

Blockchain Lab

Experiment 4

Mohit Patil
D20A 04

Aim - Hands on Solidity Programming Assignments for creating Smart Contracts.

Theory -

Primitive Data Types, Variables, and Functions (Pure & View)

In Solidity, primitive data types are the basic building blocks of smart contracts. Commonly used data types include:

- `uint` / `int` – Unsigned and signed integers of various sizes (e.g., `uint256`, `int128`).
- `bool` – Represents logical values (`true` or `false`).
- `address` – Stores a 20-byte Ethereum address, typically used for user accounts or contract addresses.
- `bytes` / `string` – Used to store binary data or textual information.

Types of Variables

Solidity supports different categories of variables:

- **State variables** – Stored permanently on the blockchain.
- **Local variables** – Temporary variables created during function execution.
- **Global variables** – Predefined variables such as `msg.sender`, `msg.value`, and `block.timestamp`.

Functions in Solidity

Functions define the logic of a smart contract. Two important function types are:

- **pure functions** – Cannot read or modify blockchain state. They rely only on input parameters and internal calculations.
- **view functions** – Can read state variables but cannot modify them.

Using `pure` and `view` appropriately helps reduce gas costs and ensures function integrity.

Inputs and Outputs of Functions

Solidity functions can accept input parameters and return one or more output values.

- **Inputs** allow users or other contracts to pass data into a function.
- **Outputs** return results after performing computations.

For example, a function may accept an Ether amount and return whether a transaction was successful. Solidity also supports **named return variables**, which enhance code readability and simplify debugging.

Visibility, Modifiers, and Constructors

Function Visibility

Visibility determines who can access a function:

- **public** – Accessible from inside and outside the contract.
- **private** – Accessible only within the same contract.
- **internal** – Accessible within the contract and its derived (child) contracts.
- **external** – Can be called only from outside the contract.

Modifiers

Modifiers are reusable code blocks that alter function behavior. They are commonly used for access control, such as restricting certain functions to the contract owner (e.g., `onlyOwner`).

Constructors

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

Control Flow: if-else and Loops

Solidity's control flow mechanisms are similar to traditional programming languages:

- **if-else statements** enable conditional execution, such as verifying sufficient balance before transferring funds.
- **Loops (`for`, `while`, `do-while`)** allow repeated execution of code, such as iterating through an array.

However, loops must be used cautiously because excessive iterations increase gas consumption, making transactions more expensive.

Data Structures: Arrays, Mappings, Structs, and Enums

Solidity provides several data structures:

- **Arrays** – Store ordered lists of elements. They can be fixed-size or dynamic. Example: an array of user addresses.
- **Mappings** – Store key-value pairs for efficient lookups. Example:
`mapping(address => uint)` for tracking account balances. Unlike arrays, mappings cannot be iterated directly.
- **Structs** – Custom data types that group related properties. Example:
`struct Player { string name; uint score; }`
- **Enums** – Define a set of predefined constants, improving readability. Example:
`enum Status { Pending, Active, Closed }`

Data Locations

Solidity uses three main data locations:

- **storage** – Permanently stored on the blockchain (used for state variables).
- **memory** – Temporary storage available only during function execution.
- **calldata** – Non-modifiable, non-persistent storage used for external function parameters. It is more gas-efficient than **memory**.

Understanding data locations is essential because they directly affect gas costs and contract performance.

Transactions: Ether, Wei, Gas, and Sending Transactions

Ether and Wei

Ether is the primary currency of Ethereum. All values are internally measured in **Wei**, the smallest unit:

1 Ether = 10^{18} Wei

Using Wei ensures high precision in financial calculations.

Gas and Gas Price

Every transaction consumes **gas**, which represents the computational effort required to execute it.

- **Gas price** determines how much Ether is paid per unit of gas.
- Higher gas prices encourage miners (validators) to prioritize a transaction.

Sending Transactions

Transactions are used to:

- Transfer Ether
- Interact with smart contracts

Common methods include:

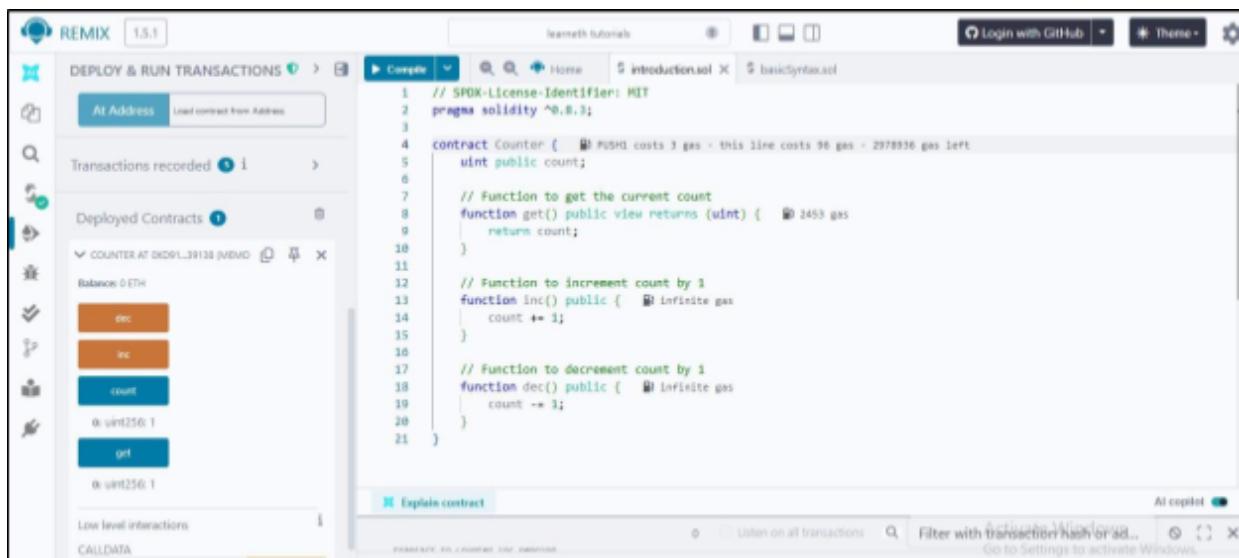
- `transfer()` – Sends Ether with fixed gas.
- `send()` – Similar to transfer but returns a boolean.
- `call()` – More flexible and commonly recommended for sending Ether.

Since every transaction requires gas, writing optimized and efficient smart contracts is crucial.

Code & Output -

Tutorial 1

1. Get counter value



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

contract Counter {
    // PUSHI costs 3 gas - this line costs 96 gas - 2978936 gas left
    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 {
        // infinite gas
        count += 1;
    }

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

2. Increment counter value

[vm] from: 0x5B3...eddC4 to: Counter.inc() 0xd91...39138 value: 0 wei data: 0x371...303c0 logs: 0 hash: 0xfc...8cdc3 Debug

status	1 Transaction mined and execution succeed
transaction hash	0xfc...8cdc3
block hash	0xc8733...50FeeFcd1145229228
block number	6
from	0x5B3...0da6a701c568545dCfcB03Fc8875f56beddC4
to	Counter.inc() 0xd91...39138
transaction cost	26417 gas

3. Decrement counter value

(vm) from: 0x583...eddC4	to: Counter.dec()	0xd91...39138	value: 0	wei data: 0xb3b...cfa82	logs: 0	hash: 0x21d...52d6d	Debug
status	1	Transaction mined and execution succeed					
transaction hash	0x21daa184e0a457ef2f35508a0094a8fc@c2eac05b53a095e6d96e1c2c4452d6d						
block hash	0xca425c3b6aa253d68e49726515232861814aF84154ab74aFe6c4683415457db8						
block number	7						
from	0x5B38Da6a781c560545dCfc803fc8875f56beddC4						
to	Counter.dec()	0xd9145CCE52D386f254917e481e844e9943f39138					
transaction cost	26461	gas					

4. Get count value

call	[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Counter.count() data: 0x066...61abd
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to	Counter.count() 0xd9145CCE52D386f254917e481eB44e9943F39138 ⓘ
execution cost	2402 gas (Cost only applies when called by a contract) ⓘ
input	0x066...61abd ⓘ
output	0x0005 ⓘ Copy
decoded input	{ } ⓘ
decoded output	[] ⓘ

Tutorial 2

The screenshot shows the REMIX IDE interface. On the left, there's a sidebar with icons for file operations and a list of deployed contracts: COUNTER, HELLOWORLD, HELLOWORLD AT 0x0000..., and MORTGAGE. Below that is a 'Low level interactions' section with a 'CALLDATA' tab and a 'Target' button. The main workspace has tabs for 'Compiled' (selected), 'Introduction.sol', 'basicSyntax.sol', and 'basicSyntax.test.sol'. The 'basicSyntax.sol' tab contains the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 // compiler version must be greater than or equal to 0.8.0 and less than 0.9.0
3 pragma solidity ^0.8.0;
4
5 contract MyContract {
6     string public name = "Alice";
7 }
```

Below the code, there's an 'Explain contract' section with tabs for 'Execution cost', 'Input', 'Output', 'Decoded Input', 'Decoded Output', and 'Logs'. The 'Execution cost' tab shows 3200 gas. The 'Input' tab shows the ABI input. The 'Output' tab shows the ABI output. The 'Decoded Input' and 'Decoded Output' tabs show the raw data as hex strings. The 'Logs' tab is empty.

Tutorial 3

The screenshot shows the REMIX IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists contracts: MYCONTRACT AT 0xE3E1...4EPP9 (MEMORY), PRIMITIVES AT 0x39C...24CT3 (MEMORY), and VARIABLES AT 0x6C3...CF142 (MEMORY). The VARIABLES contract has a balance of 0 ETH. It contains variables: doSomething (orange button), blockNumber (blue button), num (blue button), and test (blue button). The 'test' button is currently selected. The code editor shows the following Solidity code:

```

25 int public i = -123; // int is same as int256
26
27 address public addr = 0xCA35b7d91545BEF540a0e60800fFe2F4460fa733c;
28
29 // Default values
30 // Unassigned variables have a default value
31 bool public defaultBool; // false
32 uint public defaultUint; // 0
33 int public defaultInt; // 0
34 address public defaultAddress; // 0x0000000000000000000000000000000000000000
35 address public newAddr = 0x0000000000000000000000000000000000000000;
36 int public neg = -4;
37 uint8 public newU8;
38

```

The 'Explain contract' panel at the bottom shows transaction details for a call to the 'test' function.

Tutorial 4

The screenshot shows the REMIX IDE interface. The 'DEPLOY & RUN TRANSACTIONS' sidebar lists contracts: HELLOWORLD AT 0x0C06...990DF9 (MB), HELLOWORLD AT 0x4E3...86888 (ME), MYCONTRACT AT 0xE3E1...4EPP9 (MEM), PRIMITIVES AT 0x39C...24CT3 (MEMORY), and VARIABLES AT 0x6C3...CF142 (MEMORY). The VARIABLES contract has a balance of 0 ETH. It contains variables: doSomething (orange button), blockNumber (blue button), num (blue button), and test (blue button). The 'test' button is selected. The code editor shows the following Solidity code:

```

4 contract Variables {
5     // State variables are stored on the blockchain.
6     string public text = "Hello";
7     uint public num = 123;
8     uint public blockNumber;
9
10    function doSomething() public {
11        // Local variables are not saved to the blockchain.
12        uint _i = 456;
13
14        // Here are some global variables
15        uint timestamp = block.timestamp; // Current block timestamp
16        address sender = msg.sender; // address of the caller
17        blockNumber = block.number;
18    }
19

```

The 'Explain contract' panel at the bottom shows transaction details for calls to the 'num' and 'text' functions.

Tutorial 5

The screenshot shows the REMIX IDE interface. On the left, the "DEPLOY & RUN TRANSACTIONS" sidebar lists contracts: HELLOWORD, MYCONTRACT, PRIMITIVES, VARIABLES, and SIMPLESTORAGE. The SIMPLESTORAGE entry is expanded, showing a balance of 0 ETH and two functions: "set" and "get". The "set" function has a value of 60, and the "get" function returns 60. Below this is a "Low level interactions" section with a "CALLDATA" button and a "Transact" button.

The main area displays the Solidity code for the SimpleStorage contract:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract SimpleStorage {
5     // State variable to store a number
6
7     uint num;
8
9     function set(uint _num) public {
10        num = _num;
11    }
12
13    function get() public view returns (uint) {
14        return num;
15    }
16 }

```

Below the code, the "Explain contract" section shows a transaction log:

- tx: [CALL] From: 0x50380a6a701c568545dCfc00Fcb875f56bed0C4 To: simplestorage.set() Value: 0x00
- tx: [call] From: 0x50380a6a701c568545dCfc00Fcb875f56bed0C4 To: simplestorage.get() Value: 0x00

The status bar at the bottom right says "Activate Windows Go to Settings to activate Windows."

Tutorial 6

The screenshot shows the REMIX IDE interface. On the left, the "DEPLOY & RUN TRANSACTIONS" sidebar lists contracts: PRIMITIVES, VARIABLES, SIMPLESTORAGE, and VIEWANDPURE. The VIEWANDPURE entry is expanded, showing a balance of 0 ETH and two functions: "add" and "addXy". The "add" function has a value of 54, and the "addXy" function has a value of 10. Below this is a "Low level interactions" section with a "CALLDATA" button and a "Transact" button.

The main area displays the Solidity code for the ViewAndPure contract:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract ViewAndPure {
5     uint public x = 1;
6
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 }

```

Below the code, the "Explain contract" section shows a transaction log:

- tx: [call] From: 0x50380a6a701c568545dCfc00Fcb875f56bed0C4 To: ViewAndPure.x() Value: 0x00
- tx: [call] From: 0x50380a6a701c568545dCfc00Fcb875f56bed0C4 To: ViewAndPure.x() Value: 0x00

The status bar at the bottom right says "Activate Windows Go to Settings to activate Windows."

Tutorial 7

```

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

contract FunctionModifier {
    // We will use these variables to demonstrate how to use
    // modifiers.
    address public owner;
    uint public x = 10;
    bool public locked;

    constructor() {
        // Set the transaction sender as the owner of the contract.
        owner = msg.sender;
    }

    // Modifier to check that the caller is the owner of
    // the contract.
    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }
}

function changeOwner(address _newOwner) external onlyOwner {
    owner = _newOwner;
}

function decrement() external {
    x--;
}

```

Tutorial 8

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
// Pranav Titambu #2ba/0d
contract Function {
    // Functions can return multiple values.
    function returnMany() {
        return [1, true, 2];
    }

    // Return values can be named.
    function named() {
        public
        pure
        returns (uint, bool, uint)
    } {
        return (1, true, 2);
    }

    // Return values can be assigned to their name.
    // In this case the return statement can be omitted.
}

```

Tutorial 9

6. Visibility

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

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

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
- Can be called from a child contract
- Can be called from other contracts or transactions

external

- 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 95) 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 `getBaseValue` and `getInternalValue` directly. You will only be able to call them via `[base]testFunction` and `[child]testFunction`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Pranav Titambe d2Ba788

contract Base {
    // Private function can only be called
    // - inside this contract
    // Contracts that inherit this contract cannot call this function.
    function privateFunc() private pure returns (string memory) { // infinite gas
        return "private function called";
    }

    function testPrivateFunc() public pure returns (string memory) { // infinite gas
        return privateFunc();
    }
}

contract Child is Base {
    // Internal function can be called
    // - inside this contract
    // - inside contracts that inherit this contract
    function internalFunc() internal pure returns (string memory) { // infinite gas
        return "internal function called";
    }

    function testInternalFunc() public pure virtual returns (string memory) { // infinite gas
        return internalFunc();
    }
}

// Public functions can be called
// - inside this contract
// - inside contracts that inherit this contract
// - by other contracts and accounts

```

Explain contract

(vm) From: 0x503...ed04 To: Array.(constructor) Value: 0 Wei Data: 0x00...f0000 Logs: 0 Hash: 0x0

Tutorial 10

7.1 Control Flow - If/Else

Solidity supports different control flow statements that determine which parts of the contract will be executed. The conditional `if/else` statement enables contracts to make decisions depending on whether boolean conditions are either `true` or `false`.

Solidity differentiates between three different `If/Else` statements: `if`, `else`, and `else if`.

if

The `if` statement is the most basic statement that allows the contract to perform an action based on a boolean expression.

In this contracts `foo` function (line 5) the `if` statement (line 6) checks if `x` is smaller than `10`. If the statement is true, the function returns `0`.

else

The `else` statement enables our contract to perform an action if conditions are not met.

In this contract, the `foo` function uses the `else` statement (line 10) to return `2` if none of the other conditions are met.

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` statements.

Assignment

Create a new function called `isEven` in the `Base` 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.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Pranav Titambe d2Ba788

contract IfElse {
    function foo(uint x) public pure returns (uint) { // infinite gas
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

    function ternary(uint _x) public pure returns (uint) { // infinite gas
        // if (_x < 20) {
        //     return 1;
        // }
        // return 2;
        // shorthand way to write if / else statement
        return _x < 10 ? 1 : 2;
    }
}
```

Explain contract

(vm) From: 0x503...ed04 To: Array.(constructor) Value: 0 Wei Data: 0x00...f0000 Logs: 0 Hash: 0x0

Tutorial 11

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
// Author: Pronav Titame d2ba/4d

contract Loop {
    function loop() public {
        // for loop
        for (uint i = 0; i < 10; ++i) {
            if (i == 3) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
        }
    }
}

```

Tutorial 12

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
// Author: Pronav Titame d2ba/4d

contract Array {
    // Several ways to initialize an array
    uint[] public arr;
    uint[] public arr2 = [1, 2, 3];
    // Fixed sized array, all elements initialize to 0
    uint[10] public myConditionArr;

    function get(uint i) public view returns (uint) {
        // Solidity can return the entire array.
        // But this function should be avoided for
        // arrays that can grow indefinitely in length.
        return arr[i];
    }

    function push(uint i) public {
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {
        // Remove last element from array
        // This will decrease the array length by 1.
        arr.pop();
    }
}

```

Tutorial 13

The screenshot shows the REMIX IDE interface. The left sidebar displays a navigation tree under 'LEARNETH' with sections like 'Tutorials list', '8.2 Data Structures - Mappings', and '8.3 Data Structures - Structs'. The main content area shows the Solidity code for 'mappings.sol'.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 // Pranav Titambe d20a/68
5
6 contract Mapping {
7     // Mapping from address to user
8     mapping(address => uint) public myMap;
9
10    function get(address _addr) public view returns (uint) {    // 2872 gas
11        // Mapping always returns a value.
12        // If the value was never set, it will return the default value.
13        return myMap[_addr];
14    }
15
16    function set(address _addr, uint _i) public {    // 32842 gas
17        // Update the value at this address
18        myMap[_addr] = _i;
19    }
20
21    function remove(address _addr) public {    // 3544 gas
22        // Reset the value to the default value.
23        delete myMap[_addr];
24    }
25
26}
27
28 contract NestedMapping {
29     // Nested mapping (mapping from address to another mapping)
30     mapping(address => mapping(uint => bool)) public nested;
31
32     function get(address _addr, uint _i) public view returns (bool) {    // 3139 gas
33         // You can get values from a nested mapping
34     }
35
36}

```

Explains contract

(vm) From: 0x00...addC4 To: Array.(constructor) Value: 0 Wei Data: 0x00...F0003 Logs: 0 Hash:

Tutorial 14

The screenshot shows the REMIX IDE interface. The left sidebar displays a navigation tree under 'LEARNETH' with sections like 'Tutorials list', '8.2 Data Structures - Mappings', and '8.3 Data Structures - Structs'. The main content area shows the Solidity code for 'structs.sol'.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 // Pranav Titambe d20a/68
5
6 contract Todos {
7     struct Todo {
8         string text;
9         bool completed;
10    }
11
12    // An array of 'Todo' structs
13    Todo[] public todos;
14
15    function create(string memory _text) public {    // infinite gas
16        // 3 ways to initialize a struct
17        // - calling it like a function
18        todos.push(Todo(_text, false));
19
20        // key value mapping
21        todos.push(Todo({_text, completed: false}));
22
23        // initialize an empty struct and then update it
24        Todo memory todo;
25        todo.text = _text;
26        // todo.completed initialized to false
27
28        todos.push(todo);
29    }
30
31    // Solidity automatically created a getter for 'todos' so
32    // you don't actually need this function.
33
34}
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
123
124
125
126
127
127
128
129
129
130
131
132
133
133
134
135
135
136
136
137
137
138
138
139
139
140
140
141
141
142
142
143
143
144
144
145
145
146
146
147
147
148
148
149
149
150
150
151
151
152
152
153
153
154
154
155
155
156
156
157
157
158
158
159
159
160
160
161
161
162
162
163
163
164
164
165
165
166
166
167
167
168
168
169
169
170
170
171
171
172
172
173
173
174
174
175
175
176
176
177
177
178
178
179
179
180
180
181
181
182
182
183
183
184
184
185
185
186
186
187
187
188
188
189
189
190
190
191
191
192
192
193
193
194
194
195
195
196
196
197
197
198
198
199
199
200
200
201
201
202
202
203
203
204
204
205
205
206
206
207
207
208
208
209
209
210
210
211
211
212
212
213
213
214
214
215
215
216
216
217
217
218
218
219
219
220
220
221
221
222
222
223
223
224
224
225
225
226
226
227
227
228
228
229
229
230
230
231
231
232
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
240
241
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
```

```

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

// Pranav Titambe d20a/6d

contract Enum {
    // Enum representing shipping status
    enum Status {
        Pending,
        Shipped,
        Accepted,
        Rejected,
        Canceled
    }

    // Default value is the first element listed in
    // definition of the type, in this case "Pending"
    Status public status;

    // Returns uint
    // Pending - 0
    // Shipped - 1
    // Accepted - 2
    // Rejected - 3
    // Canceled - 4
    function get() public view returns (Status) {
        return status;
    }

    // Update status by passing uint into input
    function set(Status _status) public {
        status = _status;
    }
}

```

Explain contract

(vm) From: 0x5B...add4 To: Array.(constructor) Value: 0 Wei Data: 0x60...f0033

Tutorial 16

```

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

// Pranav Titambe d20a/6d

contract Datalocations {
    uint[] public arr;
    mapping(uint => address) map;
    struct MyStruct {
        uint foo;
    }
    mapping(uint => MyStruct) myStructs;

    function f() public {
        // call _f with state variables
        _f(arr, map, myStructs[1]);
    }

    // get a struct from a mapping
    MyStruct storage myStruct = myStructs[1];
    // create a struct in memory
    MyStruct memory myStruct = MyStruct();
}

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

// You can return memory variables

```

Explain contract

(vm) From: 0x5B...add4 To: Array.(constructor) Value: 0 Wei Data: 0x60...f0033

Tutorial 17

The screenshot shows the REMIX IDE interface with the following details:

- Title:** LEARNETH
- Tutorial Section:** 10.1 Transactions - Ether and Wei
- Code Editor:**

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

// Pranav Titambe d20a/68

contract EtherUnits {
    uint public oneWei = 1 wei;
    // 1 wei is equal to 1
    bool public isOneWei = 1 wei == 1;

    uint public oneEther = 1 ether;
    // 1 ether is equal to 10^18 wei
    bool public isOneEther = 1 ether == 1e18;
}
```
- Contract Explorer:** Shows a transaction log:
 - [vm] from: 0x5B3...add4 to: Array.(constructor) value: 0 wei data: 0x00...f003

Tutorial 18

The screenshot shows the REMIX IDE interface with the following details:

- Title:** LEARNETH
- Tutorial Section:** 10.2 Transactions - Gas and Gas Price
- Code Editor:**

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

// Pranav Titambe d20a/68

contract Gas {
    uint public i = 0;

    // Using up all of the gas that you send causes your transaction
    // 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;
        }
    }
}
```
- Contract Explorer:** Shows a transaction log:
 - [vm] from: 0x5B3...add4 to: Array.(constructor) value: 0 wei data: 0x00...f003

Tutorial 19

```

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

// Pranav Titambe d20a/6d

contract ReceiveEther {
    /*
        Which function is called?
        contract ReceiveEther is ReceiveEther
        send Ether
        msg.data is empty?
        yes no
        receive() exists? fallback()
        yes no
        receive() fallback()
    */

    // Function to receive Ether. msg.data must be empty
    receive() external payable {} // undefined gas

    // Fallback function is called when msg.data is not empty
    fallback() external payable {} // undefined gas

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}

```

Conclusion:

This experiment provided an in-depth exploration of Solidity programming through structured practical implementation using the Remix IDE. Fundamental concepts—including data types, variable classifications, function definitions, visibility specifiers, modifiers, constructors, control flow mechanisms, data structures, and transaction management—were systematically implemented and evaluated.

The process of designing, compiling, and deploying smart contracts on the Remix Virtual Machine (VM) enabled a comprehensive understanding of smart contract architecture and blockchain execution mechanisms. The practical exposure strengthened conceptual clarity and technical proficiency in Ethereum-based development.