

**Experiment :04**  
**Blockchain Lab**

**Aim:** Hands on Solidity Programming Assignments for creating Smart Contracts

**Theory:**

## **1. Primitive Data Types and Variables in Solidity**

Solidity provides several primitive data types that serve as the building blocks for smart contract development:

- **Integer Types (uint, int)**  
Used to store numeric values. uint stores unsigned integers (non-negative), while int stores signed integers. Commonly used sizes include uint256, which is optimized for Ethereum Virtual Machine (EVM).
- **Boolean (bool)**  
Stores logical values true or false, typically used in conditions and decision-making.
- **Address (address)**  
Represents a 20-byte Ethereum account or contract address. It is commonly used for identifying users, transferring Ether, and enforcing ownership.
- **Bytes and String (bytes, string)**  
bytes is used for raw binary data, while string is used for storing textual information.

Variables in Solidity are classified as:

- **State Variables:** Stored permanently on the blockchain.
- **Local Variables:** Declared inside functions and exist temporarily.
- **Global Variables:** Predefined variables like msg.sender, msg.value, and block.timestamp that provide blockchain-related information.

## **2. Functions and Function Types**

Functions define the logic of a smart contract. They can accept inputs, perform computations, and return outputs.

Special function types include:

- **pure functions:** Do not read or modify state variables. They only use input parameters and internal logic.

- **view functions:** Can read state variables but cannot modify them.

Using pure and view functions helps reduce gas costs and improves contract efficiency.

### 3. Inputs and Outputs to Functions

Solidity functions can take one or more input parameters and return one or more output values. Inputs allow data to be passed into the contract, while outputs return computed results to the caller.

Functions can also use **named return variables**, which enhance code readability and simplify debugging. This feature is especially useful when returning complex data structures.

### 4. Function Visibility, Modifiers, and Constructors

#### Function Visibility

Solidity provides access control through visibility specifiers:

- **public:** Can be accessed both internally and externally.
- **private:** Accessible only within the same contract.
- **internal:** Accessible within the contract and derived contracts.
- **external:** Can only be called from outside the contract.

#### Modifiers

Modifiers are reusable pieces of code used to alter function behavior. They are commonly used to enforce conditions such as ownership or authorization (e.g., `onlyOwner`).

#### Constructors

A constructor is a special function executed only once during contract deployment. It is typically used to initialize state variables, such as assigning the deployer as the contract owner.

### 5. Control Flow Statements

Solidity supports standard control flow mechanisms:

- **Conditional Statements (if-else)**  
Used to execute code based on conditions, such as validating balances before transactions.

- **Loops (for, while, do-while)**

Used to repeat execution of code blocks. However, excessive looping increases gas consumption, so loops must be used carefully in smart contracts.

## 6. Data Structures in Solidity

Solidity provides several data structures for efficient data management:

- **Arrays**

Store ordered collections of elements. Arrays can be fixed-size or dynamic.

- **Mappings**

Store key-value pairs for fast lookups, such as `mapping(address => uint)` for account balances. Mappings do not support iteration.

- **Structs**

Group related variables into a single custom data type, improving code organization.

- **Enums**

Define a set of named constants, enhancing code readability and reducing logical errors.

## 7. Data Locations

Solidity uses three data locations to manage variable storage:

- **storage**: Permanent data stored on the blockchain (state variables).
- **memory**: Temporary data used during function execution.
- **calldata**: Read-only and gas-efficient location used for external function parameters.

Choosing the correct data location is crucial for optimizing gas usage and contract performance.

## 8. Transactions, Ether, and Gas

- **Ether and Wei**

Ether is the native cryptocurrency of Ethereum. All values are internally handled in Wei, where

**1 Ether =  $10^{18}$  Wei**, allowing precise financial calculations.

- **Gas and Gas Price**

Gas measures the computational effort required to execute transactions or smart contracts. The gas price determines how much Ether is paid per unit of gas.

- **Sending Transactions**

Transactions are used to transfer Ether or interact with smart contracts. Methods like `transfer()`, `send()`, and `call()` are used, with `call()` offering greater flexibility and control.

Efficient contract design minimizes gas consumption and ensures secure execution.

## 1. basic.sol

```
2 //Shravani Patil - 40
3 // compiler version must be greater than or equal to 0.
4 pragma solidity ^0.8.3;
5
6 contract HelloWorld {
7     string public greet = "Hello World!";
8 }
9
10 contract HelloShravani {
11     string public student ="Shravani Patil-40";
12 }
```

The screenshot displays the Remix IDE interface in a web browser. The top navigation bar includes tabs for 'Blockchain E...', 'nptel.ac.in', 'NOC Home', 'D20A-ITC80', 'KJ BC\_01\_02', and 'Remix - Ethe...'. The address bar shows the URL: `remix.ethereum.org/?#activate=udapp.solidity.LearnEth&lang=en&optimize&runs=200&evmVersion=soljson-v0.8.31+commit.f3a2265js`. The left sidebar contains a 'DEPLOY & RUN TRANSACTIONS' panel with a 'VALUE' input set to '0' and 'Wei', a 'CONTRACT' dropdown showing 'HelloShravani - remix-project-org/re', and a 'Deploy' button. Below this is a 'Transactions recorded' section with a 'Run' button. The main workspace shows a list of transactions with details like 'from', 'to', 'data', and 'hash', each with a 'Debug' button. The right sidebar features the 'REMIXAI ASSISTANT' with a logo, a description of its capabilities, and buttons for 'How to use blob storage?' and 'How to use 1inch?'. At the bottom, there is a 'Scam Alert' banner and a system tray showing '27°C Sunny' and the date '03-02-2026'.

## 2. introduction.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
// SPDX-License-Identifier: MIT
//Shravani Patil - 40
contract Counter {
    uint public count;

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

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

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

The screenshot displays the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel shows the 'Counter' contract selected. The 'VALUE' field is set to '0' Wei. The 'CONTRACT' field shows 'Counter - remix-project-org/remix-w'. The 'Deploy' button is highlighted. Below, the 'Transactions recorded' section shows a list of transactions, including the deployment of the Counter contract and subsequent calls to the 'inc' and 'dec' functions. The 'Deployed Contracts' section at the bottom shows the deployed contract 'COUNTER AT 0xD91...39138'. On the right, the 'REMIXAI ASSISTANT' panel is visible, featuring a chat interface with a blue butterfly icon and a 'Select Context' dropdown. The bottom status bar indicates 'Scam Alert' and 'Initialize as git repo'.

### 3. PrimitiveDataTypes.col

REMIT 1.5.1 learneth tutorials

3. Primitive Data Types 3 / 19

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.

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)

27°C Sunny

REMIT 1.5.1 learneth tutorials

DEPLOY & RUN TRANSACTIONS

VALUE 0 Wei

CONTRACT Primitives - remix-project-org/remix- evm version: osaka

Deploy

At Address Load contract from Address

Transactions recorded 1 i

Run transactions using the latest compilation result

Save Run

Deployed Contracts 5

COUNTER AT 0XD91...39138 (h)

transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() @0xd91...39138 value: 0 wei data: 0xb3b...cfa82 logs: 0 hash: 0xc73...7f571

call to Counter.count

[call] from: 0x58380a6a701c568545dCfcB03Fc8875f56beddC4 to: Counter.count() data: 0x066...61abd

call to Counter.get

[call] from: 0x58380a6a701c568545dCfcB03Fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c

creation of HelloWorld pending...

[vm] from: 0x583...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x91a...06af9

creation of HelloShravan pending...

[vm] from: 0x583...eddC4 to: HelloShravan.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xad9...56774

creation of Primitives pending...

[vm] from: 0x583...eddC4 to: Primitives.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x910...06203

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)

27°C Sunny

## 4. Variables.sol

**LEARNETH**

**4. Variables**  
4 / 19

In this example, we use `block.timestamp` (line 14) to get a Unix timestamp of when the current block was generated and `msg.sender` (line 15) to get the caller of the contract function's address.

A list of all Global Variables is available in the [Solidity documentation](#).

Watch video tutorials on [State Variables](#), [Local Variables](#), and [Global Variables](#).

★ **Assignment**

1. Create a new public state variable called `blockNumber`.
2. Inside the function `doSomething()`, assign the value of the current block number to the state variable `blockNumber`.

Tip: Look into the global variables section of the Solidity documentation to find out how to read the current block number.

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

Next

Well done! No errors.

```
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     string public name = "Shravani Anil Patil-40";
10
11    function doSomething() public {
12        // Local variables are not saved to the blockchain.
13        uint i = 456;
14        blockNumber = block.number;
15
16        // Here are some global variables
17        uint timestamp = block.timestamp; // Current block timestamp
18        address sender = msg.sender; // address of the caller
19    }
20
21
22
23 }
```

**DEPLOY & RUN TRANSACTIONS**

GAS LIMIT  
☒ Estimated Gas  
☐ Custom 3000000

VALUE  
0 Wei

CONTRACT  
Variables - remix-project-org/remix-  
evm version: osaka

[Deploy](#)

[At Address](#) [Load contract from Address](#)

Transactions recorded 11

☒ Run transactions using the latest compilation result

[Save](#) [Run](#)

call to Counter.get

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

✓ [vm] from: 0x583...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x91a...06af9  
creation of HelloShravani pending...

✓ [vm] from: 0x583...eddC4 to: HelloShravani.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xad9...56774  
creation of Primitives pending...

✓ [vm] from: 0x583...eddC4 to: Primitives.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x910...06203  
creation of Variables pending...

✓ [vm] from: 0x583...eddC4 to: Variables.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x47a...28909  
creation of Variables pending...

✓ [vm] from: 0x583...eddC4 to: Variables.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xe8c...4c525

## 5. readandWrite.sol

The screenshot shows the Remix IDE interface. On the left, the 'LEARNETH' sidebar is open, displaying a tutorial titled '5.1 Functions - Reading and Writing to a State Variable'. The tutorial text explains how to set the visibility of a function and declare it as 'view' or 'pure'. It also includes an 'Assignment' section with two tasks: creating a public state variable 'b' of type 'bool' and initializing it to 'true', and creating a public function 'get\_b' that returns the value of 'b'. Below the assignment, there are buttons for 'Check Answer' and 'Show answer', and a 'Next' button. A green message at the bottom of the sidebar says 'Well done! No errors.'

The main editor area shows the 'readAndWrite.sol' file. The code is as follows:

```
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     string public name = "Shravani Anil Patil-40";
9     // You need to send a transaction to write to a state variable.
10    function set(uint _num) public { 22558 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) { 2497 gas
16        return num;
17    }
18
19    function get_b() public view returns (bool) { 2561 gas
20        return b;
21    }
22    // function getName() public view returns (string) {
23        return name;
24    }
```

At the bottom of the IDE, there is a 'Logs' panel showing a single log entry: 'Logs: 0 hash: 0xe8c...4c525'. The bottom status bar shows the temperature as 29°C, the weather as 'Sunny', and the time as 11:27 on 03-02-2026.

The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' panel in the Remix IDE. The 'VALUE' field is set to '0' and the 'CONTRACT' field is set to 'SimpleStorage - remix-project-org/...'. The 'Deploy' button is highlighted. Below the 'Deploy' button, there is a section for 'Transactions recorded' with a dropdown menu showing '12' transactions. The 'Run' button is also visible. The 'Deployed Contracts' section shows a list of deployed contracts, including 'COUNTER AT 0xD91...39138'.

The main editor area shows the 'Logs' panel, which displays a list of transactions. Each transaction entry includes a status icon (green checkmark), a description of the transaction, and a 'Debug' button. The transactions are as follows:

- creation of HelloWorld pending...
- [vm] from: 0x583...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x91a...06af9
- creation of HelloShravani pending...
- [vm] from: 0x583...eddC4 to: HelloShravani.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xad9...56774
- creation of Primitives pending...
- [vm] from: 0x583...eddC4 to: Primitives.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x910...06203
- creation of Variables pending...
- [vm] from: 0x583...eddC4 to: Variables.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x47a...28909
- creation of Variables pending...
- [vm] from: 0x583...eddC4 to: Variables.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xe8c...4c525
- creation of SimpleStorage pending...
- [vm] from: 0x583...eddC4 to: SimpleStorage.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0xae...d352b

The bottom status bar shows the temperature as 29°C, the weather as 'Sunny', and the time as 11:28 on 03-02-2026.



## 6. ViewAndPure.sol

0

☐ Listen on all transactions

Filter with transaction hash or ad...

creation of ViewAndPure pending...

✓

[vm] from: 0x5B3...eddC4 to: ViewAndPure.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x542...fa3d9

creation of ViewAndPure pending...

Debug

▼

✓

[vm] from: 0x5B3...eddC4 to: ViewAndPure.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x188...94217

creation of ViewAndPure pending...

Debug

▼

✓

[vm] from: 0x5B3...eddC4 to: ViewAndPure.(constructor) value: 0 wei data: 0x608...f0033 logs: 0 hash: 0x385...0e536

call to ViewAndPure.x

Debug

▼

CALL

[call] from: 0x5B38Da6a701c568545dCfc803Fc8875f56beddC4 to: ViewAndPure.x() data: 0x0c5...5699c

transact to ViewAndPure.addToX2 pending ...

Debug

▼

✓

[vm] from: 0x5B3...eddC4 to: ViewAndPure.addToX2(uint256) 0x534...0b66c value: 0 wei data: 0xfe3...00007 logs: 0 hash: 0xb7a...8578f

call to ViewAndPure.add

Debug

▼

CALL

[call] from: 0x5B38Da6a701c568545dCfc803Fc8875f56beddC4 to: ViewAndPure.add(uint256,uint256) data: 0x771...00008

Debug

▼

Did you know? You can use the help of AI for Solidity error, click on 'Ask RemixAI'.

RemixAI Copilot (enabled)

Search

ENG IN 11:42 03-02-2026

DEPLOY & RUN TRANSACTIONS

Deployed Contracts 1

VIEWANDPURE AT 0x534...0B6

Balance: 0 ETH

ADDTOX2

y: 7

Calldata Parameters transact

ADD

i: 7

j: 8

Calldata Parameters call

0: uint256: 15

addToX 8

x

Compile

s\_answer.sol variables.sol 3 readAndWrite.sol view

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

Explain contract

0 ☐ Listen on all transactions

## 7. Modifiers and Constructors.sol

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active. It displays 'Deployed Contracts' with a balance of 0 ETH. Below this, there are buttons for 'changeOwner', 'DECREMENT', 'increaseX', 'locked', 'owner', and 'x'. The 'DECREMENT' button is highlighted. In the center, the Solidity code is shown, featuring a `require` statement for a `locked` variable, a `noReentrancy` modifier, and functions `decrement` and `increaseX`. The right panel shows the compiled code. At the bottom, there is a button labeled 'Explain contract' and a checkbox for 'Listen on all transactions'.

```
41 require(!locked, "No reentrancy");
42
43 locked = true;
44 _;
45 locked = false;
46 }
47
48 function decrement(uint i) public noReentrancy { infinite gas
49     x -= i;
50
51     if (i > 1) {
52         decrement(i - 1);
53     }
54 }
55
56 function increaseX(uint y) public { infinite gas
57     x+=y;
58 }
59 }
```

## 8. inputs and Outputs.sol

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active. It displays 'Deployed Contracts' with a balance of 0 ETH. Below this, there are buttons for 'arrayOutput', 'assigned', 'destructuringAss...', 'named', 'returnMany', and 'returnTwo'. The 'arrayOutput' button is highlighted. In the center, the Solidity code is shown, featuring a `uint[]` array, a `view` modifier, and functions `arrayOutput` and `returnTwo`. The right panel shows the compiled code. At the bottom, there is a button labeled 'Explain contract' and a checkbox for 'Listen on all transactions'.

```
72 // Can use array for output
73 uint[] public arr;
74
75 function arrayOutput() public view returns (uint[] memory) { infinite gas
76     return arr;
77 }
78
79 function returnTwo() 472 gas
80     public
81     pure
82     returns (
83         int i,
84         bool b
85     )
86 {
87     i = -2;
88     b = true;
89 }
90
91 }
```



## 10.IfElse.sol

Balance: 0 ETH

EVENCHECK

y: "8"

Calldata Parameters call

0: bool: true

FOO

x: "7"

Calldata Parameters call

0: uint256: 0

ternary uint256 \_x

Low level interactions

CALLDATA

Transact

```
11 |         return 2;
12 |     }
13 | }
14 |
15 | function ternary(uint _x) public pure returns (uint) { infinite gas
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 | //Shravani Anil Patil-40
25 | function evenCheck(uint y) public pure returns (bool) { infinite gas
26 |     return y%2 == 0 ? true : false;
27 | }
28 | }
```

Explain contract

0 Listen on all transactions

## 11. Loops.sol

DEPLOY & RUN TRANSACTIONS

Deploy

At Address Load contract from Address

Transactions recorded 29

Deployed Contracts 1

LOOP AT 0X36C...CA07B (MEM)

Balance: 0 ETH

loop

count

0: uint256: 9

```
6 | function loop() public { infinite gas
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 | } // Shravani Patil-40
20 | // while loop
21 | uint j;
22 | while (j < 10) {
23 |     j++;
24 | }
25 |
26 | }
27 | }
```

Explain contract

## 12. Arrays.sol

The screenshot displays the Remix IDE interface. On the left, the 'ARRAY AT 0XD20...D2358 (MF)' window shows the execution environment with a balance of 0 ETH. It includes buttons for 'pop', 'PUSH', 'remove', 'arr', 'arr2', 'arr3', 'get', 'getArr', 'getLength', and 'myFixedSizeArr'. The 'getArr' button is highlighted, showing the output '0: uint256[3]: 0,1,2'. Below it, 'getLength' shows '0: uint256: 2'. The 'myFixedSizeArr' button is also visible. On the right, the 'ARRAYS.sol' contract is shown, starting with a comment '//Shravani Patil-40'. The contract defines a 'test()' function that pushes elements 1, 2, 3, and 4 into an array, then removes the first and second elements, leaving [1, 4, 3]. The function is public and takes infinite gas. The bottom status bar shows 'call to Array.getArr'.

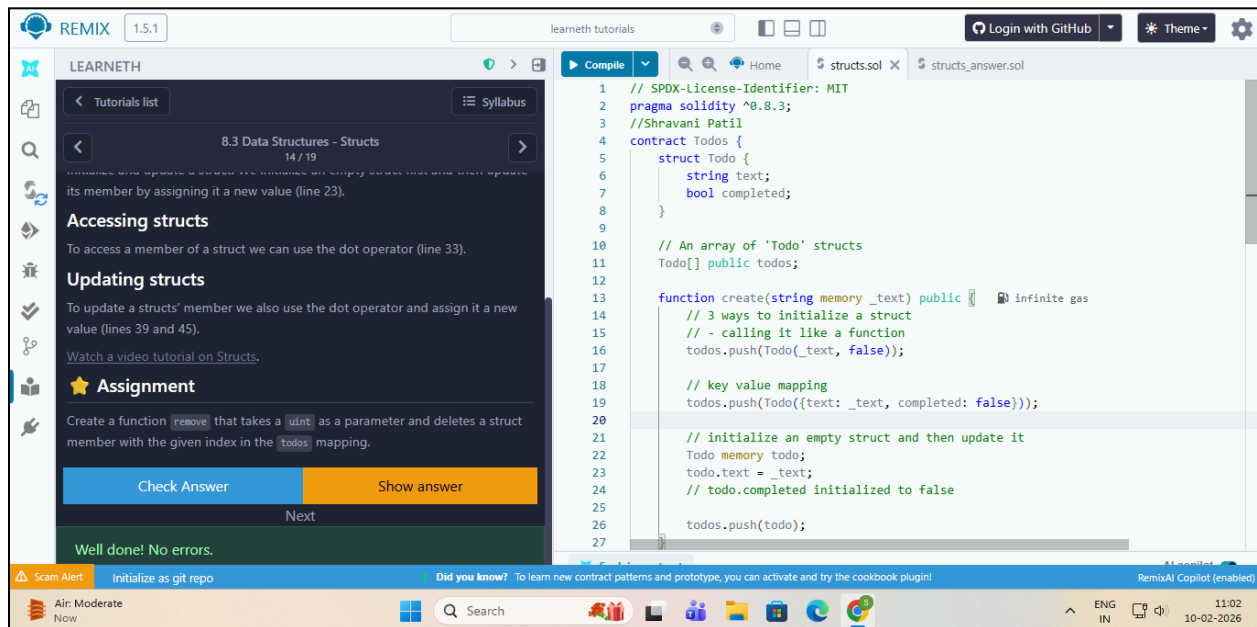
```
59 //Shravani Patil-40
60 function test() public { infinite gas
61     arr.push(1);
62     arr.push(2);
63     arr.push(3);
64     arr.push(4);
65     // [1, 2, 3, 4]
66
67     remove(1);
68     // [1, 4, 3]
69
70     remove(2);
71     // [1, 4]
72 }
73 }
```

## 13. Mappings.sol

The screenshot displays the Remix IDE interface. On the left, the 'LEARNETH' window shows a tutorial for '8.2 Data Structures - Mappings'. It explains that the delete operator can be used to delete a value associated with a key, setting it to the default value of 0. It includes an 'Assignment' section with three tasks: 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. Below the assignment, there are buttons for 'Check Answer', 'Show answer', and 'Next'. The status bar shows 'Well done! No errors.' On the right, the 'MAPPINGS.sol' contract is shown, starting with a comment '// SPDX-License-Identifier: MIT'. The contract defines a 'Mapping' contract with a 'balances' mapping of 'address' to 'uint'. It includes functions 'get', 'set', and 'remove' that work with the 'balances' mapping. The 'get' function returns the value at a given address, 'set' updates the value, and 'remove' resets the value to the default value. The 'NestedMapping' contract is also shown, which is a mapping from 'address' to another mapping of 'uint' to 'bool'. The bottom status bar shows 'Estimated execution cost: 25256 gas'.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //Shravani Patil-40
4 contract Mapping {
5     // Mapping from address to uint
6     mapping(address => uint) public balances;
7
8     function get(address _addr) public view returns (uint) { 2872 gas
9         // Get the value at this address
10         return balances[_addr];
11     }
12
13     function set(address _addr) public { 25256 gas
14         // Update the value at this address
15         balances[_addr] = _addr.balance;
16     }
17
18     function remove(address _addr) public { 5566 gas
19         // Reset the value to the default value.
20         delete balances[_addr];
21     }
22 }
23
24 contract NestedMapping {
25     // Nested mapping (mapping from address to another mapping)
26     mapping(address => mapping(uint => bool)) public nested;
27 }
```

## 14.Struct.sol



REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.3 Data Structures - Structs 14 / 19

Accessing structs

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

Updating structs

To update a 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

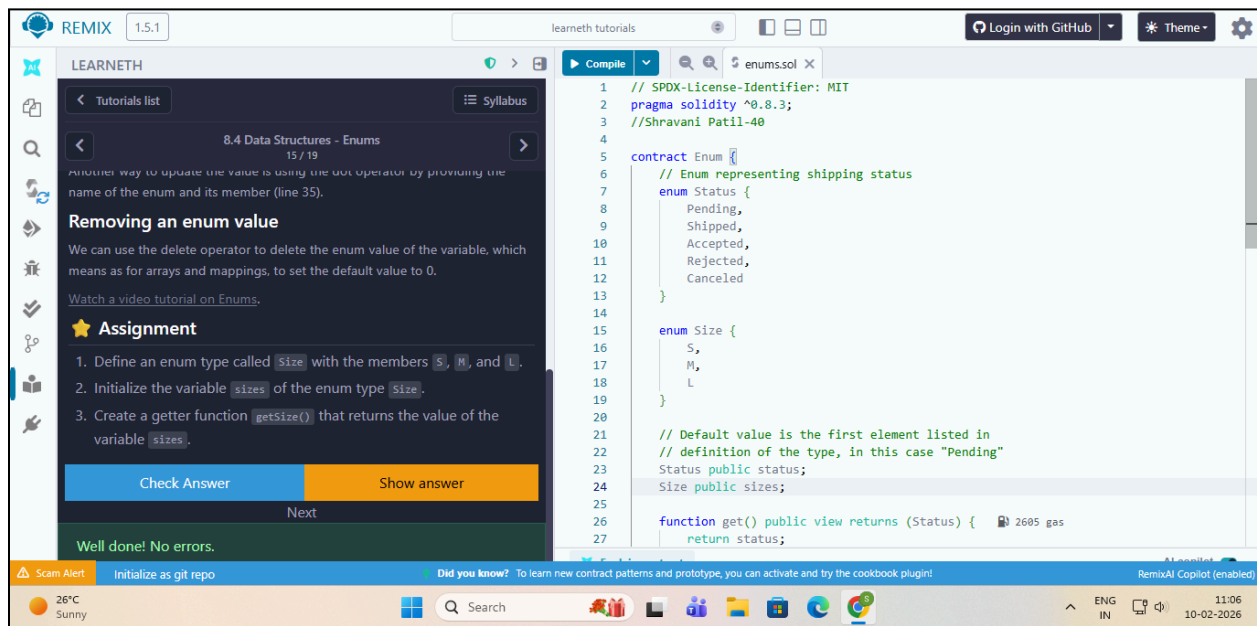
Well done! No errors.

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)

Air: Moderate Now Search ENG IN 11:02 10-02-2026

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //Shravani Patil
4 contract Todos {
5     struct Todo {
6         string text;
7         bool completed;
8     }
9
10    // An array of 'Todo' structs
11    Todo[] public todos;
12
13    function create(string memory _text) public {
14        // 3 ways to initialize a struct
15        // - calling it like a function
16        todos.push(Todo(_text, false));
17
18        // key value mapping
19        todos.push(Todo({text: _text, completed: false}));
20
21        // initialize an empty struct and then update it
22        Todo memory todo;
23        todo.text = _text;
24        // todo.completed initialized to false
25
26        todos.push(todo);
27    }
28 }
```

## 15.enums.sol



REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.4 Data Structures - Enums 15 / 19

Removing an enum value

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

Watch a video tutorial on Enums.

★ Assignment

1. Define an enum type called `Size` with the members `S`, `M`, and `L`.

2. Initialize the variable `sizes` of the enum type `Size`.

3. Create a getter function `getSize()` that returns the value of the variable `sizes`.

Check Answer Show answer

Next

Well done! No errors.

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)

26°C Sunny Search ENG IN 11:06 10-02-2026

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //Shravani Patil-40
4
5 contract Enum {
6     // Enum representing shipping status
7     enum Status {
8         Pending,
9         Shipped,
10        Accepted,
11        Rejected,
12        Canceled
13    }
14
15    enum Size {
16        S,
17        M,
18        L
19    }
20
21    // Default value is the first element listed in
22    // definition of the type, in this case "Pending"
23    Status public status;
24    Size public sizes;
25
26    function get() public view returns (Status) {
27        return status;
28    }
29 }
```

## 16. dataLocations.sol

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar displays the '9. Data Locations' assignment (16 / 19). The assignment text is as follows:

**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`.

Buttons: Check Answer, Show answer, Next

Status: Well done! No errors.

On the right, the 'dataLocations.sol' file is open, showing the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //Shravani Patil-40
4 contract DataLocations {
5     uint[] public arr;
6     mapping(uint => address) map;
7     struct MyStruct {
8         uint foo;
9     }
10    mapping(uint => MyStruct) public myStructs;
11
12    function f() public returns (MyStruct memory, MyStruct memory, MyStruct memory){
13        // call _f with state variables
14        _f(arr, map, myStructs[1]);
15        // get a struct from a mapping
16        MyStruct storage myStruct = myStructs[1];
17        myStruct.foo = 4;
18        // create a struct in memory
19        MyStruct memory myMemStruct = MyStruct(0);
20        MyStruct memory myMemStruct2 = myMemStruct;
21        myMemStruct2.foo = 1;
22
23        MyStruct memory myMemStruct3 = myStruct;
24        myMemStruct3.foo = 3;
25        return (myStruct, myMemStruct2, myMemStruct3);
26    }
27 }
```

The bottom status bar shows '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)', '26°C Sunny', 'Search', and the date '11:07 10-02-2026'.

## 17. etherandWei.sol

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar displays the '10.1 Transactions - Ether and Wei' assignment (17 / 19). The assignment text is as follows:

**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.

Buttons: Check Answer, Show answer, Next

Status: Well done! No errors.

On the right, the 'etherandWei.sol' file is open, showing the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //Shravani Patil-40
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 }
```

The bottom status bar shows 'Scam Alert', 'Initialize as git repo', 'Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!', '10 February 2026', '26°C Sunny', 'Search', and the date 'Tue 11:08 (Local time)'.

## 18.gasandGasPrice.sol

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar lists tutorials, with '10.2 Transactions - Gas and Gas Price' (18 / 19) selected. The main editor displays the Solidity code for 'gasAndGasPrice.sol'. The code includes a SPDX license header, pragma solidity ^0.8.3; and a contract named 'Gas' with a public state variable 'cost' of type 'uint' and a value of 170367. A function 'forever' is defined, which is public and payable, and it contains a 'while (true)' loop that increments a local variable 'i' by 1. The bottom status bar shows 'Well done! No errors.' and 'RemixAI Copilot (enabled)'.

## 19. SendingEther.sol

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar lists tutorials, with '10.3 Transactions - Sending Ether' (19 / 19) selected. The main editor displays the Solidity code for 'SendingEther.sol'. The code includes a SPDX license header, pragma solidity ^0.8.3; and a contract named 'ReceiveEther'. It features a 'receive()' function that is payable and calls 'fallback()' if 'msg.data' is not empty. A 'fallback()' function is also defined, which is payable and calls 'receive()' if 'msg.data' is empty. A 'getBalance()' function is also shown, which is public, view, and returns a 'uint' value of 312. The bottom status bar shows 'Well done! No errors.' and 'RemixAI Copilot (enabled)'.

## Conclusion:

This experiment provides hands-on experience in Solidity programming by exploring core blockchain concepts, data types, control structures, and transaction handling. Understanding these fundamentals enables the design and deployment of secure, efficient, and reliable smart contracts on the Ethereum blockchain.