**Blockchain Technology**

**Basic Solidity Smart Contracts – 2**

**Part-2**

**Aim:** Basic Solidity Smart contracts-2 Part-2

1. Mapping
2. Array
3. Structure
4. Require
5. Inheritance
6. Create Library
7. Use Library

**Description**:

1. **Solidity** is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum.
2. **Remix ide** is used to write and deploy the smart contract.
3. **MetaMask** is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications.
4. **Smart contracts** are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss.
5. **Mapping in Solidity acts like a hash table or dictionary in any other language.** These are used to **store the data in the form of key-value pairs**, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type.
6. **Array is a data structure**, which **stores a fixed-size sequential collection of elements of the same type**. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
7. **Structs** in Solidity **allows you to create more complicated data types that have multiple properties**. You can define your own type by creating a struct. They are useful for grouping together related data.
8. '**require**' **returns two boolean values that are either true or false**, if the specified condition returns a true value it allows the code to flow and function accordingly. If the value returned is false, it throws an error and stops the code right there.
9. **Inheritance** in the Solidity programming language **allows a programmer to extend a contractor's attributes and properties to their derived contracts**. Developers can also modify these aspects in the derived contract as well via a process known as overriding.
10. **Libraries** in solidity are **similar to contracts that contain reusable codes**. A library has functions that can be called by other contracts. Deploying a common code by creating a library reduces the gas cost.

**Code:**

1. Mapping

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract  Mapping{

    mapping(int => string) details;

    function addDetails(int id, string memory name) public{

        details[id] = name;

    }

    function updateDetails(int id, string memory name) public{

        details[id] = name;

    }

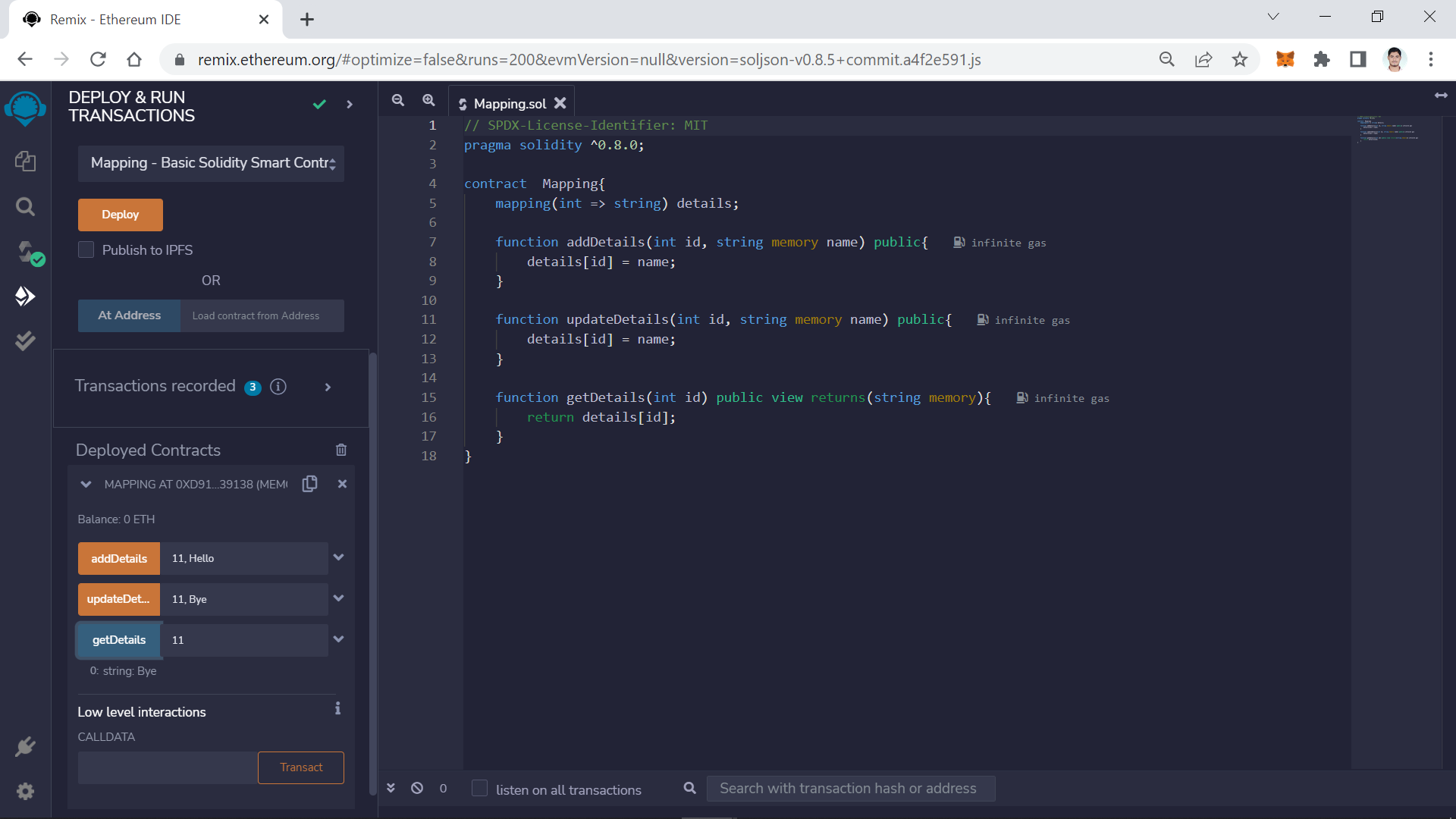
    function getDetails(int id) public view returns(string memory){

        return details[id];

    }

}

**Output:**



1. Array

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

contract Array {

    // Several ways to initialize an array

    uint[] public arr;

    uint[] public arr2 = [1, 2, 3];

    // Fixed sized array, all elements initialize to 0

    uint[10] public myFixedSizeArr;

    function get(uint i) public view returns (uint) {

        return arr[i];

    }

    // Solidity can return the entire array.

    // But this function should be avoided for

    // arrays that can grow indefinitely in length.

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

        return arr;

    }

    function push(uint i) public {

        // Append to array

        // This will increase the array length by 1.

        arr.push(i);

    }

    function pop() public {

        // Remove last element from array

        // This will decrease the array length by 1

        arr.pop();

    }

    function getLength() public view returns (uint) {

        return arr.length;

    }

    function remove(uint index) public {

        // Delete does not change the array length.

        // It resets the value at index to it's default value,

        // in this case 0

        delete arr[index];

    }

    function examples() pure external{

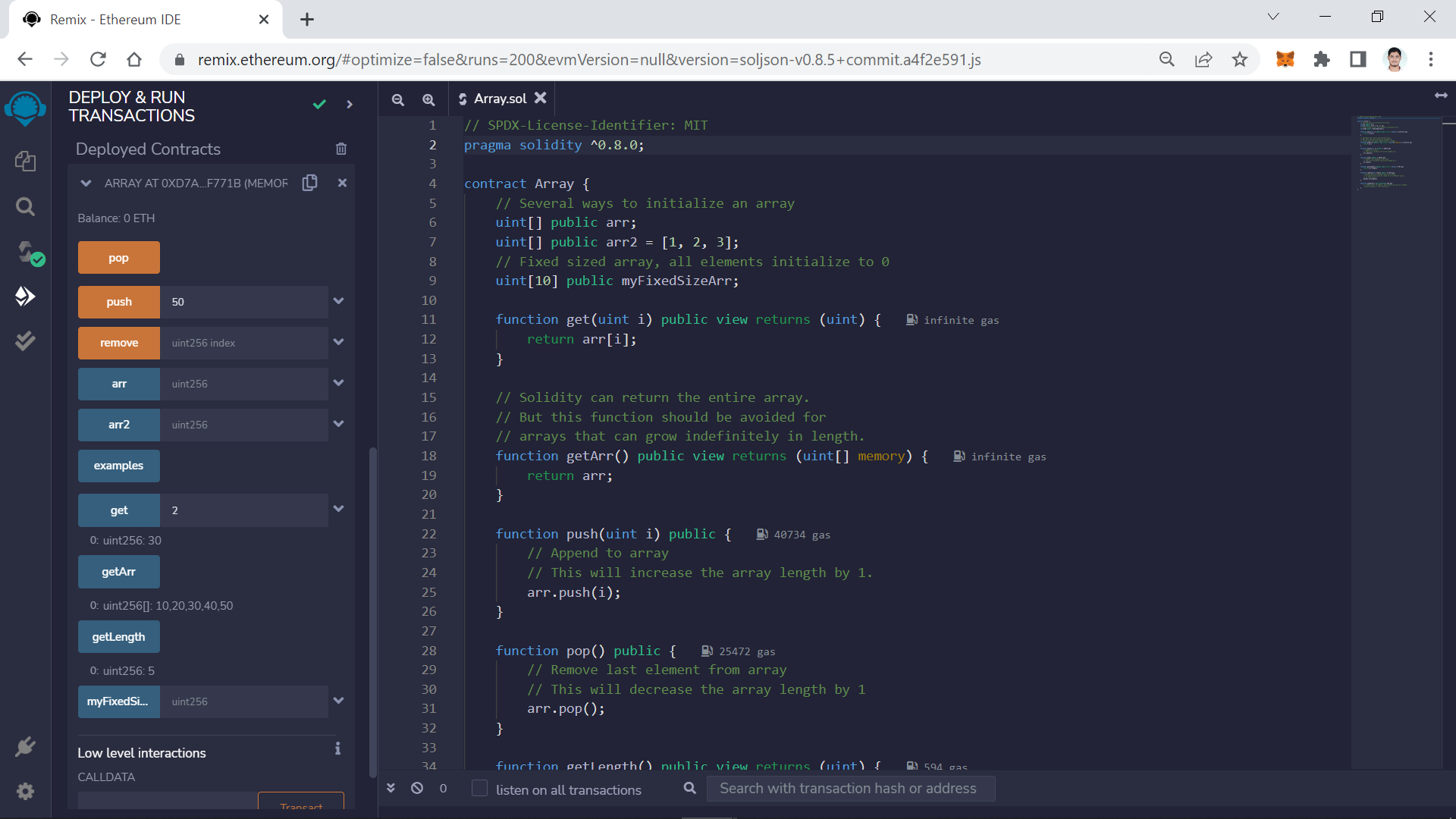
        // create array in memory, only fixed size can be created

        //uint[] memory a = new uint[](5);

    }

}

**Output:**



1. Structure

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

contract Todos {

    struct Todo {

        string text;

        bool completed;

    }

    // An array of 'Todo' structs

    Todo[] public todos;

    function create(string calldata \_text) public {

        // 3 ways to initialize a struct

        // - calling it like a function

        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);

    }

    // Solidity automatically created a getter for 'todos' so

    // you don't actually need this function.

    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 updateText(uint \_index, string calldata \_text) public {

        Todo storage todo = todos[\_index];

        todo.text = \_text;

    }

    // update completed

    function toggleCompleted(uint \_index) public {

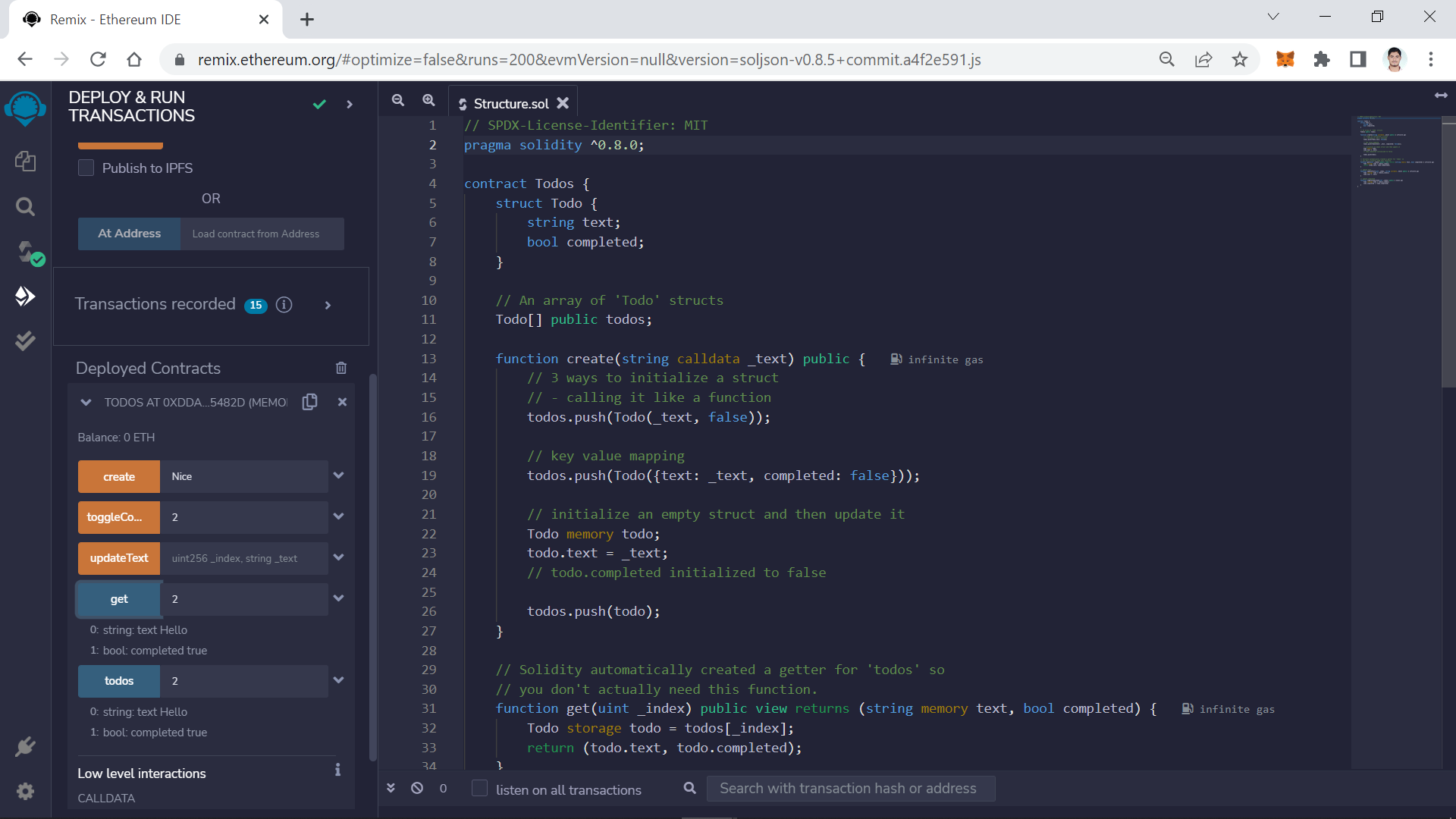
        Todo storage todo = todos[\_index];

        todo.completed = !todo.completed;

    }

}

**Output**



1. Require

//SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract requireExample{

    uint256 input;

    address sender;

    function some\_state\_changing\_fn (uint256 \_input) public returns (bool success)

    {

    sender = msg.sender;

    require(\_input >= 100, "input should be greater or equals to 100");

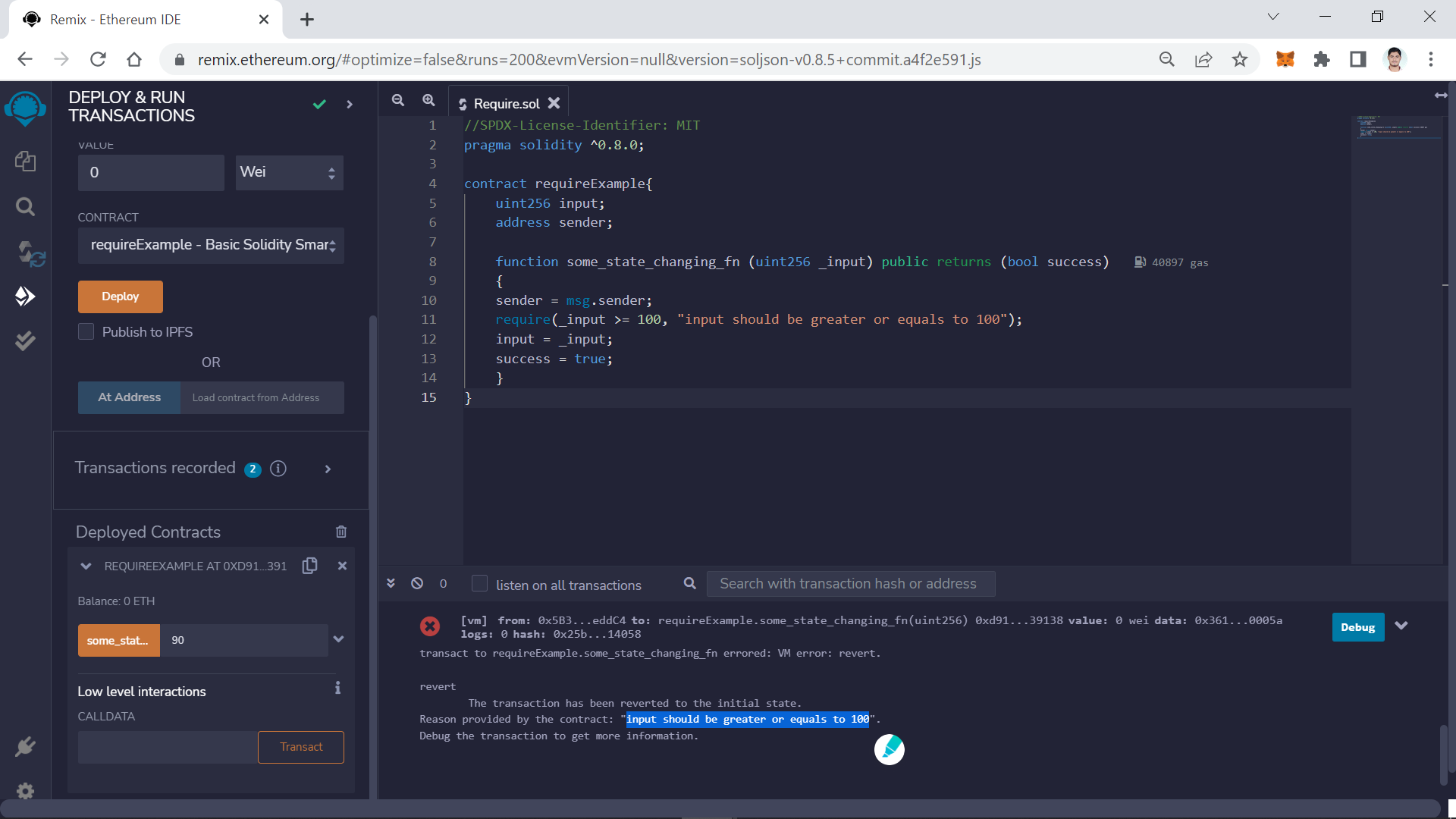
    input = \_input;

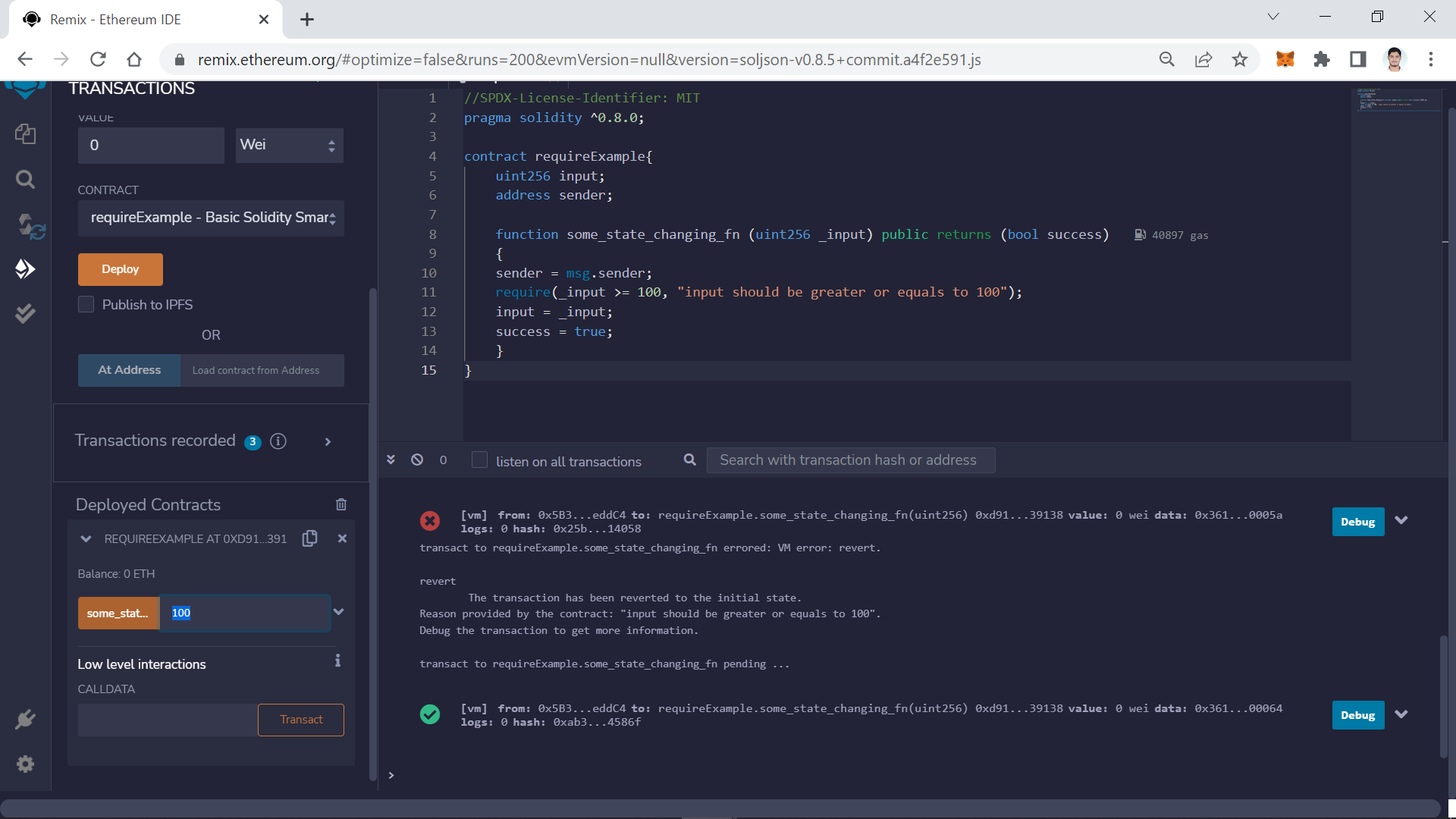
    success = true;

    }

}

**Output**





1. Inheritance

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/\* Graph of inheritance

    A

   / \

  B   C

 / \ /

F  D,E

\*/

contract A {

    function foo() public pure virtual returns (string memory) {

        return "A";

    }

}

contract B is A {

    function foo() public pure virtual override returns (string memory) {

        return "B";

    }

}

contract C is A {

    // Override A.foo()

    function foo() public pure virtual override returns (string memory) {

        return "C";

    }

}

contract D is B, C {

    function foo() public pure override(B, C) returns (string memory) {

        return super.foo();

    }

}

contract E is C, B {

    function foo() public pure override(C, B) returns (string memory) {

        return super.foo();

    }

}

contract F is A, B {

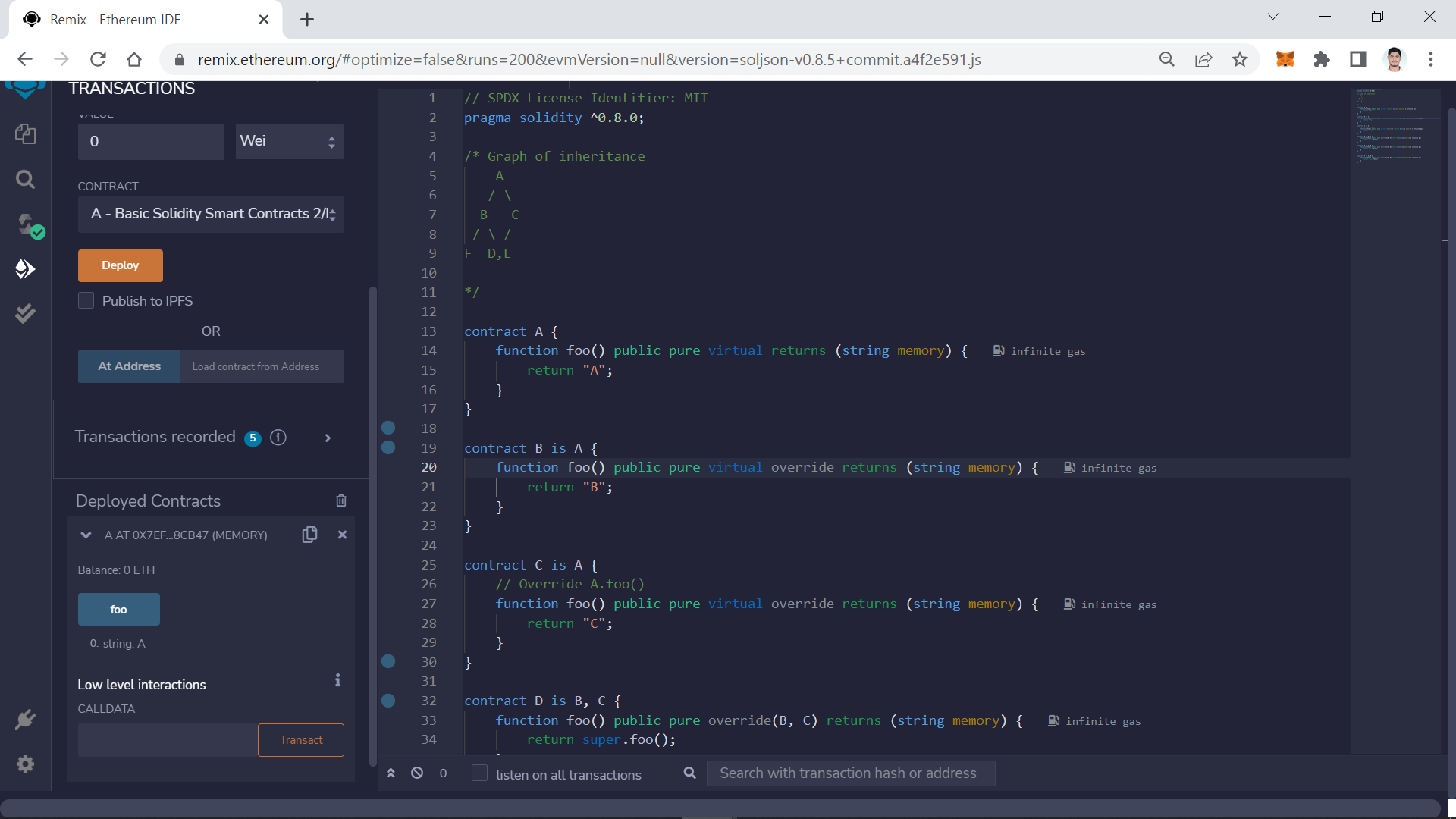
    function foo() public pure override(A, B) returns (string memory) {

        return super.foo();

    }

}

**Output**



1. CreateLibrary

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

library lib{

    function increment(uint val) public pure returns (uint){

        return val + 1;

    }

    function decrement(uint val) public pure returns (uint){

        return val - 1;

    }

    function incrementByValue(uint val, uint x) public pure returns (uint){

        return val + x;

    }

    function decrementByValue(uint val, uint x) public pure returns (uint){

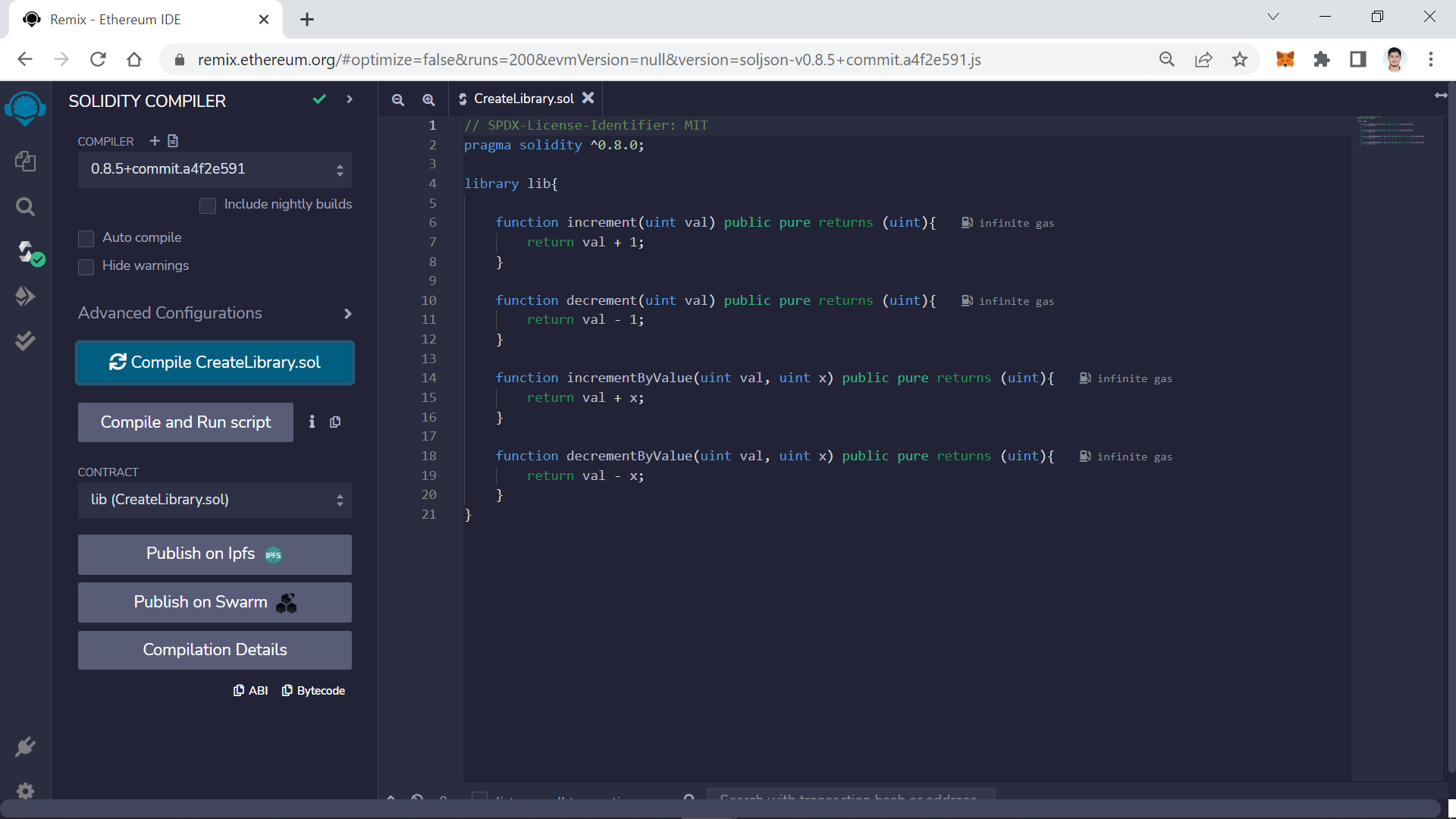
        return val - x;

    }

}

**NOTE:** To create library we have to only compile the file. No Need to deploy

**Output**



1. UseLibrary

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./CreateLibrary.sol";

contract useLibrary{

    using lib for uint;

    function testIncrement(uint userVal) public pure returns(uint){

        return lib.increment(userVal);

    }

    function testDecrement(uint userVal) public pure returns(uint){

        return lib.decrement(userVal);

    }

    function testIncrementByValue(uint userVal, uint x) public pure returns(uint){

        return lib.incrementByValue(userVal, x);

    }

    function testDecrementByValue(uint userVal, uint x) public pure returns(uint){

        return lib.decrementByValue(userVal, x);

    }

}

**Output**

