

Experiment No. 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.

Example

```
function add(uint a, uint b) public pure returns (uint sum) {  
    sum = a + b;  
}
```

```
function getValues() public view returns (uint, string memory) {  
    return (age, name);  
}
```

3. 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).

```
modifier onlyOwner() {  
    require(msg.sender == owner, "Not owner");  
    _;  
}  
function restricted() public onlyOwner {  
    // restricted logic  
}
```

- **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.

```
if (condition) {  
    // true block  
} else if (anotherCondition) {  
    // else-if block  
} else {  
    // false block  
}
```

- **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.

```
for (uint i = 0; i < 10; i++) {  
    count++;  
}
```

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 provides three main data locations for reference types such as arrays, structs, mappings, and strings:

1. Storage

- Permanent blockchain storage
- Persistent and expensive
- Default location for state variables

State variables are always stored in **storage** because they must remain on the blockchain.

2. Memory

- Temporary storage
- Exists only during function execution
- Cheaper than storage
- Used for:
 - Local variables
 - Function arguments
 - Function return values

Memory variables are deleted after the function execution is completed.

3. Calldata

- Read-only
- Non-modifiable
- Cheapest data location
- Used for external function parameters
- Contains immutable transaction data

Calldata is more gas-efficient than memory for external functions.

Important Rules

- State variables → Always **storage**
- Function parameters → Prefer **calldata** (for external functions) or **memory**
- Local reference variables → Must explicitly specify data location

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

Ether Units

- Smallest unit of Ether is **wei**
- 1 Ether = 10^{18} wei
- 1 Gwei = 10^9 wei (commonly used for gas prices)

Gas Concept

Gas represents the computational effort required to execute a transaction.

Gas Components:

- **Gas Limit**
Maximum gas a sender is willing to spend
- **Gas Price**
Price per unit of gas (in wei or gwei)

Total Transaction Fee

Total Fee = Gas Used \times Gas Price

- Total Fee = Gas Used \times Gas Price

Important Global Variables

- msg.value \rightarrow Amount of wei sent with the transaction
- tx.gasprice \rightarrow Gas price of the current transaction
- msg.sender \rightarrow Address of the caller

Sending Ether in Solidity

To send Ether, the function must be marked as payable.

Example:

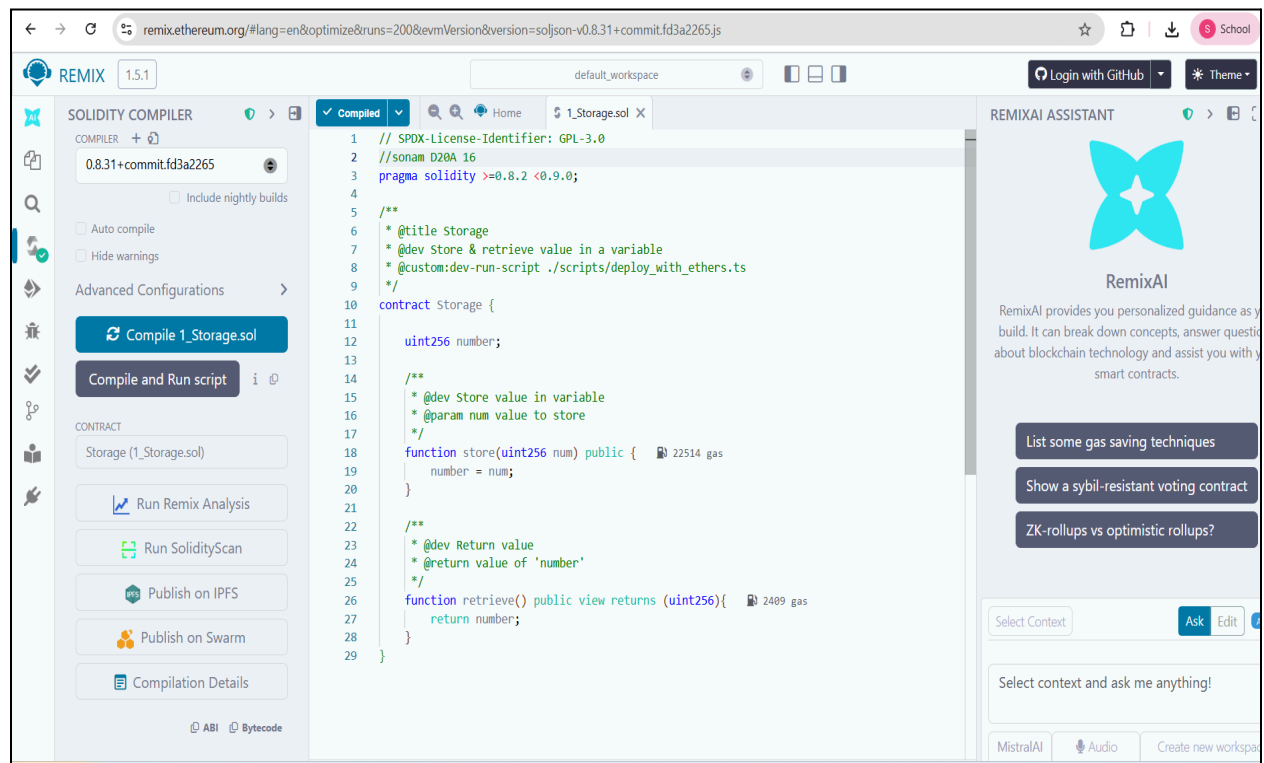
```
function sendEther(address payable recipient) public payable {  
    recipient.transfer(msg.value);  
}
```

Explanation:

- payable → Allows function to receive Ether
- recipient.transfer(msg.value) → Transfers Ether to recipient
- msg.value → Amount of Ether sent with transaction

Implementation:

- Tutorial no. 1 – Compile the code



- Tutorial no. 1 – Deploy the contract

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing the 'Counter' contract selected. The 'GAS LIMIT' is set to 'Estimated Gas' (3000000) and the 'VALUE' is '0 Wei'. The 'CONTRACT' field shows 'Counter - remix-project-org/remix-w'. The 'Deploy' button is highlighted. Below the sidebar, the 'Transactions recorded' and 'Deployed Contracts' sections are visible. The main editor displays the Solidity code for the 'Counter' contract, which includes a 'get()' function and an 'inc()' function. The 'Explain contract' panel at the bottom provides a summary of the contract's functionality and the libraries it uses.

- Tutorial no. 1 – get

The screenshot shows the Remix IDE interface after the contract has been deployed. The 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing the 'Counter' contract selected. The 'At Address' button is highlighted. The 'Transactions recorded' and 'Deployed Contracts' sections are visible. The main editor displays the Solidity code for the 'Counter' contract. The 'Explain contract' panel at the bottom provides a summary of the contract's functionality and the libraries it uses. The 'Logs' panel at the bottom shows the transaction logs, including the 'call to Counter.get' and the 'data' returned by the function.

- Tutorial no. 1 – Increment

The screenshot displays the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar shows a deployed contract named 'COUNTER AT 0xD91...39138'. The contract's state is visible, showing 'Balance: 0 ETH' and 'uint256: 0'. The central code editor displays the Solidity code for the counter, including functions 'inc()' and 'dec()'. The right sidebar shows the transaction logs, indicating a successful call to 'Counter.inc()' with a gas cost of 5218.

- Tutorial no. 1 – Decrement

The screenshot displays the Remix Ethereum IDE interface for the 'Decrement' tutorial. The central code editor shows the same Solidity code as the previous screenshot, but the transaction logs in the right sidebar show a successful call to 'Counter.dec()' with a gas cost of 9189. The left sidebar shows the contract state, with 'uint256' now set to 1. A tooltip at the bottom left indicates that the call data is used to send to the fallback function of the contract.

- Tutorial no. 2

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

LEARNETH 1.5.1 learneth tutorials

Compile Home introduction.sol basicSyntax.sol X

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract MyContract {
5     string public name = "Alice";
6 }
7
```

2. Basic Syntax 2 / 19

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.

Check Answer Show answer Next Well done! No errors.

Explain contract

AI copilot Listen on all transactions sonam

Type the library name to see available commands.

Scan 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)

26°C Mostly sunny Search ENG IN 09:43 13-02-2026

- Tutorial no. 3

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

LEARNETH 1.5.1 learneth tutorials

Compile Home introduction.sol basicSyntax.sol primitiveDataTypes.sol X

```
10
11 int8 public i8 = -1;
12 int public i256 = 456;
13 int public i = -123;
14
15 address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8f733c;
16
17 // 1. New public address (different from addr)
18 address public newAddr = 0xAb8483F64d9C6d1EcF9bB49Ae677dD3315835Cb2;
19 // 2. Public negative number
20 int8 public neg = -5;
21
22 // 3. Smallest uint size & smallest value
23 uint8 public newU = 0;
24
25 // Default values
26 bool public defaultBool;
27 uint public defaultUInt;
28 int public defaultInt;
29 address public defaultAddr;
30 }
31
```

3. Primitive Data Types 3 / 19

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.

Explain contract

AI copilot Listen on all transactions sonam

Type the library name to see available commands.

Scan 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)

26°C Mostly sunny Search ENG IN 09:46 13-02-2026

- Tutorial no. 4

The screenshot shows the Remix Ethereum IDE interface. The sidebar on the left displays the 'LEARNETH' tutorial list, with '4. Variables' selected. The main editor shows the following Solidity code:

```
2 pragma solidity ^0.8.3;
3
4 contract Variables {
5     // State variables are stored on the blockchain.
6     string public text = "Hello";
7     uint public num = 123;
8
9     // 1. New public state variable
10    uint public blockNumber;
11
12    function doSomething() public {
13        // Local variables are not saved to the blockchain.
14        uint i = 456;
15
16        // Global variables
17        uint timestamp = block.timestamp;
18        address sender = msg.sender;
19
20        // 2. Assign current block number to state variable
21        blockNumber = block.number;
22    }
23 }
24
```

The bottom console area shows the 'Explain contract' button and a search bar with 'SONAM' entered. The status bar at the bottom indicates 'Well done! No errors.'

- Tutorial no. 5

The screenshot shows the Remix Ethereum IDE interface. The sidebar on the left displays the 'LEARNETH' tutorial list, with '5.1 Functions - Reading and Writing to a State Variable' selected. The main editor shows the following Solidity code:

```
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
8     // 1. New public bool variable initialized to true
9     bool public b = true;
10
11    // Write function
12    function set(uint _num) public {
13        num = _num;
14    }
15
16    // Read function
17    function get() public view returns (uint) {
18        return num;
19    }
20
21    // 2. Function to return value of b
22    function get_b() public view returns (bool) {
23        return b;
24    }
25 }
26
```

The bottom console area shows the 'Explain contract' button and a search bar with 'Sonam' entered. The status bar at the bottom indicates 'Well done! No errors.'

• Tutorial no. 6

The screenshot shows the REMIX IDE interface for Tutorial 5.2: Functions - View and Pure. The sidebar on the left contains a 'Tutorials list' and a 'Syllabus' section. The main editor displays Solidity code for a contract named 'Learneth'. The code includes functions for adding values and a new function 'addToX2' that updates a state variable. The bottom status bar shows the system temperature as 26°C and the date as 13-02-2026.

```
7 // Promise not to modify the state.
8 function addToX(uint y) public view returns (uint) {
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) {
14     return i + j;
15 }
16
17 // New function that modifies state
18 function addToX2(uint y) public {
19     x = x + y;
20 }
21 }
22 sonam
```

• Tutorial no. 7

The screenshot shows the REMIX IDE interface for Tutorial 5.3: Functions - Modifiers and Constructors. The sidebar on the left contains a 'Tutorials list' and a 'Syllabus' section. The main editor displays Solidity code for a contract named 'Learneth'. The code includes a constructor function, a modifier 'noReentrancy', and a function 'decrement'. The bottom status bar shows the system temperature as 26°C and the date as 13-02-2026.

```
24 public
25 onlyOwner
26 validAddress(_newOwner)
27 {
28     owner = _newOwner;
29 }
30
31 modifier noReentrancy() {
32     require(!locked, "no reentrancy");
33     locked = true;
34     _;
35     locked = false;
36 }
37
38 function decrement(uint i) public noReentrancy {
39     x -= i;
40
41     if (i > 1) {
42         decrement(i - 1);
43     }
44 }
45
46 // New modifier to increase x
47 modifier increase(uint _amount) {
48     x += _amount;
49     _;
50 }
51
52 // Function body must be empty
53 function increaseX(uint _amount) public increase(_amount) {
54 }
55 }
56 }
57 sonam
```

- Tutorial no. 8

remix.ethereum.org/?#activate=udapp.solidity.LearnEth&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.f3a2265.js

REMX 1.5.1

learneth tutorials

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

5.4 Functions - Inputs and Outputs 8 / 19

Input and Output restrictions

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

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

From the [Solidity documentation](#).

Arrays can be used as parameters, as shown in the function `arrayInput` (line 71). Arrays can also be used as return parameters as shown in the function `arrayOutput` (line 76).

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

Watch a video tutorial on [Function Outputs](#).

Assignment

Create a new function called `returnTwo` that returns the values `x` and `y` without using a return statement.

Check Answer Show answer

Next

Well done! No errors.

Explain contract

AI copilot

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)

Nifty smicap -1.74%

Search

ENG IN 09:56 13-02-2026

- Tutorial no. 9

remix.ethereum.org/?#activate=udapp.solidity.LearnEth&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.f3a2265.js

REMX 1.5.1

learneth tutorials

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

6. Visibility 9 / 19

When you uncomment the `testPrivateFunc` lines so 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.

Check Answer Show answer

Next

Well done! No errors.

AI copilot

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)

Nifty smicap -1.74%

Search

ENG IN 09:58 13-02-2026

• Tutorial no. 10

The screenshot shows the Remix Ethereum IDE interface. The sidebar on the left displays the 'LEARNETH' tutorial list, with '7.1 Control Flow - If/Else' selected. The main editor shows the Solidity code for the tutorial, which includes an if-else statement and a ternary operator. The bottom status bar indicates 'Well done! No errors.' and 'RemixAI Copilot (enabled)'.

```
7   if (x < 10) {
8       return 0;
9   } else if (x < 20) {
10      return 1;
11   } else {
12      return 2;
13   }
14
15
16 function ternary(uint _x) public pure returns (uint) {
17     return _x < 10 ? 1 : 2;
18 }
19
20 // New Function
21 function evenCheck(uint _x) public pure returns (bool) {
22     return _x % 2 == 0 ? true : false;
23 }
24
25 sonam
```

• Tutorial no. 11

The screenshot shows the Remix Ethereum IDE interface. The sidebar on the left displays the 'LEARNETH' tutorial list, with '7.2 Control Flow - Loops' selected. The main editor shows the Solidity code for the tutorial, which includes a for loop and a while loop. The bottom status bar indicates 'Well done! No errors.' and 'RemixAI Copilot (enabled)'.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Loop {
5
6     // Public state variable
7     uint public count;
8
9     function loop() public {
10
11         // for loop
12         for (uint i = 0; i < 10; i++) {
13
14             if (i == 3) {
15                 continue;
16             }
17
18             if (i == 5) {
19                 break;
20             }
21
22             count++; // increment count
23         }
24
25         // while loop
26         uint j;
27         while (j < 5) { // run 5 times
28             count++;
29             j++;
30         }
31     }
32 }
33 sonam
```

- Tutorial no. 12

REMIX 1.5.1

learneth tutorials

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.1 Data Structures - Arrays 12 / 19

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.

```
12
13 // New fixed-size array (smallest possible size = 3)
14 uint[3] public arr3 = [0, 1, 2];
15
16 function get(uint i) public view returns (uint) {
17     return arr[i];
18 }
19
20 // Modified to return arr3
21 function getArr() public view returns (uint[3] memory) {
22     return arr3;
23 }
24
25 function push(uint i) public {
26     arr.push(i);
27 }
28
29 function pop() public {
30     arr.pop();
31 }
32
33 function getLength() public view returns (uint) {
34     return arr.length;
35 }
36
37 function remove(uint index) public {
38     delete arr[index];
39 }
40 }
41 sonam
```

Scan 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)

24°C Mostly sunny Search ENG IN 10:01 13-02-2026

- Tutorial no. 13

REMIX 1.5.1

learneth tutorials

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.2 Data Structures - Mappings 13 / 19

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.

```
13
14 function set(address _addr) public {
15     // Update the value at this address
16     balances[_addr] = _addr.balance;
17 }
18
19 function remove(address _addr) public {
20     // Reset the value to the default value.
21     delete balances[_addr];
22 }
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) {
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(
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 {
44         delete nested[_addr1][_i];
45     }
46 }
47
48 sonam
```

Scan 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)

24°C Search ENG 10:01 13-02-2026

- Tutorial no. 14

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

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.3 Data Structures - Structs 14 / 19

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.

```
12
13
14 function create(string memory _text) public {
15     todos.push(Todo(_text, false));
16     todos.push(Todo({text: _text, completed: false}));
17
18     Todo memory todo;
19     todo.text = _text;
20     todos.push(todo);
21 }
22
23 function get(uint _index) public view returns (string memory text, bool completed) {
24     Todo storage todo = todos[_index];
25     return (todo.text, todo.completed);
26 }
27
28 function update(uint _index, string memory _text) public {
29     Todo storage todo = todos[_index];
30     todo.text = _text;
31 }
32
33 function toggleCompleted(uint _index) public {
34     Todo storage todo = todos[_index];
35     todo.completed = !todo.completed;
36 }
37
38 // New remove function
39 function remove(uint _index) public {
40     delete todos[_index];
41 }
42
```

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)

24°C

- Tutorial no. 15

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

Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.4 Data Structures - Enums 15 / 19

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.

```
20
21
22 function set(Status _status) public {
23     status = _status;
24 }
25
26 function cancel() public {
27     status = Status.Canceled;
28 }
29
30 function reset() public {
31     delete status;
32 }
33
34 // New Enum
35 enum Size {
36     S,
37     M,
38     L
39 }
40
41 // Initialize variable of type Size
42 Size public sizes = Size.S;
43
44 // Getter function
45 function getSize() public view returns (Size) {
46     return sizes;
47 }
48
```

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)

24°C Mostly sunny

Search

ENG IN 10:07 13-02-2026

- Tutorial no. 16

The screenshot shows the REMIX IDE interface. The left sidebar contains a 'Tutorials list' with '9. Data Locations' (16 / 19) selected. The main editor displays Solidity code for 'mappings.sol'. The code includes comments and function definitions for creating memory copies from storage structs and returning arrays. The bottom status bar shows 'Well done! No errors.' and a 'Next' button.

```
34
35
36 // Create memory copy from storage struct
37 MyStruct memory myMemStruct3 = myStruct;
38 myMemStruct3.foo = 3;
39
40 // Return all three
41 return (myStruct, myMemStruct2, myMemStruct3);
42
43
44 function _f(
45     uint[] storage _arr,
46     mapping(uint => address) storage _map,
47     MyStruct storage _myStruct
48 ) internal {
49     // do something with storage variables
50 }
51
52 function g(uint[] memory _arr) public pure returns (uint[] memory) {
53     return _arr;
54 }
55
56 function h(uint[] calldata _arr) external
57     // do something with calldata array
58 }
59
60 sonam
```

- Tutorial no. 17

The screenshot shows the REMIX IDE interface. The left sidebar contains a 'Tutorials list' with '10.1 Transactions - Ether and Wei' (17 / 19) selected. The main editor displays Solidity code for 'etherAndWei.sol'. The code defines a contract 'EtherUnits' with public variables for 'oneWei', 'isOneWei', 'oneEther', 'isOneEther', 'oneGwei', and 'isOneGwei'. The bottom status bar shows 'Well done! No errors.' and a 'Next' button.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract EtherUnits {
5
6     uint public oneWei = 1 wei;
7     bool public isOneWei = 1 wei == 1;
8
9     uint public oneEther = 1 ether;
10    bool public isOneEther = 1 ether == 1e18;
11
12    // Must match test name exactly
13    uint public oneGwei = 1 gwei;
14
15    bool public isOneGwei = 1 gwei == 1e9;
16 }
17
```


- Tutorial no. 18

The screenshot displays the Remix IDE interface. The top bar shows the URL: `remix.ethereum.org/?#activate=udapp,solidity.LearnEth&lang=en&optimize&runs=200&evmVersion=soljson-v0.8.31+commit.fd3a2265.js`. The left sidebar contains a 'LEARNETH' section with a 'Tutorials list' and a 'Syllabus' button. The main workspace shows a Solidity script for calculating gas costs. The script includes a pragma statement for Solidity 0.8.3, a contract named 'Gas' with two public variables: 'i' of type 'uint' and 'cost' of type 'uint' with a value of 170367. A function named 'forever' is defined, which runs a loop until all gas is spent and the transaction fails, incrementing 'i' by 1 in each iteration. The bottom status bar indicates 'Well done! No errors.' and 'RemixAI Copilot (enabled)'.

```

1  pragma solidity ^0.8.3;
2
3
4  contract Gas {
5      uint public i = 0;
6      uint public cost = 170367;
7
8
9      function forever() public {
10         // Here we run a loop until all of the gas are spent
11         // and the transaction fails
12         while (true) {
13             i += 1;
14         }
15     }
16 }
17

```

- Tutorial no. 19

[illegible]

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.