

Blockchain Experiment 4

AIM: Hands on Solidity Programming Assignments for creating Smart Contracts

THEORY:

Q1: 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
- **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.

Q2: 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.

Q3: Visibility, Modifiers and Constructors

Function Visibility defines who can access a function:

- **public:** available both inside and outside the contract.
- **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 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.

Q4: 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.

Q5: 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. Example: 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 }.

Q6: 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.

Q7: 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.

TASKS PERFORMED:

Tutorial 1: Introduction

a. get

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active. Under 'CONTRACT', 'Counter - remix-project-org/remix-w' is selected. The 'Deploy' button is orange, and 'At Address' is blue. Below, 'Transactions recorded' shows 7 transactions. 'Deployed Contracts' shows 'COUNTER AT 0x5E1...4EFF5 (M)' with a balance of 0 ETH. The 'get' button is highlighted in blue. The 'CALLDATA' field is empty, and the 'Transact' button is orange. The main editor shows the Solidity code for the 'Counter' contract. The 'get' function is defined as follows:

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

The 'Explain contract' panel at the bottom shows a transaction log entry: '[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c'.

b. inc

The screenshot shows the Remix IDE interface after the 'inc' function has been called. The 'get' button is no longer highlighted. The 'CALLDATA' field now contains '0x00'. The 'Transact' button is orange. The main editor shows the same Solidity code. The 'Explain contract' panel at the bottom shows a transaction log entry: '[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c'.

c. dec

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing a contract named 'Counter' from 'remix-project-org/remix-w'. Below this, there are buttons for 'Deploy', 'At Address', and 'Transactions recorded'. The 'Deployed Contracts' section shows a contract at address '0x5E1...4EFF5' with a balance of 0 ETH. The 'Dec' button is highlighted. The main editor shows the Solidity code for the 'Counter' contract, which includes a 'dec' function to decrement the count by 1. The bottom section shows a transaction log with a call to 'Counter.get()'.

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

Tutorial 2: Basic Syntax

The screenshot shows the Remix IDE interface with the 'LEARNETH' sidebar. The 'Tutorials list' shows '2. Basic Syntax' as the current tutorial. The main editor shows the Solidity code for 'MyContract', which includes a 'name' variable of type 'string'. The bottom section shows a transaction log with a call to 'Counter.get()'.

```
1 // SPDX-License-Identifier: MIT
2 // compiler version must be greater than or equal to 0.8.3 and less than 0.9.0
3 pragma solidity ^0.8.3;
4
5 contract MyContract {
6     string public name = "Alice";
7 }
```

Tutorial 3: Primitive Data Types

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

LEARNETH 1.5.1 learneth tutorials

✓ Compiled primitiveDataTypes.sol

20 Negative numbers are allowed for int types.
21 Like uint, different ranges are available from int8 to int256
22 */
23 int8 public i8 = -1;
24 int public i256 = 456;
25 int public i = -123; // int is same as int256
26
27
28 address public addr = 0xCA35b7d915458F540aDe6068dFe2F44E8fa733c;
29
30 // Default values
31 // Unassigned variables have a default value
32 bool public defaultBool; // false
33 uint public defaultUint; // 0
34 int public defaultInt; // 0
35 address public defaultAddr; // 0x00
36
37 // New values
38 address public newAddr = 0x00;
39 int public neg = -12;
40 uint8 public newU = 0;
41 }

4. Variables 3 / 19

You can learn more about these data types as well as *Fixed Point Numbers, Byte Arrays, Strings*, and more in the *Solidity* documentation.

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

Watch a video tutorial on [Primitive Data Types](#).

★ **Assignment**

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

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

Check Answer Show answer

Next

Well done! No errors.

Scan Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Explain contract

0 Listen on all transactions

CALL [call] from: 0x5B380a6a701c568545dcfcb03fC8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Tutorial 4: Variables

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

LEARNETH 1.5.1 learneth tutorials

Compile primitiveDataTypes.sol variables.sol 3

```
1 // SPDX-License-Identifier: MIT  
2 pragma solidity ^0.8.3;  
3  
4 contract Variables {  
5     // State variables are stored on the blockchain.  
6     string public text = "Hello";  
7     uint public num = 123;  
8     uint public blockNumber;  
9  
10  
11     function doSomething() public { 22334 gas  
12         // Local variables are not saved to the blockchain.  
13         uint i = 456;  
14  
15         // Here are some global variables  
16         uint timestamp = block.timestamp; // Current block timestamp  
17         address sender = msg.sender; // address of the caller  
18         blockNumber = block.number;  
19     }  
20 }
```

4. Variables 4 / 19

addresses, contracts, and transactions.

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.

Check Answer Show answer

Next

Well done! No errors.

Scan Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Explain contract

0 Listen on all transactions

CALL [call] from: 0x5B380a6a701c568545dcfcb03fC8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Tutorial 5: Functions - Reading and Writing to a State Variable

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

REMX 1.5.1 learneth tutorials

LEARNETH

Tutorials list Syllabus

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

names. A common convention is to use an underscore as a prefix for the parameter name to distinguish them from state variables.

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

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

[Watch a video tutorial on Functions.](#)

★ **Assignment**

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

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

SimpleStorage.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract SimpleStorage {
5     // State variable to store a number
6     uint public num;
7     bool public b = true;
8
9     // You need to send a transaction to write to a state variable.
10    function set(uint _num) public { 22536 gas
11        num = _num;
12    }
13
14    // You can read from a state variable without sending a transaction.
15    function get() public view returns (uint) { 2475 gas
16        return num;
17    }
18
19    function get_b() public view returns (bool) { 2530 gas
20        return b;
21    }
22 }
```

Explain contract

0 Listen on all transactions

CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 6: Functions - View and Pure

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

REMX 1.5.1 learneth tutorials

LEARNETH

Tutorials list Syllabus

5.2 Functions - View and Pure 6 / 19

opcodes."

From the [Solidity documentation](#).

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

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

[Watch a video tutorial on View and Pure Functions.](#)

★ **Assignment**

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

[Check Answer](#) [Show answer](#)

Next

Well done! No errors.

ViewAndPure.sol

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

Explain contract

0 Listen on all transactions

CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 7: Functions - Modifiers and Constructors

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

5.3 Functions - Modifiers and Constructors

parameters and is especially useful when you don't know certain initialization values before the deployment of the contract.

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.

```
43
44     x = x + y;
45 }
46
47 function increaseX(uint y) public onlyOwner biggerThan0(y) increaseXbyY(y){
48 }
49
50 // Modifiers can be called before and / or after a function.
51 // This modifier prevents a function from being called while
52 // it is still executing.
53 modifier noReentrancy() {
54     require(!locked, "No reentrancy");
55
56     locked = true;
57     _;
58     locked = false;
59 }
60
61 function decrement(uint i) public noReentrancy {
62     x -= i;
63
64     if (i > 1) {
65         decrement(i - 1);
66     }
67 }
68 }
```

Explain contract

0 Listen on all transactions

CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 8: Functions - Inputs and Outputs

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

5.4 Functions - Inputs and Outputs

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.

Check Answer Show answer

Next

Well done! No errors.

```
65     return (i, b, j, x, y);
66 }
67
68 // Cannot use map for neither input nor output
69
70 // Can use array for input
71 function arrayInput(uint[] memory _arr) public {
72 }
73
74 // Can use array for output
75 uint[] public arr;
76
77 function arrayOutput() public view returns (uint[] memory) {
78     return arr;
79 }
80
81 function returnTwo()
82     public
83     pure
84     returns (
85         int i,
86         bool b
87     )
88 {
89     i = -2;
90     b = true;
91 }
```

Explain contract

0 Listen on all transactions

CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 9: Visibility

The screenshot shows the Remix IDE interface for Tutorial 9: Visibility. The sidebar on the left displays the 'LEARNETH' tutorial list, with '6. Visibility' selected. The main editor shows the following Solidity code:

```
43 // function testExternalFunc() public pure returns (string memory) {
44 //     return externalFunc();
45 // }
46
47 // State variables
48 string private privateVar = "my private variable";
49 string internal internalVar = "my internal variable";
50 string public publicVar = "my public variable";
51 // State variables cannot be external so this code won't compile.
52 // string external externalVar = "my external variable";
53
54
55 contract Child is Base {
56     // Inherited contracts do not have access to private functions
57     // and state variables.
58     // function testPrivateFunc() public pure returns (string memory) {
59     //     return privateFunc();
60     // }
61
62     // Internal function call be called inside child contracts.
63     function testInternalFunc() public pure override returns (string memory) {
64         return internalFunc();
65     }
66
67     function testInternalVar() public view returns (string memory, string memory) {
68         return (internalVar, publicVar);
69     }
70 }
```

The console at the bottom shows a transaction call: `[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddc4 to: Counter.get() data: 0x6d4...ce63c`.

Tutorial 10: Control Flow - If/Else

The screenshot shows the Remix IDE interface for Tutorial 10: Control Flow - If/Else. The sidebar on the left displays the 'LEARNETH' tutorial list, with '7.1 Control Flow - If/Else' selected. The main editor shows the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract IfElse {
5     function foo(uint x) public pure returns (uint) {
6         if (x < 10) {
7             return 0;
8         } else if (x < 20) {
9             return 1;
10        } else {
11            return 2;
12        }
13    }
14
15    function ternary(uint x) public pure returns (uint) {
16        // if (x < 10) {
17        //     return 1;
18        // }
19        // return 2;
20
21        // shorthand way to write if / else statement
22        return x < 10 ? 1 : 2;
23    }
24
25    function evenCheck(uint y) public pure returns (bool) {
26        return y % 2 == 0 ? true : false;
27    }
28 }
```

The console at the bottom shows a transaction call: `[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddc4 to: Counter.get() data: 0x6d4...ce63c`.

Tutorial 11: Control Flow - Loops

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

LEARNETH 1.5.1 learneth tutorials

7.2 Control Flow - Loops 11 / 19

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

break

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

Watch a video tutorial on Loop statements.

★ Assignment

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

Check Answer Show answer

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Loop {
5     uint public count;
6     function loop() public {
7         // for loop
8         for (uint i = 0; i < 10; i++) {
9             if (i == 5) {
10                // Skip to next iteration with continue
11                continue;
12            }
13            if (i == 5) {
14                // Exit loop with break
15                break;
16            }
17            count++;
18        }
19
20        // while loop
21        uint j;
22        while (j < 10) {
23            j++;
24        }
25    }
26 }
27
```

0 Listen on all transactions

[call] from: 0x58380a6a701c568545dCfc803Fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 12: Data Structures - Arrays

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

LEARNETH 1.5.1 learneth tutorials

8.1 Data Structures - Arrays 12 / 19

When we remove an element with the `delete` operator all other elements stay the same, which means that the length of the array will stay the same. This will create a gap in our array. If the order of the array is not important, then we can move the last element of the array to the place of the deleted element (line 46), or use a mapping. A mapping might be a better choice if we plan to remove elements in our data structure.

Array length

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

Watch a video tutorial on Arrays.

★ Assignment

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

Check Answer Show answer

Next

Well done! No errors.

```
40 // Delete does not change the array length.
41 // It resets the value at index to it's default value,
42 // in this case 0
43 delete arr[index];
44 }
45
46 contract CompactArray {
47     uint[] public arr;
48
49     // Deleting an element creates a gap in the array.
50     // One trick to keep the array compact is to
51     // move the last element into the place to delete.
52     function remove(uint index) public {
53         // Move the last element into the place to delete
54         arr[index] = arr[arr.length - 1];
55         // Remove the last element
56         arr.pop();
57     }
58
59     function test() public {
60         arr.push(1);
61         arr.push(2);
62         arr.push(3);
63         arr.push(4);
64         // [1, 2, 3, 4]
65
66         remove(1);
67         // [1, 4, 3]
68
69         remove(2);
70         // [1, 4]
71     }
72 }
73
```

0 Listen on all transactions

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 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

LEARNETH

Tutorials list

8.2 Data Structures - Mappings

mapping's name and key in brackets and assigning it a new value (line 16).

Removing values

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

Watch a video tutorial on Mappings.

★ **Assignment**

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

Check Answer Show answer

Next

Well done! No errors.

Explain contract

```
11 // key value mapping
12 // balances[address]
13
14
15 function set(address _addr) public { 25256 gas
16     // Update the value at this address
17     balances[_addr] = _addr.balance;
18 }
19
20 function remove(address _addr) public { 5566 gas
21     // Reset the value to the default value.
22     delete balances[_addr];
23 }
24
25 contract NestedMapping {
26     // Nested mapping (mapping from address to another mapping)
27     mapping(address => mapping(uint => bool)) public nested;
28
29     function get(address _addr1, uint _i) public view returns (bool) { 3159 gas
30         // You can get values from a nested mapping
31         // even when it is not initialized
32         return nested[_addr1][_i];
33     }
34
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 }
```

Tutorial 14: Data Structures - Structs

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

LEARNETH

Tutorials list

8.3 Data Structures - Structs

parameters in parentheses (line 16).

Key-value mapping: We provide the name of the struct and the keys and values as a mapping inside curly braces (line 19).

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 struct's 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.

Explain contract

```
16 // key value mapping
17 // todos[uint]
18
19 // key value mapping
20 todos.push(Todo({text: _text, completed: false}));
21
22 // initialize an empty struct and then update it
23 Todo memory todo;
24 todo.text = _text;
25 // todo.completed initialized to false
26
27 todos.push(todo);
28
29
30 // Solidity automatically created a getter for 'todos' so
31 // you don't actually need this function.
32 function get(uint _index) public view returns (string memory text, bool completed)
33 {
34     Todo storage todo = todos[_index];
35     return (todo.text, todo.completed);
36 }
37
38 // update text
39 function update(uint _index, string memory _text) public { Infinite gas
40     Todo storage todo = todos[_index];
41     todo.text = _text;
42 }
43
44 // update completed
45 function toggleCompleted(uint _index) public { 28995 gas
46     Todo storage todo = todos[_index];
47     todo.completed = !todo.completed;
48 }
49
50 function remove(uint _index) public { Infinite gas
51     delete todos[_index];
52 }
```

Tutorial 15: Data Structures - Enums

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

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

8.4 Data Structures - Enums 15 / 19

We can update the enum value of a variable by assigning it the `uint` representing the enum member (line 30). Shipped would be 1 in this example. Another way to update the value is using the dot operator by providing the name of the enum and its member (line 35).

Removing an enum value

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

Watch a video tutorial on Enums.

Assignment

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

Check Answer Show answer

Next

Well done! No errors.

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

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

function get() public view returns (Status) {
    return status;
}

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

// Update status by passing uint into input
function set(Status _status) public {
    status = _status;
}

// You can update to a specific enum like this
function cancel() public {
    status = Status.Canceled;
}

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

Explain contract

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 16: Data Locations

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

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

9. Data Locations 16 / 19

amount of gas possible.

Assignment

1. Change the value of the `myStruct` member `foo`, inside the function `f`, to 4.
2. Create a new struct `myMemStruct2` with the data location `memory` inside the function `f` and assign it the value of `myMemStruct`. Change the value of the `myMemStruct2` member `foo` to 1.
3. Create a new struct `myMemStruct3` with the data location `memory` inside the function `f` and assign it the value of `myStruct`. Change the value of the `myMemStruct3` member `foo` to 3.
4. Let the function `f` return `myStruct`, `myMemStruct2`, and `myMemStruct3`.

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

Check Answer Show answer

Next

Well done! No errors.

```
function f() public returns (MyStruct memory, MyStruct memory, MyStruct memory) {
    // call f with state variables
    _f(arr, map, myStructs[1]);
    // get a struct from a mapping
    MyStruct storage myStruct = myStructs[1];
    myStruct.foo = 4;
    // create a struct in memory
    MyStruct memory myMemStruct = MyStruct(0);
    MyStruct memory myMemStruct2 = myMemStruct;
    myMemStruct2.foo = 1;

    MyStruct memory myMemStruct3 = myStruct;
    myMemStruct3.foo = 3;
    return (myStruct, myMemStruct2, myMemStruct3);
}

function _f(
    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) {
    // do something with memory array
    _arr[0] = 1;
}

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

Explain contract

Scam Alert Initialize as git repo Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Tutorial 17: Transactions - Ether and Wei

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

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

10.1 Transactions - Ether and Wei 17 / 19

Wei is the smallest subunit of *Ether*, named after the cryptographer Wei Dai. *Ether* numbers without a suffix are treated as `wei` (line 7).

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

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.

Explain contract

```
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 }
```

Tutorial 18: Transactions - Gas and Gas Price

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

REMUX 1.5.1 learneth tutorials

LEARNETH

Tutorials list

10.2 Transactions - Gas and Gas Price 18 / 19

Gas limit

When sending a transaction, the sender specifies the maximum amount of gas that they are willing to pay for. If they set the limit too low, their transaction can 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

Well done! No errors.

Explain contract

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Gas {
5     uint public i = 0;
6     uint public cost = 170367;
7
8     // Using up all of the gas that you send causes your transaction to fail.
9     // State changes are undone.
10    // Gas spent are not refunded.
11    function forever() public {
12        // Here we run a loop until all of the gas are spent
13        // and the transaction fails
14        while (true) {
15            i += 1;
16        }
17    }
18 }
```


Tutorial 19: Transactions - Sending Ether

The screenshot displays the Remix IDE interface. On the left, a sidebar shows the 'LEARNETH' tutorial list, with '10.3 Transactions - Sending Ether' selected. The main editor area contains a Solidity smart contract named 'Charity'. The contract includes a constructor, a 'donate' function, and a 'withdraw' function. The 'withdraw' function uses the 'call' method to send Ether to the owner. The console area at the bottom shows the status 'Well done! No errors.' and a 'Scam Alert' message.

```
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");
    }
}
```

CONCLUSION:

Through this experiment, the basic concepts of Solidity programming were learned by completing practical assignments using the Remix IDE. Important topics such as data types, variables, different types of functions, visibility, modifiers, constructors, control flow statements, data structures, and transactions were studied and applied while creating smart contracts. The hands-on practice helped in understanding how to design, compile, and deploy contracts using the Remix VM. Overall, this experiment helped in building a clear understanding of blockchain concepts and provided a strong foundation for developing and managing smart contracts effectively.