Snehal Patil D20A 41

Blockchain Experiment 4

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

4. **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).

5. **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:

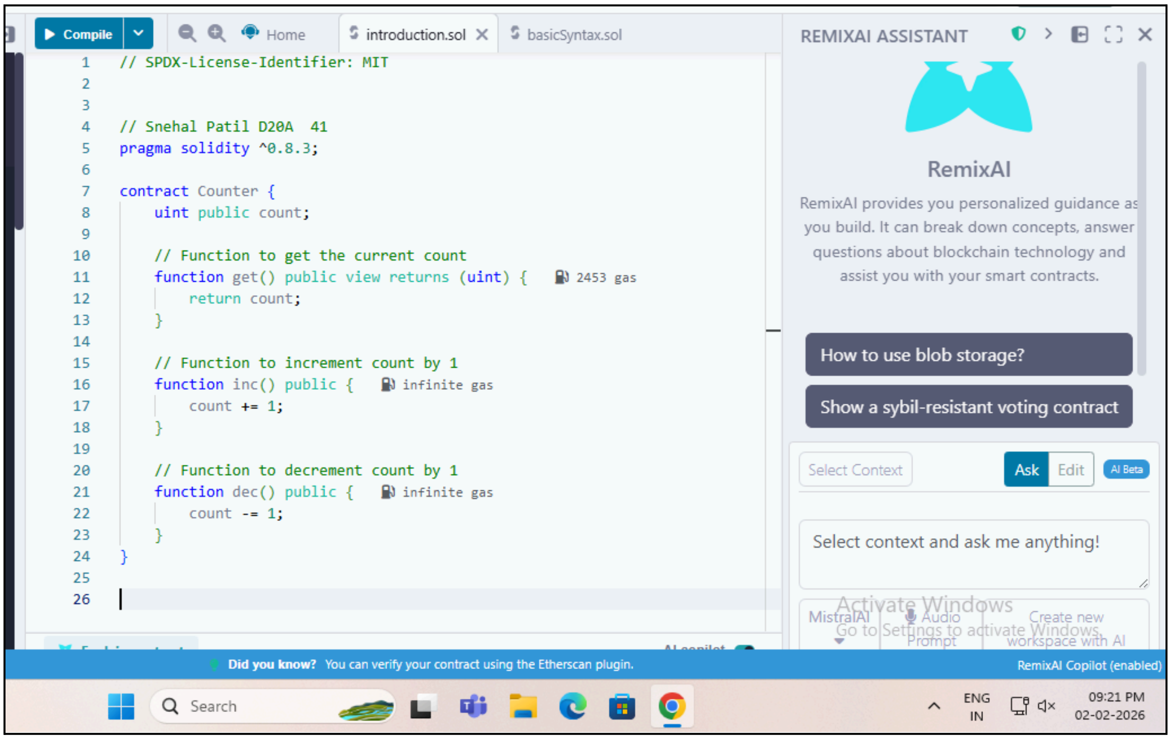
- **storage**: Data stored permanently on the blockchain. Examples: state variables.
- **memory**: Temporary data storage that exists only while a function is executing. Used for local variables and function inputs.
- **calldata**: A non-modifiable and non-persistent location used for external function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

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

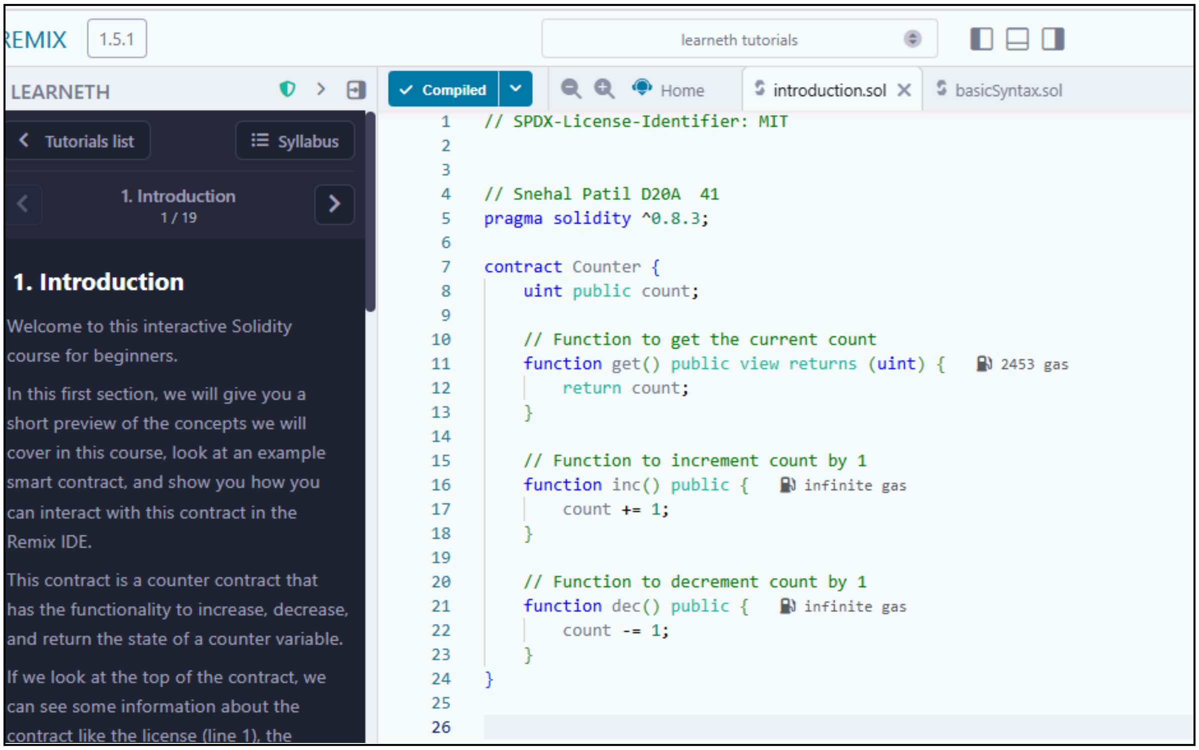


```
1   // SPDX-License-Identifier: MIT
2
3
4   // Snehal Patil D20A  41
5   pragma solidity ^0.8.3;
6
7   contract Counter {
8       uint public count;
9
10      // Function to get the current count
11      function get() public view returns (uint) {    2453 gas
12          return count;
13      }
14
15      // Function to increment count by 1
16      function inc() public {    infinite gas
17          count += 1;
18      }
19
20      // Function to decrement count by 1
21      function dec() public {    infinite gas
22          count -= 1;
23      }
24  }
25
26
```

RemixAI provides you personalized guidance as you build. It can break down concepts, answer questions about blockchain technology and assist you with your smart contracts.

How to use blob storage?
Show a sybil-resistant voting contract
Select Context
Ask
Edit
AI Beta
Select context and ask me anything!
MistralAI
Audio Prompt
Create new workspace with AI
Activate Windows
Go to Settings to activate Windows.
RemixAI Copilot (enabled)
Search
ENG IN
09:21 PM
02-02-2026

Did you know? You can verify your contract using the Etherscan plugin.



```
1   // SPDX-License-Identifier: MIT
2
3
4   // Snehal Patil D20A  41
5   pragma solidity ^0.8.3;
6
7   contract Counter {
8       uint public count;
9
10      // Function to get the current count
11      function get() public view returns (uint) {    2453 gas
12          return count;
13      }
14
15      // Function to increment count by 1
16      function inc() public {    infinite gas
17          count += 1;
18      }
19
20      // Function to decrement count by 1
21      function dec() public {    infinite gas
22          count -= 1;
23      }
24  }
25
26
```

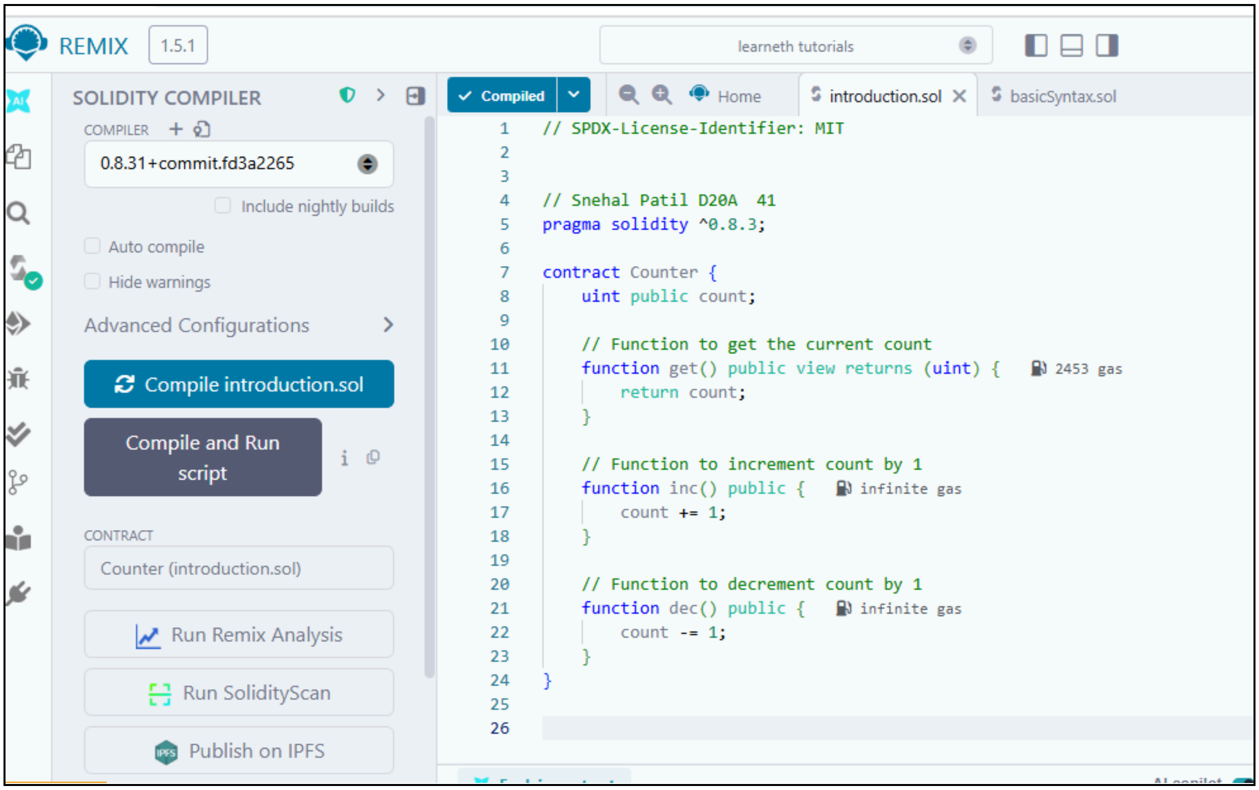
1. Introduction

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.

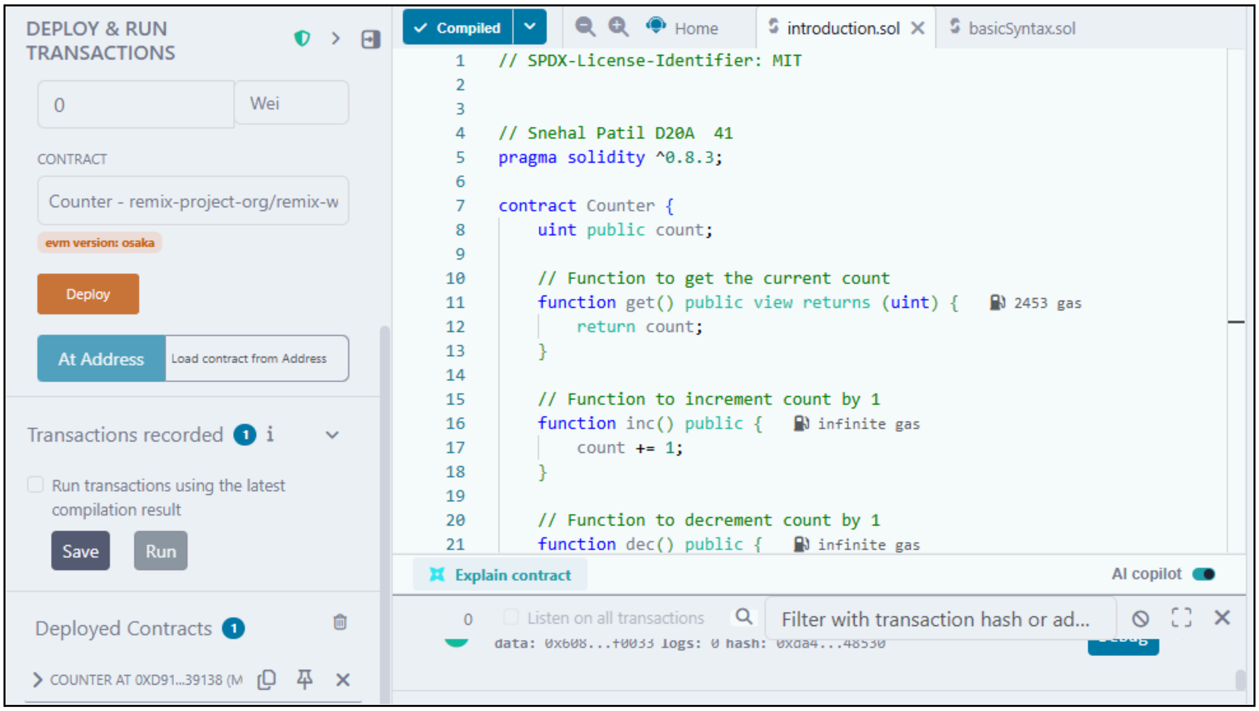
This contract is a counter contract that has the functionality to increase, decrease, and return the state of a counter variable.

If we look at the top of the contract, we can see some information about the contract like the license (line 1), the



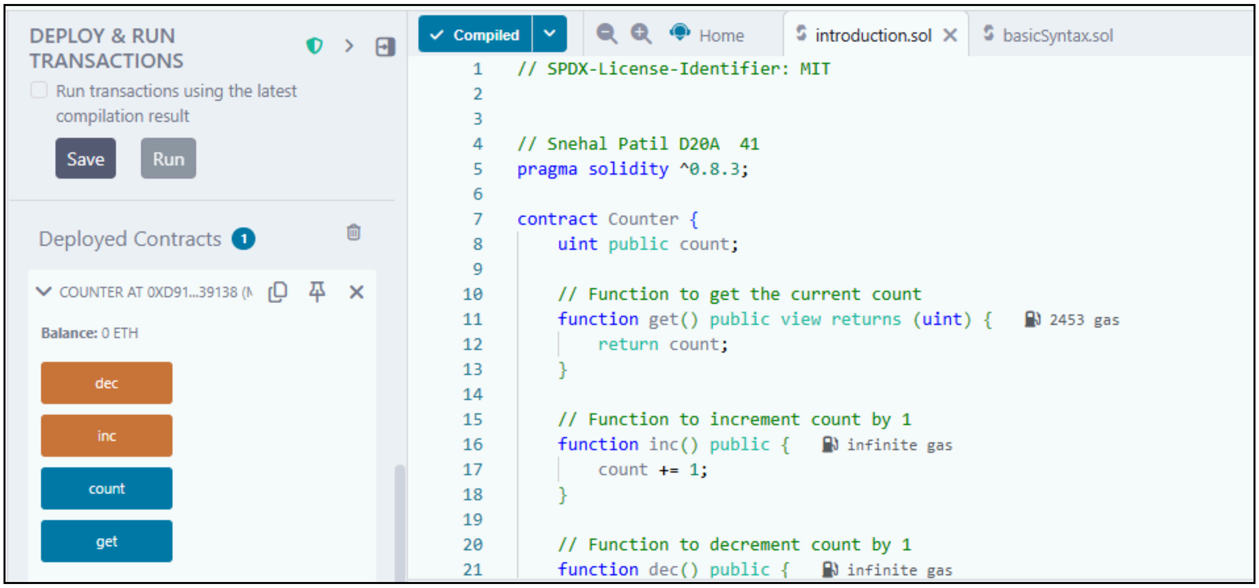
```
// SPDX-License-Identifier: MIT


// Snehal Patil D20A  41
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;
    }
}
```

- Tutorial no. 1 – Deploy the contract

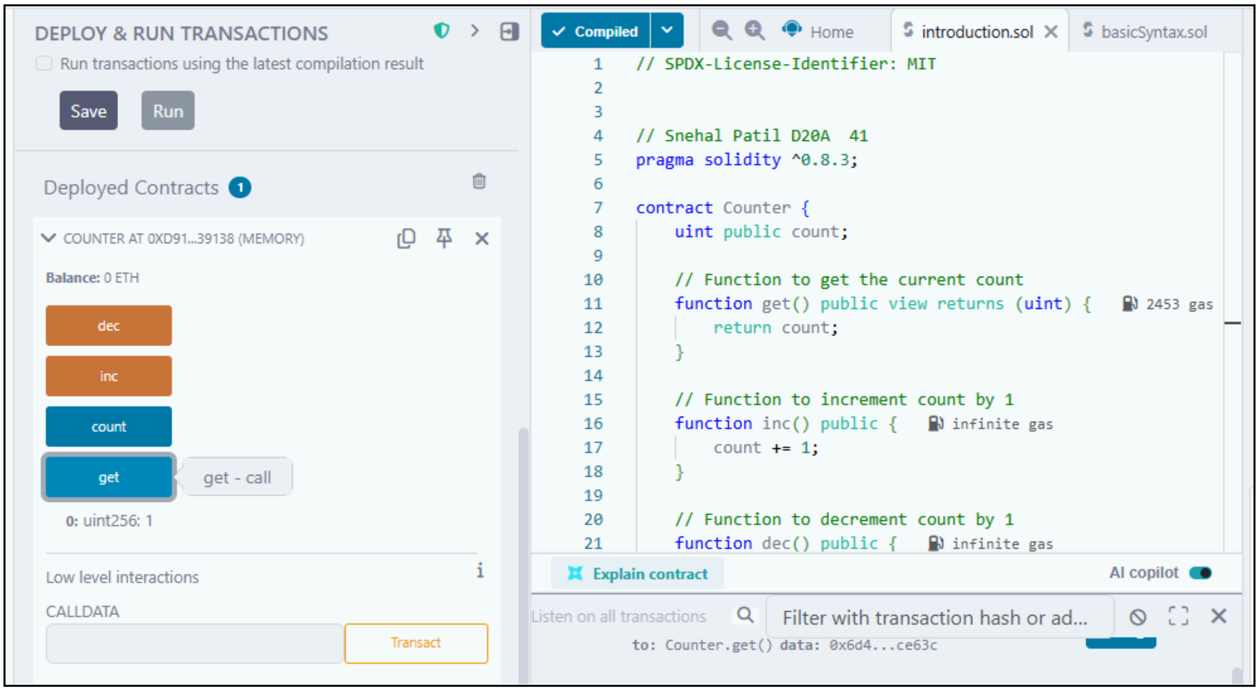

DEPLOY & RUN TRANSACTIONS
0
Wei
CONTRACT
Counter - remix-project-org/remix-w
evm version: osaka
Deploy
At Address
Load contract from Address
Transactions recorded 1
Run transactions using the latest compilation result
Save
Run
Deployed Contracts 1
COUNTER AT 0XD91...39138 (M
Compiled
Home
introduction.sol
basicSyntax.sol
Explain contract
AI copilot
0
Listen on all transactions
Filter with transaction hash or ad...

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


// Snehal Patil D20A  41
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
```

```
data: 0x608...f0033 logs: 0 hash: 0xda4...48530
```



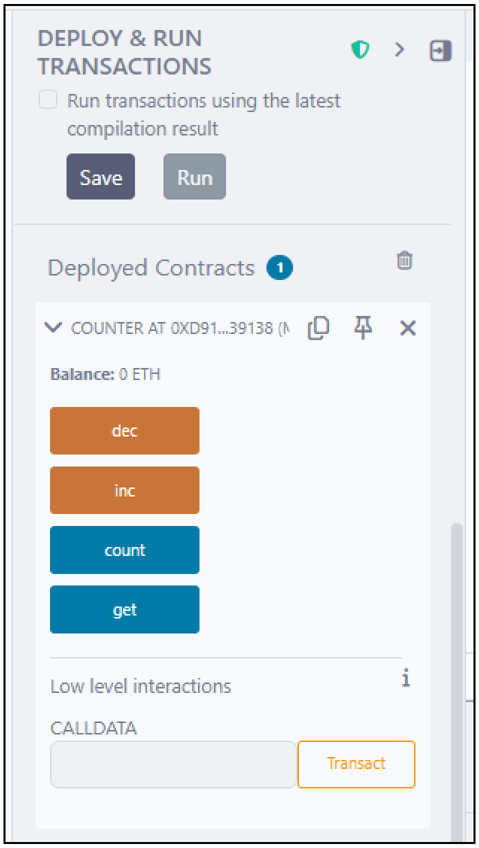
```
// SPDX-License-Identifier: MIT


// Snehal Patil D20A  41
pragma solidity ^0.8.3;

contract Counter {
    uint public count;

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

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

    // Function to decrement count by 1
    function dec() public {
```



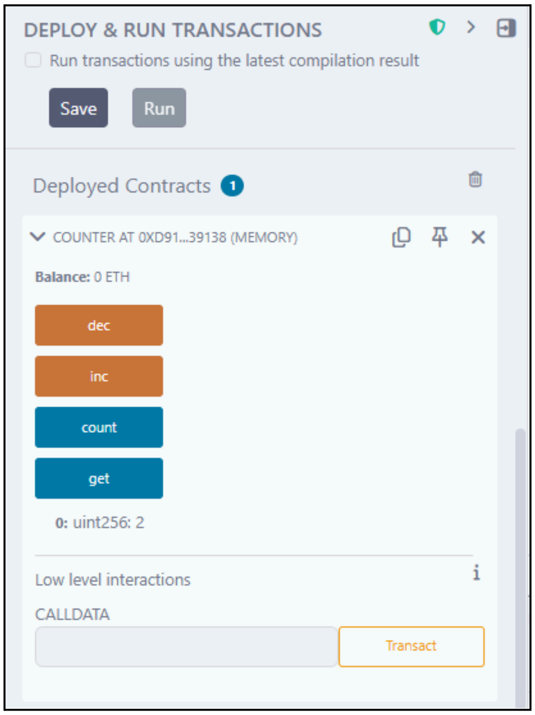
0: uint256: 1

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


// Snehal Patil D20A  41
pragma solidity ^0.8.3;

contract Counter {
    uint public count;

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

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

    // Function to decrement count by 1
    function dec() public {
```

```
to: Counter.get() data: 0x6d4...ce63c
```

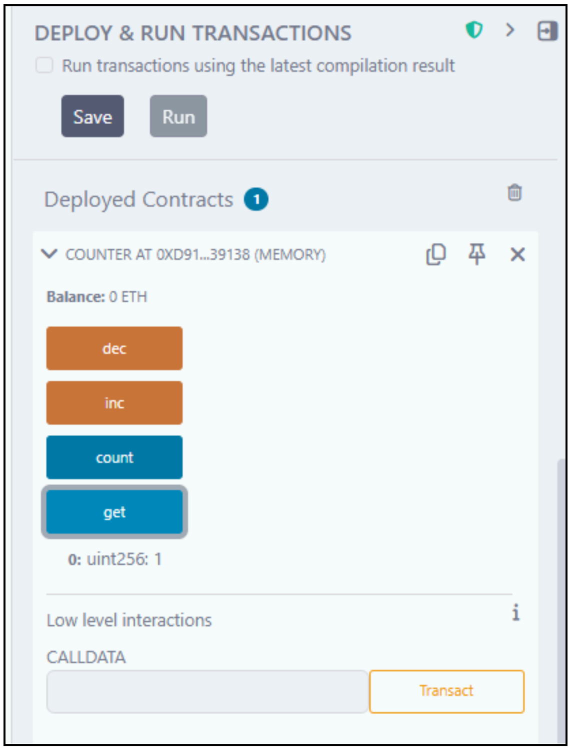

DEPLOY & RUN TRANSACTIONS
Run transactions using the latest compilation result
Save
Run
Deployed Contracts 1
COUNTER AT 0XD91...39138 (M
Balance: 0 ETH
dec
inc
count
get
Low level interactions
CALLDATA
Transact

- Tutorial no. 1 – Increment


DEPLOY & RUN TRANSACTIONS
Run transactions using the latest compilation result
Save
Run
Deployed Contracts 1
COUNTER AT 0XD91...39138 (MEMORY)
Balance: 0 ETH
dec
inc
count
get
Low level interactions
CALLDATA
Transact

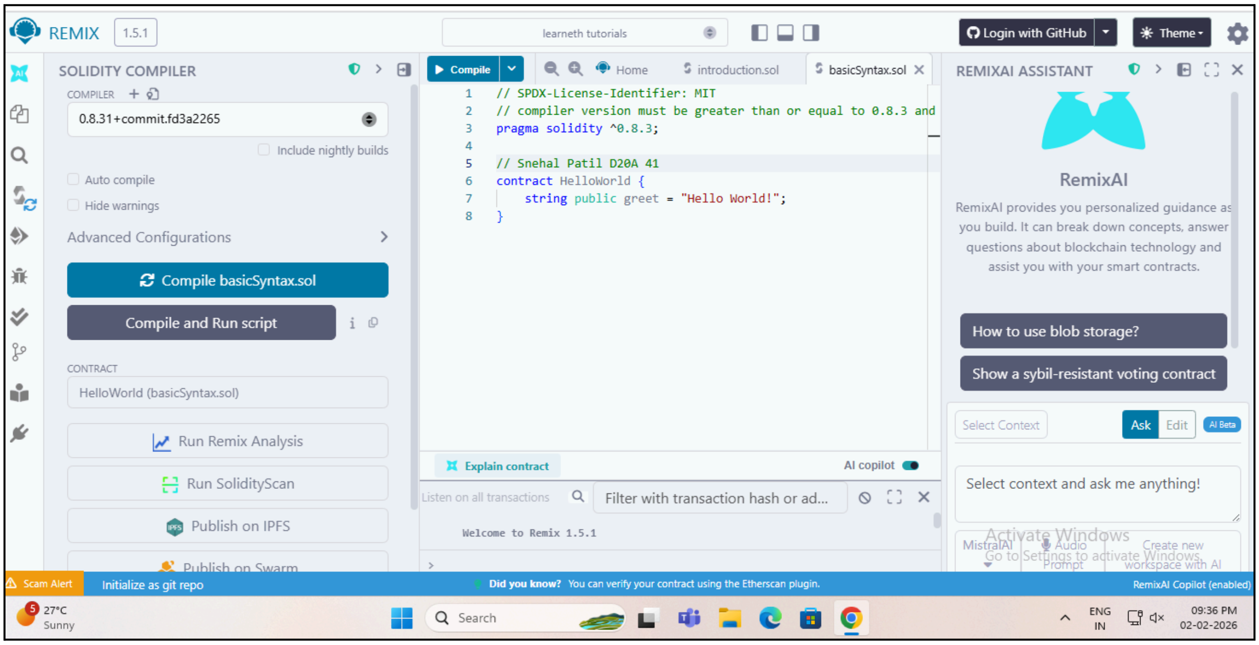
```
0: uint256: 2
```

- Tutorial no. 1 – Decrement


DEPLOY & RUN TRANSACTIONS
Run transactions using the latest compilation result
Save
Run
Deployed Contracts 1
COUNTER AT 0XD91...39138 (MEMORY)
Balance: 0 ETH
dec
inc
count
get
Low level interactions
CALLDATA
Transact

```
0: uint256: 1
```

- Tutorial no. 2

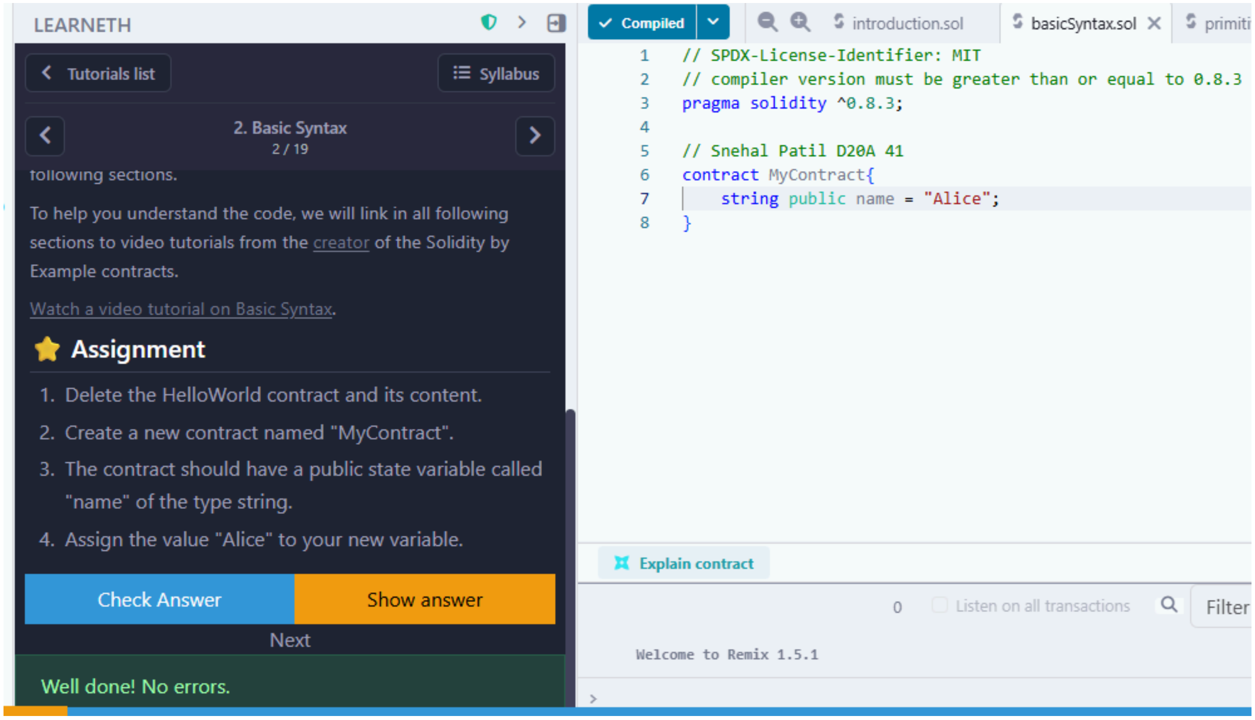


```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.3 and
pragma solidity ^0.8.3;

// Snehal Patil D20A 41
contract HelloWorld {
    string public greet = "Hello World!";
}
```

RemixAI provides you personalized guidance as you build. It can break down concepts, answer questions about blockchain technology and assist you with your smart contracts.

```
Welcome to Remix 1.5.1
```

Did you know? You can verify your contract using the Etherscan plugin.



2. Basic Syntax
2 / 19

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

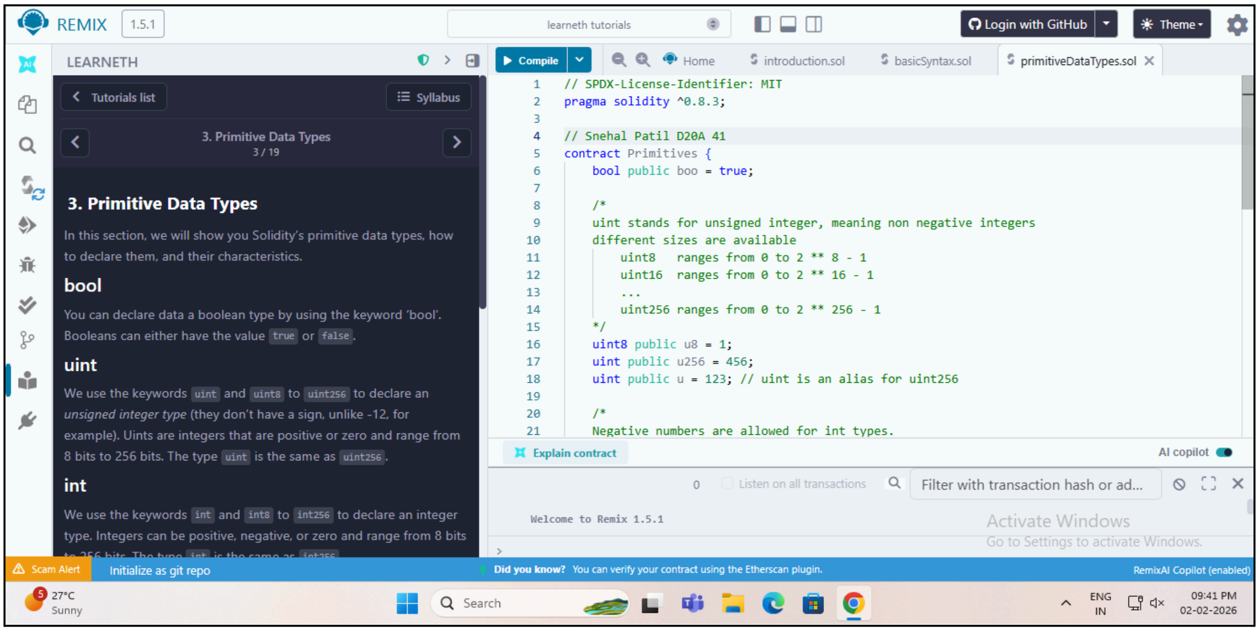
1. Delete the HelloWorld contract and its content.
2. Create a new contract named "MyContract".
3. The contract should have a public state variable called "name" of the type string.
4. Assign the value "Alice" to your new variable.

Well done! No errors.

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

// Snehal Patil D20A 41
contract MyContract{
    string public name = "Alice";
}
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 3



3. Primitive Data Types

In this section, we will show you Solidity's primitive data types, how to declare them, and their characteristics.

bool

You can declare data a boolean type by using the keyword 'bool'. Booleans can either have the value `true` or `false`.

uint

We use the keywords `uint` and `uint8` to `uint256` to declare an *unsigned integer type* (they don't have a sign, unlike -12, for example). Uints are integers that are positive or zero and range from 8 bits to 256 bits. The type `uint` is the same as `uint256`.

int

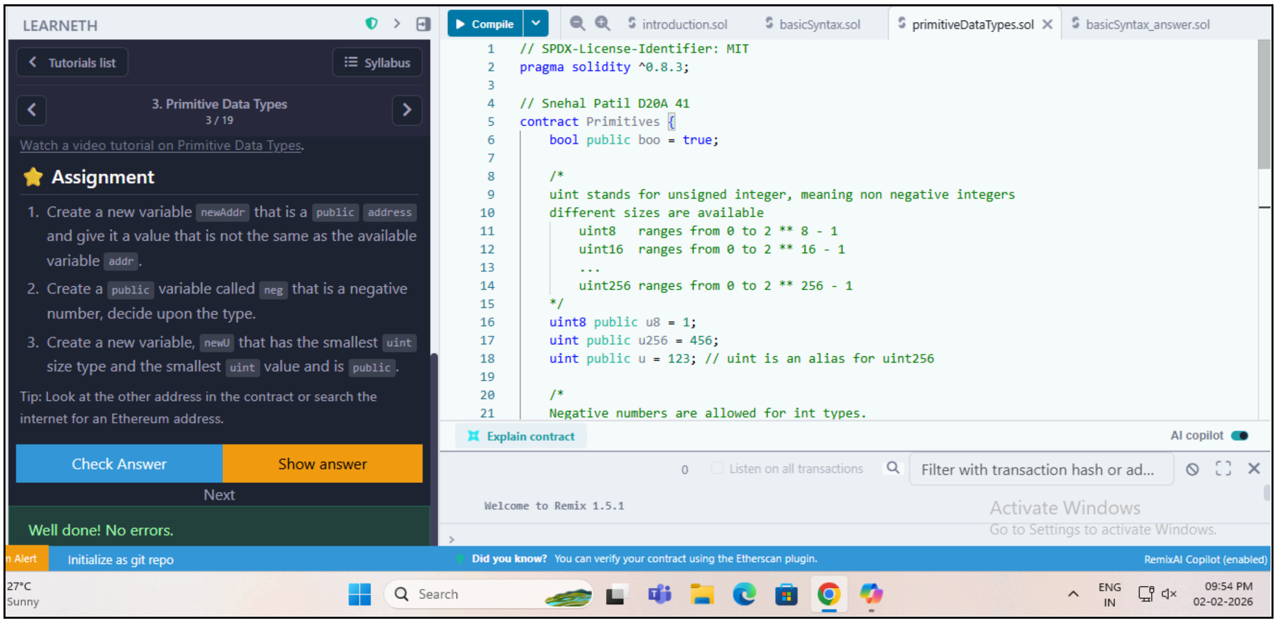
We use the keywords `int` and `int8` to `int256` to declare an integer type. Integers can be positive, negative, or zero and range from 8 bits to 256 bits. The type `int` is the same as `int256`.

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

// Snehal Patil D20A 41
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.
```

```
Welcome to Remix 1.5.1
```

Did you know? You can verify your contract using the Etherscan plugin.

Activate Windows
Go to Settings to activate Windows.



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.

Check Answer | Show answer

Next

Well done! No errors.

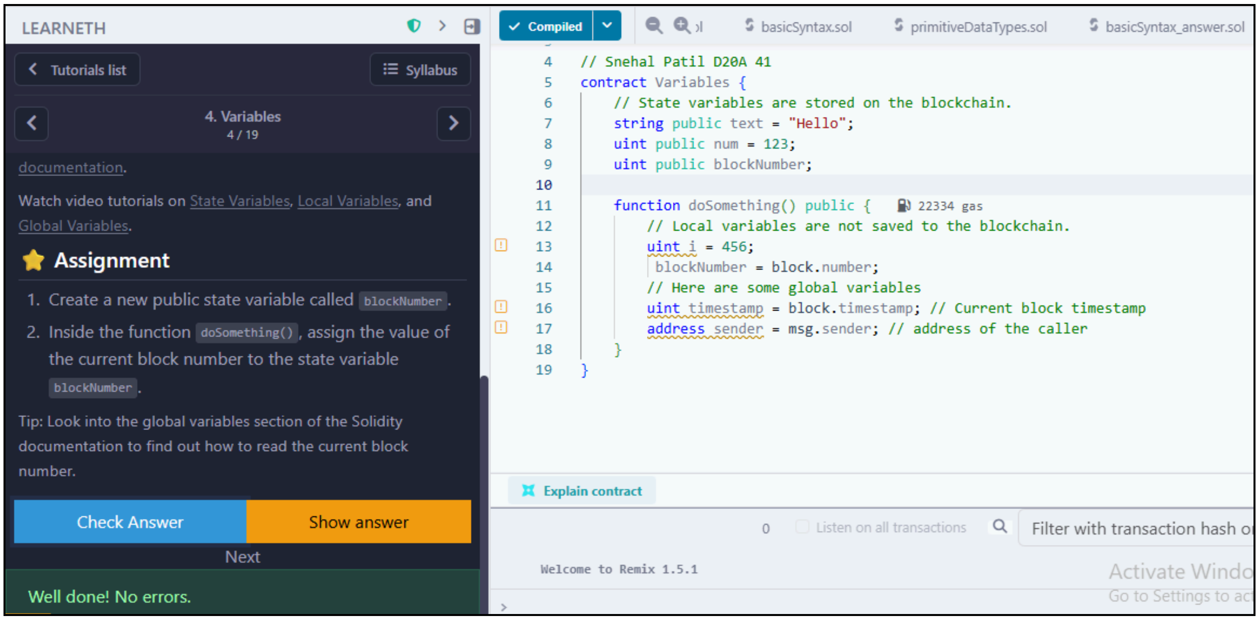
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

// Snehal Patil D20A 41
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.
```

```
Welcome to Remix 1.5.1
```

Did you know? You can verify your contract using the Etherscan plugin.

Activate Windows
Go to Settings to activate Windows.

- Tutorial no. 4



4. Variables
4 / 19

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.

Check Answer | Show answer

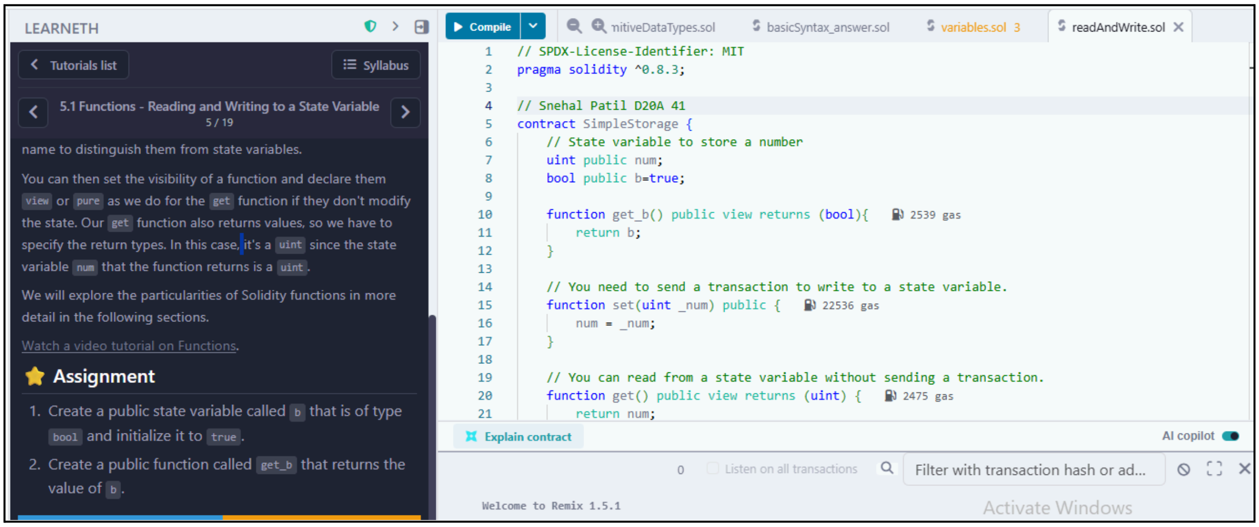
Next

Well done! No errors.

```
// Snehal Patil D20A 41
contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Hello";
    uint public num = 123;
    uint public blockNumber;

    function doSomething() public {
        // Local variables are not saved to the blockchain.
        uint i = 456;
         blockNumber = block.number;
        // Here are some global variables
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
    }
}
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 5



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

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.

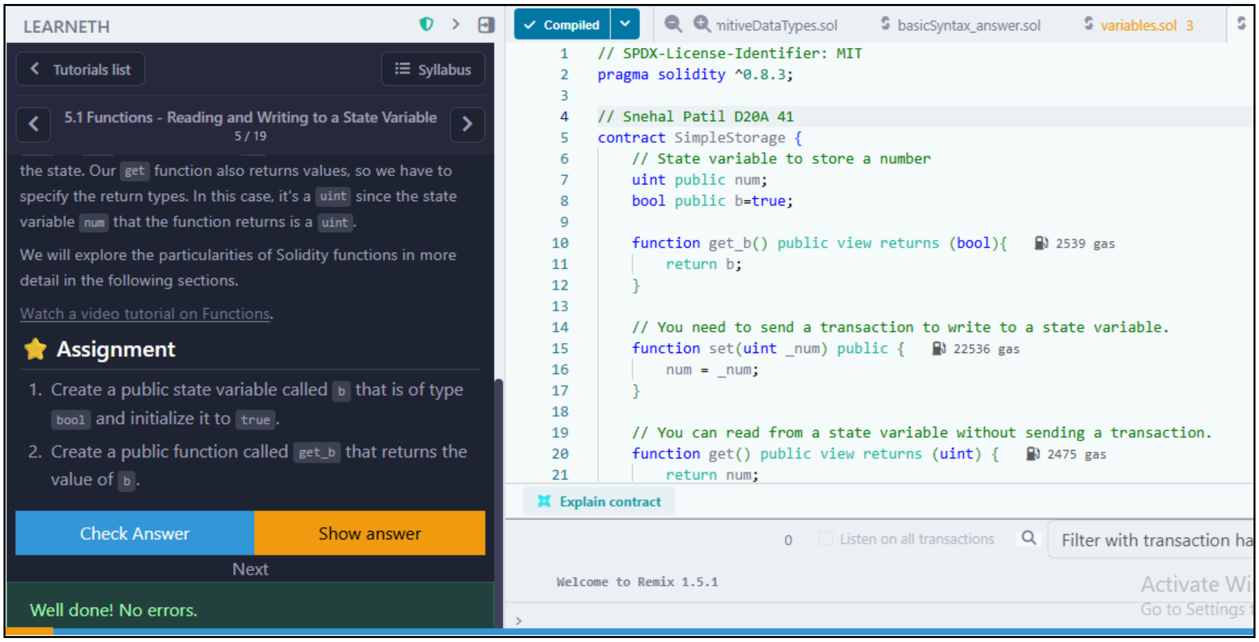
Watch a video tutorial on Functions.

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`.

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

// Snehal Patil D20A 41
contract SimpleStorage {
    // State variable to store a number
    uint public num;
    bool public b=true;

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

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

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

```
Welcome to Remix 1.5.1
```



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

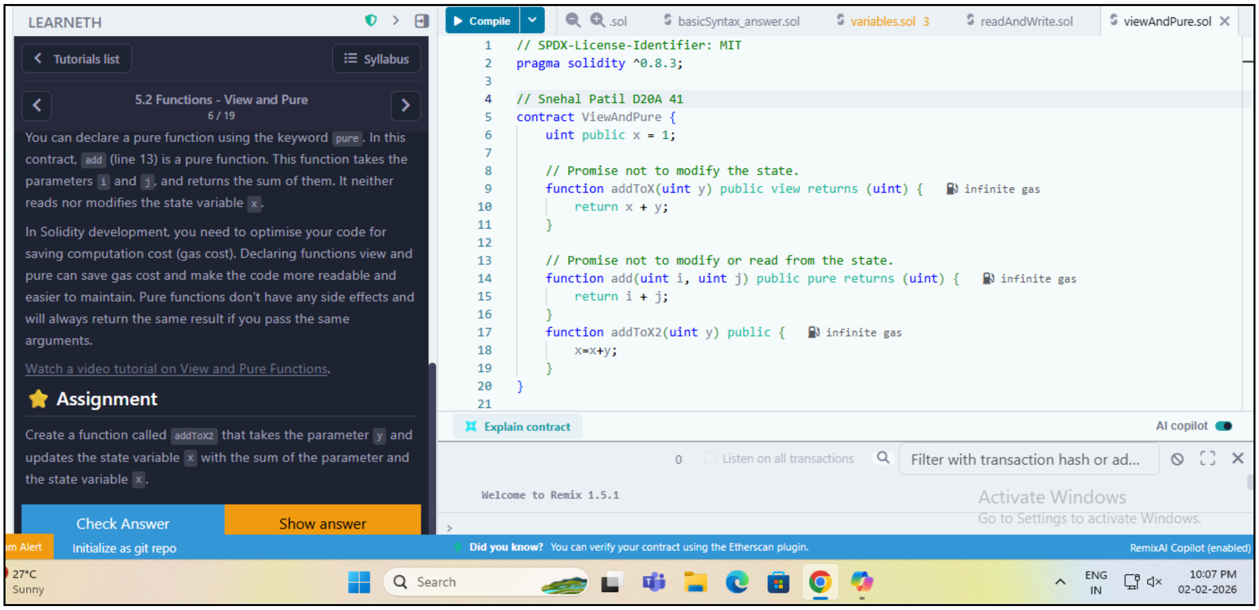
1. Create a public state variable called `b` that is of type `bool` and initialize it to `true`.
2. Create a public function called `get_b` that returns the value of `b`.

Well done! No errors.

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

// Snehal Patil D20A 41
contract SimpleStorage {
    // State variable to store a number
    uint public num;
    bool public b=true;

    function get_b() public view returns (bool){    2539 gas
        return b;
    }

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {    22536 gas
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {    2475 gas
        return num;
```

```
Welcome to Remix 1.5.1
```

- 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

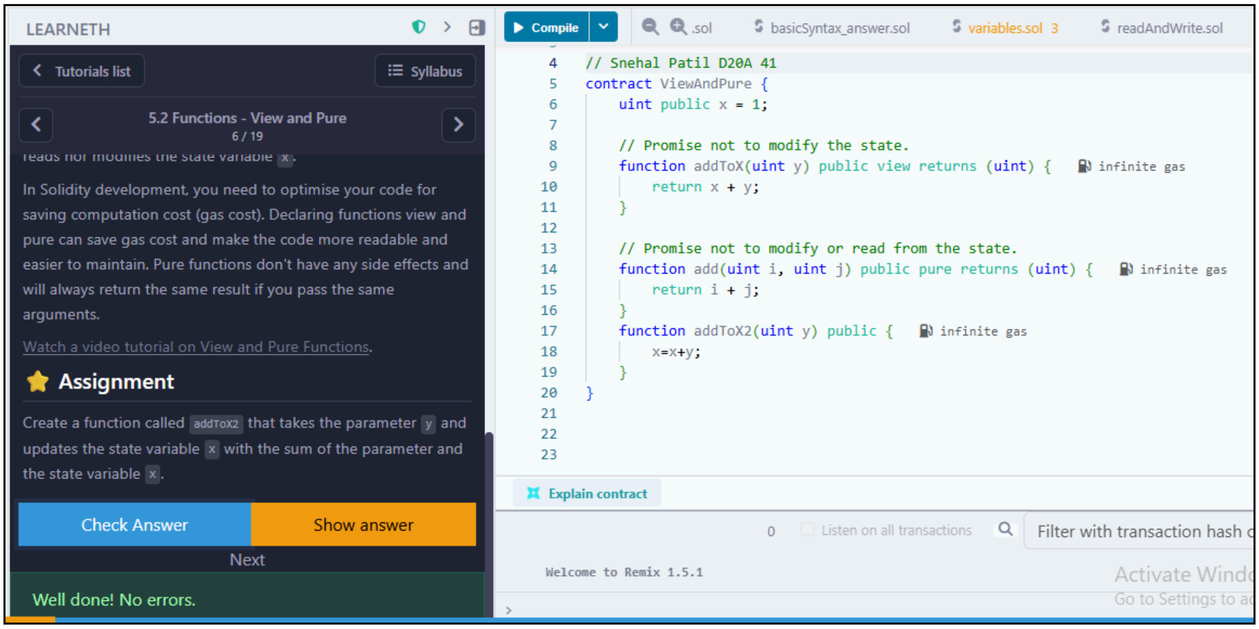
Create a function called `addToX2` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`.

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.3;
3
4   // Snehal Patil D20A 41
5   contract ViewAndPure {
6       uint public x = 1;
7
8       // Promise not to modify the state.
9       function addToX(uint y) public view returns (uint) {    infinite gas
10          return x + y;
11      }
12
13      // Promise not to modify or read from the state.
14      function add(uint i, uint j) public pure returns (uint) {    infinite gas
15          return i + j;
16      }
17      function addToX2(uint y) public {    infinite gas
18          x=x+y;
19      }
20  }
21
```

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

```
Welcome to Remix 1.5.1
```

Activate Windows
Go to Settings to activate Windows.
Check Answer
Show answer
m Alert
Initialize as git repo
Did you know? You can verify your contract using the Etherscan plugin.
RemixAI Copilot (enabled)
27°C
Sunny
Search
ENG
IN
10:07 PM
02-02-2026



5.2 Functions - View and Pure
6 / 19

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

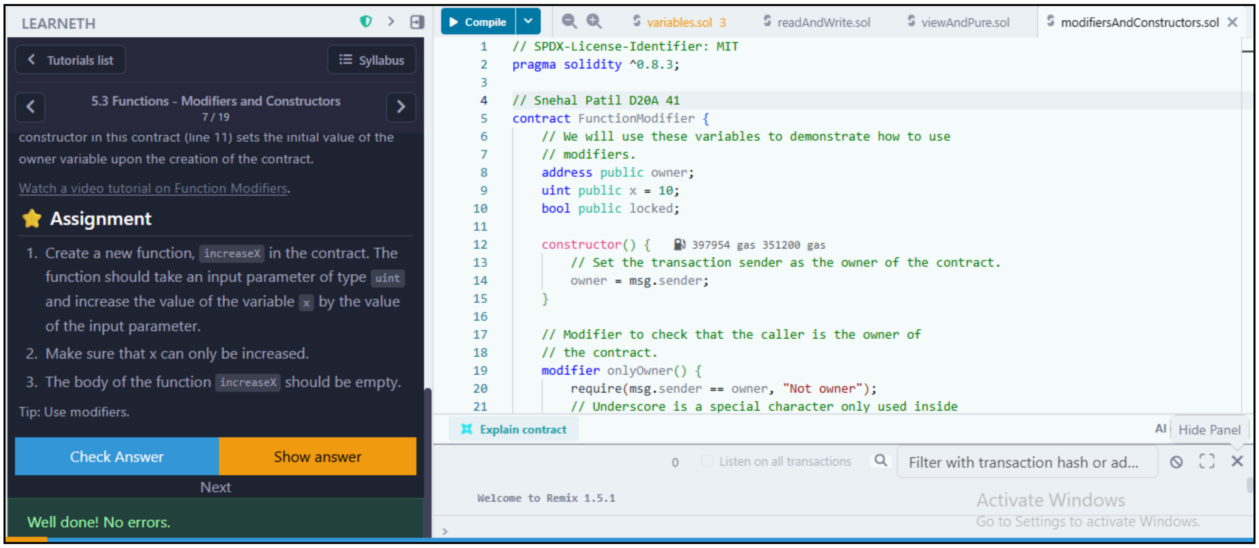
Create a function called `addToX2` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`.

```
4   // Snehal Patil D20A 41
5   contract ViewAndPure {
6       uint public x = 1;
7
8       // Promise not to modify the state.
9       function addToX(uint y) public view returns (uint) {    infinite gas
10          return x + y;
11      }
12
13      // Promise not to modify or read from the state.
14      function add(uint i, uint j) public pure returns (uint) {    infinite gas
15          return i + j;
16      }
17      function addToX2(uint y) public {    infinite gas
18          x=x+y;
19      }
20  }
21
22
23
```

Well done! No errors.

```
Welcome to Remix 1.5.1
```

- Tutorial no. 7



5.3 Functions - Modifiers and Constructors
7 / 19

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.

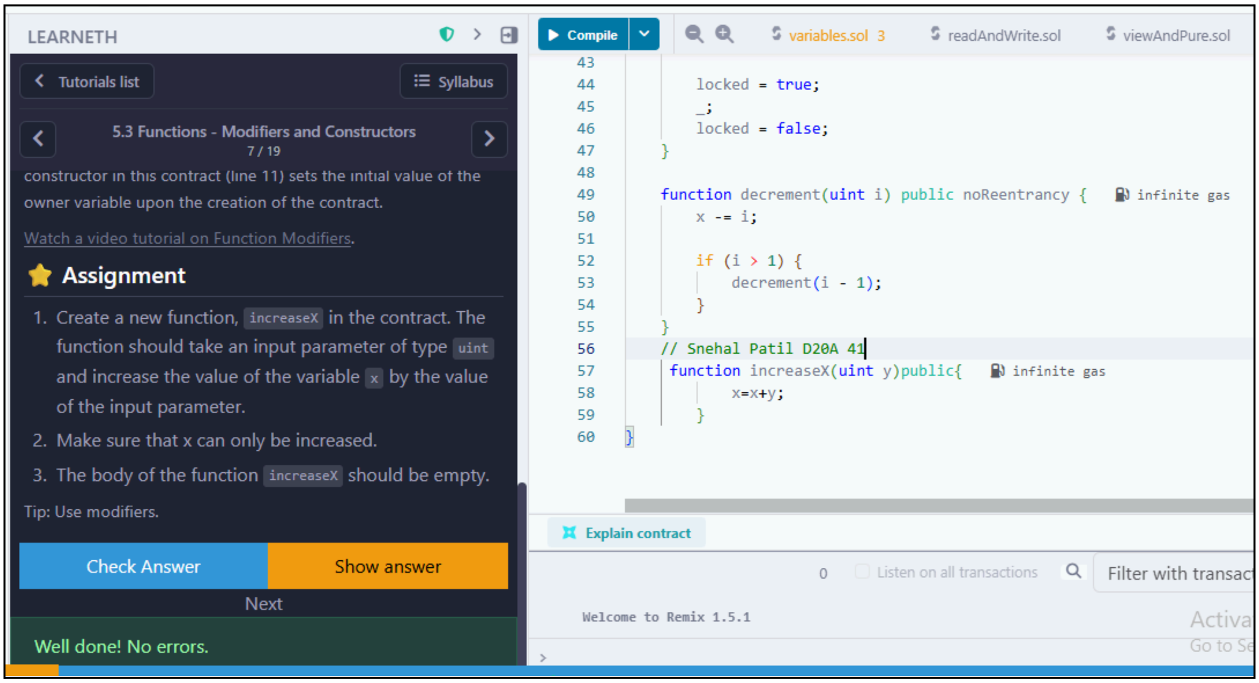
Check Answer | Show answer

Next

Well done! No errors.

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

// Snehal Patil D20A 41
contract FunctionModifier {
    // We will use these variables to demonstrate how to use
    // modifiers.
    address public owner;
    uint public x = 10;
    bool public locked;

    constructor() {
        // 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
```

```
Welcome to Remix 1.5.1
```



5.3 Functions - Modifiers and Constructors
7 / 19

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

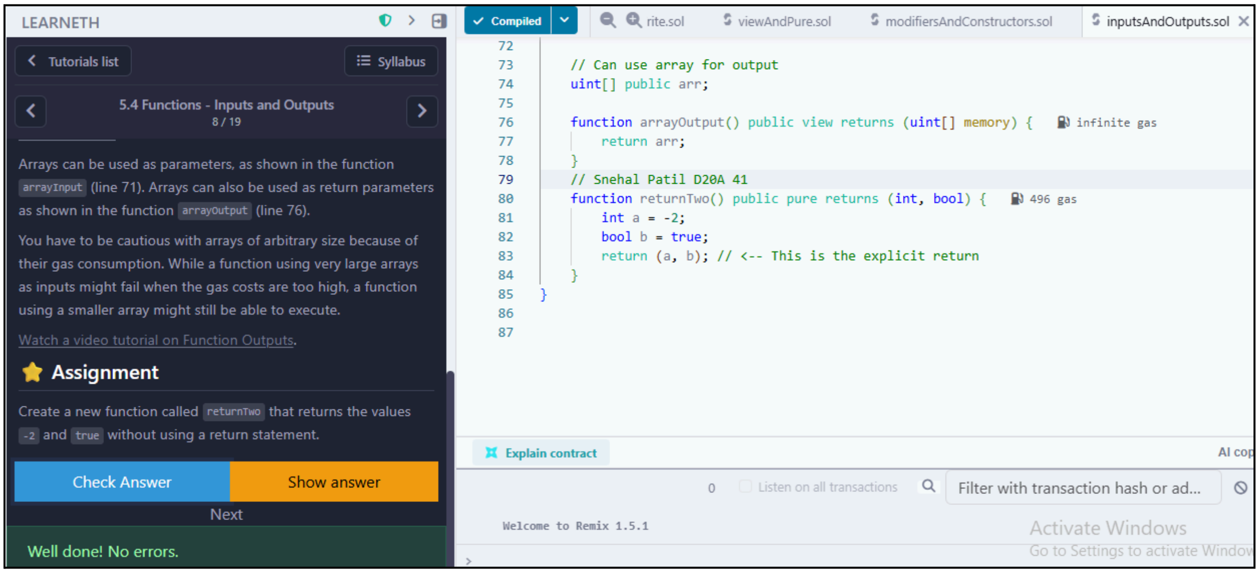
Well done! No errors.

```
        locked = true;
        _;
        locked = false;
    }

    function decrement(uint i) public noReentrancy {
        x -= i;

        if (i > 1) {
            decrement(i - 1);
        }
    }
    // Snehal Patil D20A 41
     function increaseX(uint y)public{
        x=x+y;
        }
}
```

```
Welcome to Remix 1.5.1
```

‘

- Tutorial no. 8



5.4 Functions - Inputs and Outputs
8 / 19

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

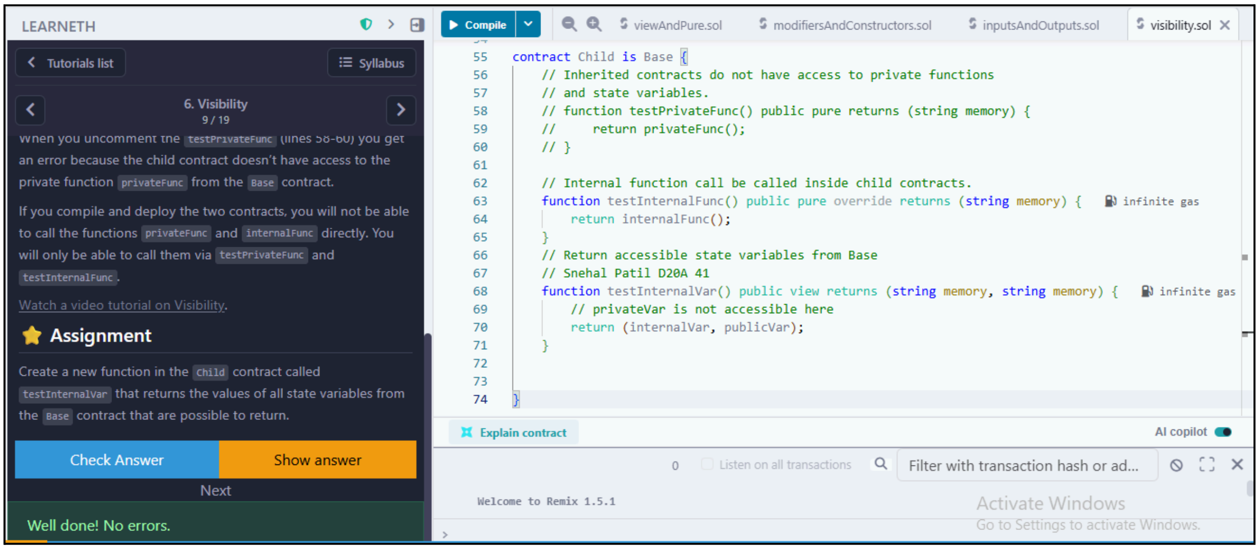
Create a new function called `returnTwo` that returns the values `-2` and `true` without using a return statement.

Well done! No errors.

```
72
73      // Can use array for output
74      uint[] public arr;
75
76      function arrayOutput() public view returns (uint[] memory) {
77          return arr;
78      }
79      // Snehal Patil D20A 41
80      function returnTwo() public pure returns (int, bool) {
81          int a = -2;
82          bool b = true;
83          return (a, b); // <-- This is the explicit return
84      }
85  }
86
87
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 9



6. Visibility
9 / 19

when you uncomment the `testPrivateFunc` (lines 58-60) you get an error because the child contract doesn't have access to the private function `privateFunc` from the `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

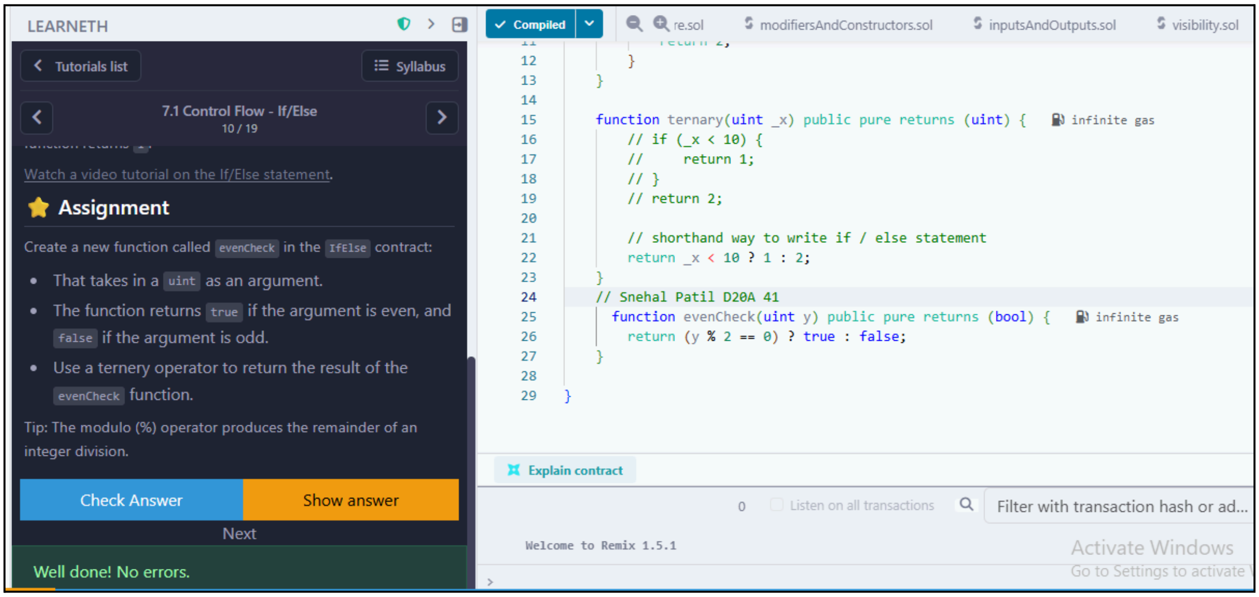
Create a new function in the `Child` contract called `testInternalVar` that returns the values of all state variables from the `Base` contract that are possible to return.

Well done! No errors.

```
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 memory) {
64          return internalFunc();
65      }
66      // Return accessible state variables from Base
67      // Snehal Patil D20A 41
68      function testInternalVar() public view returns (string memory, string memory) {
69          // privateVar is not accessible here
70          return (internalVar, publicVar);
71      }
72
73
74  }
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 10


LEARNETH
Tutorials list
Syllabus
7.1 Control Flow - If/Else
10 / 19
Compiled
re.sol
modifiersAndConstructors.sol
inputsAndOutputs.sol
visibility.sol
infinite gas
infinite gas
Explain contract
0
Listen on all transactions
Filter with transaction hash or ad...
Activate Windows
Go to Settings to activate

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

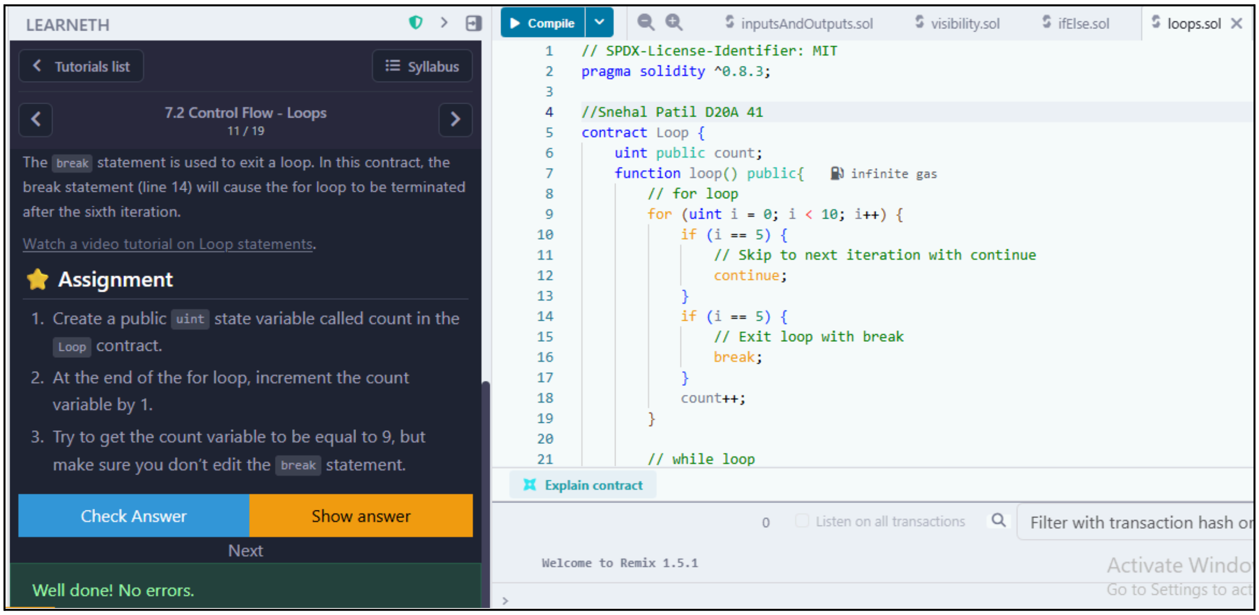
Next

Well done! No errors.

```
            return 2;
        }
    }

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

        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }
    // Snehal Patil D20A 41
     function evenCheck(uint y) public pure returns (bool) {
        return (y % 2 == 0) ? true : false;
    }
}
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 11



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

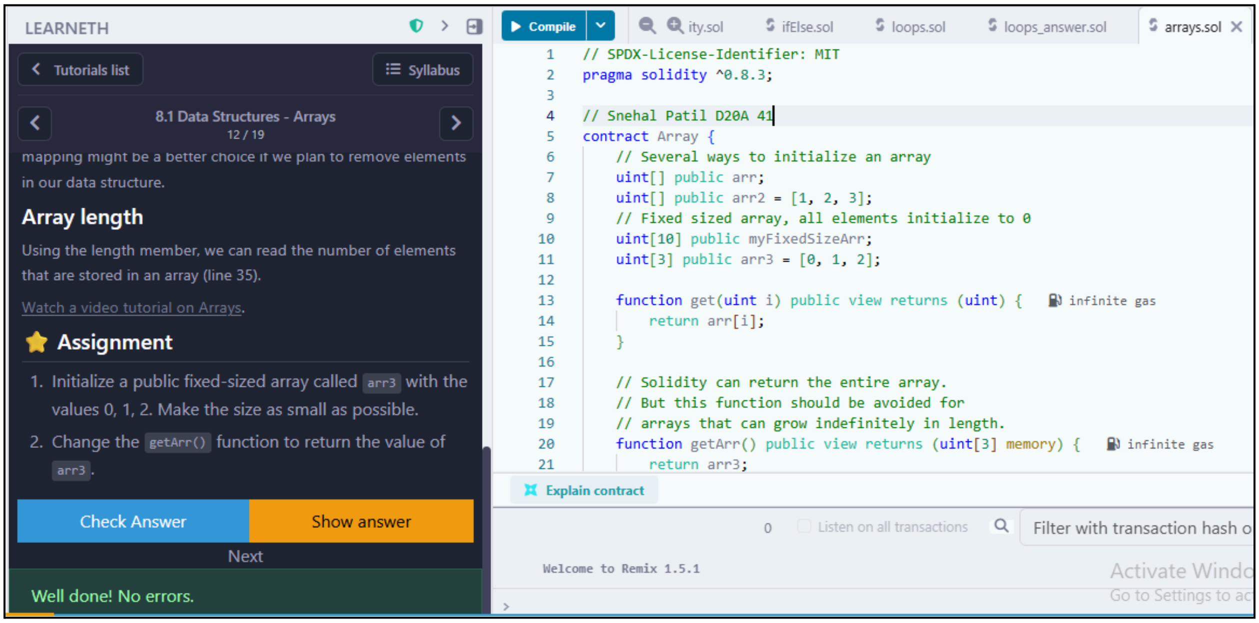
Next

Well done! No errors.

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.3;
3
4   //Snehal Patil D20A 41
5   contract Loop {
6       uint public count;
7       function loop() public{
8           // for loop
9           for (uint i = 0; i < 10; i++) {
10              if (i == 5) {
11                  // Skip to next iteration with continue
12                  continue;
13              }
14              if (i == 5) {
15                  // Exit loop with break
16                  break;
17              }
18              count++;
19          }
20
21          // while loop
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 12



8.1 Data Structures - Arrays
12 / 19

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

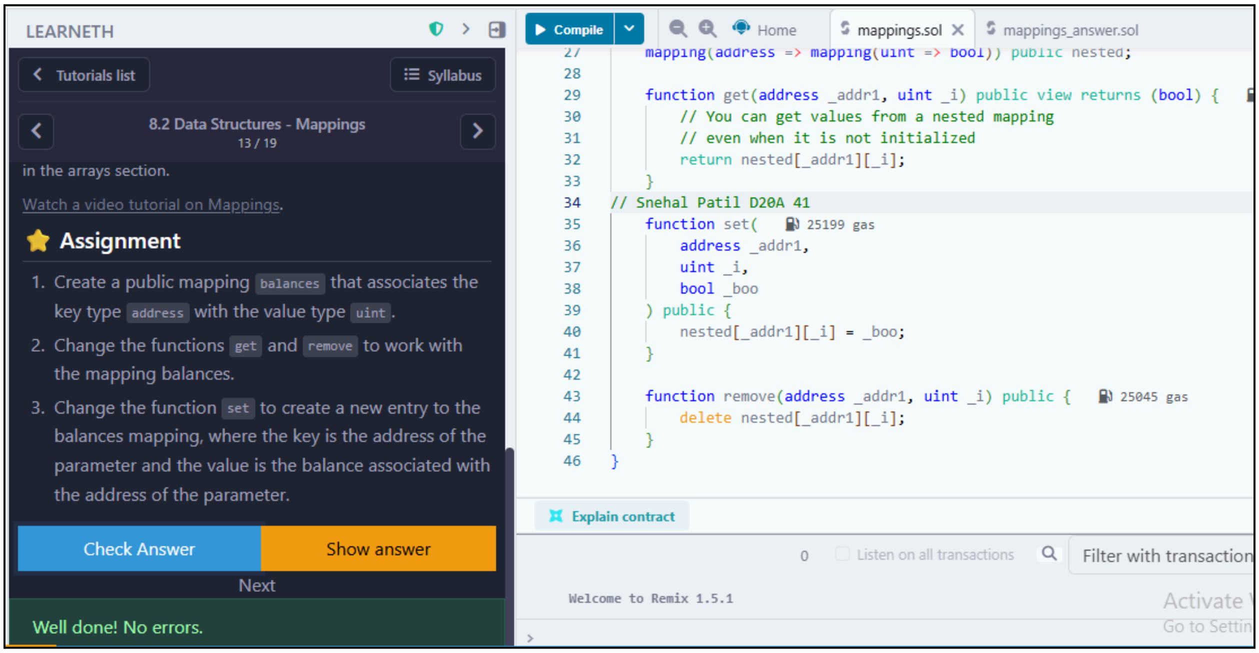
1. Initialize a public fixed-sized array called `arr3` with the values 0, 1, 2. Make the size as small as possible.
2. Change the `getArr()` function to return the value of `arr3`.

Well done! No errors.

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.3;
3
4   // Snehal Patil D20A 41
5   contract Array {
6       // Several ways to initialize an array
7       uint[] public arr;
8       uint[] public arr2 = [1, 2, 3];
9       // Fixed sized array, all elements initialize to 0
10      uint[10] public myFixedSizeArr;
11      uint[3] public arr3 = [0, 1, 2];
12
13      function get(uint i) public view returns (uint) {    infinite gas
14          return arr[i];
15      }
16
17      // Solidity can return the entire array.
18      // But this function should be avoided for
19      // arrays that can grow indefinitely in length.
20      function getArr() public view returns (uint[3] memory) {    infinite gas
21          return arr3;
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 13



8.2 Data Structures - Mappings
13 / 19

in the arrays section.

Watch a video tutorial on Mappings.

Assignment

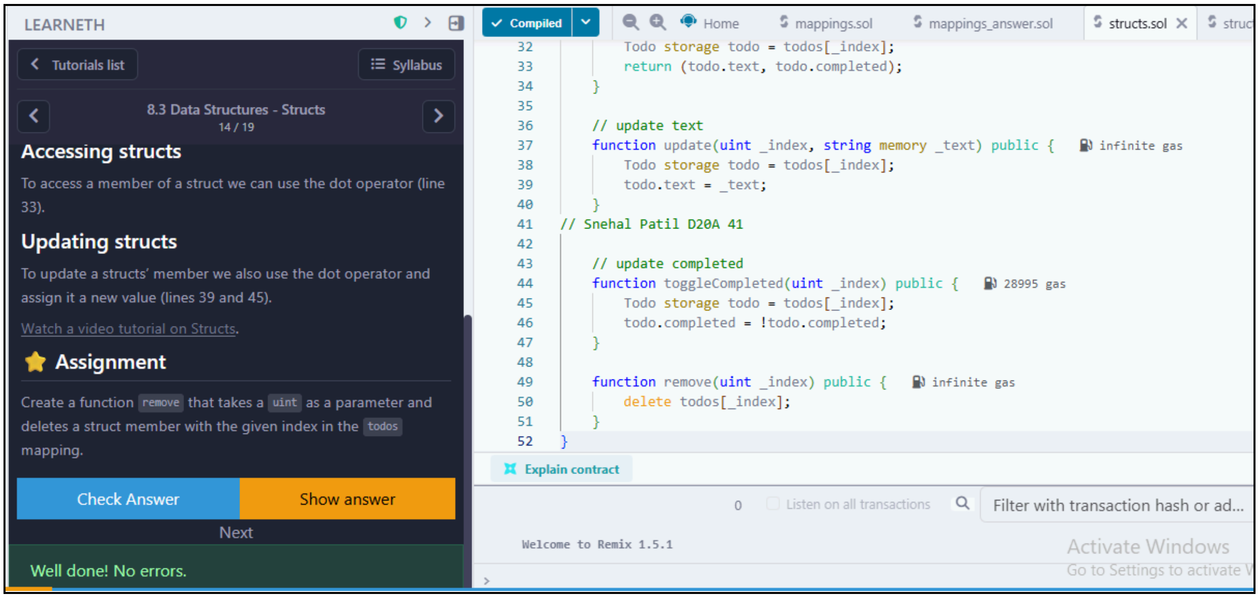
1. Create a public mapping `balances` that associates the key type `address` with the value type `uint`.
2. Change the functions `get` and `remove` to work with the mapping balances.
3. Change the function `set` to create a new entry to the balances mapping, where the key is the address of the parameter and the value is the balance associated with the address of the parameter.

Well done! No errors.

```
27      mapping(address => mapping(uint => bool)) public nested;
28
29      function get(address _addr1, uint _i) public view returns (bool) {
30          // You can get values from a nested mapping
31          // even when it is not initialized
32          return nested[_addr1][_i];
33      }
34  // Snehal Patil D20A 41
35      function set(    25199 gas
36          address _addr1,
37          uint _i,
38          bool _boo
39      ) public {
40          nested[_addr1][_i] = _boo;
41      }
42
43      function remove(address _addr1, uint _i) public {    25045 gas
44          delete nested[_addr1][_i];
45      }
46  }
```

```
Welcome to Remix 1.5.1
```

- Tutorial no. 14



8.3 Data Structures - Structs
14 / 19

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

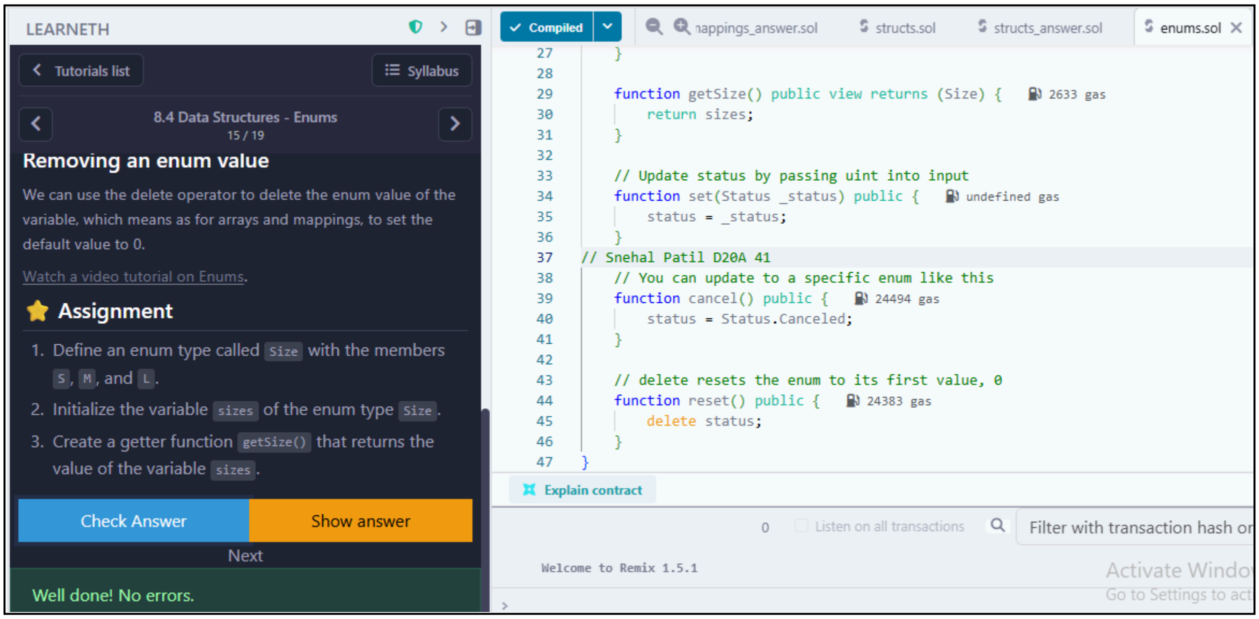
Create a function `remove` that takes a `uint` as a parameter and deletes a struct member with the given index in the `todos` mapping.

Well done! No errors.

```
32         Todo storage todo = todos[_index];
33         return (todo.text, todo.completed);
34     }
35
36     // update text
37     function update(uint _index, string memory _text) public {    infinite gas
38         Todo storage todo = todos[_index];
39         todo.text = _text;
40     }
41 // Snehal Patil D20A 41
42
43     // update completed
44     function toggleCompleted(uint _index) public {    28995 gas
45         Todo storage todo = todos[_index];
46         todo.completed = !todo.completed;
47     }
48
49     function remove(uint _index) public {    infinite gas
50         delete todos[_index];
51     }
52 }
```

```
0
Welcome to Remix 1.5.1
```

- Tutorial no. 15



8.4 Data Structures - Enums
15 / 19

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

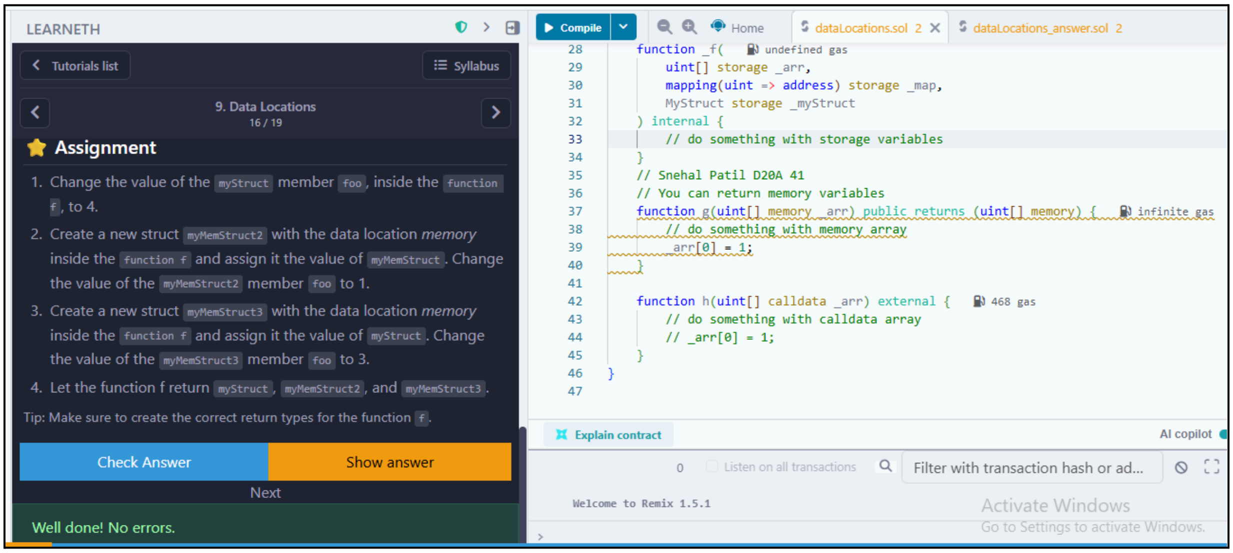
1. Define an enum type called `Size` with the members `S`, `M`, and `L`.
2. Initialize the variable `sizes` of the enum type `Size`.
3. Create a getter function `getSize()` that returns the value of the variable `sizes`.

Well done! No errors.

```
    }

    function getSize() public view returns (Size) {    2633 gas
        return sizes;
    }

    // Update status by passing uint into input
    function set(Status _status) public {    undefined gas
        status = _status;
    }
// Snehal Patil D20A 41
    // You can update to a specific enum like this
    function cancel() public {    24494 gas
        status = Status.Canceled;
    }

    // delete resets the enum to its first value, 0
    function reset() public {    24383 gas
        delete status;
    }
}
```

```
0
Welcome to Remix 1.5.1
```

- 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`.

Check Answer | Show answer

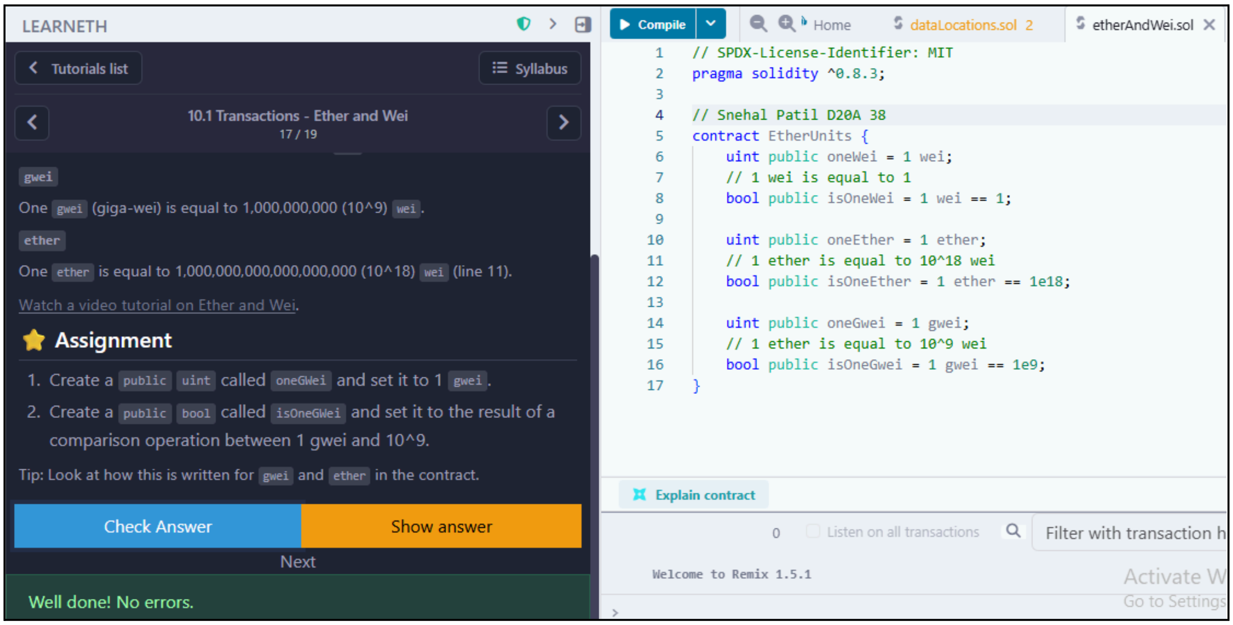
Next

Well done! No errors.

```
    function _f(
        uint[] storage _arr,
        mapping(uint => address) storage _map,
        MyStruct storage _myStruct
    ) internal {
        // do something with storage variables
    }
    // Snehal Patil D20A 41
    // You can return memory variables
    function g(uint[] memory _arr) public returns (uint[] memory) {
        // do something with memory array
        _arr[0] = 1;
    }

    function h(uint[] calldata _arr) external {
        // do something with calldata array
        // _arr[0] = 1;
    }
}

```

```
Welcome to Remix 1.5.1
```

- 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.

Check Answer | Show answer

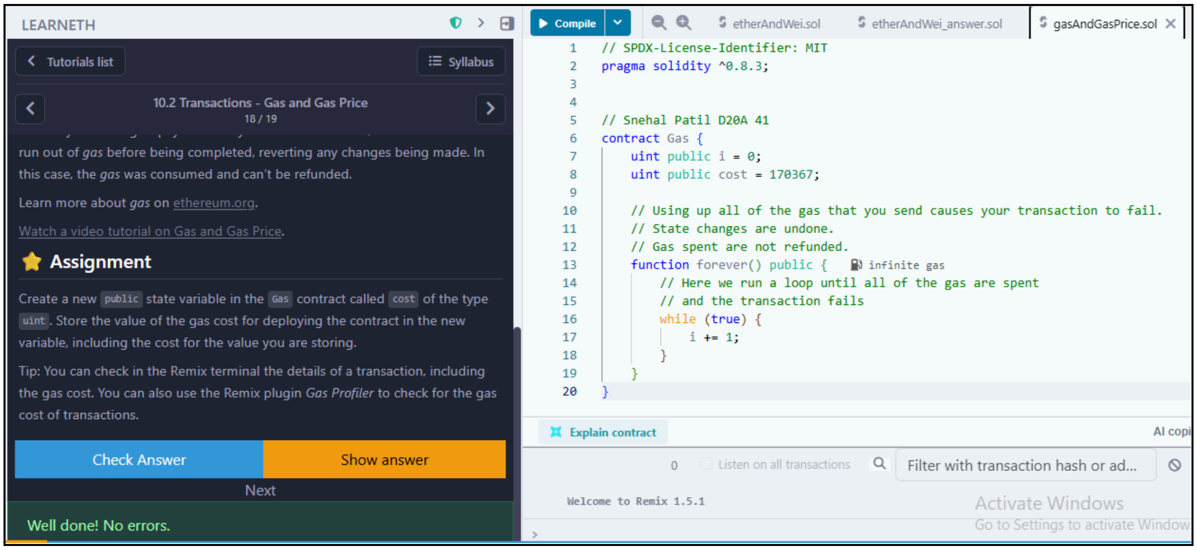
Next

Well done! No errors.

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

// Snehal Patil D20A 38
contract EtherUnits {
    uint public oneWei = 1 wei;
    // 1 wei is equal to 1
    bool public isOneWei = 1 wei == 1;

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

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

```
Welcome to Remix 1.5.1
```

- Tutorial no. 18



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.

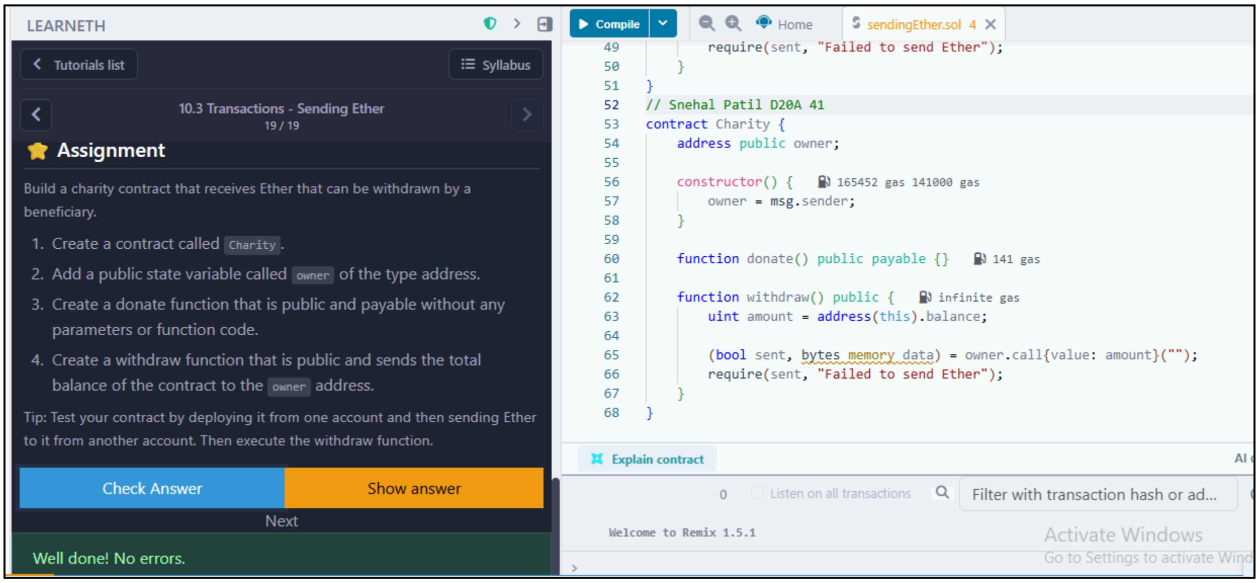
Well done! No errors.

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


// Snehal Patil D20A 41
contract Gas {
    uint public i = 0;
    uint public cost = 170367;

    // Using up all of the gas that you send causes your transaction to fail.
    // 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;
        }
    }
}
```

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

```
Welcome to Remix 1.5.1
```

- Tutorial no. 19



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.

```
        require(sent, "Failed to send Ether");
    }
}
// Snehal Patil D20A 41
contract Charity {
    address public owner;

    constructor() {    165452 gas 141000 gas
        owner = msg.sender;
    }

    function donate() public payable {}    141 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");
    }
}
```

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

```
Welcome to Remix 1.5.1
```

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.