

## 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.

#### 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.

- 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.

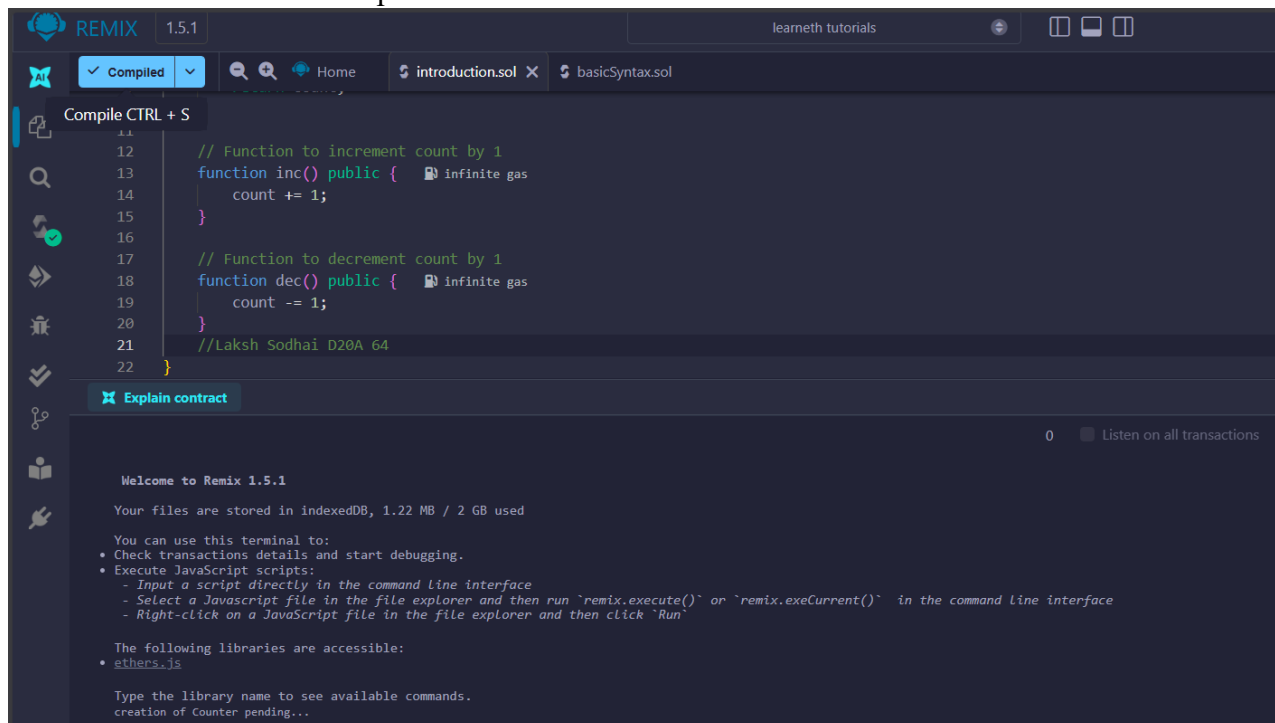
Understanding data locations is essential, as they directly impact gas costs and performance.

## 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



• Tutorial no. 1 – Deploy the contract

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing an account with address 0x5B3...eddC4 and a gas limit of 3000000. The main editor displays the Solidity code for a 'Counter' contract. The bottom panel shows the 'Explain contract' section with a list of JavaScript scripts and the libraries accessible.

```

10  }
11
12  // Function to increment count by 1
13  function inc() public { infinite gas
14      count += 1;
15  }
16
17  // Function to decrement count by 1
18  function dec() public { infinite gas
19      count -= 1;
20  }
21  //Laksh Sodhai D20A 64
22  }
    
```

The 'Explain contract' section shows the following JavaScript scripts:

- Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run "remix.execute()" or "remix.executeCurrent()" in the command line interface
  - Right-click on a Javascript file in the file explorer and then click "Run"
- The following libraries are accessible:
  - ethereum.js

The bottom panel also shows the transaction details for the deployment of the Counter contract.

• Tutorial no. 1 – get

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing the 'Run' button. The main editor displays the Solidity code for the Counter contract. The bottom panel shows the 'Explain contract' section with a list of JavaScript scripts and the libraries accessible.

```

10  }
11
12  // Function to increment count by 1
13  function inc() public { infinite gas
14      count += 1;
15  }
16
17  // Function to decrement count by 1
18  function dec() public { infinite gas
19      count -= 1;
20  }
21  //Laksh Sodhai D20A 64
22  }
    
```

The 'Explain contract' section shows the following JavaScript scripts:

- Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run "remix.execute()" or "remix.executeCurrent()" in the command line interface
  - Right-click on a Javascript file in the file explorer and then click "Run"
- The following libraries are accessible:
  - ethereum.js

The bottom panel also shows the transaction details for the deployment of the Counter contract.

## Tutorial no. 1 – Increment

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing a deployed contract named 'COUNT' at address 0x0D6...99DF9. The contract has a balance of 0 ETH and a 'count' variable of 0. The main editor displays the Solidity code for the 'inc' function, which increments the count by 1. The right sidebar shows the 'Explain contract' panel, which lists transactions for the 'inc' function, including the constructor call and the first increment transaction.

- Tutorial no. 1 – Decrement

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing a deployed contract named 'COUNT' at address 0x0D6...99DF9. The contract has a balance of 0 ETH and a 'count' variable of 0. The main editor displays the Solidity code for the 'dec' function, which decrements the count by 1. The right sidebar shows the 'Explain contract' panel, which lists transactions for the 'dec' function, including the constructor call and the first decrement transaction.

## • Tutorial no.

2

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**2. Basic Syntax** 2 / 19

We also define the **visibility** of the variable, which specifies from where you can access it. In this case, it's a **public** variable that you can access from inside and outside the contract.

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 pragma solidity ^0.8.3;
3 // Laksh Sodhai D20A 64
4
5 contract MyContract {
6     string public name = "Alice";
7 }
8
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

CALL [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddc4 to: Counter.get() data: 0x6d4...ce63c  
transact to Counter.inc pending ...

[vm] from: 0x583...edd4 to: Counter.inc() 0xc06...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transact to Counter.dec pending ...

[vm] from: 0x583...edd4 to: Counter.dec() 0xc06...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

## • Tutorial no. 3

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**3. Primitive Data Types** 3 / 19

Strings, and more in the [Solidity documentation](#).

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

Watch a video tutorial on Primitive Data Types.

**★ Assignment**

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

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

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

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Primitives {
5
6     address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;
7
8     address public newAddr = 0x58380da6a701c568545dcfc803fc8875f56beddc4;
9     // Laksh Sodhai D20A 64
10
11     int public neg = -1;
12
13     uint8 public newU = 0;
14 }
15
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

CALL [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddc4 to: Counter.get() data: 0x6d4...ce63c  
transact to Counter.inc pending ...

[vm] from: 0x583...edd4 to: Counter.inc() 0xc06...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transact to Counter.dec pending ...

[vm] from: 0x583...edd4 to: Counter.dec() 0xc06...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

## • Tutorial no.

4

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**4. Variables** 4 / 19

used to retrieve information about the blockchain, particular addresses, contracts, and transactions.

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

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

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

★ **Assignment**

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

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

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

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Variables {
5
6     uint public blockNumber;
7
8     function doSomething() public { 2255 gas
9         blockNumber = block.number;
10    }
11    // Laksh Sodhai D20A 64
12 }
13
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or ad...

CALL [call] from: 0x58380d6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...99Df9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...99Df9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

## • Tutorial no. 5

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

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

If the function takes inputs like our `set` function (line 9), you must specify the parameter types and names. A common convention is to use an underscore as a prefix for the parameter name to distinguish them from state variables.

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

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

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

★ **Assignment**

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

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

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract SimpleStorage {
5
6     bool public b = true;
7
8     function get_b() public view returns (bool) { 2472 gas
9         return b;
10    }
11    //Laksh Sodhai D20A 64
12 }
13
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or ad...

CALL [call] from: 0x58380d6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...99Df9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...99Df9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

## • Tutorial no.

6

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**5.2 Functions - View and Pure** 6 / 19

5. Using inline assembly that contains certain opcodes."

From the [Solidity documentation](#).

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

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

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

★ **Assignment**

Create a function called `addX2` that takes the parameter `x` 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.

```
8 function addX2(uint y) public {
9     x += y;
10 }
11
12 function addToView(uint y) public view returns (uint) {
13     return x + y;
14 }
15
16 function addPure(uint i, uint j) public pure returns (uint) {
17     return i + j;
18 }
19 //Laksh Sodhai D20A 64
20
21
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ... [Debug](#)

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d5d37  
transaction to Counter.dec pending ... [Debug](#)

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581 [Debug](#)

## • Tutorial no. 7

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**5.3 Functions - Modifiers and Constructors** 7 / 19

A constructor function is executed upon the creation of a contract. You can use it to run contract initialization code. The constructor can have parameters and is especially useful when you don't know certain initialization values before the deployment of the contract.

You declare a constructor using the `constructor` keyword. The constructor in this contract (line 11) sets the initial value of the owner variable upon the creation of the contract.

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

★ **Assignment**

1. Create a new function, `increaseX`, in the contract. The function should take an input parameter of type `uint` and increase the value of the variable `x` by the value of the input parameter.

2. Make sure that `x` can only be increased.

3. The body of the function `increaseX` should be empty.

Tip: Use modifiers.

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

Next

Well done! No errors.

```
57
58
59     locked = false;
60 }
61
62 function decrement(uint i) public noReentrancy {
63     x -= i;
64     if (i > 1) {
65         decrement(i - 1);
66     }
67 }
68 //Laksh Sodhai D20A 64
69
```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

[call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ... [Debug](#)

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d5d37  
transaction to Counter.dec pending ... [Debug](#)

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581 [Debug](#)



## • Tutorial no.

8

**LEARNETH** 1.5.1

learneth tutorials

Remix

LearnEth is modifying remix-project-org/remix-workshops/5.4 Functions - Inputs and Outputs/inputsAndOutputs\_test.sol

5.4 Functions - Inputs and Outputs 8 / 19

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 `-2` and `true` without using a return statement.

Check Answer Show answer

Next

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Function {
5
6     function returnTwo() public pure returns (int a, bool b) {
7         a = -2;
8         b = true;
9     }
10 }
11 //Laksh Sodhai D20A 64
12
```

**Explain contract**

AI copilot

0 Listen on all transactions Filter with transaction hash or address

[call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c Debug

transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xc06...990f9 value: 0 wei data: 0x371...303c0 logs: 0 Debug

hash: 0x687...d5d37

transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xc06...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0 Debug

hash: 0x364...b4581

## • Tutorial no. 9

**LEARNETH** 1.5.1

learneth tutorials

laksh-59 Theme

visibility.sol

6. Visibility 9 / 19

- Can be called from other contracts or transactions
- State variables can not be `external`

In this example, we have two contracts, the `Base` contract (line 4) and the `Child` contract (line 55) which inherits the functions and state variables from the `Base` contract.

When you uncomment the `testPrivateFunc` (lines 58-60) 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.

```
60 //
61
62 // Internal function call be called inside child contracts.
63 function testInternalFunc() public pure override returns (string memory) {
64     return internalFunc();
65 }
66
67 function testInternalVar() public view returns (string memory, string memory) {
68     return (internalVar, publicVar);
69 }
70 //Laksh Sodhai D20A 64
71
```

**Explain contract**

AI copilot

0 Listen on all transactions Filter with transaction hash or address

[call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c Debug

transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xc06...990f9 value: 0 wei data: 0x371...303c0 logs: 0 Debug

hash: 0x687...d5d37

transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xc06...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0 Debug

hash: 0x364...b4581

## • Tutorial no.

10

**LEARNETH**

7.1 Control Flow - If/Else

**else if**

With the `else if` statement we can combine several conditions.

If the first condition (line 6) of the `foo` function is not met, but the condition of the `else if` statement (line 8) becomes true, the function returns `1`.

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.

```

17 // return 1;
18 // }
19 // return 2;
20
21 // shorthand way to write if / else statement
22 return _x < 10 ? 1 : 2;
23
24 }
25
26 function evenCheck(uint y) public pure returns (bool) {
27     return y % 2 == 0 ? true : false;
28 }
29 //Laksh Sodhai D20A 64

```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or address

call [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transaction to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

## • Tutorial no. 11

**LEARNETH**

7.2 Control Flow - Loops

**continue**

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

**break**

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

Watch a video tutorial on Loop statements.

**Assignment**

- Create a public `uint` state variable called `count` in the `Loop` contract.
- At the end of the for loop, increment the count variable by 1.
- 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.

```

15 }
16 break;
17 count++;
18 }
19
20 // while loop
21 uint j;
22 while (j < 10) {
23     j++;
24 }
25
26 //Laksh Sodhai D20A 64
27 }
28

```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or address

call [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990f9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transaction to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990f9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

- Tutorial no.

12

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**8.1 Data Structures - Arrays** 12 / 19

We can use the `delete` operator to remove an element with a specific index from an array (line 42). When we remove an element with the `delete` operator all other elements stay the same, which means that the length of the array will stay the same. This will create a gap in our array. If the order of the array is not important, then we can move the last element of the array to the place of the deleted element (line 46), or use a mapping. A mapping might be a better choice if we plan to remove elements in our data structure.

**Array length**

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

Watch a video tutorial on Arrays.

**Assignment**

1. Initialize a public fixed-sized array called `arr` with the values 0, 1, 2. Make the size as small as possible.
2. Change the `getArr()` function to return the value of `arr[3]`.

Check Answer Show answer

Next

Well done! No errors.

```

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   //Laksh Sodhai D20A 64
74 }

```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

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

transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0 hash: 0x687...d6d37 Debug

transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0 hash: 0x364...b4581 Debug

- Tutorial no. 13

**LEARNETH** 1.5.1 learneth tutorials laksh-59 Theme

**8.2 Data Structures - Mappings** 13 / 19

We set a new value for a key by providing the mapping's name and key in brackets and assigning it a new value (line 16).

**Removing values**

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

Watch a video tutorial on Mappings.

**Assignment**

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

Check Answer Show answer

Next

Well done! No errors.

```

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   //Laksh Sodhai D20A 64
47 }

```

**Explain contract** AI copilot

0 Listen on all transactions Filter with transaction hash or address

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

transact to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0 hash: 0x687...d6d37 Debug

transact to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0 hash: 0x364...b4581 Debug

## • Tutorial no.

14

The screenshot shows the REMIX IDE interface for Tutorial 14, titled "8.3 Data Structures - Structs". The left sidebar contains a "Tutorials list" and a "Syllabus" section. The main content area displays the tutorial text, which explains key-value mapping and how to access and update struct members using the dot operator. Below the text is an "Assignment" section with a "Check Answer" button and a "Show answer" button. The right sidebar shows the "Compile" tab with the following Solidity code:

```
40 }
41
42 // update completed
43 function toggleCompleted(uint _index) public { 28995 gas
44     Todo storage todo = todos[_index];
45     todo.completed = !todo.completed;
46 }
47
48 function remove(uint _index) public { infinite gas
49     delete todos[_index];
50 }
51 //Laksh Sodhai D20A 64
52 }
```

Below the code is the "Explain contract" section, which shows a list of transactions and their details, including the VM state and the gas used.

## • Tutorial no. 15

The screenshot shows the REMIX IDE interface for Tutorial 15, titled "8.4 Data Structures - Enums". The left sidebar contains a "Tutorials list" and a "Syllabus" section. The main content area displays the tutorial text, which explains how to update the enum value of a variable by assigning it the enum member (line 30). Below the text is an "Assignment" section with a "Check Answer" button and a "Show answer" button. The right sidebar shows the "Compile" tab with the following Solidity code:

```
36 }
37
38 // You can update to a specific enum like this
39 function cancel() public { 24494 gas
40     status = Status.Canceled;
41 }
42
43 // delete resets the enum to its first value, 0
44 function reset() public { 24383 gas
45     delete status;
46 }
47 //Laksh Sodhai D20A 64
48 }
```

Below the code is the "Explain contract" section, which shows a list of transactions and their details, including the VM state and the gas used.

- Tutorial no.

16

**LEARNETH** 1.5.1

learneth tutorials

laksh-59 Theme

**9. Data Locations** 16 / 19

affect the values stored in the mapping `myStruct` (line 10).

As we said in the beginning, when creating contracts we have to be mindful of gas costs. Therefore, we need to use data locations that require the lowest amount of gas possible.

**★ Assignment**

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

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

**Check Answer** **Show answer**

Next

Well done! No errors.

```

35
36 // You can return memory variables
37 function g(uint[] memory _arr) public returns (uint[] memory) {
38     // do something with memory array
39     _arr[0] = 1;
40 }
41
42 function h(uint[] calldata _arr) external {
43     // do something with calldata array
44     _arr[0] = 1;
45 }
46 //Laksh Sodhai D20A 64
47
48

```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or ad...

call [call] from: 0x58380da701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c Debug

transaction to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0 Debug

hash: 0x687...d6d37

transaction to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0 Debug

hash: 0x364...b4581

- Tutorial no. 17

**LEARNETH** 1.5.1

learneth tutorials

laksh-59 Theme

**10.1 Transactions - Ether and Wei** 17 / 19

**wei**

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

**gwei**

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

**ether**

One **ether** is equal to 1,000,000,000,000,000,000 ( $10^{18}$ ) **wei** (line 11).

Watch a video tutorial on Ether and Wei.

**★ Assignment**

1. Create a `public uint` called `oneGwei` and set it to 1 **gwei**.
2. Create a `public bool` called `isOneGwei` and set it to the result of a comparison operation between 1 **gwei** and  $10^9$ .

Tip: Look at how this is written for **gwei** and **ether** in the contract.

**Check Answer** **Show answer**

Next

Well done! No errors.

```

5
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 //Laksh Sodhai D20A 64

```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or ad...

call [call] from: 0x58380da701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c Debug

transaction to Counter.inc pending ...

[vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0 Debug

hash: 0x687...d6d37

transaction to Counter.dec pending ...

[vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0 Debug

hash: 0x364...b4581

- Tutorial no.

18

**LEARNETH**

**10.2 Transactions - Gas and Gas Price**

**Gas limit**

When sending a transaction, the sender specifies the maximum amount of gas that they are willing to pay for. If they set the limit too low, their transaction can run out of gas before being completed, reverting any changes being made. In this case, the gas was consumed and can't be refunded.

Learn more about gas on [ethereum.org](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.

```
// Using up all of the gas that you send causes your transaction to fail.
// State changes are undone.
// Gas spent are not refunded.
function forever() public {
    // Here we run a loop until all of the gas are spent
    // and the transaction fails
    while (true) {
        i += 1;
    }
}
//Laksh Sodhai D20A 64
```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or ad...

call [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ...

vm [vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transaction to Counter.dec pending ...

vm [vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

- Tutorial no. 19

**LEARNETH**

**10.3 Transactions - Sending Ether**

33 and 38) from `payable_address` to `address`, you won't be able to use `transfer()` (line 35) or `send()` (line 41).

Watch a video tutorial on Sending Ether.

**Assignment**

Build a charity contract that receives Ether that can be withdrawn by a beneficiary.

- Create a contract called `Charity`.
- Add a public state variable called `owner` of the type `address`.
- Create a donate function that is public and payable without any parameters or function code.
- 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.

```
owner = msg.sender;

function donate() public payable {}

function withdraw() public {
    uint amount = address(this).balance;
    (bool sent, bytes memory data) = owner.call(value: amount)("");
    require(sent, "Failed to send Ether");
}
//Laksh Sodhai D20A 64
```

**Explain contract**

0 Listen on all transactions Filter with transaction hash or ad...

call [call] from: 0x58380da6a701c568545dcfc803fc8875f56beddC4 to: Counter.get() data: 0x6d4...ce63c  
transaction to Counter.inc pending ...

vm [vm] from: 0x583...eddC4 to: Counter.inc() 0xcD6...990F9 value: 0 wei data: 0x371...303c0 logs: 0  
hash: 0x687...d6d37  
transaction to Counter.dec pending ...

vm [vm] from: 0x583...eddC4 to: Counter.dec() 0xcD6...990F9 value: 0 wei data: 0xb3b...cfa82 logs: 0  
hash: 0x364...b4581

**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.