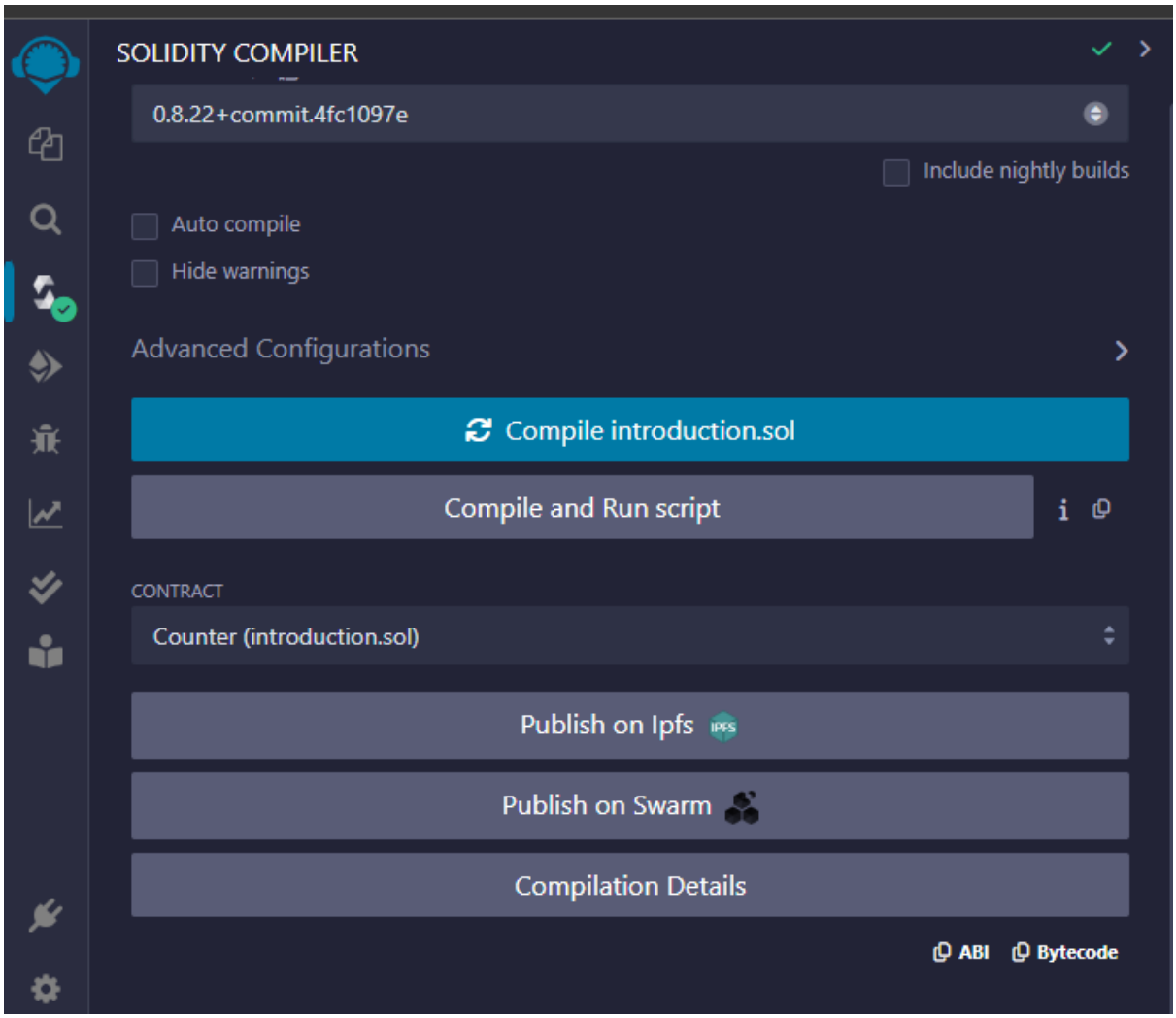
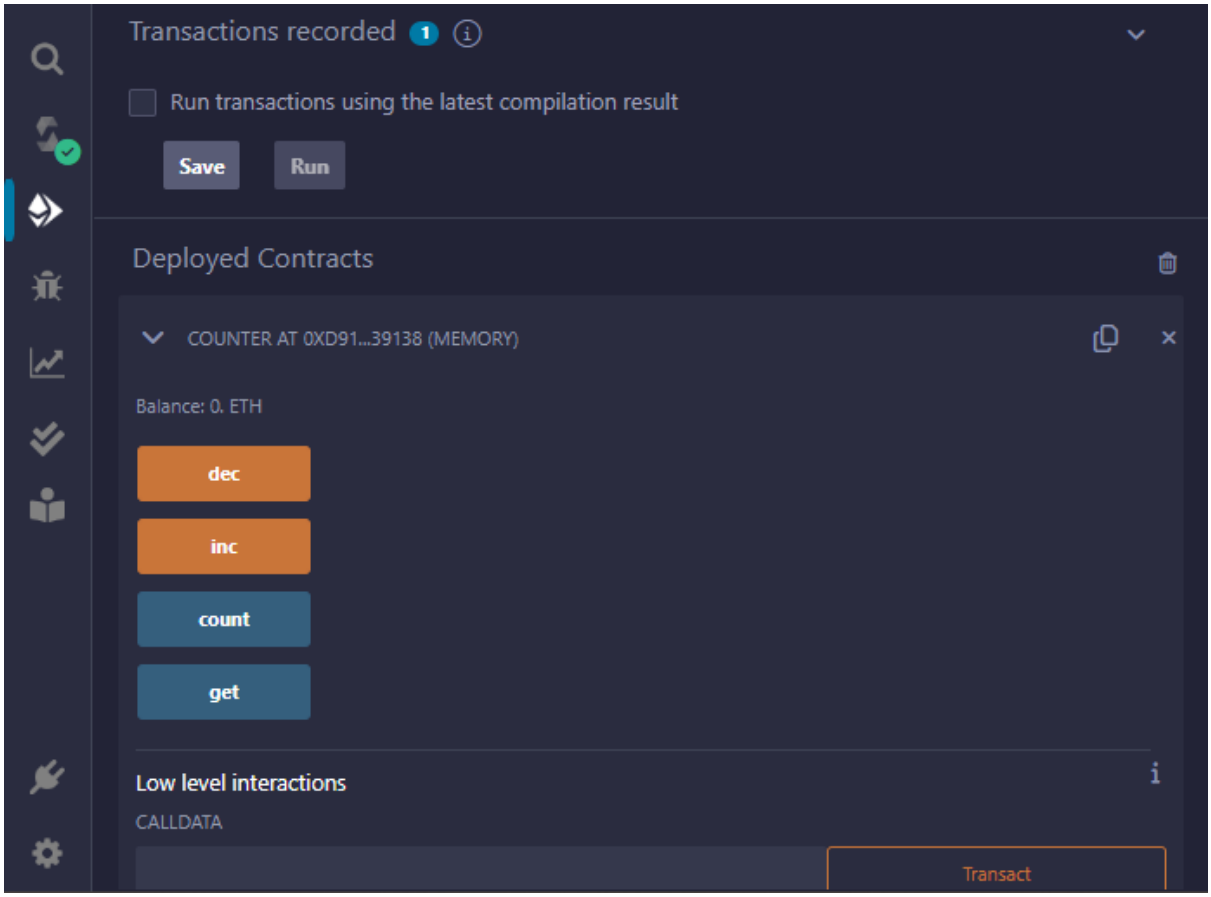
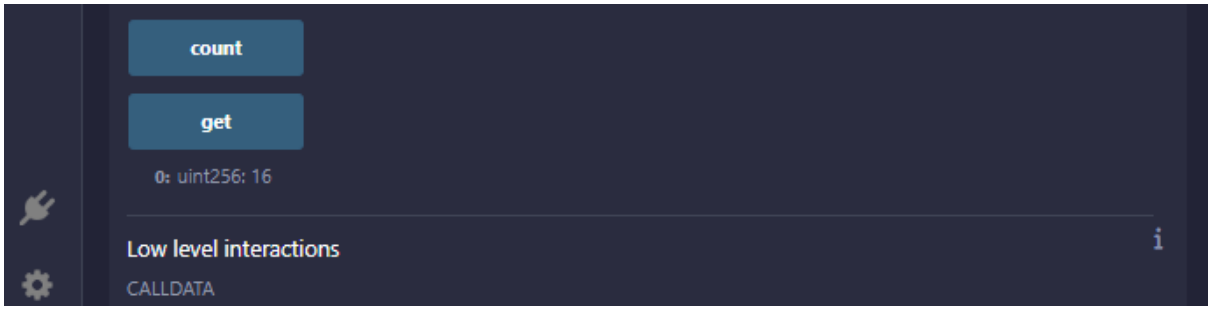
TP BLOCKCHAIN 2

Réalisé par : N'GAIBHAN-KOUASSI ANGE (TLR3)

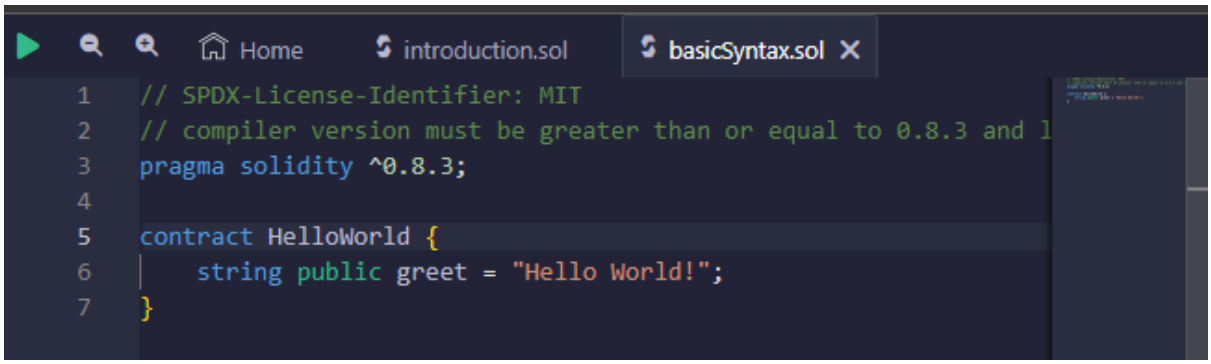


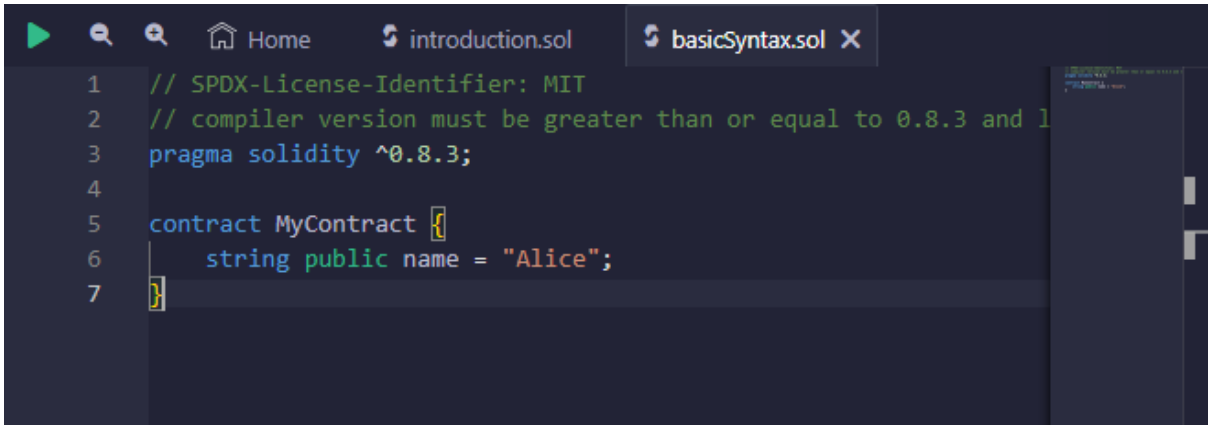


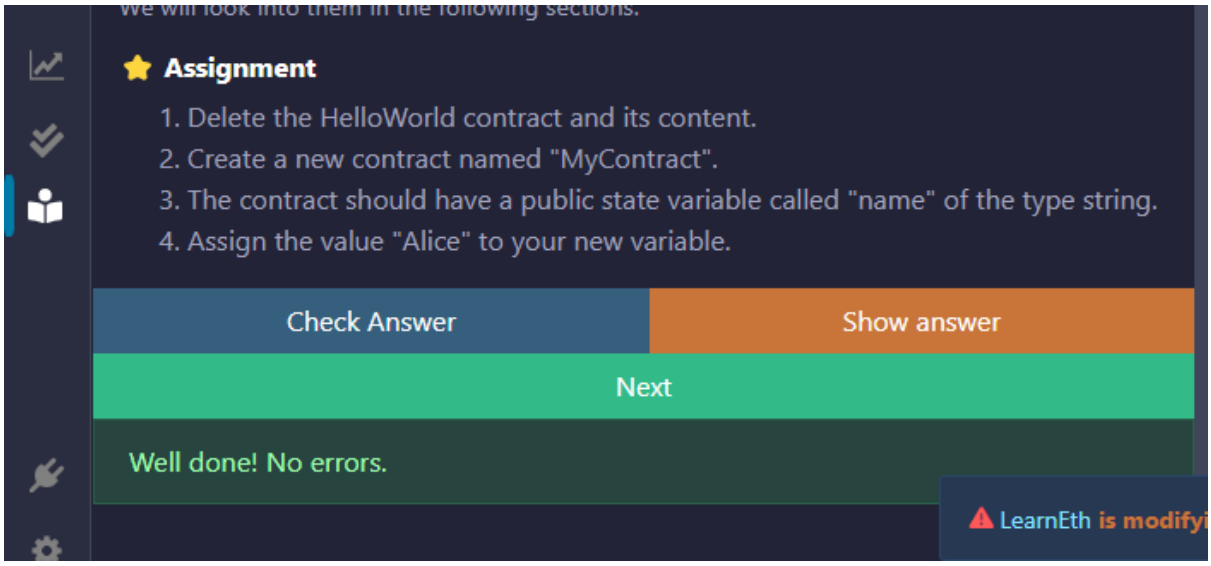


Le bytecode a été stocké dans la mémoire de la blockchain «Ethereum », dans le compte du contrat.

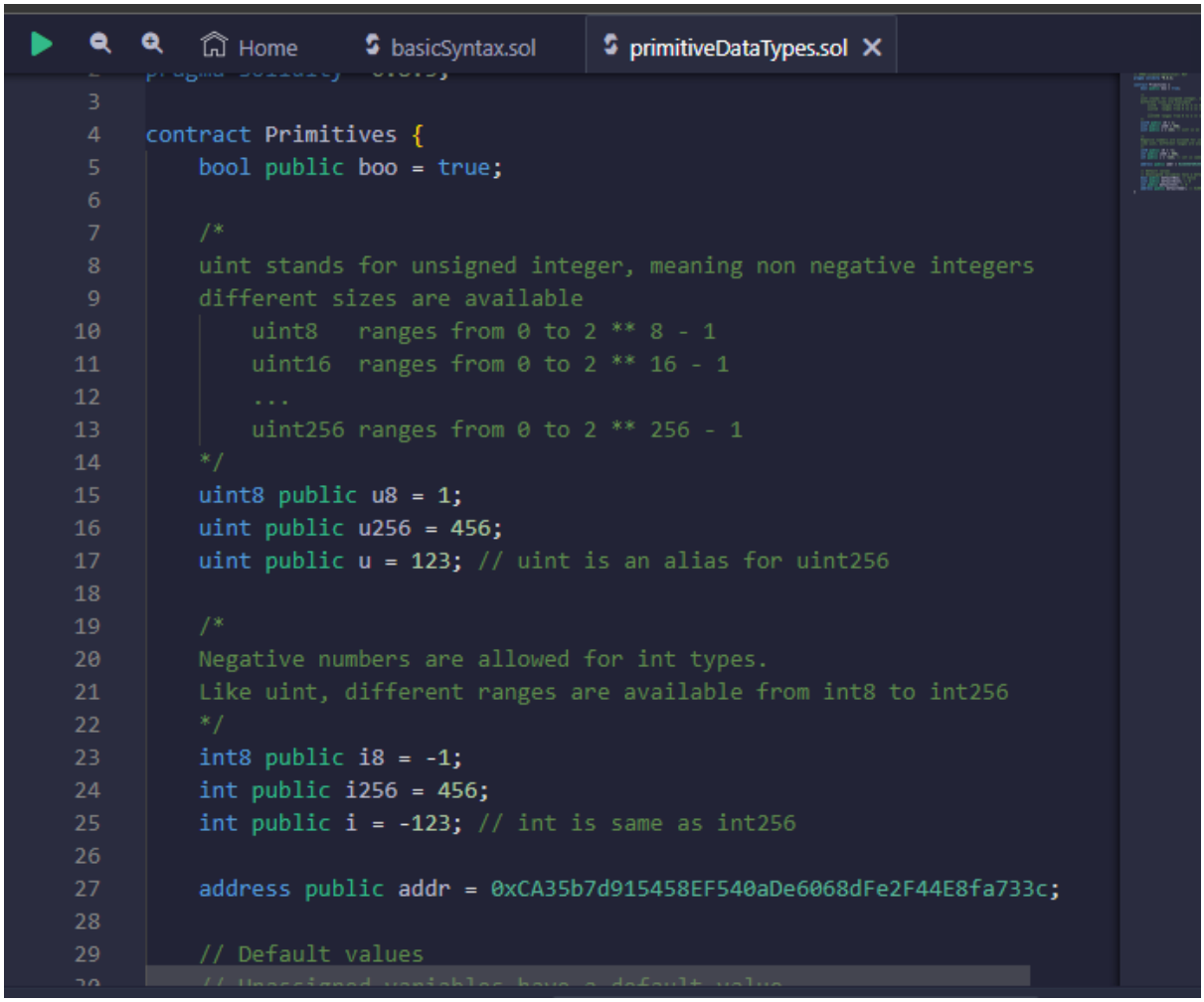
2. Basic Syntax :

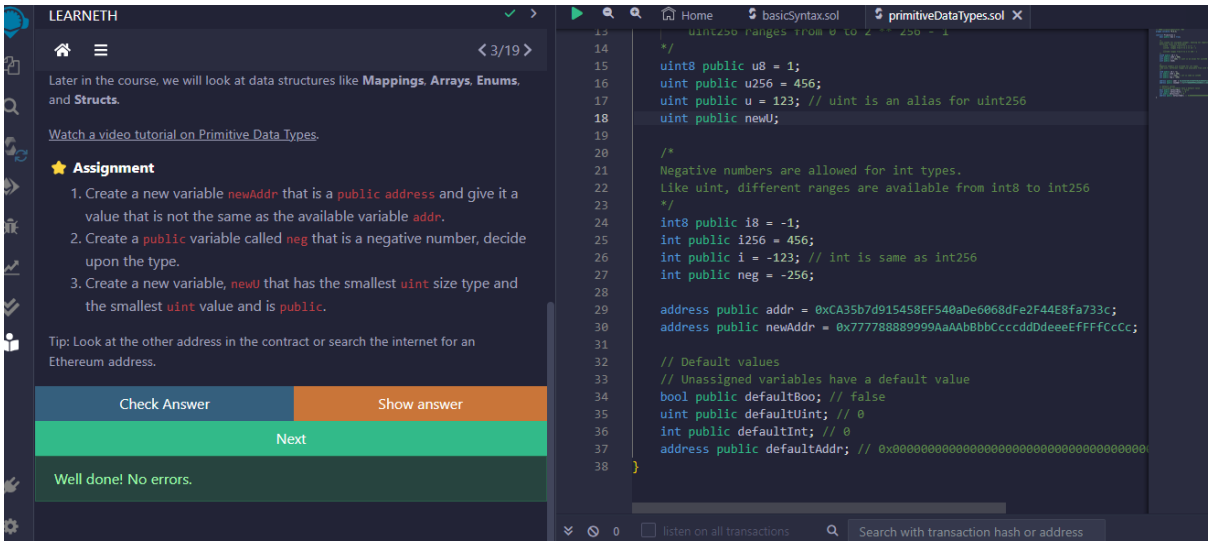




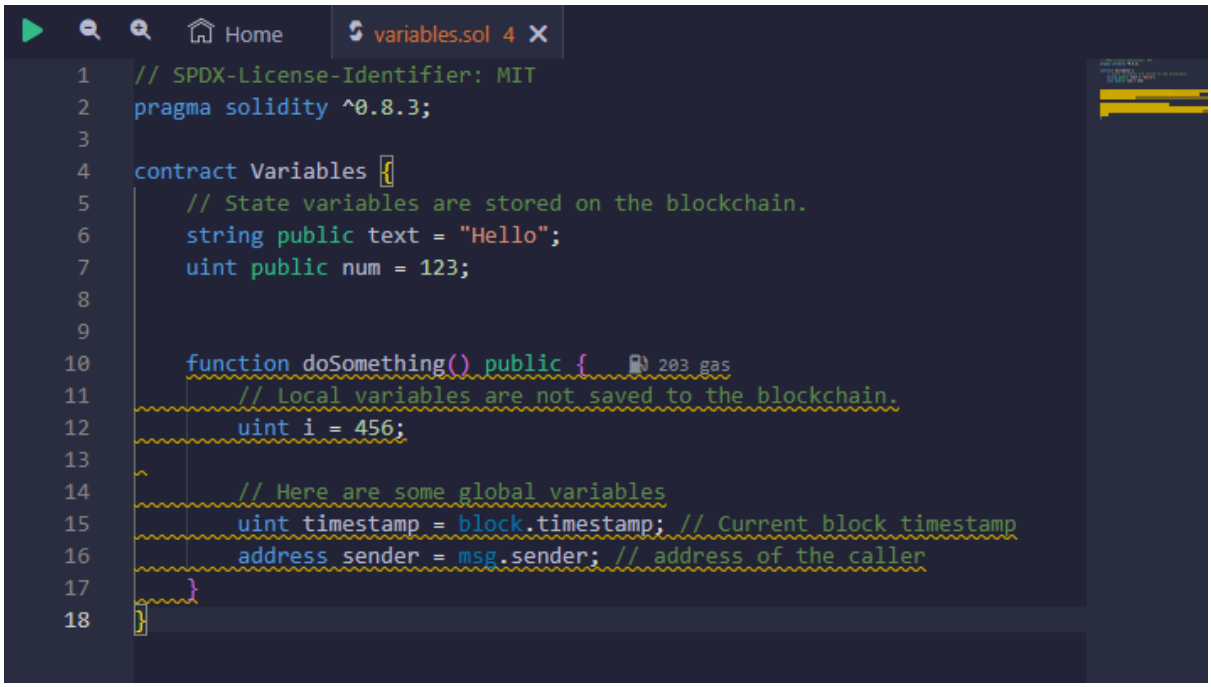


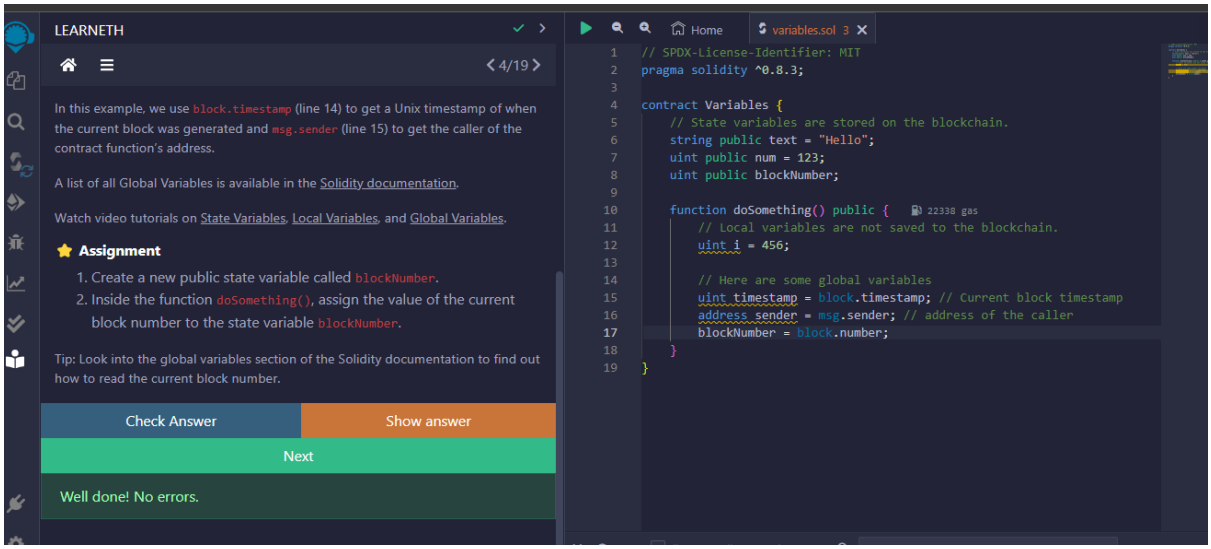
3. Primitive Data Types :



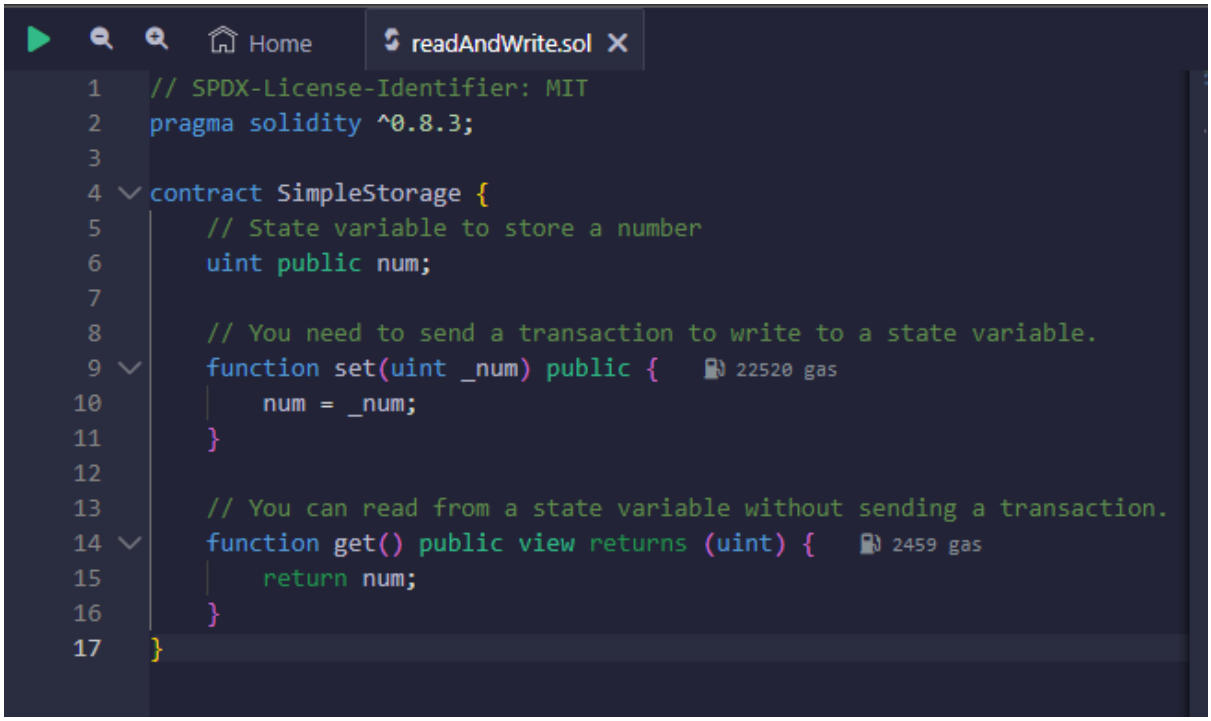


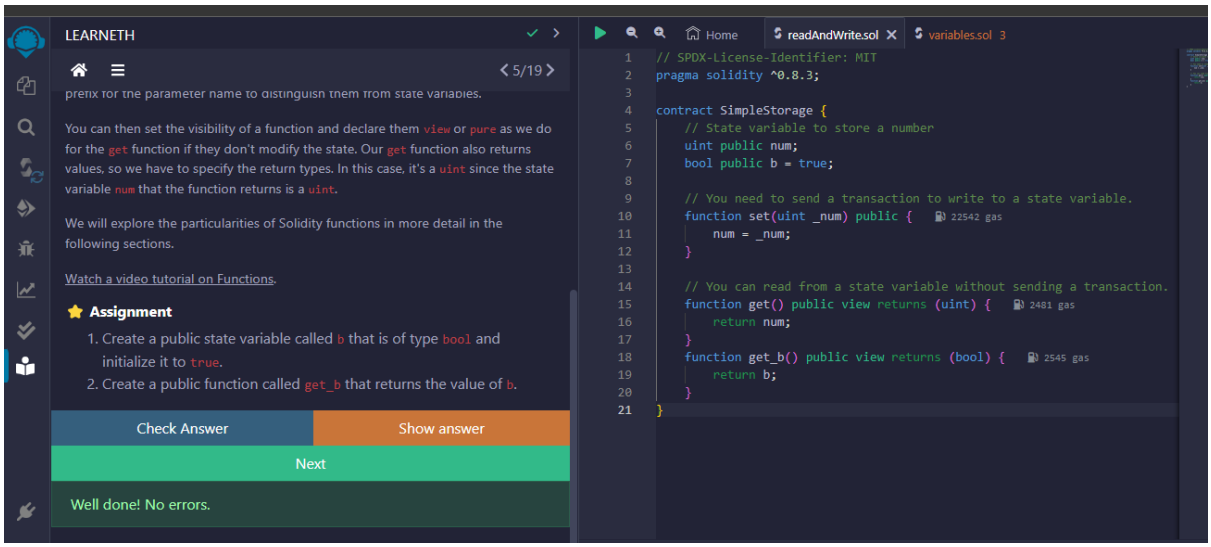
4. Les Variables :





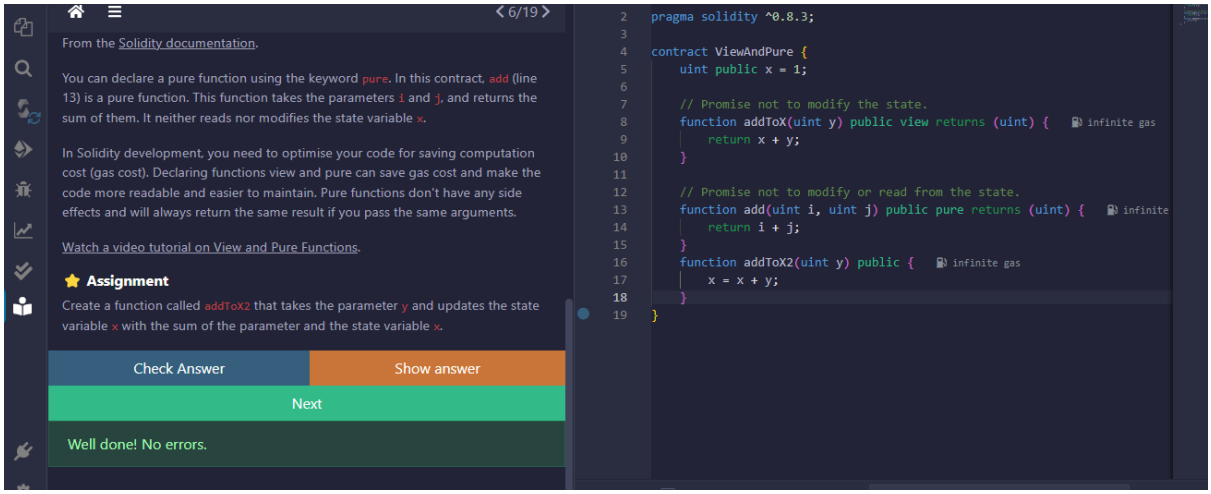
5.1 Functions - Reading and Writing to a State Variable





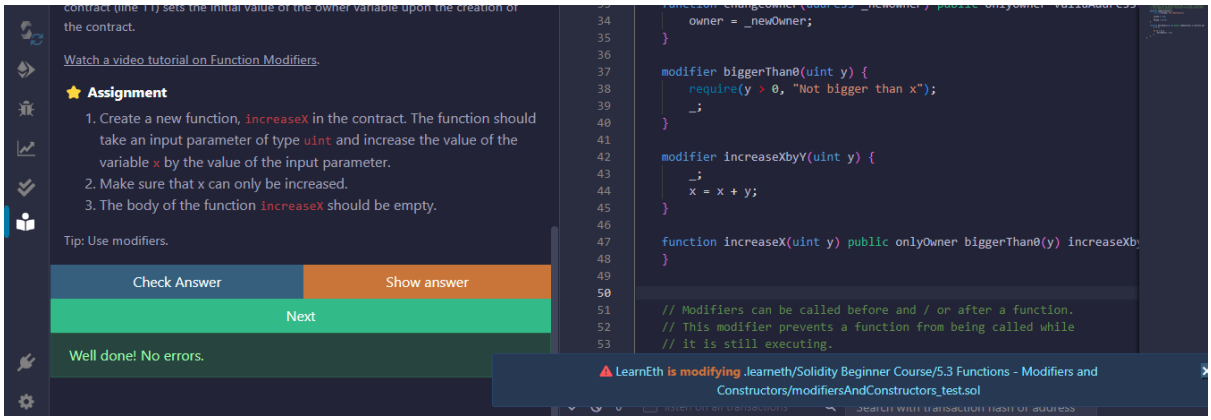
5.2 Functions - View and Pure

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

contract ViewAndPure {
    uint public x = 1;

    // Promise not to modify the state.
    function addToX(uint y) public view returns (uint) {    infinite gas
        return x + y;
    }

    // Promise not to modify or read from the state.
    function add(uint i, uint j) public pure returns (uint) {    infinite
        return i + j;
    }
```



5.3 Functions - Modifiers and Constructors

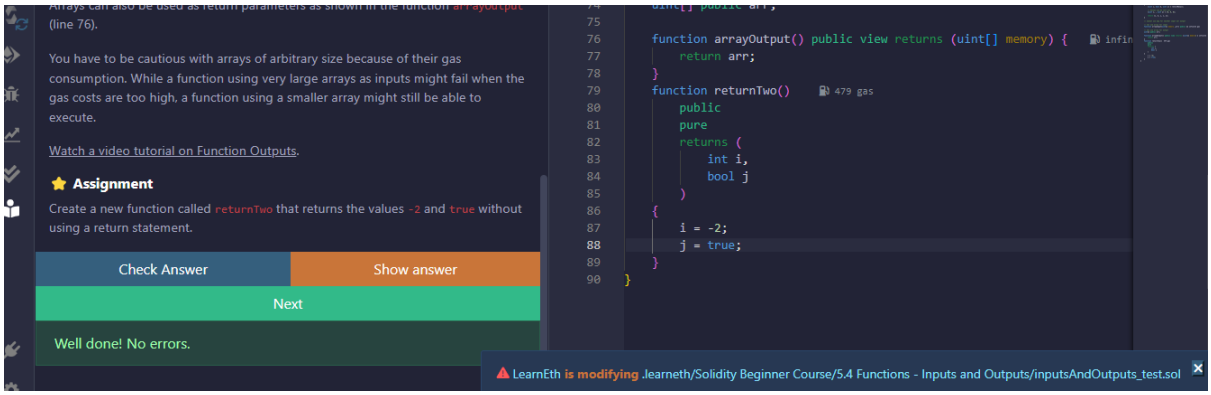
```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract FunctionModifier {
    // We will use these variables to demonstrate how to use
    // modifiers.
    address public owner;
    uint public x = 10;
    bool public locked;

    constructor() {    384546 gas 337800 gas
        // Set the transaction sender as the owner of the contract.
        owner = msg.sender;
    }

    // Modifier to check that the caller is the owner of
    // the contract.
    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        // Underscore is a special character only used inside
        // a function modifier and it tells Solidity to
        // execute the rest of the code.
        _;
    }

    // Modifiers can take inputs. This modifier checks that the
    // address passed in is not the zero address.
    modifier validAddress(address  addr) {
```



5.4 Functions - Inputs and Outputs

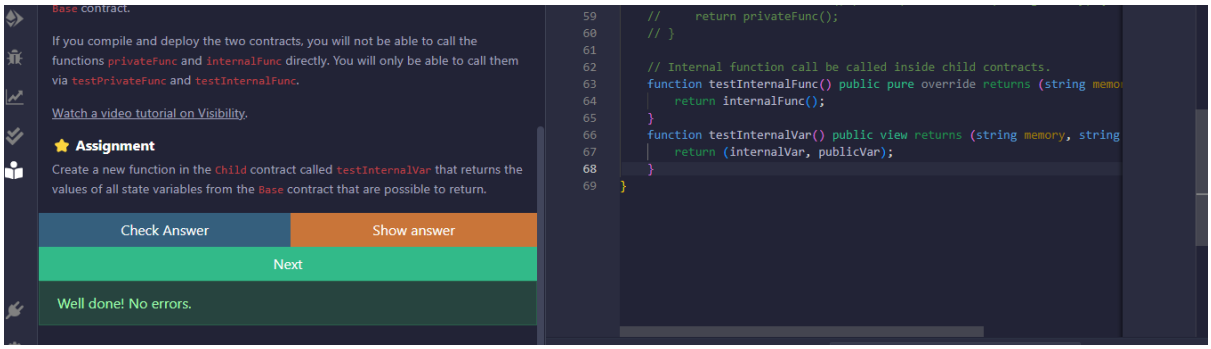
Home | inputsAndOutputs.sol

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

contract Function {
    // Functions can return multiple values.
    function returnMany()    infinite gas
        public
        pure
        returns (
            uint,
            bool,
            uint
        )
    {
        return (1, true, 2);
    }

    // Return values can be named.
    function named()    infinite gas
        public
        pure
        returns (
            uint x,
            bool b,
            uint y
        )
    {
```



6. Visibility

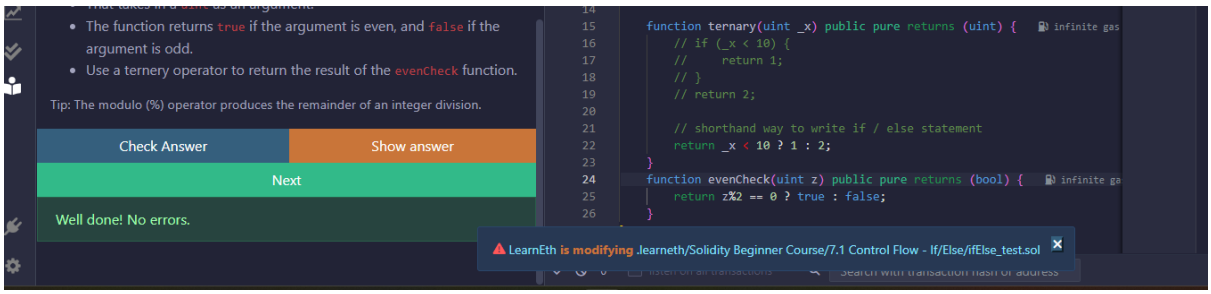
```
41      // This function will not compile since we're trying to call
42      // an external function here.
43      // function testExternalFunc() public pure returns (string memory) {
44      //     return externalFunc();
45      // }
46
47      // State variables
48      string private privateVar = "my private variable";
49      string internal internalVar = "my internal variable";
50      string public publicVar = "my public variable";
51      // State variables cannot be external so this code won't compile.
52      // string external externalVar = "my external variable";
53  }
54
55  contract Child is Base {
56      // Inherited contracts do not have access to private functions
57      // and state variables.
58      // function testPrivateFunc() public pure returns (string memory) {
59      //     return privateFunc();
60      // }
61
62      // Internal function call be called inside child contracts.
63      function testInternalFunc() public pure override returns (string memo
64          return internalFunc();
65      }
66  }
```



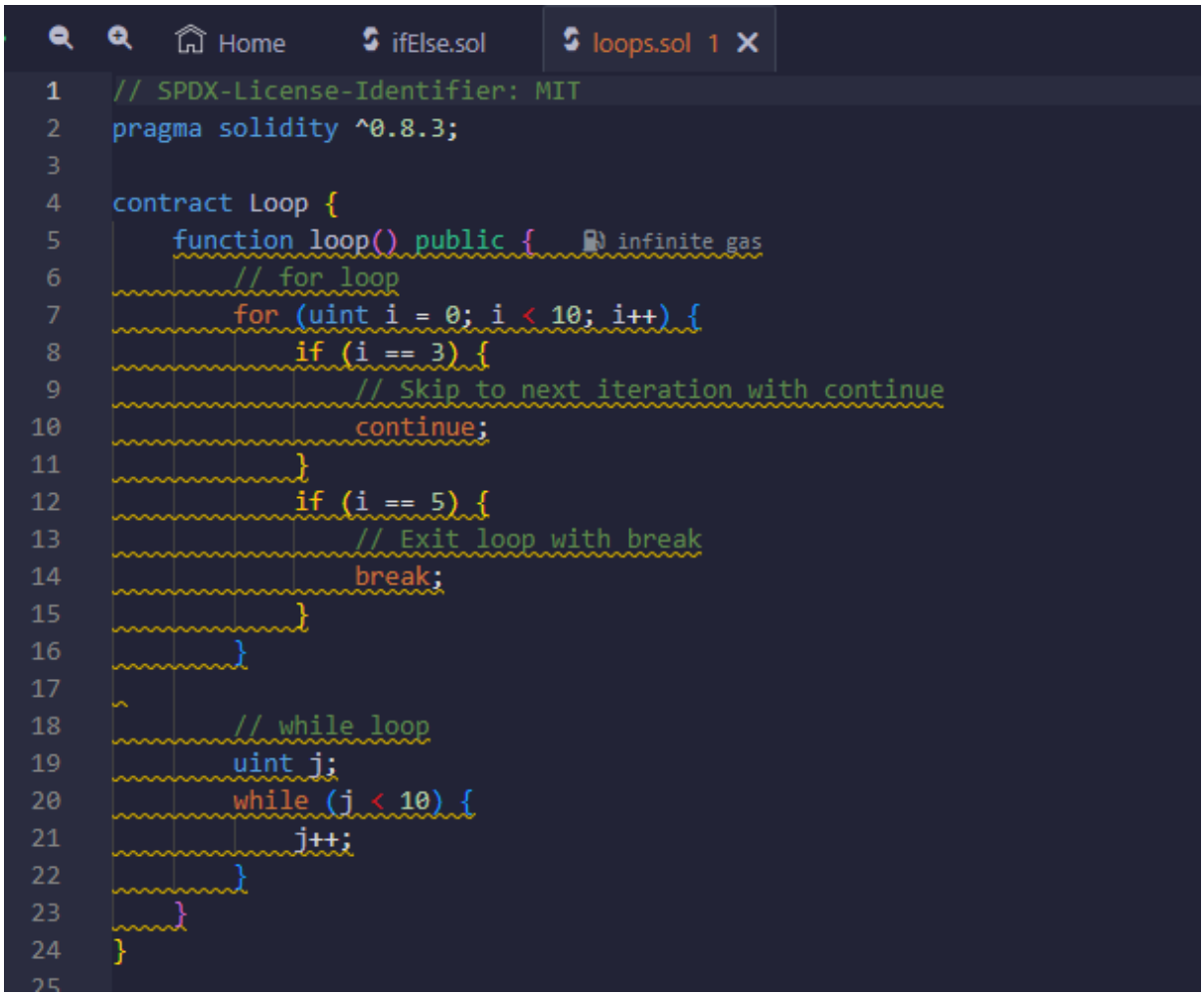
7.1 Control Flow - If/Else

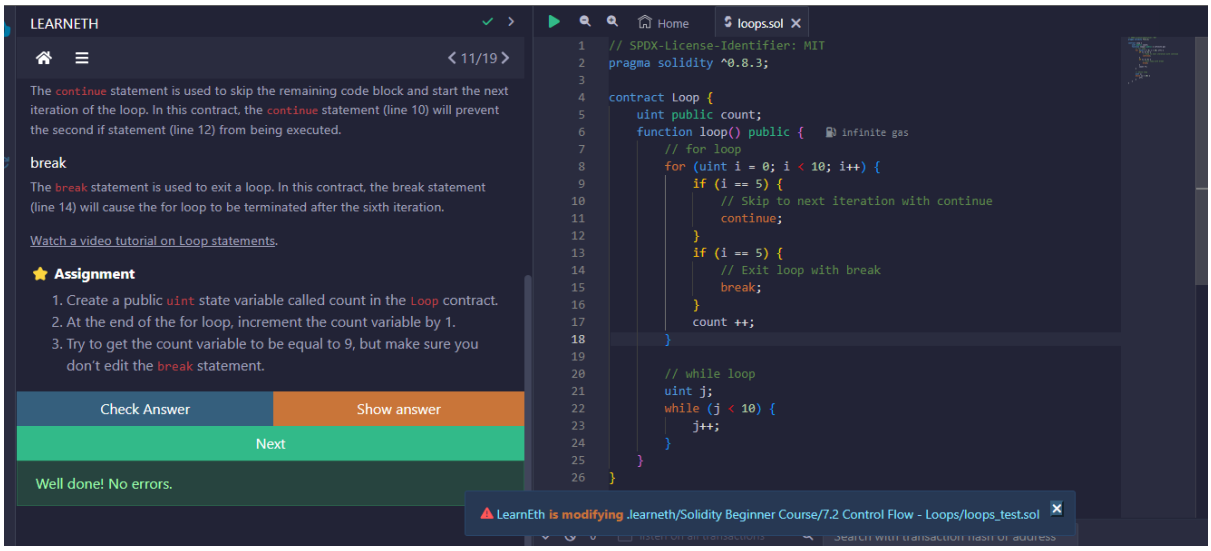
Home ifElse.sol

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

contract IfElse {
    function foo(uint x) public pure returns (uint) {    infinite gas
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

    function ternary(uint _x) public pure returns (uint) {    infinite gas
        // if (_x < 10) {
        //     return 1;
        // }
        // return 2;

        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }
}
```

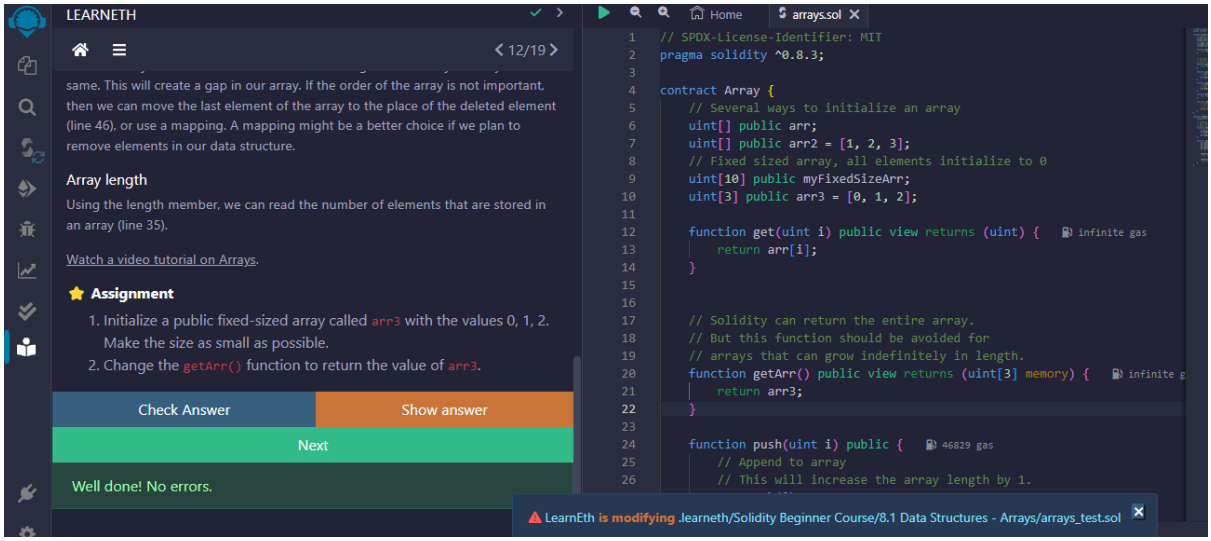


7.2 Control Flow - Loops

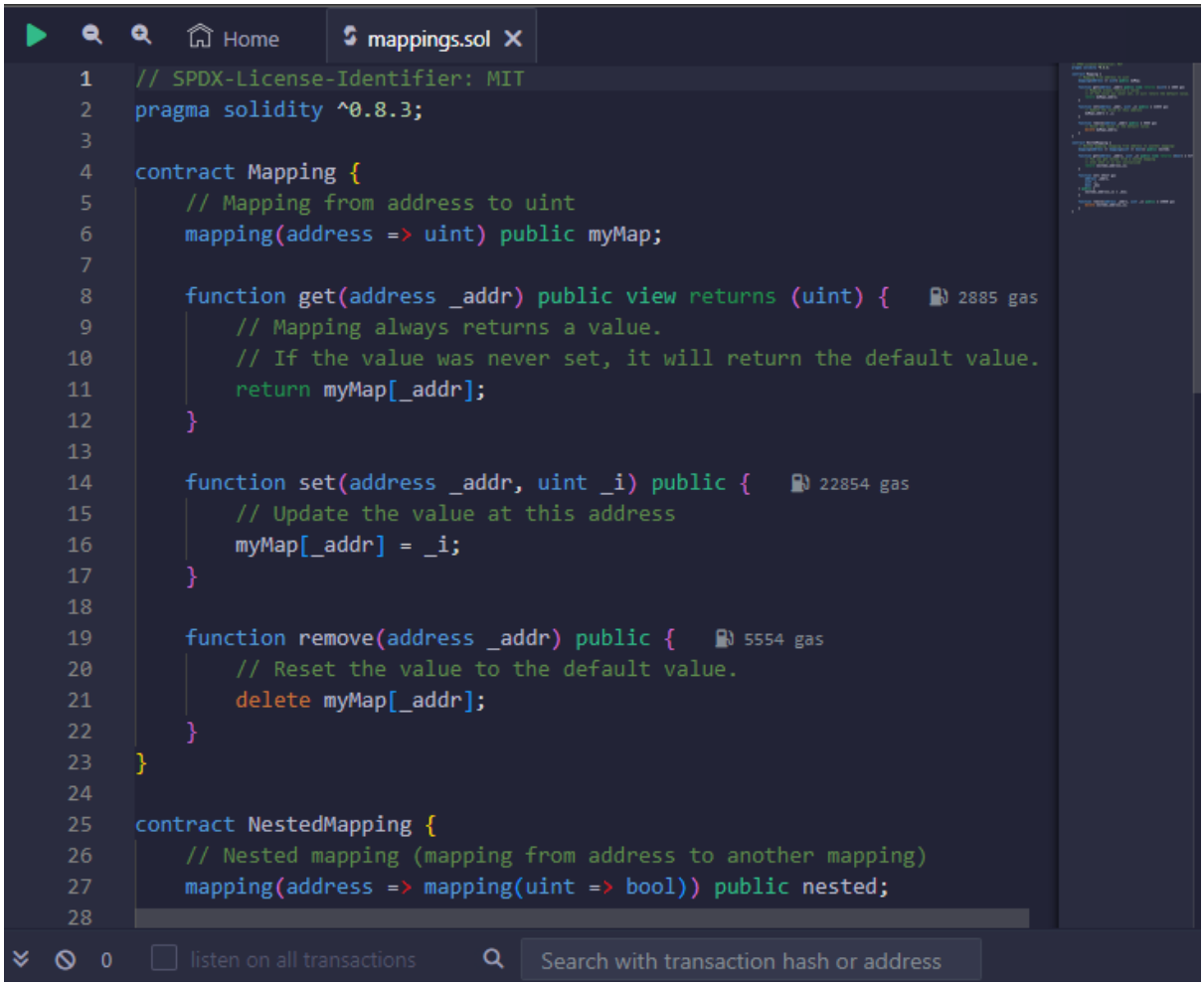


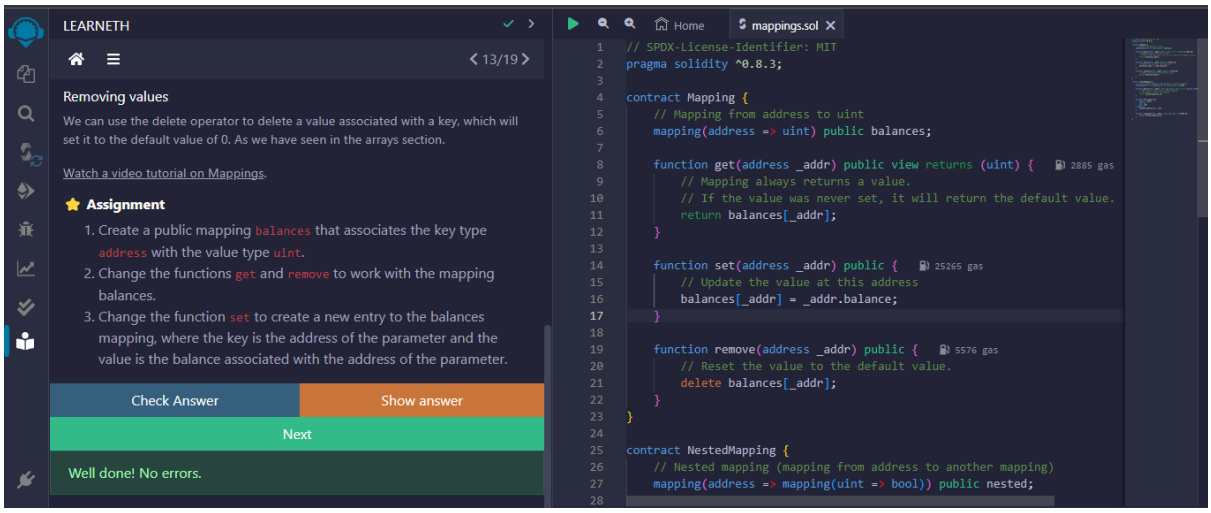


```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

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) {    infinite gas
        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) {    infinite ga
        return arr;
    }

    function push(uint i) public {    46829 gas
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {    29467 gas
```



8.2 Data Structures - Mappings

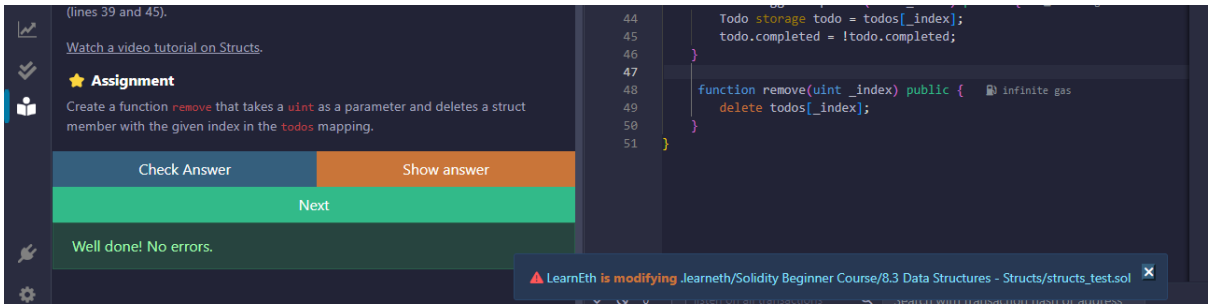




8.3 Data Structures - Structs

```sol
// 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 {    infinite gas
        // 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);
    }
```

0 listen on all transactions Search with transaction hash or address



8.4 Data Structures - Enums

Home | enums.sol

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

contract Enum {
    // Enum representing shipping status
    enum Status {
        Pending,
        Shipped,
        Accepted,
        Rejected,
        Canceled
    }

    // Default value is the first element listed in
    // definition of the type, in this case "Pending"
    Status public status;

    // Returns uint
    // Pending  - 0
    // Shipped  - 1
    // Accepted - 2
    // Rejected - 3
    // Canceled - 4
    function get() public view returns (Status) {    2590 gas
        return status;
    }

    // Update status by passing uint into input
```

the enum member (line 30). Shipped would be 1 in this example. Another way to update the value is using the dot operator by providing the name of the enum and its member (line 35).

Removing an enum value

We can use the delete operator to delete the enum value of the variable, which means as for arrays and mappings, to set the default value to 0.

Watch a video tutorial on Enums.

⭐ Assignment

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

Check Answer | Show answer

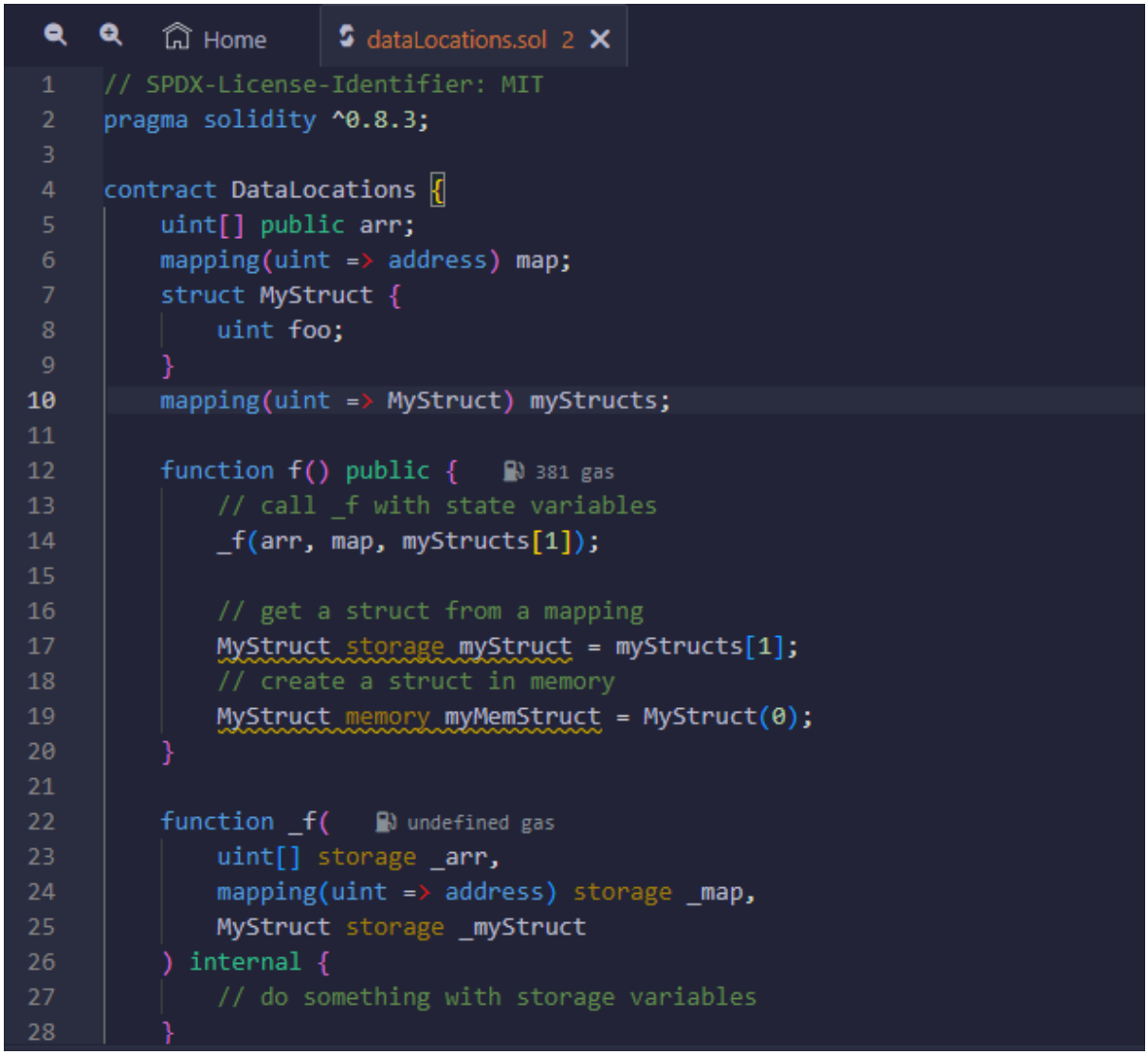
Next

Well done! No errors.

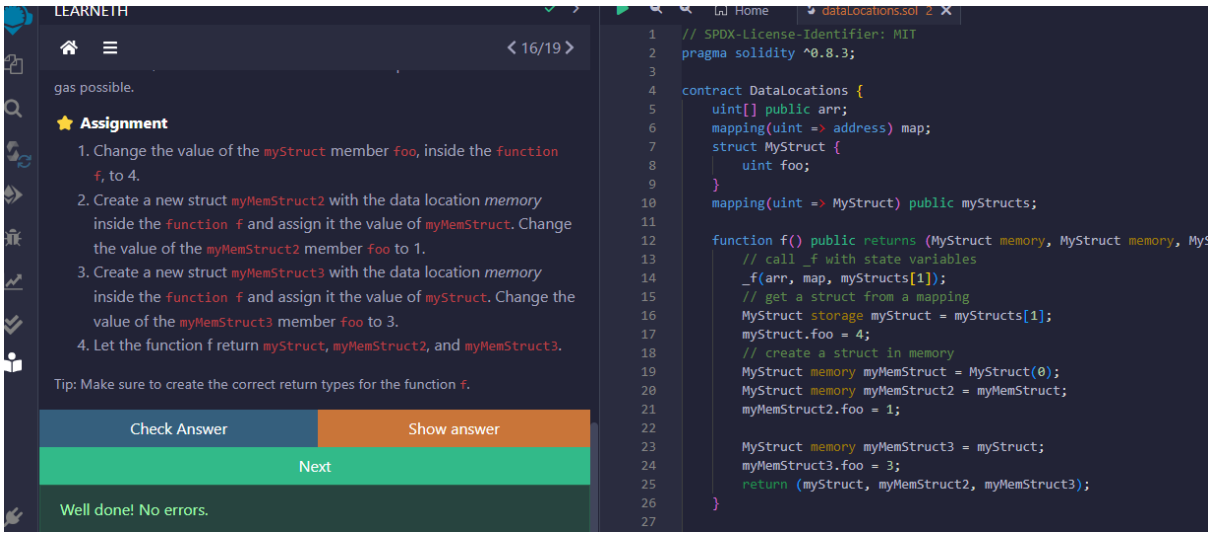
```sol
enum Size {
    S,
    M,
    L
}


// Default value is the first element listed in
// definition of the type, in this case "Pending"
Status public status;
Size public sizes;

// Returns uint
// Pending  - 0
// Shipped  - 1
// Accepted - 2
// Rejected - 3
// Canceled - 4
function get() public view returns (Status) {    2613 gas
    return status;
}
function getSize() public view returns (Size) {    2640 gas
    return sizes;
}
```

⚠ LearnEth is modifying .learneth/Solidity Beginner Course/8.4 Data Structures - Enums/enums_test.sol

9. Data Locations



```
// 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) myStructs;

    function f() public {    381 gas
        // call _f with state variables
        _f(arr, map, myStructs[1]);

        // get a struct from a mapping
        MyStruct storage myStruct = myStructs[1];
        // create a struct in memory
        MyStruct memory myMemStruct = MyStruct(0);
    }

    function _f(    undefined gas
        uint[] storage _arr,
        mapping(uint => address) storage _map,
        MyStruct storage _myStruct
    ) internal {
        // do something with storage variables
    }
```



LEARNETH

16/19

gas possible.

Assignment

1. Change the value of the `myStruct` member `foo`, inside the `function f`, to 4.
2. 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.
3. 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.
4. Let the function f return `myStruct`, `myMemStruct2`, and `myMemStruct3`.

Tip: Make sure to create the correct return types for the function `f`.

Check Answer | Show answer

Next

Well done! No errors.

```
// 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, MyS
        // call _f with state variables
        _f(arr, map, myStructs[1]);
        // get a struct from a mapping
        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);
    }
```

10.1 Transactions - Ether and Wei