Practical 4

1. Withdrawal Pattern

**Code**

pragma solidity 0.8.26;

contract WithdrawalPattern {

    address public owner;

    uint256 public lockedbalance;

    uint256 public withdrawablebalance;

    constructor() {

        owner = msg.sender;

    }

    modifier onlyowner() {

        require(msg.sender == owner, "Only the owner can call this function");

        \_;

    }

    function deposit(uint256 amount) public payable {

        require(amount > 0, "Amount must be greater than zero");

        lockedbalance += amount;

    }

    function withdraw(uint256 amount) public payable onlyowner {

        require(

            amount <= withdrawablebalance,

            "Insufficient withdrawable balance"

        );

        withdrawablebalance -= amount;

        payable(msg.sender).transfer(amount);

    }

    function unlock(uint256 amount) public onlyowner {

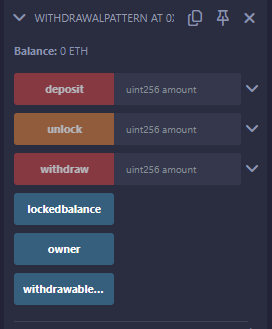
        require(amount <= lockedbalance, "Insufficient locked balance");

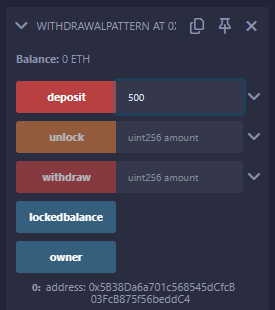
        lockedbalance -= amount;

        withdrawablebalance += amount;

    }

}



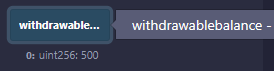




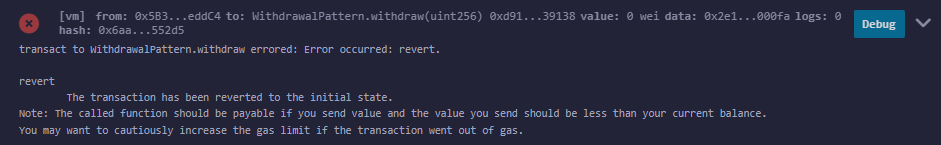












Restricted Access

**Code**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;

contract AccessRestriction {

    address public owner = msg.sender;

    uint256 public lastOwnerChange = block.timestamp;

    modifier onlyBy(address \_account) {

        require(msg.sender == \_account, "Not authorized");

        \_;

    }

    modifier onlyAfter(uint256 \_time) {

        require(block.timestamp >= \_time, "Function called too early");

        \_;

    }

    modifier costs(uint256 \_amount) {

        require(msg.value >= \_amount, "Insufficient value sent");

        \_;

        if (msg.value > \_amount) {

            (bool success, ) = msg.sender.call{value: msg.value - \_amount}("");

            require(success, "Refund failed");

        }

    }

    function changeOwner(address \_newOwner) public onlyBy(owner) {

        owner = \_newOwner;

    }

    function buyContract()

        public

        payable

        onlyAfter(lastOwnerChange + 4 weeks)

        costs(1 ether)

    {

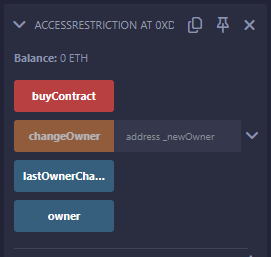
        owner = msg.sender;

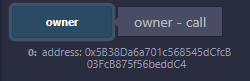
        lastOwnerChange = block.timestamp;

    }

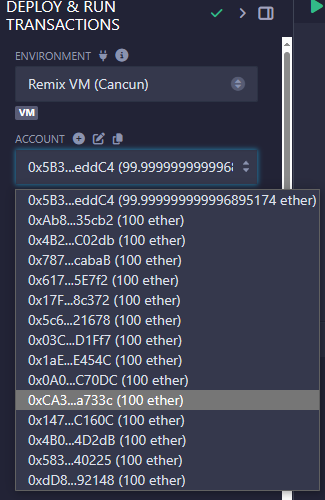
}

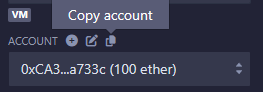
Output

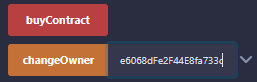


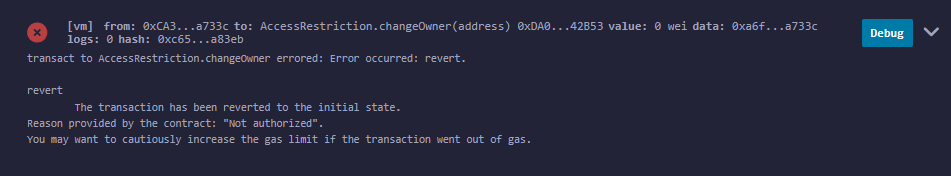


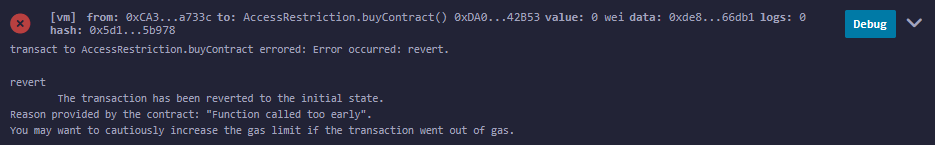


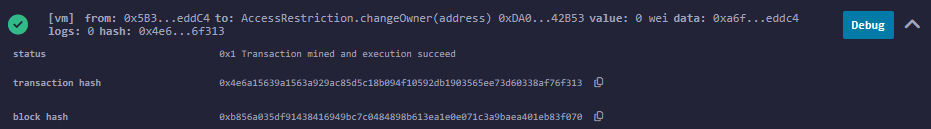


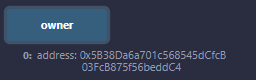












1. Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces
2. Contracts

**Code**

pragma solidity ^0.8.26;

contract Contract\_demo {

    string message = "Hello";

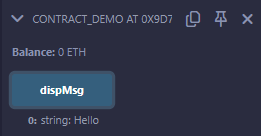
    function dispMsg() public view returns (string memory) {

        return message;

    }

}

**Output**

****

1. Inheritance

**Code**

pragma solidity 0.8.26;

contract Parent {

    uint256 internal sum;

    function setValue() external {

        uint256 a = 10;

        uint256 b = 20;

        sum = a + b;

    }

}

contract child is Parent {

    function getValue() external view returns (uint256) {

        return sum;

    }

}

contract caller {

    child cc = new child();

    function testInheritance() public returns (uint256) {

        cc.setValue();

        return cc.getValue();

    }

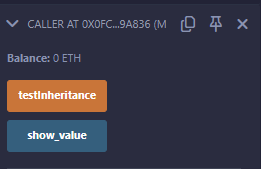
    function show\_value() public view returns (uint256) {

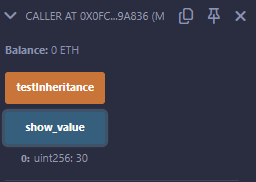
        return cc.getValue();

    }

}

**Output**

****

****

1. Constructors

**Code**

pragma solidity ^0.8.26;

// Creating a contract

contract constructorExample {

    string str;

    constructor() public {

        str = "Practical 4 (Use of Contructors)";

    }

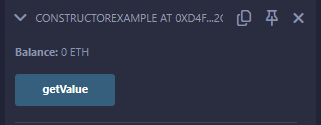
    function getValue() public view returns (string memory) {

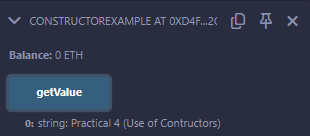
        return str;

    }

}

**Output**

****

****

1. Abstract Contracts

**Code**

pragma solidity ^0.8.26;

abstract contract Calculator {

    function getResult() external view virtual returns (uint256);

}

contract Test is Calculator {

    constructor() {}

    function getResult() external view override returns (uint256) {

        uint256 a = 15;

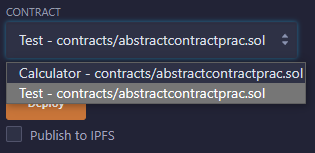
        uint256 b = 20;

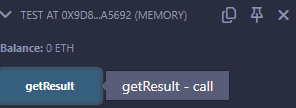
        uint256 result = a + b;

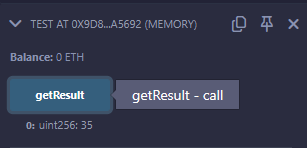
        return result;

    }

}



****

****

1. Interfaces

**Code**

pragma solidity ^0.8.26;

interface Calculator {

    function getResult() external view returns (uint256);

}

contract Test is Calculator {

    constructor() public {}

    function getResult() external view returns (uint256) {

        uint256 a = 20;

        uint256 b = 35;

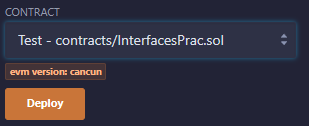
        uint256 result = a + b;

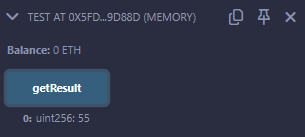
        return result;

    }

}

**Output**





1. Libraries, Assembly, Events, Error handling.
2. Libraries

Code

myLib.sol

pragma solidity 0.8.26;

library myMathLib {

    function sum(uint256 a, uint256 b) public pure returns (uint256) {

        return a + b;

    }

    function exponent(uint256 a, uint256 b) public pure returns (uint256) {

        return a\*\*b;

    }

}

using\_sollibrary.sol

pragma solidity >=0.7.0 <0.9.0;

import "contracts/myLib.sol";

contract UseLib {

    function getsum(uint256 x, uint256 y) public pure returns (uint256) {

        return myMathLib.sum(x, y);

    }

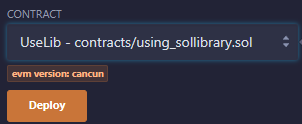
    function getexponent(uint256 x, uint256 y) public pure returns (uint256) {

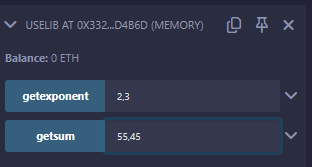
        return myMathLib.exponent(x, y);

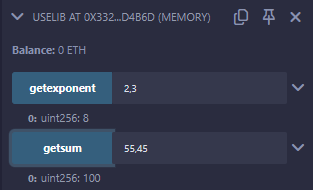
    }

}

Output







1. Assembly

Code

pragma solidity >=0.4.16 <0.9.0;

contract InlineAssembly {

    // Defining function

    function add(uint256 a) public view returns (uint256 b) {

        assembly {

            let c := add(a, 16)

            mstore(0x80, c)

            {

                let d := add(sload(c), 12)

                b := d

            }

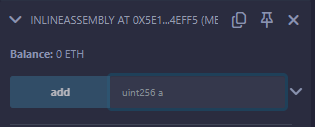
            b := add(b, c)

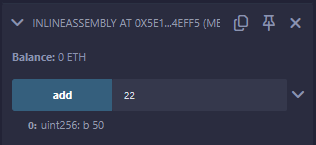
        }

    }

}

Output





1. Events

Code

pragma solidity ^0.8.26;

// Creating a contract

contract eventExample {

    // Declaring state variables

    uint256 public value = 0;

    // Declaring an event

    event Increment(address owner);

    // Defining a function for logging event

    function getValue(uint256 \_a, uint256 \_b) public {

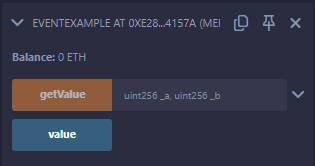
        emit Increment(msg.sender);

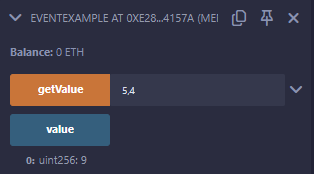
        value = \_a + \_b;

    }

}

Output







1. Error handling

Code

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;

contract ErrorDemo {

    function getSum(uint256 a, uint256 b) public pure returns (uint256) {

        uint256 sum = a + b;

        // require(sum < 255, "Invalid");

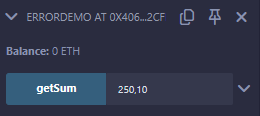
        assert(sum < 255);

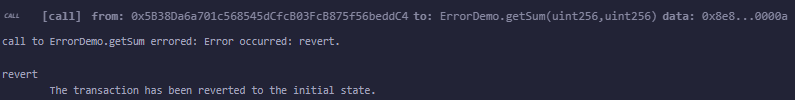
        return sum;

    }

}

Output





Practical 5

Write a program to demonstrate mining of Ether.

Code

from hashlib import sha256

import time

MAX\_NONCE = 100000000000

def hashGenerator(text):

return sha256(text.encode("ascii")).hexdigest()

def mine(block\_number, transactions, previous\_hash, prefix\_zeros):

prefix\_str = '0'\*prefix\_zeros

for nonce in range(MAX\_NONCE):

text = str(block\_number) + transactions + previous\_hash + str(nonce)

new\_hash = hashGenerator(text)

if new\_hash.startswith(prefix\_str):

print(f"Successfully mined Ethers with nonce value : {nonce}")

return new\_hash

raise BaseException(f"Couldn't find correct hash after trying {MAX\_NONCE} times")

if \_\_name\_\_ == '\_\_main\_\_':

transactions = '''

Jhon->Paul->77,

Akon->Bruno->18

'''

difficulty = 4

start = time.time()

print("Ether mining started.")

new\_hash = mine(5,transactions,'0000000xa036944e29568d0cff17edbe038f81208fecf9a66be9a2b8321c6ec7', difficulty)

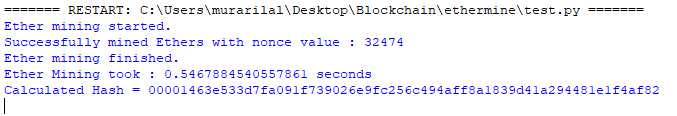
total\_time = str((time.time() - start))

print(f"Ether mining finished.")

print(f"Ether Mining took : {total\_time} seconds")

print(f"Calculated Hash = {new\_hash}")

Output



Practical 6

Create your own blockchain and demonstrate its use.

Code

from hashlib import sha256

def hashGenerator(text):

return sha256(text.encode("ascii")).hexdigest()

class Block:

def \_\_init\_\_(self,data,hash,prev\_hash):

self.data=data

self.hash=hash

self.prev\_hash=prev\_hash

class Blockchain:

def \_\_init\_\_(self):

hashLast=hashGenerator('gen\_last')

hashStart=hashGenerator('gen\_hash')

genesis=Block('gen-data',hashStart,hashLast)

self.chain=[genesis]

def add\_block(self,data):

prev\_hash=self.chain[-1].hash

hash=hashGenerator(data+prev\_hash)

block=Block(data,hash,prev\_hash)

self.chain.append(block)

bc=Blockchain()

bc.add\_block('1')

bc.add\_block('2')

bc.add\_block('3')

for block in bc.chain:

print(block.\_\_dict\_\_)

Output

