# Assignment – 1

# for

**BLOCKCHAIN TECHNOLOGY**

**UEC635**

## Submitted By: Vishav Singla Roll number: 102115204 Group: 3NC8

**Submitted to**

# Dr. Shashikant

# Assistant Professor



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB**

**INDIA**

**January – May**

**2024**

**1. Create a file in solidity to declare variables of different data types and arrays (fixed dynamic) and use a function to get their values.**

**CODE:**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

contract DataTypes {

    uint8 public u8 = 8;

    uint public u256 = 256;

    int256 public i256 = -256;

    bool public b = true;

    string public str = "abc";

    bytes32 public b32bytes = "test";

    uint[] public arr1 = [1, 2, 3];

    uint[10] public arr2 = [3, 4, 5, 6, 7];

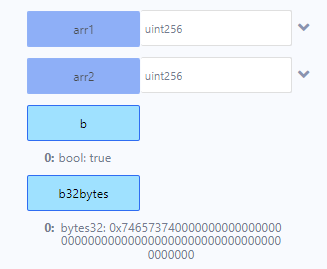
    function getValues() public view returns(uint8, uint, int256, bool, string memory, bytes32, uint[] memory, uint[10] memory) {

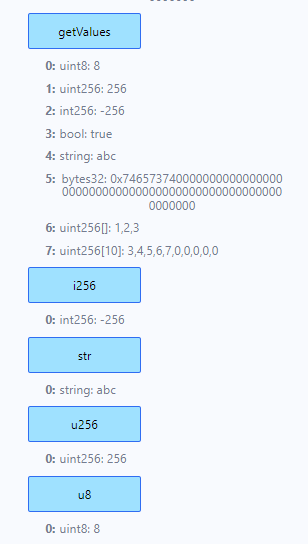
        return (u8, u256, i256, b, str, b32bytes, arr1, arr2);

    }

}

**OUTPUT:**

****

****

**2. Create a file in solidity to declare functions and experiment with its scope as**

**(public/private, pure/view and returns/no-returns**

**CODE:**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

contract FunctionScope {

    uint public num;

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

        return "Public function";

    }

    function privateFn() private pure returns(string memory) {

        return "Private function";

    }

    function viewFn() public view returns(uint) {

        return num;

    }

    function pureFn() public pure returns(uint) {

        return 1;

    }

    function setNum(uint \_num) public {

        num = \_num;

    }

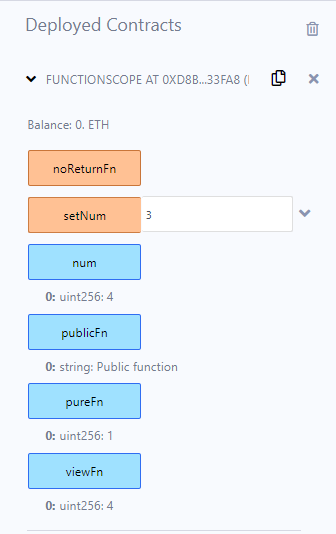
    function noReturnFn() public {

        num += 1;

    }

}

**OUTPUT:**



**3. Write Smart contracts to perform STACK and QUEUE operations in solidity.**

**CODE:**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

contract Stack {

    uint[] public stack;

    function push(uint x) public {

        stack.push(x);

    }

    function pop() public {

        stack.pop();

    }

    function getSize() view public returns(uint) {

        return stack.length;

    }

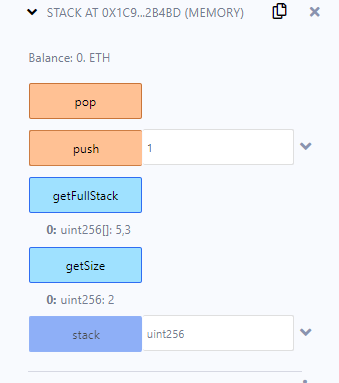
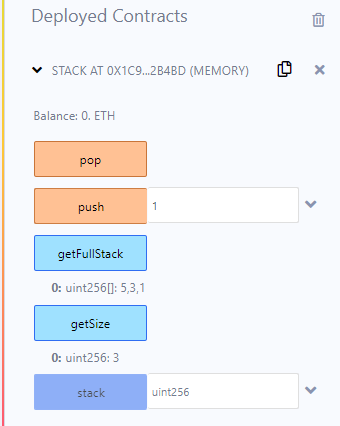
    function getFullStack() public view returns(uint[] memory) {

        return stack;

    }

}

**OUTPUT:**

****

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

contract Queue {

    uint[] public queue;

    function enqueue(uint x) public {

        queue.push(x);

    }

    function dequeue() public {

        require(queue.length > 0, "Queue is empty");

        for(uint i = 0; i < queue.length - 1; i++) {

            queue[i] = queue[i+1];

        }

        queue.pop();

    }

    function getSize() view public returns(uint) {

        return queue.length;

    }

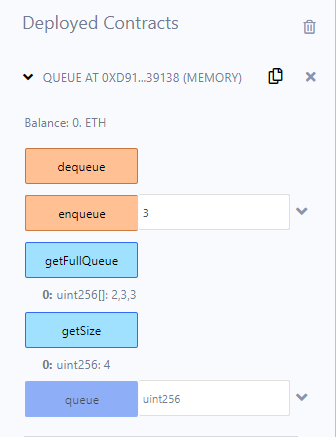
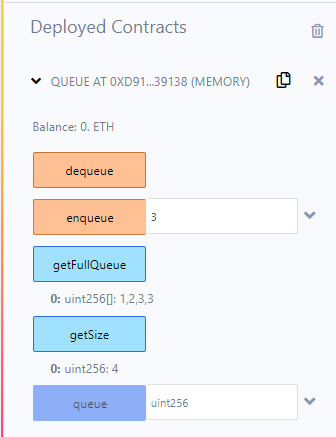
        function getFullQueue() public view returns(uint[] memory) {

        return queue;

    }

}

**OUTPUT:**

****

**4. Write different contracts in a single file with different functions to perform**

**multidimensional array operations.**

**CODE:**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

contract Contract1 {

    uint[2][3] public arr1;

    function setArr1(uint i, uint j, uint val) public {

        arr1[i][j] = val;

    }

    function getArr1Length() public view returns (uint) {

        return arr1.length;

    }

}

contract Contract2 {

    uint[3][2] public arr2;

    function setArr2(uint i, uint j, uint val) public {

        arr2[i][j] = val;

    }

    function sumArr2() public view returns (uint) {

      uint sum = 0;

      for (uint i = 0; i < arr2.length; i++) {

        for (uint j = 0; j < arr2[i].length; j++) {

          sum += arr2[i][j];

        }

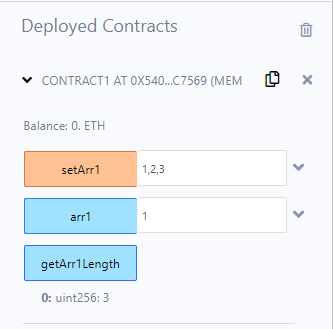
      }

      return sum;

    }

}

**OUTPUT:**



**5. . Write smart contracts in solidity and call a function from contract1 to contract2 to give input for solving quadratic equation and the computation need to done in function declared in different contract.**

**CODE:**

**QuadEquation.sol**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

contract QuadEquation {

    function solve(uint a, uint b, uint c) public pure returns (uint, uint) {

        uint temp = (b\*\*2) - (4\*a\*c);

        uint root1 = (2\*a) + sqrt(temp);

        uint root2 = (2\*a) - sqrt(temp);

        return (root1, root2);

    }

    function sqrt(uint x) public pure returns (uint y) {

        uint z = (x + 1) / 2;

        y = x;

        while (z < y) {

            y = z;

            z = (x / z + z) / 2;

        }

    }

}

**MainContract.sol**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

import "./QuadEquation.sol";

contract MainContract {

    QuadEquation quadEquation;

    constructor() {

        quadEquation = new QuadEquation();

    }

    function quadraticRoots(uint a, uint b, uint c) public view returns(uint x1, uint x2) {

        (x1, x2) = quadEquation.solve(a, b, c);

    }

}

**InputData.sol**

// SPDX-License-Identifier: GPL-3.0

//Vishav Singla - 102115204

pragma solidity ^0.8.0;

import "./MainContract.sol";

contract InputData {

    MainContract main;

    constructor() {

        main = new MainContract();

    }

    function getRoots(uint a, uint b, uint c) public view returns(uint, uint) {

        return main.quadraticRoots(a, b, c);

    }

}

**OUTPUT:**

