**Solidity Assignment Fiza Sayyed**

**1)Hello World using solidity:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.7;

contract HelloWorld {

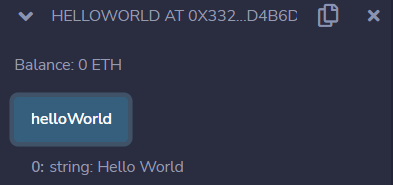
    function helloWorld() public pure returns (string memory) {

        return "Hello World";

    }

}

**OUTPUT:**

****

**2)Program to find Factorial of number via pure functions:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.7;

contract Factorial {

    function factorial(uint num) public pure returns (uint) {

         //num = number of which factorial is calculated

        uint fact=1;

        for(uint i=1; i<=num; i++)  //factorial logic: Ex: 5!=5\*4\*3\*2\*1 thus i = 1 will be incremented upto 5

        {

        fact=fact\*i;

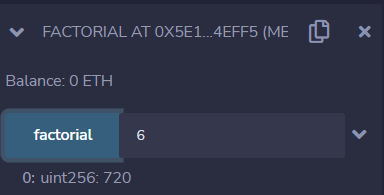
        }

        return fact;

    }

}

**OUTPUT:**

****

**3)Implementing decentralised voting system for 3 candidates, each voter can vote twice**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Election2 {

//defining structure with mutliple candidate variables

struct Candidate{

    uint id;

    string candidateName;

    uint voteCount;

}

//Giving references using mapping

mapping (address=>uint) public voters;

//0 , 1 =

mapping (uint=> Candidate)public candidates;

uint public candidatesCount;

function addCandidate(string memory \_name) private {

    candidatesCount++;

    candidates[candidatesCount]=Candidate(candidatesCount,\_name,0);

}

constructor() {

    addCandidate("Donald Trump");    //adding 3 candidates

    addCandidate("Joe Baiden");

    addCandidate("Kamala Harris");

}

event consolePrint( string, address);

function vote(uint \_candidateId) public{

   // Voter can vote twice :

    require(voters[msg.sender] <2); //msg.sender => person who has initiated smart contract

    require(\_candidateId>0 && \_candidateId<=candidatesCount); //correct set of candidates

    emit consolePrint("value of sender is ",msg.sender);

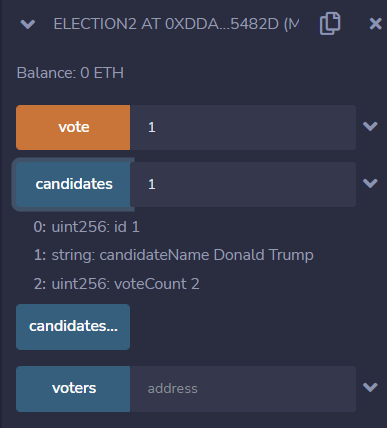
    voters[msg.sender] += 1;

    candidates[\_candidateId].voteCount++;

}

}

**OUTPUT:**

****

**4)Palindrome Program : to write a code to return palindrome of a string, if it is palindrome transfer 50 ETH from one account to manager account.**

**5)Write a contract 'Time' which implements a function named getTime:**

**For current Epoch & Unix timestamp used : https://www.epochconverter.com/**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Time{

    function getTime(uint time) public view returns(uint){

        //checking entered time exists after current time

        if(time>block.timestamp){

            return time+4830;  //1 hour, 20 minutes and 30 seconds = 3600 + 1200 + 20 seconds = 4830

        }

        else {

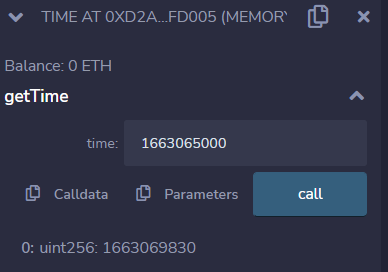
            return 0;

        }

    }

}

**OUTPUT:**



**6)Problem Statement**

Write a contract 'ThreeAndSeven' which implements a function named check.

check() accepts a number and return true if number is fully divided by 3 or 7 and greater than 10 else false. This function should not consume any gas.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract ThreeAndSeven{

    function check(int num) public pure returns (bool){

   //checking i fnumber is divisible by 3 and 7

    if(num%3==0 && num%7==0)  //using '&&' and operator to fulfill both conditions

    {

        if(num>10){

            return true;

        }

        else {

            return false;

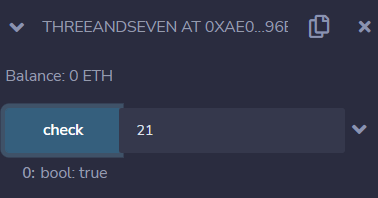
        }

    }

    }

}

OUTPUT:



**7)EvenOdd:** Problem Statement

Write a contract 'EvenOdd' to which implements a function named check.check() accepts a number and return whether the passed number is odd or even without consuming gas. (Ignore various checks on passed parameters for now)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract EvenOdd{

    function check(int num) public pure returns(string memory) {

        if(num % 2==0){

            return "The number is even number";

        }

        else {

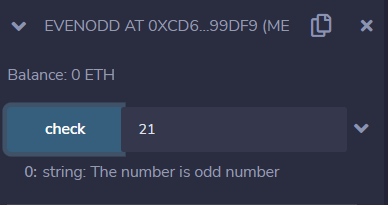
            return "The number is odd number";

        }

    }

}

**OUTPUT:**

****

**8)Problem Statement**

Write a contract 'Calculator' to which returns addition, subtraction, multiplication and division of two passed integers without consuming gas. (Ignore various checks on passed parameters for now)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract Calculator{

  //addition

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

        return a+b;

    }

    //subtraction

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

        return a-b;

    }

     //multiplication

    function multiply(uint a, uint b) public pure returns (uint)

    {

        return a\*b;

    }

     //division

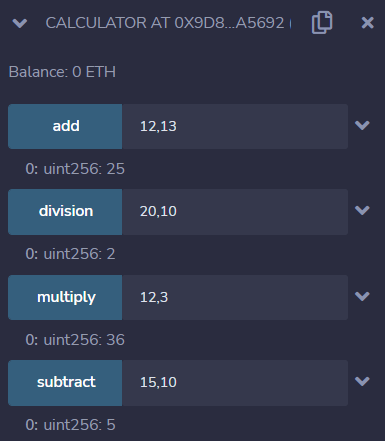
    function division(uint a, uint b) public pure returns (uint){

        return a/b;

    }

}

**OUTPUT:**

****

**9)Program to find Sum of Digits:**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.7;

contract SumofDigits{

function digitSum(int n) public pure returns (int) {

        int a;

        int sum = 0;

        while (n > 0) {

            a = n % 10;    //ex: n = 135 135%10= 5, a=5 similar for all digits

            sum = sum + a;

            n = n / 10;

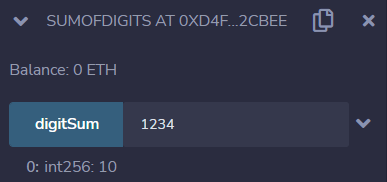
        }

        return sum;

    }

}

**OUTPUT:**

****

**10)Problem Statement**

Write a contract named "AttendanceRegister" which will be deployed by teacher. There will be a function add which will take student name, class & joiningDate and will store it where:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

struct student{

        string Name;

        uint Class;

        string JoiningDate;

    }

contract AttendanceRegister{

    address public Teacher = msg.sender;

    mapping (uint => student) public data;

    event Register(address Teacher, student Data);

    modifier onlyTeacher() {

        require(Teacher == msg.sender, "You are not a teacher");

        \_;

    }

    function add(uint check, string memory name, uint class, string memory date) public onlyTeacher {

        data[check] = student(name,class,date);

        emit Register(msg.sender, data[check]);

    }

}

OUTPUT:

