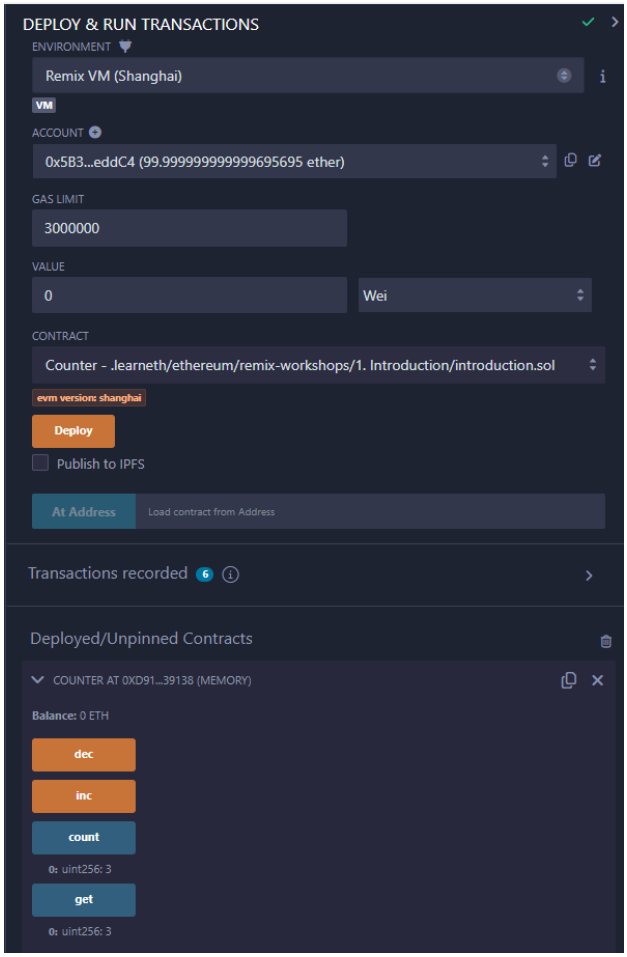
Blockchain Experiment 2

Shashwat Tripathi
Div: D15A INFT
Roll No: 64

AIM: Hands-on Solidity Programming Assignments for creating Smart Contracts

TASKS PERFORMED:
Go to LearnETH Tutorials provided by Remix IDE
Explore through the Solidity Basics Course
Complete all the 19 Assignments provided with the Course

PROGRAM & OUTPUT :
1. Introduction
Compile this contract.
Deploy it to the Remix VM.
Interact with your contract.



2. Basic Syntax
Delete the HelloWorld contract and its content.
Create a new contract named "MyContract".
The contract should have a public state variable called "name" of the type string.
Assign the value "Alice" to your new variable.

```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.3 and less
than 0.9.0
pragma solidity ^0.8.3;

contract MyContract {
    string public name = "Alice";
}
```

3. Primitive Data Types
Create a new variable newAddr that is a public address and give it a value that is not the same as the available variable addr.
Create a public variable called neg that is a negative number, decide upon the type.
Create a new variable, newU that has the smallest uint size type and the smallest uint value and is public.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Primitives {
    address public newAddr =
0x0000000000000000000000000000000000000000;
    int public neg = -7;
    uint8 public newU = 0;
}
```

4. Variables
Create a new public state variable called blockNumber.
Inside the function doSomething(), assign the value of the current block number to the state variable blockNumber.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Variables {
    uint256 public blockNumber;

    function doSomething() public {
        blockNumber = block.number;
```

```
    }
}
```

5.1 Functions - Reading and Writing to a State Variable
Create a public state variable called b that is of type bool and initialize it to true.
Create a public function called get_b that returns the value of b.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract SimpleStorage {
    bool public b = true;

    function get_b() public returns (bool) {
        return b;
    }
}
```

5.2 Functions - View and Pure
Create a function called addToX2 that takes the parameter y and updates the state variable x with the sum of the parameter and the state variable x.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract ViewAndPure {
    uint public x = 1;

    function addToX2(uint y) public {
        x = x + y;
    }
}
```

5.3 Functions - Modifiers and Constructors
Create a new function, increaseX in the contract. The function should take an input parameter of type uint and increase the value of the variable x by the value of the input parameter.
Make sure that x can only be increased.
The body of the function increaseX should be empty.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract FunctionModifier {
    address public owner;
    uint public x = 10;
```

```
    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }

    modifier biggerThan0(uint y) {
        require(y > 0, "Not bigger than x");
        _;
    }

    modifier increaseXbyY(uint y) {
        _;
        x = x + y;
    }

    function increaseX(uint y) public onlyOwner biggerThan0(y)
increaseXbyY(y){
    }
}
```

5.4 Functions - Inputs and Outputs

Create a new function called returnTwo that returns the values -2 and true without using a return statement.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Function {
    function returnTwo() public pure returns (int i, bool flag) {
        i = -2;
        flag = true;
    }
}
```

6. Visibilty

Create a new function in the Child contract called testInternalVar that returns the values of all state variables from the Base contract that are possible to return.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Base {
    string private privateVar = "my private variable";
    string internal internalVar = "my internal variable";
    string public publicVar = "my public variable";
}

contract Child is Base {
    function testInternalVar() public view returns (string memory,
string memory) {
        return (internalVar, publicVar);
    }
}
```

7.1 Control Flow - If/Else
Create a new function called evenCheck in the IfElse contract:

That takes in a uint as an argument.
The function returns true if the argument is even, and false if the argument is odd.
Use a ternery operator to return the result of the evenCheck function.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract IfElse {
    function evenCheck(uint num) public pure returns (bool) {
        return (num % 2) == 0;
    }
}
```

7.2 Control Flow - Loops
Create a public uint state variable called count in the Loop contract.
At the end of the for loop, increment the count variable by 1.
Try to get the count variable to be equal to 9, but make sure you don't edit the break statement.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Loop {
    uint public count;
    function loop() public{
        // for loop
        for (uint i = 0; i < 10; i++) {
```

```
            if (i == 5) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
            count++;
        }

        // while loop
        uint j;
        while (j < 10) {
            j++;
        }
    }
}
```

8.1 Data Structures - Arrays

Initialize a public fixed-sized array called arr3 with the values 0, 1, 2. Make the size as small as possible.

Change the getArr() function to return the value of arr3.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Array {
    uint[3] public arr3 = [0, 1, 2];

    function getArr() public view returns (uint[3] memory) {
        return arr3;
    }
}
```

8.2 Data Structures - Mappings

Create a public mapping balances that associates the key type address with the value type uint.

Change the functions get and remove to work with the mapping balances.

Change the function set to create a new entry to the balances mapping, where the key is the address of the parameter and the value is the balance associated with the address of the parameter.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
```

```
contract Mapping {
    // Mapping from address to uint
    mapping(address => uint) public balances;

    function get(address _addr) public view returns (uint) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default
value.
        return balances[_addr];
    }

    function set(address _addr) public {
        // Update the value at this address
        balances[_addr] = _addr.balance;
    }

    function remove(address _addr) public {
        // Reset the value to the default value.
        delete balances[_addr];
    }
}
```

8.3 Data Structures - Structs

Create a function remove that takes a uint as a parameter and deletes a struct member with the given index in the todos mapping.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Todos {
    struct Todo {
        string text;
        bool completed;
    }

    // An array of 'Todo' structs
    Todo[] public todos;

    function create(string memory _text) public {
        todos.push(Todo(_text, false));

        // key value mapping
        todos.push(Todo({text: _text, completed: false}));

        // initialize an empty struct and then update it
```

```
        Todo memory todo;
        todo.text = _text;
        // todo.completed initialized to false

        todos.push(todo);
    }

    function get(uint _index) public view returns (string memory
text, bool completed) {
        Todo storage todo = todos[_index];
        return (todo.text, todo.completed);
    }

    // update text
    function update(uint _index, string memory _text) public {
        Todo storage todo = todos[_index];
        todo.text = _text;
    }

    // update completed
    function toggleCompleted(uint _index) public {
        Todo storage todo = todos[_index];
        todo.completed = !todo.completed;
    }

    function remove(uint index) public {
        delete todos[index];
    }
}
```

8.4 Data Structures - Enums

Define an enum type called Size with the members S, M, and L.
Initialize the variable sizes of the enum type Size.
Create a getter function getSize() that returns the value of the variable sizes.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Enum {
    enum Size {
        S,
        M,
        L
    }
```

```
    Size public sizes;

    function getSize() public view returns (Size) {
        return sizes;
    }
}
```

9. Data Locations

Change the value of the myStruct member foo, inside the function f, to 4.
Create a new struct myMemStruct2 with the data location memory inside the function f and assign it the value of myMemStruct. Change the value of the myMemStruct2 member foo to 1.
Create a new struct myMemStruct3 with the data location memory inside the function f and assign it the value of myStruct. Change the value of the myMemStruct3 member foo to 3.
Let the function f return myStruct, myMemStruct2, and myMemStruct3.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract DataLocations {
    uint[] public arr;
    mapping(uint => address) map;
    struct MyStruct {
        uint foo;
    }
    mapping(uint => MyStruct) public myStructs;

    function f() public returns (MyStruct memory, MyStruct memory,
MyStruct memory){
        _f(arr, map, myStructs[1]);
        MyStruct storage myStruct = myStructs[1];
        myStruct.foo = 4;
        // create a struct in memory
        MyStruct memory myMemStruct = MyStruct(0);
        MyStruct memory myMemStruct2 = myMemStruct;
        myMemStruct2.foo = 1;

        MyStruct memory myMemStruct3 = myStruct;
        myMemStruct3.foo = 3;
        return (myStruct, myMemStruct2, myMemStruct3);
    }

    function _f(
        uint[] storage _arr,
        mapping(uint => address) storage _map,
        MyStruct storage _myStruct
```

```
    ) internal {
        // do something with storage variables
    }
}
```

10.1 Transactions - Ether and Wei
Create a public uint called oneGWei and set it to 1 gwei.
Create a public bool called isOneGWei and set it to the result of a comparison operation between 1 gwei and 10^9.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract EtherUnits {
    uint public oneGwei = 1 gwei;
    bool public isOneGwei = 1 gwei == 1e9;
}
```

10.2 Transactions - Gas and Gas Price
Create a new public state variable in the Gas contract called cost of the type uint. Store the value of the gas cost for deploying the contract in the new variable, including the cost for the value you are storing.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Gas {
    uint public i = 0;
    uint public cost = 170367;

    function forever() public {
        while (true) {
            i += 1;
        }
    }
}
```

10.3 Transactions - Sending Ether
Create a contract called Charity.
Add a public state variable called owner of the type address.
Create a donate function that is public and payable without any parameters or function code.
Create a withdraw function that is public and sends the total balance of the contract to the owner address.

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.3;

contract Charity {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function donate() public payable {}

    function withdraw() public {
        uint amount = address(this).balance;

        (bool sent, bytes memory data) = owner.call{value:
amount}("");
        require(sent, "Failed to send Ether");
    }
}
```

CONCLUSION:

Understood the basics of Solidity Programming in writing Smart Contracts and Deploying them on the Remix VM.

Successfully performed the Assignments given in the Tutorial.