Program\_01.sol

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Hello\_World {

    string public hello = "Hello World!";

}

Program\_02.sol

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract datatype {

    bool public boolean = true;

    int32 public int\_32;

    int32 public int\_var = type(int32).min;

    uint64 public int\_64;

    uint256 public uint\_var = type(uint256).max;

    string public str = "String";

    int256 public a = 1000;

    int256 public b = 100;

    function add\_a\_b() public view returns (int256) {

        int256 int\_x = a + b;

        return int\_x;

    }

    function sub\_a\_b() public view returns (int256) {

        int256 int\_x = a - b;

        return int\_x;

    }

}

Practice.sol

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.9.0;

contract Practice {

    uint256[4] public arr = [10, 20, 30, 50];

    uint256[] public arr1;

    // byte array

    bytes3 public b3;

    bytes2 public b2;

    bytes public b1 = "abc";

    function pushElementB1() public {

        b1.push("d");

    }

    function getElementB1(uint256 i) public view returns (bytes1) {

        return b1[i];

    }

    function getLengthOfB1() public view returns (uint256) {

        return b1.length;

    }

    function setterBytes() public {

        b3 = "abc";

        b2 = "ab";

    }

    uint256 public ages = 10;

    uint256 public counts;

    // constructor

    constructor(uint256 new\_count) {

        counts = new\_count;

    }

    function setters(uint256 newAge) public {

        age = newAge;

    }

    function getValue() public pure returns (uint256) {

        uint256 roll = 100;

        return roll;

    }

    function setter(uint256 index, uint256 value) public {

        arr[index] = value;

    }

    function lengthArr() public view returns (uint256) {

        return arr.length;

    }

    function pushElement(uint256 item) public {

        arr1.push(item);

    }

    function popElement() public {

        arr1.pop();

    }

    bool public values;

    function check(uint256 a) public returns (bool) {

        if (a > 100) {

            values = true;

        } else {

            values = false;

        }

        return values;

    }

    enum user {

        allowed,

        not\_allowed,

        wait

    }

    user public u1 = user.allowed;

    uint256 public lottery = 1000;

    function owner() public {

        if (u1 == user.allowed) {

            lottery = 0;

        }

    }

    function changeOwner() public {

        u1 = user.not\_allowed;

    }

    uint256 public age = 10;

    uint256 public count;

    mapping(uint256 => string) public roll\_no;

    function setter(uint256 keys, string memory value) public {

        roll\_no[keys] = value;

    }

    struct Student {

        string name;

        uint256 class;

    }

    mapping(uint256 => Student) public data;

    function setterStudentData(

        uint256 roll,

        uint256 class,

        string memory name

    ) public {

        data[roll] = Student({class: class, name: name});

    }

}

contract A {

    function f1() public pure returns (uint256) {

        return 1;

    }

    function f2() private pure returns (uint256) {

        return 2;

    }

    function f3() internal pure returns (uint256) {

        uint256 x = f2();

        return x;

    }

    function f4() external pure returns (uint256) {

        return 4;

    }

}

contract C {

    A obj = new A();

    uint256 public cx = obj.f1();

}

Banking Application

Banking.sol

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.22 <0.9.0;

contract banking {

    mapping(address => uint256) public Account;

    mapping(address => bool) public userExists;

    function createAccount() public payable returns (string memory) {

        require(userExists[msg.sender] == false, "Account Already Exists");

        Account[msg.sender] = msg.value;

        userExists[msg.sender] = true;

        return "account created";

    }

    function deposit(uint256 amount) public payable returns (string memory) {

        require(userExists[msg.sender] == true, "Account is not created");

        require(amount > 0, "Value for deposit is Zero");

        Account[msg.sender] = Account[msg.sender] + amount;

        return "Deposited Succesfully";

    }

    function withdraw(uint256 amount) public payable returns (string memory) {

        require(

            Account[msg.sender] > amount,

            "Insufficeint balance in Bank account"

        );

        require(userExists[msg.sender] == true, "Account is not created");

        require(amount > 0, "Enter non-zero value for withdrawal");

        Account[msg.sender] = Account[msg.sender] - amount;

        msg.sender.transfer(amount);

        return "Withdrawal Succesful";

    }

    function TransferAmount(address payable userAddress, uint256 amount)

        public

        returns (string memory)

    {

        require(

            Account[msg.sender] > amount,

            "insufficient balance in Bank account"

        );

        require(userExists[msg.sender] == true, "Account is not created");

        require(

            userExists[userAddress] == true,

            "to Transfer account does not exists in bank accounts "

        );

        require(amount > 0, "Enter non-zero value for sending");

        Account[msg.sender] = Account[msg.sender] - amount;

        Account[userAddress] = Account[userAddress] + amount;

        return "Transfer successful";

    }

    function sendAmount(address payable toAddress, uint256 amount)

        public

        payable

        returns (string memory)

    {

        require(amount > 0, "Enter non-zero value for withdrawal");

        require(userExists[msg.sender] == true, "Account is not created");

        require(

            Account[msg.sender] > amount,

            "insufficient balance in Bank account"

        );

        Account[msg.sender] = Account[msg.sender] - amount;

        toAddress.transfer(amount);

        return "transfer success";

    }

    function AccountBalance() public view returns (uint256) {

        return Account[msg.sender];

    }

    function accountExist() public view returns (bool) {

        return userExists[msg.sender];

    }

}

Counter Application

Counter.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Counter {

    uint public count;

    // Function to get the current count

    function get() public view returns (uint) {

        return count;

    }

    // Function to increment count by 1

    function inc() public {

        count += 1;

    }

    // Function to decrement count by 1

    function dec() public {

        // This function will fail if count = 0

        count -= 1;

    }

}