

Experiment No. 4

Aim: Hands on Solidity Programming Assignments for creating Smart Contracts

Theory:

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

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

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

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

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

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

2. Inputs and Outputs to Functions

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

Example

```
function add(uint a, uint b) public pure returns (uint sum) {  
    sum = a + b;  
}  
  
function getValues() public view returns (uint, string memory) {  
    return (age, name);  
}
```

3. Visibility, Modifiers and Constructors

- **Function Visibility** defines who can access a function:
 - 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 contracts.
- **Modifiers** are reusable code blocks that change the behavior of functions. They are often used for access control, such as restricting sensitive functions to the contract owner (`onlyOwner`).

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

- **Constructors** are special functions executed only once during contract deployment. They initialize important values, such as setting the deploying account as the owner of the contract.

4. Control Flow: if-else, loops

Control flow in Solidity is similar to traditional programming languages:

- **if-else** allows conditional decision-making in contract logic, e.g., checking if a balance is sufficient before transferring funds.

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

- **Loops** (for, while, do-while) enable repeated execution of code. For example, iterating through an array of users. However, loops must be used carefully, as excessive iterations increase gas consumption, potentially making the contract expensive to execute.

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

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

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

6. Data Locations

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

1. Storage

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

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

2. Memory

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

Memory variables are deleted after the function execution is completed.

3. Calldata

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

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

Important Rules

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

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

Ether Units

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

Gas Concept

Gas represents the computational effort required to execute a transaction.

Gas Components:

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

Total Transaction Fee

Total Fee = Gas Used × Gas Price

- Total Fee = Gas Used × Gas Price

Important Global Variables

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

Sending Ether in Solidity

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

Example:

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

Explanation:

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

Implementation:

- Tutorial no. 1 – Compile the code

The screenshot shows the Remix Ethereum IDE interface. On the left, the Solidity Compiler sidebar is open, showing the version 0.8.31+commit.fd3a2265. The main workspace displays the source code for a Storage contract named 1_Storage.sol. The code defines a storage variable 'number' and two functions: 'store' and 'retrieve'. The 'store' function takes a uint256 parameter 'num' and stores it in the 'number' variable. The 'retrieve' function returns the current value of 'number'. The RemixAI assistant is visible on the right side of the interface.

```
// SPDX-License-Identifier: GPL-3.0
// sonam D20A 16
pragma solidity >=0.8.2 <0.9.0;

/*
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @custom:dev-run-script ./scripts/deploy_with_ETHER.ts
 */
contract Storage {
    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint256) {
        return number;
    }
}
```

- Tutorial no. 1 – Deploy the contract

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with options like 'DEPLOY & RUN TRANSACTIONS' and 'Deploy'. The main area displays the Solidity code for a 'Counter' contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

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;
    }
}
```

Below the code, there's an 'Explain contract' section with instructions and accessible libraries. At the bottom, a transaction log shows a successful deployment of the contract at address 0x0d91...39138.

- Tutorial no. 1 – get

The screenshot shows the Remix IDE interface after the contract has been deployed. The sidebar now shows the deployed contract at address 0x0d91...39138. The 'DEPLOY & RUN TRANSACTIONS' section includes buttons for 'dec', 'inc', 'count', and 'get'. The 'get' button is highlighted. The 'Logs' section shows three log entries from the 'get' function calls. The 'Explain contract' section remains the same as in the previous screenshot.

- Tutorial no. 1 – Increment

The screenshot shows the Remix Ethereum IDE interface. The code editor contains the following Solidity code:

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

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

The sidebar on the left shows the deployed contract "COUNTER AT 0xd91...39138" with its methods: dec, inc, count, get, and two low-level interactions: sonam and Tranact.

The "Explain contract" panel at the bottom provides transaction details for each method:

- dec: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c
- inc: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c
- count: [VM] from: 0x5B3...edd4 to: Counter.inc() 0xd91...39138 value: 0 wei data: 0x371...303c0 logs: 0 hash: e83b...27875
- get: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c

The status bar at the bottom indicates it's 26°C, Mostly sunny, and the date is 13-02-2026.

- Tutorial no. 1 – Decrement

The screenshot shows the Remix Ethereum IDE interface. The code editor contains the same Solidity code as the previous screenshot, but with a note in the comments of the dec function:

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

// Function to decrement count by 1
function dec() public {
    infinite gas
    count -= 1;
    PUSH1 costs 3 gas - this line costs 5218 gas - 9189 gas left
}
```

The sidebar on the left shows the deployed contract "COUNTER AT 0xd91...39138" with its methods: dec, inc, count, get, and two low-level interactions: sonam and Tranact.

The "Explain contract" panel at the bottom provides transaction details for each method, including the gas cost note for the dec function:

- dec: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c
- inc: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c
- count: [VM] from: 0x5B3...edd4 to: Counter.dec() 0xd91...39138 value: 0 wei data: 0xb3b...cf82 logs: 0 hash: e81f7...da31a
- get: [call] from: 0x5B380a6a701c568545dCfc803FcB875f56bedd4 to: Counter.get() data: 0x6d4...ce63c

The status bar at the bottom indicates it's 26°C, Mostly sunny, and the date is 13-02-2026.

- Tutorial no. 2

The screenshot shows the REMIX Ethereum IDE interface. The left sidebar displays the 'LEARNETH' navigation menu with the 'Tutorials list' selected, showing '2. Basic Syntax' (2 / 19). The main workspace shows the Solidity code for a contract named 'MyContract' with a single variable 'name' set to 'Alice'. Below the code, there is an 'Assignment' section with the following tasks:

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.

On the right, there is an 'Explain contract' panel with a search bar containing 'sonam'. The status bar at the bottom shows weather information (26°C, Mostly sunny), system icons, and the date/time (13-02-2026).

- Tutorial no. 3

The screenshot shows the REMIX Ethereum IDE interface. The left sidebar displays the 'LEARNETH' navigation menu with the 'Tutorials list' selected, showing '3. Primitive Data Types' (3 / 19). The main workspace shows the Solidity code for a contract with various primitive data types: int8, int256, int, address, uint8, and bool. Below the code, there is an 'Assignment' section with the following tasks:

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.

On the right, there is an 'Explain contract' panel with a search bar containing 'sonam'. The status bar at the bottom shows weather information (26°C, Mostly sunny), system icons, and the date/time (13-02-2026).

- Tutorial no. 4

The screenshot shows the Remix Ethereum IDE interface. The top navigation bar includes tabs for 'Experiment No. 4 Steps' and 'Remix - Ethereum IDE'. The main workspace displays a Solidity contract named 'Variables' with the following code:

```
pragma solidity ^0.8.3;

contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Hello";
    uint public num = 123;

    // 1. New public state variable
    uint public blockNumber;

    function doSomething() public {
        // Local variables are not saved to the blockchain.
        uint i = 456;

        // Global variables
        uint timestamp = block.timestamp;
        address sender = msg.sender;

        // 2. Assign current block number to state variable
        blockNumber = block.number;
    }
}
```

The sidebar on the left is titled 'LEARNETH' and shows a 'Tutorials list' for 'Variables' (4 / 19). It includes a tip about global variables and a section for 'Assignment' with two tasks:

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

A tip below suggests looking at the global variables section of the Solidity documentation for more information.

At the bottom of the sidebar, there are 'Check Answer', 'Show answer', and 'Next' buttons. A green bar at the bottom of the sidebar says 'Well done! No errors.'

- Tutorial no. 5

The screenshot shows the Remix Ethereum IDE interface. The top navigation bar includes tabs for 'Experiment No. 4 Steps' and 'Remix - Ethereum IDE'. The main workspace displays a Solidity contract named 'SimpleStorage' with the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract SimpleStorage {
    // State variable to store a number
    uint public num;

    // 1. New public bool variable initialized to true
    bool public b = true;

    // Write function
    function set(uint _num) public {
        num = _num;
    }

    // Read function
    function get() public view returns (uint) {
        return num;
    }

    // 2. Function to return value of b
    function get_b() public view returns (bool) {
        return b;
    }
}
```

The sidebar on the left is titled 'LEARNETH' and shows a 'Tutorials list' for 'Functions - Reading and Writing to a State' (5 / 19). It includes a tip about setting visibility and a section for 'Assignment' with two tasks:

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

A tip below suggests exploring the particularities of Solidity functions in more detail.

At the bottom of the sidebar, there are 'Check Answer', 'Show answer', and 'Next' buttons. A green bar at the bottom of the sidebar says 'Well done! No errors.'

At the very bottom of the screen, there are several status icons and a footer bar with the text 'Did you know? To learn new contract patterns and prototype, you can activate and try the cookbook plugin!' and 'RemixAI Copilot (enabled)'.

- Tutorial no. 6

The screenshot shows the REMIX Ethereum IDE interface. The left sidebar displays a navigation tree for 'LEARNETH' tutorials, with '2.6 Functions - View and Pure' selected. The main workspace shows the following Solidity code:

```

7 // Promise not to modify the state.
8 function addToX(uint y) public view returns (uint) {
9     return x + y;
10 }
11
12 // Promise not to modify or read from the state.
13 function add(uint i, uint j) public pure returns (uint) {
14     return i + j;
15 }
16
17 // New function that modifies state
18 function addToX2(uint y) public {
19     x = x + y;
20 }
21
22 sonam

```

The code implements three functions: `addToX` (view), `add` (pure), and `addToX2` (modifies state). A note in the code explains that `add` is a pure function that takes parameters `i` and `j` and returns their sum. It neither reads nor modifies the state variable `x`. A tip at the bottom suggests optimizing code for saving computation cost (gas cost) by declaring functions as view or pure.

Assignment:

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

Buttons: Check Answer, Show answer, Next.

Feedback: Well done! No errors.

System Status: Scan Alert, Initialize as git repo, 26°C Mostly sunny, AI copilot (enabled).

- Tutorial no. 7

The screenshot shows the REMIX Ethereum IDE interface. The left sidebar displays a navigation tree for 'LEARNETH' tutorials, with '5.3 Functions - Modifiers and Constructors' selected. The main workspace shows the following Solidity code:

```

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

```

The code demonstrates the use of modifiers like `noReentrancy` and `increase`. A note in the code explains that a constructor function is executed upon the creation of a contract and can have parameters and be especially useful when you don't know certain initialization values before deployment.

Assignment:

1. Create a new function, `increased`, 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 `increased` should be empty.

Tip: Use modifiers.

Buttons: Check Answer, Show answer, Next.

Feedback: Well done! No errors.

System Status: Scan Alert, Initialize as git repo, Olympic Games Today's updates, AI copilot (enabled).

● Tutorial no. 8

The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays the 'LEARNETH' tutorial series, specifically '5.4 Functions - Inputs and Outputs'. The main workspace contains the following Solidity code:

```
x = 1;
b = true;
y = 2;

function returnTwo()
public
pure
returns (int x, bool b)
{
    x = -2;
    b = true;
}

function destructingAssignments()
public
pure
returns (uint i, bool b, uint j) {
    (uint i, , uint j) = returnMany();
    (uint x, , uint y) = (4, 5, 6);
    return (i, b, j, x, y);
}

function arrayInput(uint[] memory _arr) public {}

uint[] public arr;

function arrayOutput() public view returns (uint[] memory) {
    return arr;
}
```

The code includes several comments explaining restrictions on input and output parameters. Below the code, there are 'Check Answer' and 'Show answer' buttons, and a message 'Well done! No errors.'

● Tutorial no. 9

The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays the 'LEARNETH' tutorial series, specifically '6. Visibility'. The main workspace contains the following Solidity code:

```
// External function
function externalFunc() external pure returns (string memory) {
    return "external function called";
}

// State variables
string private privateVar = "my private variable";
string internal internalVar = "my internal variable";
string public publicVar = "my public variable";

contract Child is Base {
    // Override internal function test
    function testInternalFunc() public pure override returns (string memory) {
        return internalFunc();
    }

    // New function added
    // Returns all Base state variables accessible in Child
    function testInternalVar() public view returns (string memory, string memory) {
        return (internalVar, publicVar);
    }
}
```

The code demonstrates visibility rules between contracts. Below the code, there are 'Check Answer' and 'Show answer' buttons, and a message 'Well done! No errors.'

● Tutorial no. 10

The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays a 'Tutorials list' for 'LEARNETH' under '7.1 Control Flow - If/Else'. It includes a 'Syllabus' section and a video thumbnail for '7.1 Control Flow - If/Else'. Below this, an 'Assignment' section asks to create a function 'evenCheck' that returns `true` for even numbers and `false` for odd numbers using a ternary operator. A tip notes that the modulo (%) operator produces the remainder of an integer division. At the bottom of the sidebar are 'Check Answer' and 'Show answer' buttons. The right side of the interface shows the Solidity code for the 'modifiersAndConstructors.sol' contract, which includes the following code:

```
if (x < 10) {
    return 0;
} else if (x > 10) {
    return 1;
} else {
    return 2;
}

function ternary(uint _x) public pure returns (uint) {
    return _x < 10 ? 1 : 2;
}

// New Function
function evenCheck(uint _x) public pure returns (bool) {
    return _x % 2 == 0 ? true : false;
}
```

The status bar at the bottom shows the date as 13-02-2026.

● Tutorial no. 11

The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays a 'Tutorials list' for 'LEARNETH' under '7.2 Control Flow - Loops'. It includes a 'Syllabus' section and a video thumbnail for '7.2 Control Flow - Loops'. Below this, an 'Assignment' section asks to create a public state variable 'count' in a 'Loop' contract, increment it in a for loop, and use a while loop to run 5 times. A tip notes that the `break` statement exits a loop. At the bottom of the sidebar are 'Check Answer' and 'Show answer' buttons. The right side of the interface shows the Solidity code for the 'Loop' contract, which includes the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Loop {
    // Public state variable
    uint public count;

    function loop() public {
        // For loop
        for (uint i = 0; i < 10; i++) {
            if (i == 3) {
                continue;
            }
            if (i == 5) {
                break;
            }
            count++; // increment count
        }

        // While loop
        uint j;
        while (j < 5) { // run 5 times
            count++;
            j++;
        }
    }
}
```

The status bar at the bottom shows the date as 13-02-2026.

● Tutorial no. 12

The screenshot shows the REMIX IDE interface. On the left, there's a sidebar with various icons and a search bar. The main area displays a Solidity contract named `arrays.sol` with the following code:

```

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

```

Below the code, there are two buttons: "Check Answer" and "Show answer". A green box at the bottom says "Well done! No errors." The status bar at the bottom right shows the date and time: 13-02-2026, 10:01.

● Tutorial no. 13

The screenshot shows the REMIX IDE interface. On the left, there's a sidebar with various icons and a search bar. The main area displays a Solidity contract named `mappings.sol` with the following code:

```

13
14 function set(address _addr) public {
15     // Update the value at this address
16     balances[_addr] = _addr.balance;
17 }
18
19 function remove(address _addr) public {
20     // Reset the value to the default value.
21     delete balances[_addr];
22 }
23
24 contract NestedMapping {
25     // Nested mapping (mapping from address to another mapping)
26     mapping(address => mapping(uint => bool)) public nested;
27
28     function get(address _addr1, uint _i) public view returns (bool) {
29         // You can get values from a nested mapping
30         // even when it is not initialized
31         return nested[_addr1][_i];
32     }
33
34     function set(
35         address _addr1,
36         uint _i,
37         bool _boo
38     ) public {
39         nested[_addr1][_i] = _boo;
40     }
41
42     function remove(address _addr1, uint _i) public {
43         delete nested[_addr1][_i];
44     }
45 }
46
47
48 sonam

```

Below the code, there are two buttons: "Check Answer" and "Show answer". A green box at the bottom says "Well done! No errors." The status bar at the bottom right shows the date and time: 13-02-2026, 10:01.

● Tutorial no. 14

The screenshot shows the REMIX Ethereum IDE interface. On the left, there's a sidebar with navigation links like 'Tutorials list' and 'Syllabus'. The main area displays a Solidity code editor with the following code:

```
function create(string memory _text) public {
    todos.push(Todo(_text, false));
    todos.push(Todo({_text: _text, completed: false}));
}

Todo memory todo;
todo.text = _text;
todos.push(todo);
}

function get(uint _index) public view returns (string memory text, bool completed) {
    Todo storage todo = todos[_index];
    return (todo.text, todo.completed);
}

function update(uint _index, string memory _text) public {
    Todo storage todo = todos[_index];
    todo.text = _text;
}

function toggleCompleted(uint _index) public {
    Todo storage todo = todos[_index];
    todo.completed = !todo.completed;
}

// New remove function
function remove(uint _index) public {
    delete todos[_index];
}
```

Below the code editor, there are two buttons: 'Check Answer' and 'Show answer'. A message at the bottom says 'Well done! No errors.'

● Tutorial no. 15

The screenshot shows the REMIX Ethereum IDE interface. On the left, there's a sidebar with navigation links like 'Tutorials list' and 'Syllabus'. The main area displays a Solidity code editor with the following code:

```
function set(Status _status) public {
    status = _status;
}

function cancel() public {
    status = Status.Canceled;
}

function reset() public {
    delete status;
}

// New Enum
enum Size {
    S,
    M,
    L
}

// Initialize variable of type Size
Size public sizes = Size.S;

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

Below the code editor, there are two buttons: 'Check Answer' and 'Show answer'. A message at the bottom says 'Well done! No errors.'

● Tutorial no. 16

The screenshot shows the REMIX IDE interface. The left sidebar displays the 'LEARNETH' syllabus with the current section being '9. Data Locations'. The main workspace contains the following Solidity code:

```
// Create memory copy from storage struct
MyStruct memory myMemStruct3 = myStruct;
myMemStruct3.foo = 3;

// Return all three
return (myStruct, myMemStruct2, myMemStruct3);

}

function f(
    uint[] storage _arr,
    mapping(uint => address) storage _map,
    MyStruct storage _myStruct
) internal {
    // do something with storage variables
}

function g(uint[] memory _arr) public pure returns (uint[] memory) {
    return _arr;
}

function h(uint[] calldata _arr) external
    // do something with calldata array
}

}

sonan |
```

Below the code, there are two buttons: 'Check Answer' and 'Show answer'. A message says 'Well done! No errors.' The status bar at the bottom indicates 'Scam Alert' and 'Initialize as git repo'.

● Tutorial no. 17

The screenshot shows the REMIX IDE interface. The left sidebar displays the 'LEARNETH' syllabus with the current section being '10.1 Transactions - Ether and Wei'. The main workspace contains the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract EtherUnits {
    uint public oneWei = 1 wei;
    bool public isOneWei = 1 wei == 1;

    uint public oneEther = 1 ether;
    bool public isOneEther = 1 ether == 1e18;

    // Must match test name exactly
    uint public oneGwei = 1 gwei;

    bool public isOneGwei = 1 gwei == 1e9;
}
```

Below the code, there are two buttons: 'Check Answer' and 'Show answer'. A message says 'Well done! No errors.' The status bar at the bottom indicates 'Scam Alert' and 'Initialize as git repo'.

- Tutorial no. 18

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons and a "LEARNETH" section containing a "Tutorials list" and a "Syllabus". The main workspace displays a Solidity code editor with the following code:

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

Below the code editor, there are two buttons: "Check Answer" and "Show answer". A green message bar at the bottom says "Well done! No errors.".

- Tutorial no. 19

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons and a "LEARNETH" section containing a "Tutorials list" and a "Syllabus". The main workspace displays a Solidity code editor with the following code:

```
5
6 // Owner of the contract
7 address public owner;
8
9 // Set owner when contract is deployed
10 constructor() {
11     owner = msg.sender;
12 }
13
14 // Function to receive donations
15 function donate() public payable {
16     // No code needed
17     // Ether is stored in contract balance
18 }
19
20 // Withdraw all Ether to owner
21 function withdraw() public {
22     require(msg.sender == owner, "Not owner");
23
24     uint amount = address(this).balance;
25
26     (bool sent,) = payable(owner).call{value: amount}("");
27     require(sent, "Failed to send Ether");
28 }
29
30 }
```

Below the code editor, there are two buttons: "Check Answer" and "Show answer". A green message bar at the bottom says "Well done! No errors.".

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.