

## Experiment No. 4

Name: Atharva Prabhu

Batch: B

Div: D20A

Roll No: 45

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

### Theory:

#### 1. Primitive Data Types, Variables, Functions – pure, view

In Solidity, primitive data types form the foundation of smart contract development. Commonly used types include:

- **uint / int:** unsigned and signed integers of different sizes (e.g., uint256, int128).
- **bool:** represents logical values (true or false).
- **address:** holds a 20-byte Ethereum account address, often used for storing user accounts or contract addresses.
- **bytes / string:** store binary data or textual data.

Variables in Solidity can be **state variables** (stored on the blockchain permanently), **local variables** (temporary, created during function execution), or **global variables** (special predefined variables such as msg.sender, msg.value, and block.timestamp).

Functions allow execution of contract logic. Special types of functions include:

- **pure:** cannot read or modify blockchain state; they work only with inputs and internal computations.
- **view:** can read state variables but cannot alter them. This classification helps optimize gas usage and enforces function integrity.

#### 2. Inputs and Outputs to Functions

Functions in Solidity can accept input arguments and return one or more output values. Inputs enable users or other contracts to pass data into the contract, while outputs make it possible to return results after computation. For example, a function can accept an amount in Ether and return whether the transfer was successful. Solidity also allows named return variables, which improve readability and debugging.

#### 3. Visibility, Modifiers and Constructors

- **Function Visibility** defines who can access a function:
  - o **public:** available both inside and outside the contract.
  - o **private:** only accessible within the same contract.

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

### 3. Control Flow: if-else, loops

Control flow in Solidity is similar to traditional programming languages:

- **if-else** allows conditional decision-making in contract logic, e.g., checking if a balance is sufficient before transferring funds.
- **Loops** (for, while, do-while) enable repeated execution of code. For example, iterating through an array of users. However, loops must be used carefully, as excessive iterations increase gas consumption, potentially making the contract expensive to execute.

### 5. Data Structures: Arrays, Mappings, Structs, Enums

- **Arrays**: Can be fixed or dynamic and are used to store ordered lists of elements. Example: an array of addresses for registered users.
- **Mappings**: Key-value pairs that allow quick lookups. Example: mapping(address => uint) for storing balances. Unlike arrays, mappings do not support iteration.
- **Structs**: Allow grouping of related properties into a single data type, such as creating a struct Player {string name; uint score;}.
- **Enums**: Used to define a set of predefined constants, making code more readable. Example: enum Status { Pending, Active, Closed }.

## 6. Data Locations

Solidity uses three primary data locations for storing variables:

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

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

- **Ether and Wei:** Ether is the main currency in Ethereum. All values are measured in Wei, the smallest unit (1 Ether =  $10^{18}$  Wei). This ensures high precision in financial transactions.
- **Gas and Gas Price:** Every transaction consumes gas, which represents computational effort. The gas price determines how much Ether is paid per unit of gas. A higher gas price incentivizes miners to prioritize the transaction.
- **Sending Transactions:** Transactions are used for transferring Ether or interacting with contracts. Functions like `transfer()` and `send()` are commonly used, while `call()` provides more flexibility. Each transaction requires gas, making efficiency in contract design very important.

## Implementation:

- Tutorial no. 1 – Compile the code

The screenshot displays the Remix IDE interface. On the left, a sidebar contains a 'Tutorials list' with '1. Introduction' selected. The main panel on the left shows the introduction text, which explains that the tutorial will cover a counter contract. The right panel is a code editor showing a Solidity script for a 'Counter' contract. The script includes a license identifier, author information, a pragma statement for Solidity version 0.8.3, and three functions: 'get' to retrieve the current count, 'inc' to increment the count by 1, and 'dec' to decrement the count by 1. The bottom status bar includes a 'Scam Alert' icon, a button to 'Initialize as git repo', a 'Did you know?' message about uniswap v4 hooks, and a 'RemixAI Copilot (enabled)' indicator.

```
1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D28A
3 pragma solidity ^0.8.3;
4
5 contract Counter {
6     uint public count;
7
8     // Function to get the current count
9     function get() public view returns (uint) { 2453 gas
10         return count;
11     }
12
13     // Function to increment count by 1
14     function inc() public { infinite gas
15         count += 1;
16     }
17
18     // Function to decrement count by 1
19     function dec() public { infinite gas
20         count -= 1;
21     }
22 }
```

- Tutorial no. 1 – Deploy the contract

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing the environment set to 'Remix VM (Osaka)', the account '0x5B3...eddC4', and a gas limit of 3,000,000. The contract name is 'Counter - remix-project-org/remix-worksp'. The 'Deploy' button is visible. The main editor shows the Solidity code for the 'Counter' contract, which includes a 'get' function to retrieve the current count. The bottom status bar indicates 'Scam Alert', 'Initialize as git repo', and 'RemixAI Copilot (enabled)'.

```

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

```

- Tutorial no. 1 – get

The screenshot shows the 'Deployed Contracts' panel in the Remix IDE. It displays the 'Counter' contract at address '0XD91...39138'. The balance is '0 ETH'. There are four buttons: 'dec', 'inc', 'count', and 'get'. Below these buttons, the state is shown as '0: uint256: 0'. At the bottom, there is a 'Low level interactions' section with a 'CALLDATA' input field and a 'Transact' button.

- Tutorial no. 1 – Increment

Deployed Contracts 1

▼ COUNTER AT 0XD91...39138 (MEMO:)

Balance: 0 ETH

dec

inc

count

get

0: uint256: 3

Low level interactions i

CALLDATA

Transact

- Tutorial no. 1 – Decrement

Deployed Contracts 1

▼ COUNTER AT 0XD91...39138 (MEMO:)

Balance: 0 ETH

dec

inc

count

get

0: uint256: 2

Low level interactions i

CALLDATA

Transact

## • Tutorial no. 2

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH Tutorials list Syllabus

2. Basic Syntax 2 / 19

Don't worry if you didn't understand some concepts like *visibility*, *data types*, or *state variables*. We will look into them in the following sections.

To help you understand the code, we will link in all following sections to video tutorials from the [creator](#) of the Solidity by Example contracts.

Watch a video tutorial on Basic Syntax.

★ Assignment

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

Check Answer Show answer

Next

Well done! No errors.

```
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 // Atharva Prabhu D20A
4 pragma solidity ^0.8.3;
5
6 contract MyContract {
7     // We declare a public string variable named 'name'
8     // and assign it the value "Alice"
9     string public name = "Alice";
10 }
```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

## • Tutorial no. 3

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH Tutorials list Syllabus

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.

```
1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Primitives {
6     string public name = "Alice";
7
8     // 1. A public address variable with a unique value
9     address public newAddr = 0xab8483f64d9c6d1ecf9b849ae677d03315835cb2;
10
11     // 2. A signed integer to hold a negative number
12     // int8 is sufficient for small negative numbers and saves space
13     int8 public neg = -50;
14
15     // 3. The smallest uint size (uint8) and smallest value (0)
16     uint8 public newU = 0;
17 }
```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

- Tutorial no. 4

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list

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.

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

```
1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Variable {
6     uint256 public blockNumber
7     string public name
8     uint public number
9
10    uint public blockNumber;
11    function doSomething() public {
12        // Local variable (Exists only during function execution)
13        uint i = 456;
14
15        // 2. Use a Global Variable to get the current block number
16        // 'block.number' is the global variable you're looking for!
17        blockNumber = block.number;
18
19        // Other examples of global variables:
20        uint timestamp = block.timestamp; // Current block timestamp
21        address sender = msg.sender; // Address of the person calling this function
22    }
```

- Tutorial no. 5

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list

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

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.

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

```
1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract SimpleStorage {
6     // State variable to store a number
7     uint public num;
8
9     // 1. Create a public state variable 'b' initialized to true
10    bool public b = true;
11
12    // Function that changes the state (Writing)
13    function set(uint _num) public {
14        num = _num;
15    }
16
17    // 2. Create a public function 'get_b' that returns the value of 'b'
18    // We use 'view' because we are reading, not changing, the state.
19    function get_b() public view returns (bool) {
20        return b;
21    }
22 }
```



## • Tutorial no. 6

**LEARNETH**

Tutorials list | Syllabus

5.2 Functions - View and Pure  
6 / 19

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

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

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

★ **Assignment**

Create a function called `addtoX2` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`.

Check Answer | Show answer

Next

Well done! No errors.

```

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

```

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

RemixAI Copilot (enabled)

## • Tutorial no. 7

**LEARNETH**

Tutorials list | Syllabus

5.3 Functions - Modifiers and Constructors  
7 / 19

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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract FunctionModifier {
6     // We will use these variables to demonstrate how to use
7     // modifiers.
8     address public owner;
9     uint public x = 10;
10    bool public locked;
11
12    constructor() {
13        // Set the transaction sender as the owner of the contract.
14        owner = msg.sender;
15    }
16
17    // Modifier to check that the caller is the owner of
18    // the contract.
19    modifier onlyOwner() {
20        require(msg.sender == owner, "Not owner");
21        // Underscore is a special character only used inside
22        // a function modifier and it tells Solidity to
23        // execute the rest of the code.
24        _;
25    }
26
27    // Modifiers can take inputs. This modifier checks that the

```

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

RemixAI Copilot (enabled)

## • Tutorial no. 8

**LEARNETH** 5.4 Functions - Inputs and Outputs 8 / 19

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).

Solidity unit testing 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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Function {
6     // Functions can return multiple values.
7     function returnMany() infinite gas
8     public
9     pure
10    returns (
11        uint,
12        bool,
13        uint
14    )
15 {
16     return (1, true, 2);
17 }
18
19 // Return values can be named.
20 function named() infinite gas
21 public
22 pure
23 returns (
24     uint x,
25     bool b,
26     uint y
27 )

```

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)

## • Tutorial no. 9

**LEARNETH** 6. Visibility 9 / 19

### 6. Visibility

The `visibility` specifier is used to control who has access to functions and state variables.

There are four types of visibilities: `external`, `public`, `internal`, and `private`.

They regulate if functions and state variables can be called from inside the contract, from contracts that derive from the contract (child contracts), or from other contracts and transactions.

#### private

- Can be called from inside the contract

#### internal

- Can be called from inside the contract
- Can be called from a child contract

#### public

- Can be called from inside the contract

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Base {
6     // Private function can only be called
7     // - inside this contract
8     // Contracts that inherit this contract cannot call this function.
9     function privateFunc() private pure returns (string memory) { infinite gas
10         return "private function called";
11     }
12
13     function testPrivateFunc() public pure returns (string memory) { infinite gas
14         return
15             contract Base is Base
16             {
17                 // Internal
18                 // - inside
19                 // - inside contracts that inherit this contract
20                 function internalFunc() internal pure returns (string memory) { infinite gas
21                     return "internal function called";
22                 }
23
24                 function testInternalFunc() public pure virtual returns (string memory) { infinite
25                     return internalFunc();
26                 }
27             }

```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

## • Tutorial no. 10

**LEARNETH**

**7.1 Control Flow - If/Else**

10 / 19

[Watch a video tutorial on the if/else statement.](#)

★ **Assignment**

Create a new function called `evenCheck` in the `ifElse` contract:

- That takes in a `uint` as an argument.
- The function returns `true` if the argument is even, and `false` if the argument is odd.
- Use a ternary operator to return the result of the `evenCheck` function.

Tip: The modulo (%) operator produces the remainder of an integer division.

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

Next

Well done! No errors.

**ifElse.sol**

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.0;
4
5 contract IfElse {
6
7     // Original example function for context
8     function foo(uint x) public pure returns (uint) {
9         if (x < 10) {
10             return 0;
11         } else if (x < 20) {
12             return 1;
13         } else {
14             return 2;
15         }
16     }
17
18     // ★ Assignment Solution
19     function evenCheck(uint _number) public pure returns (bool) {
20         // We use the modulo operator (%) to find the remainder.
21         // If _number % 2 is 0, it's even.
22         return _number % 2 == 0 ? true : false;
23     }
24 }

```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

## • Tutorial no. 11

**LEARNETH**

**7.2 Control Flow - Loops**

11 / 19

[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.

**loops.sol**

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Loop {
6     // 1. Create a public uint state variable called count
7     uint public count;
8
9     function loop() public {
10         // To get count to 9, we need the loop to run 9 times.
11         // If there is a 'break' at i == 9, the loop stops before i reaches 10.
12         for (uint i = 0; i < 10; i++) {
13             if (i == 9) {
14                 break; // Stop the loop when i is 9
15             }
16
17             // 2. Increment count by 1 at the end of the for loop logic
18             count++;
19         }
20     }
21 }

```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

- Tutorial no. 12

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.1 Data Structures - Arrays 12 / 19

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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.0;
4
5 contract Array {
6     //
7     // contract Array is Array
8     //
9     uint[3] arr3;
10
11     // Note: We use "pure" if we weren't reading state,
12     // but since we are reading arr3, we use "view".
13     function getArr() public view returns (uint[3] memory) {
14         return arr3;
15     }
16 }

```

Scam Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

- Tutorial no. 13

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

8.2 Data Structures - Mappings 13 / 19

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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Mapping {
6     // Mapping from address to uint
7     mapping(address => uint) public balances;
8
9     function get(address _addr) public view returns (uint) {
10         // Mapping always returns a value.
11         // If the value was never set, it will return the default value.
12         return balances[_addr];
13     }
14
15     function set(address _addr) public {
16         // Update the value at this address
17         balances[_addr] = _addr.balance;
18     }
19
20     function remove(address _addr) public {
21         // Reset the value to the default value.
22         delete balances[_addr];
23     }
24 }
25
26 contract NestedMapping {
27     // Nested mapping (mapping from address to another mapping)

```

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)

- Tutorial no. 14

**LEARNETH**

Tutorials list

8.3 Data Structures - Structs  
14 / 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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Todos {
6     struct Todo {
7         string text;
8         bool completed;
9     }
10
11     // An array of 'Todo' structs
12     Todo[] public todos;
13
14     function create(string memory _text) public {
15         // 3 ways to initialize a struct
16         // - calling it like a function
17         todos.push(Todo(_text, false));
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 }

```

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)

- Tutorial no. 15

**LEARNETH**

Tutorials list

8.4 Data Structures - Enums  
15 / 19

Another way to update the value is using the dot operator by providing the name of the enum and its member (line 35).

### Removing an enum value

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

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

### ★ Assignment

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

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

Next

Well done! No errors.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
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 }

```

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)

- Tutorial no. 16

REMIX 1.5.1 learneth tutorials

Learneth Tutorials List

9. Data Locations 16 / 19

### ★ Assignment

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

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

Check Answer Show answer

Next

Well done! No errors.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract DataLocations {
6     uint[] public arr;
7     mapping(uint => address) map;
8     struct MyStruct {
9         uint foo;
10    }
11    mapping(uint => MyStruct) public myStructs;
12
13    function f() public returns (MyStruct memory, MyStruct memory, MyStruct memory){
14        // call f with state variables
15        _f(arr, map, myStructs[1]);
16        // get a struct from a mapping
17        MyStruct storage myStruct = myStructs[1];
18        myStruct.foo = 4;
19        // create a struct in memory
20        MyStruct memory myMemStruct = MyStruct(0);
21        MyStruct memory myMemStruct2 = myMemStruct;
22        myMemStruct2.foo = 1;
23
24        MyStruct memory myMemStruct3 = myStruct;
25        myMemStruct3.foo = 3;
26        return (myStruct, myMemStruct2, myMemStruct3);
27    }
28 }

```

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)

- Tutorial no. 17

REMIX 1.5.1 learneth tutorials

Learneth Tutorials List

10.1 Transactions - Ether and Wei 17 / 19

Enter 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.

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract EtherUnits {
6     uint public oneWei = 1 wei;
7     // 1 wei is equal to 1
8     bool public isOneWei = 1 wei == 1;
9
10    uint public oneEther = 1 ether;
11    // 1 ether is equal to 10^18 wei
12    bool public isOneEther = 1 ether == 1e18;
13
14    uint public oneGwei = 1 gwei;
15    // 1 ether is equal to 10^9 wei
16    bool public isOneGwei = 1 gwei == 1e9;
17 }

```

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)

- Tutorial no. 18

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

10.2 Transactions - Gas and Gas Price 18 / 19

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](https://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.

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)

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Gas {
6     uint public i = 0;
7     uint public cost = 170367;
8
9     // Using up all of the gas that you send causes your transaction to fail.
10    // State changes are undone.
11    // Gas spent are not refunded.
12    function forever() public { infinite gas
13        // Here we run a loop until all of the gas are spent
14        // and the transaction fails
15        while (true) {
16            i += 1;
17        }
18    }
19 }

```

- Tutorial no. 19

REMIX 1.5.1 learneth tutorials Login with GitHub Theme

LEARNETH

Tutorials list Syllabus

10.3 Transactions - Sending Ether 19 / 19

1. Create a contract called `Charity`.
2. Add a public state variable called `owner` of the type address.
3. Create a donate function that is public and payable without any parameters or function code.
4. Create a withdraw function that is public and sends the total balance of the contract to the `owner` address.

Tip: Test your contract by deploying it from one account and then sending Ether to it from another account. Then execute the withdraw function.

Check Answer Show answer

Next

Well done! No errors.

Scam Alert Initialize as git repo Did you know? You can learn Solidity basics and more using the Learneth plugin. RemixAI Copilot (enabled)

```

1 // SPDX-License-Identifier: MIT
2 // Atharva Prabhu D20A
3 pragma solidity ^0.8.3;
4
5 contract Charity {
6     // 1. Add a public state variable called owner of the type address
7     address public owner;
8
9     // We set the owner to the person who deploys the contract
10    constructor() { 257937 gas 233400 gas
11        owner = msg.sender;
12    }
13
14    // 2. Create a donate function: public and payable
15    // No parameters or function code required
16    function donate() public payable { 141 gas
17        // This allows the contract to accept Ether via this function call
18    }
19
20    // 3. Create a withdraw function: public
21    // Sends the total balance of the contract to the owner address
22    function withdraw() public { infinite gas
23        // Safety Check: Only the owner should be able to trigger a withdraw
24        require(msg.sender == owner, "Only the owner can withdraw funds");
25
26        // Get the current balance of the contract
27    }
28 }

```

**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.