Experiment: 04

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.

- **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.

4. 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:

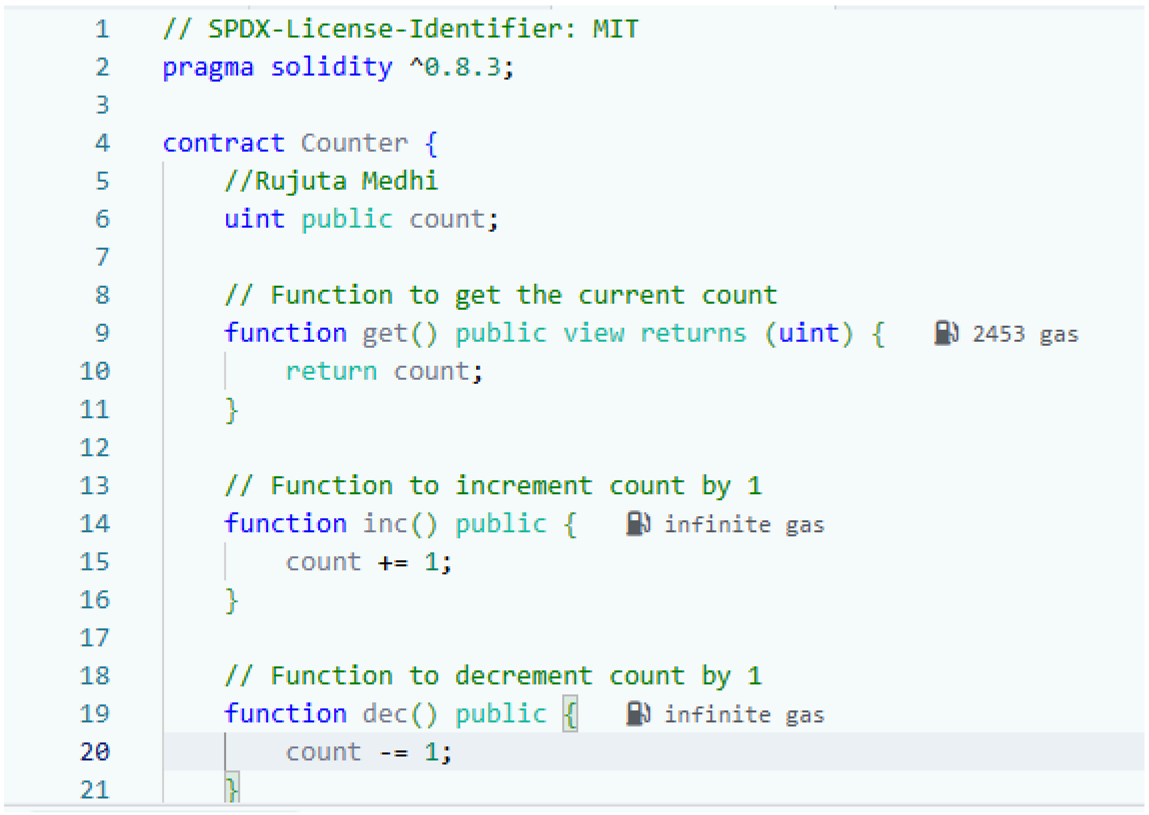
- **storage**: Data stored permanently on the blockchain. Examples: state variables.
- **memory**: Temporary data storage that exists only while a function is executing. Used for local variables and function inputs.
- **calldata**: A non-modifiable and non-persistent location used for external function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

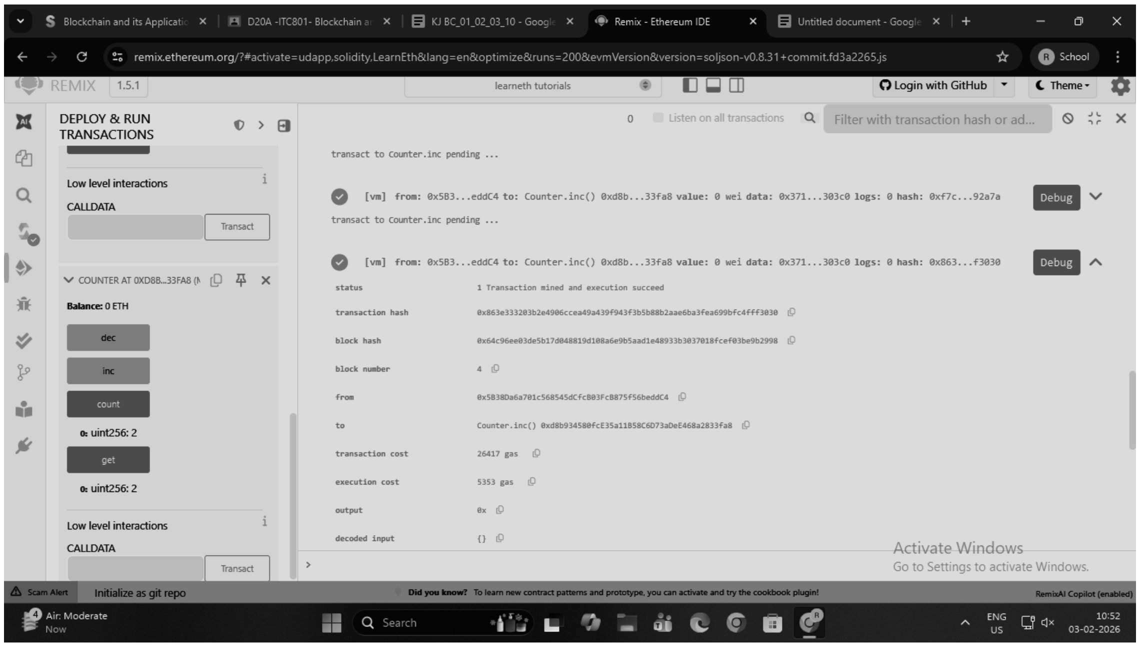
- **Ether and Wei**: Ether is the main currency in Ethereum. All values are measured in Wei, the smallest unit (1 Ether = 10^{18} Wei). This ensures high precision in financial transactions.
- **Gas and Gas Price**: Every transaction consumes gas, which represents computational effort. The gas price 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:

1. Introduction



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

contract Counter {
    //Rujuta Medhi
    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;
    }
```



DEPLOY & RUN TRANSACTIONS

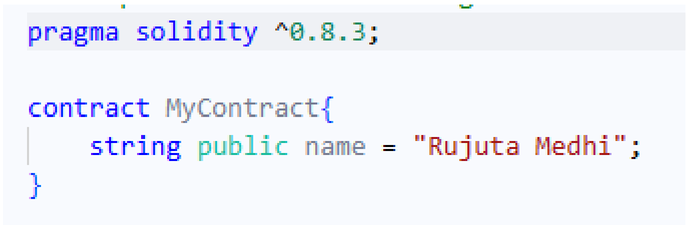
Low level interactions
CALLDATA
Transact

COUNTER AT 0XD8B...33FA8
Balance: 0 ETH
dec
inc
count
0: uint256: 2
get
0: uint256: 2

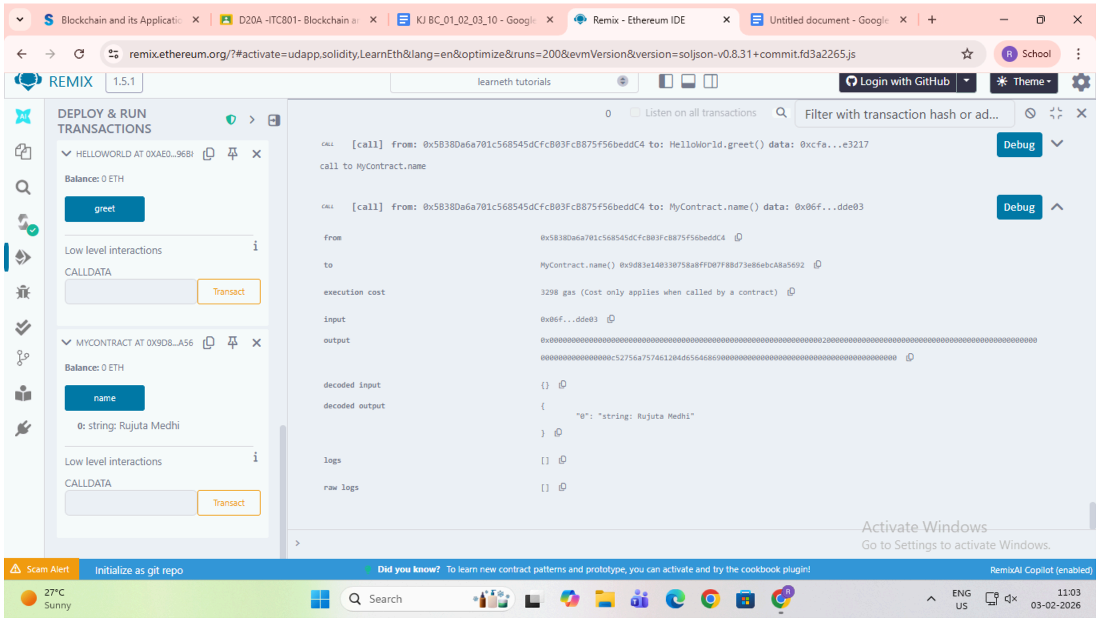
Low level interactions
CALLDATA
Transact

```
transact to Counter.inc pending ...

[vm] from: 0x5B3...eddC4 to: Counter.inc() 0xd8b...33fa8 value: 0 wei data: 0x371...303c0 logs: 0 hash: 0xf7c...92a7a
transact to Counter.inc pending ...

[vm] from: 0x5B3...eddC4 to: Counter.inc() 0xd8b...33fa8 value: 0 wei data: 0x371...303c0 logs: 0 hash: 0x863...f3030
status              1 Transaction mined and execution succeed
transaction hash    0x863e333203b2e4906ccea49a439f943f3b5b88b2aae6ba3fea699bfc4fff3030
block hash          0x64c96ee03de5b17d048819d108a6e9b5aad1e48933b3037018fcef03be9b2998
block number        4
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Counter.inc() 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
transaction cost    26417 gas
execution cost      5353 gas
output              0x
decoded input       {}
```

2. Basic Syntax



```
pragma solidity ^0.8.3;

contract MyContract{
    string public name = "Rujuta Medhi";
}
```



DEPLOY & RUN TRANSACTIONS

HELLOWORLD AT 0XAE0...968
Balance: 0 ETH
greet

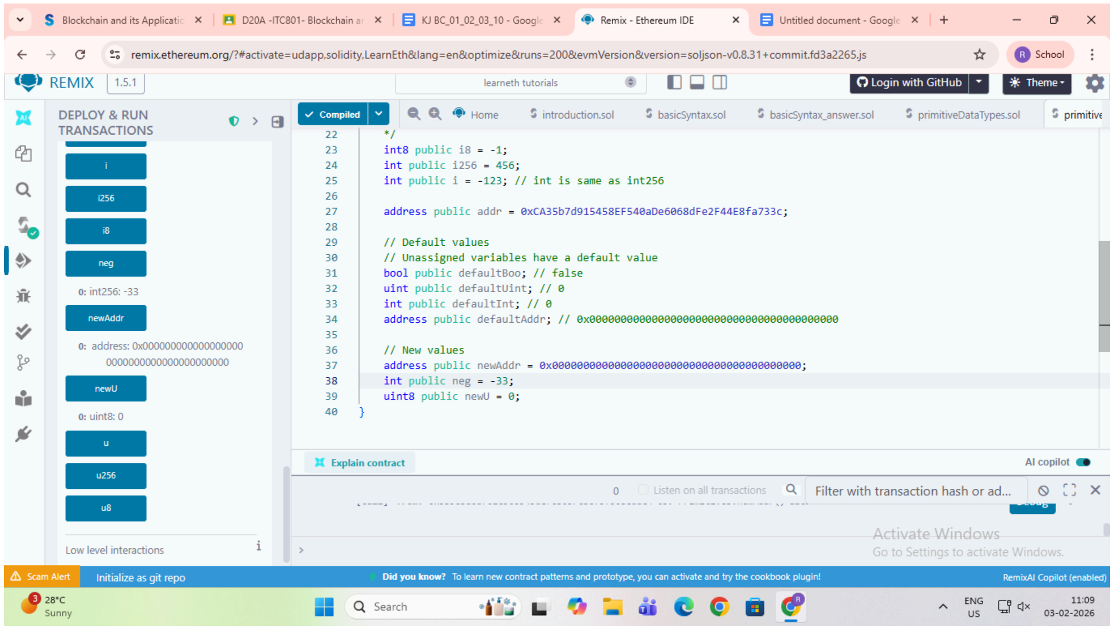
Low level interactions
CALLDATA
Transact

MYCONTRACT AT 0X9D8...A56
Balance: 0 ETH
name
0: string: Rujuta Medhi

Low level interactions
CALLDATA
Transact

```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: HelloWorld.greet() data: 0xcfa...e3217
call to MyContract.name

[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MyContract.name() data: 0x06f...dde03
from            0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to              MyContract.name() 0x9d83e140330758a8fFD07F8Bd73e86ebcA8a5692
execution cost  3298 gas (Cost only applies when called by a contract)
input           0x06f...dde03
output          0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000
                000000000000000c52756a757461204d65646869000000000000000000000000000000000000000000
decoded input   {}
decoded output  {
                    "0": "string: Rujuta Medhi"
                }
logs            []
raw logs        []
```

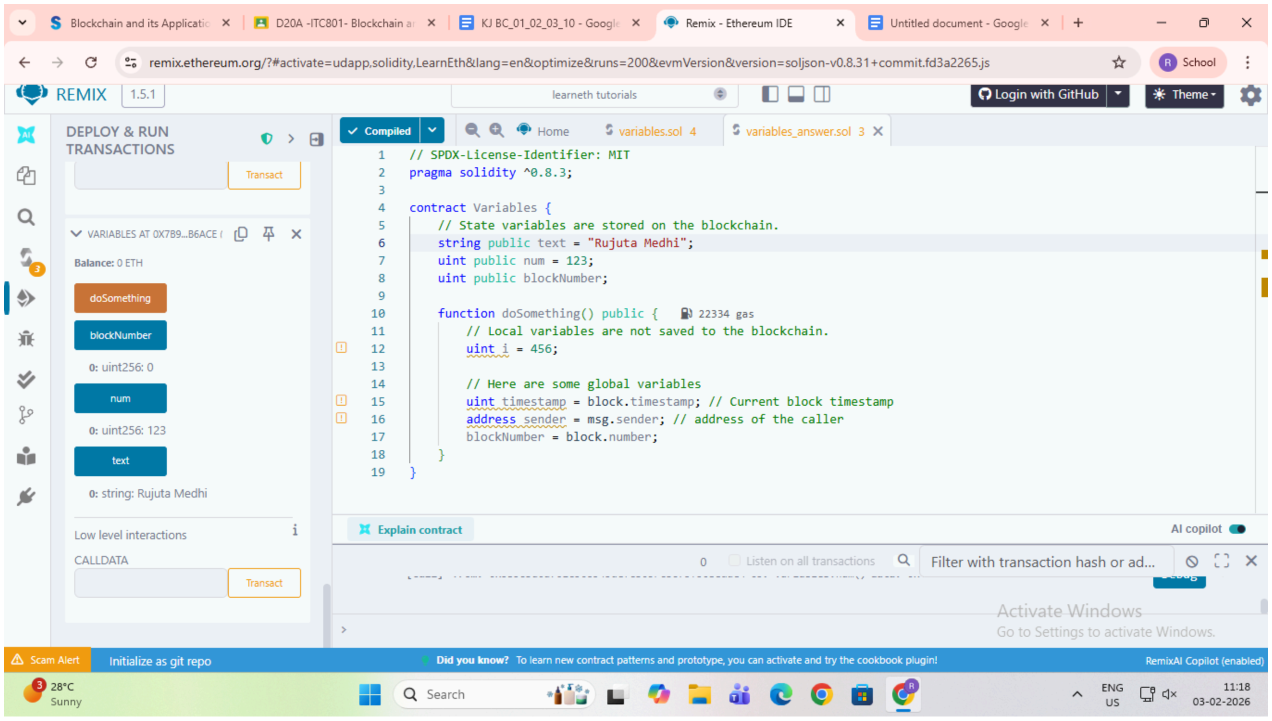
3. Primitive Data Types



```
    */
    int8 public i8 = -1;
    int public i256 = 456;
    int public i = -123; // int is same as int256

    address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

    // Default values
    // Unassigned variables have a default value
    bool public defaultBoo; // false
    uint public defaultUint; // 0
    int public defaultInt; // 0
    address public defaultAddr; // 0x0000000000000000000000000000000000000000

    // New values
    address public newAddr = 0x0000000000000000000000000000000000000000;
    int public neg = -33;
    uint8 public newU = 0;
}
```

```
0: int256: -33
0: address: 0x0000000000000000000000000000000000000000
0: uint8: 0
```

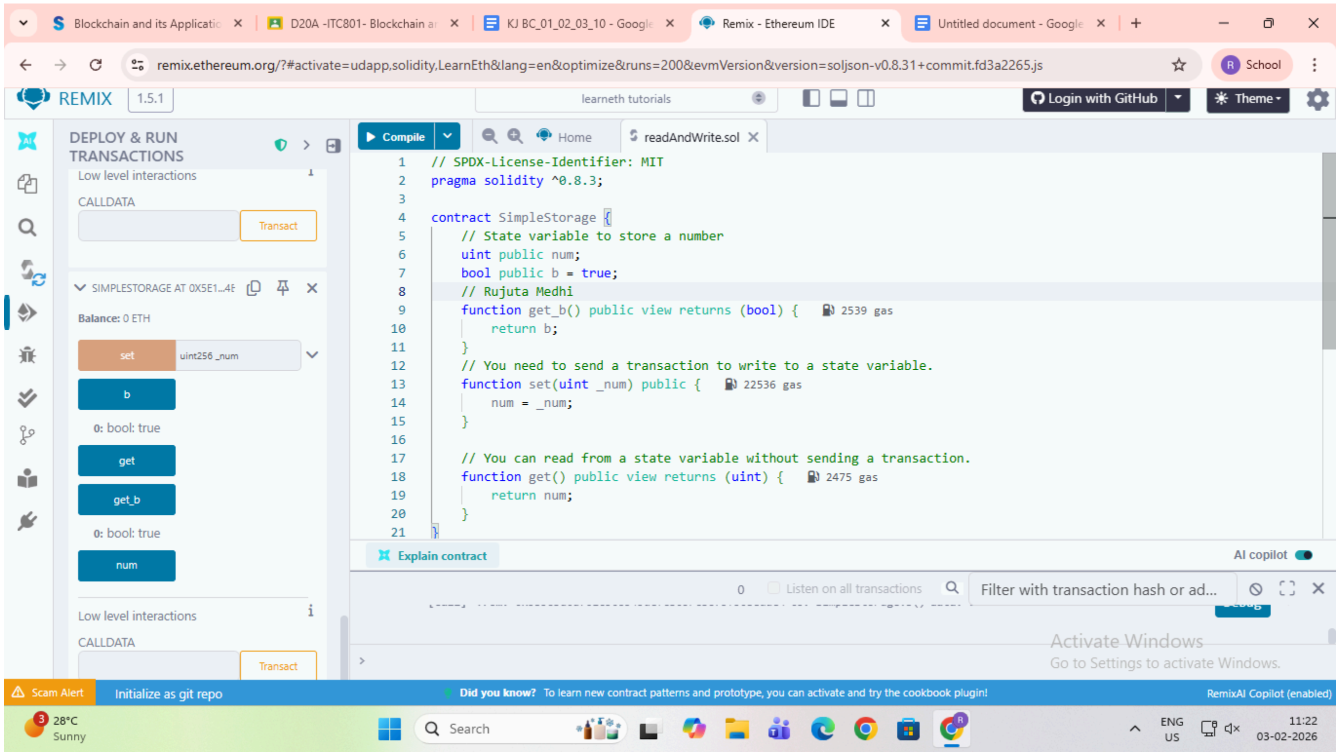
4. Variables



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

contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Rujuta Medhi";
    uint public num = 123;
    uint public blockNumber;

    function doSomething() public {
        // Local variables are not saved to the blockchain.
        uint i = 456;

        // Here are some global variables
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
        blockNumber = block.number;
    }
}
```

```
Balance: 0 ETH
blockNumber
0: uint256: 0
num
0: uint256: 123
text
0: string: Rujuta Medhi
```

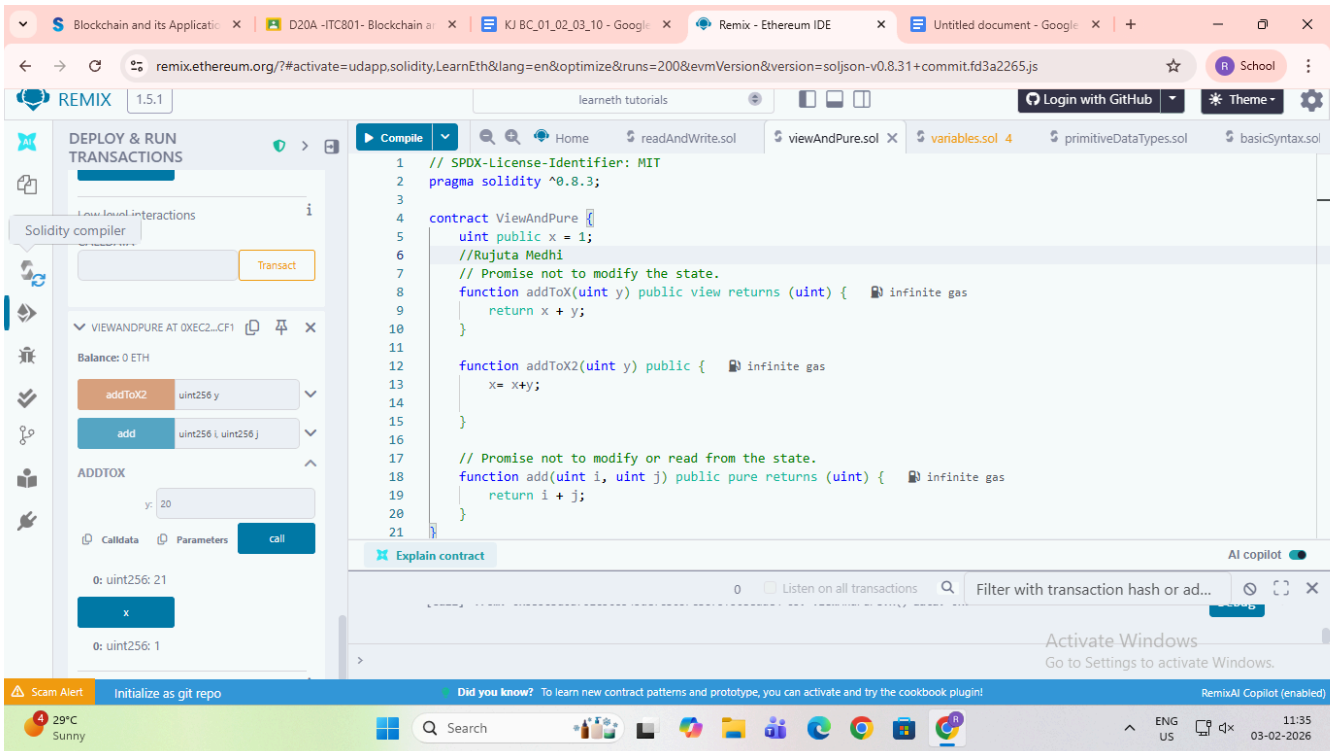
5. Functions - Reading and Writing to a State Variable


DEPLOY & RUN TRANSACTIONS
Low level interactions
CALLDATA
Transact
SIMPLESTORAGE AT 0X5E1...4E
Balance: 0 ETH
set
uint256 _num
b
get
get_b
num
Compile
Home
readAndWrite.sol
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
Scam 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)

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

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

```
0: bool: true
0: bool: true
```

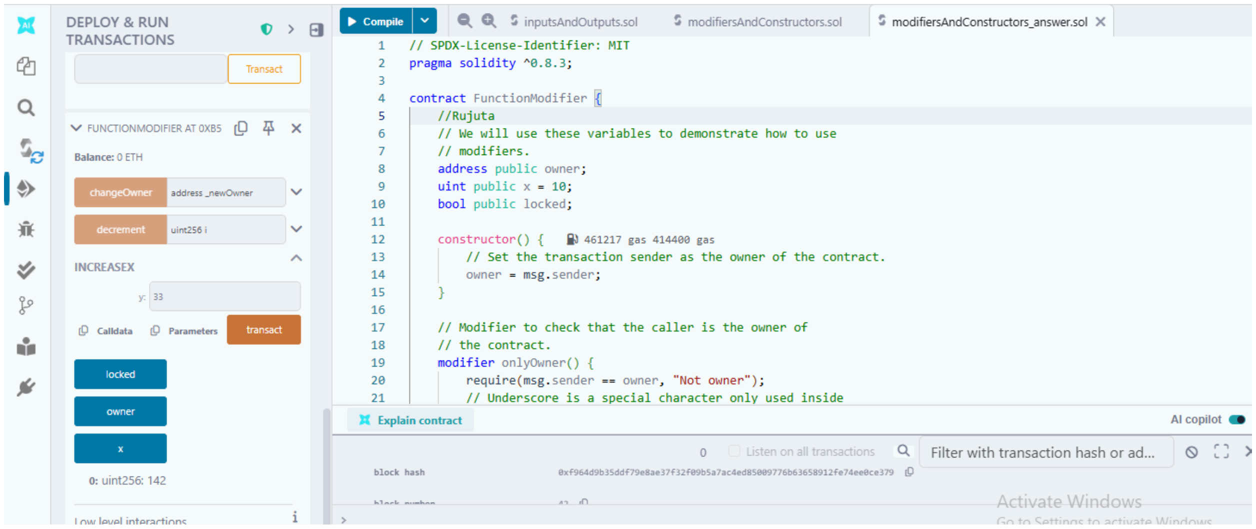
6. Functions - View and Pure



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

contract ViewAndPure {
    uint public x = 1;
    //Rujuta Medhi
    // Promise not to modify the state.
    function addToX(uint y) public view returns (uint) {   infinite gas
        return x + y;
    }

    function addToX2(uint y) public {   infinite gas
        x= x+y;

    }

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

```
0: uint256: 21
0: uint256: 1
```

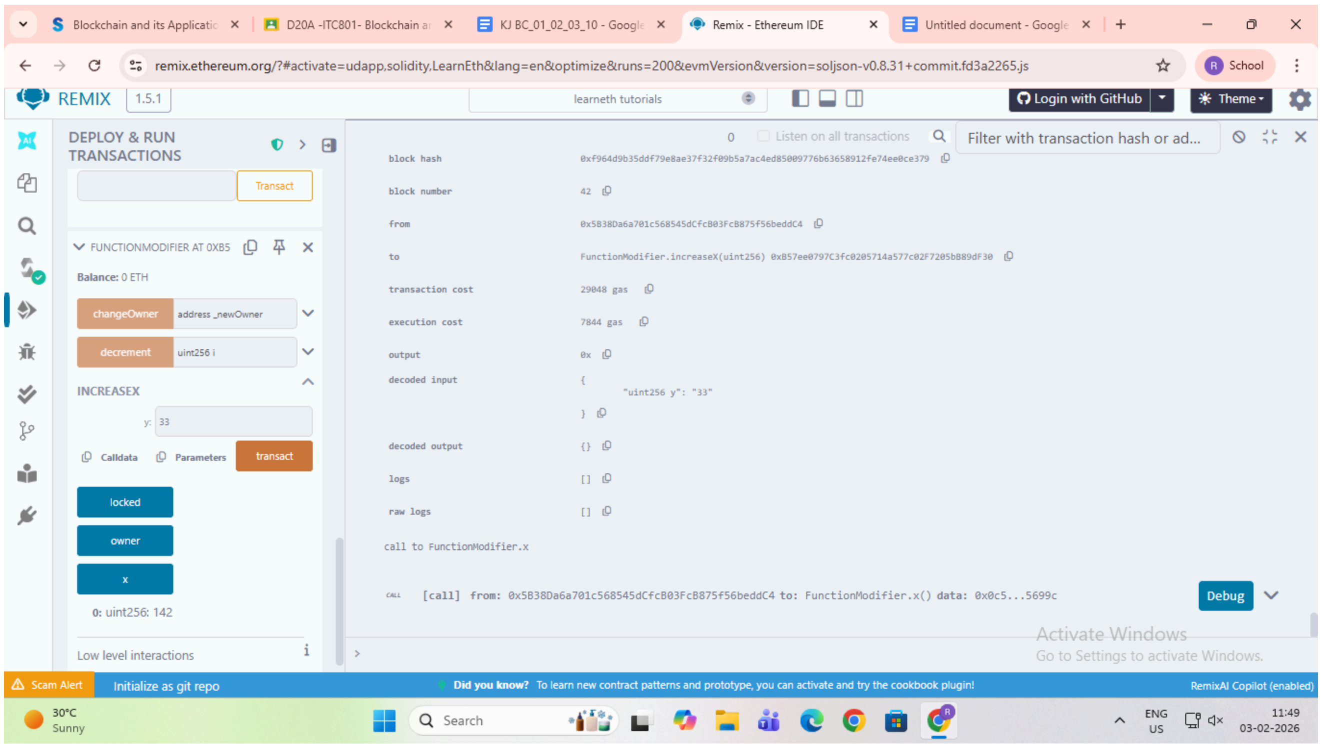
7. Functions - Modifiers and Constructors



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

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

    constructor() {    461217 gas 414400 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
```

```
0: uint256: 142
```

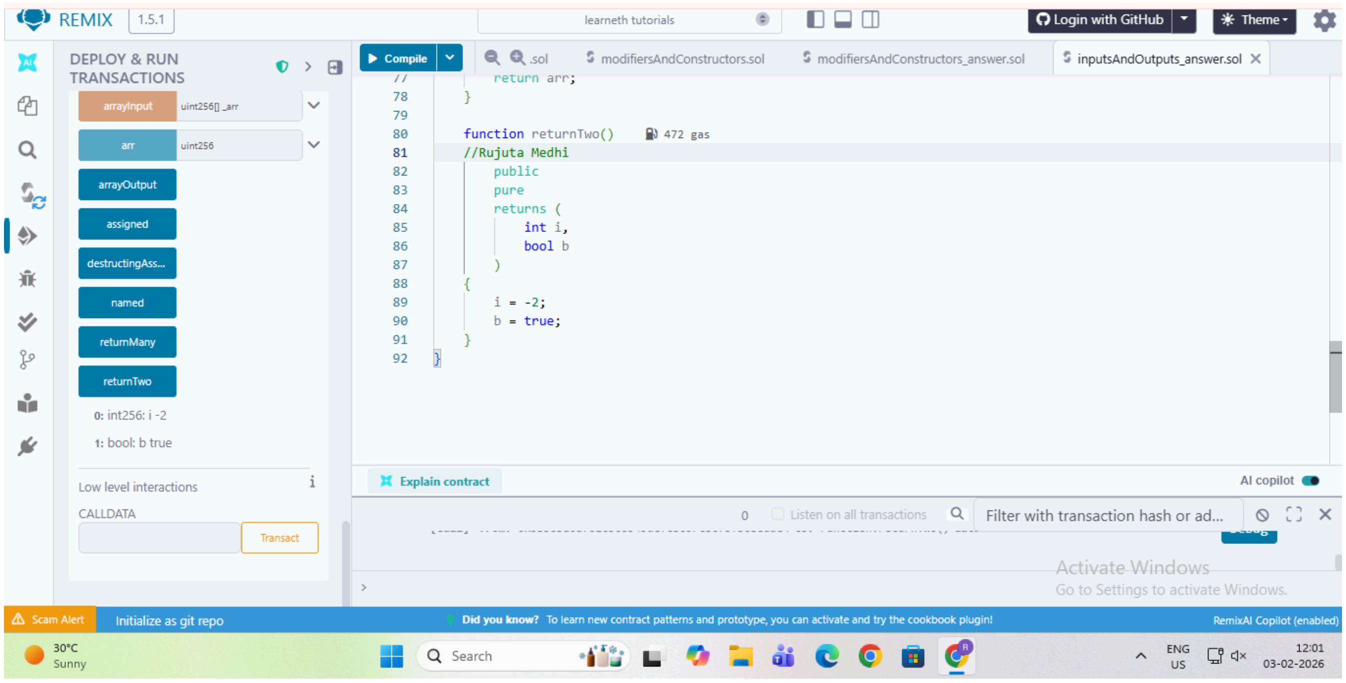
```
block hash          0xf964d9b35ddf79e8ae37f32f09b5a7ac4ed85009776b63658912fe74ee0ce379
```


Blockchain and its Applicatic
D20A -ITC801- Blockchain ar
KJ BC_01_02_03_10 - Google
Remix - Ethereum IDE
Untitled document - Google
remix.ethereum.org/?#activate=udapp,solidity,LearnEth&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
School
REMIX
1.5.1
learneth tutorials
Login with GitHub
Theme
DEPLOY & RUN TRANSACTIONS
Transact
FUNCTIONMODIFIER AT 0XB5
Balance: 0 ETH
changeOwner
address _newOwner
decrement
uint256 i
INCREASEX
y: 33
Calldata
Parameters
transact
locked
owner
x
Low level interactions
Listen on all transactions
Filter with transaction hash or ad...
Debug
Scam 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)
30°C
Sunny
Search
ENG
US
11:49
03-02-2026

```
0: uint256: 142
```

```
block hash          0xf964d9b35ddf79e8ae37f32f09b5a7ac4ed85009776b63658912fe74ee0ce379
block number        42
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  FunctionModifier.increaseX(uint256) 0xB57ee0797C3fc0205714a577c02F7205bB89dF30
transaction cost    29048 gas
execution cost      7844 gas
output              0x
decoded input       {
                        "uint256 y": "33"
                    }
decoded output      {}
logs                []
raw logs            []

call to FunctionModifier.x

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: FunctionModifier.x() data: 0x0c5...5699c
```

8. Functions - Inputs and Outputs

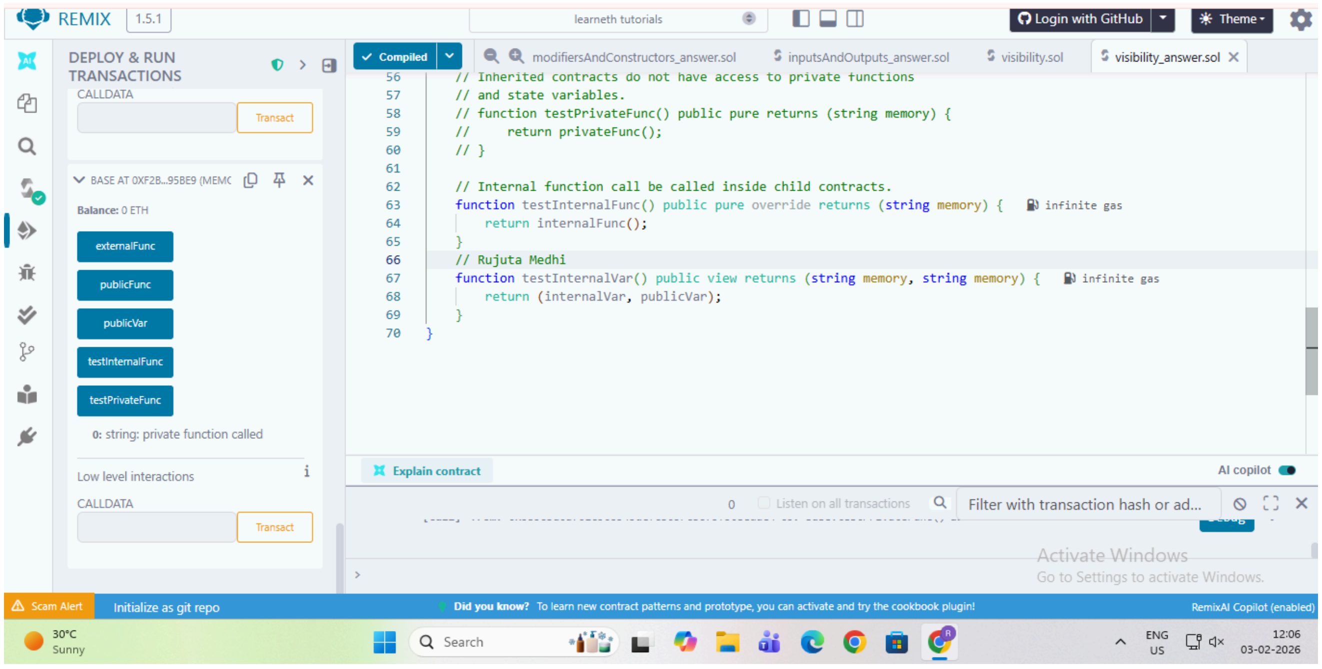


```
        return arr;
    }

    function returnTwo()      472 gas
    //Rujuta Medhi
        public
        pure
        returns (
            int i,
            bool b
        )
    {
        i = -2;
        b = true;
    }
}
```

```
0: int256: i -2
1: bool: b true
```

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

9. Visibility

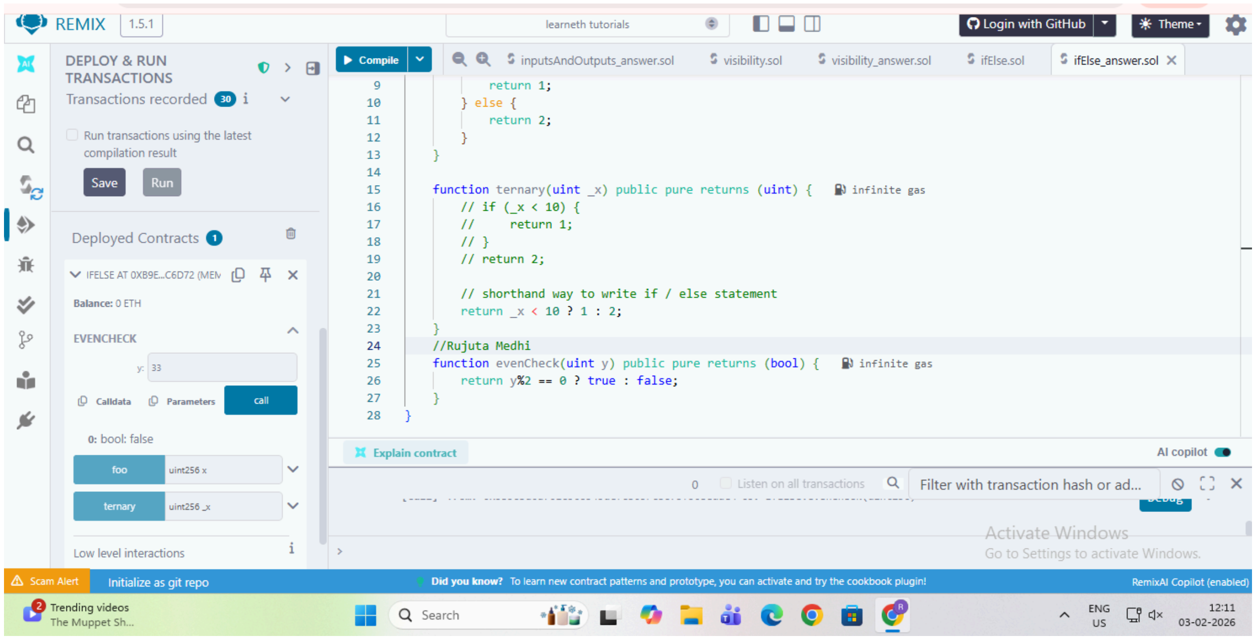


```
    // Inherited contracts do not have access to private functions
    // and state variables.
    // function testPrivateFunc() public pure returns (string memory) {
    //     return privateFunc();
    // }

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

```
0: string: private function called
```

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

10. Control Flow - If/Else



DEPLOY & RUN TRANSACTIONS

Transactions recorded 30

Deployed Contracts 1

IFELSE AT 0XB9E...C6D72 (MEM

Balance: 0 ETH

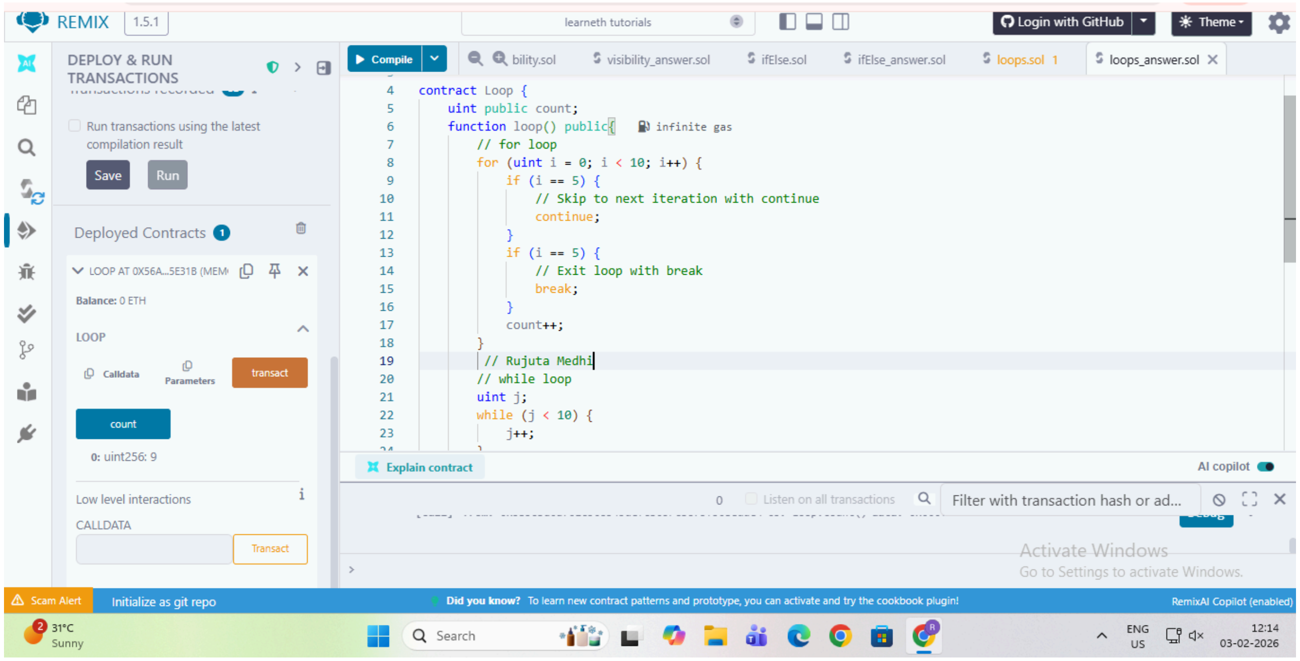
EVENCHECK

y: 33

0: bool: false

```
        return 1;
    } else {
        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;
}
//Rujuta Medhi
function evenCheck(uint y) public pure returns (bool) {
    return y%2 == 0 ? true : false;
}
}
```

11. Control Flow - Loops



DEPLOY & RUN TRANSACTIONS

Deployed Contracts 1

LOOP AT 0X56A...5E31B (MEM

Balance: 0 ETH

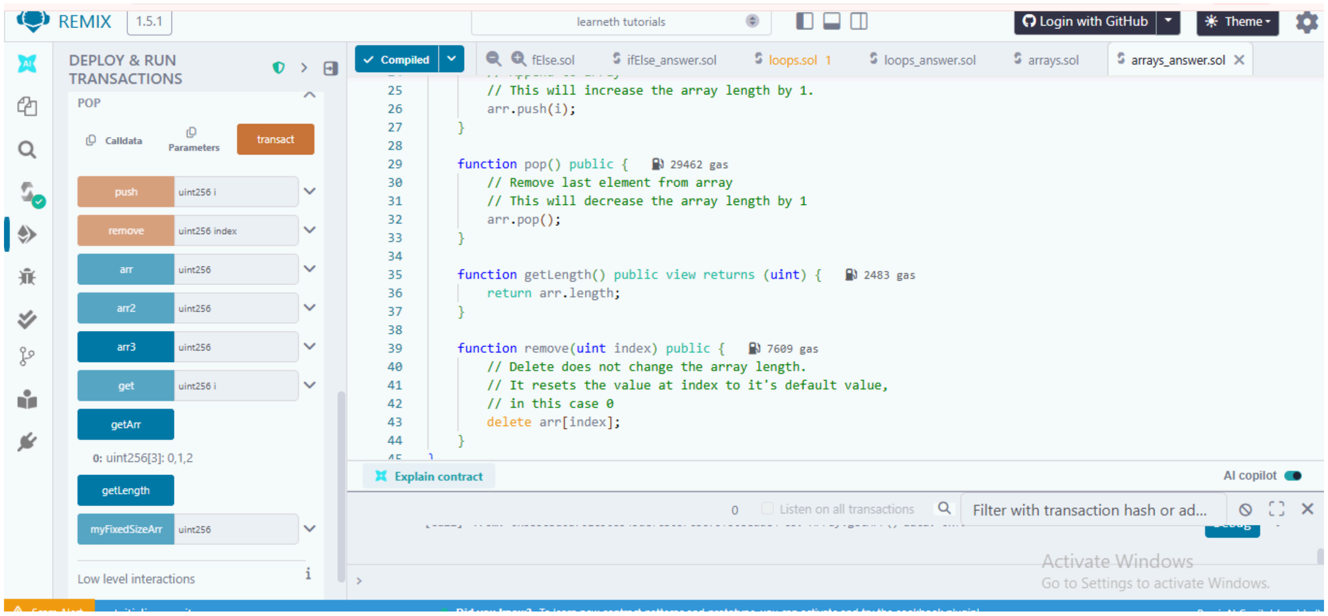
LOOP

count

0: uint256: 9

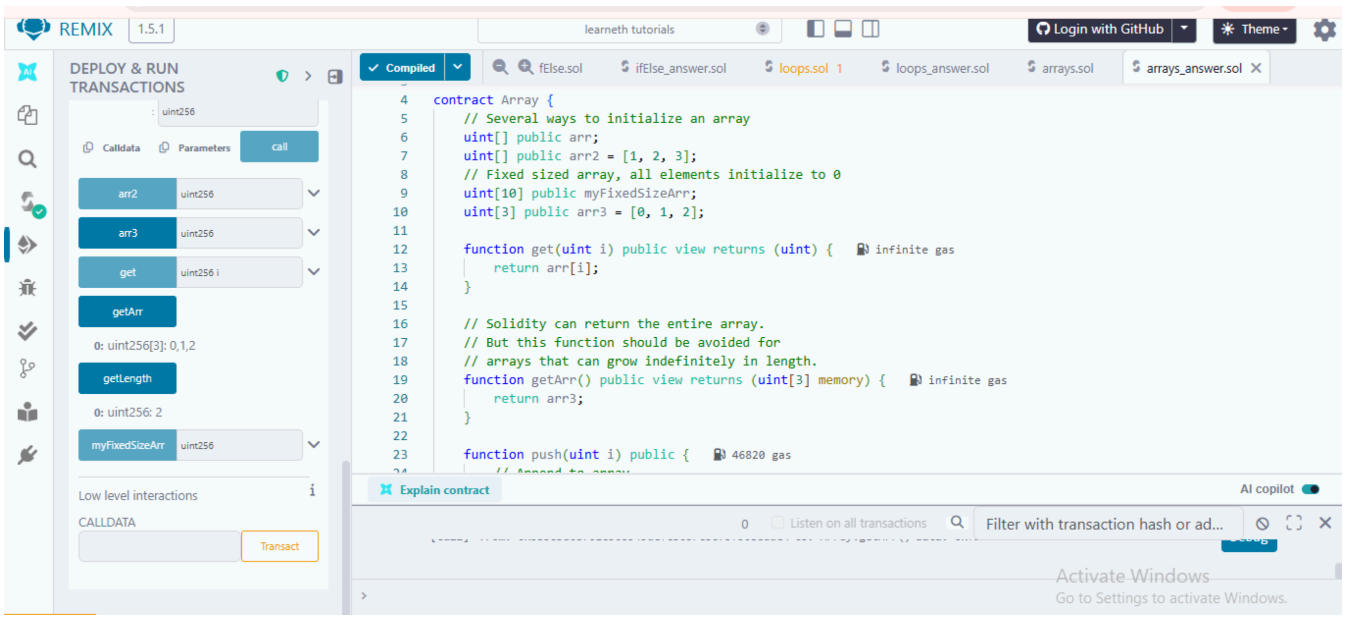
```
contract Loop {
    uint public count;
    function loop() public{
        // for loop
        for (uint i = 0; i < 10; i++) {
            if (i == 5) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
            count++;
        }
        // Rujuta Medhi
        // while loop
        uint j;
        while (j < 10) {
            j++;
```

12. Data Structures - Arrays



0: uint256[3]: 0,1,2

```
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {    29462 gas
        // Remove last element from array
        // This will decrease the array length by 1
        arr.pop();
    }

    function getLength() public view returns (uint) {    2483 gas
        return arr.length;
    }

    function remove(uint index) public {    7609 gas
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case 0
        delete arr[index];
    }
```

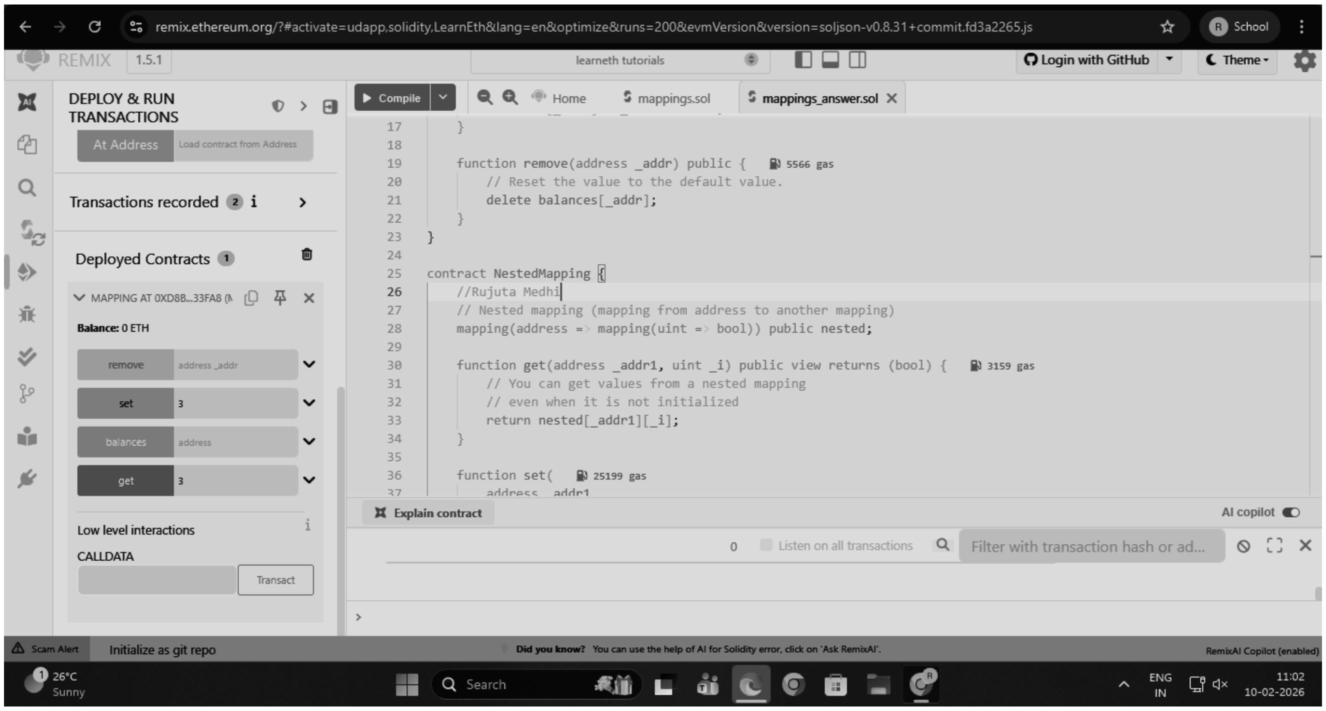


0: uint256[3]: 0,1,2

0: uint256: 2

```
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 gas
        return arr3;
    }

    function push(uint i) public {    46820 gas
```

13. Data Structures - Mappings



```
remix.ethereum.org/?#activate=udapp,solidity,LearnEth&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
```

DEPLOY & RUN TRANSACTIONS

Transactions recorded 2

Deployed Contracts 1

MAPPING AT 0XD8B...33FA8

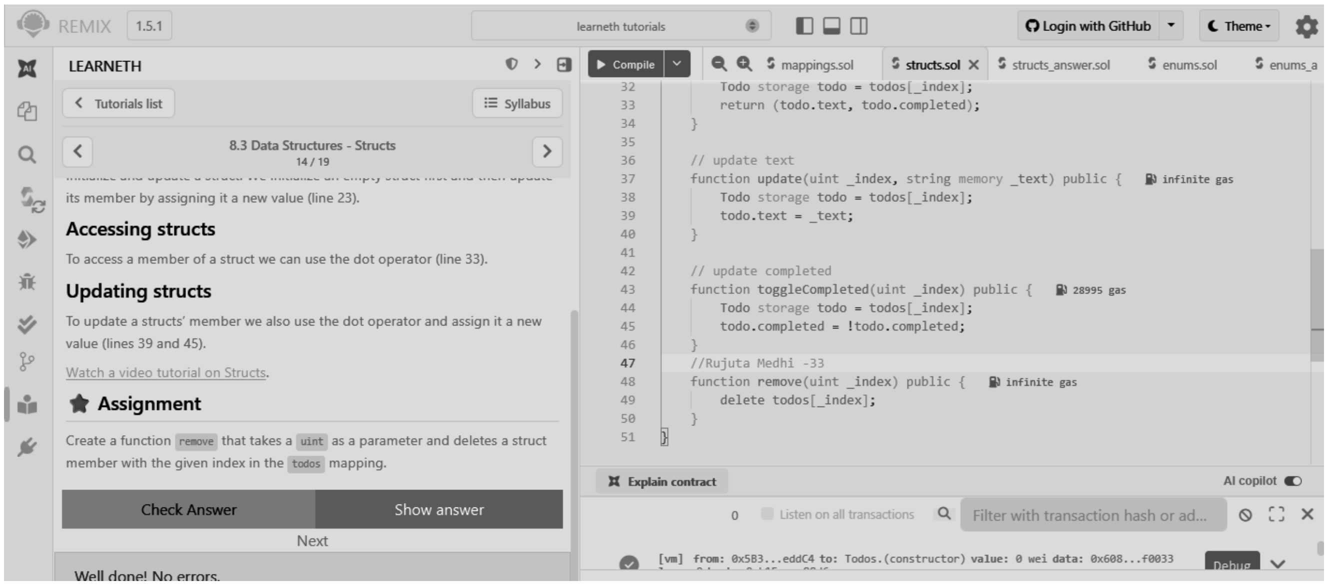
Balance: 0 ETH

remove	address _addr
set	3
balances	address
get	3

```
    }

    function remove(address _addr) public {    5566 gas
        // Reset the value to the default value.
        delete balances[_addr];
    }
}

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

    function get(address _addr1, uint _i) public view returns (bool) {    3159 gas
        // You can get values from a nested mapping
        // even when it is not initialized
        return nested[_addr1][_i];
    }

    function set(    25199 gas
        address _addr1
```

Compile
Home
mappings.sol
mappings_answer.sol
At Address
Load contract from Address
Low level interactions
CALLDATA
Transact
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
Scam Alert
Initialize as git repo
Did you know? You can use the help of AI for Solidity error, click on 'Ask RemixAI'.
RemixAI Copilot (enabled)
26°C Sunny
Search
ENG IN
11:02
10-02-2026

14. Data Structures - Structs



LEARNETH

8.3 Data Structures - Structs 14 / 19

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

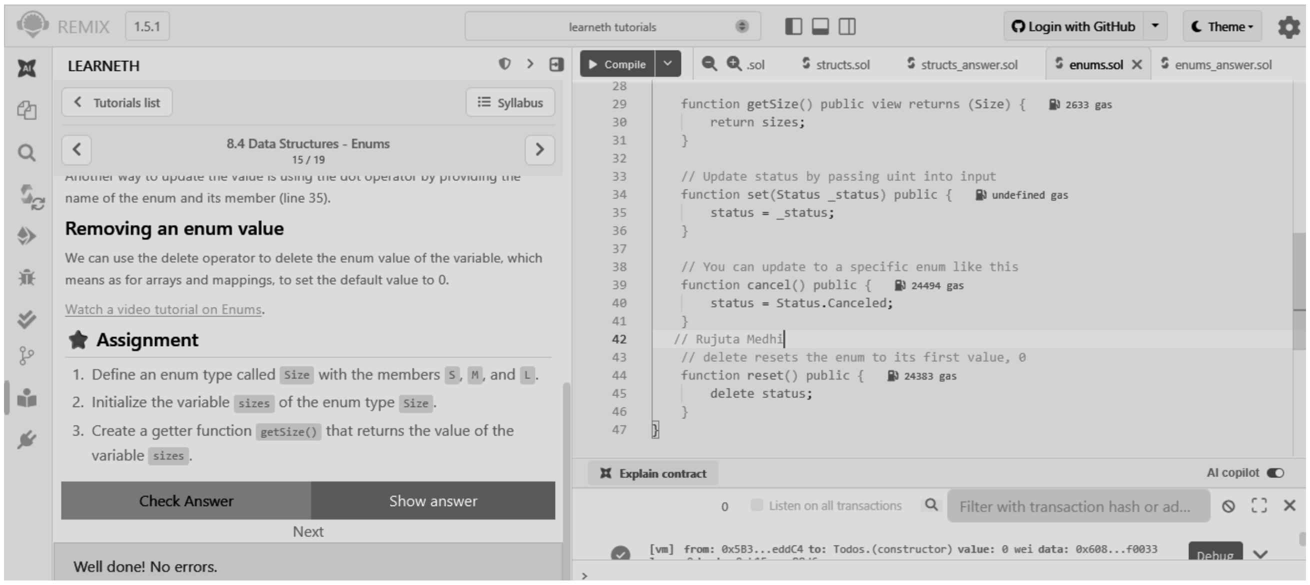
Create a function `remove` that takes a `uint` as a parameter and deletes a struct member with the given index in the `todos` mapping.

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
42        // update completed
43        function toggleCompleted(uint _index) public {    28995 gas
44            Todo storage todo = todos[_index];
45            todo.completed = !todo.completed;
46        }
47        //Rujuta Medhi -33
48        function remove(uint _index) public {    infinite gas
49            delete todos[_index];
50        }
51    }
```

```
[vm] from: 0x5B3...eddC4 to: Todos.(constructor) value: 0 wei data: 0x608...f0033
```

15. Data Structures - Enums



8.4 Data Structures - Enums
15 / 19

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`.

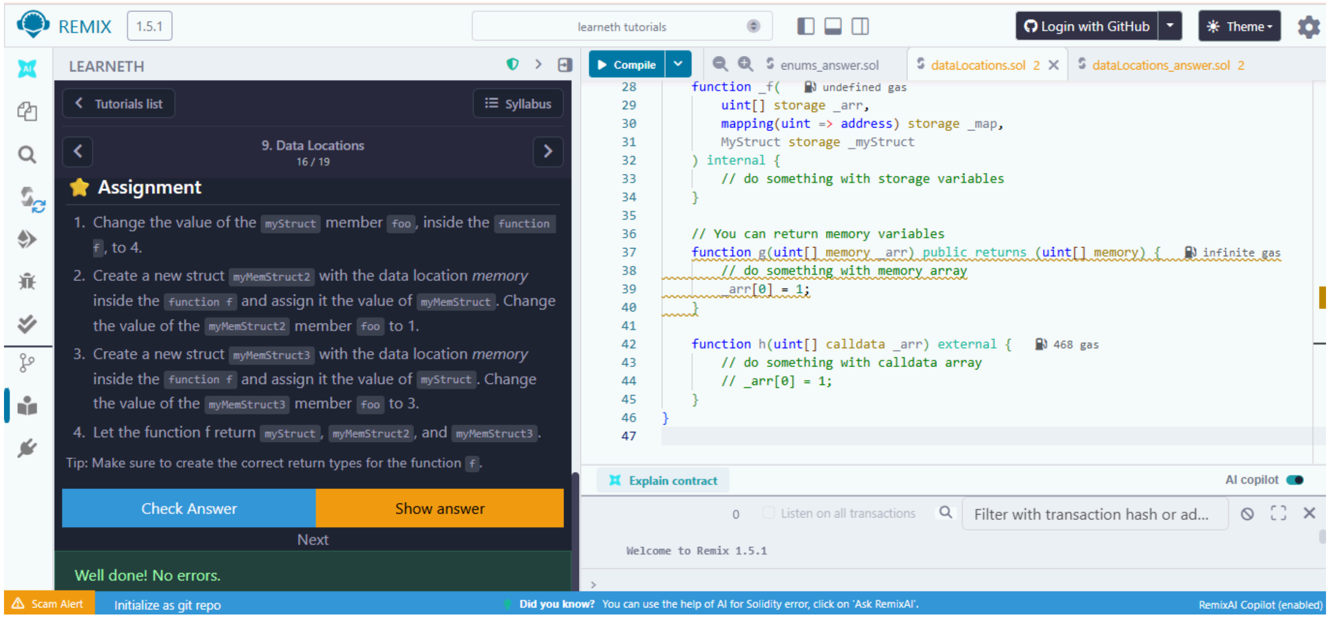
Well done! No errors.

```
    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;
    }

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

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

```
[vm] from: 0x5B3...eddC4 to: Todos.(constructor) value: 0 wei data: 0x608...f0033
```

16. Data Locations



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.

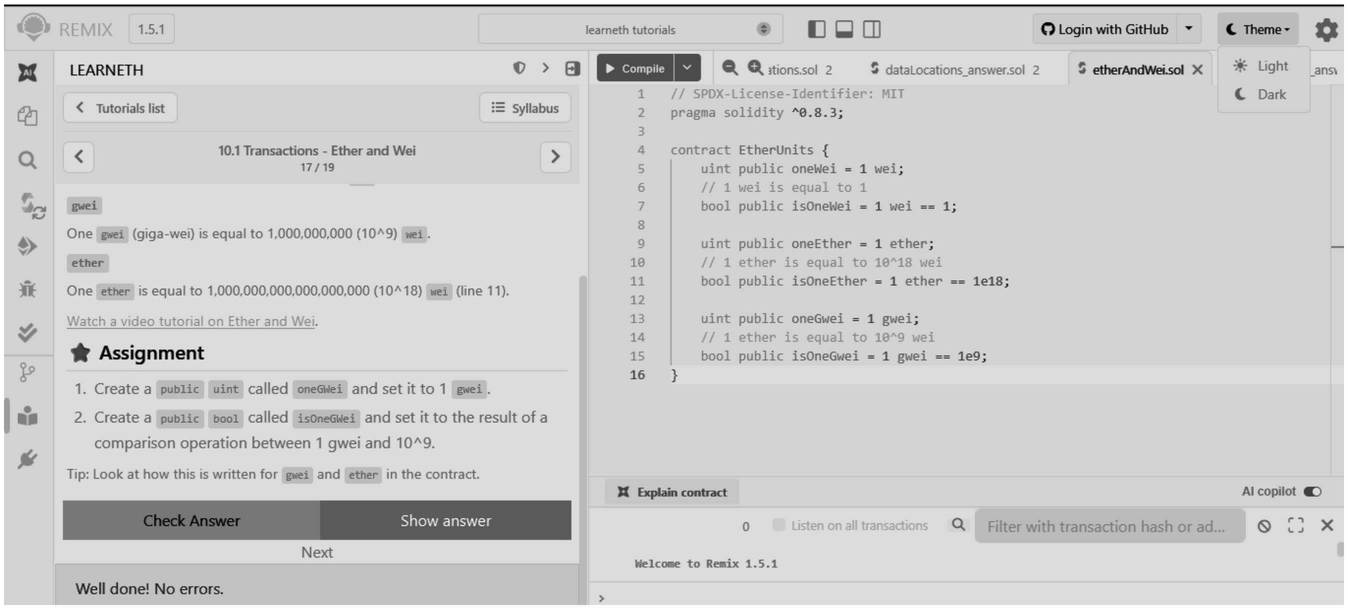
```
    function _f(    undefined gas
        uint[] storage _arr,
        mapping(uint => address) storage _map,
        MyStruct storage _myStruct
    ) internal {
        // do something with storage variables
    }

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

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

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

```
Welcome to Remix 1.5.1
```

Scam Alert
Initialize as git repo
Did you know? You can use the help of AI for Solidity error, click on 'Ask RemixAI'.
RemixAI Copilot (enabled)

17. Transactions - Ether and Wei



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

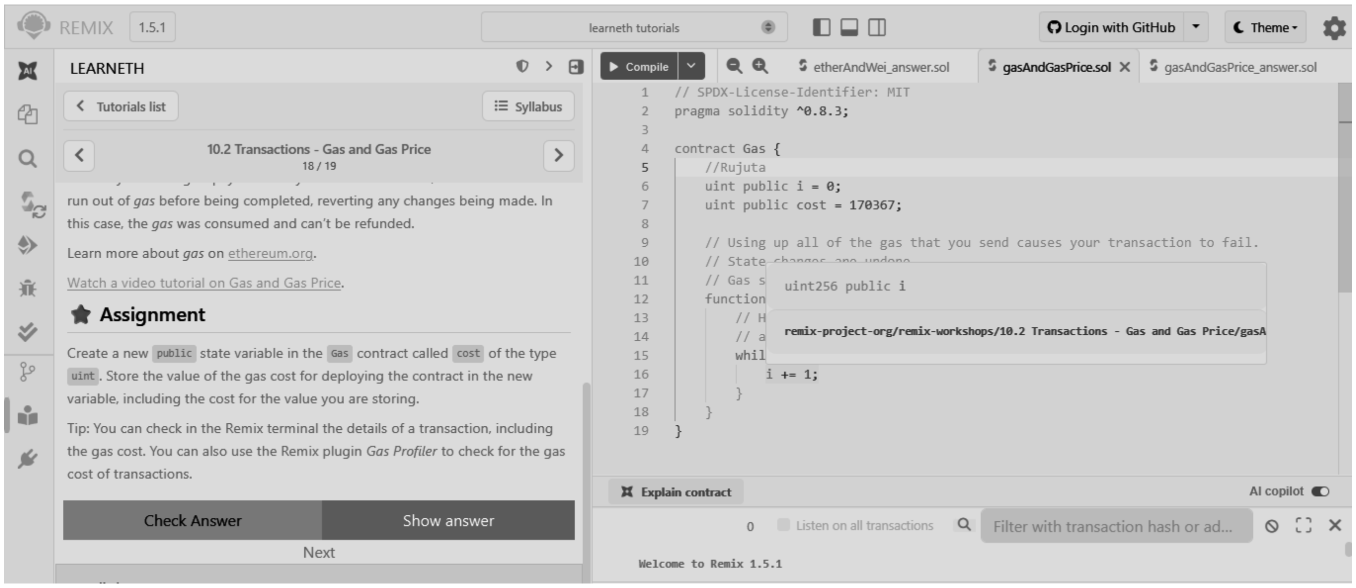
Well done! No errors.

```
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
13      uint public oneGwei = 1 gwei;
14      // 1 ether is equal to 10^9 wei
15      bool public isOneGwei = 1 gwei == 1e9;
16  }
```

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

```
Welcome to Remix 1.5.1
```

18. Transactions - Gas and Gas Price



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.

Check Answer | Show answer

Next

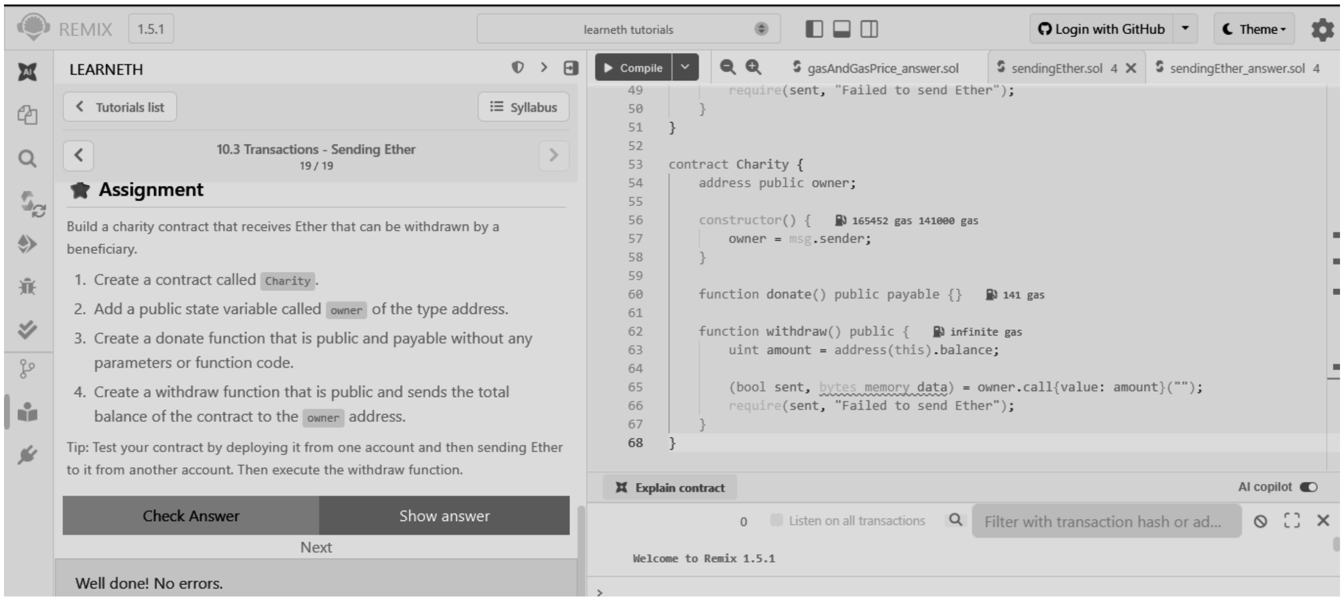
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Gas {
    //Rujuta
    uint public i = 0;
    uint public cost = 170367;

    // Using up all of the gas that you send causes your transaction to fail.
    // State
    // Gas s
    function
        // H
        // a
        whil
            i += 1;
        }
    }
}
```

```
uint256 public i

remix-project-org/remix-workshops/10.2 Transactions - Gas and Gas Price/gasA
```

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

```
Welcome to Remix 1.5.1
```

19. Transactions - Sending Ether



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");
    }
}

contract Charity {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function donate() public payable {}

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

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

```
Welcome to Remix 1.5.1
```

Conclusion: This experiment provided practical exposure to Solidity programming and the process of creating smart contracts on a blockchain platform. Through hands-on implementation, the fundamental concepts of blockchain such as decentralization, immutability, and trustless execution were clearly understood. Designing and deploying smart contracts helped in learning Solidity syntax, data types, functions, modifiers, and transaction handling.