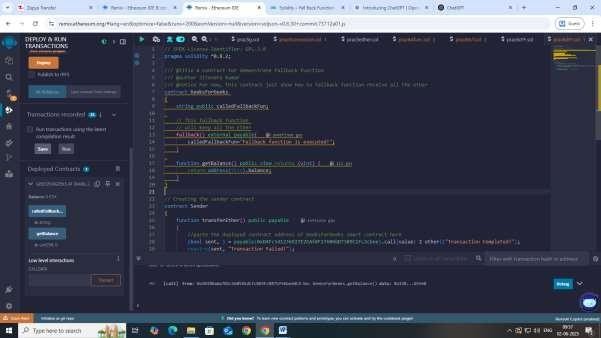
}

function getBalance() public view returns (uint) { return address(this).balance;

}

}

**Output:**

****

1. **Function Overloading**

pragma solidity ^0.8.2; contract Test {

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(1,2);

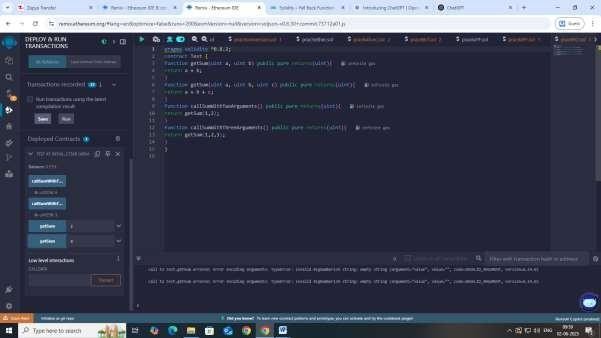
}

function callSumWithThreeArguments() public pure returns(uint){ return getSum(1,2,3);

}

}

Output:



1. **Mathematical Functions**

pragma solidity ^0.8.2; contract Test {

function callAddMod() public pure returns(uint){ return addmod(4, 5, 3);

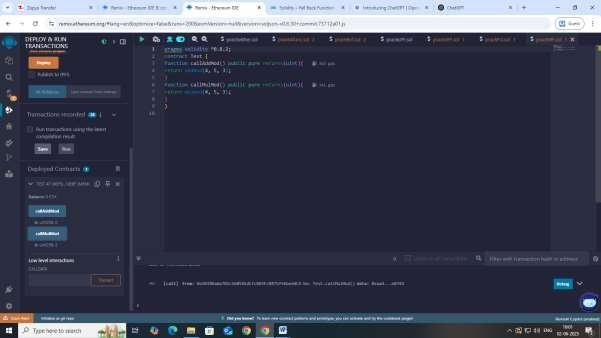
}

function callMulMod() public pure returns(uint){ return mulmod(4, 5, 3);

}

}

**Output:**

****

1. **Cryptographic Functions**

pragma solidity ^0.8.2; contract Test {

function callsha256() public pure returns( bytes32 result){ return sha256("ronaldo");

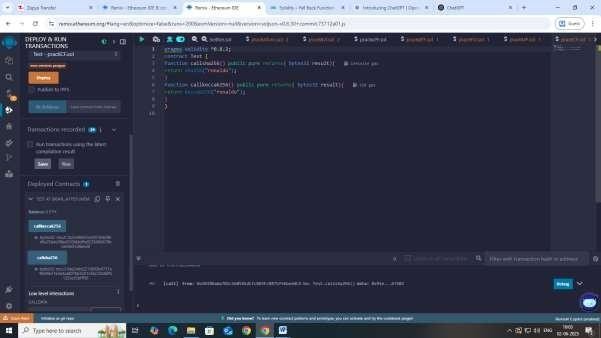
}

function callkeccak256() public pure returns( bytes32 result){ return keccak256("ronaldo");

}

}

**Output:**

****

**Practical 7**

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

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

// Solidity program to

// demonstrate how to

// write a smart contract pragma solidity ^0.8.2;

// Defining a contract contract Test

{

// Declaring state variables uint public var1;

uint public var2; uint public sum;

// Defining public function

// that sets the value of

// the state variable

function set(uint x, uint y) public

{

var1 = x; var2=y; sum=var1+var2;

}

// Defining function to

// print the sum of

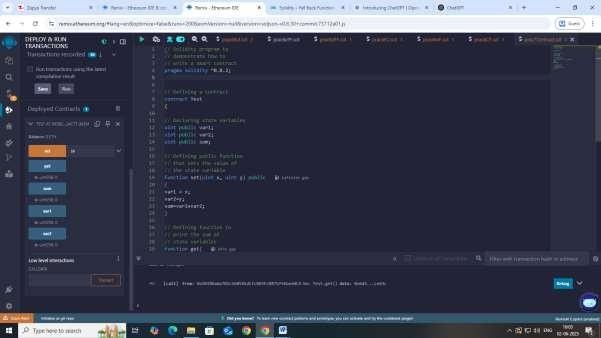
// state variables function get(

) public view returns (uint) { return sum;

}

}

**Output:**

****

1. **Inheritance**

**// Solidity program to demonstrate Single Inheritance pragma solidity ^0.8.2;**

**// Defining contract contract parent{**

**// Declaring internal state varaiable uint internal sum;**

**// Defining external function to set value of internal state variable sum function setValue() external {**

**uint a = 10; uint b = 20; sum = a + b;**

**}**

**}**

**// Defining child contract contract child is parent{**

**// Defining external function to return value of internal state variable sum function getValue() external view returns(uint) {**

**return sum;**

**}**

**}**

**// Defining calling contract contract caller {**

**// Creating child contract object child cc = new child();**

**// Defining function to call setValue and getValue functions function testInheritance() public {**

**cc.setValue();**

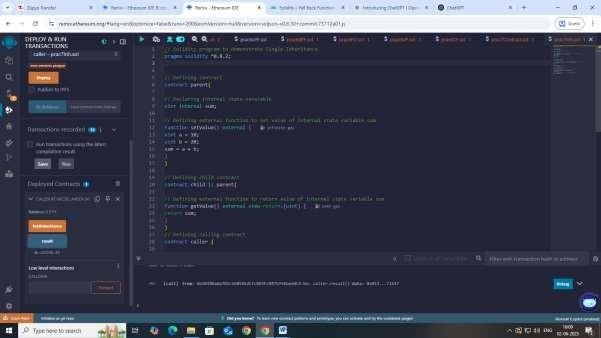
**}**

**function result() public view returns(uint ){ return cc.getValue();**

**}**

**}**

**Output:**

****

1. **Constructors**

// Solidity program to demonstrate

// creating a constructor pragma solidity ^0.8.2;

// Creating a contract

contract constructorExample {

// Declaring state variable string str;

// Creating a constructor

// to set value of 'str' constructor() public { str = "GeeksForGeeks";

}

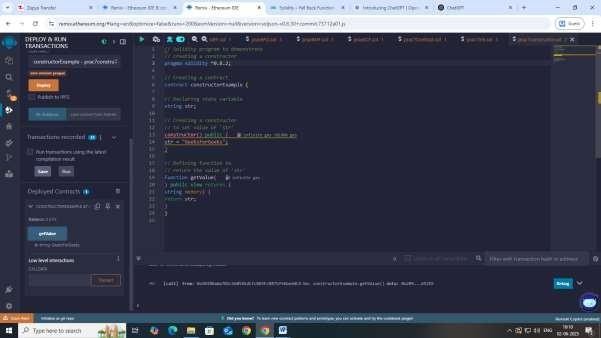
// Defining function to

// return the value of 'str' function getValue(

) public view returns ( string memory) { return str;

}

}



1. **Abstract Class**

// SPDX-License-Identifier: MIT pragma solidity ^0.8.2;

/// @title A contract for describes the properties of Abstract Contract

/// @author Jitendra Kumar

/// @notice For now, this contract just show the functionalities of Abstract Contract abstract contract AbstractContract {

// Declaring functions function getStr(

string memory \_strIn) public view virtual returns( string memory);

function setValue(uint \_in1, uint \_in2) public virtual; function add() public virtual returns(uint);

}

// child contract 'DerivedContract' inheriting an abstract parent contract 'AbstractContract' contract DerivedContract is AbstractContract{

// Declaring private variables uint private num1;

uint private num2;

// Defining functions inherited from abstract parent contract function getStr(

string memory \_strIn) public pure override returns( string memory){

return \_strIn;

}

function setValue(

uint \_in1, uint \_in2) public override{ num1 = \_in1;

num2 = \_in2;

}

function add() public view override returns(uint){ return (num2 + num1);

}

}

// Caller contract contract Call{

// Creating an instance of an abstract contract AbstractContract abs;

// Creating an object of child contract constructor(){

abs = new DerivedContract();

}

// Calling functions inherited from abstract contract function getValues(

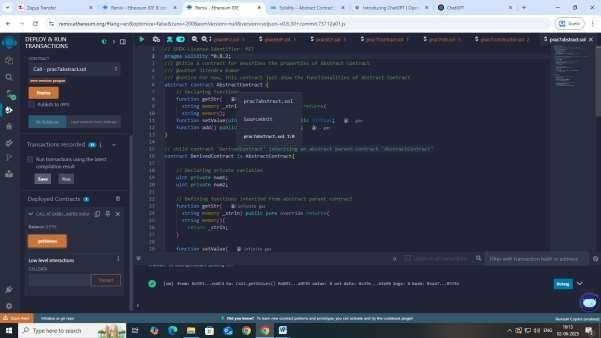
) public returns (string memory,uint){ abs.setValue(10, 16);

return (abs.getStr("GeeksForGeeks"),abs.add());

}

}

**Output:**

****

**Practical 8**

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

**(I) Libraries**

**(II). Assembly**

**(III) Events**

**(IV) Error Handling**

**(I) Libraries**

// Solidity program to demonstrate

// how to deploy a library pragma solidity ^0.8.2;

// Defining Library library LibExample {

// Function to power of

// an unsigned integer function pow(uint a, uint b)

public view returns (uint, address) { return (a \*\* b, address(this));

}

}

// Defining calling contract contract LibraryExample {

// Deploying library using

// "for" keyword

using LibExample for uint; address owner = address(this);

// Calling function pow to

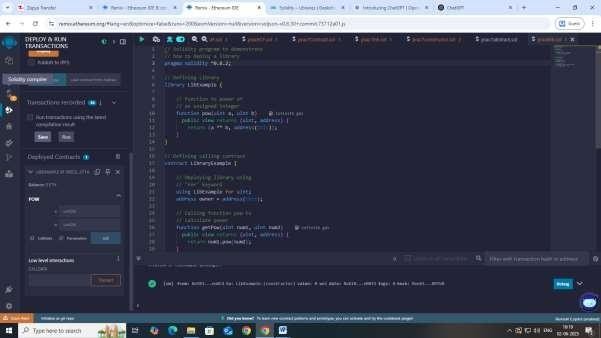
// calculate power

function getPow(uint num1, uint num2) public view returns (uint, address) { return num1.pow(num2);

}

}

**Output:**

****

1. **Assembly**

// SPDX-License-Identifier: GPL-3.0 pragma solidity ^0.8.2;

/// @title A contract for demonstrate Inline Assembly

/// @author Jitendra Kumar

/// @notice For now, this contract just show the function execution using inline assembly contract InlineAssembly {

// Defining function

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

// Inline assembly code assembly {

// Creating a new variable 'c'

// Calculate the sum of 'a+16'

// with the 'add' opcode

// assign the value to the 'c' let c := add(a, 16)

// Use 'mstore' opcode to

// store 'c' in memory

// at memory address 0x80 mstore(0x80, c)

{

// Creating a new variable'

// Calculate the sum of 'sload(c)+12'

// means values in variable 'c'

// with the 'add' opcode

// assign the value to 'd' let d := add(sload(c), 12)

// assign the value of 'd' to 'b' b := d

// 'd' is deallocated now

}

// Calculate the sum of 'b+c' with the 'add' opcode

// assign the value to 'b' b := add(b, c)

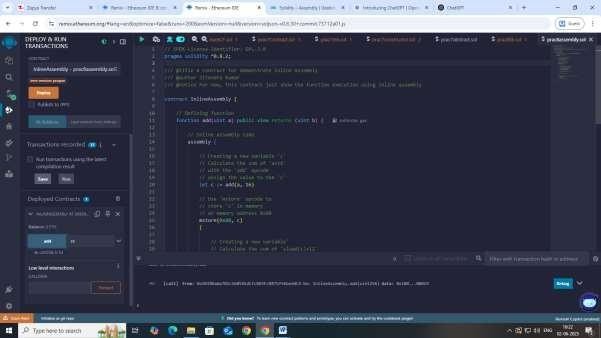
// 'c' is deallocated here

}

}

}

**Output:**

****

1. **Events**

// Solidity program to demonstrate

// creating an event pragma solidity ^0.8.2;

// Creating a contract contract eventExample {

// Declaring state variables uint256 public value = 0;

// Declaring an event

event Increment(address owner);

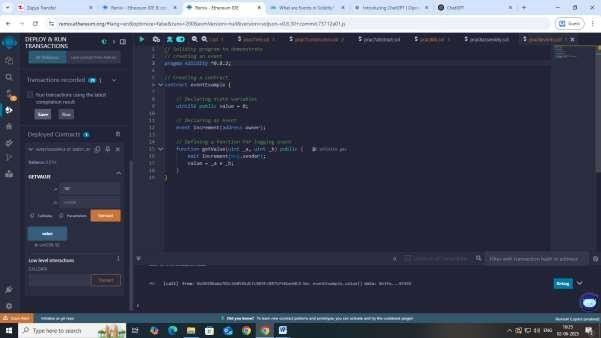
// Defining a function for logging event function getValue(uint \_a, uint \_b) public {

emit Increment(msg.sender); value = \_a + \_b;

}

}

**Output:**

****

1. **Error Handling**

pragma solidity ^0.8.2;

// Creating a contract contract requireStatement {

// Defining function to check input function checkInput(

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

require(\_input >= 0, "invalid uint8"); require(\_input <= 255, "invalid uint8"); return "Input is Uint8";

}

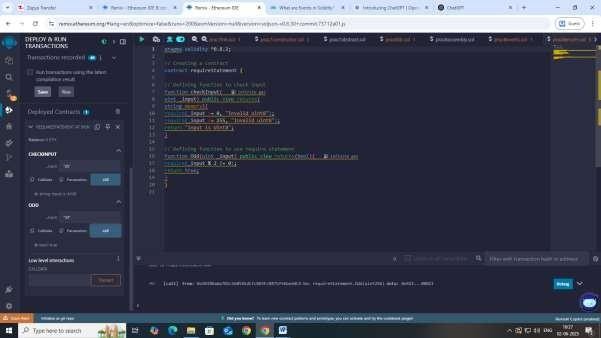
// Defining function to use require statement function Odd(uint \_input) public view returns(bool){ require(\_input % 2 != 0);

return true;

}

}

**Output:**

****

// Solidity program to demonstrate assert statement

// SPDX-License-Identifier: MIT pragma solidity ^0.8.2;

contract AssertExample { uint public balance;

function deposit(uint amount) public { balance += amount;

// After deposit, the balance should always be non-negative. assert(balance >= 0);

}

function withdraw(uint amount) public {

// Check if the withdrawal amount is valid. require(amount <= balance, "Insufficient balance"); balance -= amount;

// After withdrawal, the balance should always be non-negative. assert(balance >= 0);

}

function transfer(address recipient, uint amount) public {

// Check if the transfer amount is valid. require(amount <= balance, "Insufficient balance"); balance -= amount;

// After transfer, the balance should always be non-negative. assert(balance >= 0);

// Verify that recipient address is not zero

assert(recipient != address(0));

// Send the amount to the recipient (This is a simplified example).

// In a real-world scenario, you'd likely use a payable transfer.

}

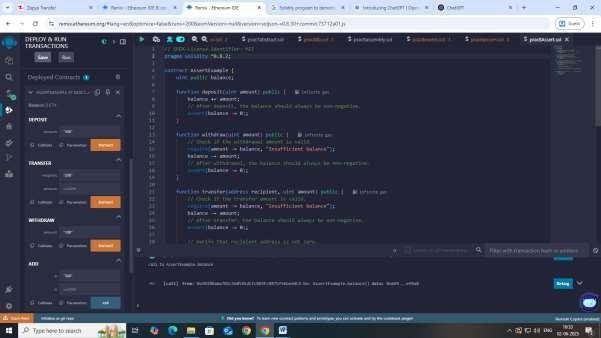
function add(uint a, uint b) public pure returns (uint) { uint result = a + b;

// Check for overflow assert(result >= a); return result;

}

}

**Output:**

****