**Write a smart contract on a test network, for Bank account of a customer for**

// following operations: Deposit money | Withdraw Money | Show balance

contract Bank{

mapping(address => uint) public user\_account;

mapping(address => bool) public user\_exist;

function create\_account() public payable returns(string memory){

require(user\_exist[msg.sender] == false, "Account Already created!");

user\_account[msg.sender] = msg.value;

user\_exist[msg.sender] = true;

return "Account created";

}

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

require(user\_exist[msg.sender] == true, "Account not created!");

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

user\_account[msg.sender] += amount;

return "Amount deposisted sucessfully";

}

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

require(user\_exist[msg.sender] == true, "Account not created!");

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

require(user\_account[msg.sender] >= amount, "Amount is greater than money deposisted");

user\_account[msg.sender] -= amount;

return "Amount withdrawn sucessfully";

}

function account\_balance() public view returns(uint){

return user\_account[msg.sender];

}

function account\_exists() public view returns(bool){

return user\_exist[msg.sender];

}

}

**Write a Program in solidity to create Student Data.**

//SPDX-License-Identifier: Unlicensed

pragma solidity ^0.8.0;

contract Student {

struct student {

uint256 prn;

string name;

string class;

string department;

}

uint256 PRN;

mapping(uint256 => student) studentMap;

function addStudent(

string memory name,

string memory class,

string memory department

) public {

PRN += 1;

studentMap[PRN] = student(PRN, name, class, department);

}

function getStudent(uint256 \_id) public view returns (student memory) {

return studentMap[\_id];

}

function totalStudents() public view returns (uint256) {

return (PRN);

}

fallback() external {

addStudent("Unknown", "FE", "CSE");

}

}

**Write a Program in Solidityy for addition, subtraction, multiplication and division.**// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleMath {

    // Function to add two numbers

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

        return a + b;

    }

    // Function to subtract two numbers

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

        require(a >= b, "Subtraction results in a negative value");

        return a - b;

    }

    // Function to multiply two numbers

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

        return a \* b;

    }

    // Function to divide two numbers

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

        require(b > 0, "Division by zero is not allowed");

        return a / b;

    }

}

**Write a Program using array in solidity:-**

1. **Accept and Display array marks**
2. **Find addition of marks in array**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MarksArray {

    // Declare an array to store student marks

    uint256[] public marks;

    // Function to accept and store an array of marks

    function setMarks(uint256[] memory \_marks) public {

        marks = \_marks;  // Store the array in the marks state variable

    }

    // Function to get the array of marks

    function getMarks() public view returns (uint256[] memory) {

        return marks;  // Return the marks array

    }

    // Function to calculate and return the sum of the marks

    function getMarksSum() public view returns (uint256) {

        uint256 sum = 0;

        // Iterate over the marks array to calculate the sum

        for (uint256 i = 0; i < marks.length; i++) {

            sum += marks[i];

        }

        return sum;  // Return the sum of the marks

    }

}

**Write a Program in solidity using Do-while loop accept marks of only pass students in array if mark>40 and only accept array size 10.**// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract PassMarks {

uint256[] public passMarks; // Array to store passing marks

uint256 public maxSize = 10; // Max size of the array (can store up to 10 marks)

uint256 public studentCount = 0; // Count of students added

// Function to accept individual marks and store them if they are greater than 40

function acceptMarks(uint256 \_mark) public {

// Check if the mark is greater than 40 and if the array has space

require(\_mark > 40, "Mark must be greater than 40.");

require(studentCount < maxSize, "Max number of students reached.");

// Add the mark to the array and increment the student count

passMarks.push(\_mark);

studentCount++;

}

// Function to retrieve the list of passing marks

function getPassMarks() public view returns (uint256[] memory) {

return passMarks;

}

// Function to get the current number of accepted students

function getPassStudentCount() public view returns (uint256) {

return studentCount;

}

}  
  
  
**Write a Program in Solidity to display Employee Details.**// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EmployeeDetails {

    // Define the Employee structure

    struct Employee {

        uint256 id;           // Employee ID

        string name;          // Employee Name

        uint256 salary;       // Employee Salary

        uint256 joiningDate;  // Joining Date (represented as a timestamp)

    }

    // Mapping to store employee details by employee ID

    mapping(uint256 => Employee) public employees;

    // Variable to keep track of the number of employees

    uint256 public employeeCount;

    // Function to add an employee

    function addEmployee(uint256 \_id, string memory \_name, uint256 \_salary, uint256 \_joiningDate) public {

        // Create a new Employee struct

        Employee memory newEmployee = Employee({

            id: \_id,

            name: \_name,

            salary: \_salary,

            joiningDate: \_joiningDate

        });

        // Store the employee details in the mapping

        employees[\_id] = newEmployee;

        // Increment the employee count

        employeeCount++;

    }

    // Function to get employee details by ID

    function getEmployee(uint256 \_id) public view returns (uint256, string memory, uint256, uint256) {

        // Retrieve the employee from the mapping

        Employee memory emp = employees[\_id];

        // Return the employee details

        return (emp.id, emp.name, emp.salary, emp.joiningDate);

    }

    // Function to get the total number of employees

    function getEmployeeCount() public view returns (uint256) {

        return employeeCount;

    }

}