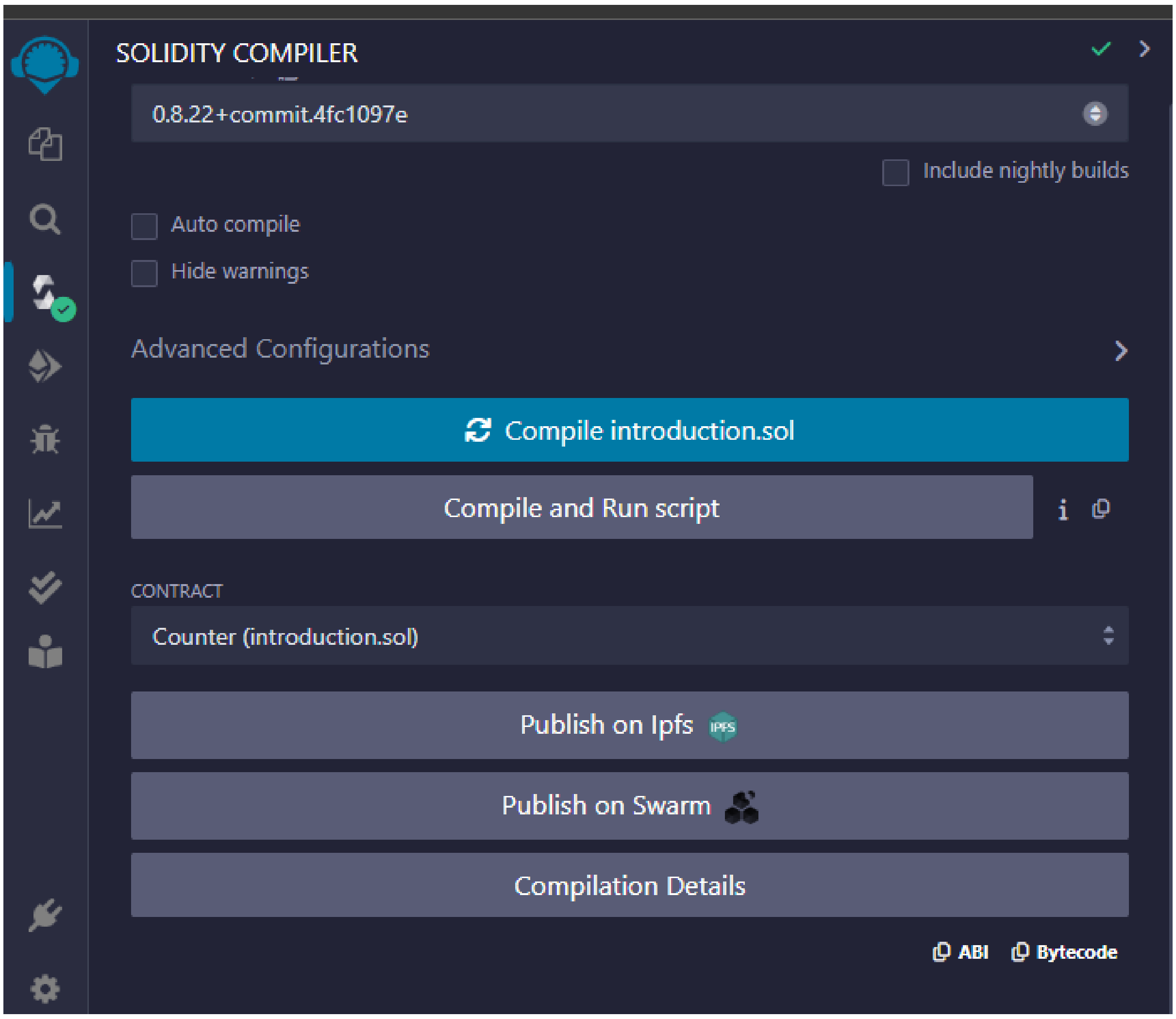
TP 2 : OUATTARA CANIDANNAN INFO 3

1.Introduction


SOLIDITY COMPILER

0.8.22+commit.4fc1097e

Include nightly builds

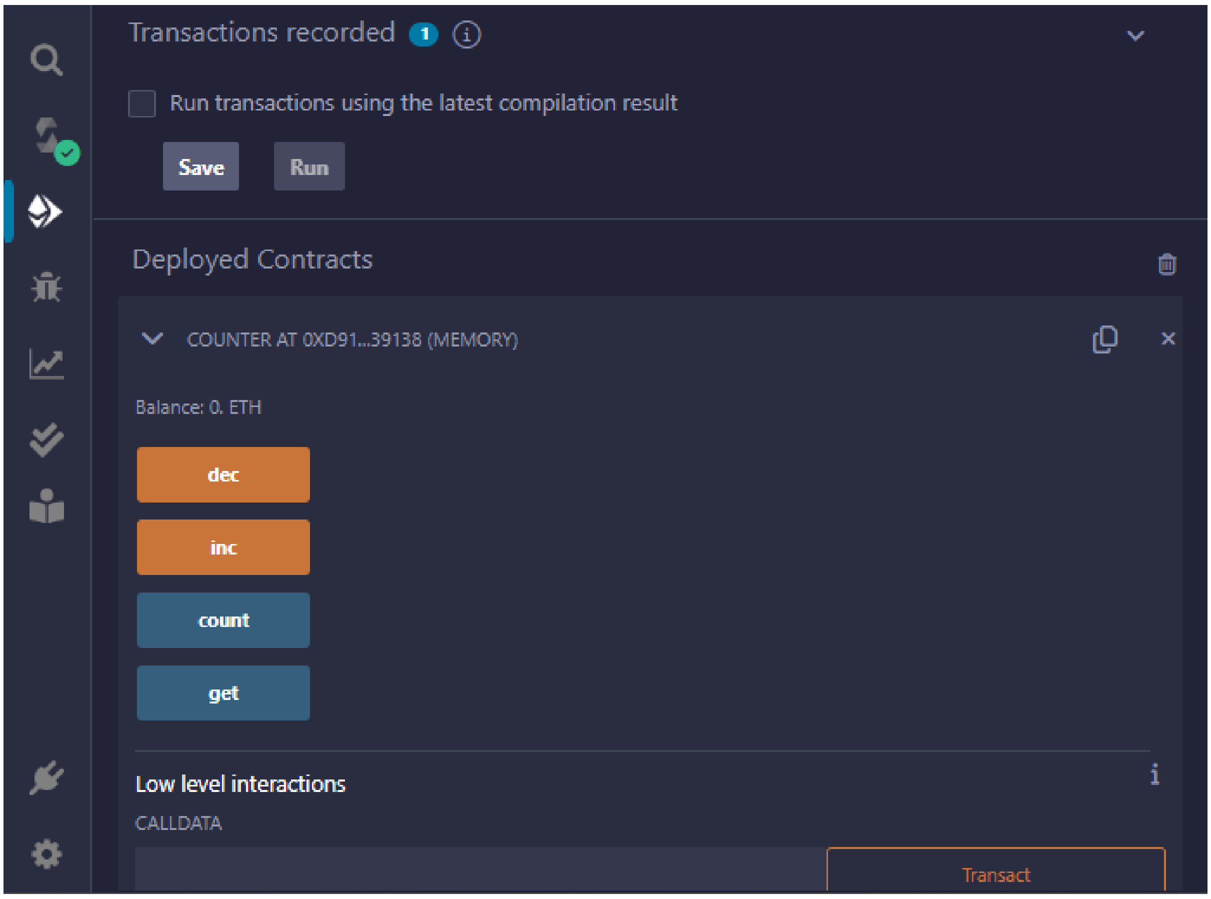
Auto compile

Hide warnings

Advanced Configurations

CONTRACT

Counter (introduction.sol)


Transactions recorded 1

Run transactions using the latest compilation result

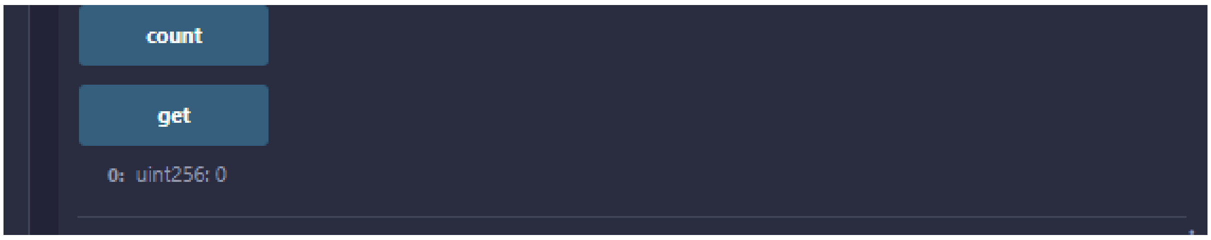
Deployed Contracts

COUNTER AT 0XD91...39138 (MEMORY)

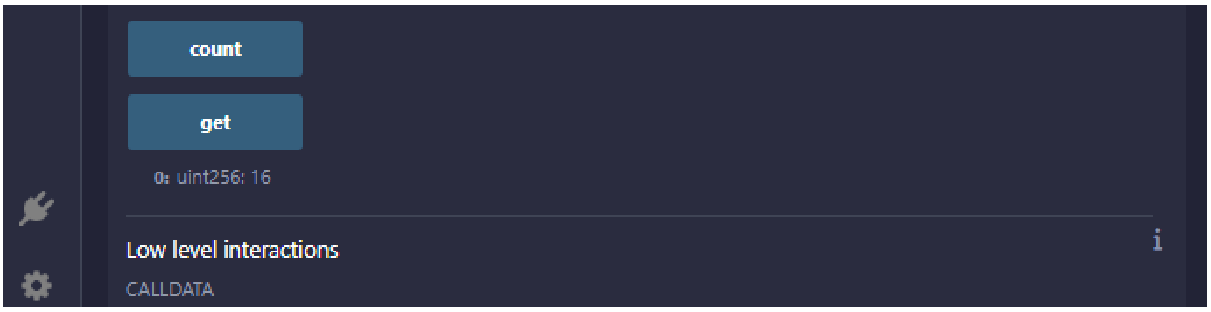
Balance: 0. ETH

Low level interactions

CALLDATA



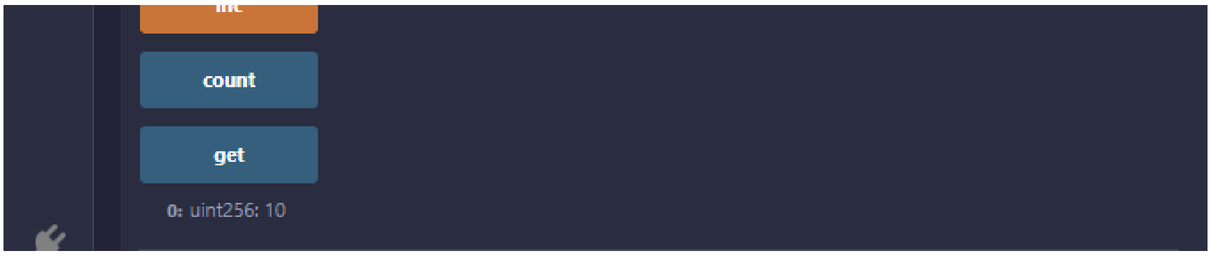
0: uint256: 0



0: uint256: 16

Low level interactions

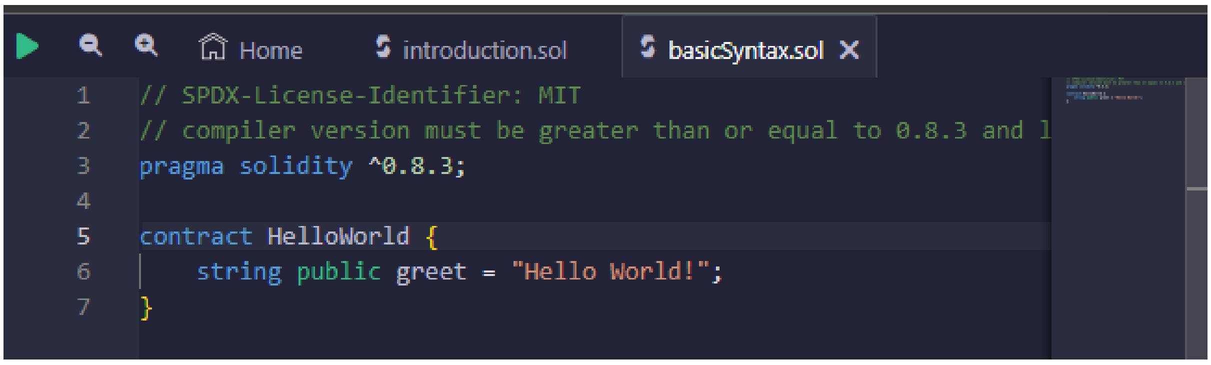
CALLDATA



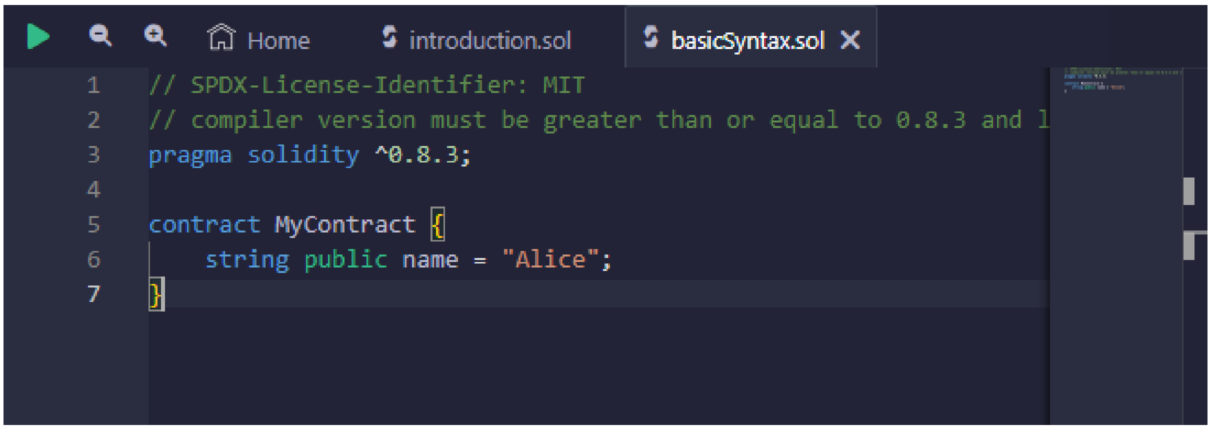
0: uint256: 10

Le bytecode est stocké dans la mémoire de la blockchain Ethereum, plus précisément dans le compte du contrat.

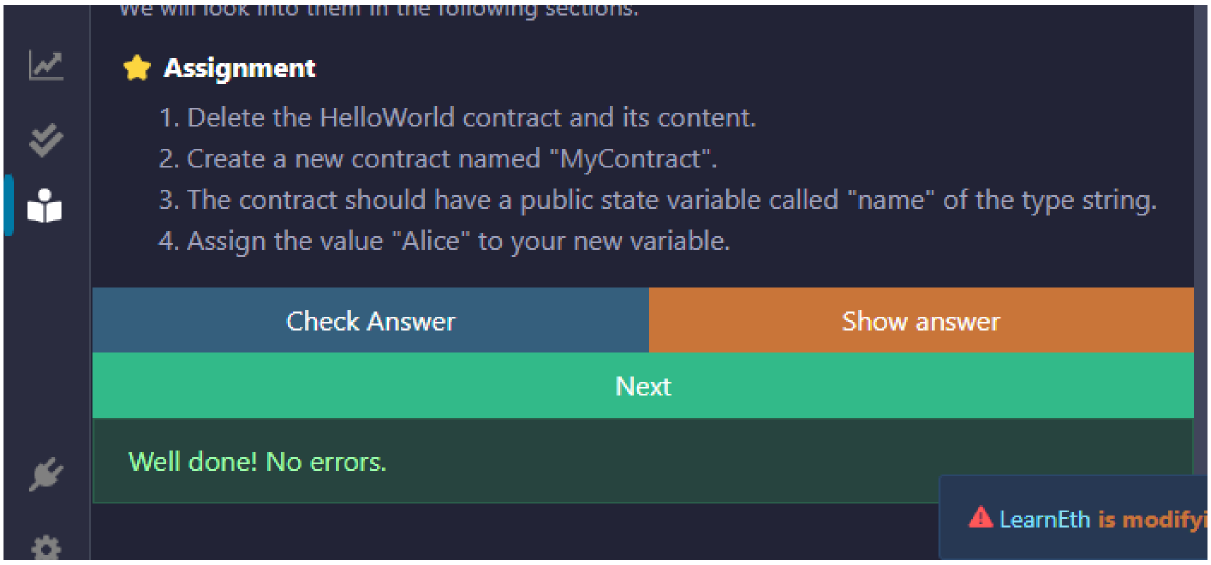
2. Basic Syntax



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

contract HelloWorld {
    string public greet = "Hello World!";
}
```



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

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



We will look into them in the following sections.

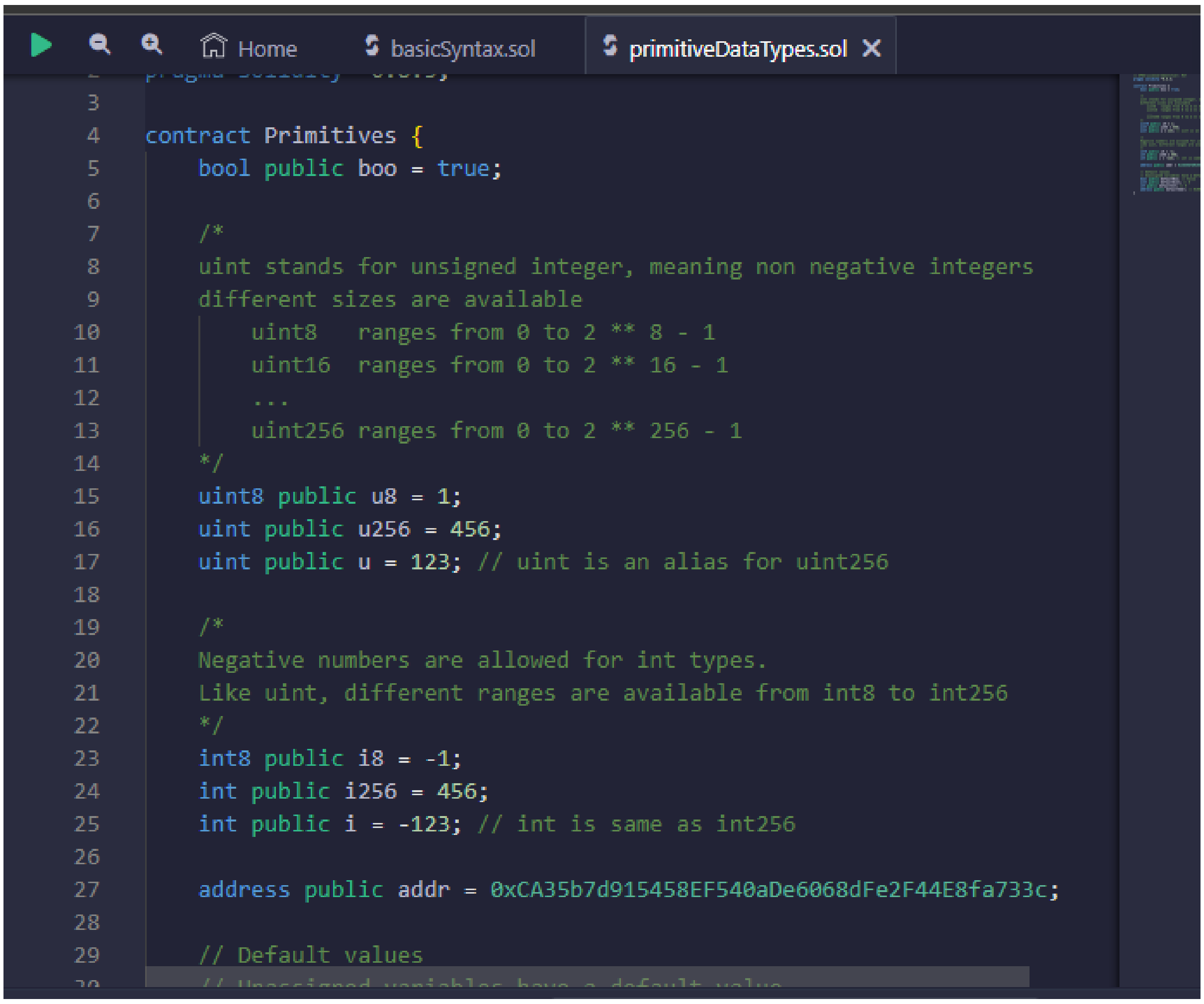
Assignment

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

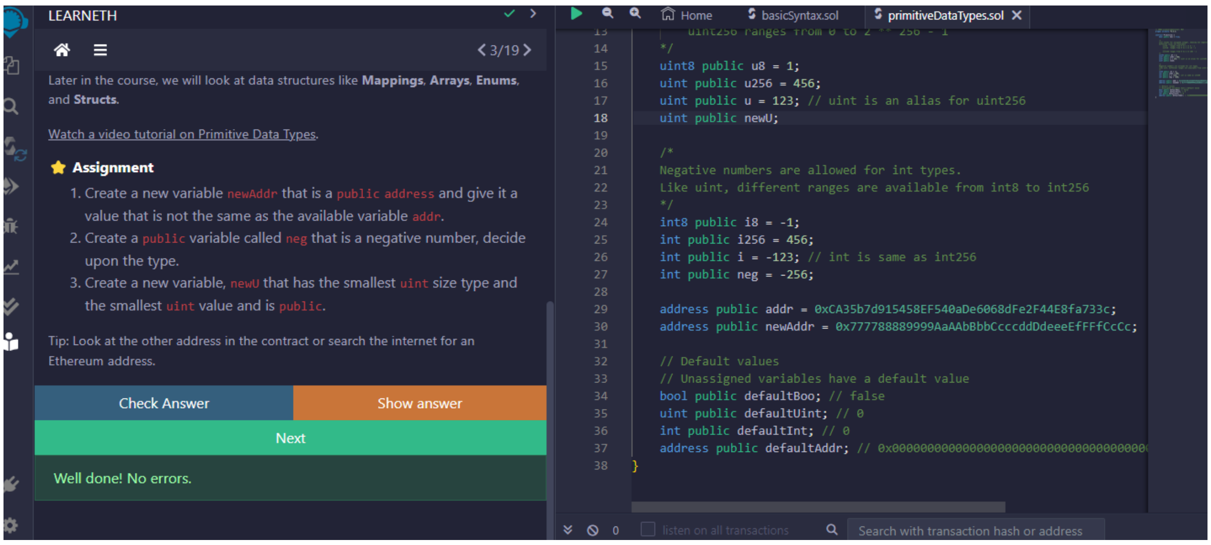
Well done! No errors.

LearnEth is modifyi

3. Primitive Data Types



```
contract Primitives {
    bool public boo = true;

    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available
        uint8   ranges from 0 to 2 ** 8 - 1
        uint16  ranges from 0 to 2 ** 16 - 1
        ...
        uint256 ranges from 0 to 2 ** 256 - 1
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256

    /*
    Negative numbers are allowed for int types.
    Like uint, different ranges are available from int8 to int256
    */
    int8 public i8 = -1;
    int public i256 = 456;
    int public i = -123; // int is same as int256

    address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

    // Default values
```



LEARNETH

3/19

Later in the course, we will look at data structures like **Mappings**, **Arrays**, **Enums**, and **Structs**.

Watch a video tutorial on Primitive Data Types.

Assignment

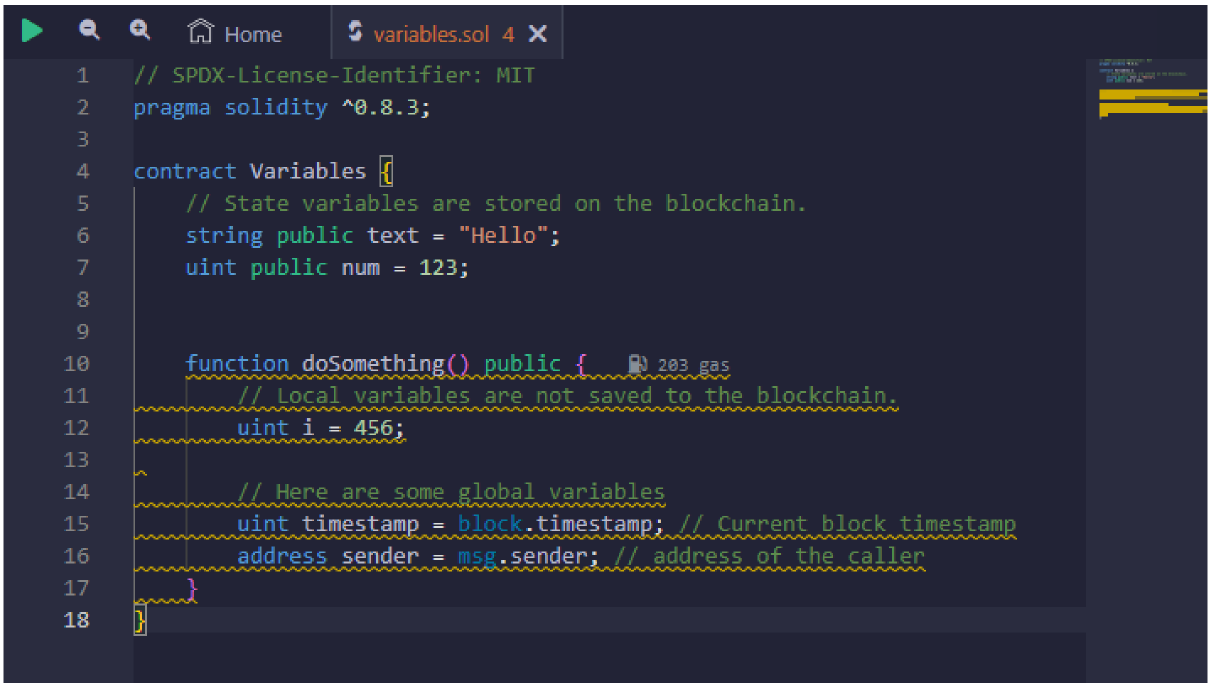
1. Create a new variable `newAddr` that is a `public address` and give it a value that is not the same as the available variable `addr`.
2. Create a `public` variable called `neg` that is a negative number, decide upon the type.
3. Create a new variable, `newU` that has the smallest `uint` size type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

Tip: Look at the other address in the contract or search the internet for an Ethereum address.

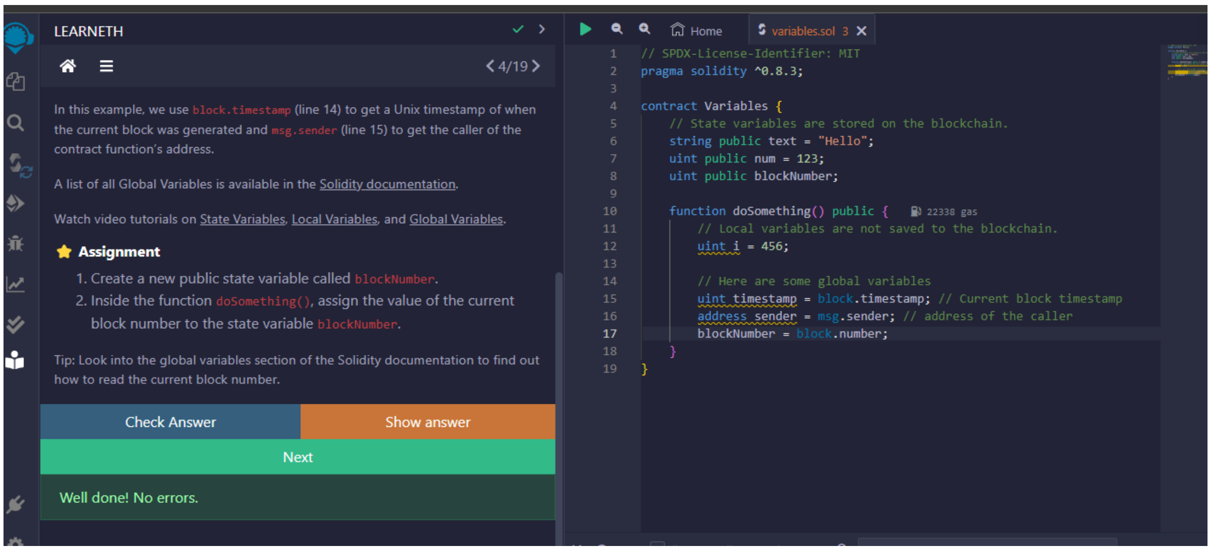
Well done! No errors.

```
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256
    uint public newU;

    /*
    Negative numbers are allowed for int types.
    Like uint, different ranges are available from int8 to int256
    */
    int8 public i8 = -1;
    int public i256 = 456;
    int public i = -123; // int is same as int256
    int public neg = -256;

    address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;
    address public newAddr = 0x777788889999AaAAbBbbCcccddDdeeeEfFFfCcCc;

    // Default values
    // Unassigned variables have a default value
    bool public defaultBoo; // false
    uint public defaultUint; // 0
    int public defaultInt; // 0
    address public defaultAddr; // 0x0000000000000000000000000000000000000
}
```

4. Variables



```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.3;
3
4  contract Variables {
5      // State variables are stored on the blockchain.
6      string public text = "Hello";
7      uint public num = 123;
8
9
10     function doSomething() public {
11         // Local variables are not saved to the blockchain.
12         uint i = 456;
13
14         // Here are some global variables
15         uint timestamp = block.timestamp; // Current block timestamp
16         address sender = msg.sender; // address of the caller
17     }
18 }
```



In this example, we use `block.timestamp` (line 14) to get a Unix timestamp of when the current block was generated and `msg.sender` (line 15) to get the caller of the contract function's address.

A list of all Global Variables is available in the Solidity documentation.

Watch video tutorials on State Variables, Local Variables, and Global Variables.

Assignment

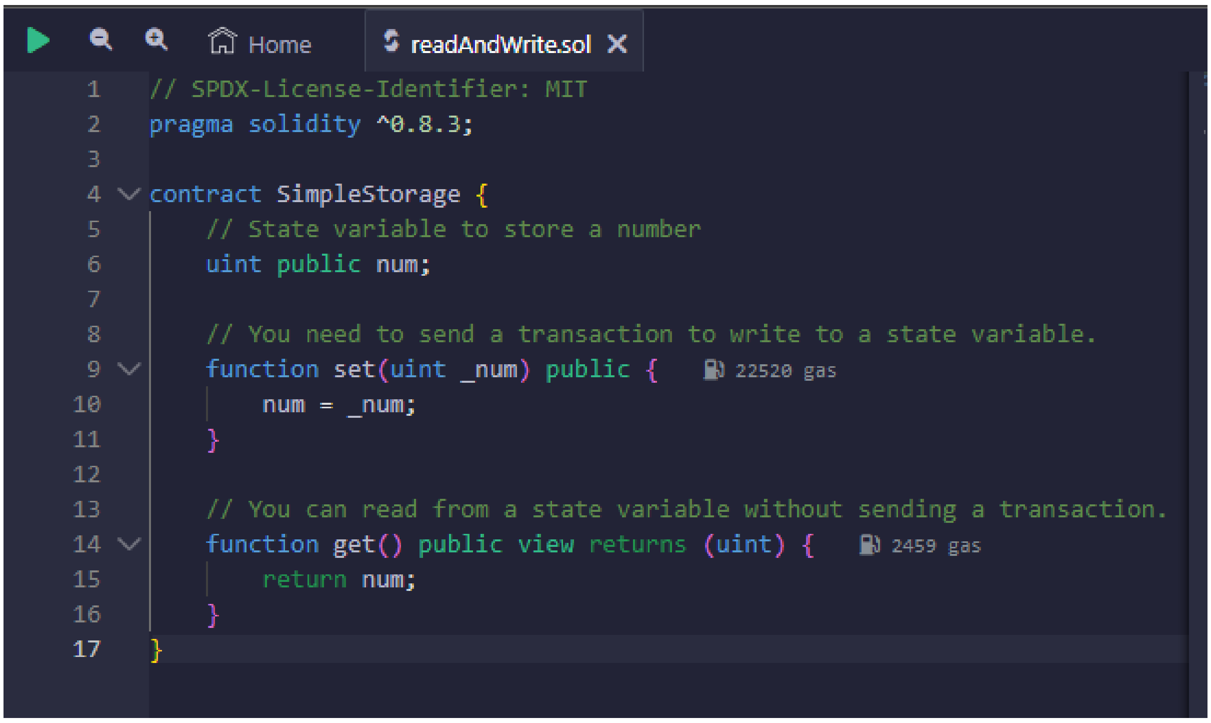
1. Create a new public state variable called `blockNumber`.
2. Inside the function `doSomething()`, assign the value of the current block number to the state variable `blockNumber`.

Tip: Look into the global variables section of the Solidity documentation to find out how to read the current block number.

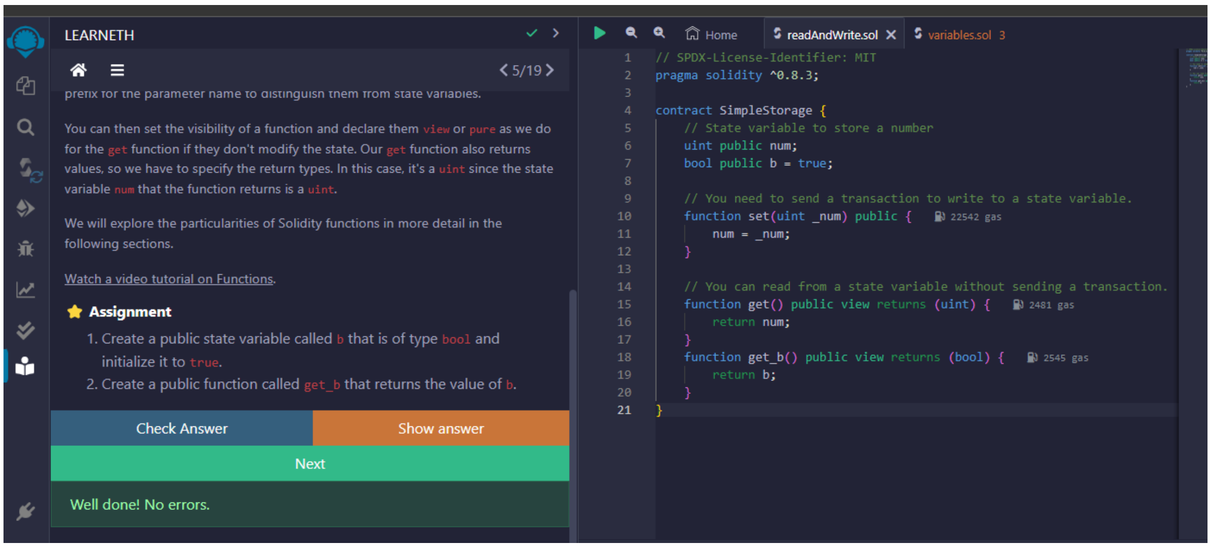
Well done! No errors.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.3;
3
4  contract Variables {
5      // State variables are stored on the blockchain.
6      string public text = "Hello";
7      uint public num = 123;
8      uint public blockNumber;
9
10     function doSomething() public {
11         // Local variables are not saved to the blockchain.
12         uint i = 456;
13
14         // Here are some global variables
15         uint timestamp = block.timestamp; // Current block timestamp
16         address sender = msg.sender; // address of the caller
17         blockNumber = block.number;
18     }
19 }
```

5.1 Functions - Reading and Writing to a State Variable



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

contract SimpleStorage {
    // State variable to store a number
    uint public num;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {    22520 gas
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {    2459 gas
        return num;
    }
}
```



prefix for the parameter name to distinguish them from state variables.

You can then set the visibility of a function and declare them `view` or `pure` as we do for the `get` function if they don't modify the state. Our `get` function also returns values, so we have to specify the return types. In this case, it's a `uint` since the state variable `num` that the function returns is a `uint`.

We will explore the particularities of Solidity functions in more detail in the following sections.

Watch a video tutorial on Functions.

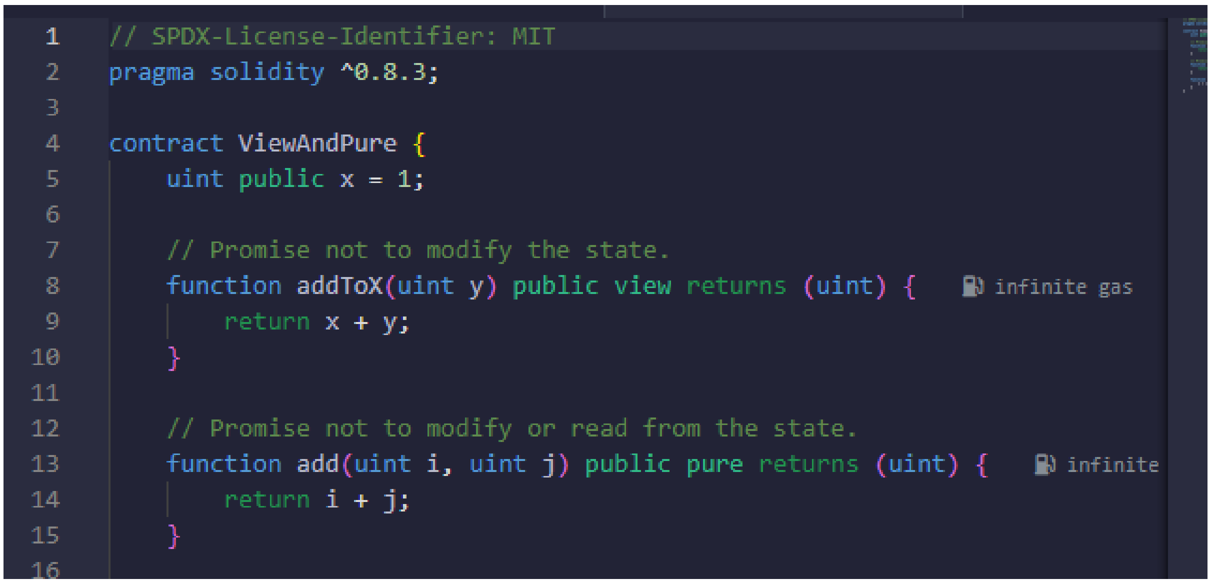
Assignment

1. Create a public state variable called `b` that is of type `bool` and initialize it to `true`.
2. Create a public function called `get_b` that returns the value of `b`.

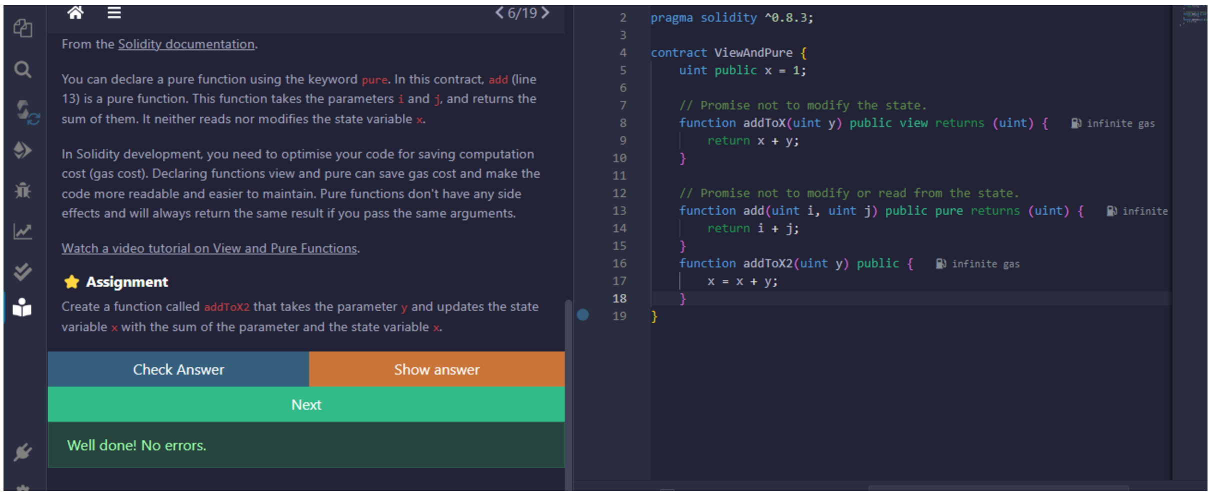
Well done! No errors.

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

contract SimpleStorage {
    // State variable to store a number
    uint public num;
    bool public b = true;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {    22542 gas
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {    2481 gas
        return num;
    }
    function get_b() public view returns (bool) {    2545 gas
        return b;
    }
}
```

5.2 Functions - View and Pure



```
 1  // SPDX-License-Identifier: MIT
 2  pragma solidity ^0.8.3;
 3
 4  contract ViewAndPure {
 5      uint public x = 1;
 6
 7      // Promise not to modify the state.
 8      function addToX(uint y) public view returns (uint) {    infinite gas
 9          return x + y;
10      }
11
12      // Promise not to modify or read from the state.
13      function add(uint i, uint j) public pure returns (uint) {    infinite
14          return i + j;
15      }
16
```



From the Solidity documentation.

You can declare a pure function using the keyword `pure`. In this contract, `add` (line 13) is a pure function. This function takes the parameters `i` and `j`, and returns the sum of them. It neither reads nor modifies the state variable `x`.

In Solidity development, you need to optimise your code for saving computation cost (gas cost). Declaring functions view and pure can save gas cost and make the code more readable and easier to maintain. Pure functions don't have any side effects and will always return the same result if you pass the same arguments.

Watch a video tutorial on View and Pure Functions.

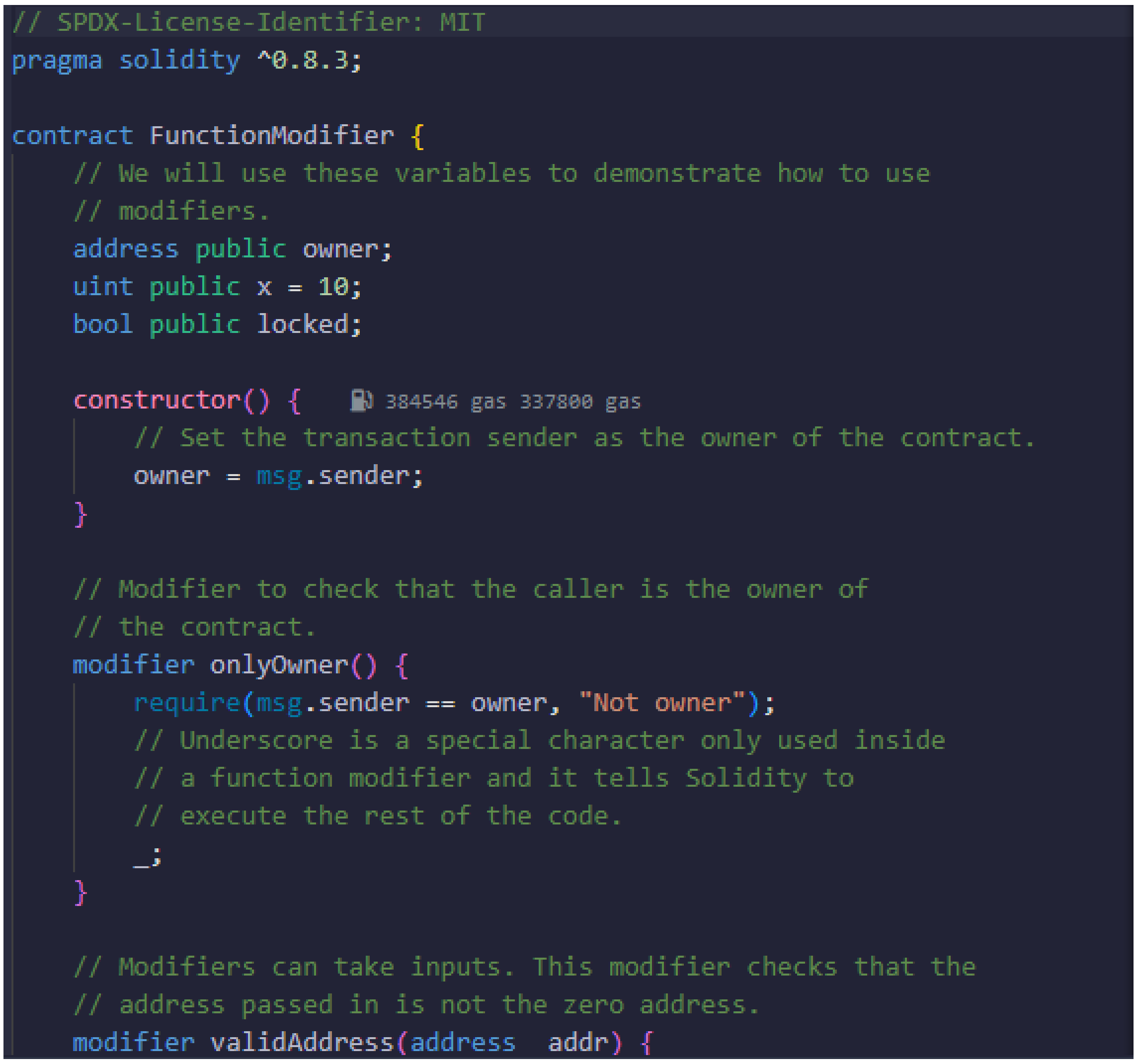
Assignment

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`.

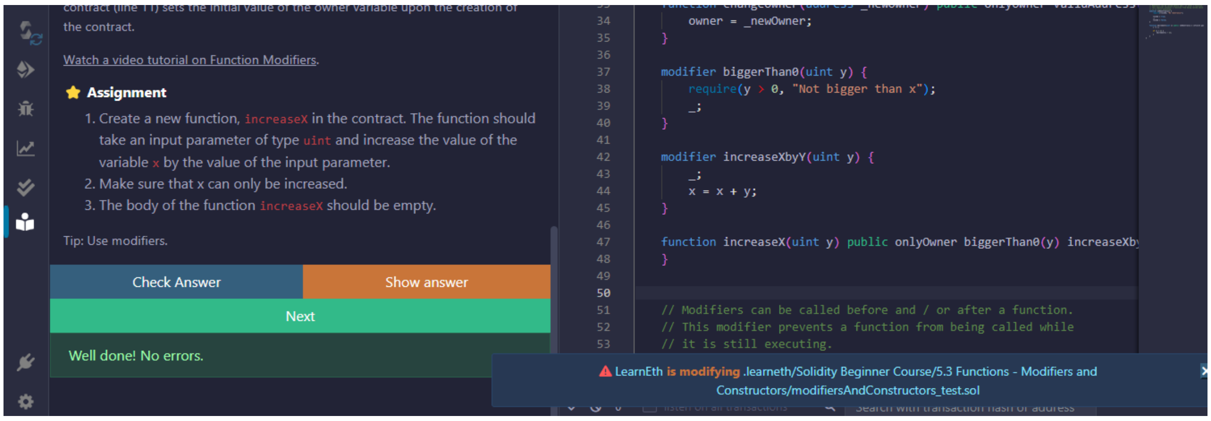
Well done! No errors.

```
 2  pragma solidity ^0.8.3;
 3
 4  contract ViewAndPure {
 5      uint public x = 1;
 6
 7      // Promise not to modify the state.
 8      function addToX(uint y) public view returns (uint) {    infinite gas
 9          return x + y;
10      }
11
12      // Promise not to modify or read from the state.
13      function add(uint i, uint j) public pure returns (uint) {    infinite
14          return i + j;
15      }
16      function addToX2(uint y) public {    infinite gas
17          x = x + y;
18      }
19  }
```

5.3 Functions - Modifiers and Constructors



```solidity
// 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) {
```



the contract.

Watch a video tutorial on Function Modifiers.

Assignment

1. 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.
2. Make sure that x can only be increased.
3. The body of the function `increaseX` should be empty.

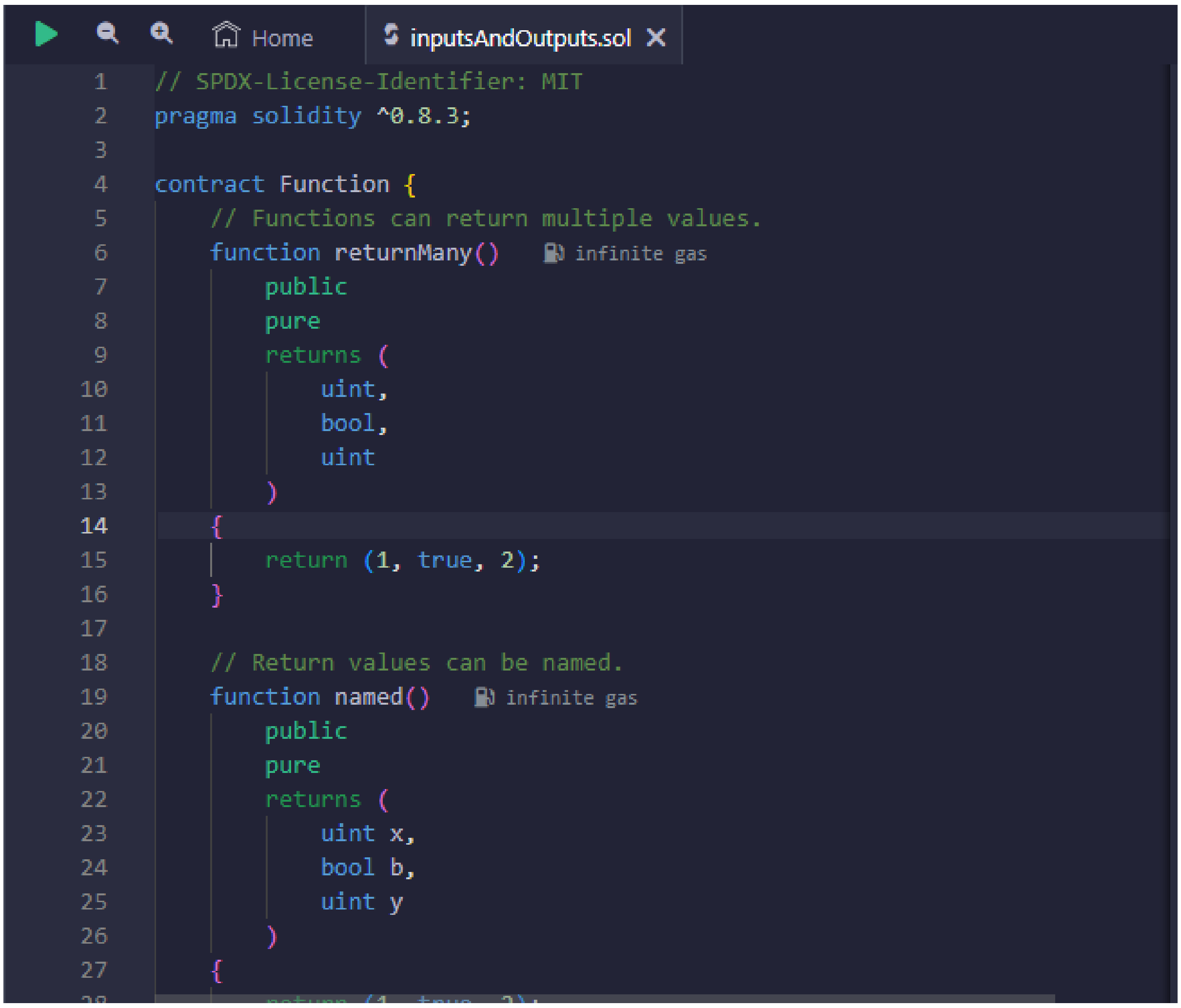
Tip: Use modifiers.

Well done! No errors.

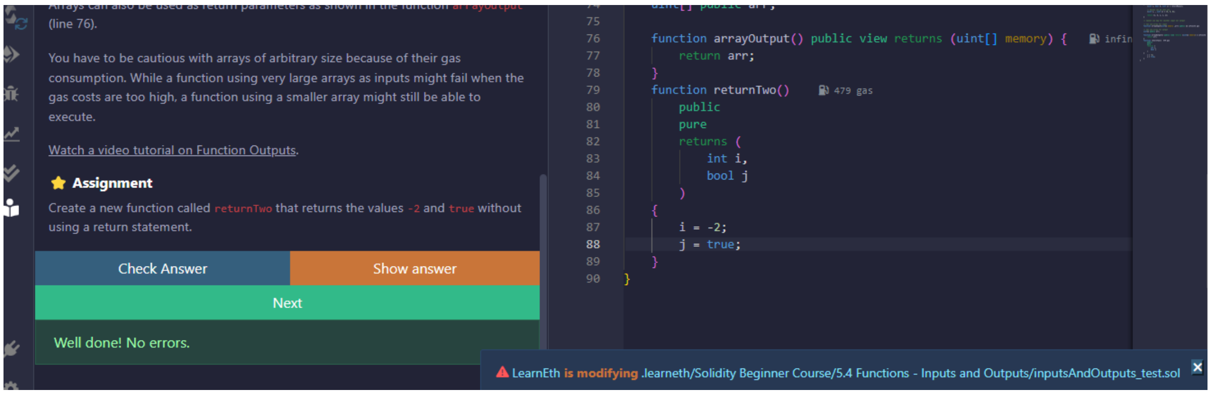
```solidity
        owner = _newOwner;
    }

    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) increaseXby
    }


    // Modifiers can be called before and / or after a function.
    // This modifier prevents a function from being called while
    // it is still executing.
```

LearnEth is modifying .learneth/Solidity Beginner Course/5.3 Functions - Modifiers and Constructors/modifiersAndConstructors_test.sol

5.4 Functions - Inputs and Outputs



```
// 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
        )
    {
        return (1, true, 2);
```



(line 76).

You have to be cautious with arrays of arbitrary size because of their gas consumption. While a function using very large arrays as inputs might fail when the gas costs are too high, a function using a smaller array might still be able to execute.

Watch a video tutorial on Function Outputs.

Assignment

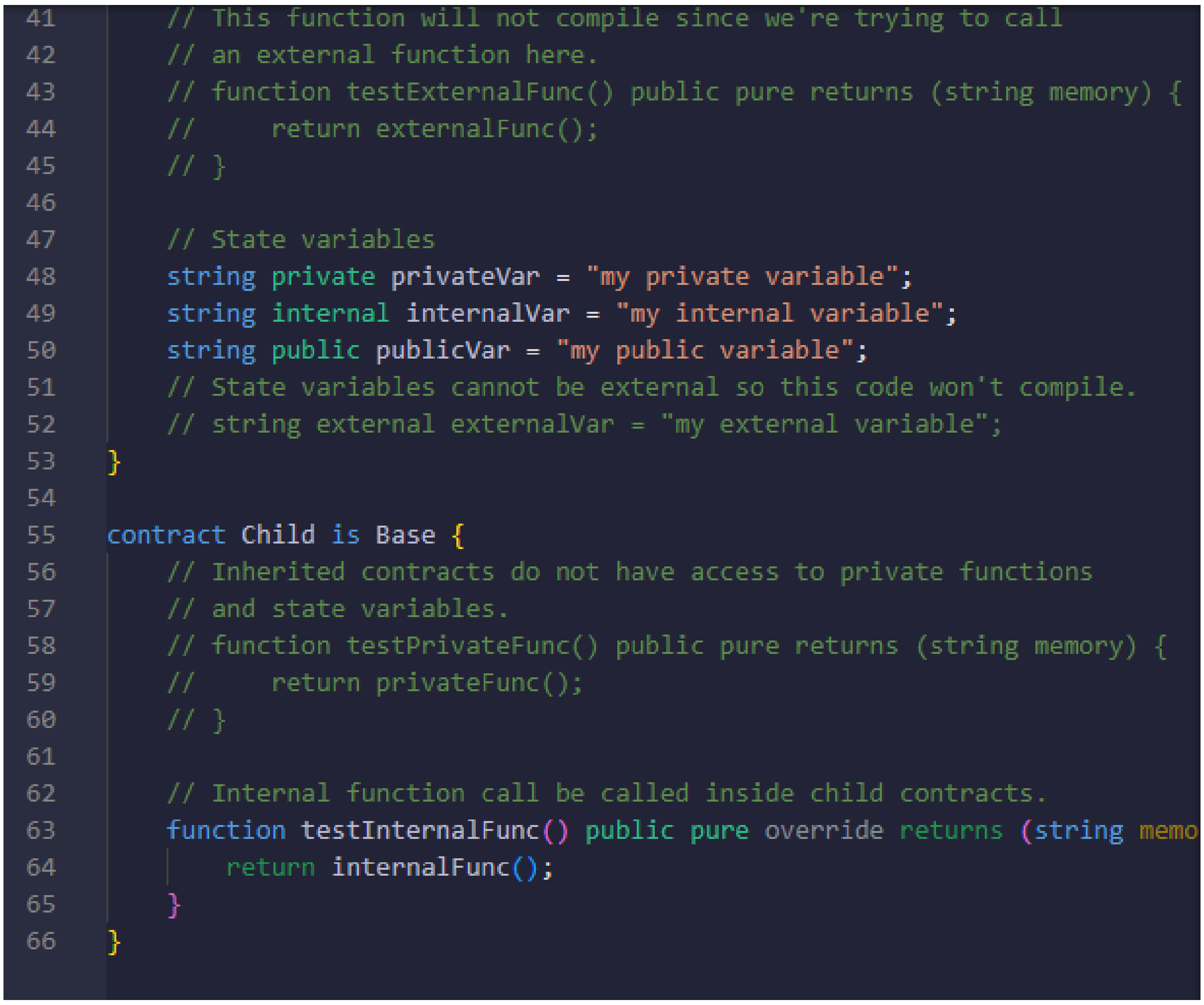
Create a new function called `returnTwo` that returns the values `-2` and `true` without using a return statement.

Well done! No errors.

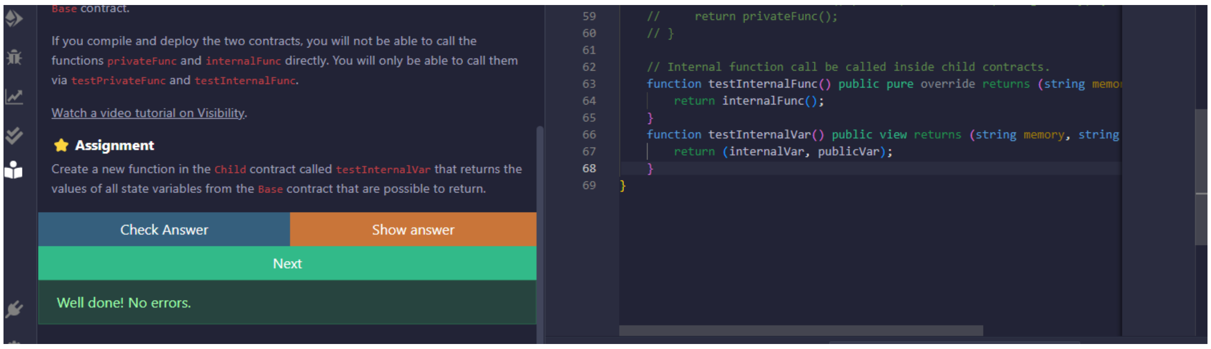
```
75
76      function arrayOutput() public view returns (uint[] memory) {    infin
77          return arr;
78      }
79      function returnTwo()     479 gas
80          public
81          pure
82          returns (
83              int i,
84              bool j
85          )
86      {
87          i = -2;
88          j = true;
89      }
90  }
```

LearnEth is modifying .learneth/Solidity Beginner Course/5.4 Functions - Inputs and Outputs/inputsAndOutputs_test.sol

6. Visibility



```
    // This function will not compile since we're trying to call
    // an external function here.
    // function testExternalFunc() public pure returns (string memory) {
    //     return externalFunc();
    // }

    // State variables
    string private privateVar = "my private variable";
    string internal internalVar = "my internal variable";
    string public publicVar = "my public variable";
    // State variables cannot be external so this code won't compile.
    // string external externalVar = "my external variable";
}

contract Child is Base {
    // Inherited contracts do not have access to private functions
    // and state variables.
    // function testPrivateFunc() public pure returns (string memory) {
    //     return privateFunc();
    // }

    // Internal function call be called inside child contracts.
    function testInternalFunc() public pure override returns (string memo
        return internalFunc();
    }
}
```



Base contract.

If you compile and deploy the two contracts, you will not be able to call the functions `privateFunc` and `internalFunc` directly. You will only be able to call them via `testPrivateFunc` and `testInternalFunc`.

Watch a video tutorial on Visibility.

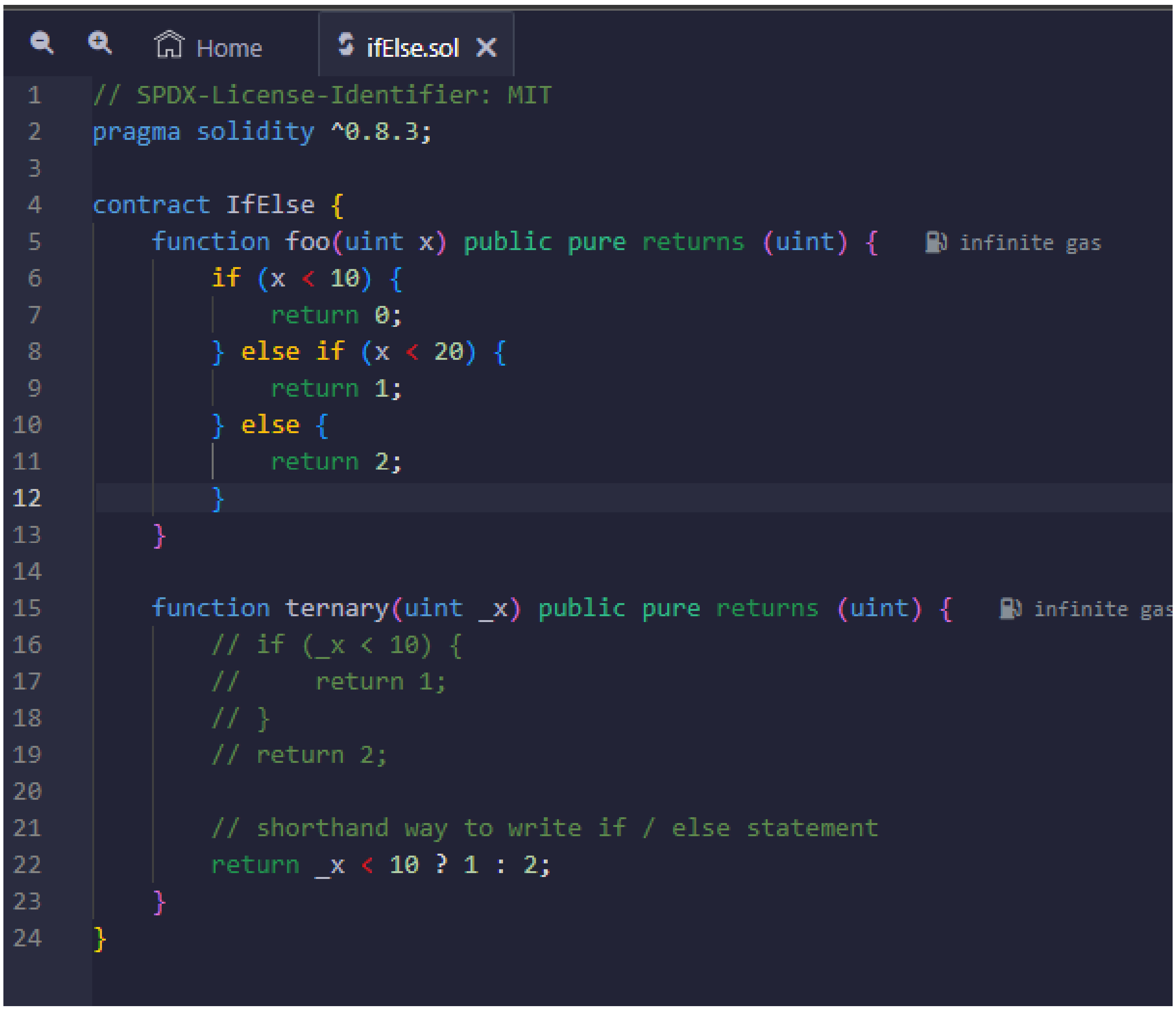
⭐ **Assignment**

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.

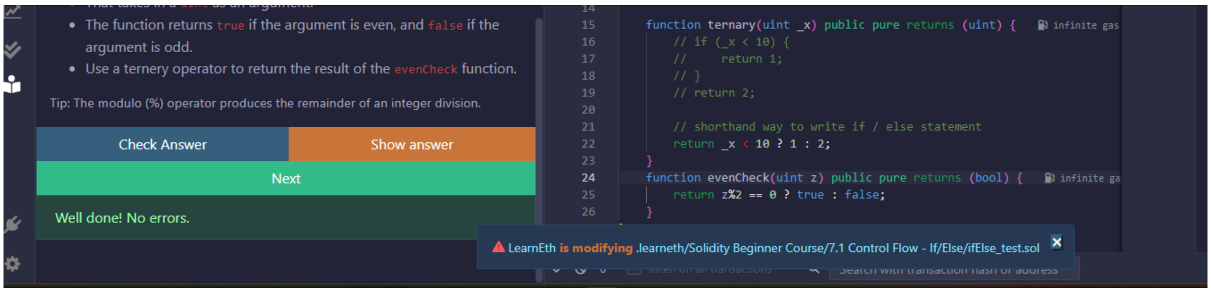
Well done! No errors.

```
    //     return privateFunc();
    // }

    // Internal function call be called inside child contracts.
    function testInternalFunc() public pure override returns (string memo
        return internalFunc();
    }
    function testInternalVar() public view returns (string memory, string
        return (internalVar, publicVar);
    }
}
```

7.1 Control Flow - If/Else



```solidity
// 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;
    }
}
```



- 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.

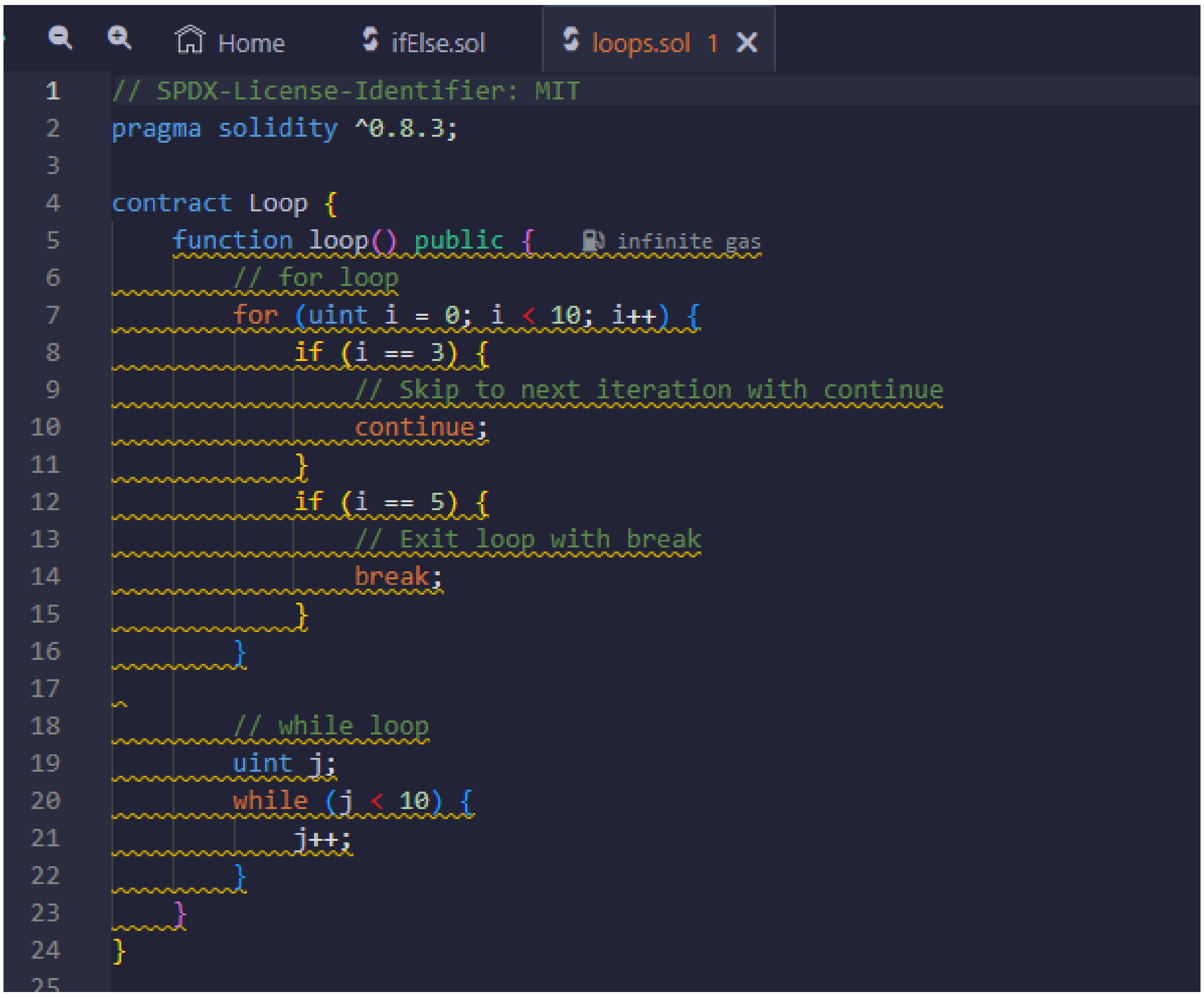
Tip: The modulo (%) operator produces the remainder of an integer division.

Well done! No errors.

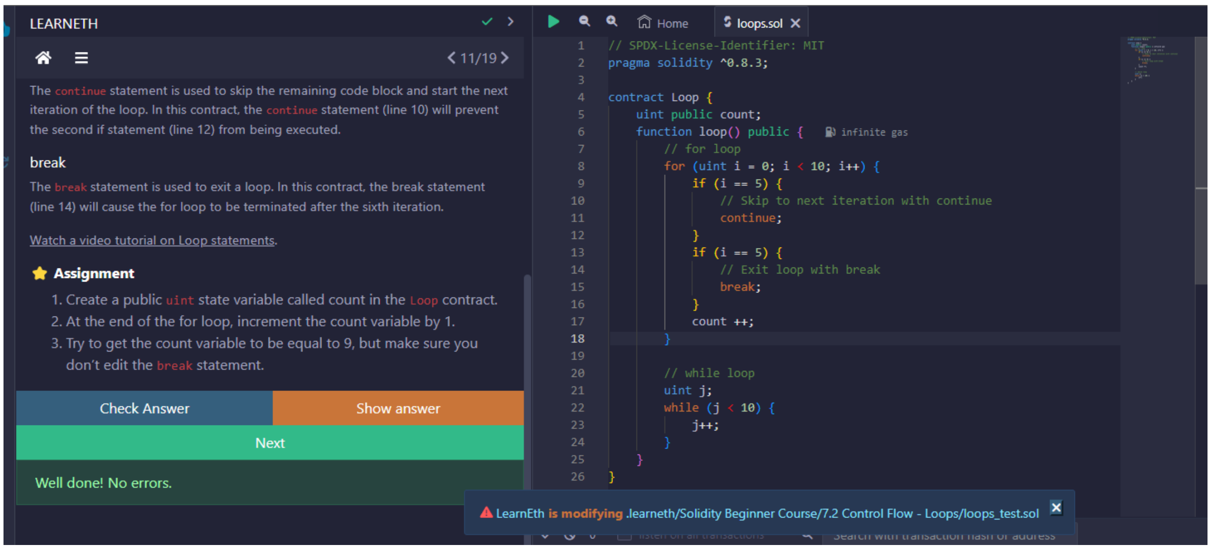
```solidity
    function ternary(uint _x) public pure returns (uint) {      infinite gas
        // if (_x < 10) {
        //     return 1;
        // }
        // return 2;

        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }
    function evenCheck(uint z) public pure returns (bool) {      infinite ga
        return z%2 == 0 ? true : false;
    }
```

LearnEth is modifying .learneth/Solidity Beginner Course/7.1 Control Flow - If/Else/ifElse_test.sol

7.2 Control Flow - Loops



```
 1  // SPDX-License-Identifier: MIT
 2  pragma solidity ^0.8.3;
 3
 4  contract Loop {
 5      function loop() public {    infinite gas
 6          // for loop
 7          for (uint i = 0; i < 10; i++) {
 8              if (i == 3) {
 9                  // Skip to next iteration with continue
10                  continue;
11              }
12              if (i == 5) {
13                  // Exit loop with break
14                  break;
15              }
16          }
17
18          // while loop
19          uint j;
20          while (j < 10) {
21              j++;
22          }
23      }
24  }
25
```



The continue statement is used to skip the remaining code block and start the next iteration of the loop. In this contract, the continue statement (line 10) will prevent the second if statement (line 12) from being executed.

break

The break statement is used to exit a loop. In this contract, the break statement (line 14) will cause the for loop to be terminated after the sixth iteration.

Watch a video tutorial on Loop statements.

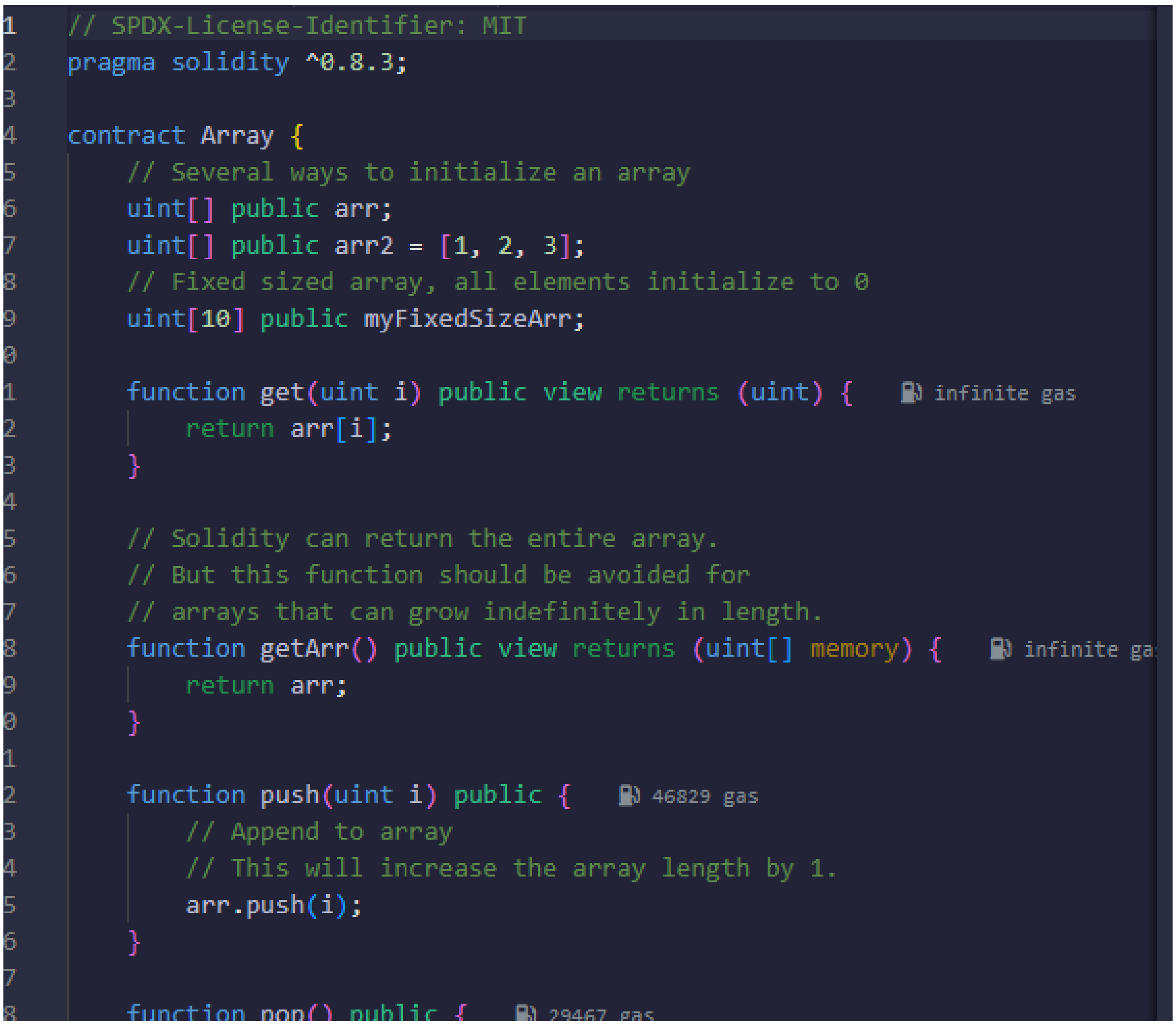
Assignment

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

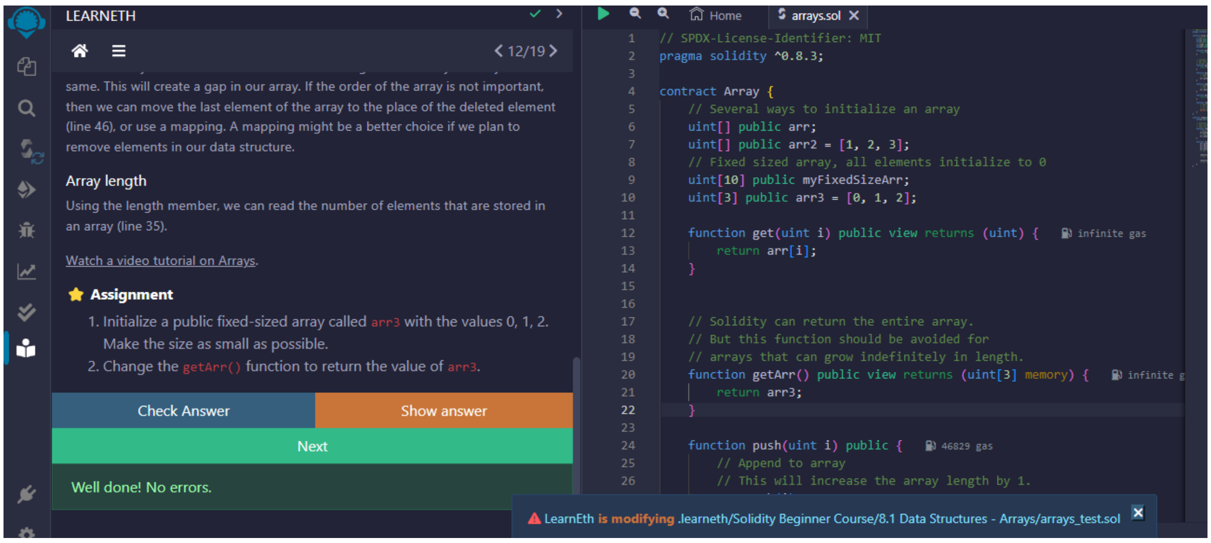
Well done! No errors.

```
 1  // SPDX-License-Identifier: MIT
 2  pragma solidity ^0.8.3;
 3
 4  contract Loop {
 5      uint public count;
 6      function loop() public {    infinite gas
 7          // for loop
 8          for (uint i = 0; i < 10; i++) {
 9              if (i == 5) {
10                  // Skip to next iteration with continue
11                  continue;
12              }
13              if (i == 5) {
14                  // Exit loop with break
15                  break;
16              }
17              count ++;
18          }
19
20          // while loop
21          uint j;
22          while (j < 10) {
23              j++;
24          }
25      }
26  }
```

LearnEth is modifying .learneth/Solidity Beginner Course/7.2 Control Flow - Loops/loops_test.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
```



same. This will create a gap in our array. If the order of the array is not important, then we can move the last element of the array to the place of the deleted element (line 46), or use a mapping. A mapping might be a better choice if we plan to remove elements in our data structure.

Array length

Using the length member, we can read the number of elements that are stored in an array (line 35).

Watch a video tutorial on Arrays.

Assignment

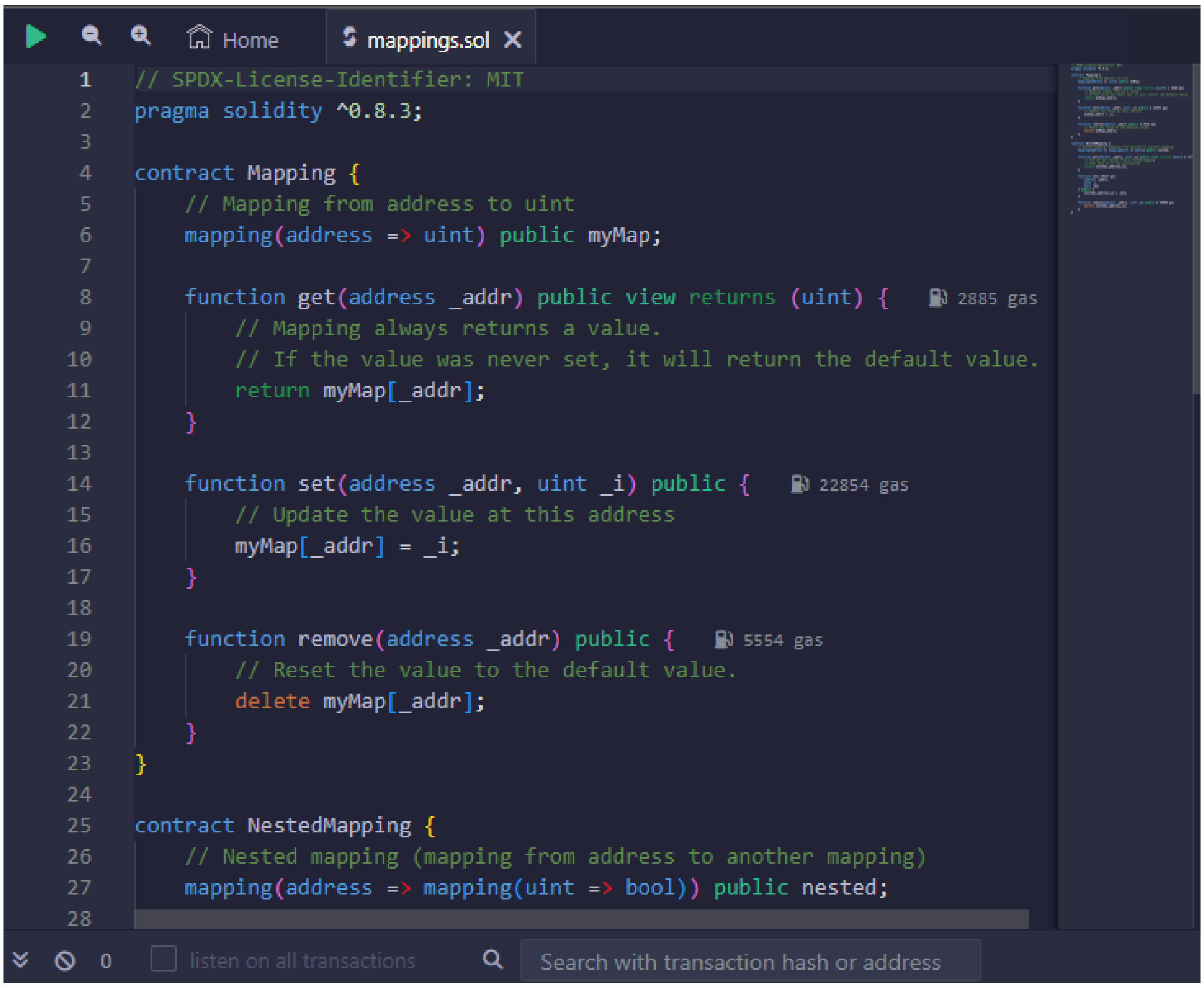
1. Initialize a public fixed-sized array called `arr3` with the values 0, 1, 2. Make the size as small as possible.
2. Change the `getArr()` function to return the value of `arr3`.

Well done! No errors.

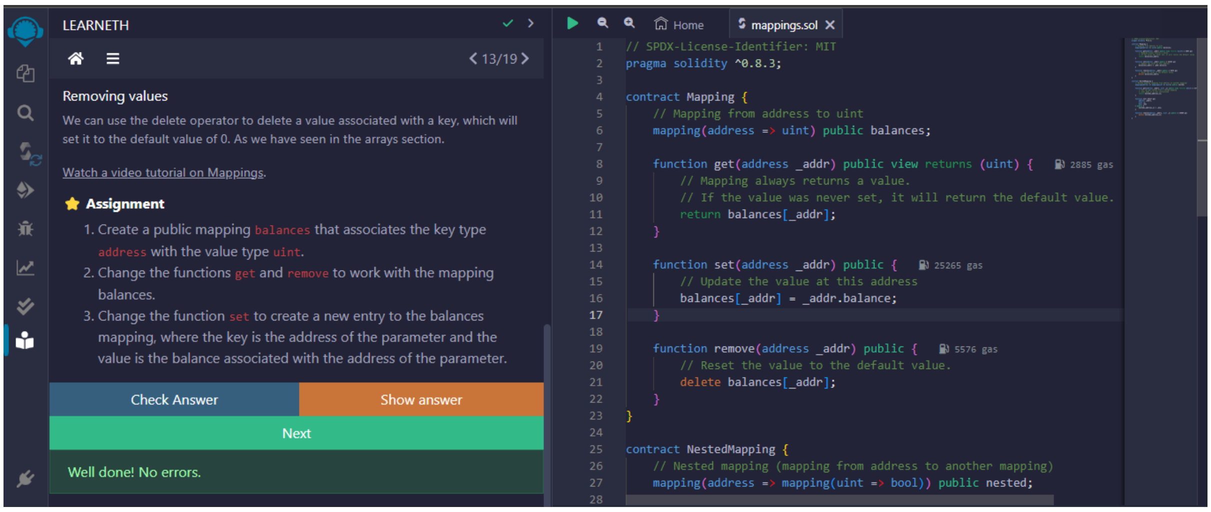
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

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

    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[3] memory) {    infinite g
        return arr3;
    }

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

LearnEth is modifying .learneth/Solidity Beginner Course/8.1 Data Structures - Arrays/arrays_test.sol

8.2 Data Structures - Mappings


Home
mappings.sol
2885 gas
22854 gas
5554 gas
0
listen on all transactions
Search with transaction hash or address

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

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

    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 myMap[_addr];
    }

    function set(address _addr, uint _i) public {
        // Update the value at this address
        myMap[_addr] = _i;
    }

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

contract NestedMapping {
    // Nested mapping (mapping from address to another mapping)
    mapping(address => mapping(uint => bool)) public nested;
```



Removing values

We can use the delete operator to delete a value associated with a key, which will set it to the default value of 0. As we have seen in the arrays section.

Watch a video tutorial on Mappings.

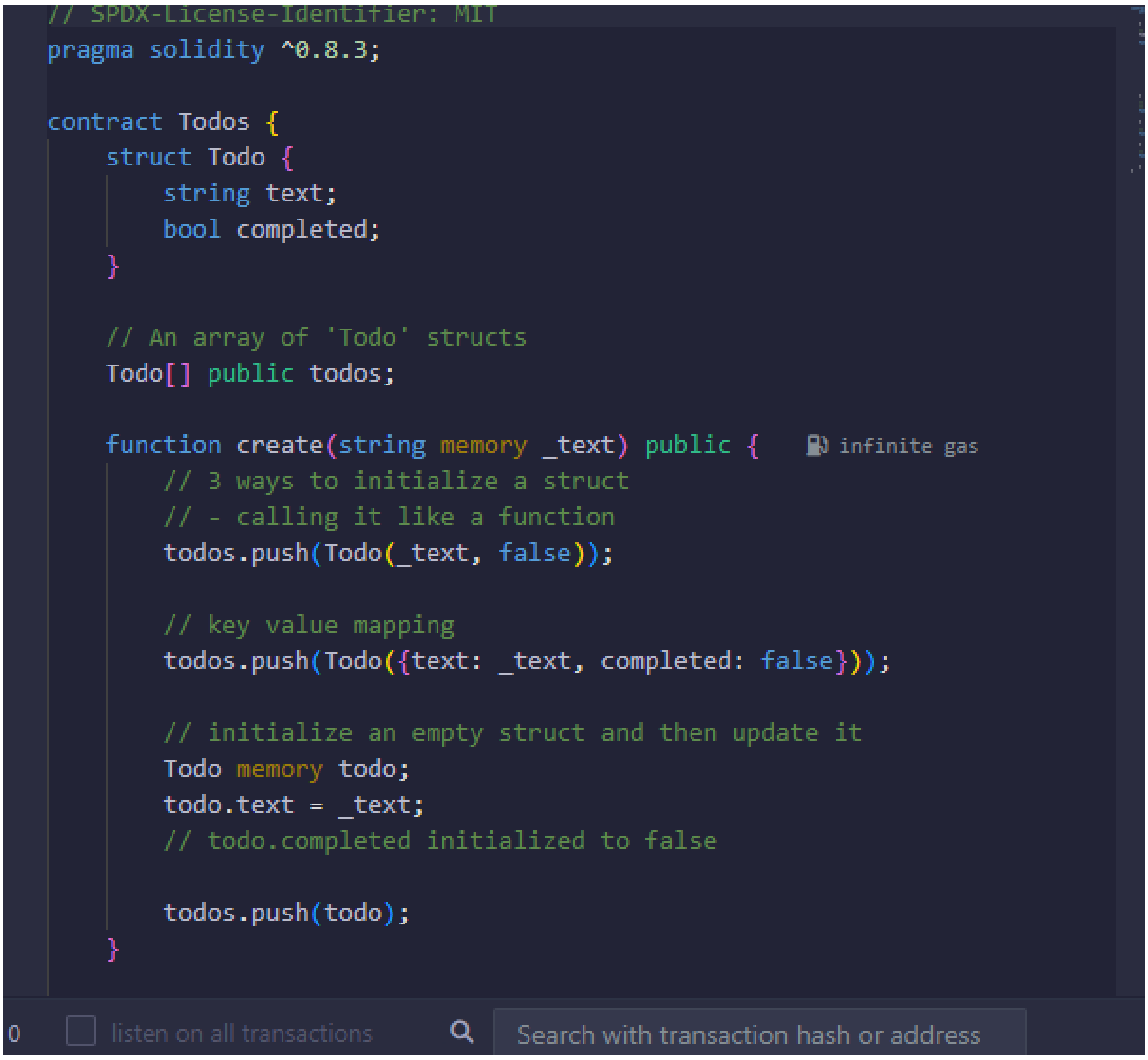
Assignment

1. Create a public mapping `balances` that associates the key type `address` with the value type `uint`.
2. Change the functions `get` and `remove` to work with the mapping balances.
3. 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.

Well done! No errors.

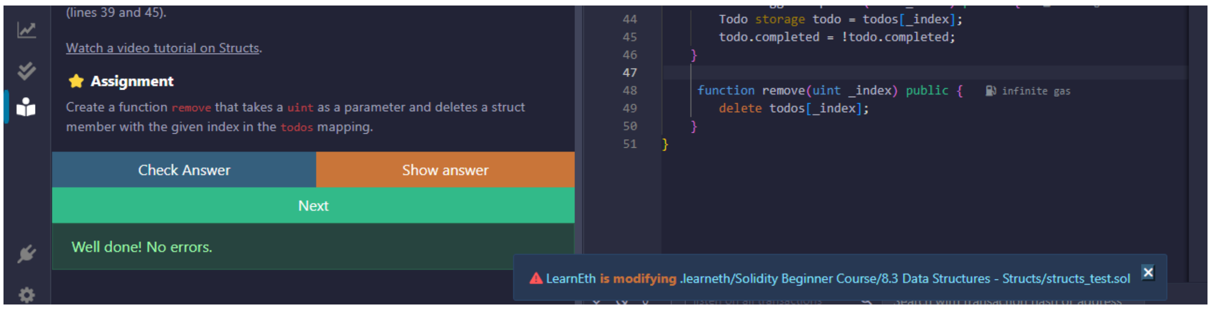
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

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];
    }
}

contract NestedMapping {
    // Nested mapping (mapping from address to another mapping)
    mapping(address => mapping(uint => bool)) public nested;
```

8.3 Data Structures - Structs



```
// 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



(lines 39 and 45).

Watch a video tutorial on Structs.

Assignment

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

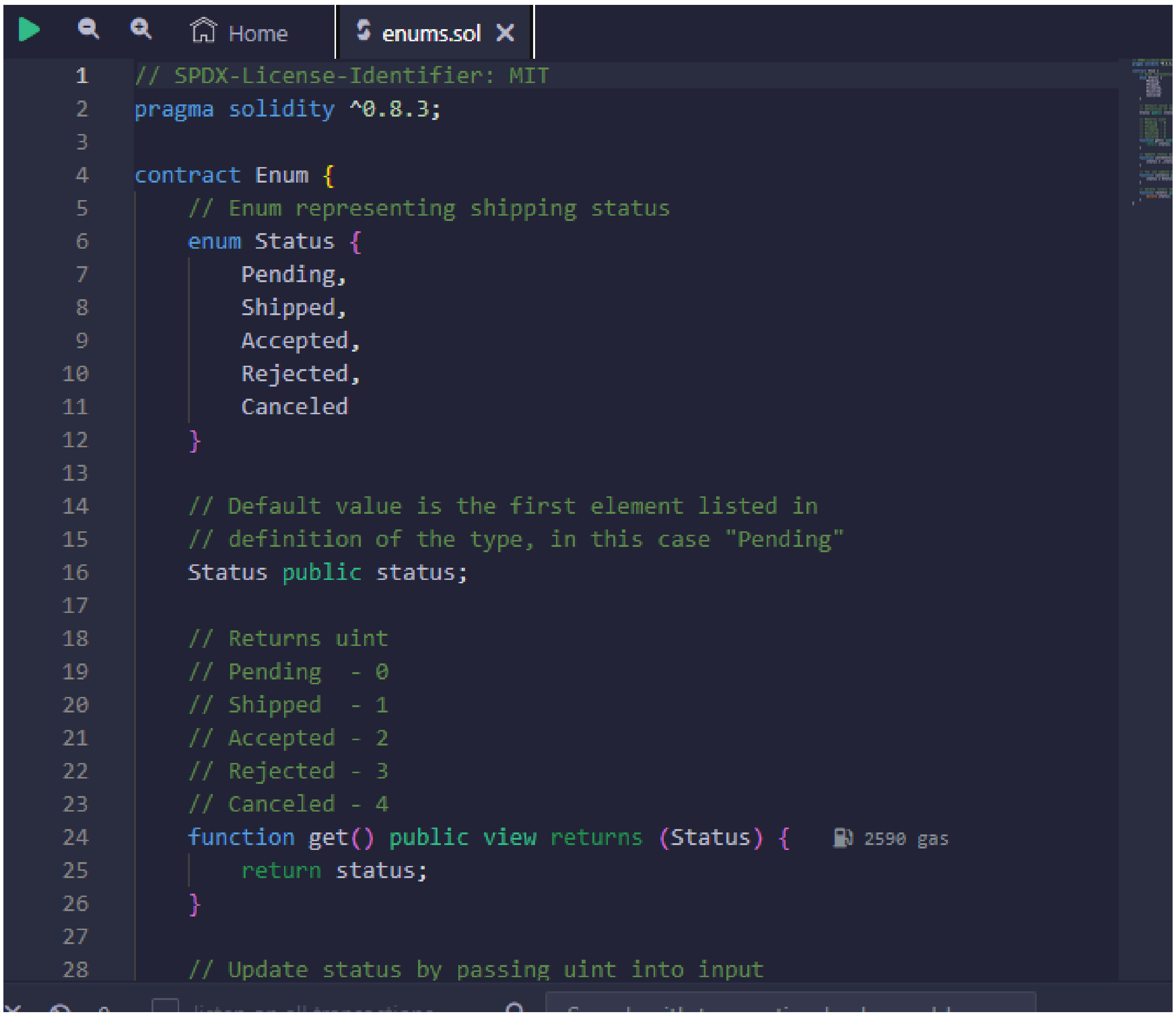
Check Answer
Show answer
Next

Well done! No errors.

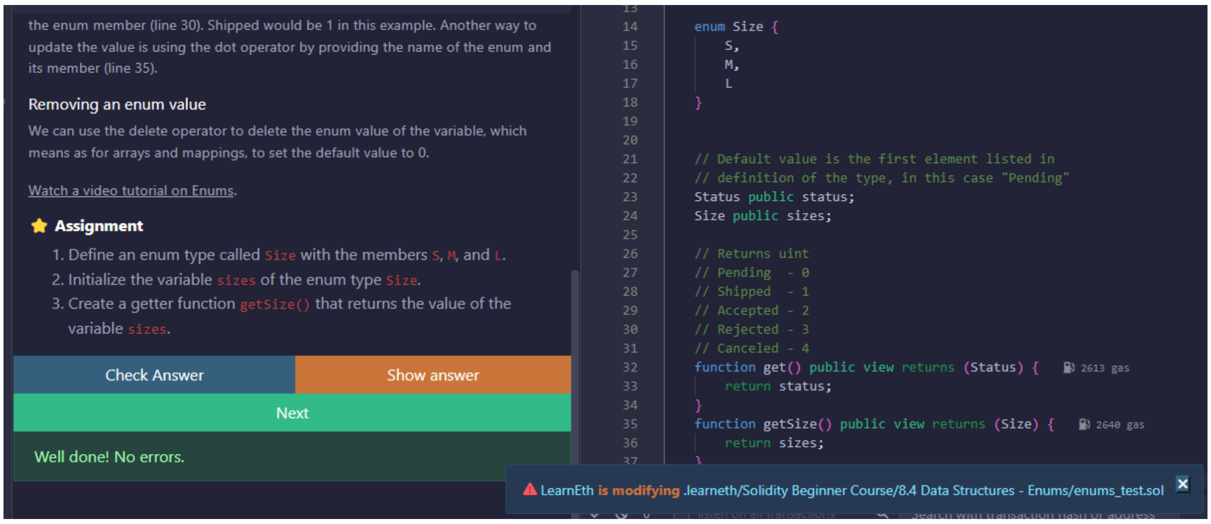
```
        Todo storage todo = todos[_index];
        todo.completed = !todo.completed;
    }

    function remove(uint _index) public {    infinite gas
        delete todos[_index];
    }
}
```

LearnEth is modifying .learneth/Solidity Beginner Course/8.3 Data Structures - Structs/structs_test.sol

8.4 Data Structures - Enums



```solidity
// 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

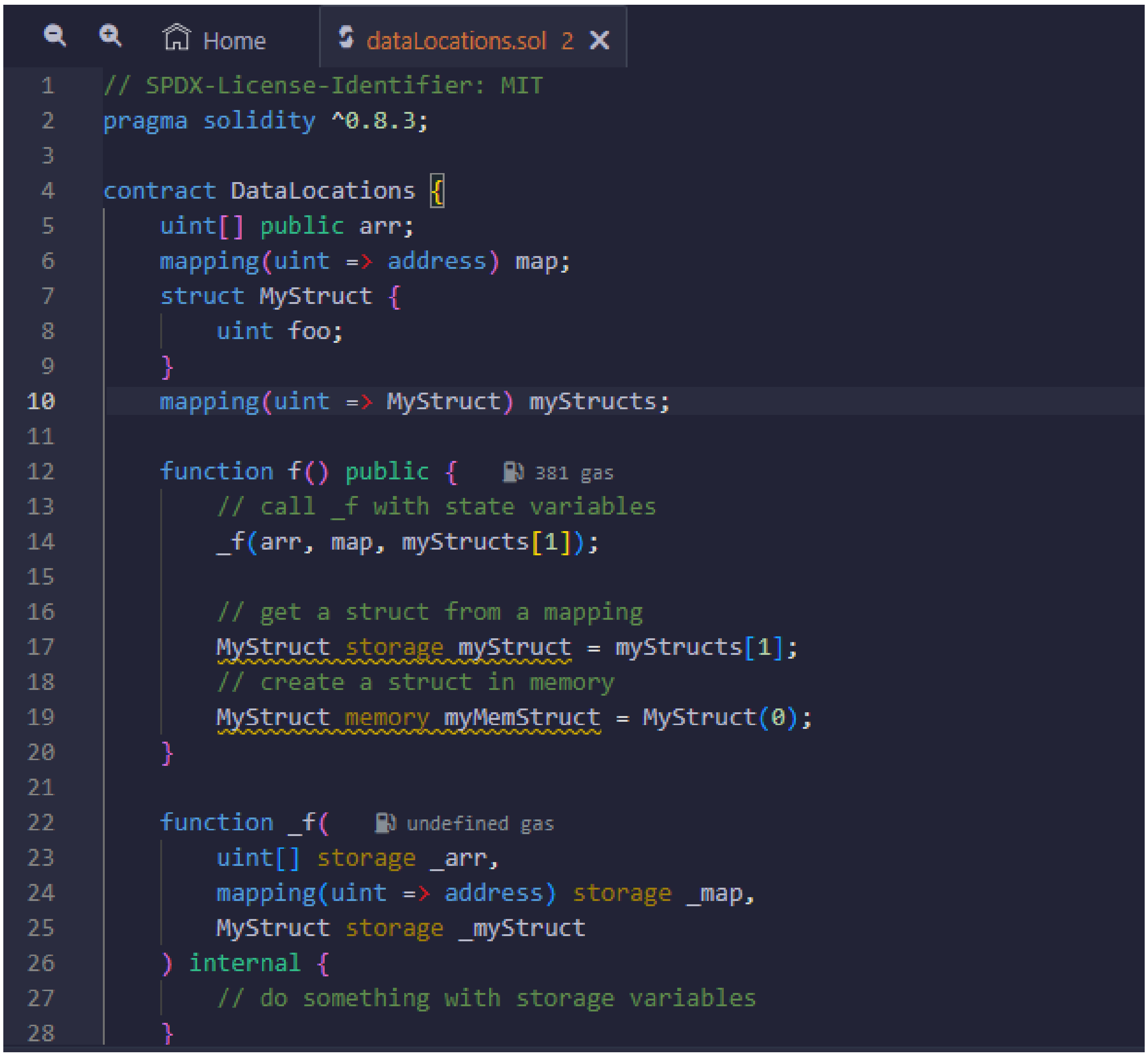
1. Define an enum type called `Size` with the members `S`, `M`, and `L`.
2. Initialize the variable `sizes` of the enum type `Size`.
3. Create a getter function `getSize()` that returns the value of the variable `sizes`.

Well done! No errors.

```solidity
    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;
    }
```

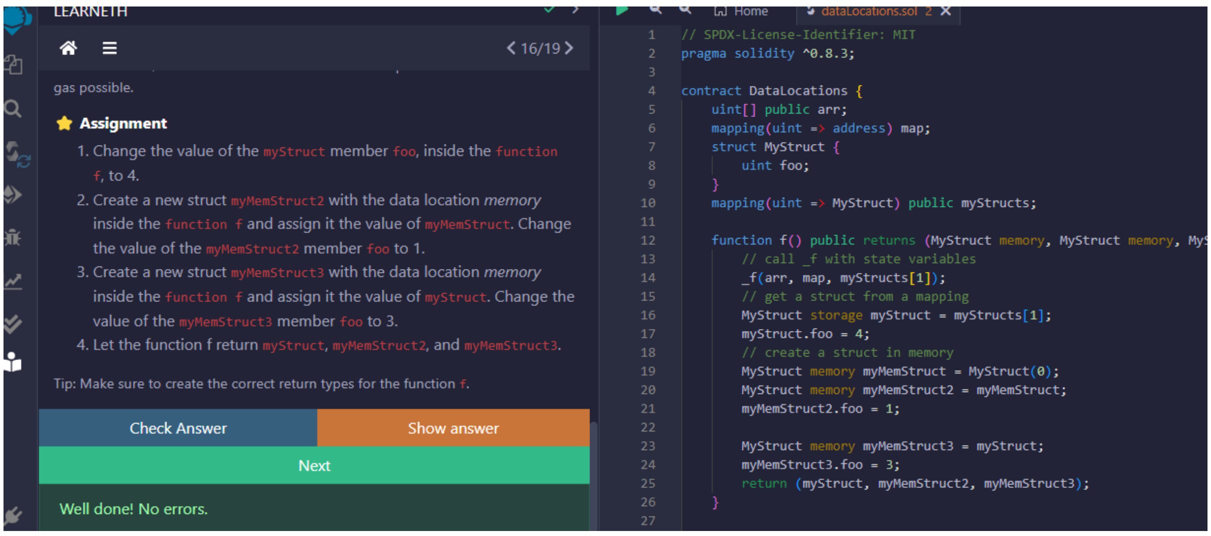
LearnEth is modifying .learneth/Solidity Beginner Course/8.4 Data Structures - Enums/enums_test.sol

9. Data Locations



Home | dataLocations.sol 2

```solidity
// 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

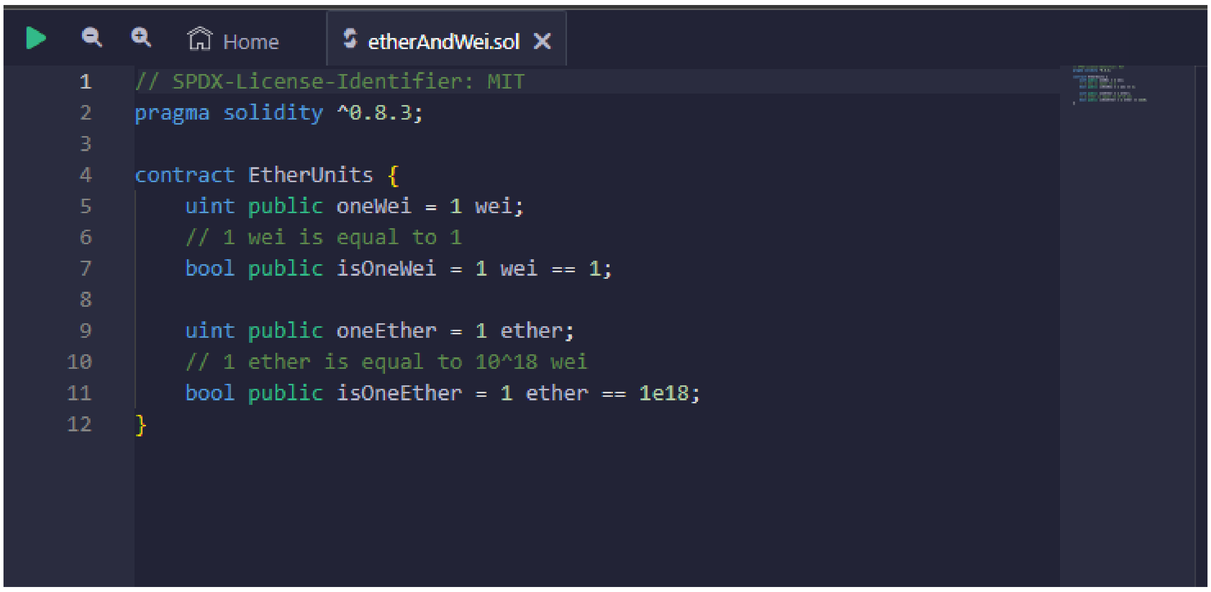
Next

Well done! No errors.

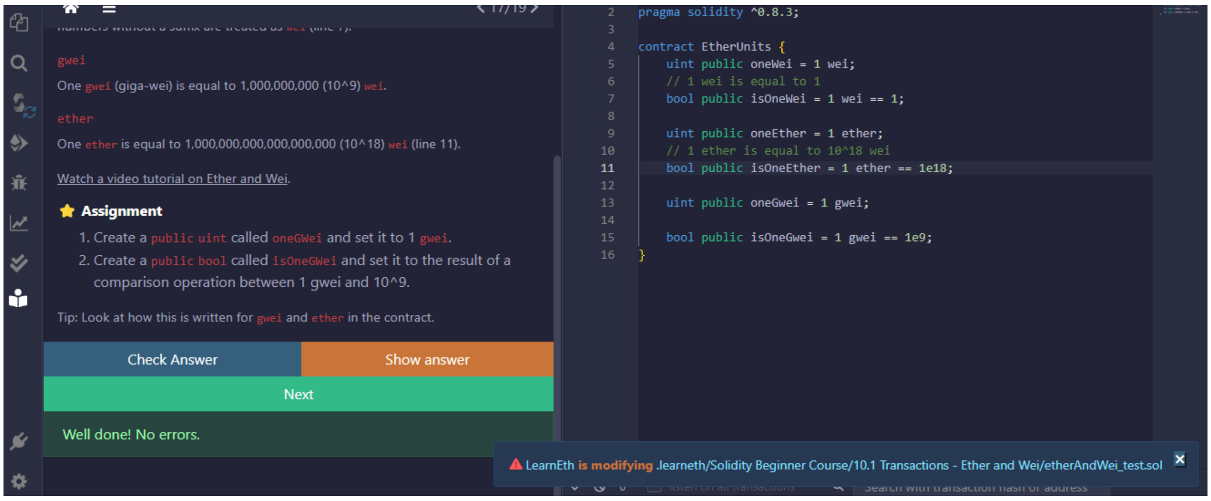
Home | dataLocations.sol 2

```solidity
// 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, My
        // 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



```
 1  // SPDX-License-Identifier: MIT
 2  pragma solidity ^0.8.3;
 3
 4  contract EtherUnits {
 5      uint public oneWei = 1 wei;
 6      // 1 wei is equal to 1
 7      bool public isOneWei = 1 wei == 1;
 8
 9      uint public oneEther = 1 ether;
10      // 1 ether is equal to 10^18 wei
11      bool public isOneEther = 1 ether == 1e18;
12  }
```



gwei

One gwei (giga-wei) is equal to 1,000,000,000 (10^9) wei.

ether

One ether is equal to 1,000,000,000,000,000,000 (10^18) wei (line 11).

Watch a video tutorial on Ether and Wei.

⭐ **Assignment**

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

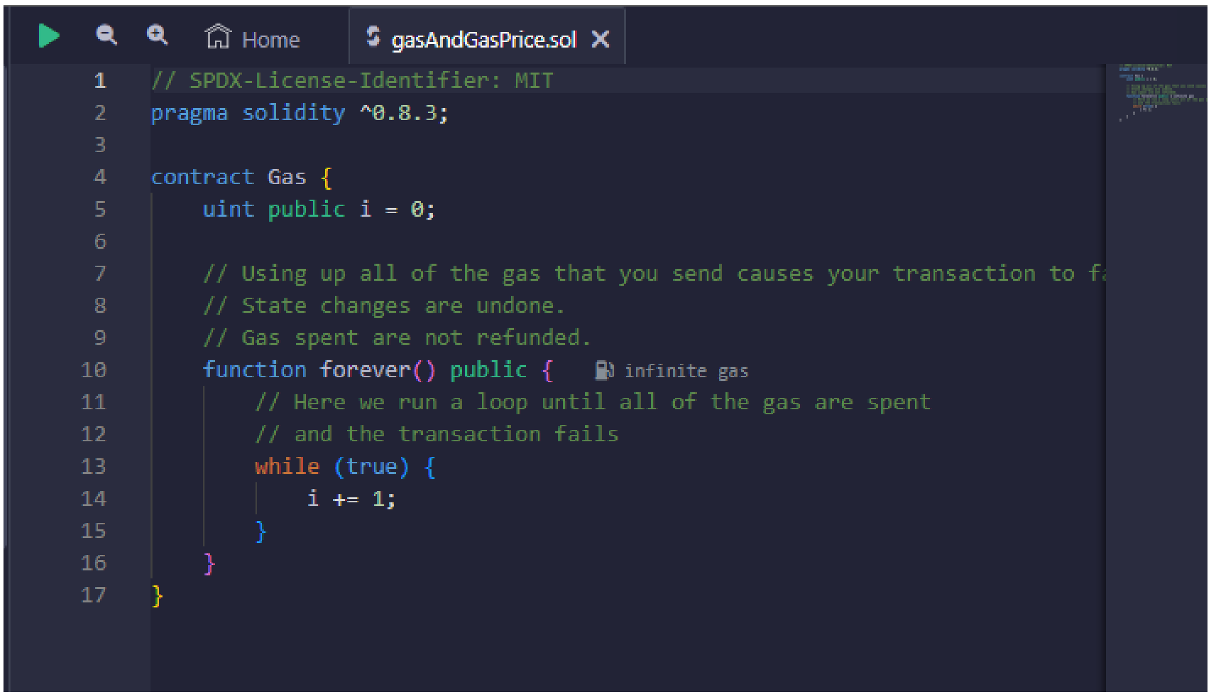
Tip: Look at how this is written for gwei and ether in the contract.

Well done! No errors.

```
 2  pragma solidity ^0.8.3;
 3
 4  contract EtherUnits {
 5      uint public oneWei = 1 wei;
 6      // 1 wei is equal to 1
 7      bool public isOneWei = 1 wei == 1;
 8
 9      uint public oneEther = 1 ether;
10      // 1 ether is equal to 10^18 wei
11      bool public isOneEther = 1 ether == 1e18;
12
13      uint public oneGwei = 1 gwei;
14
15      bool public isOneGwei = 1 gwei == 1e9;
16  }
```

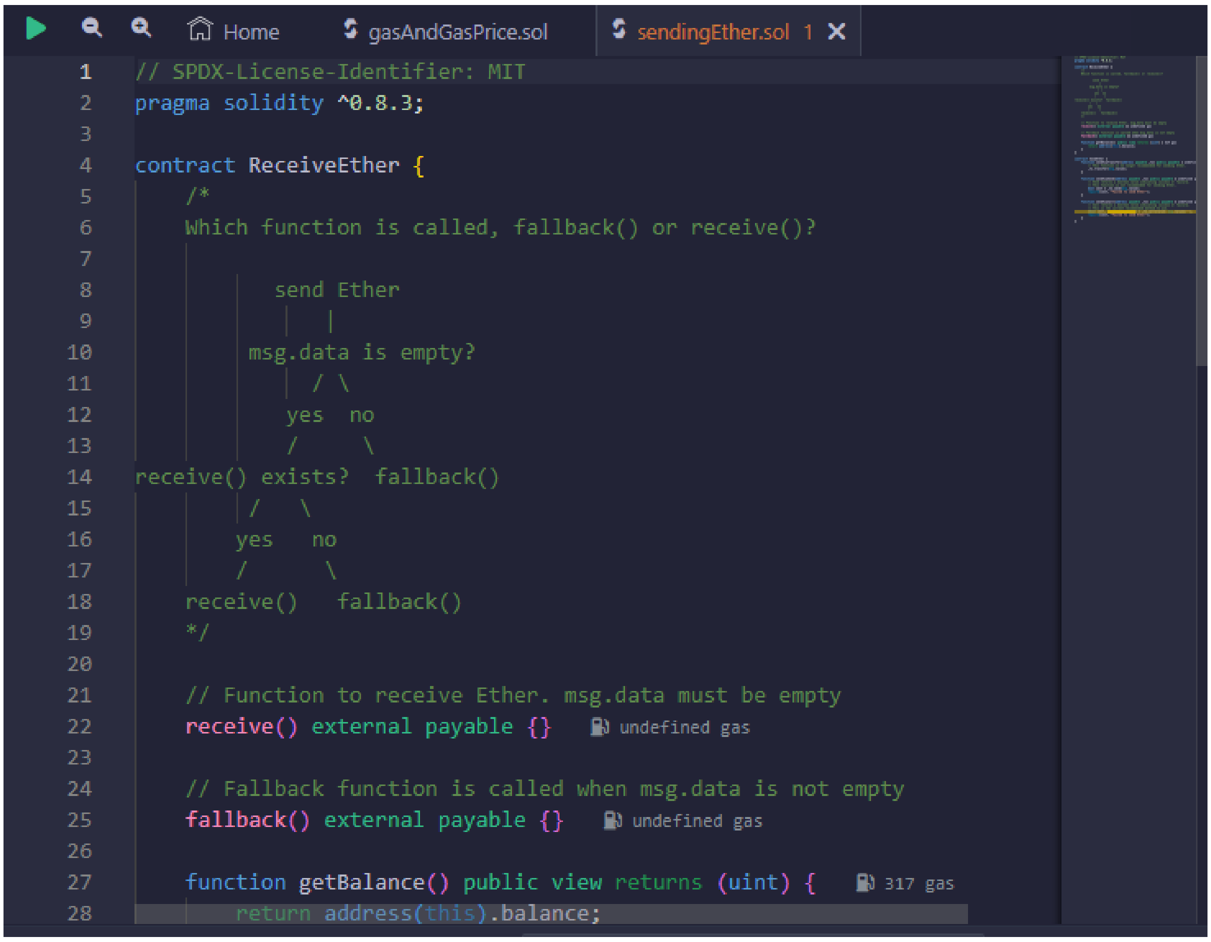
LearnEth is modifying .learneth/Solidity Beginner Course/10.1 Transactions - Ether and Wei/etherAndWei_test.sol

10.2 Transactions - Gas and Gas Price

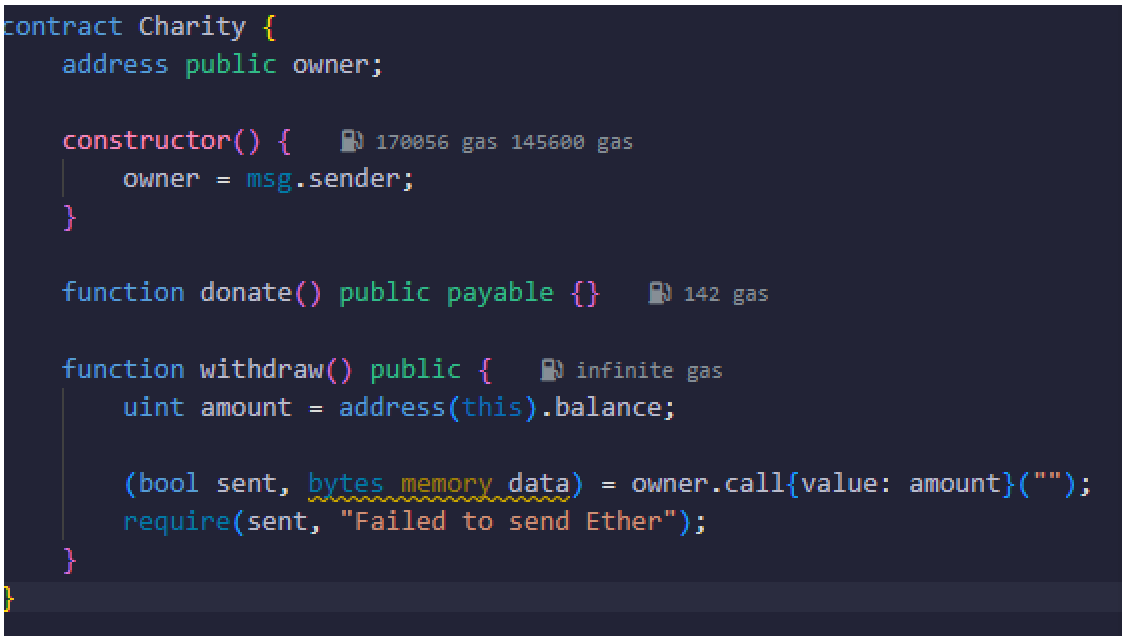


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Gas {
    uint public i = 0;

    // Using up all of the gas that you send causes your transaction to fa
    // State changes are undone.
    // Gas spent are not refunded.
    function forever() public {    infinite gas
        // Here we run a loop until all of the gas are spent
        // and the transaction fails
        while (true) {
            i += 1;
        }
    }
}
```

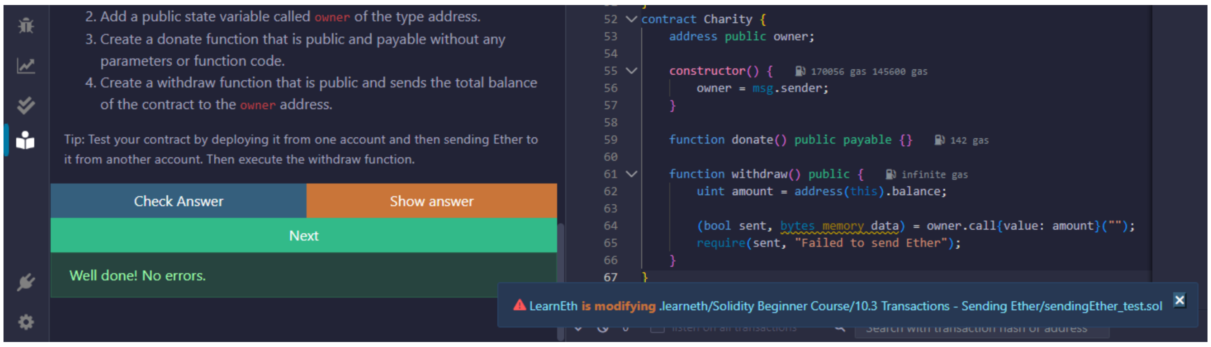
10.3 Transactions - Sending Ether



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

contract ReceiveEther {
    /*
    Which function is called, fallback() or receive()?

            send Ether
                |
          msg.data is empty?
               / \
             yes  no
             /      \
receive() exists?  fallback()
         /   \
        yes   no
        /      \
    receive()   fallback()
    */

    // Function to receive Ether. msg.data must be empty
    receive() external payable {}    undefined gas

    // Fallback function is called when msg.data is not empty
    fallback() external payable {}    undefined gas

    function getBalance() public view returns (uint) {    317 gas
        return address(this).balance;
```



```
contract Charity {
    address public owner;

    constructor() {    170056 gas 145600 gas
        owner = msg.sender;
    }

    function donate() public payable {}    142 gas

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

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



2. Add a public state variable called owner of the type address.
3. Create a donate function that is public and payable without any parameters or function code.
4. Create a withdraw function that is public and sends the total balance of the contract to the owner address.

Tip: Test your contract by deploying it from one account and then sending Ether to it from another account. Then execute the withdraw function.

Well done! No errors.

```
contract Charity {
    address public owner;

    constructor() {    170056 gas 145600 gas
        owner = msg.sender;
    }

    function donate() public payable {}    142 gas

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

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

LearnEth is modifying .learneth/Solidity Beginner Course/10.3 Transactions - Sending Ether/sendingEther_test.sol