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+UmK
vdOaH+NC/pIemG4yPINiaYx3G+Uh9GTTHJDpDWvCxdqjEDvdYTJVQE2uKxgvLLPNc
OhenKxogCriQqG81biZnofxXpbq225k9UFlIVvKTcdAb52PSTsPTOy6vWso+HrkhmnIAH
gbd0E/FJVh68KuiIQdZ8mmu1U7E9qqOOGot6qPsQZAdL0sHyc3Kl9kgLSLerZ3lwn3KcU
Fekb5hbXyqOCI3jgxP7L8a1zuDvusWxhThy1/+T194y9+Mj7m2haX3iOONXTfXx1B3Dbv
B8Q==
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

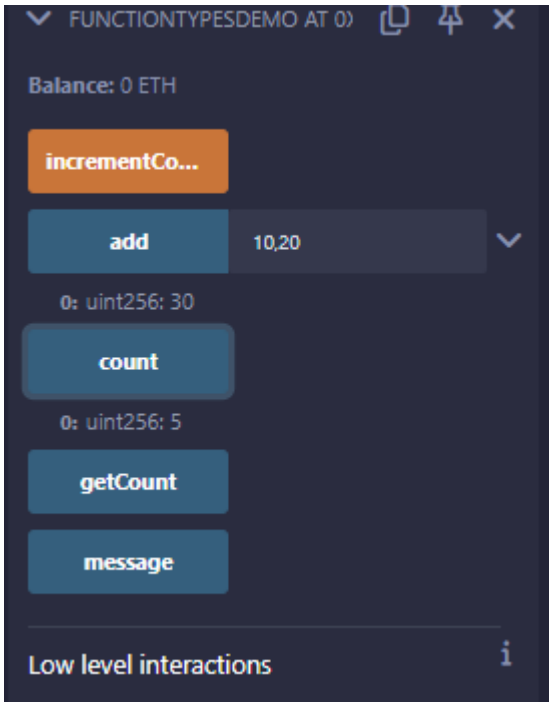
Aim:- Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.

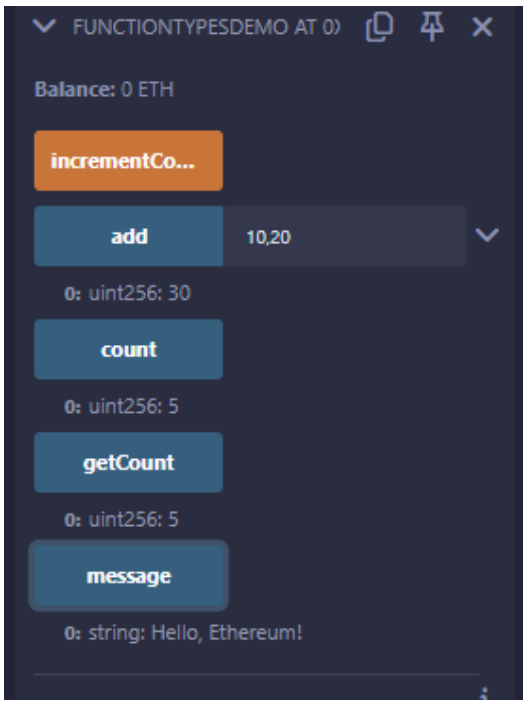
Code:

```
// 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

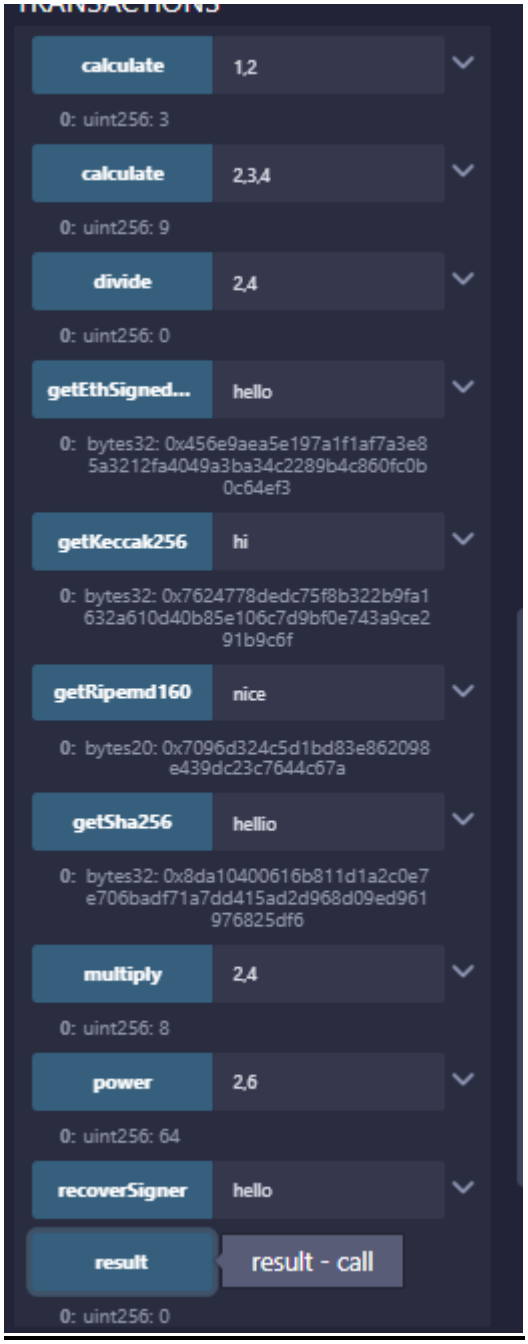
Aim: Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.

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:


calculate
1,2
0: uint256: 3
calculate
2,3,4
0: uint256: 9
divide
2,4
0: uint256: 0
getEthSigned...
hello
0: bytes32: 0x456e9aea5e197a1f1af7a3e85a3212fa4049a3ba34c2289b4c860fc0b0c64ef3
getKeccak256
hi
0: bytes32: 0x7624778dedc75f8b322b9fa1632a610d40b85e106c7d9bf0e743a9ce291b9c6f
getRipemd160
nice
0: bytes20: 0x7096d324c5d1bd83e862098e439dc23c7644c67a
getSha256
hellio
0: bytes32: 0x8da10400616b811d1a2c0e7e706badf71a7dd415ad2d968d09ed961976825df6
multiply
2,4
0: uint256: 8
power
2,6
0: uint256: 64
recoverSigner
hello
result
result - call
0: uint256: 0

Practical 8

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

Code:

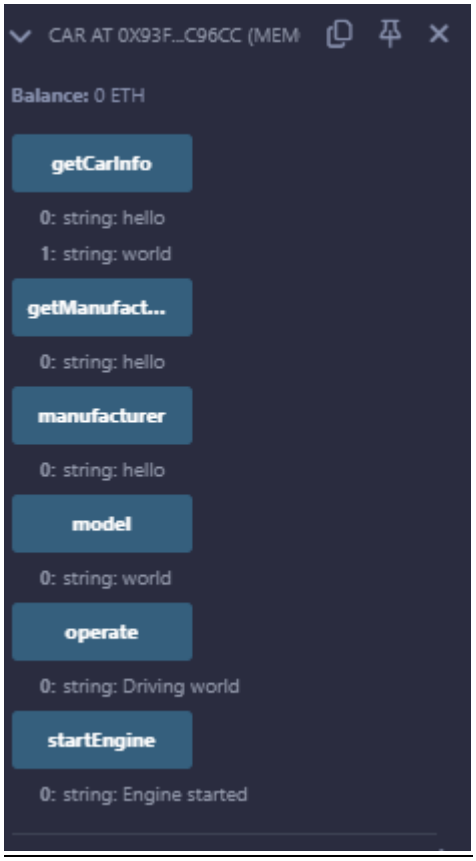
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
// ---------------------------------------
// Interface: Defines external interaction
// ---------------------------------------
interface IVehicle {
    function startEngine() external pure returns (string memory);
}
// ---------------------------------------
// 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 :