Parle Tilak Vidyalaya Associations

SATHAYE COLLEGE (Autonomous)

Vile-Parle (East), Mumbai – 400 057

Practical Journal

DEEP LEARNING

Submitted by

STUDENT NAME: DHIRAJ RAKESH PAWAR

Seat No.: 06

M.Sc. [I.T.]-Information Technology Part II

2024 – 2025

Parle Tilak Vidyalaya Association's

SATHAYE COLLEGE (Autonomous)

Vile-Parle (East), Mumbai – 400 057.

CERTIFICATE

This is to certify that DHIRAJ RAKESH PAWAR Seat No 06 has successfully completed all the practicals in the subject of BLOCKCHAIN for M.Sc.I.T. Part-II SEM – IV as prescribed by University of Mumbai for the year 2024-2025.

Coordinator	Professor in Charge M.Sc. [I.T.]	External Examiner
Date:	Date:	Date:

Index

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:

```
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+Uh9GTTH
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:

```
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:")
        print(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 (f
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from p
✅ 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()
```

Output:

```
⛏ 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': '5b0087d56d98b0a8caf97c28ee28ff434
{'Index': 1, 'Previous Hash': '5b0087d56d98b0a8caf97c28ee28ff434bc984c9708842361d5a0725a0b15fac', 'Timestamp': '2025-05-07 03:01:35', 'Data': {'sender
{'Index': 2, 'Previous Hash': '0000037cc25274fa244cc6e1af1fdd6afa8aa6fd901d5c3889e9372b88e1c9ac', 'Timestamp': '2025-05-07 03:01:36', 'Data': {'sender
{'Index': 3, 'Previous Hash': '00002bae30c62e2b6c2194ad790c0988af3a787656194d1023ee8bb20bc4eb68', 'Timestamp': '2025-05-07 03:01:36', 'Data': {'sender
```

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
```

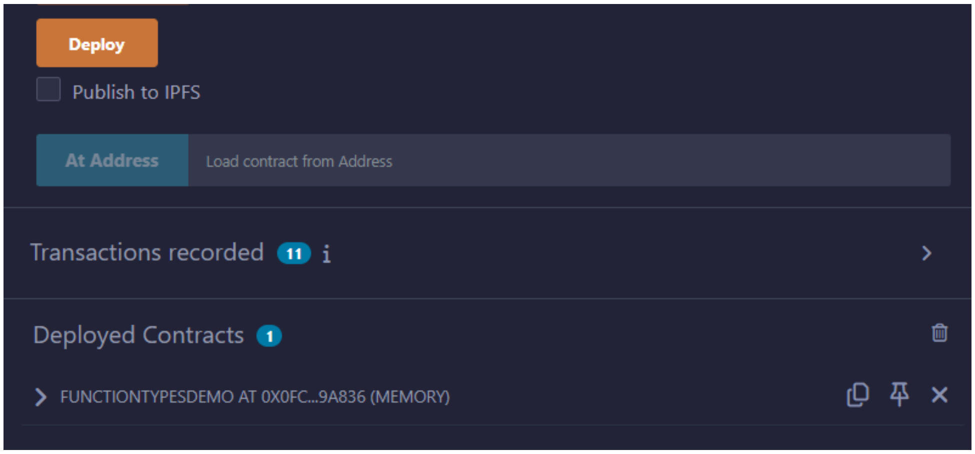
Practical – 6

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

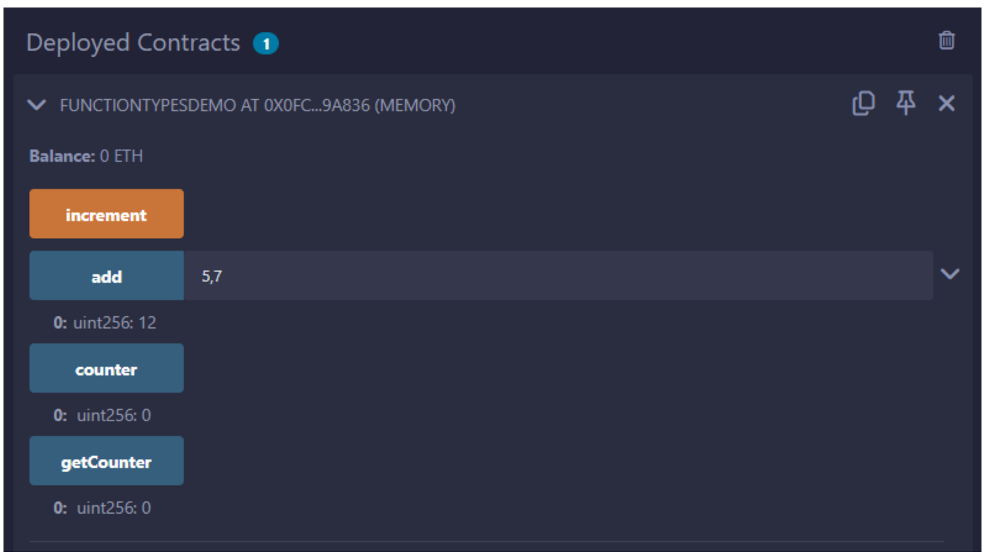
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract FunctionTypesDemo {

    uint public counter = 0;

    // 🔁 Regular function: modifies the state
    function increment() public {
        counter += 1;
    }

    // 👀 View function: reads state, doesn't modify
    function getCounter() public view returns (uint) {
        return counter;
    }

    // 🧮 Pure function: no access to state
    function add(uint a, uint b) public pure returns (uint) {
        return a + b;
    }

    // 💰 Receive function: triggered when contract receives Ether with no data
    receive() external payable {
        // You could add logic here for receiving ETH
    }

    // 🛠️ Fallback function: triggered when no other function matches
    fallback() external payable {
        // Fallback logic can go here
    }

}
```

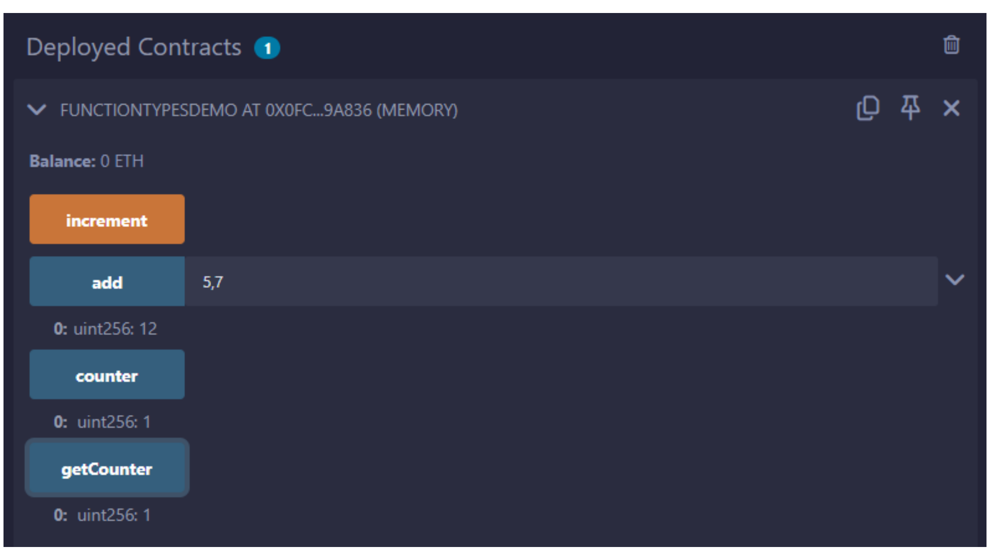
Output:



Before Increament



After Increament



Practical – 7

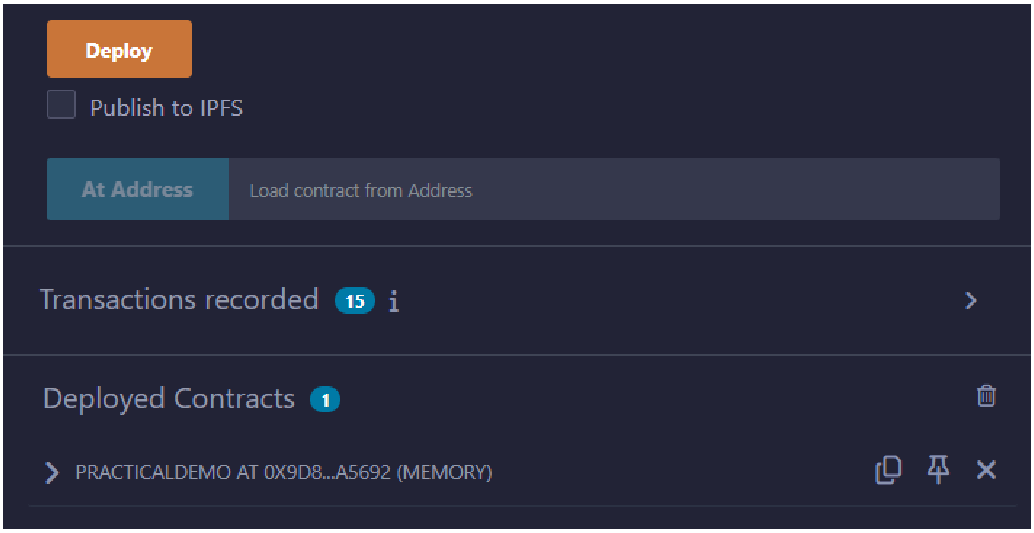
Aim: Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.

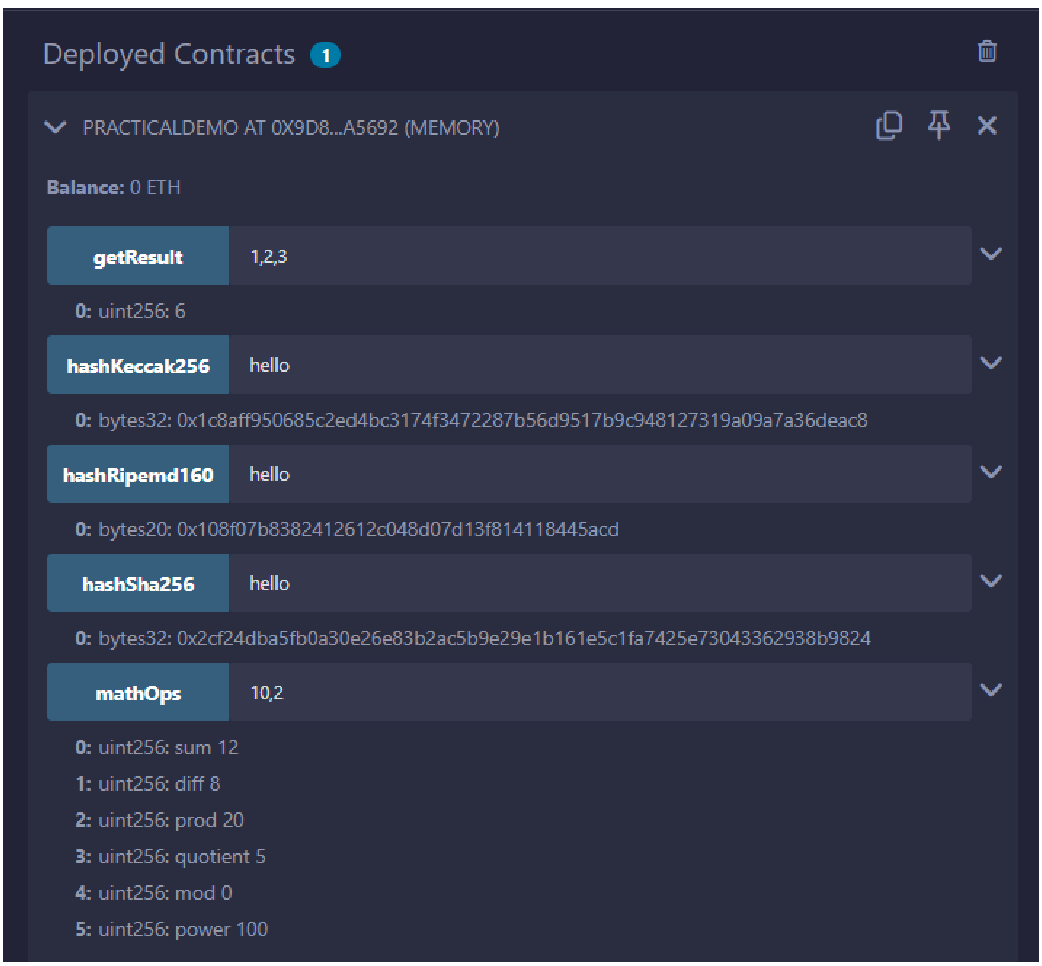
Code:

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

contract PracticalDemo {

    // 👉 Function Overloading Example
    function getResult(uint a, uint b, uint c) public pure returns (uint) {
        return a + b + c;
    }

    // 🧮 Mathematical Functions
    function mathOps(uint a, uint b) public pure returns (
        uint sum,
        uint diff,
        uint prod,
        uint quotient,
        uint mod,
        uint power
    ) {
        require(b != 0, "Division by zero"); // avoids divide-by-zero error
        sum = a + b;
        diff = a - b;
        prod = a * b;
        quotient = a / b;
        mod = a % b;
        power = a ** b;
    }

    // 🔐 Cryptographic Hash Functions

    function hashKeccak256(string memory input) public pure returns (bytes32) {
        return keccak256(abi.encodePacked(input));
    }

    function hashSha256(string memory input) public pure returns (bytes32) {
        return sha256(abi.encodePacked(input));
    }

    function hashRipemd160(string memory input) public pure returns (bytes20) {
        return ripemd160(abi.encodePacked(input));
```

```
    }
}
```

Output:


Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 15 i
Deployed Contracts 1
PRACTICALDEMO AT 0X9D8...A5692 (MEMORY)


Deployed Contracts 1
PRACTICALDEMO AT 0X9D8...A5692 (MEMORY)
Balance: 0 ETH
getResult
1,2,3
0: uint256: 6
hashKeccak256
hello
0: bytes32: 0x1c8aff950685c2ed4bc3174f3472287b56d9517b9c948127319a09a7a36deac8
hashRipemd160
hello
0: bytes20: 0x108f07b8382412612c048d07d13f814118445acd
hashSha256
hello
0: bytes32: 0x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
mathOps
10,2
0: uint256: sum 12
1: uint256: diff 8
2: uint256: prod 20
3: uint256: quotient 5
4: uint256: mod 0
5: uint256: power 100

Practical – 8

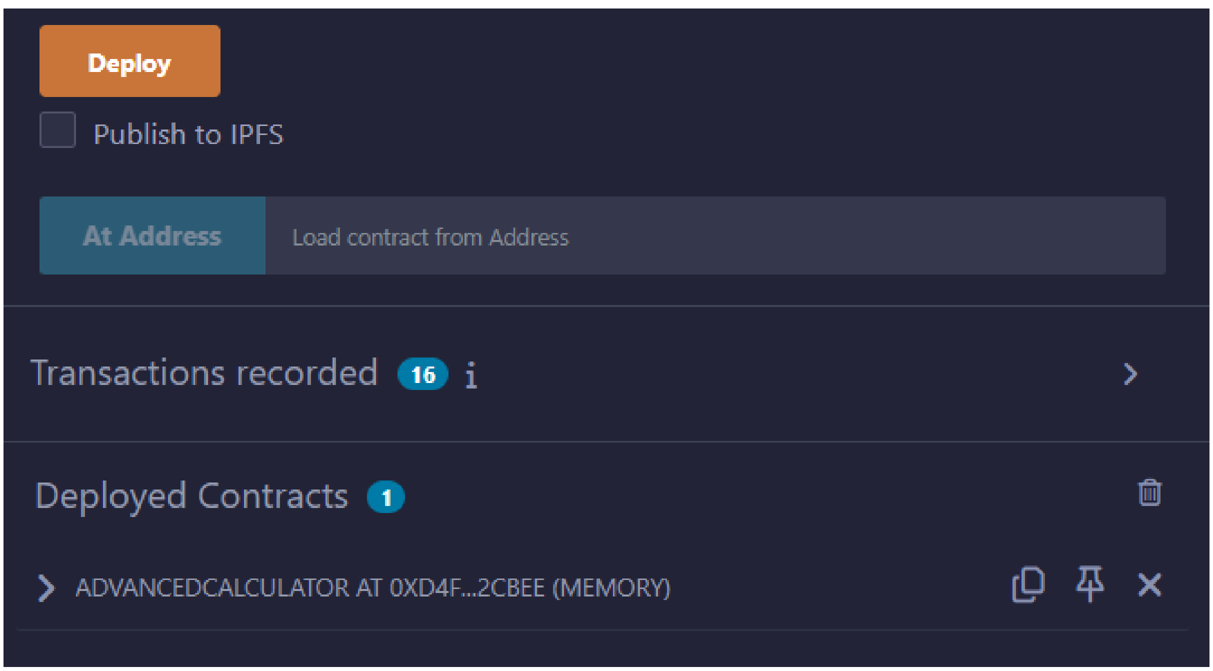
Aim: Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.

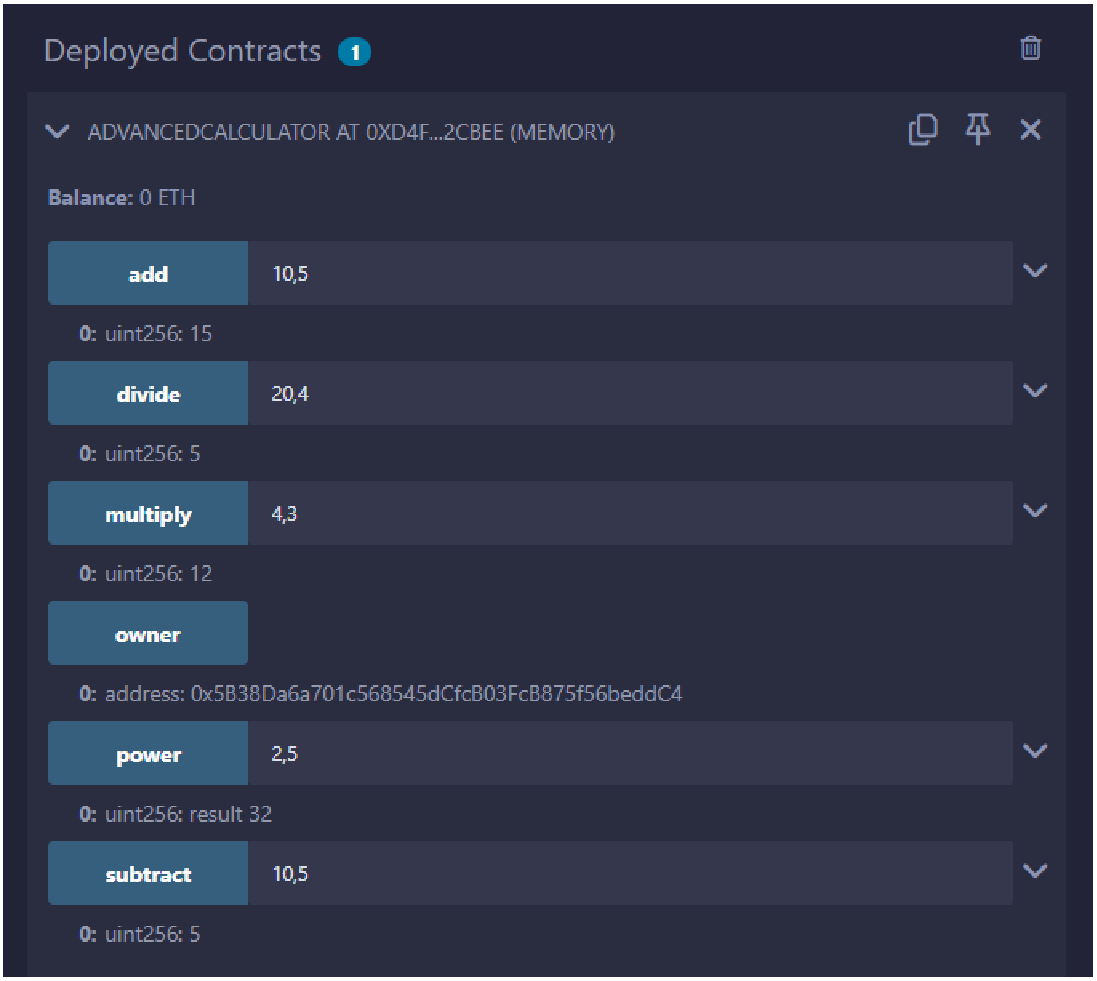
Code:

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

/// @title Interface Definition
interface ICalculator {
    function add(uint a, uint b) external pure returns (uint);
    function subtract(uint a, uint b) external pure returns (uint);
}

/// @title Abstract Contract for Shared Math Logic
abstract contract MathBase {
    // Abstract function (must be implemented)
    function multiply(uint a, uint b) public pure virtual returns (uint);

    // Concrete function (optional override)
    function divide(uint a, uint b) public pure returns (uint) {
        require(b != 0, "Cannot divide by zero");
        return a / b;
    }
}

/// @title Main Calculator Contract
/// @notice Implements interface and inherits from abstract base
contract Calculator is ICalculator, MathBase {
    address public owner;

    // Constructor
    constructor() {
        owner = msg.sender;
    }

    // Interface Implementations
    function add(uint a, uint b) external pure override returns (uint) {
        return a + b;
    }

    function subtract(uint a, uint b) external pure override returns (uint) {
        return a - b;
    }
```

```
    // Implementing abstract method
    function multiply(uint a, uint b) public pure override returns (uint) {
        return a * b;
    }

    // Inherited divide() method from MathBase is also available
}

/// @title Advanced Calculator Inheriting Calculator
contract AdvancedCalculator is Calculator {
    function power(uint base, uint exponent) public pure returns (uint result) {
        result = 1;
        for (uint i = 0; i < exponent; i++) {
            result *= base;
        }
    }
}
```

Output:




Deployed Contracts 1
ADVANCEDCALCULATOR AT 0XD4F...2CBEE (MEMORY)
Balance: 0 ETH
add
10,5
0: uint256: 15
divide
20,4
0: uint256: 5
multiply
4,3
0: uint256: 12
owner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
power
2,5
0: uint256: result 32
subtract
10,5
0: uint256: 5

Practical – 9

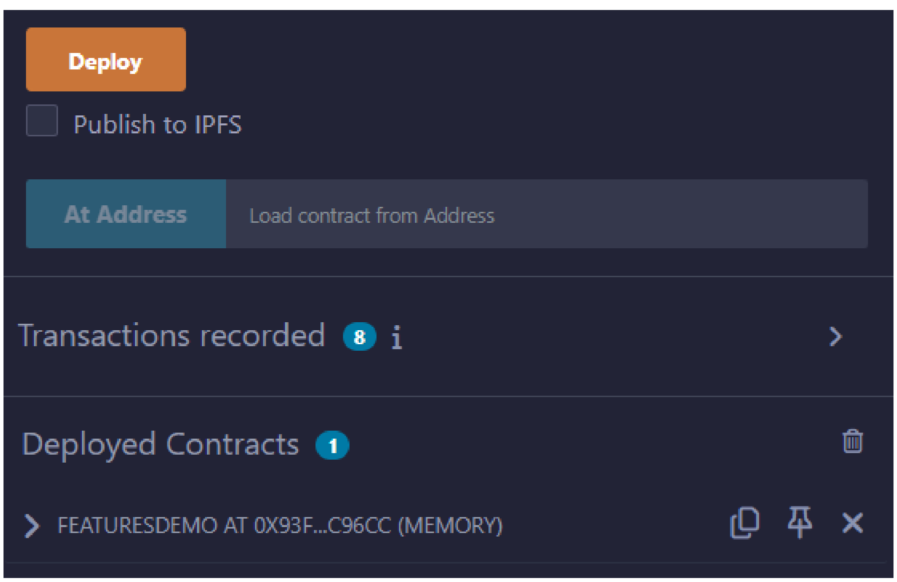
Aim: Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling.

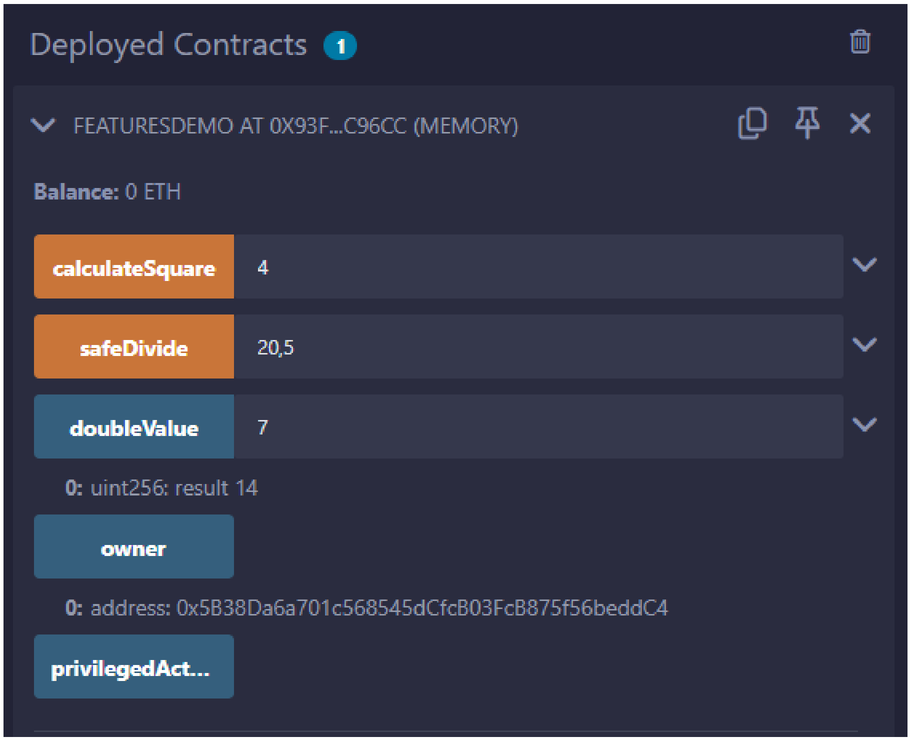
Code:

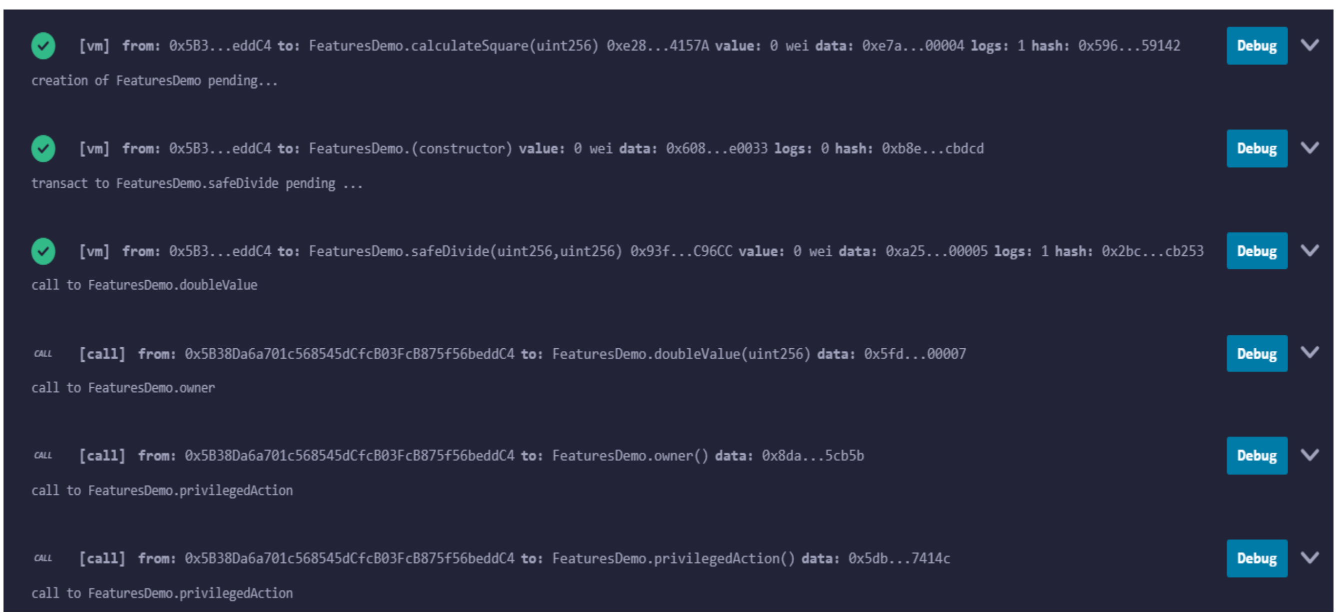
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/// @title MathLib - A library with reusable math 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;
    }
}

/// @title Contract demonstrating library, assembly, events, and error handling
contract FeaturesDemo {
    using MathLib for uint;

    address public owner;

    // Event to log computations
    event Computed(address indexed user, string operation, uint input, uint result);

    // Custom error for unauthorized access
    error NotOwner(address caller);

    constructor() {
        owner = msg.sender;
    }

    /// @notice Uses MathLib and emits event
    function calculateSquare(uint value) external returns (uint) {
        uint result = value.square();
        emit Computed(msg.sender, "square", value, result);
        return result;
    }

    /// @notice Uses inline assembly to double a value
    function doubleValue(uint x) public pure returns (uint result) {
        assembly {
```

```
            result := mul(x, 2)
        }
    }

    /// @notice Restricted to owner, demonstrates custom error
    function privilegedAction() external view {
        if (msg.sender != owner) {
            revert NotOwner(msg.sender);
        }
        // privileged logic here
    }

    /// @notice Demonstrates require, assert, and emit
    function safeDivide(uint a, uint b) external returns (uint result) {
        require(b != 0, "Division by zero");

        result = a / b;

        // assert is used for internal checks
        assert(result * b <= a); // This should always hold

        emit Computed(msg.sender, "divide", a, result);
    }
}
```

Output:




Deployed Contracts 1
FEATURESDEMO AT 0X93F...C96CC (MEMORY)
Balance: 0 ETH
calculateSquare
4
safeDivide
20,5
doubleValue
7
0: uint256: result 14
owner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
privilegedAct...


[vm] from: 0x5B3...eddC4 to: FeaturesDemo.calculateSquare(uint256) 0xe28...4157A value: 0 wei data: 0xe7a...00004 logs: 1 hash: 0x596...59142
Debug
creation of FeaturesDemo pending...
[vm] from: 0x5B3...eddC4 to: FeaturesDemo.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0xb8e...cbdcd
Debug
transact to FeaturesDemo.safeDivide pending ...
[vm] from: 0x5B3...eddC4 to: FeaturesDemo.safeDivide(uint256,uint256) 0x93f...C96CC value: 0 wei data: 0xa25...00005 logs: 1 hash: 0x2bc...cb253
Debug
call to FeaturesDemo.doubleValue
CALL
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: FeaturesDemo.doubleValue(uint256) data: 0x5fd...00007
Debug
call to FeaturesDemo.owner
CALL
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: FeaturesDemo.owner() data: 0x8da...5cb5b
Debug
call to FeaturesDemo.privilegedAction
CALL
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: FeaturesDemo.privilegedAction() data: 0x5db...7414c
Debug
call to FeaturesDemo.privilegedAction