Experiment No. 4

Name: Atharva Prabhu Batch: B

Div: D20A Roll No: 45

Aim: Hands on Solidity Programming Assignments for creating Smart Contracts

Theory:

1. Primitive Data Types, Variables, Functions – pure, view

In Solidity, primitive data types form the foundation of smart contract development. Commonly used types include:

- **uint / int**: unsigned and signed integers of different sizes (e.g., uint256, int128).
- **bool**: represents logical values (true or false).
- **address**: holds a 20-byte Ethereum account address, often used for storing user accounts or contract addresses.
- **bytes / string**: store binary data or textual data.

Variables in Solidity can be **state variables** (stored on the blockchain permanently), **local variables** (temporary, created during function execution), or **global variables** (special predefined variables such as msg.sender, msg.value, and block.timestamp).

Functions allow execution of contract logic. Special types of functions include:

- **pure**: cannot read or modify blockchain state; they work only with inputs and internal computations.
- **view**: can read state variables but cannot alter them. This classification helps optimize gas usage and enforces function integrity.

2. Inputs and Outputs to Functions

Functions in Solidity can accept input arguments and return one or more output values. Inputs enable users or other contracts to pass data into the contract, while outputs make it possible to return results after computation. For example, a function can accept an amount in Ether and return whether the transfer was successful. Solidity also allows named return variables, which improve readability and debugging.

3. Visibility, Modifiers and Constructors

- **Function Visibility** defines who can access a function:
    - o public: available both inside and outside the contract.
    - o private: only accessible within the same contract.

- internal: accessible within the contract and its child contracts.
- external: can be called only by external accounts or other contract

- **Modifiers** are reusable code blocks that change the behavior of functions. They are often used for access control, such as restricting sensitive functions to the contract owner (onlyOwner).
- **Constructors** are special functions executed only once during contract deployment. They initialize important values, such as setting the deploying account as the owner of the contract.

3. Control Flow: if-else, loops

Control flow in Solidity is similar to traditional programming languages:

- **if-else** allows conditional decision-making in contract logic, e.g., checking if a balance is sufficient before transferring funds.
- **Loops** (for, while, do-while) enable repeated execution of code. For example, iterating through an array of users. However, loops must be used carefully, as excessive iterations increase gas consumption, potentially making the contract expensive to execute.

5. Data Structures: Arrays, Mappings, Structs, Enums

- **Arrays**: Can be fixed or dynamic and are used to store ordered lists of elements. Example: an array of addresses for registered users.
- **Mappings**: Key-value pairs that allow quick lookups. Example: mapping(address => uint) for storing balances. Unlike arrays, mappings do not support iteration.
- **Structs**: Allow grouping of related properties into a single data type, such as creating a struct Player {string name; uint score;}.
- **Enums**: Used to define a set of predefined constants, making code more readable. Example: enum Status { Pending, Active, Closed }.

6. Data Locations

Solidity uses three primary data locations for storing variables:

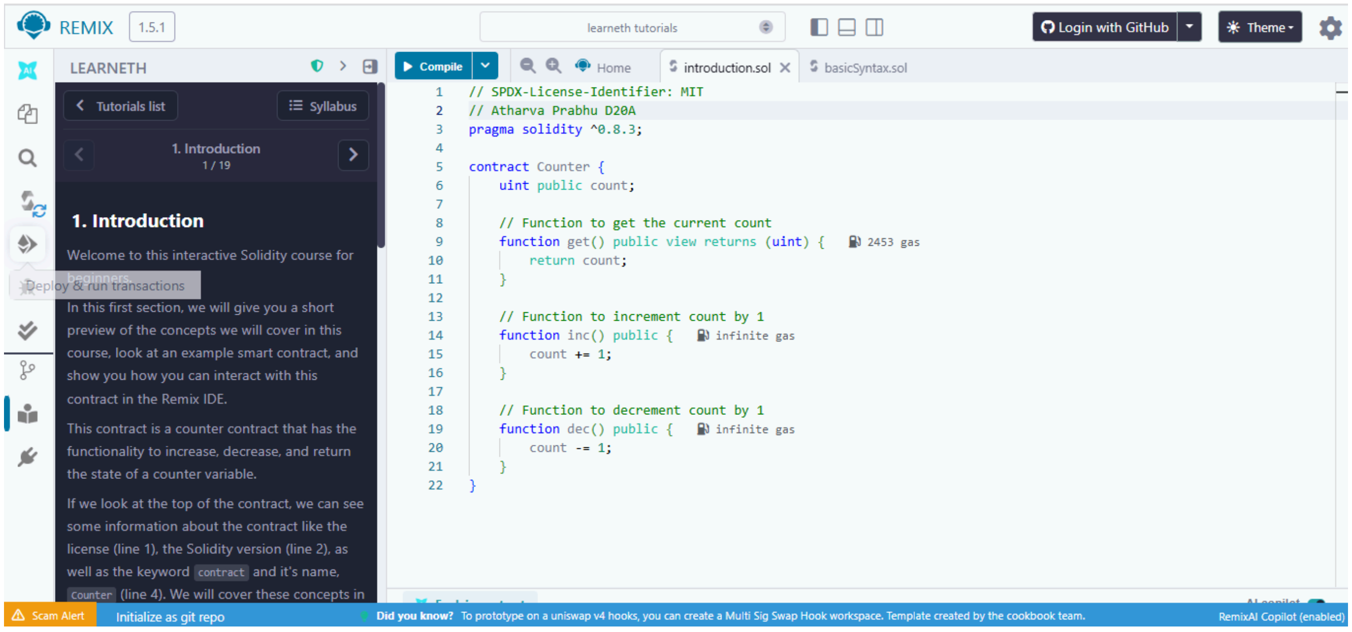
- **storage**: Data stored permanently on the blockchain. Examples: state variables.
- **memory**: Temporary data storage that exists only while a function is executing. Used for local variables and function inputs.
- **calldata**: A non-modifiable and non-persistent location used for external function parameters. It is gas-efficient compared to memory. Understanding data locations is essential, as they directly impact gas costs and performance.

7. Transactions: Ether and Wei, Gas and Gas Price, Sending Transactions

- **Ether and Wei**: Ether is the main currency in Ethereum. All values are measured in Wei, the smallest unit (1 Ether = 10^{18} Wei). This ensures high precision in financial transactions.
- **Gas and Gas Price**: Every transaction consumes gas, which represents computational effort. The gas price determines how much Ether is paid per unit of gas. A higher gas price incentivizes miners to prioritize the transaction.
- **Sending Transactions**: Transactions are used for transferring Ether or interacting with contracts. Functions like transfer() and send() are commonly used, while call() provides more flexibility. Each transaction requires gas, making efficiency in contract design very important.

Implementation:

- Tutorial no. 1 – Compile the code



1. Introduction

Welcome to this interactive Solidity course for beginners.

In this first section, we will give you a short preview of the concepts we will cover in this course, look at an example smart contract, and show you how you can interact with this contract in the Remix IDE.

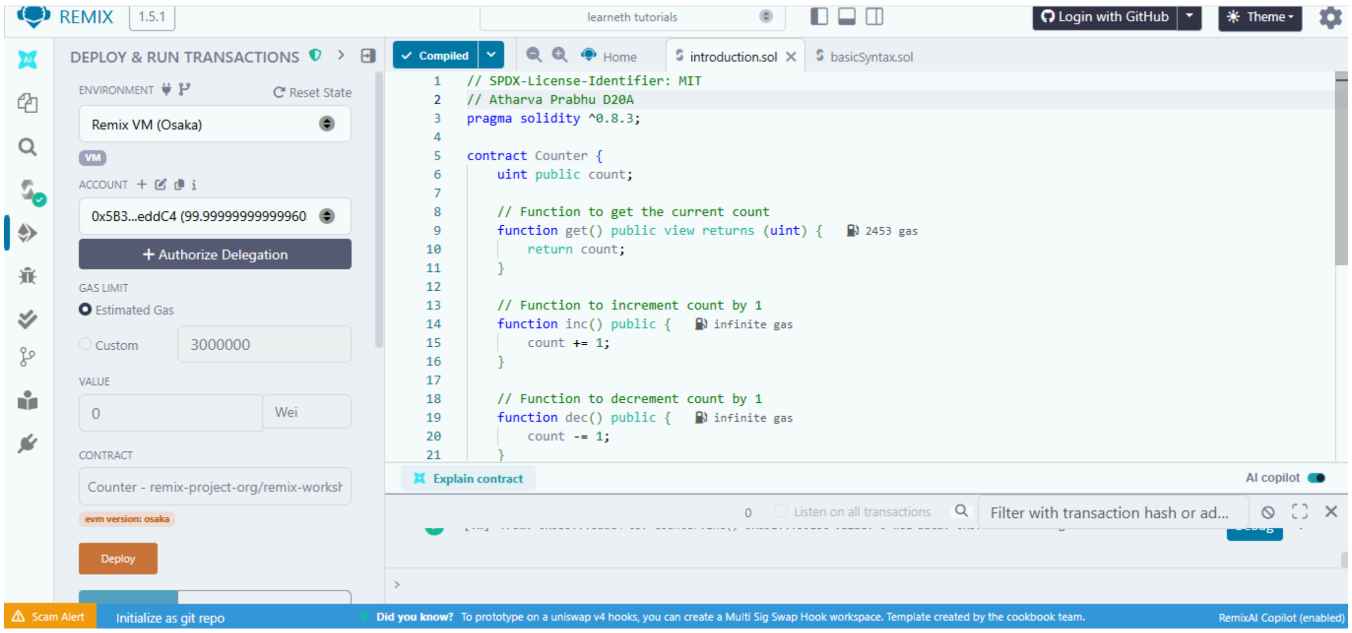
This contract is a counter contract that has the functionality to increase, decrease, and return the state of a counter variable.

If we look at the top of the contract, we can see some information about the contract like the license (line 1), the Solidity version (line 2), as well as the keyword `contract` and it's name, `Counter` (line 4). We will cover these concepts in

```solidity
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {    2453 gas
        return count;
    }

    // Function to increment count by 1
    function inc() public {    infinite gas
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {    infinite gas
        count -= 1;
    }
}
```

Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

- Tutorial no. 1 – Deploy the contract



ENVIRONMENT: Remix VM (Osaka)

ACCOUNT: 0x5B3...eddC4 (99.99999999999960

GAS LIMIT: Estimated Gas; Custom 3000000

VALUE: 0 Wei

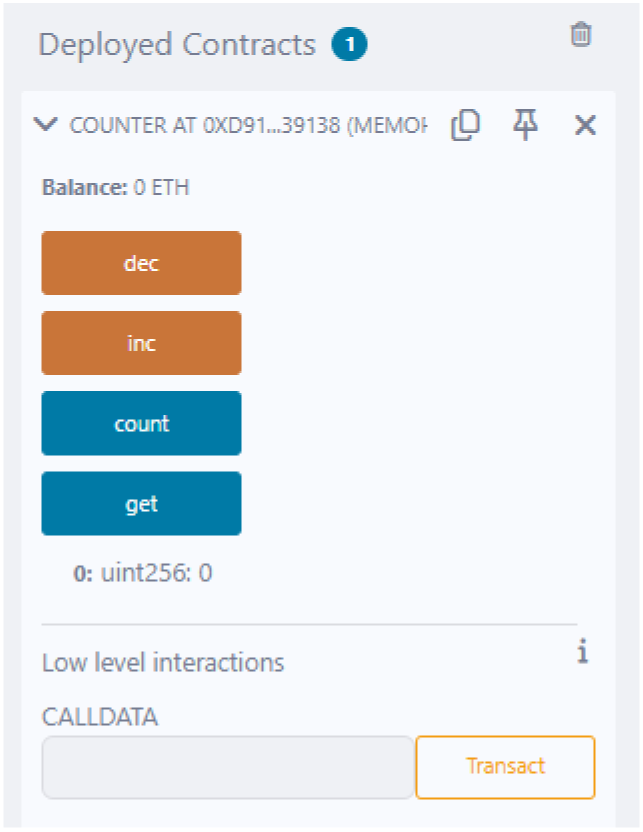
CONTRACT: Counter - remix-project-org/remix-worksh

evm version: osaka

```
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {    2453 gas
        return count;
    }

    // Function to increment count by 1
    function inc() public {    infinite gas
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {    infinite gas
        count -= 1;
    }
```

Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

- Tutorial no. 1 – get



Deployed Contracts 1

COUNTER AT 0XD91...39138 (MEMO

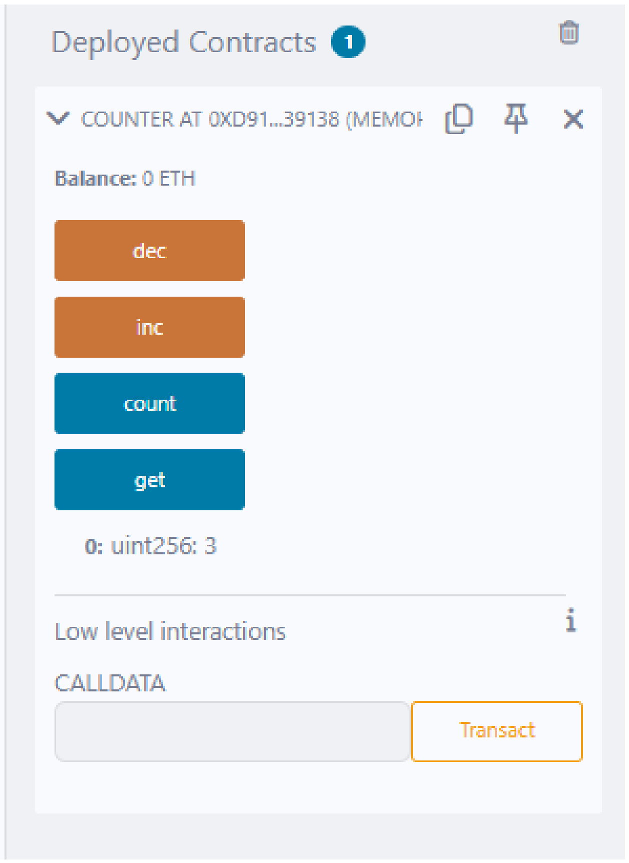
Balance: 0 ETH

0: uint256: 0

Low level interactions

CALLDATA

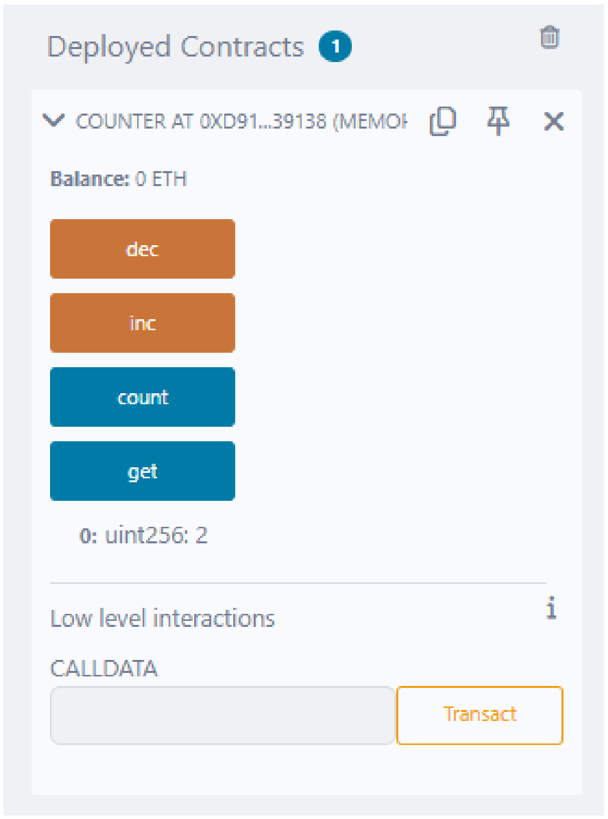
- Tutorial no. 1 – Increment



Balance: 0 ETH

0: uint256: 3

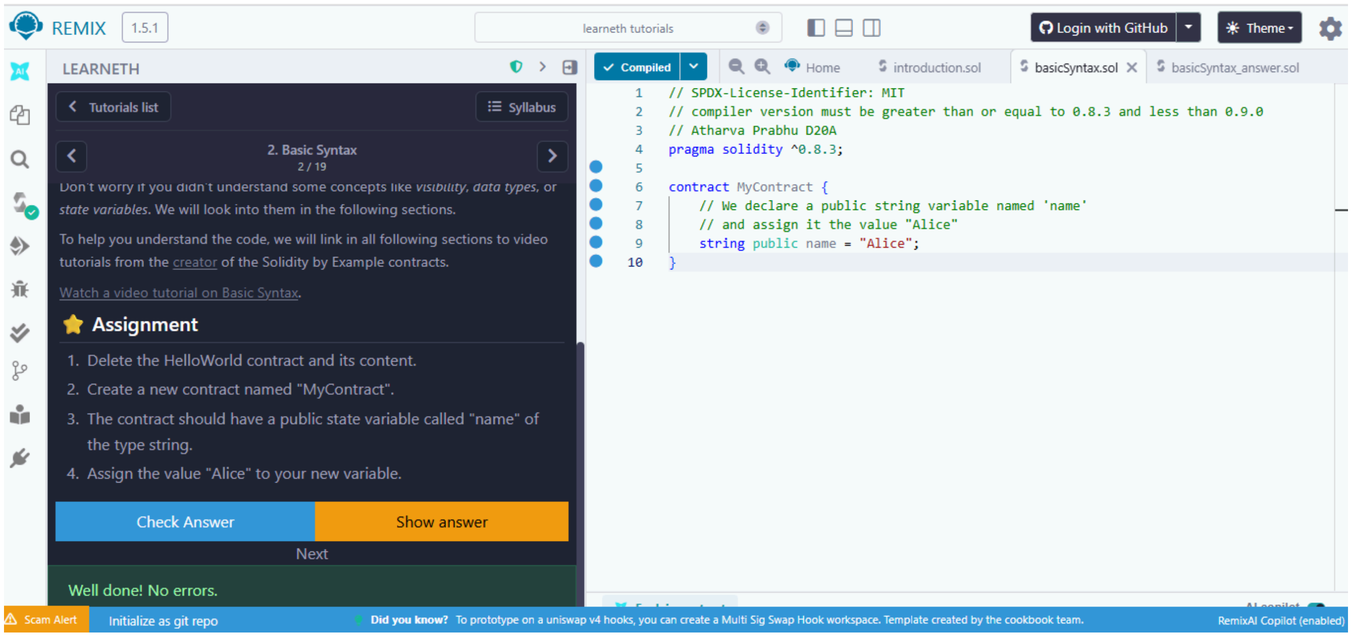
- Tutorial no. 1 – Decrement



Balance: 0 ETH

0: uint256: 2

- Tutorial no. 2



Don't worry if you didn't understand some concepts like *visibility*, *data types*, or *state variables*. We will look into them in the following sections.

To help you understand the code, we will link in all following sections to video tutorials from the creator of the Solidity by Example contracts.

Watch a video tutorial on Basic Syntax.

Assignment

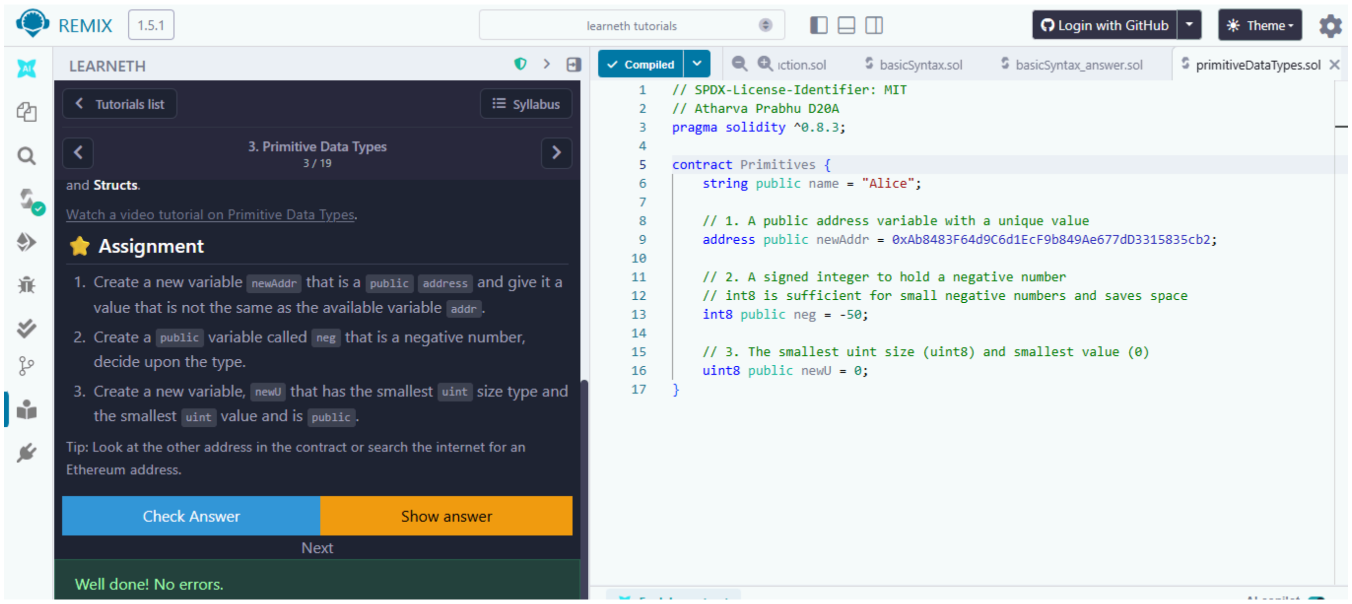
1. Delete the HelloWorld contract and its content.
2. Create a new contract named "MyContract".
3. The contract should have a public state variable called "name" of the type string.
4. Assign the value "Alice" to your new variable.

Well done! No errors.

Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.3 and less than 0.9.0
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract MyContract {
    // We declare a public string variable named 'name'
    // and assign it the value "Alice"
    string public name = "Alice";
}
```

- Tutorial no. 3



and **Structs**.

Watch a video tutorial on Primitive Data Types.

Assignment

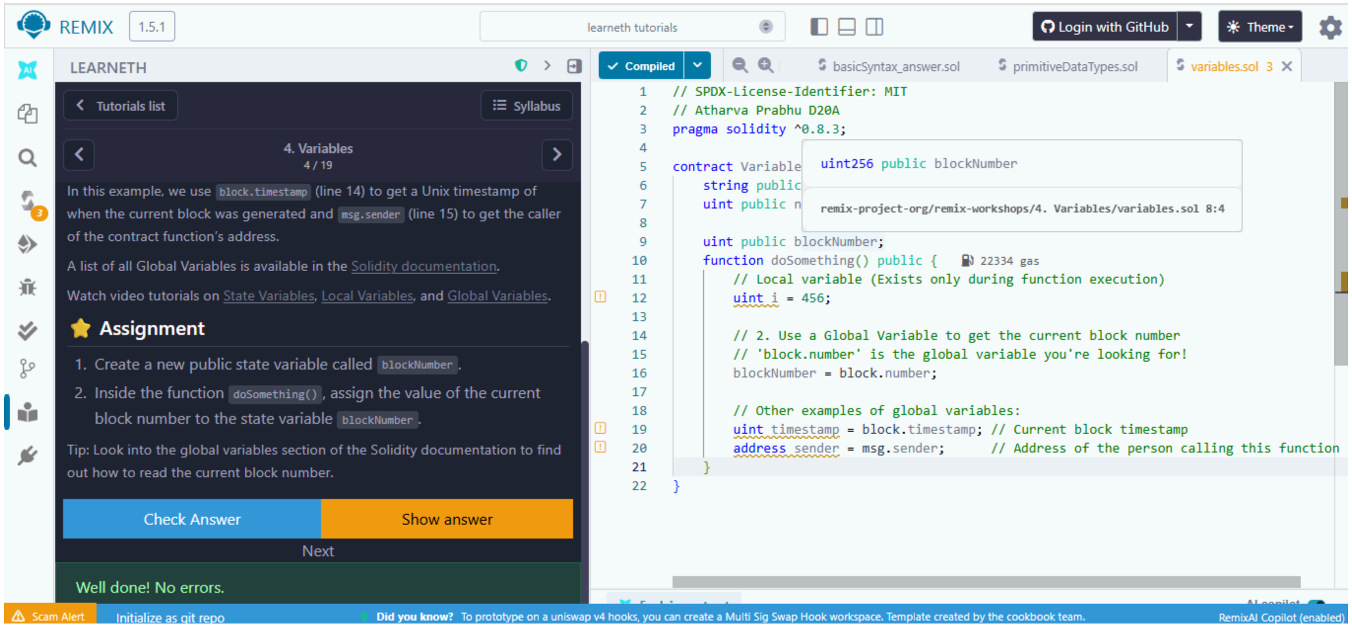
1. Create a new variable `newAddr` that is a `public` `address` and give it a value that is not the same as the available variable `addr`.
2. Create a `public` variable called `neg` that is a negative number, decide upon the type.
3. Create a new variable, `newU` that has the smallest `uint` size type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

```
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract Primitives {
    string public name = "Alice";

    // 1. A public address variable with a unique value
    address public newAddr = 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2;

    // 2. A signed integer to hold a negative number
    // int8 is sufficient for small negative numbers and saves space
    int8 public neg = -50;

    // 3. The smallest uint size (uint8) and smallest value (0)
    uint8 public newU = 0;
}
```

- Tutorial no. 4



4. Variables
4 / 19

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.

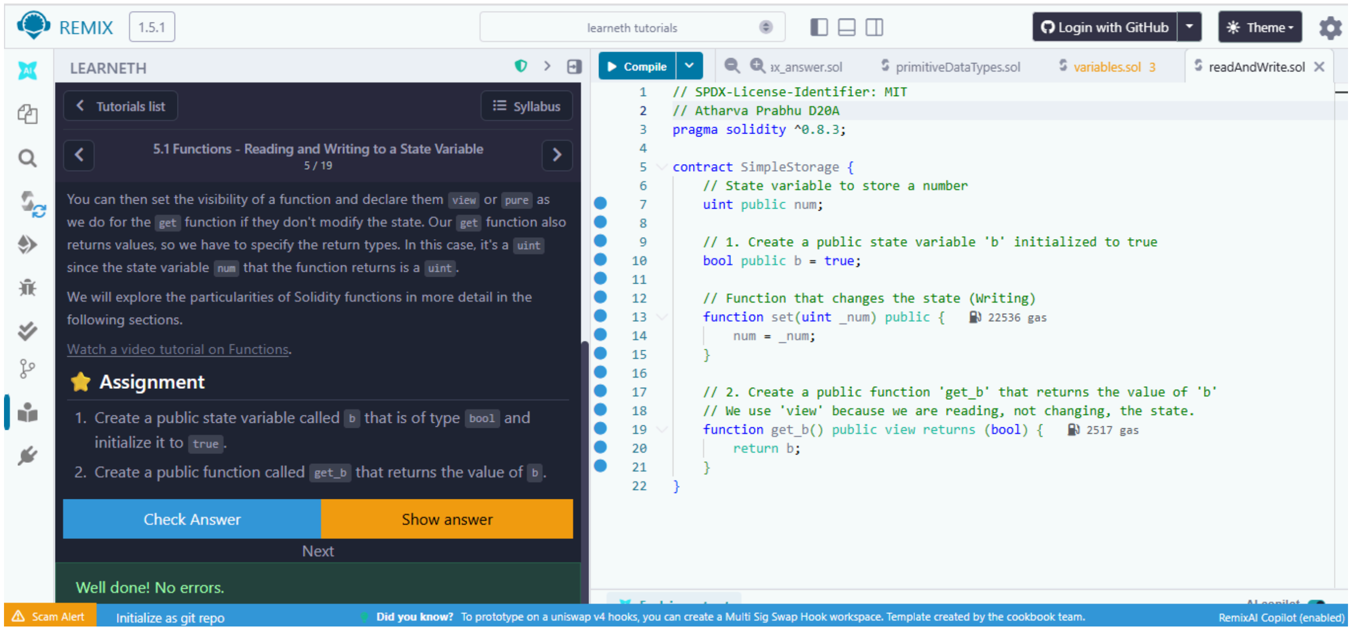
Well done! No errors.

```solidity
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract Variable
    string public
    uint public n

    uint public blockNumber;
    function doSomething() public {    22334 gas
        // Local variable (Exists only during function execution)
        uint i = 456;

        // 2. Use a Global Variable to get the current block number
        // 'block.number' is the global variable you're looking for!
        blockNumber = block.number;

        // Other examples of global variables:
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender;       // Address of the person calling this function
    }
}
```

```
uint256 public blockNumber
remix-project-org/remix-workshops/4. Variables/variables.sol 8:4
```

Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

- Tutorial no. 5



5.1 Functions - Reading and Writing to a State Variable
5 / 19

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

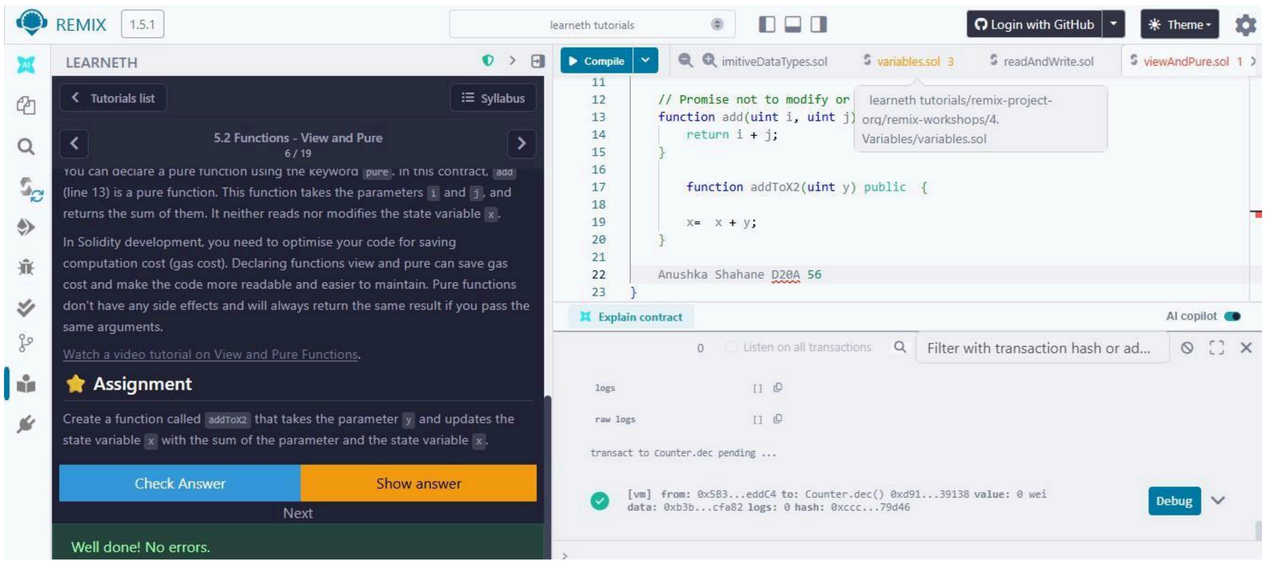
1. Create a public state variable called `b` that is of type `bool` and initialize it to `true`.
2. Create a public function called `get_b` that returns the value of `b`.

Well done! No errors.

```solidity
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

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

    // 1. Create a public state variable 'b' initialized to true
    bool public b = true;

    // Function that changes the state (Writing)
    function set(uint _num) public {    22536 gas
        num = _num;
    }

    // 2. Create a public function 'get_b' that returns the value of 'b'
    // We use 'view' because we are reading, not changing, the state.
    function get_b() public view returns (bool) {    2517 gas
        return b;
    }
}
```

Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

- Tutorial no. 6



5.2 Functions - View and Pure
6 / 19

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`.

Check Answer | Show answer

Next

Well done! No errors.

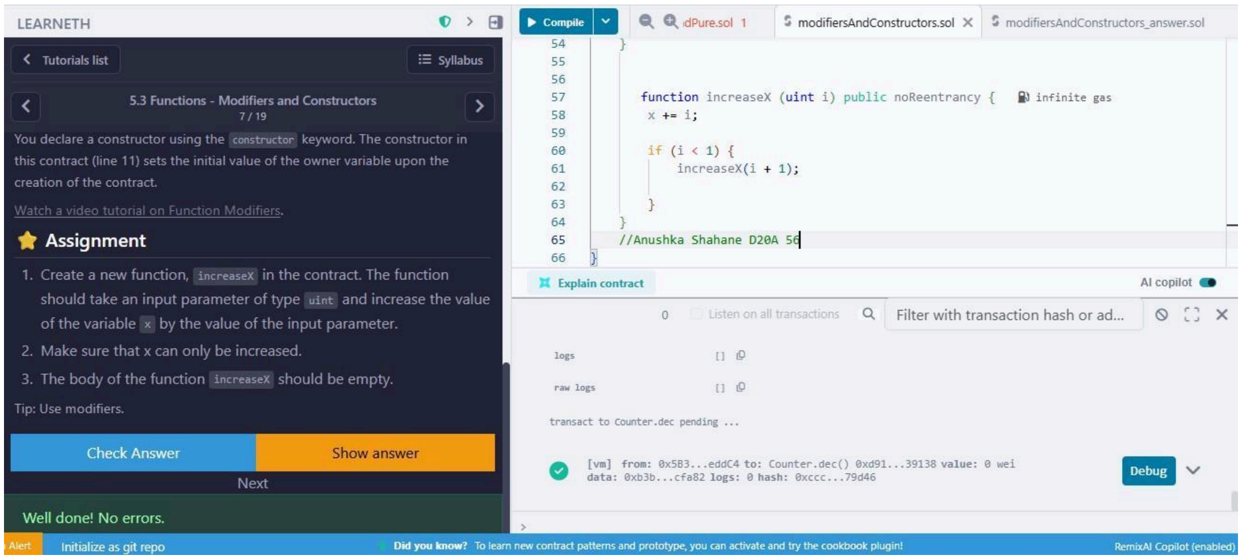
```
11
12      // Promise not to modify or
13      function add(uint i, uint j)
14          return i + j;
15      }
16
17          function addToX2(uint y) public  {
18
19          x=  x + y;
20      }
21
22      Anushka Shahane D20A 56
23  }
```

learneth tutorials/remix-project-org/remix-workshops/4. Variables/variables.sol

Explain contract
AI copilot
0
Listen on all transactions
Filter with transaction hash or ad...

```
logs        []
raw logs    []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

- Tutorial no. 7


LEARNETH
Tutorials list
Syllabus
Compile
dPure.sol 1
modifiersAndConstructors.sol
modifiersAndConstructors_answer.sol

5.3 Functions - Modifiers and Constructors
7 / 19

You declare a constructor using the `constructor` keyword. The constructor in this contract (line 11) sets the initial value of the owner variable upon the creation of 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.

Check Answer | Show answer

Next

Well done! No errors.

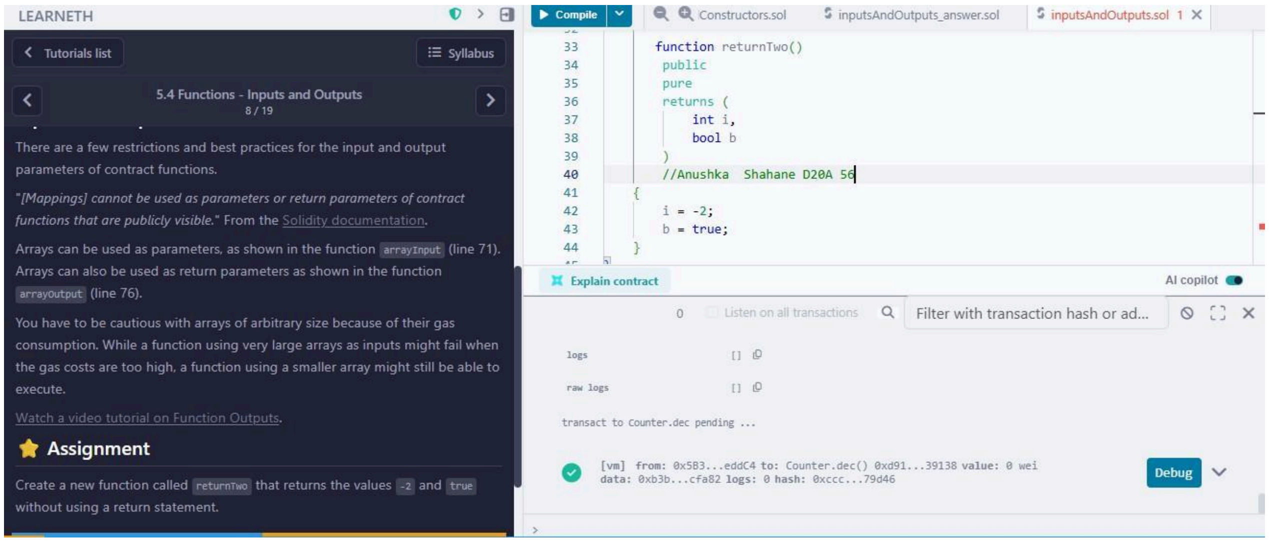
```
54      }
55
56
57          function increaseX (uint i) public noReentrancy {     infinite gas
58           x += i;
59
60           if (i < 1) {
61               increaseX(i + 1);
62
63           }
64      }
65      //Anushka Shahane D20A 56
66  }
```

Explain contract
AI copilot
0
Listen on all transactions
Filter with transaction hash or ad...

```
logs        []
raw logs    []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!

- Tutorial no. 8



LEARNETH

Tutorials list | Syllabus

5.4 Functions - Inputs and Outputs
8 / 19

There are a few restrictions and best practices for the input and output parameters of contract functions.

"[Mappings] cannot be used as parameters or return parameters of contract functions that are publicly visible." From the Solidity documentation.

Arrays can be used as parameters, as shown in the function `arrayInput` (line 71). Arrays can also be used as return parameters as shown in the function `arrayoutput` (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.

Compile | Constructors.sol | inputsAndOutputs_answer.sol | inputsAndOutputs.sol 1

```
    function returnTwo()
     public
     pure
     returns (
         int i,
         bool b
     )
     //Anushka  Shahane D20A 56
    {
     i = -2;
     b = true;
    }
```

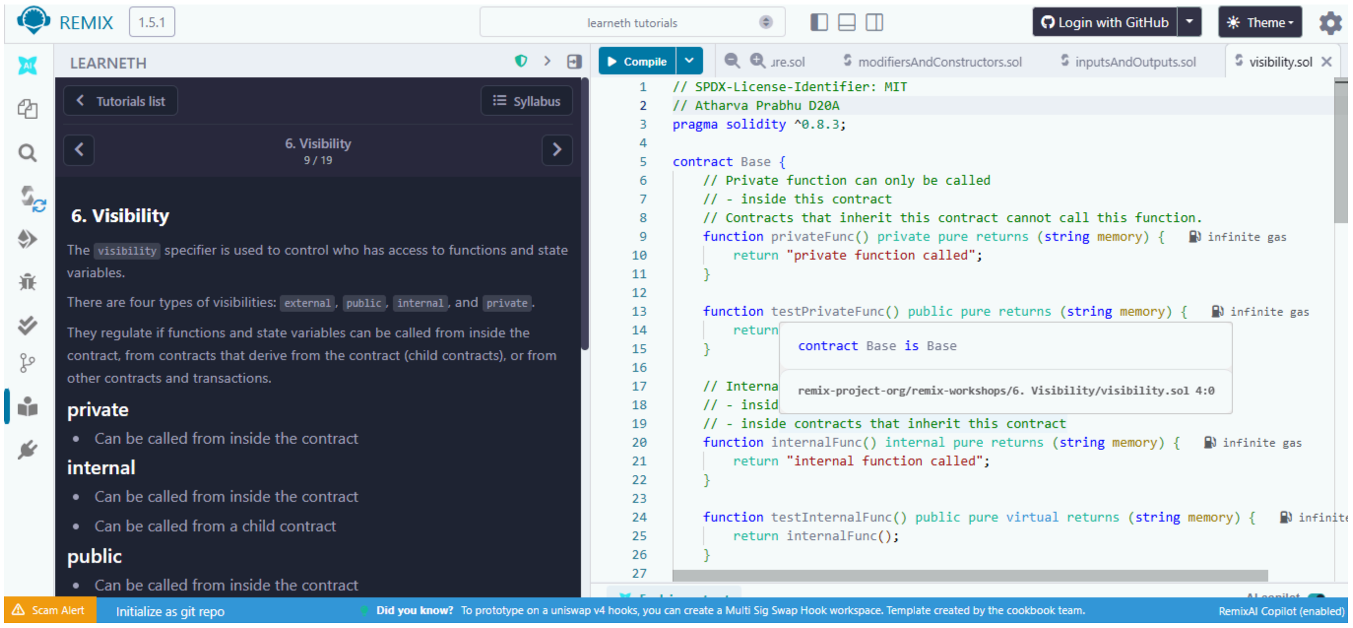
Explain contract | AI copilot

0 Listen on all transactions | Filter with transaction hash or ad...

```
logs                  []
raw logs              []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Debug

- Tutorial no. 9



REMIX 1.5.1 | learneth tutorials | Login with GitHub | Theme

LEARNETH

Tutorials list | Syllabus

6. Visibility
9 / 19

6. Visibility

The `visibility` specifier is used to control who has access to functions and state variables.

There are four types of visibilities: `external`, `public`, `internal`, and `private`.

They regulate if functions and state variables can be called from inside the contract, from contracts that derive from the contract (child contracts), or from other contracts and transactions.

private

- Can be called from inside the contract

internal

- Can be called from inside the contract
- Can be called from a child contract

public

- Can be called from inside the contract

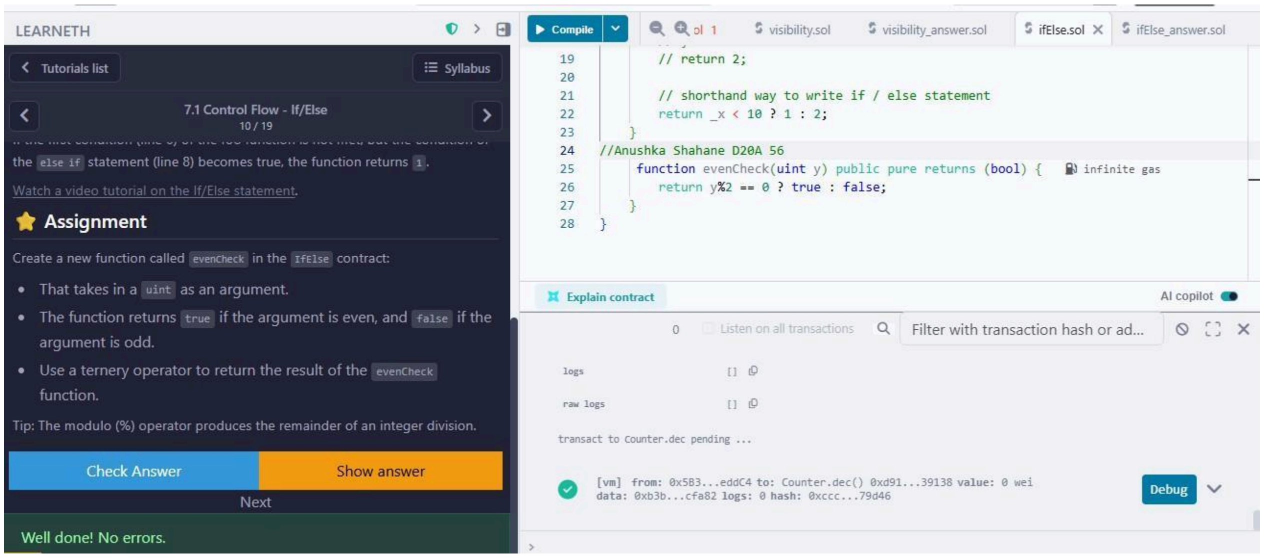
Compile | ure.sol | modifiersAndConstructors.sol | inputsAndOutputs.sol | visibility.sol

```
// SPDX-License-Identifier: MIT
// Atharva Prabhu D20A
pragma solidity ^0.8.3;

contract Base {
    // Private function can only be called
    // - inside this contract
    // Contracts that inherit this contract cannot call this function.
    function privateFunc() private pure returns (string memory) {   infinite gas
        return "private function called";
    }

    function testPrivateFunc() public pure returns (string memory) {   infinite gas
        return
    }

    // Interna
    // - insid
    // - inside contracts that inherit this contract
    function internalFunc() internal pure returns (string memory) {   infinite gas
        return "internal function called";
    }

    function testInternalFunc() public pure virtual returns (string memory) {   infinite
        return internalFunc();
    }
```

```
contract Base is Base

remix-project-org/remix-workshops/6. Visibility/visibility.sol 4:0
```

Scam Alert | Initialize as git repo | Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. | RemixAI Copilot (enabled)

- Tutorial no. 10



LEARNETH

Tutorials list | Syllabus

7.1 Control Flow - If/Else
10 / 19

the `else if` statement (line 8) becomes true, the function returns `1`.

Watch a video tutorial on the If/Else statement.

Assignment

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.

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

Check Answer | Show answer

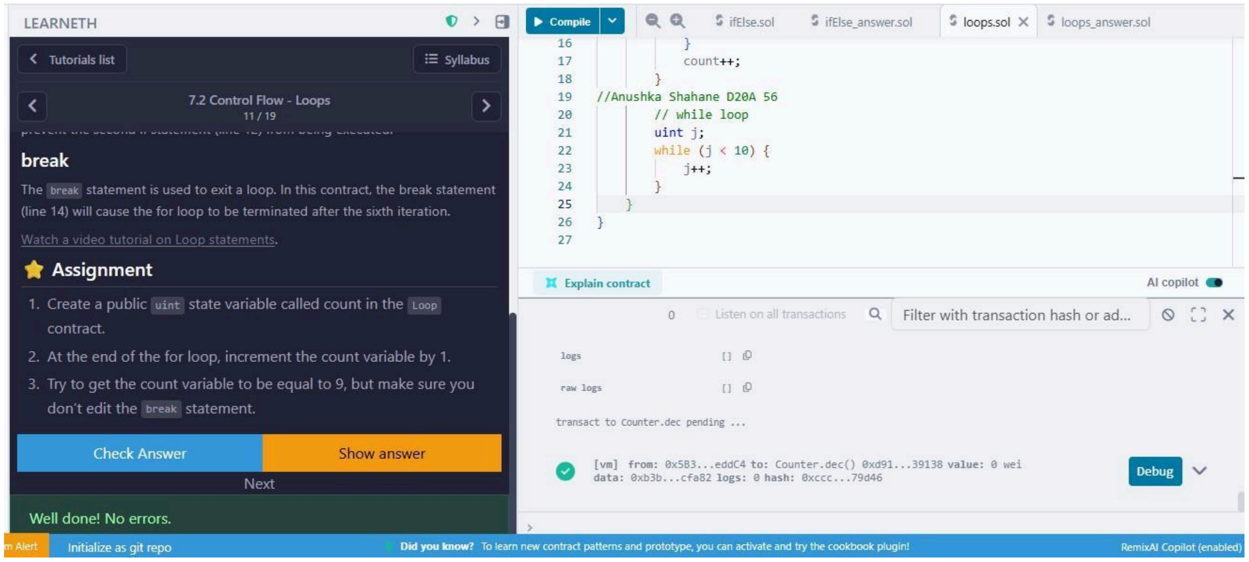
Next

Well done! No errors.

```
        // return 2;

        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }
//Anushka Shahane D20A 56
    function evenCheck(uint y) public pure returns (bool) {
        return y%2 == 0 ? true : false;
    }
}
```

```
logs        []
raw logs    []
transact to Counter.dec pending ...
[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

- Tutorial no. 11



LEARNETH

Tutorials list | Syllabus

7.2 Control Flow - Loops
11 / 19

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.

Check Answer | Show answer

Next

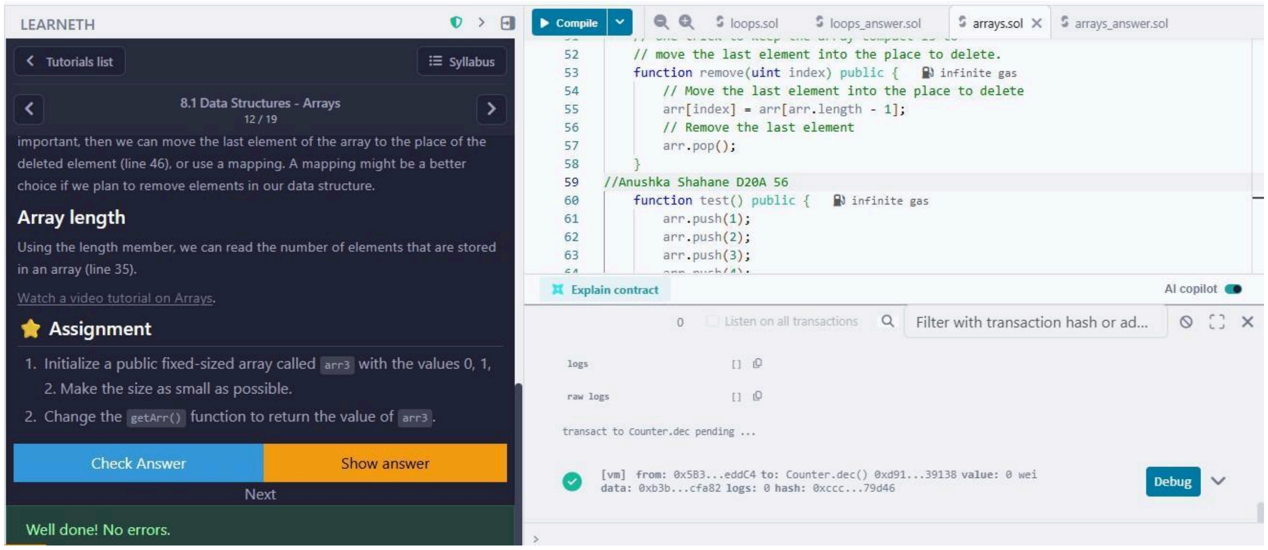
Well done! No errors.

```
            }
            count++;
        }
//Anushka Shahane D20A 56
        // while loop
        uint j;
        while (j < 10) {
            j++;
        }
    }
}
```

```
logs        []
raw logs    []
transact to Counter.dec pending ...
[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!

- Tutorial no. 12



8.1 Data Structures - Arrays
12 / 19

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`.

Check Answer | Show answer

Next

Well done! No errors.

```
    // move the last element into the place to delete.
    function remove(uint index) public {     infinite gas
        // Move the last element into the place to delete
        arr[index] = arr[arr.length - 1];
        // Remove the last element
        arr.pop();
    }
//Anushka Shahane D20A 56
    function test() public {     infinite gas
        arr.push(1);
        arr.push(2);
        arr.push(3);
```

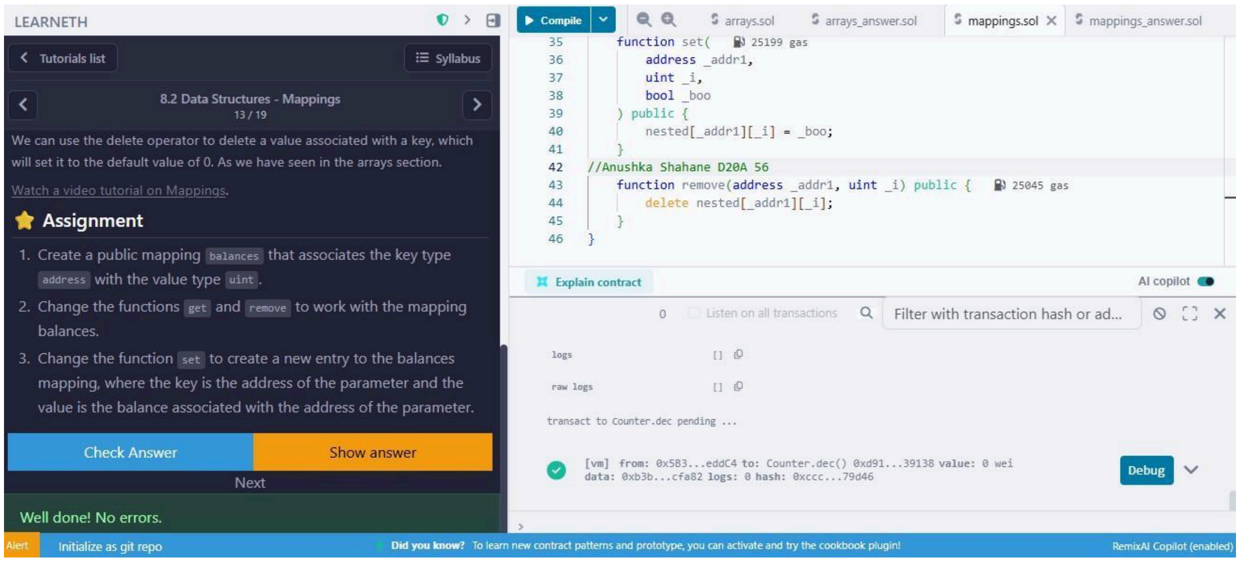
Explain contract   AI copilot

0   Listen on all transactions   Filter with transaction hash or ad...

```
logs        []
raw logs    []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Debug

- Tutorial no. 13



8.2 Data Structures - Mappings
13 / 19

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.

Check Answer | Show answer

Next

Well done! No errors.

```
    function set(     25199 gas
        address _addr1,
        uint _i,
        bool _boo
    ) public {
        nested[_addr1][_i] = _boo;
    }
//Anushka Shahane D20A 56
    function remove(address _addr1, uint _i) public {     25045 gas
        delete nested[_addr1][_i];
    }
}
```

Explain contract   AI copilot

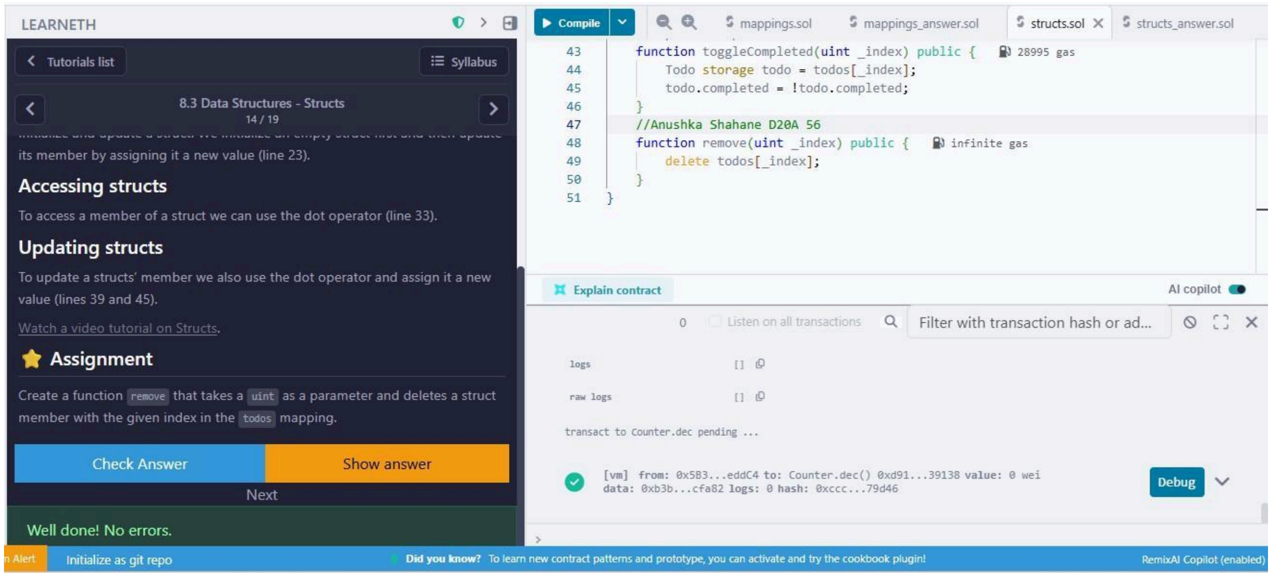
0   Listen on all transactions   Filter with transaction hash or ad...

```
logs        []
raw logs    []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Debug

Alert   Initialize as git repo   Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!   RemixAI Copilot (enabled)

- Tutorial no. 14



initialize and update a struct. We initialize an empty struct first and then update its member by assigning it a new value (line 23).

Accessing structs

To access a member of a struct we can use the dot operator (line 33).

Updating structs

To update a structs' member we also use the dot operator and assign it a new value (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

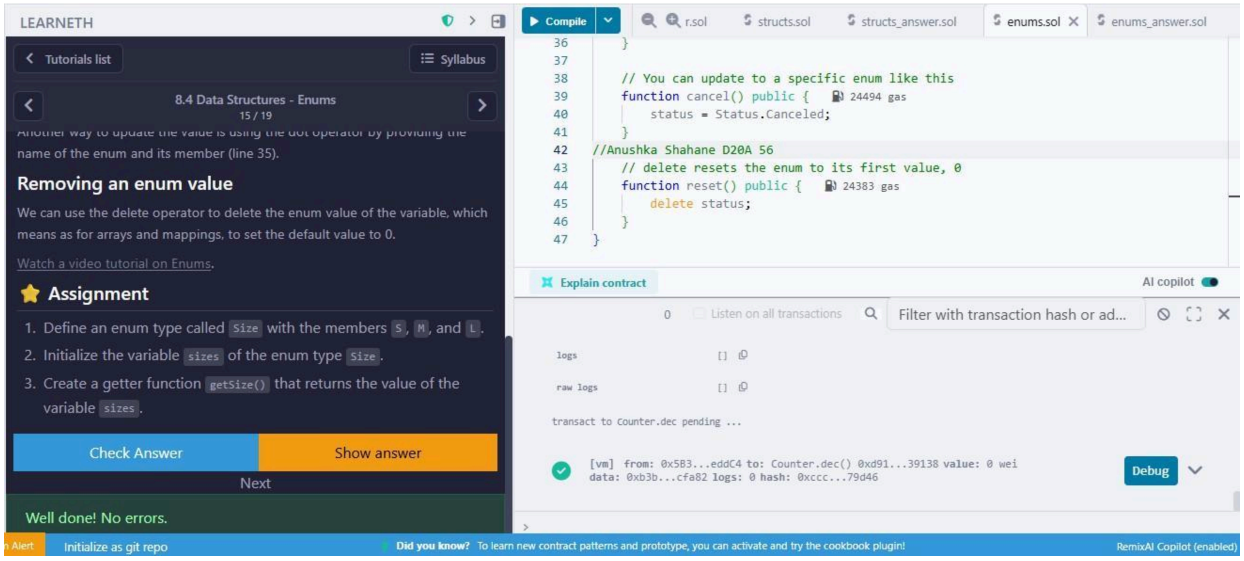
Well done! No errors.

```
    function toggleCompleted(uint _index) public {     28995 gas
        Todo storage todo = todos[_index];
        todo.completed = !todo.completed;
    }
    //Anushka Shahane D20A 56
    function remove(uint _index) public {     infinite gas
        delete todos[_index];
    }
}
```

```
0
logs        []
raw logs    []
transact to Counter.dec pending ...
[vm] from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!

- Tutorial no. 15



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

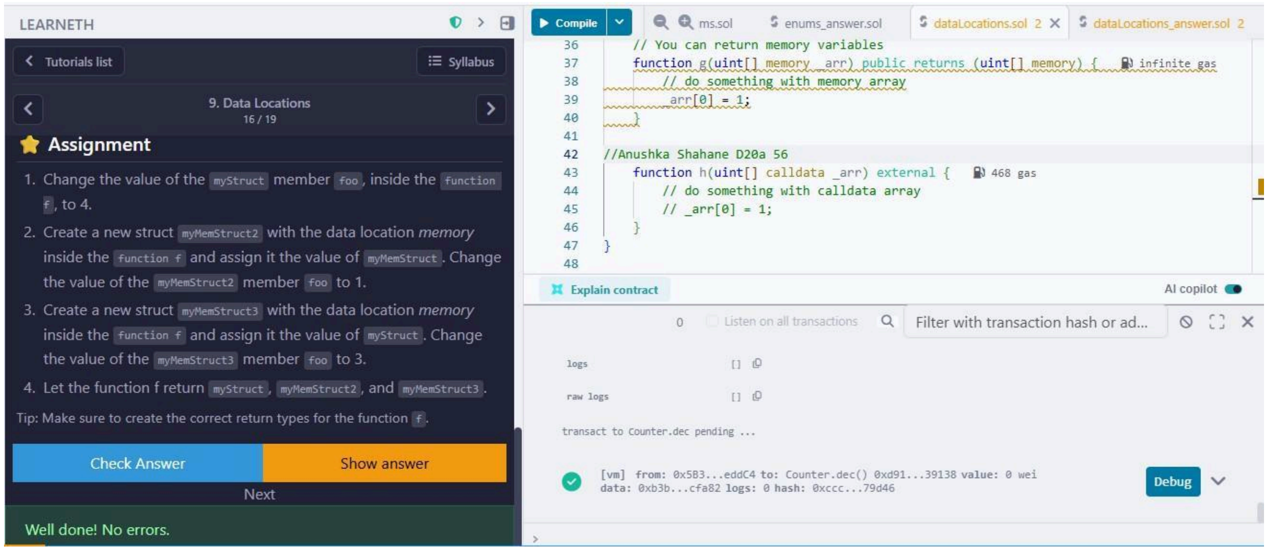
Well done! No errors.

```
    }

    // You can update to a specific enum like this
    function cancel() public {     24494 gas
        status = Status.Canceled;
    }
//Anushka Shahane D20A 56
    // delete resets the enum to its first value, 0
    function reset() public {     24383 gas
        delete status;
    }
}
```

```
0
logs        []
raw logs    []
transact to Counter.dec pending ...
[vm] from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!

- Tutorial no. 16



9. Data Locations
16 / 19

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`.

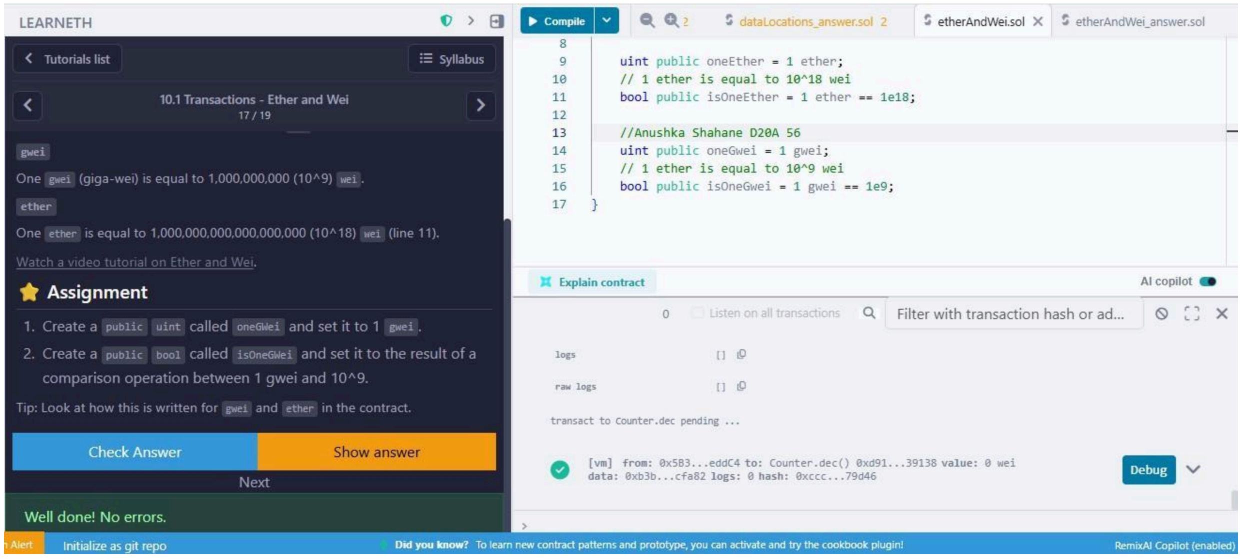
Tip: Make sure to create the correct return types for the function `f`.

Well done! No errors.

```
    // You can return memory variables
    function g(uint[] memory _arr) public returns (uint[] memory) {    infinite gas
        // do something with memory array
        arr[0] = 1;
    }

//Anushka Shahane D20a 56
    function h(uint[] calldata _arr) external {    468 gas
        // do something with calldata array
        // _arr[0] = 1;
    }
}
```

```
0   Listen on all transactions   Filter with transaction hash or ad...

logs        []
raw logs    []

transact to Counter.dec pending ...

[vm] from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

- Tutorial no. 17



10.1 Transactions - Ether and Wei
17 / 19

`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.

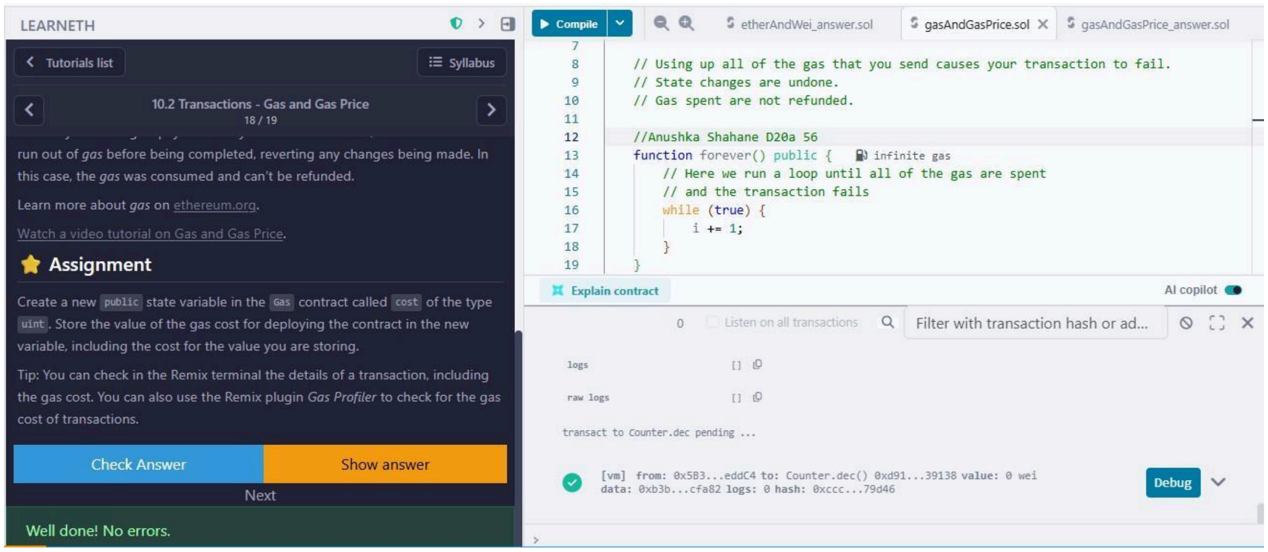
Well done! No errors.

```
    uint public oneEther = 1 ether;
    // 1 ether is equal to 10^18 wei
    bool public isOneEther = 1 ether == 1e18;

    //Anushka Shahane D20A 56
    uint public oneGwei = 1 gwei;
    // 1 ether is equal to 10^9 wei
    bool public isOneGwei = 1 gwei == 1e9;
}
```

```
0   Listen on all transactions   Filter with transaction hash or ad...

logs        []
raw logs    []

transact to Counter.dec pending ...

[vm] from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!

- Tutorial no. 18



LEARNETH

10.2 Transactions - Gas and Gas Price
18 / 19

run out of *gas* before being completed, reverting any changes being made. In this case, the *gas* was consumed and can't be refunded.

Learn more about *gas* on ethereum.org.

Watch a video tutorial on Gas and Gas Price.

Assignment

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.

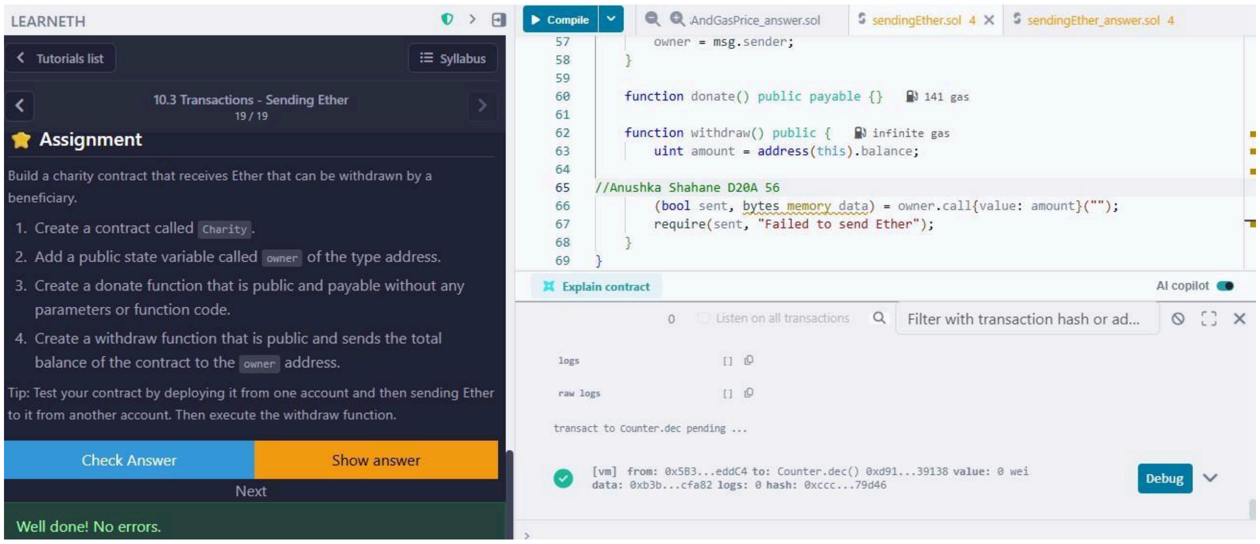
Tip: You can check in the Remix terminal the details of a transaction, including the gas cost. You can also use the Remix plugin *Gas Profiler* to check for the gas cost of transactions.

Well done! No errors.

```
    // Using up all of the gas that you send causes your transaction to fail.
    // State changes are undone.
    // Gas spent are not refunded.

    //Anushka Shahane D20a 56
    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;
        }
    }
```

```
logs          []
raw logs      []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

- Tutorial no. 19



LEARNETH

10.3 Transactions - Sending Ether
19 / 19

Assignment

Build a charity contract that receives Ether that can be withdrawn by a beneficiary.

1. Create a contract called `Charity`.
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.

```
        owner = msg.sender;
    }

    function donate() public payable {}    141 gas

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

//Anushka Shahane D20A 56
        (bool sent, bytes memory data) = owner.call{value: amount}("");
        require(sent, "Failed to send Ether");
    }
}
```

```
logs          []
raw logs      []

transact to Counter.dec pending ...

[vm]  from: 0x5B3...eddC4 to: Counter.dec() 0xd91...39138 value: 0 wei
data: 0xb3b...cfa82 logs: 0 hash: 0xccc...79d46
```

Conclusion: Through this experiment, the fundamentals of Solidity programming were explored by completing practical assignments in the Remix IDE. Concepts such as data types, variables, functions, visibility, modifiers, constructors, control flow, data structures, and transactions were implemented and understood. The hands-on practice helped in designing, compiling, and deploying smart contracts on the Remix VM, thereby strengthening the understanding of blockchain concepts. This experiment provided a strong foundation for developing and managing smart contracts efficiently.