**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Topic** | **Date of Practical** | **Date of**  **Submission** | **Remark** |
| **1.** | **Practical No 1** |  |  |  |
| a | Develop a secure messaging application where users can exchange messages securely using RSA encryption. Implement a mechanism for generating RSA key pairs and  encrypting/decrypting messages. |  |  |  |
| b | Allow users to create multiple transactions and display them in  an organised format. |  |  |  |
| c | Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the  receiver's account. |  |  |  |
| d | Implement a function to add new blocks to the miner and dump  the blockchain. |  |  |  |
|  |  |  |  |  |
| **2** | **Practical No. 2** |  |  |  |
| a | Write a python program to demonstrate mining. |  |  |  |
| b | Demonstrate the use of the Bitcoin Core API to interact with a  Bitcoin Core node. |  |  |  |
|  |  |  |  |  |
| **3** | **Practical No. 3** |  |  |  |
| a | Write a Solidity program that demonstrates various types of  functions including regular functions, view functions, pure functions, and the fallback function. |  |  |  |
| b | Write a Solidity program that demonstrates function  overloading, mathematical functions, and  cryptographic functions. |  |  |  |
| c | Write a Solidity program that demonstrates various features including contracts, inheritance,  constructors, abstract contracts, interfaces. |  |  |  |
| d | Write a Solidity program that demonstrates use of libraries,  assembly, events, and error handling. |  |  |  |
|  |  |  |  |  |
| **4** | **Practical No. 4** |  |  |  |
| a | Demonstration on interacting with NFT |  |  |  |

**Practical 1**

**a) Develop a secure messaging application where users can exchange messages securely using RSA encryption. Implement a mechanism for generating RSA key pairs and encrypting/decrypting messages.**

**Code:-**

# import libraries 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

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

@property

def identity(self):

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

class Transaction:

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

self.recipient = recipient self.value = value

self.time = datetime.datetime.now() def to\_dict(self):

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)

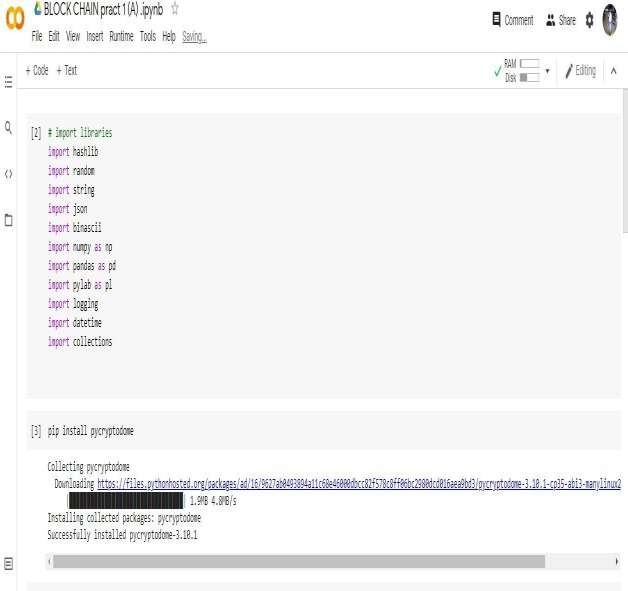
h = SHA.new(str(self.to\_dict()).encode('utf8')) return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh = Client() Ramesh = Client()

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

print (signature)

**Output:-**

****



**b. Allow users to create multiple transactions and display them in an organised format. Code:-**

def display\_transaction(transaction): #for transaction in transactions: dict = transaction.to\_dict()

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

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

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

print (' ')

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

print (' ')

transactions = [] Dinesh = Client() Ramesh = Client()

Seema = Client() Vijay = Client()

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

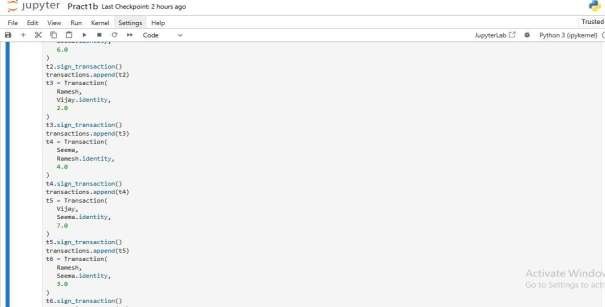
)

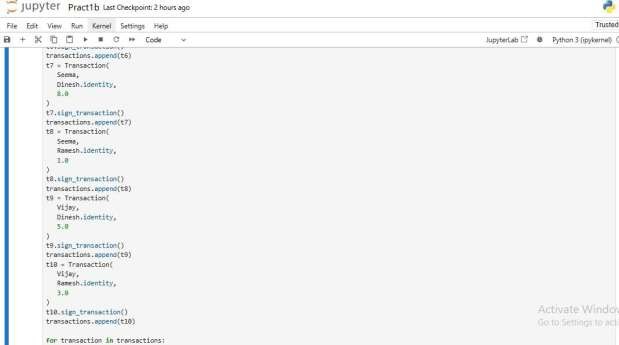
t10.sign\_transaction() transactions.append(t10)

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

**Output:-**

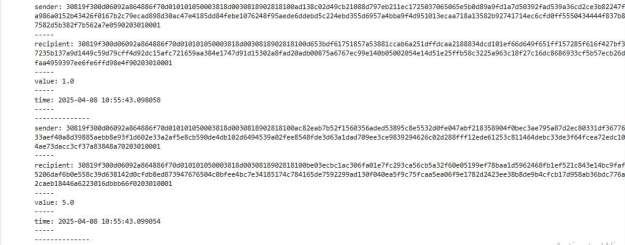
****

****



****







1. **Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the receiver's account.**

**Code:**

class Transaction:

# Simulate a bank with account balances account\_balances = {}

def init(self, sender, receiver, amount): self.sender = sender

self.receiver = receiver self.amount = amount

def transfer(self):

# Check if both sender and receiver exist in the system if self.sender not in Transaction.account\_balances:

return f"Sender '{self.sender}' does not have an account." if self.receiver not in Transaction.account\_balances:

return f"Receiver '{self.receiver}' does not have an account."

# Check if the sender has enough funds

if Transaction.account\_balances[self.sender] < self.amount: return f"Insufficient funds in sender's account."

# Transfer the amount Transaction.account\_balances[self.sender] -= self.amount Transaction.account\_balances[self.receiver] += self.amount

return f"Transfer of {self.amount} from {self.sender} to {self.receiver} successful."

@classmethod

def create\_account(cls, name, initial\_balance=0): cls.account\_balances[name] = initial\_balance

# Example usage:

# Creating accounts Transaction.create\_account("Alice", 1000)

Transaction.create\_account("Bob", 500)

# Creating a transaction

txn = Transaction("Alice", "Bob", 200) result = txn.transfer()

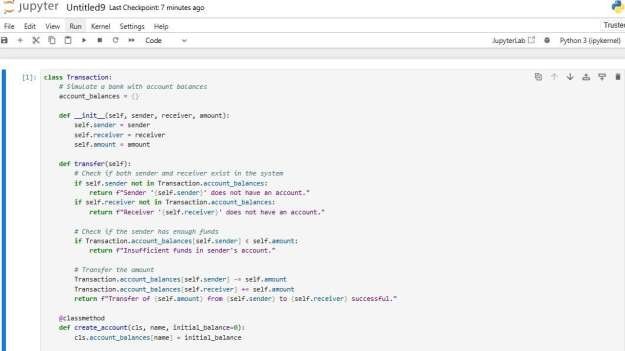
print(result)

# Checking balances

print("Alice's balance:", Transaction.account\_balances["Alice"])

print("Bob's balance:", Transaction.account\_balances["Bob"])

**Output:**

****

1. **Implement a function to add new blocks to the miner and dump the blockchain. Code:**

class Block:

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

last\_block\_hash = ""

Dinesh = Client()

t0 = Transaction ( "Genesis", Dinesh.identity, 500.0

)

block0 = Block()

block0.previous\_block\_hash = None Nonce = None

block0.verified\_transactions.append (t0)

digest = hash (block0) last\_block\_hash = digest

TPCoins = []

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\_transactions: display\_transaction (transaction)

print (' ')

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

TPCoins.append (block0) dump\_blockchain(TPCoins) def sha256(message):

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

def mine(message, difficulty=1):

assert difficulty >= 1 prefix = '1' \* difficulty for i in range(1000):

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

print ("after " + str(i) + " iterations found nonce: "+ digest) return digest

last\_transaction\_index = 0

block = Block() for i in range(3):

temp\_transaction = transactions[last\_transaction\_index] # validate transaction

# if valid

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

mine ("test message", 2)

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine (block, 2)

digest = hash (block) TPCoins.append (block) last\_block\_hash = digest

# Miner 2 adds a block block = Block()

for i in range(3):

temp\_transaction = transactions[last\_transaction\_index] # validate transaction

# if valid

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

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine(block, 2)

digest = hash (block) TPCoins.append (block) last\_block\_hash = digest # Miner 3 adds a block block = Block()

for i in range(3):

temp\_transaction = transactions[last\_transaction\_index]

#display\_transaction (temp\_transaction) # validate transaction

# if valid

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

block.previous\_block\_hash = last\_block\_hash block.Nonce = mine (block, 2)

digest = hash (block)

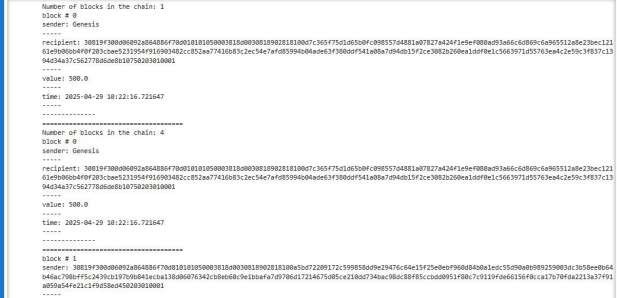
TPCoins.append (block) last\_block\_hash = digest dump\_blockchain(TPCoins)

**Output:**

****



****



****



****

**Practical 2**

* 1. **Write a python program to demonstrate mining. Code:-**

import hashlib import time

class Block:

def init(self, index, previous\_hash, timestamp, data, hash, nonce=0): self.index = index

self.previous\_hash = previous\_hash self.timestamp = timestamp self.data = data

self.hash = hash self.nonce = nonce

def calculate\_hash(index, previous\_hash, timestamp, data, nonce): return

hashlib.sha256(f"{index}{previous\_hash}{timestamp}{data}{nonce}".encode()).hexdigest ()

def create\_genesis\_block():

return Block(0, "0", time.time(), "Genesis Block", calculate\_hash(0, "0", time.time(), "Genesis Block", 0))

def proof\_of\_work(block, difficulty=2): nonce = 0

while True:

hash\_try = calculate\_hash(block.index, block.previous\_hash, block.timestamp, block.data, nonce)

if hash\_try.startswith('0' \* difficulty): return nonce, hash\_try

nonce += 1

def mine\_block(previous\_block, data):

index = previous\_block.index + 1 timestamp = time.time()

nonce, hash\_new = proof\_of\_work(Block(index, previous\_block.hash, timestamp, data, "temp"), difficulty=2)

return Block(index, previous\_block.hash, timestamp, data, hash\_new, nonce)

blockchain = [create\_genesis\_block()] previous\_block = blockchain[0]

num\_blocks\_to\_add = 3

for i in range(num\_blocks\_to\_add): block\_data = f"Block #{i+1}"

new\_block = mine\_block(previous\_block, block\_data) blockchain.append(new\_block)

previous\_block = new\_block

print(f"Block #{new\_block.index} mined: {new\_block.hash}")

for block in blockchain:

print(f"Index: {block.index}, Hash: {block.hash}, Previous Hash:

{block.previous\_hash}, Data: {block.data}, Nonce: {block.nonce}")

**Output:**

****

* 1. **Demonstrate the use of the Bitcoin Core API to interact with a Bitcoin Core node. Code:-**

import requests

# Task 1: Get information regarding the current block def get\_current\_block\_info():

response = requests.get("https://blockchain.info/latestblock") block\_info = response.json()

print("Current block information:") print("Block height:", block\_info['height']) print("Block hash:", block\_info['hash']) print("Block index:", block\_info['block\_index']) print("Timestamp:", block\_info['time'])

# Task 3: Get balance of an address def get\_address\_balance(address):

response = requests.get(f"https://blockchain.info/q/addressbalance/{address}") balance = float(response.text) / 10\*\*8

print("Balance of address", address, ":", balance, "BTC")

# Example usage

if name == " main ":

# Task 1: Get information regarding the current block get\_current\_block\_info()

# Task 3: Get balance of an address

address = "3Dh2ft6UsqjbTNzs5zrp7uK17Gqg1Pg5u5" get\_address\_balance(address)

**Output:-**

****

**Practical 3**

**a) Implement and demonstrate the use of the following in Solidity:**

(I) Functions (II) View Functions (III) Pure Functions (IV) Fallback Functions

1. **Functions**

pragma solidity

^0.5.0; contract SolidityTest {

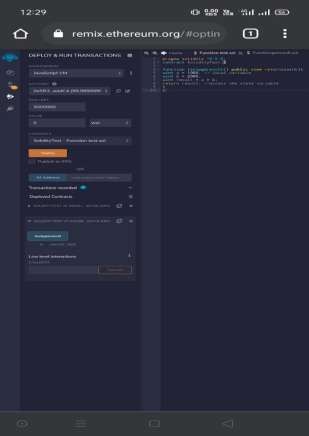
function testpgmresult() public view returns(uint){ uint a = 1000; // local variable

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

return result; //access the state variable

}}

**Output:**



1. **View Functions** pragma solidity ^0.5.0; contract Test {

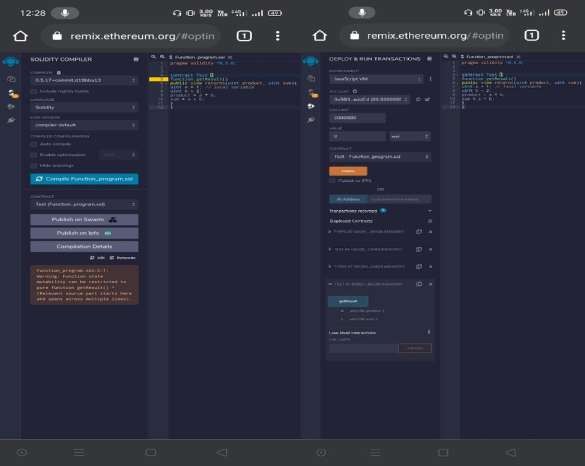
function getResult() public view returns(uint product, uint sum){ uint a = 1; // local variable

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

}

}

**Output:**

****

1. **Pure Functions** pragma solidity ^0.5.0; contract C {

//private state variable uint private data;

//public state variable uint public info;

//constructor constructor() public { info = 10;

}

//private function

function increment(uint a) private pure returns(uint) { return a + 1; }

//public function

function updateData(uint a) public { data = a; }

function getData() public view returns(uint) { return data; }

function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}

//Derived Contract contract E is C { uint private result; C private c;

constructor() public { c = new C();

}

function getComputedResult() public {

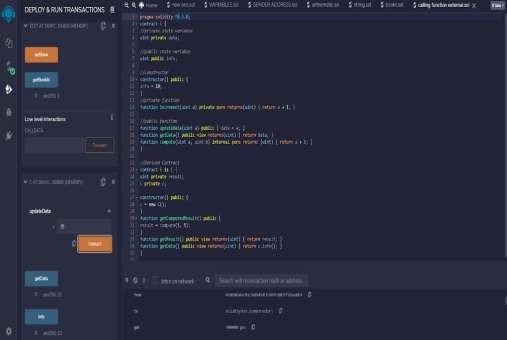
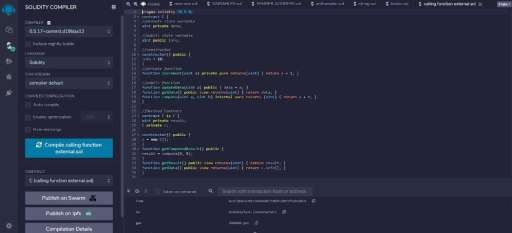
result = compute(3, 5);

}

function getResult() public view returns(uint) { return result; } function getData() public view returns(uint) { return c.info(); }

}

**Output:**



1. **Fallback function**

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract FallbackDemo {

event Received(address sender, uint amount, string message);

// This fallback function is triggered if an unknown function is called fallback() external payable {

emit Received(msg.sender, msg.value, "Fallback function was called");

}

// Receive function to receive Ether directly receive() external payable {

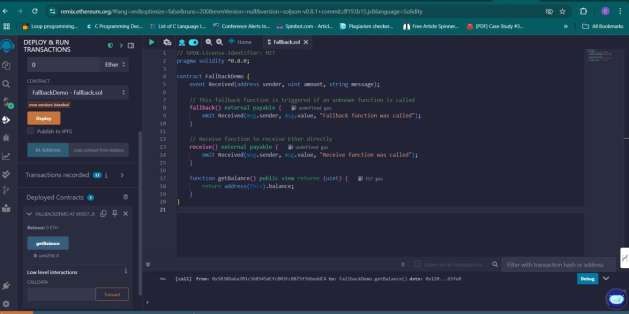
emit Received(msg.sender, msg.value, "Receive function was called");

}

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

}}

**Output:**

****

1. **Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.**
   1. Function Overloading (II) Mathematical functions (III) Cryptographic functions
      1. **Function Overloading** pragma solidity ^0.5.0; 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;

}

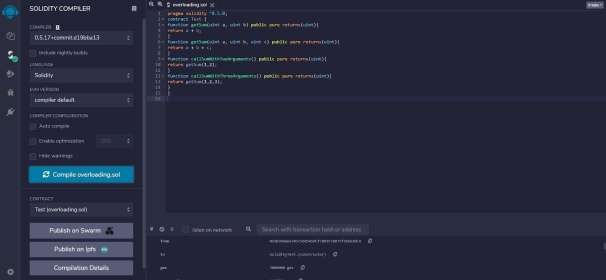
function callSumWithTwoArguments() public pure returns(uint){ return getSum(1,2);

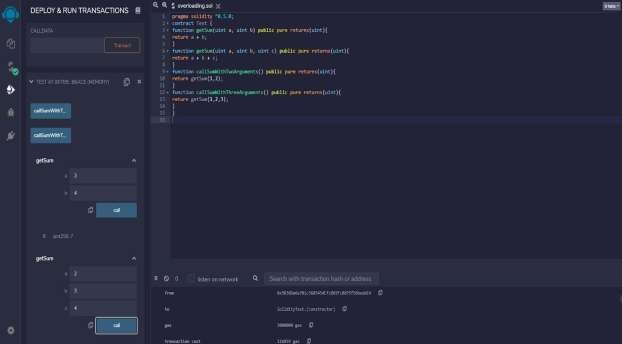
}

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

}}

**Output:**

****



* + 1. **Mathematical Functions** pragma solidity ^0.5.0; contract Test {

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

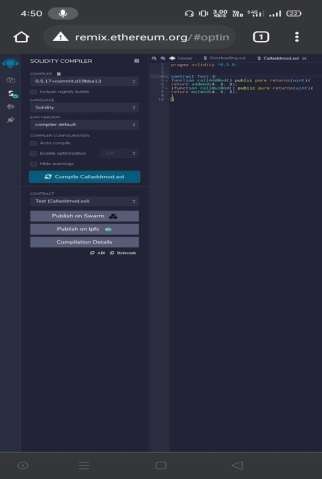
}

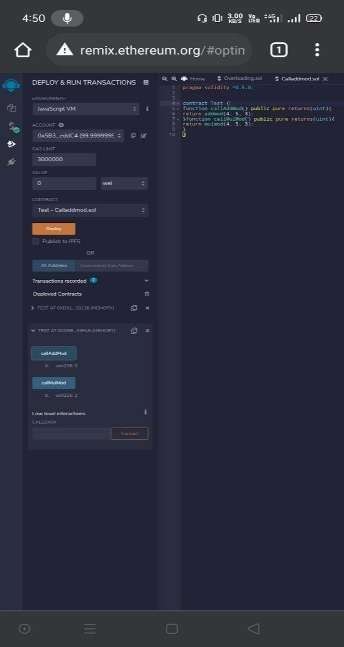
function callMulMod() public pure returns(uint){ return mulmod(4, 5, 3);

}

}

**Output:**





* + 1. **Cryptographic Functions**

pragma solidity ^0.5.0; contract Test {

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

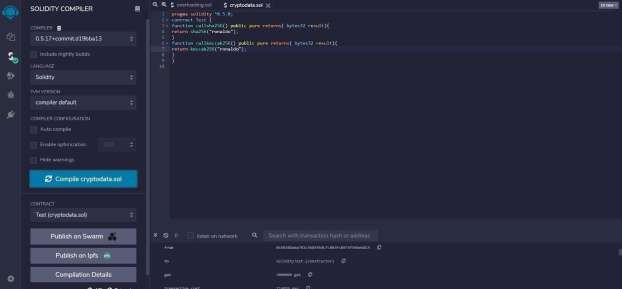
}

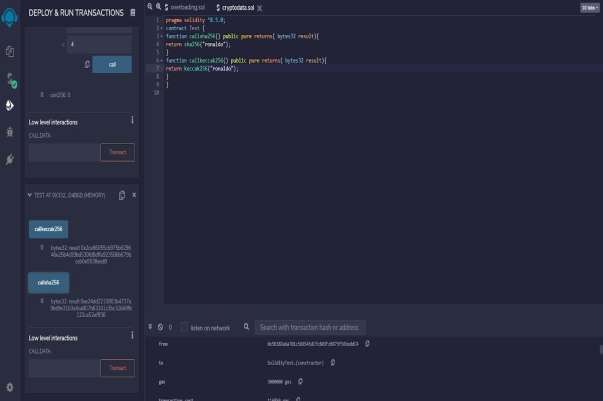
function callkeccak256() public pure returns( bytes32 result){ return keccak256("ronaldo");

}

}

**Output:**

****

****

1. **Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.**
   1. Contracts (II) Inheritance (III) Constructors (IV) Abstract Class (V) Interfaces
2. **Contracts**

// Solidity program to

// demonstrate how to

// write a smart contract

pragma solidity >= 0.4.16 < 0.7.0;

// 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

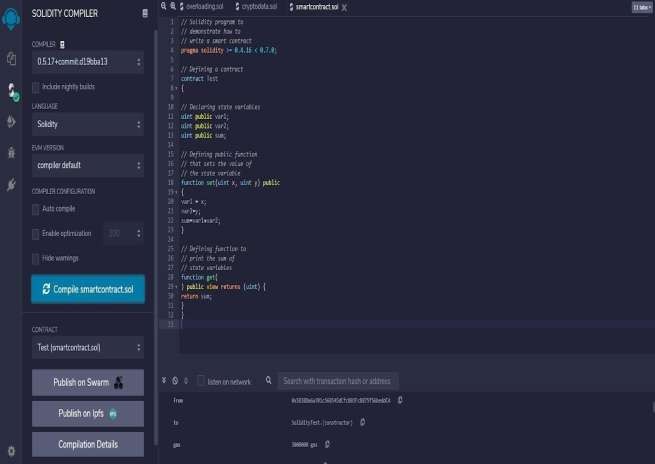
function get(

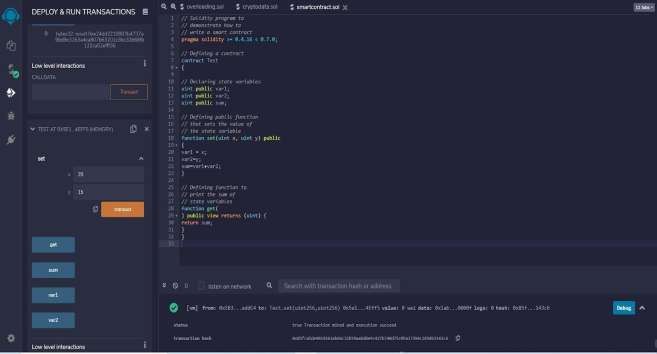
) public view returns (uint) { return sum;

}

}

**Output:**

****



1. **Inheritance**

// Solidity program to demonstrate Single Inheritance pragma solidity >=0.4.22 <0.6.0;

// 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();

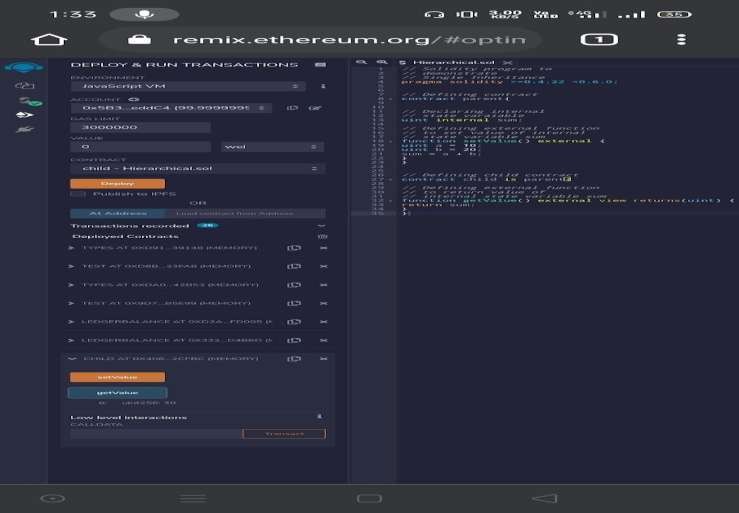
}

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

}

}

**Output:**

****

1. **Constructors**

// Solidity program to demonstrate

// creating a constructor pragma solidity ^0.5.0;

// 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

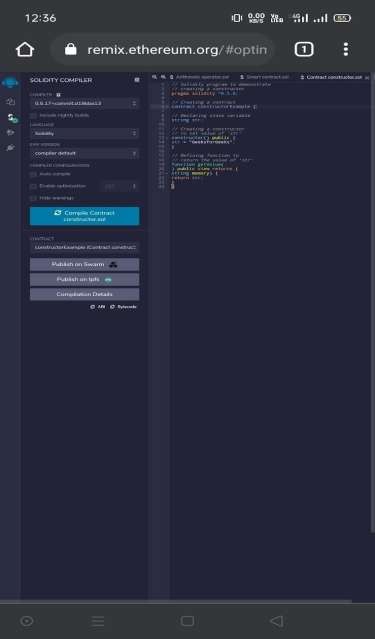
// return the value of 'str' function getValue(

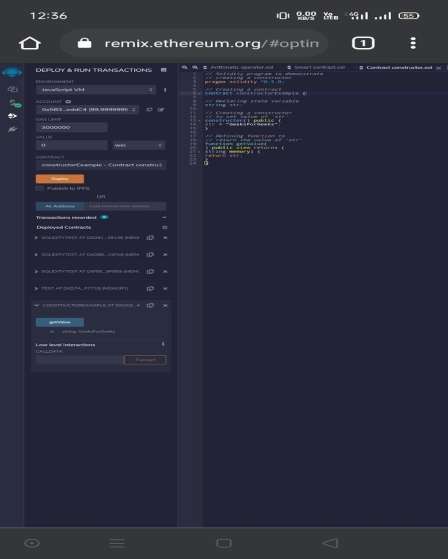
) public view returns ( string memory) { return str;

}

}

**Output:**

****



1. **Abstract Class**

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

/// @title Abstract Contract Demo

//

// Abstract Contract

//

abstract contract Shape {

function area() public view virtual returns (uint); function perimeter() public view virtual returns (uint);

}

//

// Derived Contract: Rectangle

//

contract Rectangle is Shape { uint public length;

uint public width;

constructor(uint \_length, uint \_width) { length = \_length;

width = \_width;

}

// Implement area

function area() public view override returns (uint) {

return length \* width;

}

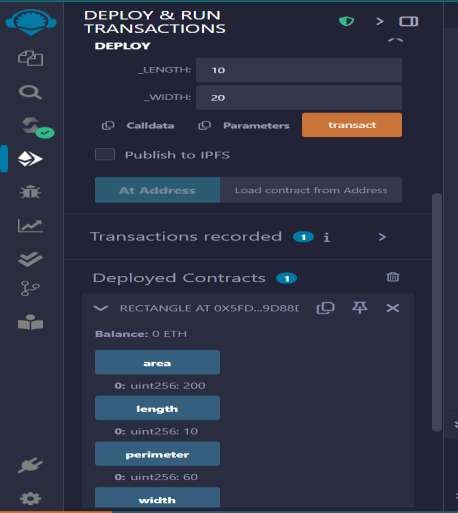
// Implement perimeter

function perimeter() public view override returns (uint) { return 2 \* (length + width);

}

}

**Output:**

****

1. **Interface**

An interface defines function signatures but does not implement them.

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

interface ICalculator {

function add(uint a, uint b) external pure returns (uint); function subtract(uint a, uint b) external pure returns (uint);

}

contract Calculator is ICalculator {

function add(uint a, uint b) public pure override returns (uint) { return a + b;

}

function subtract(uint a, uint b) public pure override returns (uint) { return a - b;

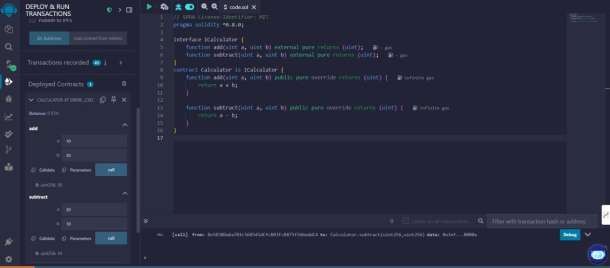
}

}

**Explanation:**

* ICalculator defines two function signatures: add() and subtract().
* Any contract that inherits ICalculator must implement these functions.

Example Usage in a Derived Contract:



**D. Implement and demonstrate the use of the following in Solidity:**

(I) Libraries (II) Assembly (III) Events (IV) Error Handling

1. **Libraries**

pragma solidity ^0.5.0;

// 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";

}

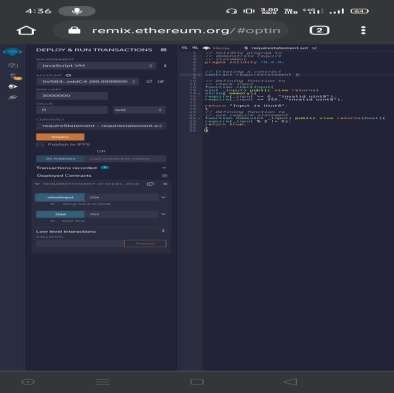
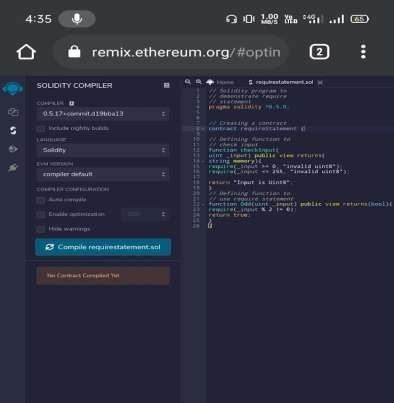
// Defining function to use require statement function Odd(uint \_input) public view returns(bool){ require(\_input % 2 != 0);

return true;

}

}

**Output:**

****

1. **Assembly**

pragma solidity ^0.5.0;

// Creating a contract contract assertStatement {

// Defining a state variable bool result;

// Defining a function to check condition function checkOverflow(

uint \_num1, uint \_num2) public { uint8 sum = \_num1 + \_num2; assert(sum<=255);

result = true;

}

// Defining a function to print result of assert statement function getResult() public view returns(string memory){ if(result == true){

return "No Overflow";

}

else{

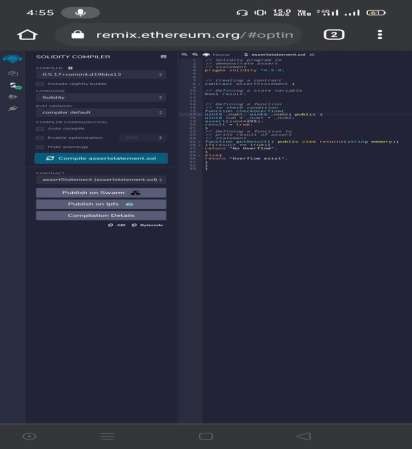
return "Overflow exist";

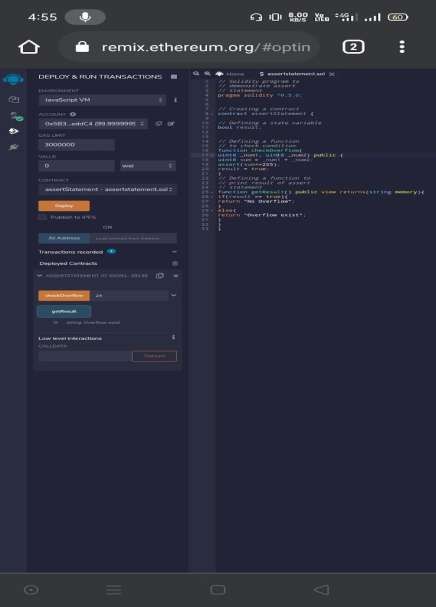
}

}

}

**Output:**

****



1. **Events**

pragma solidity ^0.5.0;

// Creating a contract contract assertStatement {

// Defining a state variable bool result;

// Defining a function

// to check condition

function checkOverflow(uint8 sum) public { assert(sum<=255);

result = true;

}

// Defining a function to print result of assert statement function getResult() public view returns(string memory){ if(result == true){

return "No Overflow";

}

else{

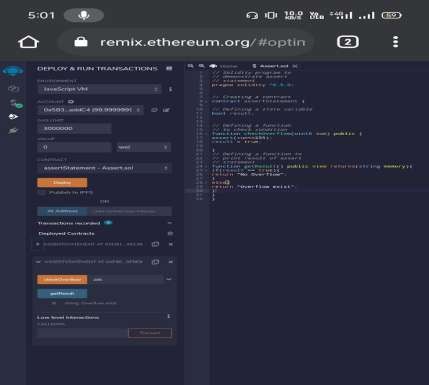
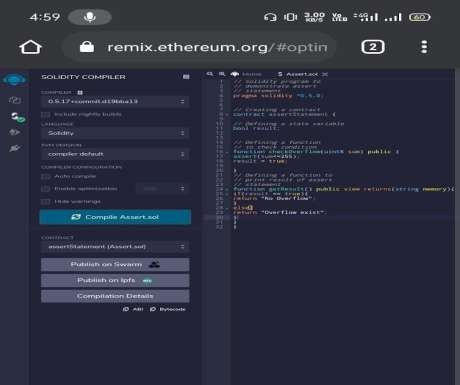
return "Overflow exist";

}

}

}

**Output:**

****

1. **Error Handling**

pragma solidity ^0.5.0;

// Creating a contract contract revertStatement {

// Defining a function to check condition function checkOverflow(

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

uint sum = \_num1 + \_num2; if(sum < 0 || sum > 255){ revert(" Overflow Exist");

}

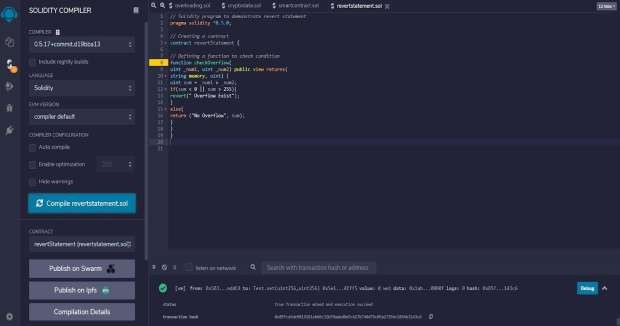
else{

return ("No Overflow", sum);

}}

}}

**Output:**

****

**Practical 4.**

**Demonstration on interacting with NFT NFT (By using Metamask)**

**Step 1: Create Your Wallet (MetaMask)**

* 1. Go to https://metamask.io and install the browser extension.
  2. Click “Create Wallet”.
  3. Save your **secret recovery phrase** somewhere safe.



**How to Install MetaMask Browser Extension**

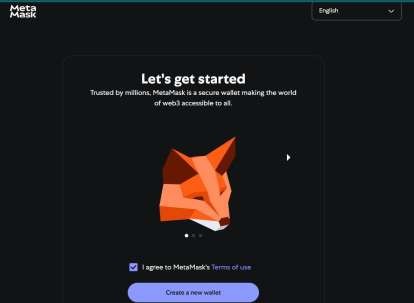
□\_;**Step-by-Step (for Chrome, Brave, Edge)**

1. **Open your browser** (Google Chrome, Brave, or Microsoft Edge).
2. **Go to the official website**:

🔗 https://metamask.io

1. Click **"Download"** from the top menu.
2. It will detect your browser. Click:
   * **Chrome/Edge** → Click **"Install MetaMask for [Your Browser]"**
3. You’ll be taken to the **Chrome Web Store** (or equivalent).
4. Click **“Add to Chrome”** or **“Add Extension”**

**Create your wallet**

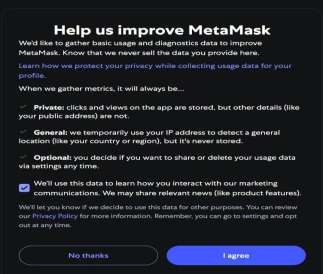
****

Add the password



Secure the wallet

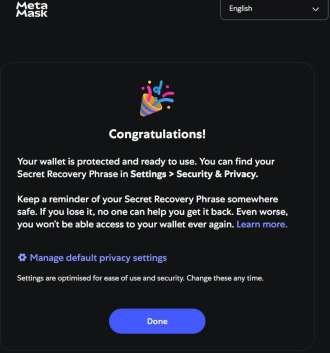


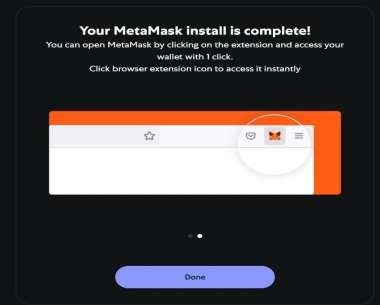




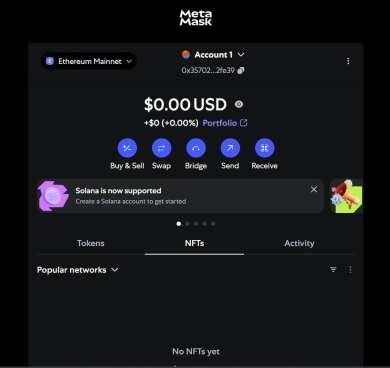
Type the missing words



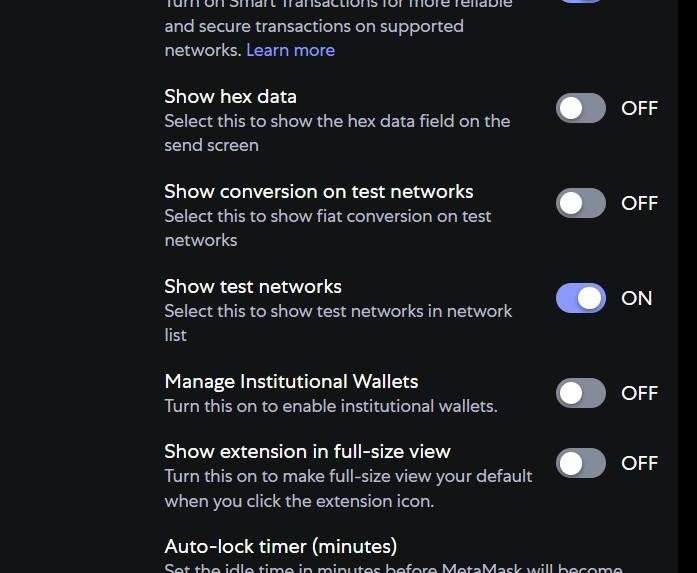




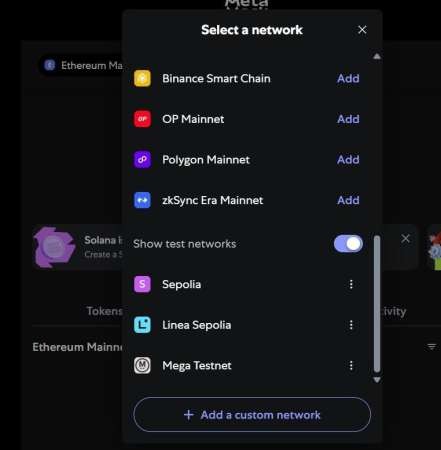
Click on the 3 dots on the right hand side select Setting-Advance



Scroll down make the **show test network ON**

****

1. Open MetaMask.
2. Click on your **profile icon** (top right corner).
3. Select **“Settings”**.
4. Navigate to **“Advanced”**.
5. Scroll down and **toggle ON** the option: **“Show test networks”**.



**Step 2: Switch to the Sepolia Test Network**

1. Close the settings.
2. Click the **network dropdown** at the top (it might say "Ethereum Mainnet").
3. From the list, select **“Sepolia Test Network”**.Alchemy

**Step 3: Obtain Sepolia Test ETH**

To interact with the Sepolia testnet, you'll need some test ETH:

1. Visit a Sepolia faucet, such as Alchemy's Sepolia Faucet.
2. Enter your MetaMask wallet address.

**How to Get Sepolia ETH – Step-by-Step**

**Option 1: Use Alchemy's Faucet**

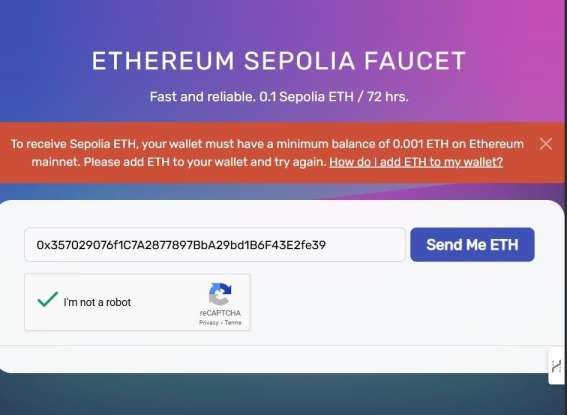
1. Open this website in your browser: https://sepoliafaucet.com
2. You’ll see a field to enter your **wallet address**.
3. Copy your wallet address from MetaMask:
   * Click MetaMask ◻
   * At the top, click your address to copy (looks like 0x123...)
4. Paste the address into the faucet field.
5. Click **“Send me ETH”**
6. Wait a few seconds — it will send you test ETH for Sepolia.
7. Follow the instructions to receive test ETH.ChainList+15Moralis Developers+15MetaMask Help Center+15

**Or**

**Option 2 Sepolia Network Details (For Manual Addition)**

If Sepolia doesn't appear automatically, you can add it manually:

* **Network Name**: Sepolia Test Network
* **New RPC URL**: https://rpc.sepolia.org
* **Chain ID**: 11155111
* **Currency Symbol**: SepoliaETH
* **Block Explorer URL**: https://sepolia.etherscan.io





**Step 4: Go to OpenSea Testnet**

1. Visit: https://testnets.opensea.io
2. Click “Connect Wallet” and choose MetaMask.
3. Approve in MetaMask

**Step 5: Create Your NFT**

1. Click **“Create”** on OpenSea.
2. Upload a picture (e.g., cat.jpg or your digital art).
3. Add:
   * **Name** (e.g., My First NFT)
   * **Description** (e.g., This is my first ever NFT)
   * **Blockchain**: Select **Goerli/ Sepolia**
4. Click “Create”.