

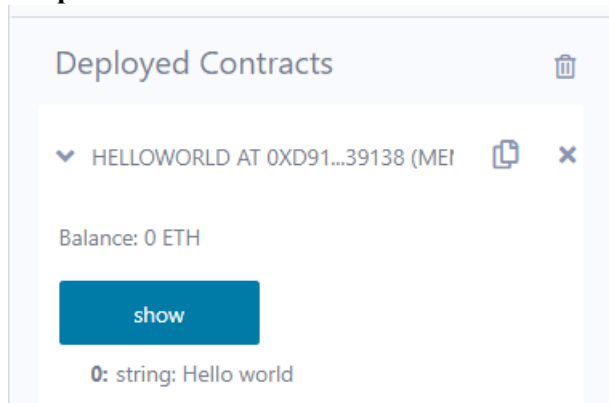
# Practical No. 3

1) Write a solidity smart contract to display hello world message.

**Program:**

```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract HelloWorld{
    function show() public pure returns (string memory){
        return "Hello world";
    }
}
```

**Output:**



2) Write a solidity smart contract to demonstrate state variable, local variable and global variable.

**Program:**

```
//SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
contract SolidityTest {
    uint public storedData = 10; // State variable
    function calculate(uint256 a, uint256 b) public pure returns (uint) {
        uint result; // Local variable exist within function
        result = a + b;    return result;
    }
}
```

```

function getCurrentTimestamp() public view returns (uint256) {
    return block.timestamp; // Global timestamp variable
}
}

```

**Output:**

The screenshot displays a web interface for managing deployed contracts. At the top, a tab labeled 'Deployed Contracts' shows a contract named 'SOLIDITYTEST AT 0XD91...39138 (MEM)'. Below this, the contract's balance is shown as '0. ETH'. There are three interactive buttons: 'calculate', 'getCurrentTim...', and 'storedData'. The 'calculate' button has a dropdown menu showing '5,6'. The 'getCurrentTim...' button shows a value of '0: uint256: 1699453744'. The 'storedData' button shows a value of '0: uint256: 10'.

Below the contract details, a transaction log is shown. It includes a call to 'SolidityTest.getCurrentTimestamp()' with data '0x6c9...230db' and a call to 'SolidityTest.storedData()' with data '0x2a1...afcd9'. The transaction details for the 'storedData()' call are as follows:

Field	Value
from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	SolidityTest.storedData() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost	2402 gas (Cost only applies when called by a contract)
input	0x2a1...afcd9
decoded input	{}
decoded output	{ "0": "uint256: 10" }
logs	[]

**3)Write a solidity smart contract to demonstrate getter and setter methods.**

**Program:**

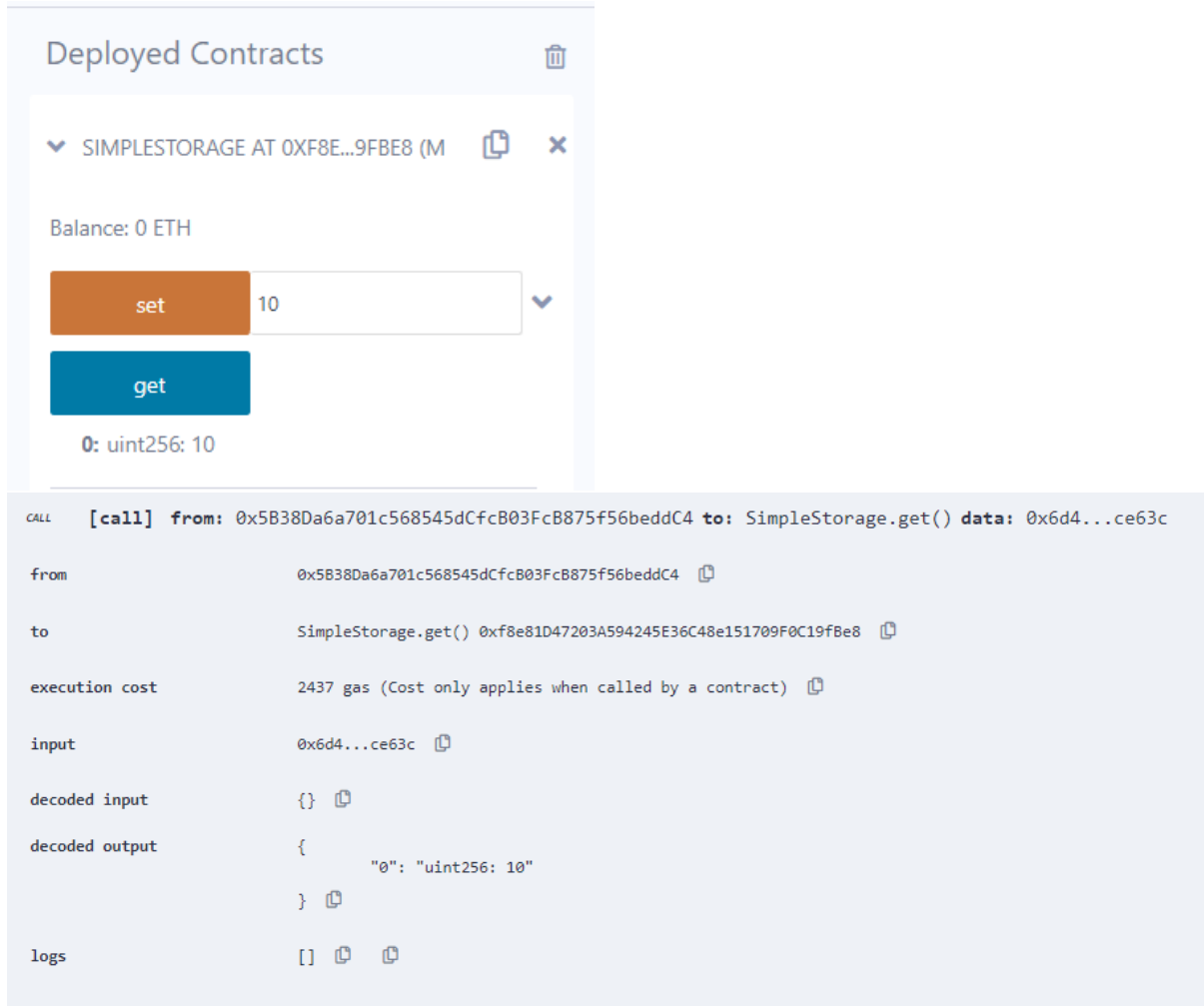
```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16<0.8.20;
contract SimpleStorage{
    uint storedData;
    function set(uint x) public {
        storedData=x;
    }
    function get() public view returns (uint){
        return storedData;
    }
}

```

```
}
```

### Output:



Deployed Contracts

▼ SIMPLESTORAGE AT 0XF8E...9FBE8 (M)

Balance: 0 ETH

set 10

get

0: uint256: 10

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SimpleStorage.get() data: 0x6d4...ce63c

from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to SimpleStorage.get() 0xf8e81D47203A594245E36C48e151709F0C19fBe8

execution cost 2437 gas (Cost only applies when called by a contract)

input 0x6d4...ce63c

decoded input {}

decoded output {  
 "0": "uint256: 10"  
}

logs []

### 4) Write a solidity smart contract to demonstrate function modifier.

#### Program:

```
// Demonstration of modifier
// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract Owner
{
    address owner;
    uint price;
    constructor()
    {
        owner = msg.sender;
    }
    modifier onlyOwner
    {
        require(msg.sender == owner, 'Only owner call this function !');
        _;
    }
    function changePrice(uint _price) public onlyOwner
    {

```

```

    price = _price;
  }
}

```

### Output:

The screenshot displays a web interface for managing deployed contracts. At the top, there's a section titled "Deployed Contracts" with a trash icon. Below it, a dropdown menu shows "OWNER AT 0XD91...39138 (MEMORY)" with copy and close icons. Under this, it shows "Balance: 0 ETH". A button labeled "changePrice" is next to an input field containing the value "25".

Below the interface, a transaction confirmation message is shown: "[vm] from: 0x5B3...eddC4 to: Owner.changePrice(uint256) 0xd91...39138 value: 0 wei data: 0xa2b...00019 logs: 0 hash: 0x4ee...267de".

A detailed transaction log follows, listing various fields:

- status: true Transaction mined and execution succeed
- transaction hash: 0x4eeb69db7a08ad5d9b2a3f6203e091c61c344e023a957ce9ffeb28f1e78267de
- block hash: 0x74e0b9ebcb4fe595515113297dec050f59be4182af4d20d5bac39a4f6c7faae4
- block number: 2
- from: 0x5B38Da6a701c568545dCfcB03Fc8875f56beddC4
- to: Owner.changePrice(uint256) 0xd9145CCE52D386f254917e481e844e9943F39138
- gas: 52750 gas
- transaction cost: 45869 gas
- execution cost: 24605 gas
- input: 0xa2b...00019
- decoded input: {"uint256 \_price": "25"}
- decoded output: {}
- logs: []
- val: 0 wei

### 5) Write a Solidity program to demonstrate arrays Push operation and Pop operation.

#### Program:

```

// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract ArrayOperations {
    uint[] public array;
    function pushValue(uint value) public {
        array.push( value);
    }
    function popValue() public {
        require(array.length > 0, "Array is empty");
        array.pop();
    }
    function getArray() public view returns (uint[] memory) {
        return array;
    }
}

```

## Output:

The screenshot displays a web interface for managing deployed contracts. At the top, a tab labeled 'Deployed Contracts' is active. Below it, a contract named 'ARRAYOPERATIONS AT 0X9D7...B5E9...' is selected. The contract's balance is shown as '0 ETH'. There are several interactive buttons: 'popValue' (orange), 'pushValue' (orange) with a value of '8' entered, 'array' (blue) with a value of '2' entered, and 'getArray' (blue). Below these buttons, the state is shown as '0: uint256: 3' and '0: uint256[]: 6,5,3,4,8'. At the bottom, a transaction call log is visible, showing a call to 'ArrayOperations.getArray()' with input data '0xd50...4ea1d'. The log details the transaction from a specific address, the execution cost of 15068 gas, and the decoded output: {'0': 'uint256[]: 6,5,3,4,8'}. The logs array is empty.

## 6) Write a Solidity program to demonstrate creating a fixed-size array and access array element.

### Program:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.8.20;
contract FixedArrayDemo {
    int[5] public array;
    constructor() {
        array[0] = 10;
        array[1] = 20;
        array[2] = 30;
        array[3] = 40;
        array[4] = 50;
    }
    function getArrayElement(uint index) public view returns (int) {
        require(index < array.length, "Index out of range");
        return array[index];
    }
}
```

```

    }
}

```

### Output:

The screenshot shows a web interface for 'Deployed Contracts'. It displays a contract named 'FIXEDARRAYDEMO AT 0X93F...C96CC'. The balance is 0 ETH. There are two input fields: 'array' with the value 5 and 'getArrayElement' with the value 2. Below these, it shows '0: int256: 30'. Below the interface, a call log is shown for a transaction from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to FixedArrayDemo.getArrayElement(uint256) 0x93f8ddd876c7d8E3323723500e83E202A7C96CC. The call log shows the execution cost (2732 gas), input (0x142...00002), decoded input ({"uint256 index": "2"}), decoded output ({"0": "int256: 30"}), and logs (empty).

### 7) Write a Solidity program to demonstrate creating a fixed-size array and access array element.

#### Program:

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.8.20;
contract DynamicArray {
    uint[] public dynamicArray;
    function addValue(uint _value) public {
        dynamicArray.push(_value);
    }
    function getArrayElement(uint index) public view returns (uint) {
        require(index < dynamicArray.length, "Index out of range");
        return dynamicArray[index];
    }
    function getArrayLength() public view returns (uint) {
        return dynamicArray.length;
    }
}

```

## Output:

Deployed Contracts

▼ DYNAMICARRAYDEMO AT 0XD7C...FA

Balance: 0 ETH

addValue 8

dynamicArray 3

0: uint256: 8

getArrayElement 2

0: uint256: 6

getArrayLength

0: uint256: 4

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: DynamicArrayDemo.getArrayElement(uint256) data: 0x142...00002

from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to DynamicArrayDemo.getArrayElement(uint256) 0xd7Ca4e99F7C171B9ea2De80d3363c47009afaC5F

execution cost 5013 gas (Cost only applies when called by a contract)

input 0x142...00002

decoded input { "uint256 index": "2" }

decoded output { "0": "uint256: 6" }

## 8) Write a solidity smart contract to demonstrate use of structure.

### Program:

```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract StructureDemo1 {
    struct Book{
        string title;
        string author;
        uint book_id;
    }
    Book b1;
    function setBook() public {
        b1 = Book('Blockchain Revolution','Alex Tapscott',10);
    }
    function getBookId() public view returns (uint) {
        return b1.book_id;
    }
}
```

```

function getBookDetails() public view returns (string memory,string memory,uint) {
    return (b1.title,b1.author,b1.booK_id);
}
}

```

**Output:**

The screenshot displays a web interface for managing deployed contracts. At the top, a section titled "Deployed Contracts" shows a contract named "STRUCTUREDEMO1 AT 0XF27...501CB". Below this, the contract's balance is shown as "0 ETH". There are three buttons: "setBook" (orange), "getBookDetails" (blue), and "getBookId" (blue). The "getBookDetails" button is active, showing the following data: "0: string: Blockchain Revolution", "1: string: Alex Tapscott", and "2: uint256: 10". Below this, the "getBookId" button is active, showing "0: uint256: 10".

Below the contract interface, a transaction call log is shown. The call is from "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4" to "StructureDemo1.getBookId()" with data "0x731...0d33c". The execution cost is 2421 gas. The decoded output is a JSON object: {"0": "uint256: 10"}. The logs are empty.

**9) Write a solidity smart contract to calculate percentage of marks obtained by students for six subject in final examination.**

**Program:**

```

// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract PercentageCalculator {
    function calculatePercentage() public pure returns (uint) {
        uint totalMarks = 600;
        uint subject1=70;
        uint subject2=75;
        uint subject3=67;
        uint subject4=87;
        uint subject5=90;
    }
}

```



```

    uint subject6=85;
    uint total = subject1 + subject2 + subject3 + subject4 + subject5 + subject6;
    uint percentage = (total * 100) / totalMarks;
    return percentage;
}
}

```

**Output:**

The screenshot displays a web interface for a deployed contract named 'PERCENTAGECALCULATOR AT 0XF2B..'. The contract's balance is shown as '0 ETH'. A blue button labeled 'calculatePerce...' is visible. Below the button, the output of the function is shown as '0: uint256: 79'. Below the interface, a transaction log is displayed, showing a call to 'PercentageCalculator.calculatePercentage()' with input '0x182...6f8c7' and output '{"0": "uint256: 79"}'.

**10) Write a solidity smart contract to find the factorial of entered number.**

**Program:**

```

// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract FactorialCalculator {
    function calculateFactorial(uint256 num) public pure returns (uint256) {
        require(num >= 0, "Number must be non-negative");
        uint256 result = 1;
        for (uint256 i = 1; i <= num; i++) {
            result = result * i;
        }
        return result;
    }
}

```

## Output:

**Deployed Contracts**

▼ FACTORIALCALCULATOR AT 0XB9E...C

Balance: 0 ETH

**calculateFactorial** 5

0: uint256: 120

**CALL [call] from:** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 **to:** FactorialCalculator.calculateFactorial(uint256) **data:** 0x3d4...00005

**from** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

**to** FactorialCalculator.calculateFactorial(uint256) 0xB9e2A2008d3A58adD8CC1cE9c158F6D4bB9C6d72

**execution cost** 2821 gas (Cost only applies when called by a contract)

**input** 0x3d4...00005

**decoded input** { "uint256 num": "5" }

**decoded output** { "0": "uint256: 120" }

**logs** []

## 11) Write a solidity smart contract to check whether entered number is palindrome or not.

### Program:

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Palindrome {

function isPalindrome(uint256 num) public pure returns (bool) {

uint256 original = num;

uint256 reverse = 0;

while (num > 0) {

uint256 remainder = num % 10;

reverse = reverse \* 10 + remainder;

num = num / 10;

}



return original == reverse;

}


}

## Output:

Deployed Contracts









▼ PALINDROME AT 0X56A...5E31B (MEM)  

Balance: 0 ETH

**isPalindrome** 121 

0: bool: true

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Palindrome.isPalindrome(uint256) data: 0x041...00079

from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 
to	Palindrome.isPalindrome(uint256) 0x56a2777e796eF23399e9E1d791E1A0410a75E31b 
execution cost	3241 gas (Cost only applies when called by a contract) 
input	0x041...00079 
decoded input	<pre>{   "uint256 num": "121" }</pre> 
decoded output	<pre>{   "0": "bool: true" }</pre> 
logs	<pre>[]</pre>  

## 12) Write a solidity smart contract to generate Fibonacci Series up to given number.

### Program:

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract FibonacciSeries {

```
    function generateFibonacci(uint256 number) public pure returns (uint256[] memory) {
        require(number > 0, "Number must be greater than 0");
        uint256[] memory series = new uint256[](number);
        series[0] = 0;
        if (number > 1) {
            series[1] = 1;
        }
        for (uint256 i = 2; i < number; i++) {
            series[i] = series[i - 1] + series[i - 2];
        }
        return series;
    }
```

```
}
```

## Output:

Deployed Contracts

▼ FIBONACCISERIES AT 0XB54...5EEEB ⓘ

Balance: 0 ETH

generateFibon... 5 ▼

0: uint256[]: 0,1,1,2,3

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: FibonacciSeries.generateFibonacci(uint256) data: 0xe20...00005

from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ

to FibonacciSeries.generateFibonacci(uint256) 0xb5465ED8EcD4F79dD48E10A7C8e7a50664e5eeEB ⓘ

execution cost 5245 gas (Cost only applies when called by a contract) ⓘ

input 0xe20...00005 ⓘ

decoded input { "uint256 number": "5" } ⓘ

decoded output { "0": "uint256[]: 0,1,1,2,3" } ⓘ

logs [] ⓘ ⓘ

## 13) Write a solidity smart contract to check whether entered number is prime number or not.

### Program:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract PrimeNumber {
    function isPrime(uint num) public pure returns (bool) {
        require(num > 1, "Number must be greater than 1");
        for (uint i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

## Output:

Deployed Contracts

▼ PRIMENUMBER AT 0X1D1...3B8BD (M)

Balance: 0 ETH

**isPrime**  ▼

0: bool: true

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: PrimeNumber.isPrime(uint256) data: 0x427...00043

from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	PrimeNumber.isPrime(uint256) 0x1d142a62E2e98474093545D4A3A0f7DB9503B88BD
execution cost	18684 gas (Cost only applies when called by a contract)
input	0x427...00043
decoded input	<pre>{  "uint256 num": "67"}</pre>
decoded output	<pre>{  "0": "bool: true"}</pre>
logs	<pre>[]</pre>

**14) Write a solidity smart contract to create arithmetic calculator which includes functions for operations addition, subtraction, multiplication, division etc.**

### Program:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract Calculator {
    function add(uint a, uint b) public pure returns (uint) {
        return a + b;
    }
    function subtract(uint a, uint b) public pure returns (uint) {
        return a - b;
    }
    function multiply(uint a, uint b) public pure returns (uint) {
        return a * b;
    }
    function divide(uint a, uint b) public pure returns (uint) {
        return a / b;
    }
}
```

## Output:

### Deployed Contracts

▼ CALCULATOR AT 0X8B8...EFF42 (MEM)

Balance: 0 ETH

add

5,6

▼

0: uint256: 11

divide

6,2

▼

0: uint256: 3

multiply

4,3

▼

0: uint256: 12

subtract

12,6

▼

0: uint256: 6

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.subtract(uint256,uint256) data: 0x3ef...00006

from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	Calculator.subtract(uint256,uint256) 0x8B801270f3e02eA2AACcf134333D5E5A019eFF42
execution cost	963 gas (Cost only applies when called by a contract)
input	0x3ef...00006
decoded input	<pre>{   "uint256 a": "12",   "uint256 b": "6" }</pre>
decoded output	<pre>{   "0": "uint256: 6" }</pre>
logs	[ ]

## 15) Write a solidity smart contract to demonstrate view function and pure function.

### Program:

// SPDX-License-Identifier: MIT

pragma solidity >= 0.7 <0.9;

contract Function {

uint a = 1; uint b = 2; // local variable

function getViewResult() public view returns(uint product, uint sum){

product = a \* b;

sum = a + b;

}

function getPureResult() public pure returns(uint product , uint sum){

uint c = 3;

uint d = 2;

product = c\*d;

```

    sum = c+d;
  }
}

```

**Output:**

The screenshot displays a web interface for managing deployed contracts. At the top, there's a section titled "Deployed Contracts" with a trash icon. Below it, a dropdown menu shows "FUNCTION AT 0XF8E...9FBE8 (MEMOF)" with copy and close icons. The main area shows the contract's state: "Balance: 0. ETH". There are two buttons: "getPureResult" and "getViewResult". Below "getPureResult", the output is shown: "0: uint256: product 6" and "1: uint256: sum 5". Below "getViewResult", the output is: "0: uint256: product 2" and "1: uint256: sum 3".

Below the contract interface, a transaction call log is shown. It details a call to the "Function.getViewResult()" method. The "from" field shows the caller's address: "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4". The "to" field shows the contract address: "Function.getViewResult() 0xf8e81D47203A594245E36C48e151709F0C19fBe8". The "execution cost" is "5254 gas (Cost only applies when called by a contract)". The "input" is "0x7c2...2b352". The "decoded input" is an empty object "{}". The "decoded output" is a JSON object: {"0": "uint256: product 2", "1": "uint256: sum 3"}. The "logs" field is empty.

**16) Write a solidity smart contract to demonstrate inbuilt mathematical functions.**

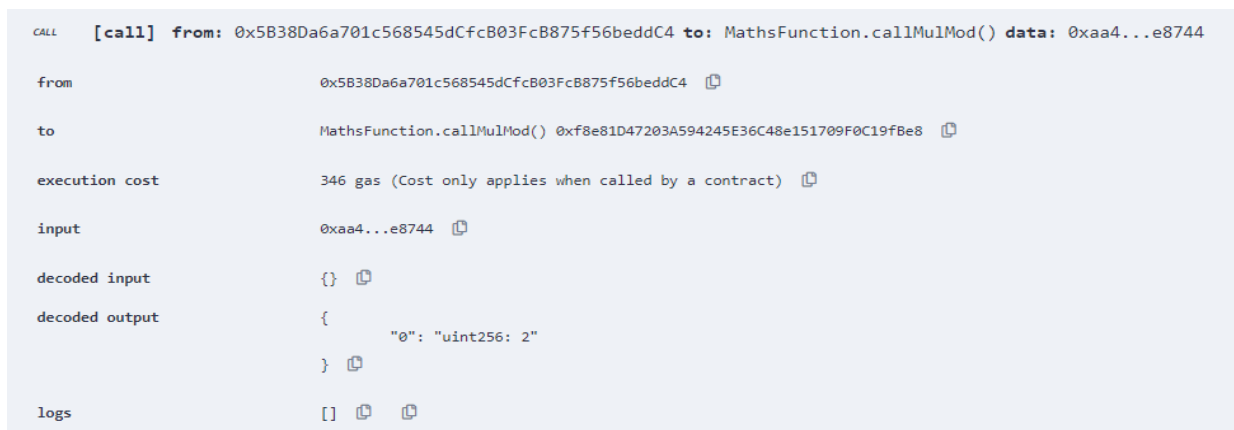
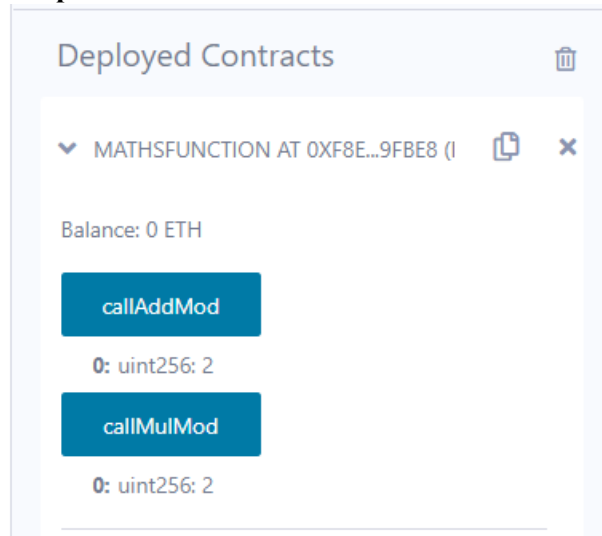
**Program:**

```

// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract MathsFunction{
    function callAddMod() public pure returns(uint){
        return addmod(2, 8, 4);
    }
    function callMulMod() public pure returns(uint){
        return mulmod(5, 4, 3);
    }
}

```

## Output:



## 17) Write a solidity smart contract to demonstrate inheritance in contract.

### Program:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract C{
    uint public data =30;
    uint internal iData =10;
    function x() public returns (uint){
        data =3;
        return data;
    }
}
contract Caller{
    C c=new C();
    function f() public view returns(uint){
        return c.data();
    }
}
contract D is C{
    function y() public returns(uint){
        iData=3;
    }
}
```



```

        return iData;
    }
    function getResult() public pure returns(uint){
        uint a=1;
        uint b=2;
        uint result=a+b;
        return result;
    }
}

```

### Output:

#### Deployed Contracts

▼ C AT 0xEF9...10EBF (MEMORY)
📄 ✕

Balance: 0. ETH

X

data

0: uint256: 3

✓

**[vm]** from: 0x5B3...eddC4 to: C.x() 0xEF9...10eBf value: 0 wei data: 0x0c5...5699c logs: 0 hash: 0x67f...fd0c9

status	0x1 Transaction mined and execution succeed
transaction hash	0x67fd77dca7f4221489e4dcf26709d47b053e8d2118c2090ae606bd9c8cbfd0c9 <span>📄</span>
block hash	0xc251cccc7bcabc32bf5e998bdf16dd2c78cdf8918ce0ad800086acbae9622fca <span>📄</span>
block number	30 <span>📄</span>
from	0x5B38Da6a701c568545dCfc803FcB875f56beddC4 <span>📄</span>
to	C.x() 0xEF9f1ACE83dfbB8f559Da621f4aEA72C6EB10eBf <span>📄</span>
gas	gas <span>📄</span>
transaction cost	23686 gas <span>📄</span>
execution cost	2622 gas <span>📄</span>
input	0x0c5...5699c <span>📄</span>
decoded input	{ } <span>📄</span>
decoded output	{ "0": "uint256: 3" }
logs	[ ] <span>📄</span> <span>📄</span>

call to C.data

### 18) Write a solidity smart contract to demonstrate events.

#### Program:

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.8.20;
contract EvenDemo

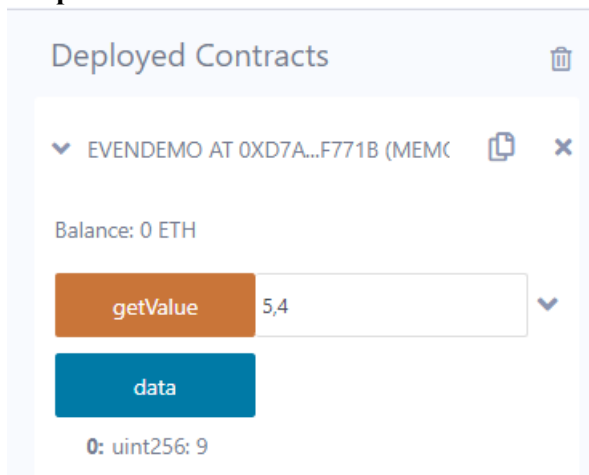
```

```

{
  uint256 public data = 0; //state variable
  event Increment(address owner); //Declaring an event
  function getValue(uint _a, uint _b) public
  {
    emit Increment(msg.sender);
    data = _a + _b;
  }
}

```

**Output:**



**19) Write a solidity smart contract to demonstrate assert statement and revert statement.**

**Program:**

**Assert**

// SPDX-License-Identifier: MIT

pragma solidity >=0.7 <0.9;

contract assertStatement {

bool result; // Defining a state variable // Defining a function to check condition

function checkOverflow(uint \_num1, uint \_num2) public {

uint sum = \_num1 + \_num2; assert(sum<=255); result = true;

}

// Defining a function to print result of assert statement

```

function getResult() public view returns(string memory) {
    if(result == true){ return "No Overflow"; }
    else{ return "Overflow exist"; }
}
}

```

### Output:

The screenshot shows a web interface for a deployed contract named "ASSERTSTATEMENT AT 0XD91...39138". The interface includes a "Balance: 0. ETH" display, a "checkOverflow" button, an input field with the value "10,12", and a "getResult" button. Below the interface, the transaction details are shown for a call to `assertStatement.getResult()` with data `0xde2...92789`. The transaction was executed from `0x5B38Da6a701c568545dCfcB03FcB875f56beddC4` to `assertStatement.getResult() 0xd9145CCE52D386f254917e481eB44e9943F39138` with an execution cost of 2906 gas. The decoded output is `{ "0": "string: No Overflow" }`.

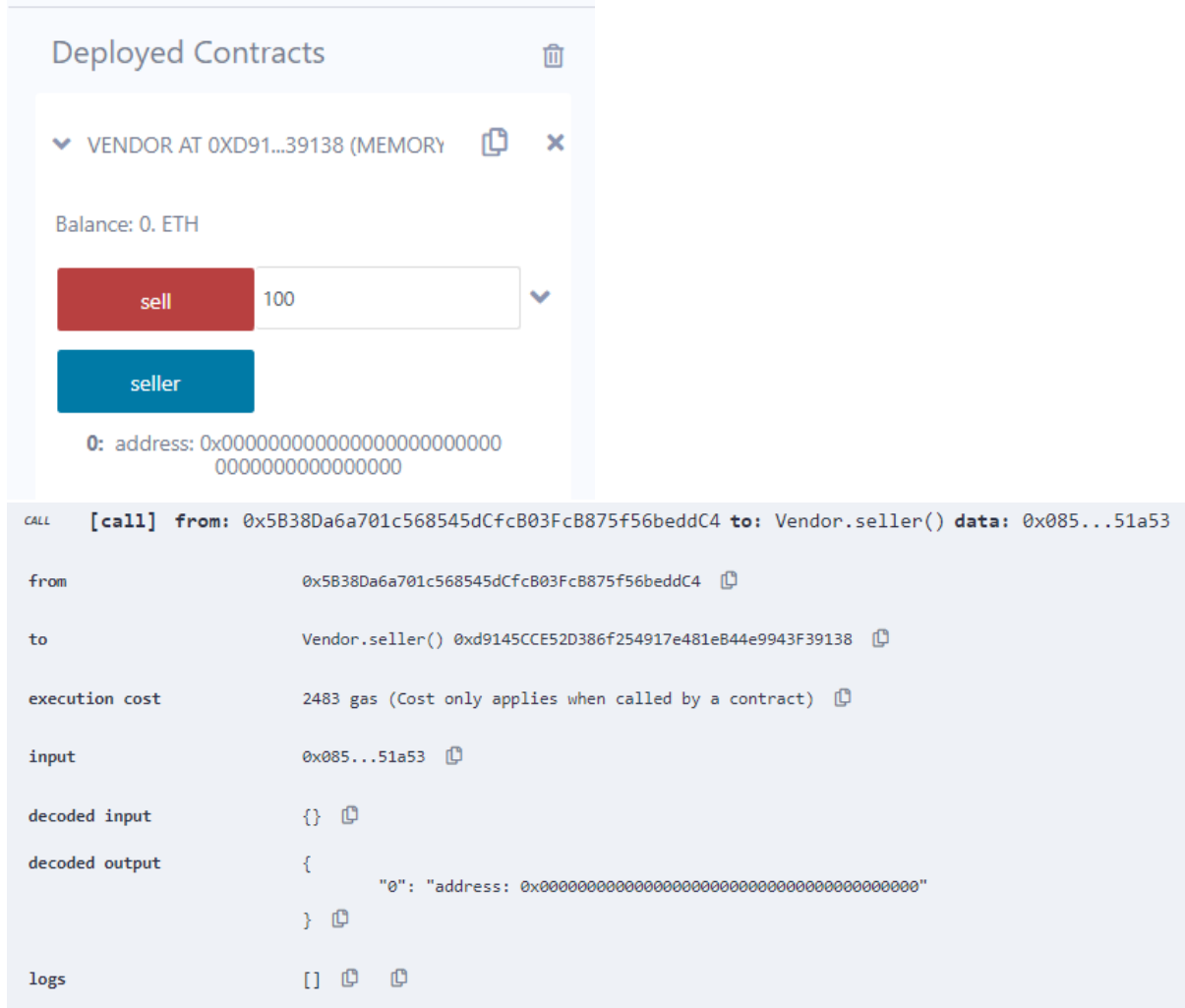
### Revert:

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.7 <0.9;
contract Vendor{
    address public seller;
    modifier onlySeller()
    {
        require(msg.sender==seller,'Only seller can call this');
        _;
    }
    function sell(uint amount) public payable onlySeller{
        if(amount >msg.value/2 ether)
            revert("Not enough Ether provided.");
    }
}

```

## Output:



Deployed Contracts

▼ VENDOR AT 0XD91...39138 (MEMORY)

Balance: 0. ETH

sell 100

seller

0: address: 0x00000000000000000000000000000000

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Vendor.seller() data: 0x085...51a53

from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to Vendor.seller() 0xd9145CCE52D386f254917e481e844e9943F39138

execution cost 2483 gas (Cost only applies when called by a contract)

input 0x085...51a53

decoded input {}

decoded output { "0": "address: 0x00000000000000000000000000000000" }

logs []

**20) Write a solidity smart contract for Bank Account which provides operations such as check account balance, withdraw amount and deposit amount etc.**

### Program:

// SPDX-License-Identifier: MIT

pragma solidity >=0.7 <0.9;

contract Banking {

mapping (address => uint) public userAccount; // Balance

mapping (address => bool) public userExists;

function createAcc() public payable returns (string memory) {

require(!userExists[msg.sender], 'Account already created');

if (msg.value == 0) { userAccount[msg.sender] = 0;

}

else {

userAccount[msg.sender] = msg.value;

}

userExists[msg.sender] = true;

return 'Account created!';

}

function deposit() public payable returns (string memory) {

```

    require(userExists[msg.sender], 'Account is not created');
    require(msg.value > 0, 'Value for deposit is not zero');
    userAccount[msg.sender] += msg.value;
    return 'Deposited successfully';
}

function withdraw(uint amount) public returns(string memory) {
    require(userExists[msg.sender], 'Account is not created');
    require(userAccount[msg.sender] >= amount, 'Insufficient balance in bank account');
    require(amount > 0, 'Enter a non-zero value for withdrawal');
    userAccount[msg.sender] -= amount;
    payable(msg.sender).transfer(amount);
    return 'Withdrawal successful';
}

function userAccountBalance() public view returns (uint) {
    return userAccount[msg.sender];
}
}

```

### Output:

The screenshot displays a web interface for a deployed Solidity contract named "BANKING AT 0x540...C7569". The interface shows a balance of 0.000000000000000005 ETH. There are buttons for "createAcc", "deposit", "withdraw" (with a value of 50), "userAccount" (with an address), and "userAccountBa...". Below the interface is a transaction log showing a successful "deposit" transaction with a value of 100 wei.

Label	Value
status	true Transaction mined and execution succeed
transaction hash	0x83608ebcd15a04b4804ab9f58c865437823e0802d41e9c217f9e745cdfecb59
block hash	0x97d742557ab8a1213a73c609168a544ba3eb51f43b6f078ce676741d27435d4f
block number	29
from	0x5838Da6a701c568545dCfcB875f56beddC4
to	Banking.deposit() 0x540d7E428D5207B30EE03F2551Cbb5751D3c7569
gas	53462 gas
transaction cost	46488 gas
execution cost	25424 gas
input	0xd0e...30db0
decoded input	()
decoded output	{ "0": "string: Deposited successfully" }

**21) Write a program in solidity to create a structured student with Roll no, Name, Class, Department, Course enrolled as variables.**

**1)Add information of 5 students**

**2)Search for a student using Roll no**

**3)Display all Information**

**Program:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.7 <0.9;

```
contract StudentInfo {
    struct Student {
        uint256 rollNo;
        string name;
        string class;
        string department;
        string courseEnrolled;
    }
    mapping(uint256 => Student) public students;
    uint256 public studentCount;
    event StudentAdded(uint256 rollNo, string name);
    function addStudent(uint256 _rollNo,string memory _name,string memory _class,
    string memory _department,string memory _courseEnrolled) public {
        require(_rollNo != 0, "Roll number must be greater than 0");
        require(bytes(_name).length > 0, "Name cannot be empty");
        students[_rollNo] = Student({
            rollNo: _rollNo,
            name: _name,
            class: _class,
            department: _department,
            courseEnrolled: _courseEnrolled
        });
        studentCount++;
        emit StudentAdded(_rollNo, _name);
    }
    function getStudent(uint256 _rollNo) public view returns (uint256 rollNo,string memory
name,string memory class,
    string memory department,string memory courseEnrolled){
        Student storage student = students[_rollNo];
        return (student.rollNo,student.name,student.class,student.department,student.courseEnrolled );
    }
    function getAllStudents() public view returns (Student[] memory) {
        Student[] memory studentList = new Student[](studentCount);
        for (uint256 i = 1; i <= studentCount; i++) {
            studentList[i - 1] = students[i];
        }
        return studentList;
    }
}
```

## Output:

**Deployed Contracts**

STUDENTINFO AT 0X358...D5EE3 (MEI)

Balance: 0. ETH

**addStudent**

\_rollNo: 40

\_name: siddhivinayak

\_class: SY

\_department: MCA

\_courseEnrolled: 2023

Calldata Parameters transact

**getAllStudents**

0: tuple(uint256,string,string,string,string): 0,,,,,2,gayatri,SY,MCA,2023,0,,,,

**getStudent** 37

0: uint256: rollNo 37

1: string: name tanuja

2: string: class SY

3: string: department MCA

4: string: courseEnrolled 2023

**studentCount**

0: uint256: 3

**students** 02

0: uint256: rollNo 2

1: string: name gayatri

2: string: class SY

3: string: department MCA

4: string: courseEnrolled 2023

**CALL [call] from:** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 **to:** StudentInfo.students(uint256) **data:** 0x06e...00002

**from** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

**to** StudentInfo.students(uint256) 0x358AA13c52544ECCECF6B0ADD0f801012ADAD5eE3

**execution cost** 15528 gas (Cost only applies when called by a contract)

**input** 0x06e...00002

**decoded input** { "uint256 ": "2" }

**decoded output** { "0": "uint256: rollNo 2", "1": "string: name gayatri", "2": "string: class SY", "3": "string: department MCA", "4": "string: courseEnrolled 2023" }

**logs** []

**22) Create a structure Consumer with Name , Address, Consumer ID, Units and Amount as members. Write a program in solidity to calculate the total electricity bill according to the given condition:**

**For first 50 units Rs. 0.50/unit. For next 100 units Rs. 0.75/unit. For next 100 units Rs. 1.20/unit. For unit above 250 Rs. 50/unit. An additional surcharge of 20% is added to the bill. Display the information of 5 such consumers along with their units consumed and amount.**

**Program:**

// SPDX-License-Identifier: MIT

```

pragma solidity ^0.8.0;
contract ElectricityBillCalculator {
    struct Consumer {
        string Name;
        string Address;
        uint ConsumerID;
        uint Units;
        uint Amount;
    }
    Consumer[5] public consumers;
    function calculateBill(uint _consumerIndex, string memory name, string memory
consumerAddress,
    uint consumerID, uint units) public returns (uint) {
        require(_consumerIndex < 5, "Consumer index should be less than 5");
        uint billAmount;
        uint surcharge;
        uint total_bill;
        if (units <= 50) {
            billAmount = units * 50/100;
            return billAmount;
        }
        else if (units <= 150) {
            billAmount = 25 + ((units - 50) * 75/100);
        } else if (units <= 250) {
            billAmount = 25+75 + ((units - 150) * 120/100);
        } else {
            billAmount = 25+75+120 + ((units - 250) * 50);
        }
        surcharge =billAmount*20/100;
        total_bill = billAmount + surcharge;
        consumers[_consumerIndex] = Consumer({
            Name: name,
            Address: consumerAddress,
            ConsumerID: consumerID,
            Units: units,
            Amount: billAmount
        });
        return total_bill;
    }
    function getAllConsumersInfo() public view returns (Consumer[5] memory) {
        return consumers; }
}

```



Output:

Deployed Contracts

ELECTRICITYBILLCALCULATOR AT 0XC

Balance: 0. ETH

calculateBill

\_consumerIndex: 1

name: tanuja

consumerAddress: mandangad

consumerID: 12

units: 120

Calldata

Parameters

transact

consumers

1

0: string: Name tanuja

1: string: Address mandangad

2: uint256: ConsumerID 12

3: uint256: Units 120

4: uint256: Amount 77

getAllConsume...

0: tuple(string,string,uint256,uint256,uint256)[5]: „0,0,0,tanuja,mandangad,12,120,77,,,0,0,0,,,0,0,0,0,0

```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ElectricityBillCalculator.consumers(uint256) data: 0x465...00001
call to ElectricityBillCalculator.getAllConsumersInfo

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ElectricityBillCalculator.getAllConsumersInfo() data: 0x3cf...0a59f

from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to ElectricityBillCalculator.getAllConsumersInfo() @xd9145CCE520386f254917e481eB44e9943f39138
execution cost 67043 gas (Cost only applies when called by a contract)
input 0x3cf...0a59f
decoded input {}
decoded output {
  "0": "tuple(string,string,uint256,uint256,uint256)[5]: „0,0,0,tanuja,mandangad,12,120,77,,,0,0,0,,,0,0,0,0,0"
}
logs []
```