

Experiment No. 4

- **Aim:** Hands on Solidity Programming Assignments for creating Smart Contracts
- **Lab Objectives:** To explore Blockchain concepts.
- **Lab Outcomes (LO):** Design Smart Contract using Solidity (L02)

- **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

- **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.
- **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.
 - external: can be called only by external accounts or other contracts

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.
- **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

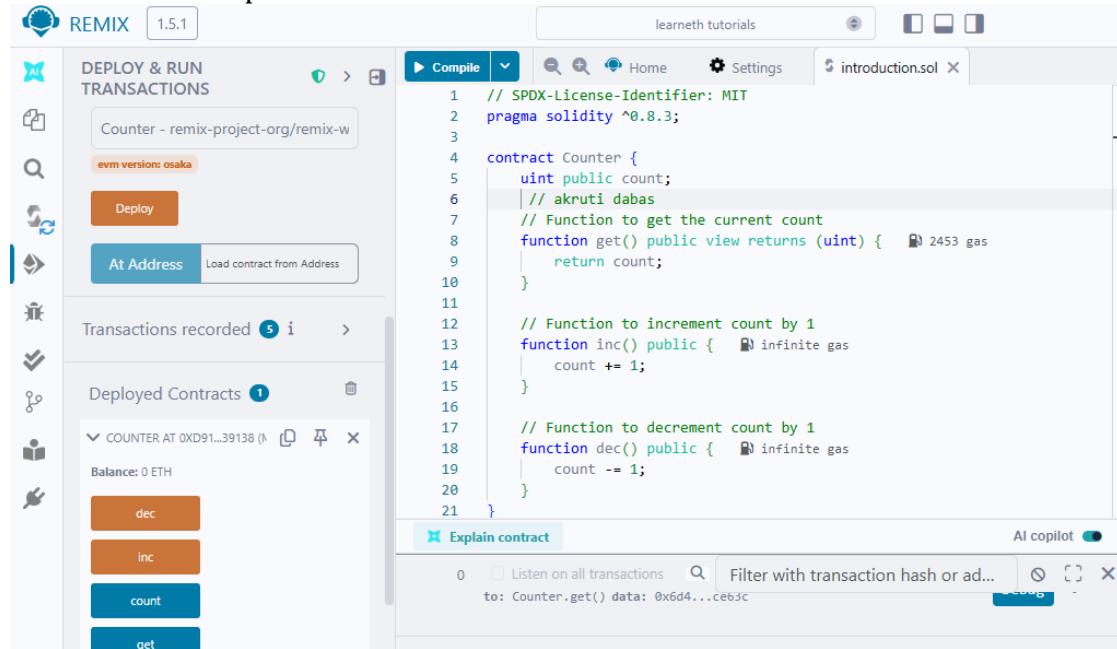
performance.

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

➤ Implementation:

- Tutorial no. 1 – Compile the code

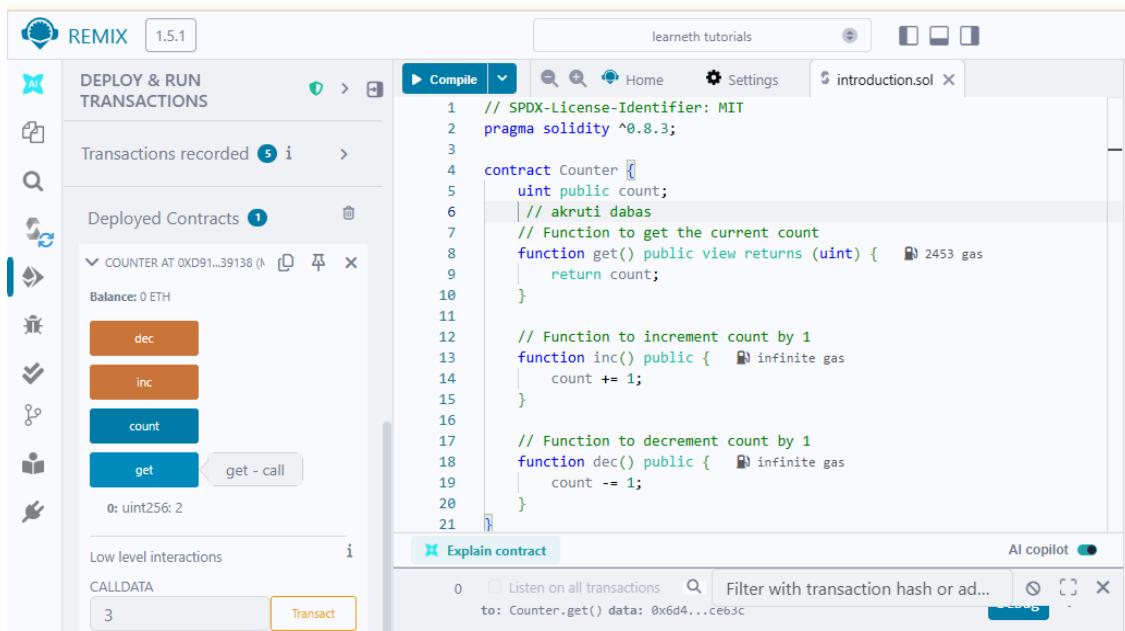


The screenshot shows the REMIX IDE interface. On the left, the sidebar has sections for 'DEPLOY & RUN TRANSACTIONS' and 'Deployed Contracts'. Under 'Deployed Contracts', there is a deployed contract named 'COUNTER AT 0xD91...39138' with a balance of 0 ETH. It has four functions: 'dec', 'inc', 'count', and 'get'. On the right, the main panel shows the Solidity code for the Counter contract. The code includes SPDX-License-Identifier: MIT, pragma solidity ^0.8.3, and three functions: get(), inc(), and dec(). The 'Explain contract' section at the bottom provides details about the get() function.

```

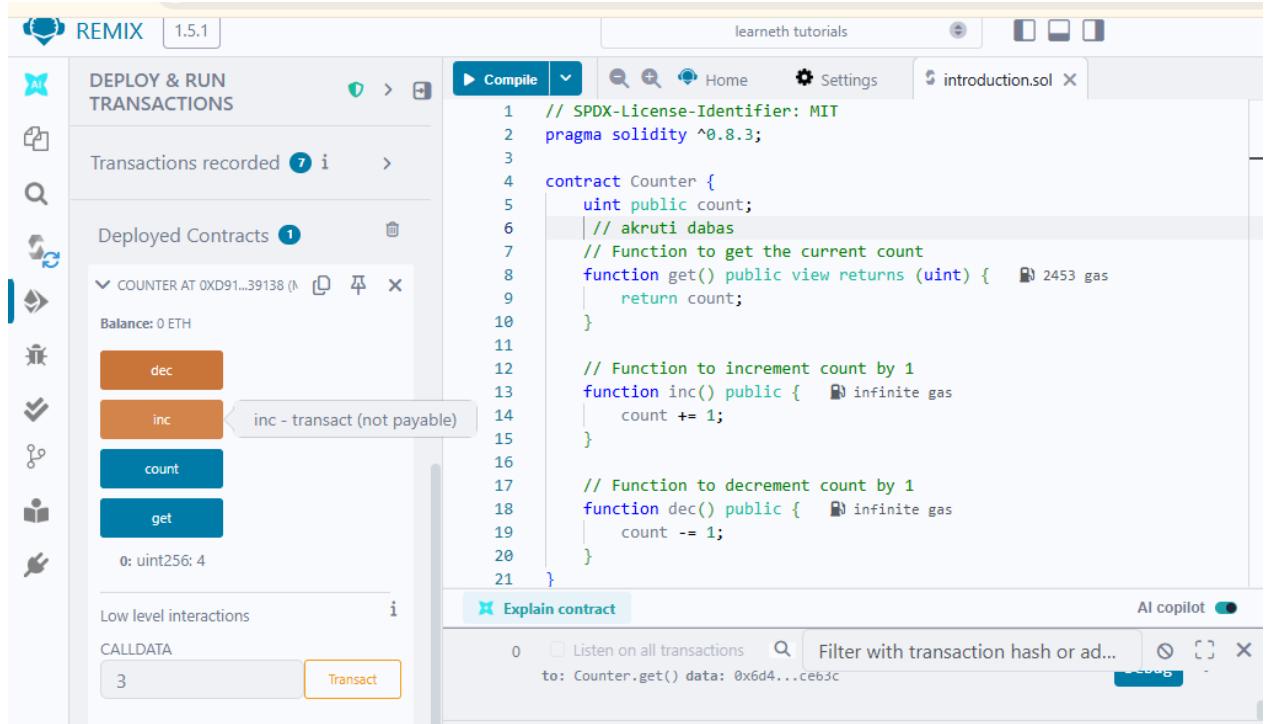
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Counter {
5     uint public count;
6     // akruti dabas
7     // Function to get the current count
8     function get() public view returns (uint) { 2453 gas
9         return count;
10    }
11
12    // Function to increment count 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 }
```

- Tutorial no. 1 – get



The screenshot shows the REMIX IDE interface. The 'get' button under 'Deployed Contracts' is highlighted. A tooltip 'get - call' appears over it. Below the button, the result '0: uint256: 2' is displayed. At the bottom, the 'CALLDATA' field contains the value '3' and the 'Transact' button is highlighted. The rest of the interface is similar to the previous screenshot, showing the deployed contract and its functions.

- Tutorial no. 1 – Increment



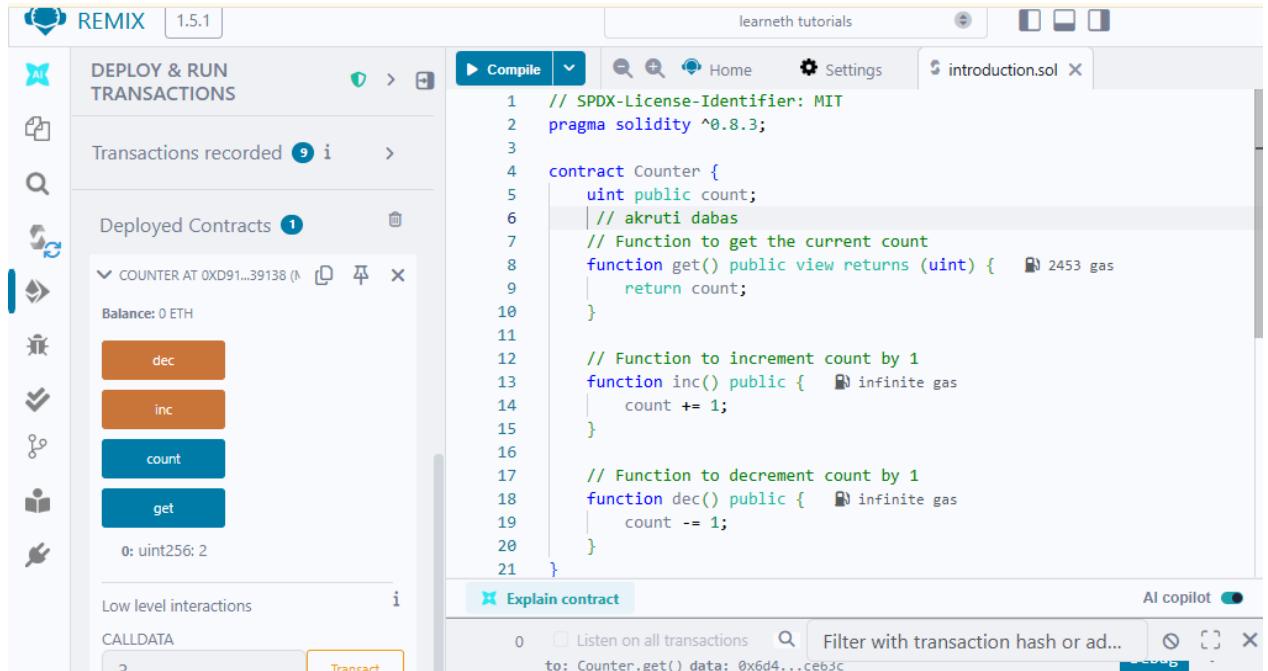
The screenshot shows the REMIX IDE interface with a Solidity contract named `introduction.sol`. The code defines a `Counter` contract with four functions: `dec`, `inc`, `count`, and `get`. The `inc` function is highlighted as the current transaction being interacted with. The `Deployed Contracts` section shows a deployed instance of the `Counter` contract with a balance of 0 ETH. The `Low level interactions` section shows a `CALldata` input of 3 and a `Transact` button.

```

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

```

- Tutorial no. 1 – Decrement



This screenshot is identical to the previous one, showing the REMIX IDE interface with the `introduction.sol` contract. The `inc` function is still highlighted as the transaction being interacted with. The `Deployed Contracts` section shows the same deployed `Counter` contract with a balance of 0 ETH. The `Low level interactions` section shows a `CALldata` input of 3 and a `Transact` button.

```

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

```

Akruti Dabas D20A 18

● Tutorial no. 2

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, currently at '2. Basic Syntax'. A list of tasks is shown:

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.

Below the tasks are buttons for 'Check Answer', 'Show answer', 'Next', and a green 'Well done! No errors.' message.

The main workspace shows a Solidity code editor with the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract MyContract {
    string public name = "Alice";
}
```

The status bar at the bottom right indicates 'AI copilot' is active.

● Tutorial no. 3

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, currently at '3. Primitive Data Types'. A list of tasks is shown:

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`.

A tip is provided: 'Tip: Look at the other address in the contract or search the internet for an Ethereum address.'

Below the tasks are buttons for 'Check Answer', 'Show answer', 'Next', and a green 'Well done! No errors.' message.

The main workspace shows a Solidity code editor with the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//AKRUTI DABAS
contract Primitives {
    bool public boo = true;

    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available
        uint8 ranges from 0 to 2 ** 8 - 1
        uint16 ranges from 0 to 2 ** 16 - 1
        ...
        uint256 ranges from 0 to 2 ** 256 - 1
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256

    /*
    Negative numbers are allowed for int types.
    Like uint, different ranges are available from int8 to int256
    
```

The status bar at the bottom right indicates 'AI copilot' is active.

Akruti Dabas D20A 18

● Tutorial no. 4

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, currently at '4. Variables' (4/19). The main workspace shows a Solidity contract named 'Variables'. The code includes comments explaining state variables, local variables, and global variables like timestamp and sender. The code block is as follows:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Hello";
    uint public num = 123;
    uint public blockNumber;

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

        // Here are some global variables
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller

        blockNumber = block.number;
    }
}
```

Below the code, there are buttons for 'Check Answer', 'Show answer', and 'Next'. A green message bar at the bottom says 'Well done! No errors.'

● Tutorial no. 5

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, currently at '5.1 Functions - Reading and Writing to a State Variable' (5/19). The main workspace shows a Solidity contract named 'SimpleStorage'. The code defines a state variable 'num' and a function 'set' to update it. It also includes a view function 'get' to read the value. The code block is as follows:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//AKRUTI DABAS
contract SimpleStorage {
    // State variable to store a number
    uint public num;

    // State variable to store a boolean
    bool public b = true;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {
        return num;
    }

    // Function to get the value of b
}
```

Below the code, there are buttons for 'Check Answer', 'Show answer', and 'Next'. A green message bar at the bottom says 'Well done! No errors.'

Akruti Dabas D20A 18

● Tutorial no. 6

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' course navigation, including 'Tutorials list', 'Syllabus', and a section titled '5.2 Functions - View and Pure' which states: 'have any side effects and will always return the same result if you pass the same arguments.' Below this is an 'Assignment' section with instructions to create a function called `addToX2` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`. On the right, the main workspace shows the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//AKRUTI DABAS
contract ViewAndPure {
    uint public x = 1;

    // Promise not to modify the state.
    function addToX(uint y) public view returns (uint) {
        return x + y;
    }

    // Promise not to modify or read from the state.
    function add(uint i, uint j) public pure returns (uint) {
        return i + j;
    }

    function addToX2(uint y) public returns (uint) {
        x += y;
        return x;
    }
}
```

Below the code, there are buttons for 'Check Answer', 'Show answer', 'Next', and a green 'Well done! No errors.' message.

● Tutorial no. 7

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' course navigation, including 'Tutorials list', 'syllabus', and a section titled '5.3 Functions - Modifiers and Constructors' which states: 'or the owner variable upon the creation of the contract.' Below this is an 'Assignment' section with instructions to:

- Create a new function, `increaser`, 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.
- Make sure that `x` can only be increased.
- The body of the function `increaser` should be empty.

Tip: Use modifiers.

On the right, the main workspace shows the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
// akruti dabas
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");
        // Underscore is a special character only used inside
        // a function modifier and it tells Solidity to
    }
}
```

Below the code, there are buttons for 'Check Answer', 'Show answer', 'Next', and a green 'Well done! No errors.' message. A status bar at the bottom right indicates 'Activate Windows' and 'Go to Settings to activate Windows.'

- Tutorial no. 8

```

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

```

- Tutorial no. 9

```

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

```

Akruti Dabas D20A 18

● Tutorial no. 10

The screenshot shows the REMIX IDE interface. On the left, the 'Tutorials list' shows '7.1 Control Flow - If/Else' as the current tutorial. The main area displays the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract IfElse {
    function foo(uint x) public pure returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

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

    function evenCheck(uint x) public pure returns (bool) {
        // infinite gas
    }
}
```

The 'Compile' dropdown is set to '1.5.1'. Below the code, there's an 'Explain contract' button and an AI copilot toggle. At the bottom, there are buttons for 'Check Answer', 'Show answer', 'Next', and a green message 'Well done! No errors.'

● Tutorial no. 11

The screenshot shows the REMIX IDE interface. On the left, the 'Tutorials list' shows '7.2 Control Flow - Loops' as the current tutorial. The main area displays the following Solidity code:

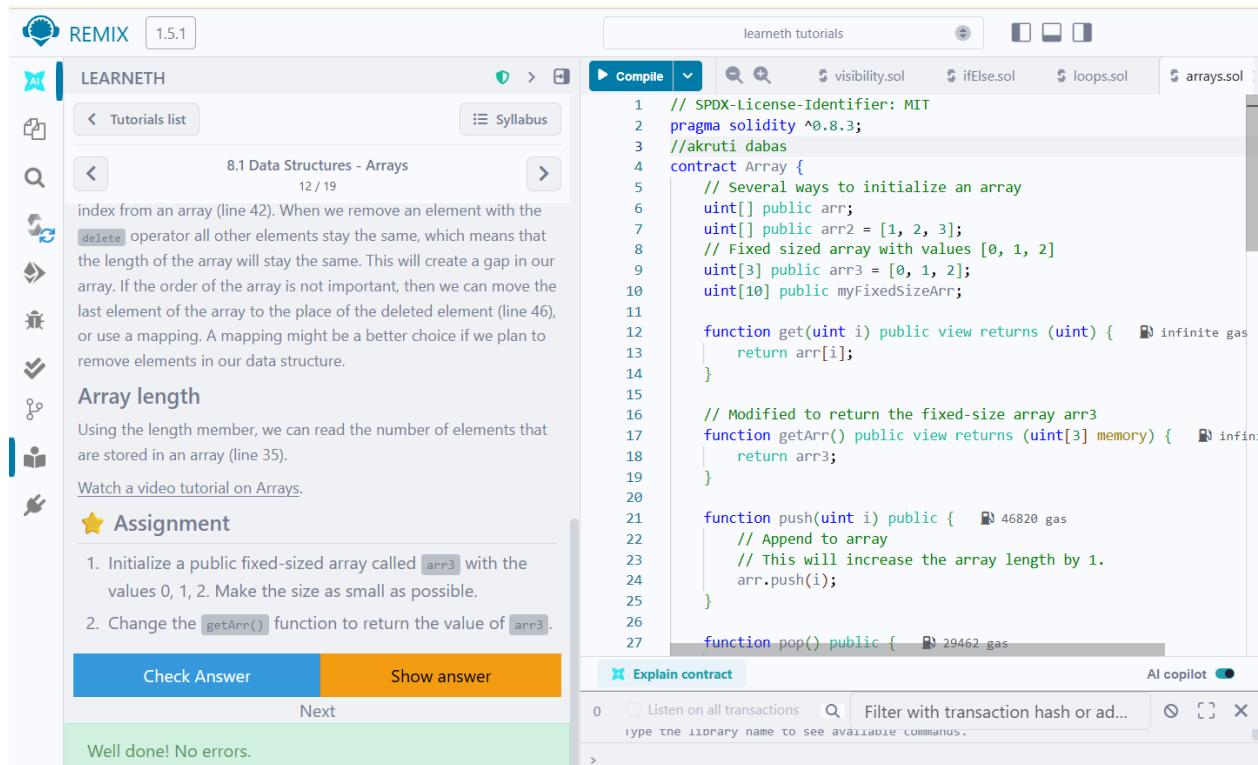
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract Loop {
    uint public count;

    function loop() public {
        // for loop
        for (uint i = 0; i < 9; i++) {
            count += 1;
        }

        // while loop
        uint j = 0; // Initialize j
        while (j < 10) {
            j++;
        }
    }
}
```

The 'Compile' dropdown is set to '1.5.1'. Below the code, there's an 'Explain contract' button and an AI copilot toggle. At the bottom, there are buttons for 'Check Answer', 'Show answer', 'Next', and a green message 'Well done! No errors.'

● Tutorial no. 12



The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' tutorial sidebar for '8.1 Data Structures - Arrays' is visible, showing a brief text about arrays and their initialization. The main workspace on the right displays the following Solidity code:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //akruti dabas
4 contract Array {
5     // Several ways to initialize an array
6     uint[] public arr;
7     uint[] public arr2 = [1, 2, 3];
8     // Fixed sized array with values [0, 1, 2]
9     uint[3] public arr3 = [0, 1, 2];
10    uint[10] public myFixedSizeArr;
11
12    function get(uint i) public view returns (uint) {
13        return arr[i];
14    }
15
16    // Modified to return the fixed-size array arr3
17    function getArr() public view returns (uint[3] memory) {
18        return arr3;
19    }
20
21    function push(uint i) public {
22        // Append to array
23        // This will increase the array length by 1.
24        arr.push(i);
25    }
26
27    function pop() public {
28
29
30
31
32
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
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
238
239
239
239
240
240
240
241
241
241
242
242
242
243
243
243
244
244
244
245
245
245
246
246
246
247
247
247
248
248
248
249
249
249
250
250
250
251
251
251
252
252
252
253
253
253
254
254
254
255
255
255
256
256
256
257
257
257
258
258
258
259
259
259
260
260
260
261
261
261
262
262
262
263
263
263
264
264
264
265
265
265
266
266
266
267
267
267
268
268
268
269
269
269
270
270
270
271
271
271
272
272
272
273
273
273
274
274
274
275
275
275
276
276
276
277
277
277
278
278
278
279
279
279
280
280
280
281
281
281
282
282
282
283
283
283
284
284
284
285
285
285
286
286
286
287
287
287
288
288
288
289
289
289
290
290
290
291
291
291
292
292
292
293
293
293
294
294
294
295
295
295
296
296
296
297
297
297
298
298
298
299
299
299
300
300
300
301
301
301
302
302
302
303
303
303
304
304
304
305
305
305
306
306
306
307
307
307
308
308
308
309
309
309
310
310
310
311
311
311
312
312
312
313
313
313
314
314
314
315
315
315
316
316
316
317
317
317
318
318
318
319
319
319
320
320
320
321
321
321
322
322
322
323
323
323
324
324
324
325
325
325
326
326
326
327
327
327
328
328
328
329
329
329
330
330
330
331
331
331
332
332
332
333
333
333
334
334
334
335
335
335
336
336
336
337
337
337
338
338
338
339
339
339
340
340
340
341
341
341
342
342
342
343
343
343
344
344
344
345
345
345
346
346
346
347
347
347
348
348
348
349
349
349
350
350
350
351
351
351
352
352
352
353
353
353
354
354
354
355
355
355
356
356
356
357
357
357
358
358
358
359
359
359
360
360
360
361
361
361
362
362
362
363
363
363
364
364
364
365
365
365
366
366
366
367
367
367
368
368
368
369
369
369
370
370
370
371
371
371
372
372
372
373
373
373
374
374
374
375
375
375
376
376
376
377
377
377
378
378
378
379
379
379
380
380
380
381
381
381
382
382
382
383
383
383
384
384
384
385
385
385
386
386
386
387
387
387
388
388
388
389
389
389
390
390
390
391
391
391
392
392
392
393
393
393
394
394
394
395
395
395
396
396
396
397
397
397
398
398
398
399
399
399
400
400
400
401
401
401
402
402
402
403
403
403
404
404
404
405
405
405
406
406
406
407
407
407
408
408
408
409
409
409
410
410
410
411
411
411
412
412
412
413
413
413
414
414
414
415
415
415
416
416
416
417
417
417
418
418
418
419
419
419
420
420
420
421
421
421
422
422
422
423
423
423
424
424
424
425
425
425
426
426
426
427
427
427
428
428
428
429
429
429
430
430
430
431
431
431
432
432
432
433
433
433
434
434
434
435
435
435
436
436
436
437
437
437
438
438
438
439
439
439
440
440
440
441
441
441
442
442
442
443
443
443
444
444
444
445
445
445
446
446
446
447
447
447
448
448
448
449
449
449
450
450
450
451
451
451
452
452
452
453
453
453
454
454
454
455
455
455
456
456
456
457
457
457
458
458
458
459
459
459
460
460
460
461
461
461
462
462
462
463
463
463
464
464
464
465
465
465
466
466
466
467
467
467
468
468
468
469
469
469
470
470
470
471
471
471
472
472
472
473
473
473
474
474
474
475
475
475
476
476
476
477
477
477
478
478
478
479
479
479
480
480
480
481
481
481
482
482
482
483
483
483
484
484
484
485
485
485
486
486
486
487
487
487
488
488
488
489
489
489
490
490
490
491
491
491
492
492
492
493
493
493
494
494
494
495
495
495
496
496
496
497
497
497
498
498
498
499
499
499
500
500
500
501
501
501
502
502
502
503
503
503
504
504
504
505
505
505
506
506
506
507
507
507
508
508
508
509
509
509
510
510
510
511
511
511
512
512
512
513
513
513
514
514
514
515
515
515
516
516
516
517
517
517
518
518
518
519
519
519
520
520
520
521
521
521
522
522
522
523
523
523
524
524
524
525
525
525
526
526
526
527
527
527
528
528
528
529
529
529
530
530
530
531
531
531
532
532
532
533
533
533
534
534
534
535
535
535
536
536
536
537
537
537
538
538
538
539
539
539
540
540
540
541
541
541
542
542
542
543
543
543
544
544
544
545
545
545
546
546
546
547
547
547
548
548
548
549
549
549
550
550
550
551
551
551
552
552
552
553
553
553
554
554
554
555
555
555
556
556
556
557
557
557
558
558
558
559
559
559
560
560
560
561
561
561
562
562
562
563
563
563
564
564
564
565
565
565
566
566
566
567
567
567
568
568
568
569
569
569
570
570
570
571
571
571
572
572
572
573
573
573
574
574
574
575
575
575
576
576
576
577
577
577
578
578
578
579
579
579
580
580
580
581
581
581
582
582
582
583
583
583
584
584
584
585
585
585
586
586
586
587
587
587
588
588
588
589
589
589
590
590
590
591
591
591
592
592
592
593
593
593
594
594
594
595
595
595
596
596
596
597
597
597
598
598
598
599
599
599
600
600
600
601
601
601
602
602
602
603
603
603
604
604
604
605
605
605
606
606
606
607
607
607
608
608
608
609
609
609
610
610
610
611
611
611
612
612
612
613
613
613
614
614
614
615
615
615
616
616
616
617
617
617
618
618
618
619
619
619
620
620
620
621
621
621
622
622
622
623
623
623
624
624
624
625
625
625
626
626
626
627
627
627
628
628
628
629
629
629
630
630
630
631
631
631
632
632
632
633
633
633
634
634
634
635
635
635
636
636
636
637
637
637
638
638
638
639
639
639
640
640
640
641
641
641
642
642
642
643
643
643
644
644
644
645
645
645
646
646
646
647
647
647
648
648
648
649
649
649
650
650
650
651
651
651
652
652
652
653
653
653
654
654
654
655
655
655
656
656
656
657
657
657
658
658
658
659
659
659
660
660
660
661
661
661
662
662
662
663
663
663
664
664
664
665
665
665
666
666
666
667
667
667
668
668
668
669
669
669
670
670
670
671
671
671
672
672
672
673
673
673
674
674
674
675
675
675
676
676
676
677
677
677
678
678
678
679
679
679
680
680
680
681
681
681
682
682
682
683
683
683
684
684
684
685
685
685
686
686
686
687
687
687
688
688
688
689
689
689
690
690
690
691
691
691
692
692
692
693
693
693
694
694
694
695
695
695
696
696
696
697
697
697
698
698
698
699
699
699
700
700
700
701
701
701
702
702
702
703
703
703
704
704
704
705
705
705
706
706
706
707
707
707
708
708
708
709
709
709
710
710
710
711
711
711
712
712
712
713
713
713
714
714
714
715
715
715
716
716
716
717
717
717
718
718
718
719
719
719
720
720
720
721
721
721
722
722
722
723
723
723
724
724
724
725
725
725
726
726
726
727
727
727
728
728
728
729
729
729
730
730
730
731
731
731
732
732
732
733
733
733
734
734
734
735
735
735
736
736
736
737
737
737
738
738
738
739
739
739
740
740
740
741
741
741
742
742
742
743
743
743
744
744
744
745
745
745
746
746
746
747
747
747
748
748
748
749
749
749
750
750
750
751
751
751
752
752
752
753
753
753
754
754
754
755
755
755
756
756
756
757
757
757
758
758
758
759
759
759
760
760
760
761
761
761
762
762
762
763
763
763
764
764
764
765
765
765
766
766
766
767
767
767
768
768
768
769
769
769
770
770
770
771
771
771
772
772
772
773
773
773
774
774
774
775
775
775
776
776
776
777
777
777
778
778
778
779
779
779
780
780
780
781
781
781
782
782
782
783
783
783
784
784
784
785
785
785
786
786
786
787
787
787
788
788
788
789
789
789
790
790
790
791
791
791
792
792
792
793
793
793
794
794
794
795
795
795
796
796
796
797
797
797
798
798
798
799
799
799
800
800
800
801
801
801
802
802
802
803
803
803
804
804
804
805
805
805
806
806
806
807
807
807
808
808
808
809
809
809
810
810
810
811
811
811
812
812
812
813
813
813
814
814
814
815
815
815
816
816
816
817
817
817
818
818
818
819
819
819
820
820
820
821
821
821
822
822
822
823
823
823
824
824
824
825
825
825
826
826
826
827
827
827
828
828
828
829
829
829
830
830
830
831
831
831
832
832
832
833
833
833
834
834
834
835
835
835
836
836
836
837
837
837
838
838
838
839
839
839
840
840
840
841
841
841
842
842
842
843
843
843
844
844
844
845
845
845
846
846
846
847
847
847
848
848
848
849
849
849
850
850
850
851
851
851
852
852
852
853
853
853
854
854
854
855
855
855
856
856
856
857
857
857
858
858
858
859
859
859
860
860
860
861
861
861
862
862
862
863
863
863
864
864
864
865
865
865
866
866
866
867
867
867
868
868
868
869
869
869
870
870
870
871
871
871
872
872
872
873
873
873
874
874
874
875
875
875
876
876
876
877
877
877
878
878
878
879
879
879
880
880
880
881
881
881
882
882
882
883
883
883
884
884
884
885
885
885
886
886
886
887
887
887
888
888
888
889
889
889
890
890
890
891
891
891
892
892
892
893
893
893
894
894
894
895
895
895
896
896
896
897
897
897
898
898
898
899
899
899
900
900
900
901
901
901
902
902
902
903
903
903
904
904
904
905
905
905
906
906
906
907
907
907
908
908
908
909
909
909
910
910
910
911
911
911
912
912
912
913
913
913
914
914
914
915
915
915
916
916
916
917
917
917
918
918
918
919
919
919
920
920
920
921
921
921
922
922
922
923
923
923
924
924
924
925
925
925
926
926
926
927
927
927
928
928
928
929
929
929
930
930
930
931
931
931
932
932
932
933
933
933
934
934
934
935
935
935
936
936
936
937
937
937
938
938
938
939
939
939
940
940
940
941
941
941
942
942
942
943
943
943
944
944
944
945
945
945
946
946
946
947
947
947
948
948
948
949
949
949
950
950
950
951
951
951
952
952
952
953
953
953
954
954
954
955
955
955
956
956
956
957
957
957
958
958
958
959
959
959
960
960
960
961
961
961
962
962
962
963
963
963
964
964
964
965
965
965
966
966
966
967
967
967
968
968
968
969
969
969
970
970
970
971
971
971
972
972
972
973
973
973
974
974
974
975
975
975
976
976
976
977
977
977
978
978
978
979
979
979
980
980
980
981
981
981
982
982
982
983
983
983
984
984
984
985
985
985
986
986
986
987
987
987
988
988
988
989
989
989
990
990
990
991
991
991
992
992
992
993
993
993
994
994
994
995
995
995
996
996
996
997
997
997
998
998
998
999
999
999
1000
1000
1000
1001
1001
1001
1002
1002
1002
1003
1003
1003
1004
1004
1004
1005
1005
1005
1006
1006
1006
1007
1007
1007
1008
1008
1008
1009
1009
1009
1010
1010
1010
1011
1011
1011
1012
1012
1012
1013
1013
1013
1014
1014
1014
1015
1015
1015
1016
1016
1016
1017
1017
1017
1018
1018
1018
1019
1019
1019
1020
1020
1020
1021
1021
1021
1022
1022
1022
1023
1023
1023
1024
1024
1024
1025
1025
1025
1026
1026
1026
1027
1027
1027
1028
1028
1028
1029
1029
1029
1030
1030
1030
1031
1031
1031
1032
1032
1032
1033
1033
1033
1034
1034
1034
1035
1035
1035
1036
1036
1036
1037
1037
1037
1038
1038
1038
1039
1039
1039
1040
1040
1040
1041
1041
1041
1042
1042
1042
1043
1043
1043
1044
1044
1044
1045
1045
1045
1046
1046
1046
1047
1047
1047
1048
1048
1048
1049
1049
1049
1050
1050
1050
1051
1051
1051
1052
1052
1052
1053
1053
1053
1054
1054
1054
1055
1055
1055
1056
1056
1056
1057
1057
1057
1058
1058
1058
1059
1059
1059
1060
1060
1060
1061
1061
1061
1062
1062
1062
1063
1063
1063
1064
1064
1064
1065
1065

```

Akruti Dabas D20A 18

● Tutorial no. 14

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, including 'Tutorials list' and 'Syllabus'. The main content area is titled '8.3 Data Structures - Structs' (14 / 19). It contains sections on 'Accessing structs' and 'Updating structs', both using the dot operator. A 'Watch a video tutorial on Structs.' link is present. Below these is a 'Assignment' section with a star icon, asking to create a function `remove` that takes a `uint` parameter and deletes a struct member with the given index from the `todos` mapping. At the bottom of the content area, there are 'Check Answer' and 'Show answer' buttons, and a 'Next' button. A green banner at the bottom says 'Well done! No errors.'

On the right, the code editor shows the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract Todos {
    struct Todo {
        string text;
        bool completed;
    }
    // An array of 'Todo' structs
    Todo[] public todos;
}

function create(string memory _text) public {
    // 3 ways to initialize a struct
    // - calling it like a function
    todos.push(Todo(_text, false));
    // key value mapping
    todos.push(Todo({text: _text, completed: false}));
}

// initialize an empty struct and then update it
Todo memory todo;
todo.text = _text;
// todo.completed initialized to false
```

Below the code editor is an 'Explain contract' panel and an 'AI copilot' button.

● Tutorial no. 15

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial navigation, including 'Tutorials list' and 'Syllabus'. The main content area is titled '8.4 Data Structures - Enums' (15 / 19). It contains a note about updating enum values using the dot operator. Below this is a 'Removing an enum value' section, which states that we can use the delete operator to delete the enum value of the variable, which means as for arrays and mappings, to set the default value to 0. A 'Watch a video tutorial on Enums.' link is present. Below these is a 'Assignment' section with a star icon, listing three tasks: defining an enum type `Size`, initializing a variable `sizes`, and creating a getter function `getSize()`. At the bottom of the content area, there are 'Check Answer' and 'Show answer' buttons, and a 'Next' button. A green banner at the bottom says 'Well done! No errors.'

On the right, the code editor shows the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract Enum {
    // Enum representing shipping status
    enum Status {
        Pending,
        Shipped,
        Accepted,
        Rejected,
        Canceled
    }

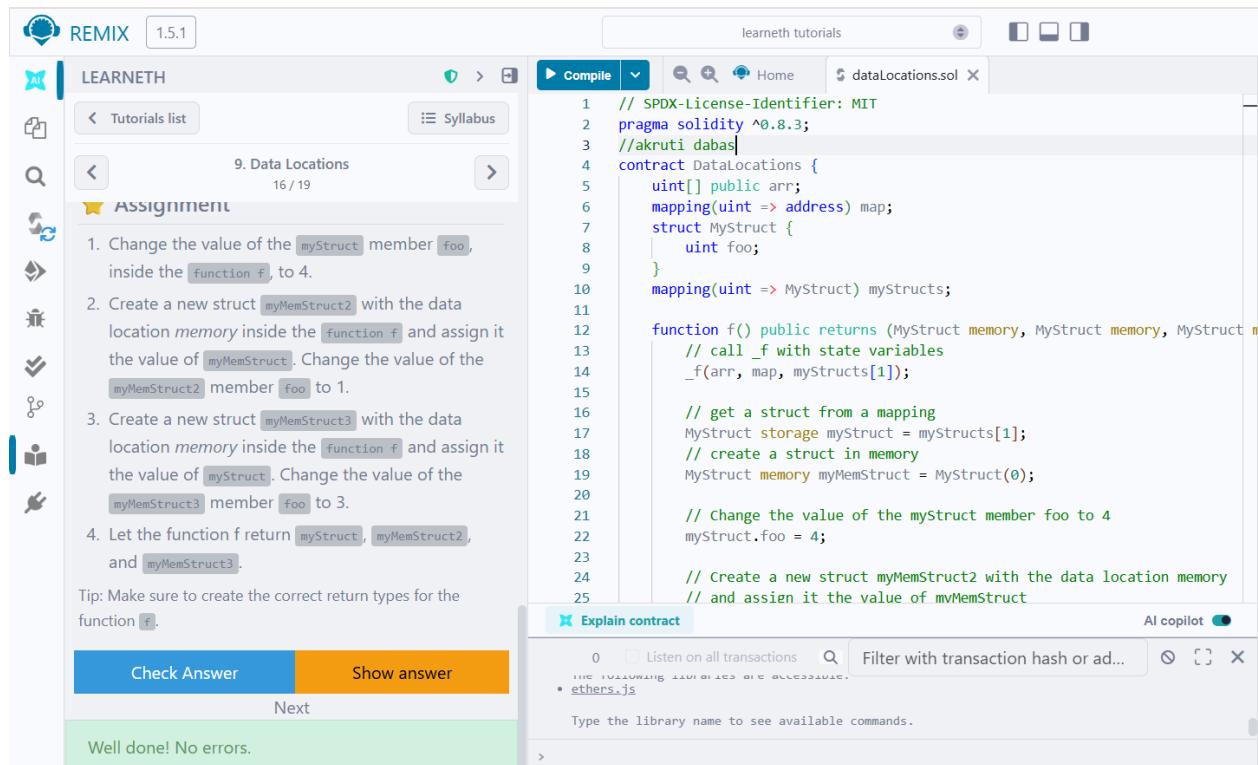
    enum Size {
        S,
        M,
        L
    }

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

    function get() public view returns (Status) {
    }
```

Below the code editor is an 'Explain contract' panel and an 'AI copilot' button.

● Tutorial no. 16



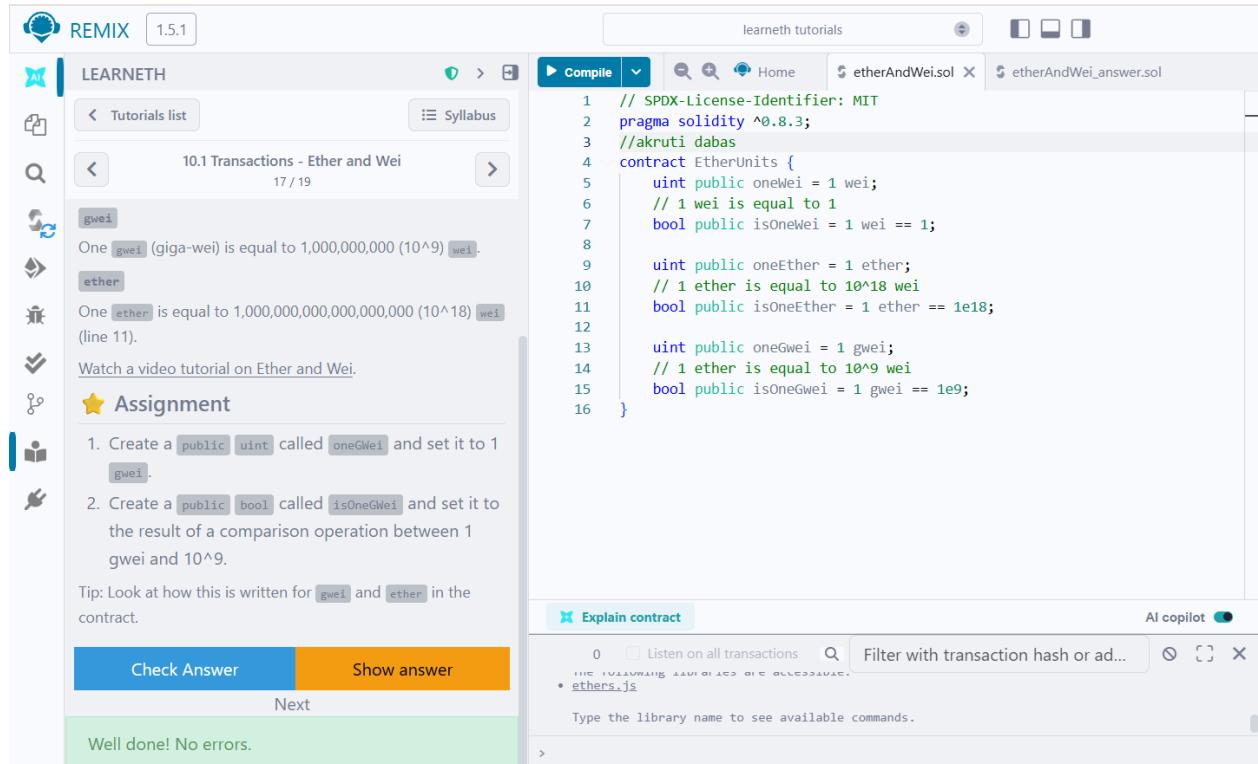
The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' syllabus and a list of tutorials, with '9. Data Locations' selected. The main workspace shows a Solidity contract named 'DataLocations.sol'. The code defines a mapping from uint to address, a struct MyStruct with a uint member 'foo', and a function 'f()' that returns three MyStruct objects. An 'Explain contract' panel is open at the bottom right.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3 //akruti dabas|
4 contract DataLocations {
5     uint[] public arr;
6     mapping(uint => address) map;
7     struct MyStruct {
8         uint foo;
9     }
10    mapping(uint => MyStruct) myStructs;
11
12    function f() public returns (MyStruct memory, MyStruct memory, MyStruct memory) {
13        // call _f with state variables
14        _f(arr, map, myStructs[1]);
15
16        // get a struct from a mapping
17        MyStruct storage myStruct = myStructs[1];
18        // create a struct in memory
19        MyStruct memory myMemStruct = MyStruct(0);
20
21        // Change the value of the myStruct member foo to 4
22        myStruct.foo = 4;
23
24        // Create a new struct myMemStruct2 with the data location memory
25        // and assign it the value of myMemStruct
    }
}

```

● Tutorial no. 17



The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' syllabus and a list of tutorials, with '10.1 Transactions - Ether and Wei' selected. The main workspace shows a Solidity contract named 'etherAndWei.sol'. The code defines a struct 'EtherUnits' with fields for oneWei, oneEther, and oneGwei, along with boolean functions to check if values are equal to these units. An 'Explain contract' panel is open at the bottom right.

```

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

```

Akruti Dabas D20A 18

● Tutorial no. 18

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial list, with '10.2 Transactions - Gas and Gas Price' selected. The main workspace shows a Solidity code editor with the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract Gas {
    uint public i = 0;
    uint public cost = 170367;

    // 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;
        }
    }
}
```

Below the code editor, there is an 'Explain contract' panel and a terminal window.

● Tutorial no. 19

The screenshot shows the REMIX IDE interface. On the left, the sidebar displays the 'LEARNETH' tutorial list, with '10.3 Transactions - Sending Ether' selected. The main workspace shows a Solidity code editor with the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;
//akruti dabas
contract ReceiveEther {
    /*
        Which function is called, fallback() or receive()?
    */
    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 {}  312 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;
}
```

Below the code editor, there is an 'Explain contract' panel and a terminal window.

➤ **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.