**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No** | **Practical** | **Date** | **Sign** |
| 1 | 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. |  |  |
| 2 | Allow users to create multiple transactions and display them in an organised format. |  |  |
| 3 | 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. |  |  |
| 4 | Implement a function to add new blocks to the miner and dump the blockchain. |  |  |
| 5 | Write a python program to demonstrate mining. |  |  |
| 6 | Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function. |  |  |
| 7 | Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions. |  |  |
| 8 | Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces. |  |  |
| 9 | Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling. |  |  |

**Practical 1**

**Aim :- 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 :**

# Install cryptography if not already installed

!pip install cryptography

from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.primitives import serialization, hashes

import base64

# Function to generate RSA key pairs

def generate\_rsa\_key\_pair():

    private\_key = rsa.generate\_private\_key(public\_exponent=65537, key\_size=2048)

    public\_key = private\_key.public\_key()

    return private\_key, public\_key

# Function to serialize keys

def serialize\_keys(private\_key, public\_key):

    priv\_pem = private\_key.private\_bytes(

        encoding=serialization.Encoding.PEM,

        format=serialization.PrivateFormat.PKCS8,

        encryption\_algorithm=serialization.NoEncryption()

    )

    pub\_pem = public\_key.public\_bytes(

        encoding=serialization.Encoding.PEM,

        format=serialization.PublicFormat.SubjectPublicKeyInfo

    )

    return priv\_pem, pub\_pem

# Encrypt a message using public key

def encrypt\_message(public\_key, message):

    encrypted = public\_key.encrypt(

        message.encode(),

        padding.OAEP(

            mgf=padding.MGF1(algorithm=hashes.SHA256()),

            algorithm=hashes.SHA256(),

            label=None

        )

    )

    return base64.b64encode(encrypted).decode()

# Decrypt a message using private key

def decrypt\_message(private\_key, encrypted\_message):

    decrypted = private\_key.decrypt(

        base64.b64decode(encrypted\_message),

        padding.OAEP(

            mgf=padding.MGF1(algorithm=hashes.SHA256()),

            algorithm=hashes.SHA256(),

            label=None

        )

    )

    return decrypted.decode()

# Simulating two users: Alice and Bob

print("Generating RSA key pairs for Alice and Bob...")

alice\_private, alice\_public = generate\_rsa\_key\_pair()

bob\_private, bob\_public = generate\_rsa\_key\_pair()

# Alice sends a message to Bob

message\_from\_alice = "Hi Bob, this is Alice. The message is secure!"

print("\nOriginal message from Alice:", message\_from\_alice)

encrypted\_message = encrypt\_message(bob\_public, message\_from\_alice)

print("Encrypted message (sent to Bob):", encrypted\_message)

# Bob decrypts the message

decrypted\_message = decrypt\_message(bob\_private, encrypted\_message)

print("Decrypted message by Bob:", decrypted\_message)

**Output:**

Requirement already satisfied: cryptography in /usr/local/lib/python3.11/dist-packages (43.0.3)

Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography) (1.17.1)

Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography) (2.22)

Generating RSA key pairs for Alice and Bob...

Original message from Alice: Hi Bob, this is Alice. The message is secure!

Encrypted message (sent to Bob): KGlFiU+Y9e1x0C0prVI1CLMr0hHQZqYygLv/EcXM2Sp8vSBeG7bgcK0u4/+UgpT+UmKvdOaH+NC/pIemG4yPINiaYx3G+Uh9GTTHJDpDWvCxdqjEDvdYTJVQE2uKxgvLLPNcOhenKxogCriQqG81biZnofxXpbq225k9UFlIVvKTcdAb52PSTsPTOy6vWso+HrkhmnIAHgbd0E/FJVh68KuiIQdZ8mmu1U7E9qqOOGot6qPsQZAdL0sHyc3Kl9kgLSLerZ3lwn3KcUFekb5hbXyqOCI3jgxP7L8a1zuDvusWxhThy1/+T194y9+Mj7m2haX3iOONXTfXx1B3DbvB8Q==

Decrypted message by Bob: Hi Bob, this is Alice. The message is secure!

**Practical 2**

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

**Code:**

!pip install pandas

import pandas as pd

transactions = []

def create\_transaction(sender, receiver, amount, description=""):

    transaction = {

        "Sender": sender,

        "Receiver": receiver,

        "Amount": amount,

        "Description": description

    }

    transactions.append(transaction)

    print("✅ Transaction recorded successfully!")

def display\_transactions():

    if transactions:

        df = pd.DataFrame(transactions)

        print("\n📄 All Transactions:")

        display(df)

    else:

        print("⚠️ No transactions found.")

# Example usage

create\_transaction("Alice", "Bob", 150, "Payment for services")

create\_transaction("Bob", "Charlie", 75, "Dinner split")

create\_transaction("Charlie", "Alice", 25, "Coffee refund")

# Display all transactions

display\_transactions()

**Output:**

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)

Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)

Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)

Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)

Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)

✅ Transaction recorded successfully!

✅ Transaction recorded successfully!

✅ Transaction recorded successfully!

📄 All Transactions:

|  | **Sender** | **Receiver** | **Amount** | **Description** |
| --- | --- | --- | --- | --- |
| **0** | Alice | Bob | 150 | Payment for services |
| **1** | Bob | Charlie | 75 | Dinner split |
| **2** | Charlie | Alice | 25 | Coffee refund |

**Practical 3**

**Aim:- 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:**

# Simulate user accounts with balances

accounts = {

    "Alice": 500,

    "Bob": 300,

    "Charlie": 200

}

# Transaction class definition

class Transaction:

    def \_\_init\_\_(self, sender, receiver, amount):

        self.sender = sender

        self.receiver = receiver

        self.amount = amount

    def transfer(self):

        # Check if users exist

        if self.sender not in accounts or self.receiver not in accounts:

            print("❌ Either sender or receiver account doesn't exist.")

            return

        # Check for sufficient funds

        if accounts[self.sender] < self.amount:

            print(f"❌ Insufficient funds in {self.sender}'s account.")

            return

        # Perform transfer

        accounts[self.sender] -= self.amount

        accounts[self.receiver] += self.amount

        print(f"✅ {self.amount} transferred from {self.sender} to {self.receiver}.")

# Example usage

print("💰 Initial account balances:", accounts)

# Create transactions

t1 = Transaction("Alice", "Bob", 100)

t1.transfer()

t2 = Transaction("Bob", "Charlie", 50)

t2.transfer()

t3 = Transaction("Charlie", "Alice", 300)  # This should fail due to insufficient funds

t3.transfer()

print("\n💼 Final account balances:", accounts)

**Output:**

💰 Initial account balances: {'Alice': 500, 'Bob': 300, 'Charlie': 200}

✅ 100 transferred from Alice to Bob.

✅ 50 transferred from Bob to Charlie.

❌ Insufficient funds in Charlie's account.

💼 Final account balances: {'Alice': 400, 'Bob': 350, 'Charlie': 250}

**Practical 4**

**Aim:- Implement a function to add new blocks to the miner and dump the blockchain.**

**Code:**

import hashlib

import time

import json

# Block class

class Block:

    def \_\_init\_\_(self, index, previous\_hash, timestamp, data, nonce=0):

        self.index = index

        self.previous\_hash = previous\_hash

        self.timestamp = timestamp

        self.data = data

        self.nonce = nonce

        self.hash = self.calculate\_hash()

    def calculate\_hash(self):

        block\_string = f"{self.index}{self.previous\_hash}{self.timestamp}{json.dumps(self.data)}{self.nonce}"

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

# Blockchain class

class Blockchain:

    def \_\_init\_\_(self):

        self.chain = [self.create\_genesis\_block()]

        self.difficulty = 4  # Number of leading zeros in the hash

    def create\_genesis\_block(self):

        return Block(0, "0", time.time(), "Genesis Block")

    def get\_latest\_block(self):

        return self.chain[-1]

    def mine\_block(self, data):

        previous\_block = self.get\_latest\_block()

        index = previous\_block.index + 1

        timestamp = time.time()

        nonce = 0

        print(f"⛏️ Mining block #{index}...")

        new\_block = Block(index, previous\_block.hash, timestamp, data, nonce)

        while not new\_block.hash.startswith('0' \* self.difficulty):

            new\_block.nonce += 1

            new\_block.hash = new\_block.calculate\_hash()

        self.chain.append(new\_block)

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

    def dump\_chain(self):

        print("\n📦 Blockchain Dump:")

        for block in self.chain:

            print({

                'Index': block.index,

                'Previous Hash': block.previous\_hash,

                'Timestamp': time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(block.timestamp)),

                'Data': block.data,

                'Nonce': block.nonce,

                'Hash': block.hash

            })

# Create the blockchain

my\_blockchain = Blockchain()

# Add (mine) blocks

my\_blockchain.mine\_block({"sender": "Alice", "receiver": "Bob", "amount": 50})

my\_blockchain.mine\_block({"sender": "Bob", "receiver": "Charlie", "amount": 25})

my\_blockchain.mine\_block({"sender": "Charlie", "receiver": "Alice", "amount": 10})

# Dump the entire blockchain

my\_blockchain.dump\_chain()

**Outpput:**

⛏️ Mining block #1...

✅ Block #1 mined: 0000037cc25274fa244cc6e1af1fdd6afa8aa6fd901d5c3889e9372b88e1c9ac

⛏️ Mining block #2...

✅ Block #2 mined: 00002bae30c62e2b6c2194ad790c0988af3a787656194d1023ee8bb20bc4eb68

⛏️ Mining block #3...

✅ Block #3 mined: 00007bd9b89e93898220a22f0b1674089558dccaed8b0e4ea54c8217175fb424

📦 Blockchain Dump:

{'Index': 0, 'Previous Hash': '0', 'Timestamp': '2025-05-07 03:01:35', 'Data': 'Genesis Block', 'Nonce': 0, 'Hash': '5b0087d56d98b0a8caf97c28ee28ff434bc984c9708842361d5a0725a0b15fac'}

{'Index': 1, 'Previous Hash': '5b0087d56d98b0a8caf97c28ee28ff434bc984c9708842361d5a0725a0b15fac', 'Timestamp': '2025-05-07 03:01:35', 'Data': {'sender': 'Alice', 'receiver': 'Bob', 'amount': 50}, 'Nonce': 11545, 'Hash': '0000037cc25274fa244cc6e1af1fdd6afa8aa6fd901d5c3889e9372b88e1c9ac'}

{'Index': 2, 'Previous Hash': '0000037cc25274fa244cc6e1af1fdd6afa8aa6fd901d5c3889e9372b88e1c9ac', 'Timestamp': '2025-05-07 03:01:36', 'Data': {'sender': 'Bob', 'receiver': 'Charlie', 'amount': 25}, 'Nonce': 37227, 'Hash': '00002bae30c62e2b6c2194ad790c0988af3a787656194d1023ee8bb20bc4eb68'}

{'Index': 3, 'Previous Hash': '00002bae30c62e2b6c2194ad790c0988af3a787656194d1023ee8bb20bc4eb68', 'Timestamp': '2025-05-07 03:01:36', 'Data': {'sender': 'Charlie', 'receiver': 'Alice', 'amount': 10}, 'Nonce': 100290, 'Hash': '00007bd9b89e93898220a22f0b1674089558dccaed8b0e4ea54c8217175fb424'}

**Practical 5**

**Aim:- Write a python program to demonstrate mining.**

**Code:**

import hashlib

import time

# Define the mining function

def mine\_block(block\_data, difficulty):

    prefix\_str = '0' \* difficulty

    nonce = 0

    start\_time = time.time()

    print("⛏️ Starting mining...")

    while True:

        text = f"{block\_data}{nonce}"

        hash\_result = hashlib.sha256(text.encode()).hexdigest()

        if hash\_result.startswith(prefix\_str):

            end\_time = time.time()

            print(f"✅ Block mined successfully!")

            print(f"🔢 Nonce: {nonce}")

            print(f"🔐 Hash: {hash\_result}")

            print(f"⏱️ Time taken: {end\_time - start\_time:.2f} seconds")

            break

        nonce += 1

# Example usage

block\_data = "Alice pays Bob 10 BTC"

difficulty = 4  # Increase for higher difficulty

mine\_block(block\_data, difficulty)

**Output:**

⛏️ Starting mining...

✅ Block mined successfully!

🔢 Nonce: 2040

🔐 Hash: 000077330197cd1a3f60c19a7990fa2b8ee23911e7d378d6e37d7d5d11d10b21

⏱️ Time taken: 0.01 seconds

**Practical 6**

**Aim:- Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract FunctionTypesDemo {

    uint public count;

    string public message;

    constructor() {

        count = 0;

        message = "Hello, Ethereum!";

    }

    // Regular function: modifies the contract state

    function incrementCount() public {

        count += 1;

    }

    // View function: reads state but does not modify it

    function getCount() public view returns (uint) {

        return count;

    }

    // Pure function: does not read or modify state

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

        return a + b;

    }

    // Fallback function: called when no other function matches or when plain ether is sent

    fallback() external payable {

        message = "Fallback function called";

    }

    // Receive function: explicitly handles plain Ether transfers

receive() external payable {

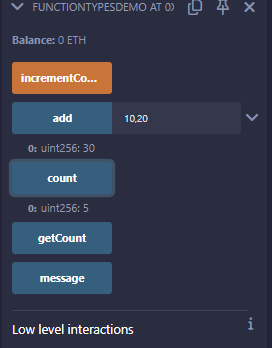
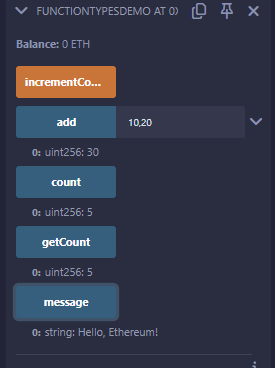
        message = "Receive function called";

    }

}

**Output**

****

**** ****

**Practical 7**

**Aim: Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract AdvancedFunctionDemo {

    // State variable

    uint public result;

    // --- Function Overloading ---

    // Overload 1: adds two numbers

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

        return a + b;

    }

    // Overload 2: adds three numbers

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

        return a + b + c;

    }

    // --- Mathematical Operations ---

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

        return a \* b;

    }

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

        require(b != 0, "Division by zero");

        return a / b;

    }

    function power(uint base, uint exponent) public pure returns (uint) {

        return base \*\* exponent;

    }

    // --- Cryptographic Hash Functions ---

    function getKeccak256(string memory input) public pure returns (bytes32) {

        return keccak256(abi.encodePacked(input));

    }

    function getSha256(string memory input) public pure returns (bytes32) {

        return sha256(abi.encodePacked(input));

    }

    function getRipemd160(string memory input) public pure returns (bytes20) {

        return ripemd160(abi.encodePacked(input));

    }

    // --- ecrecover Example (Signature Verification) ---

    // This recovers the signer address from a message and its signature

    function recoverSigner(bytes32 messageHash, uint8 v, bytes32 r, bytes32 s) public pure returns (address) {

        return ecrecover(messageHash, v, r, s);

    }

    // Utility function to hash a message in Ethereum style (with prefix)

    function getEthSignedMessageHash(string memory message) public pure returns (bytes32) {

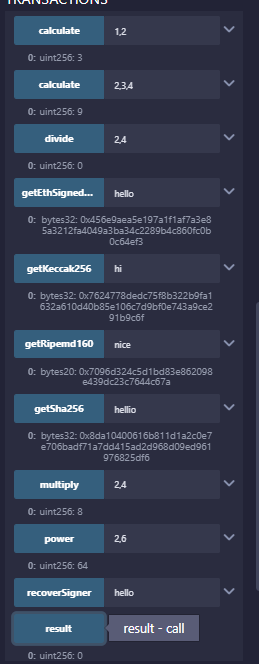
        bytes32 msgHash = keccak256(abi.encodePacked(message));

        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", msgHash));

    }

}

**Output:**

****

**Practical 8**

**Aim: Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// --------------------------------------

// Interface: Defines external interaction

// --------------------------------------

interface IVehicle {

    function startEngine() external pure returns (string memory);

}

// --------------------------------------

// Abstract Contract: Provides base logic

// --------------------------------------

abstract contract Machine {

    string public manufacturer;

    constructor(string memory \_manufacturer) {

        manufacturer = \_manufacturer;

    }

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

        return manufacturer;

    }

    // Abstract method to be implemented in derived contracts

    function operate() public view virtual returns (string memory);

}

// --------------------------------------

// Base Contract: Implements abstract and interface

// --------------------------------------

contract Car is Machine, IVehicle {

    string public model;

    constructor(string memory \_manufacturer, string memory \_model)

        Machine(\_manufacturer)

    {

        model = \_model;

    }

    // Implementing abstract method

    function operate() public view override returns (string memory) {

        return string(abi.encodePacked("Driving ", model));

    }

    // Implementing interface method

    function startEngine() public pure override returns (string memory) {

        return "Engine started";

    }

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

        return (manufacturer, model);

    }

}

// --------------------------------------

// Derived Contract: Inherits from Car

// --------------------------------------

contract ElectricCar is Car {

    uint public batteryLevel;

    constructor(

        string memory \_manufacturer,

        string memory \_model,

        uint \_batteryLevel

    ) Car(\_manufacturer, \_model) {

        batteryLevel = \_batteryLevel;

    }

    function recharge() public {

        batteryLevel = 100;

    }

    function getElectricCarInfo()

        public

        view

        returns (string memory, string memory, uint)

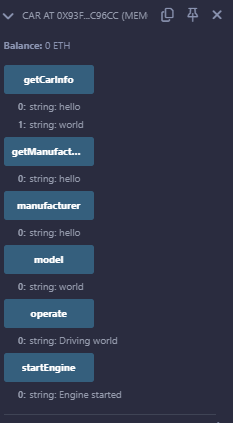
    {

        return (manufacturer, model, batteryLevel);

    }

}

**Output :**

****

**Practical 9**

**Aim: Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling.**

**Code:**

**// SPDX-License-Identifier: MIT**

**pragma solidity ^0.8.0;**

**// -----------------------------**

**// Library: Utility functions**

**// -----------------------------**

**library MathLib {**

**function square(uint x) internal pure returns (uint) {**

**return x \* x;**

**}**

**function cube(uint x) internal pure returns (uint) {**

**return x \* x \* x;**

**}**

**}**

**// -----------------------------**

**// Main Contract**

**// -----------------------------**

**contract AdvancedFeatures {**

**using MathLib for uint;**

**uint public result;**

**// -------------------------**

**// Events**

**// -------------------------**

**event Computed(string operation, uint value);**

**event ErrorHandled(string reason);**

**// -------------------------**

**// Custom Error**

**// -------------------------**

**error DivisionByZero();**

**error UnderflowError(uint a, uint b);**

**// -------------------------**

**// Function using a Library**

**// -------------------------**

**function computeSquare(uint x) public {**

**result = x.square();**

**emit Computed("square", result);**

**}**

**function computeCube(uint x) public {**

**result = x.cube();**

**emit Computed("cube", result);**

**}**

**// -------------------------**

**// Function with Inline Assembly**

**// -------------------------**

**function multiplyAssembly(uint a, uint b) public returns (uint product) {**

**assembly {**

**product := mul(a, b)**

**}**

**result = product;**

**emit Computed("assemblyMultiply", result);**

**}**

**// -------------------------**

**// Function with Error Handling**

**// -------------------------**

**function safeDivide(uint a, uint b) public returns (uint) {**

**if (b == 0) {**

**emit ErrorHandled("Division by zero attempted");**

**revert DivisionByZero();**

**}**

**result = a / b;**

**emit Computed("safeDivide", result);**

**return result;**

**}**

**function safeSubtract(uint a, uint b) public returns (uint) {**

**if (b > a) {**

**emit ErrorHandled("Underflow detected");**

**revert UnderflowError(a, b);**

**}**

**result = a - b;**

**emit Computed("safeSubtract", result);**

**return result;**

**}**

**}**

**Output :**

****