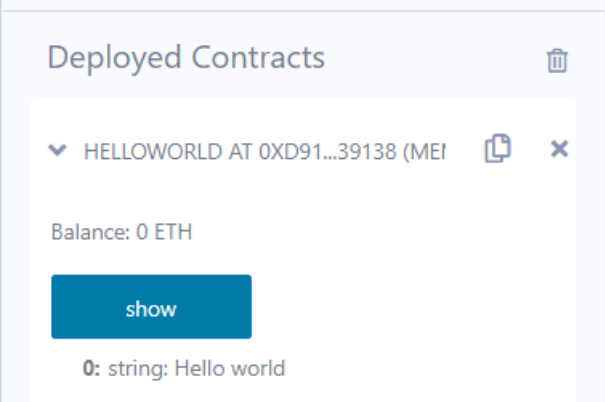
Practical No. 3

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:




```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: HelloWorld.show() data: 0xcc8...0f6f3

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      HelloWorld.show() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost          715 gas (Cost only applies when called by a contract)
input                   0xcc8...0f6f3
decoded input           {}
decoded output          {
                            "0": "string: Hello world"
                        }
logs                    []
```

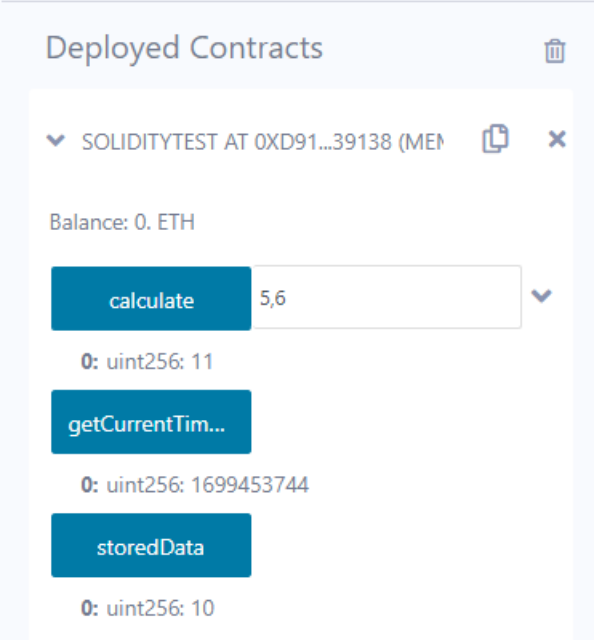
2) Write a solidity smart contract to demonstrate state variable, local variable and global variable.

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:




```
CALL    [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.getCurrentTimestamp() data: 0x6c9...230db
call to SolidityTest.storedData

CALL    [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.storedData() data: 0x2a1...afcd9

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                    []
```

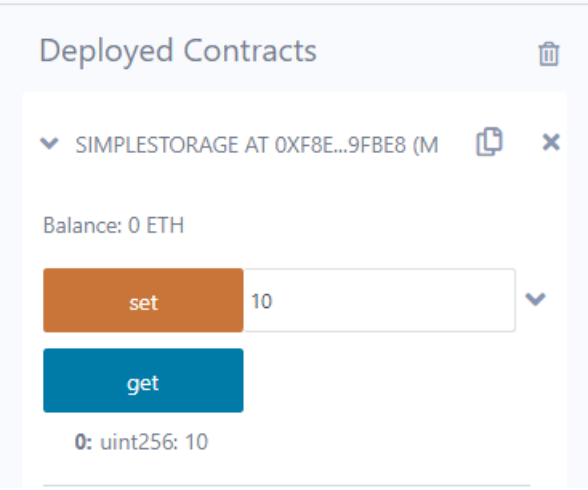
3)Write a solidity smart contract to demonstrate getter and setter methods.

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:




```
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                    []
```

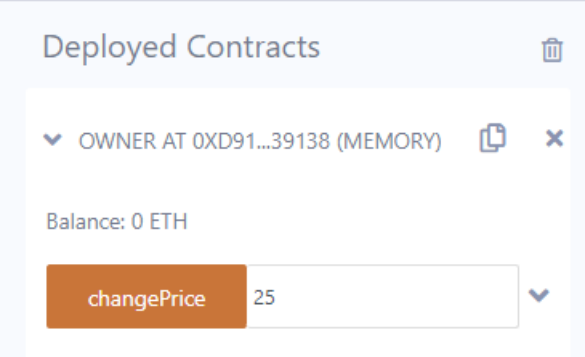
4) Write a solidity smart contract to demonstrate function modifier.

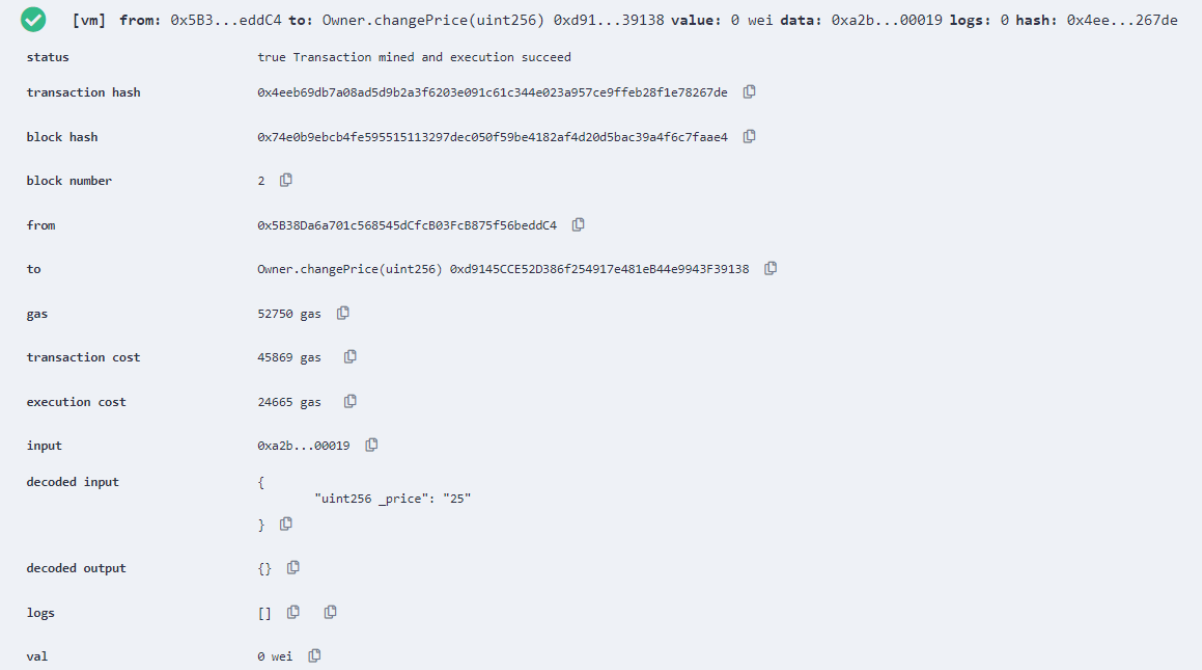
Program:

```
// 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:



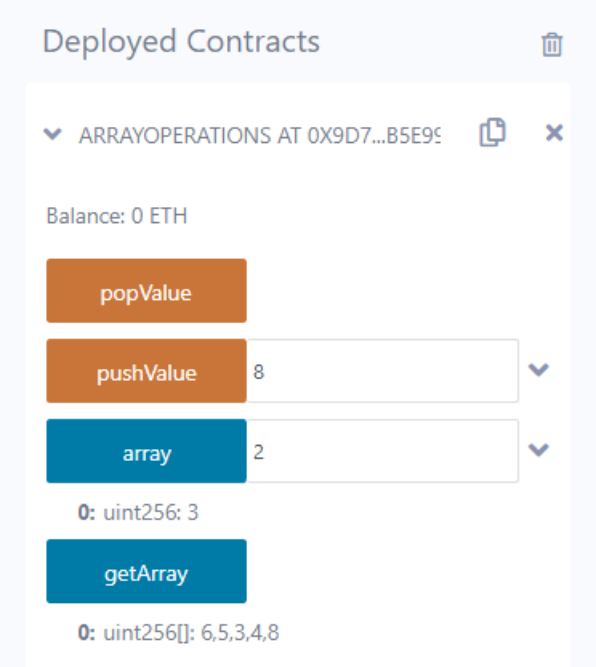


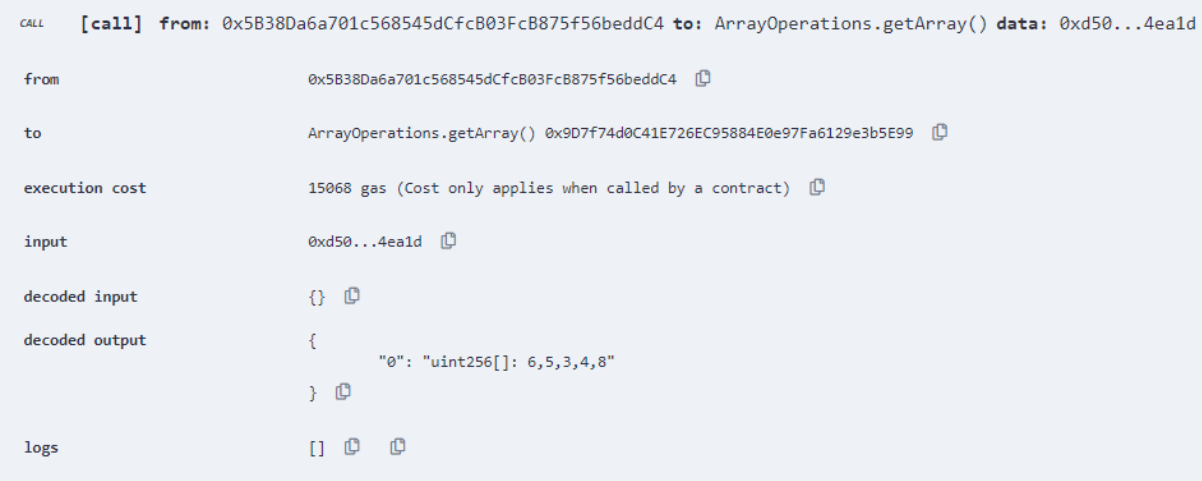
5) Write a Solidity program to demonstrate arrays Push operation and Pop operation.

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:





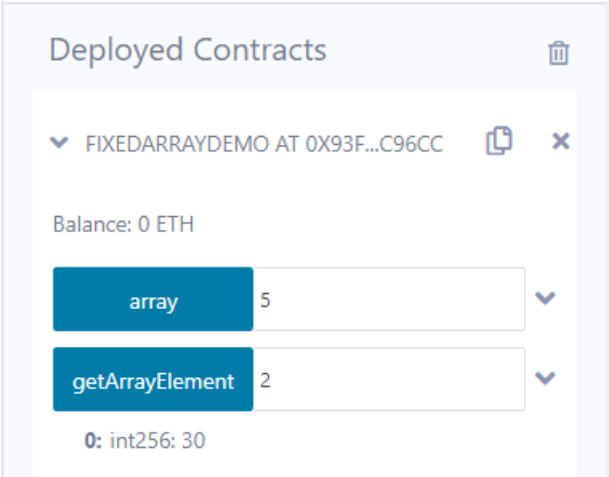
6) Write a Solidity program to demonstrate creating a fixed-size array and access array element.

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:

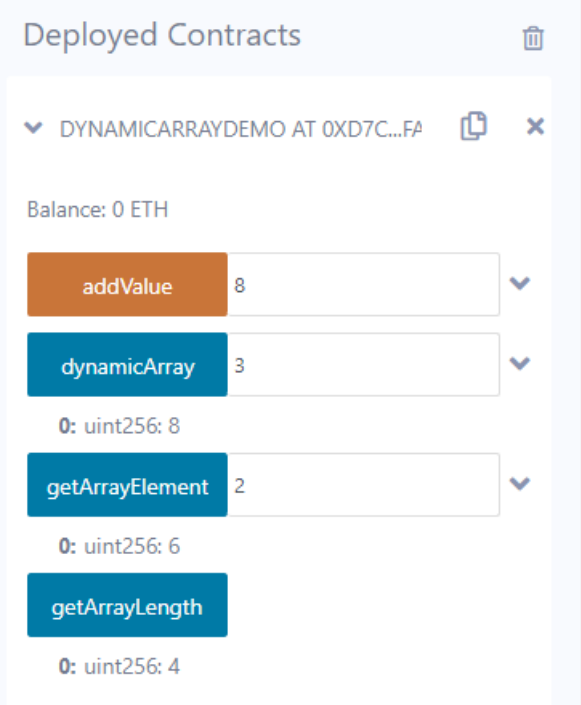


7) Write a Solidity program to demonstrate creating a fixed-size array and access array element.

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:



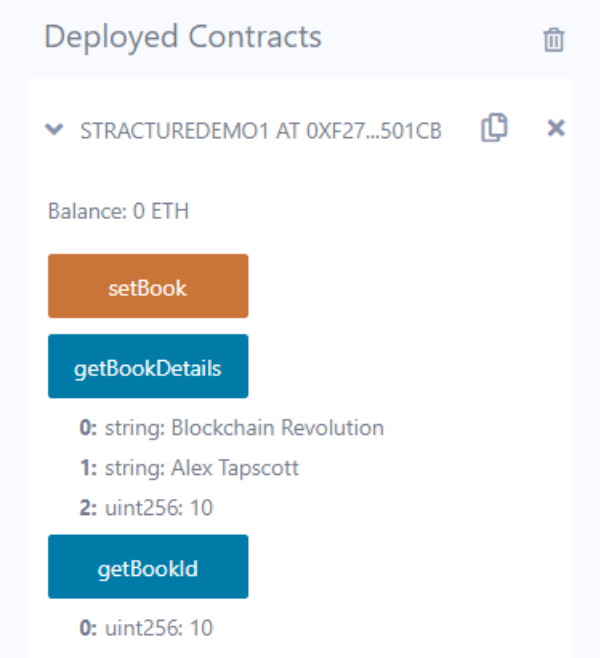
8) Write a solidity smart contract to demonstrate use of structure.

Program:

```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.4.16 <0.8.20;
contract StractureDemo1 {
    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:



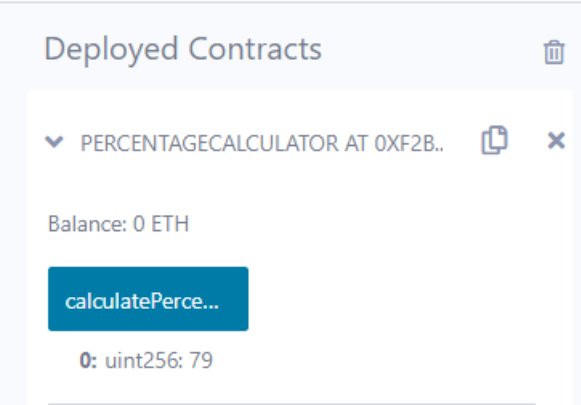
9) Write a solidity smart contract to calculate percentage of marks obtained by students for six subject in final examination.

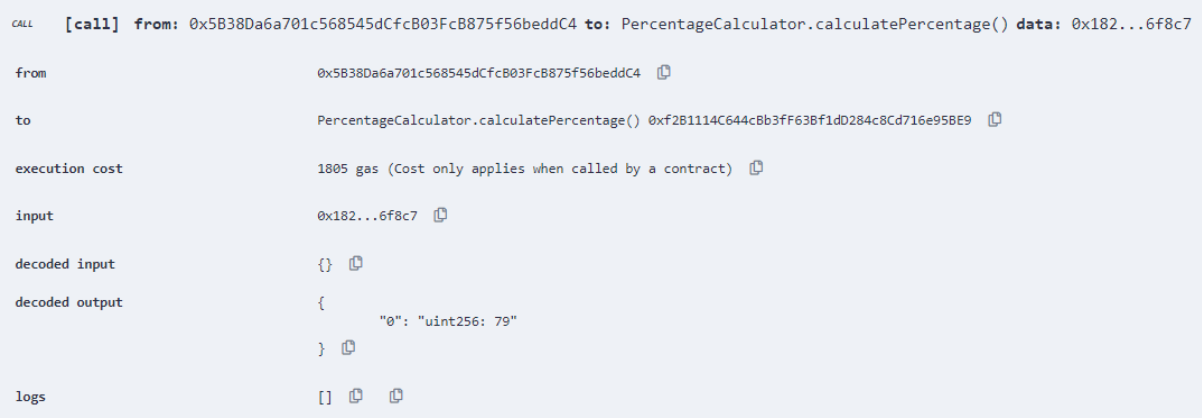
Program:

```
// 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:



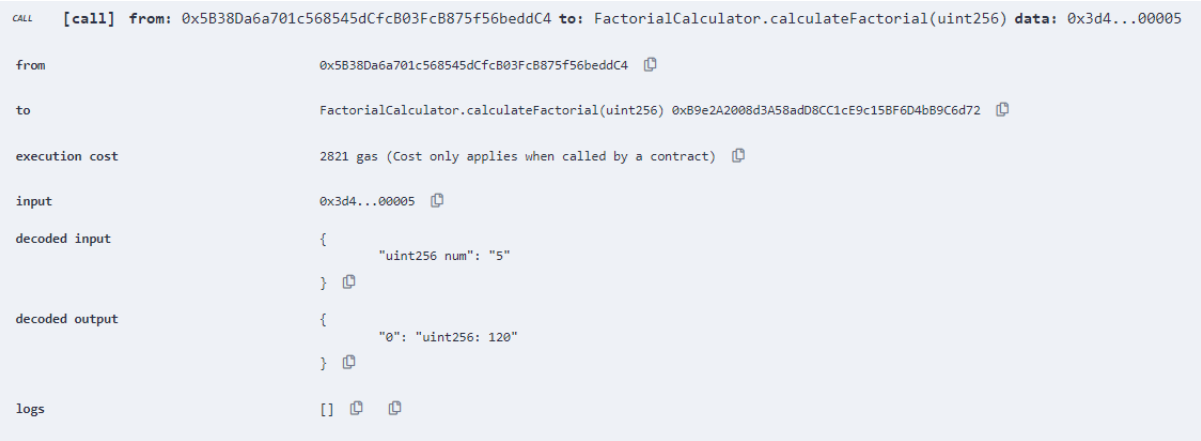


10) Write a solidity smart contract to find the factorial of entered number.

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;
    }
}
```

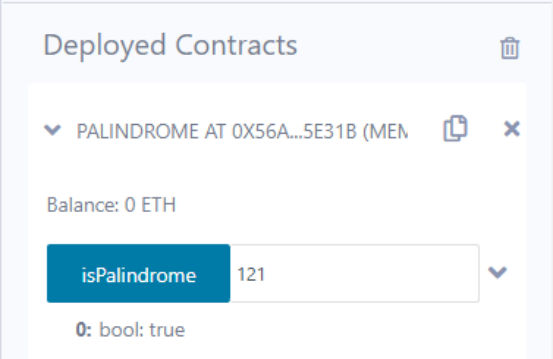
Output:



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:



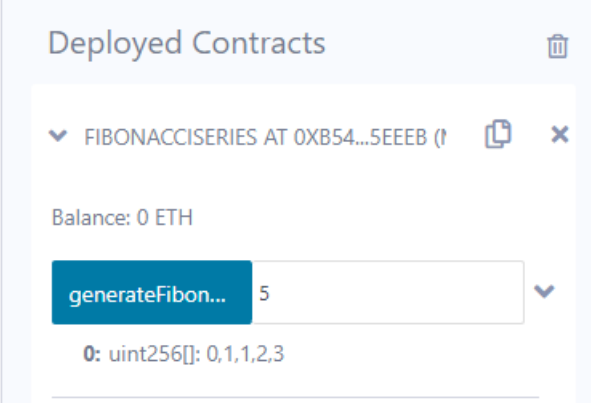

```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Palindrome.isPalindrome(uint256) data: 0x041...00079

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      Palindrome.isPalindrome(uint256) 0x56a2777e796eF23399e9E1d791E1A0410a75E31b

execution cost          3241 gas (Cost only applies when called by a contract)

input                   0x041...00079

decoded input           {
                            "uint256 num": "121"
                        }

decoded output          {
                            "0": "bool: true"
                        }

logs                    []
```

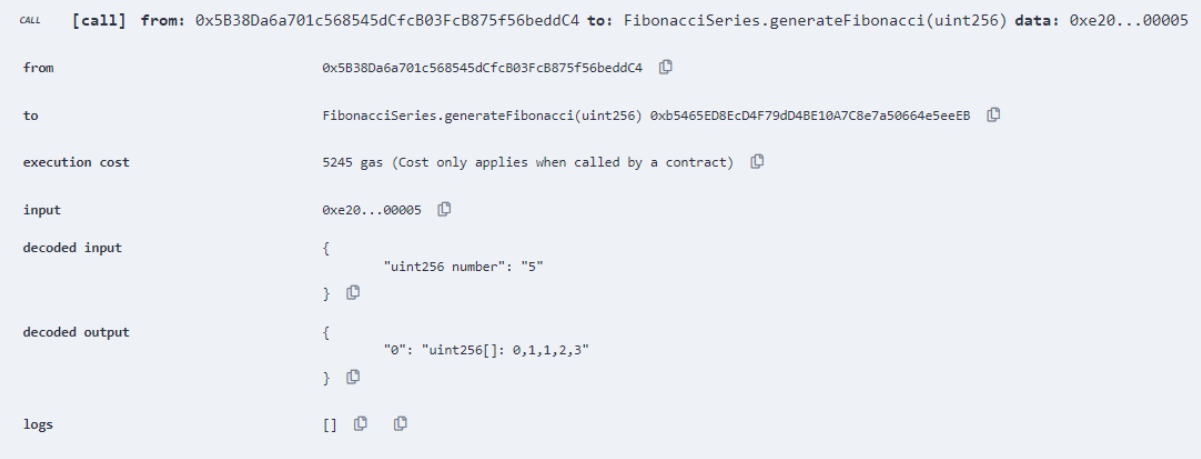
12) Write a solidity smart contract to generate Fibonacci Series up to given number.

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:



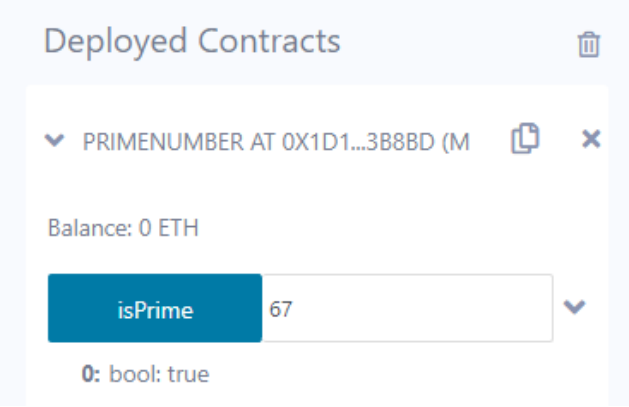


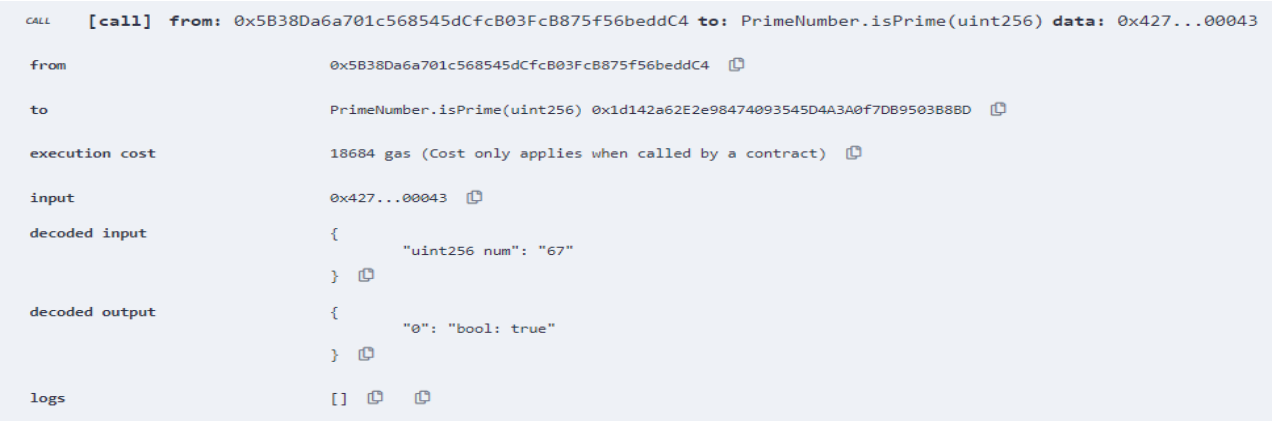
13) Write a solidity smart contract to check whether entered number is prime number or not.

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:



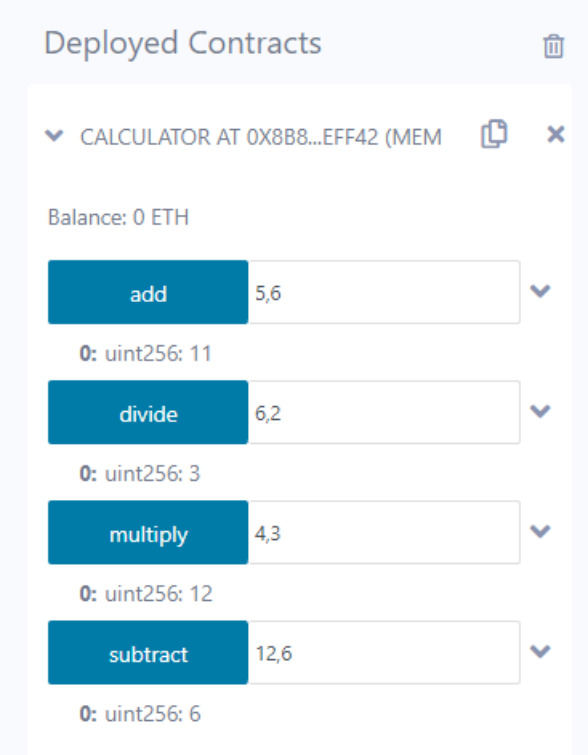


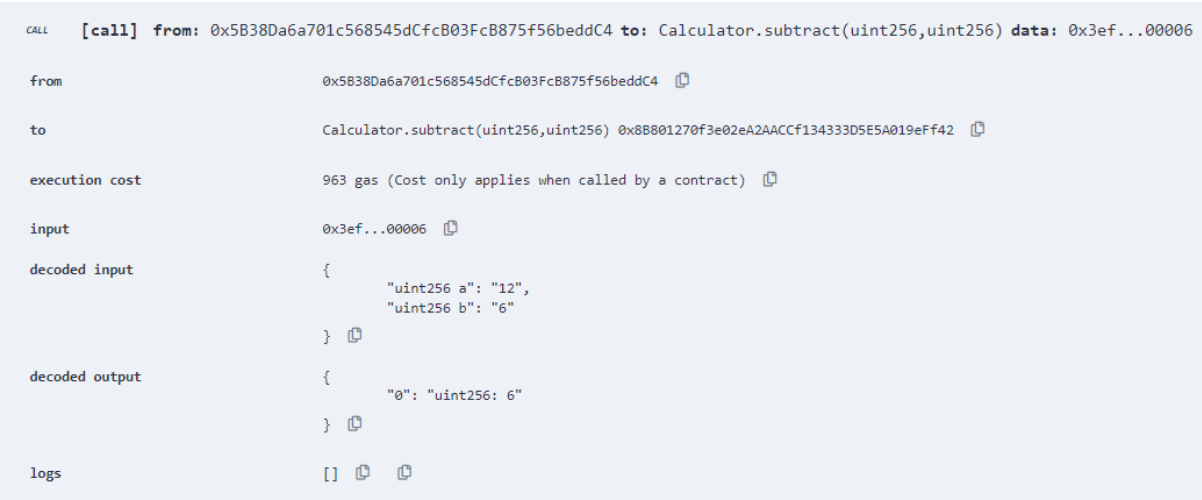
14) Write a solidity smart contract to create arithmetic calculator which includes functions for operations addition, subtraction, multiplication, division etc.

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:





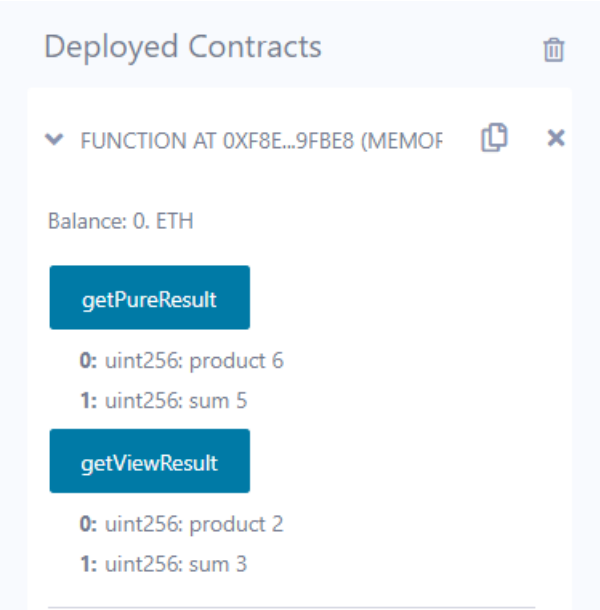
15) Write a solidity smart contract to demonstrate view function and pure function.

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:

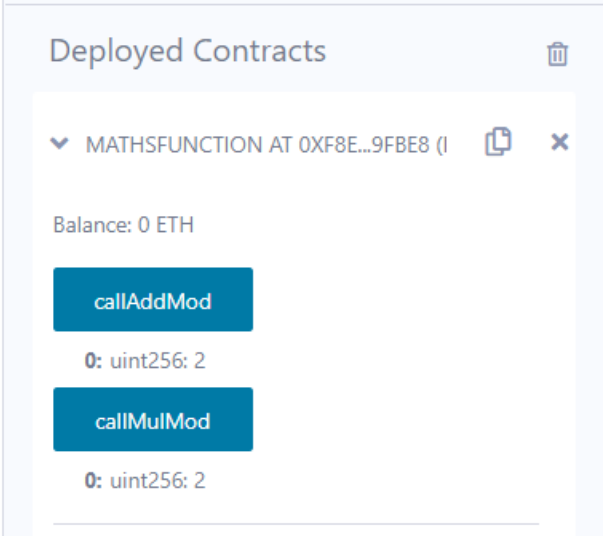


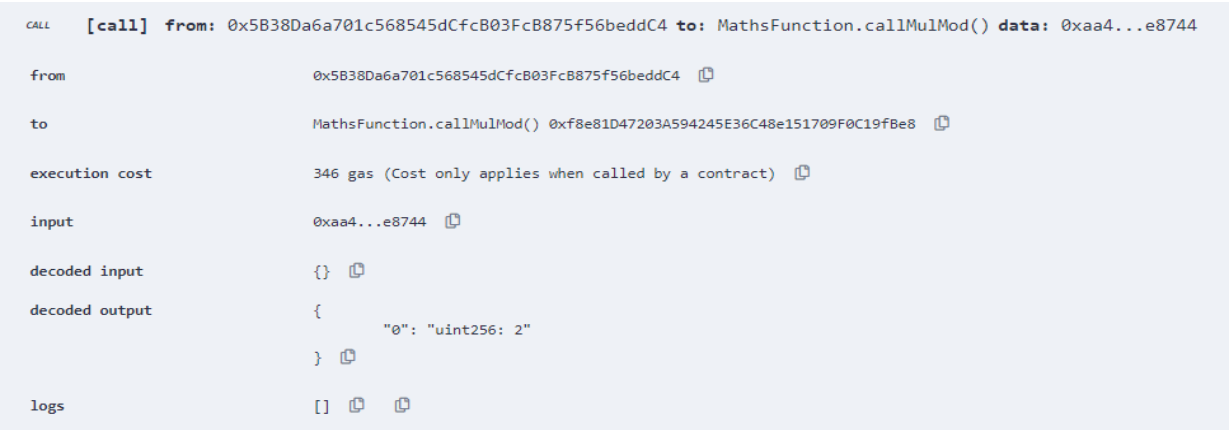
16) Write a solidity smart contract to demonstrate inbuilt mathematical functions.

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:





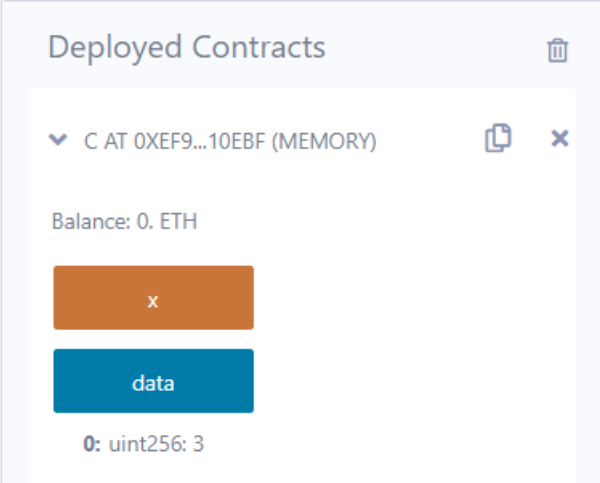
17) Write a solidity smart contract to demonstrate inheritance in contract.

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:



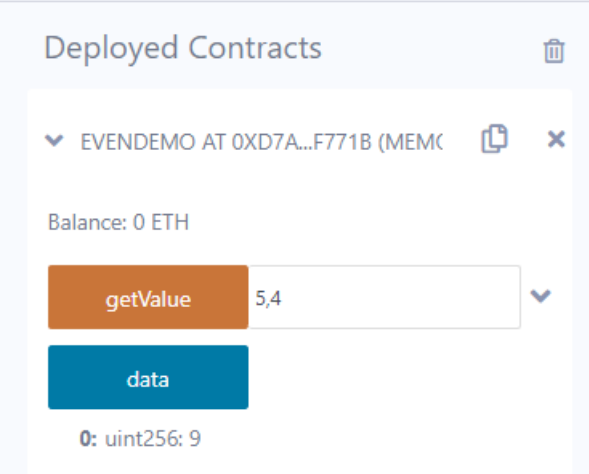
18) Write a solidity smart contract to demonstrate events.

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:




```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: EvenDemo.data() data: 0x73d...4a13a

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      EvenDemo.data() 0xD7ACd2a9FD159E69Bb102A1ca21C9a3e3A5F771B

execution cost          2429 gas (Cost only applies when called by a contract)

input                   0x73d...4a13a

decoded input           {}

decoded output          {
                            "0": "uint256: 9"
                        }

logs                    []
```

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

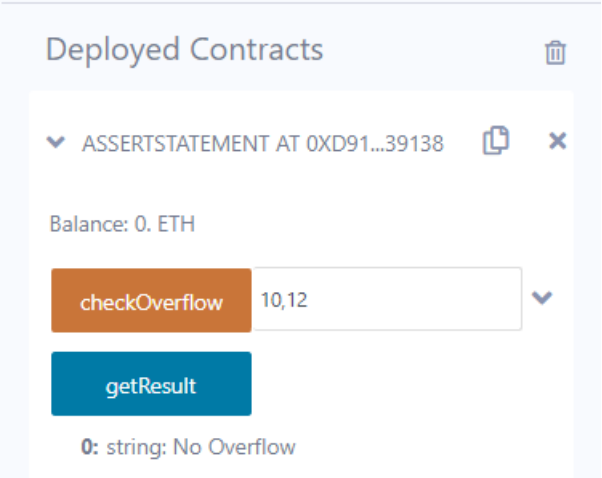
Program:

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"; }
    }
}
```

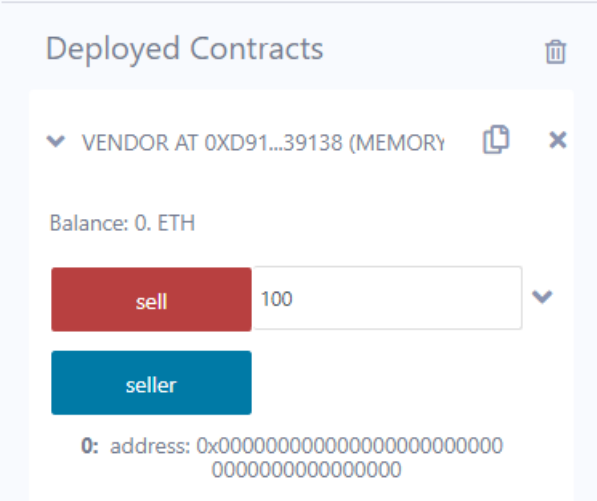
Output:



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:




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

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      Vendor.seller() 0xd9145CCE52D386f254917e481eB44e9943F39138

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

input                   0x085...51a53

decoded input           {}

decoded output          {
                            "0": "address: 0x0000000000000000000000000000000000000000"
                        }

logs                    []
```

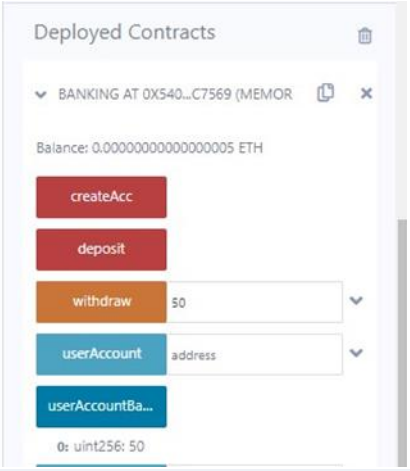
20) Write a solidity smart contract for Bank Account which provides operations such as check account balance, withdraw amount and deposit amount etc.

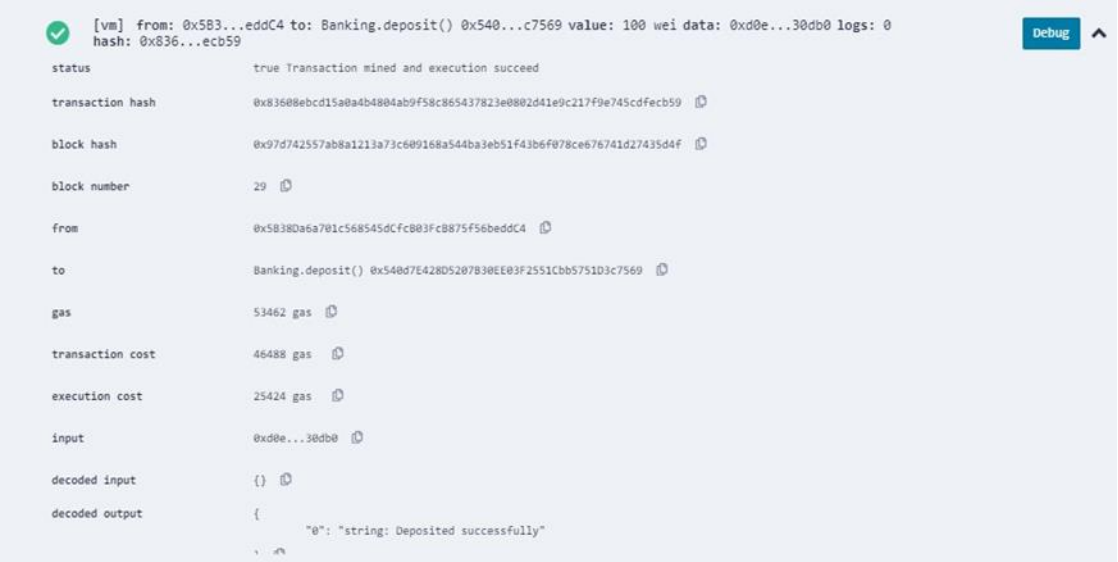
Program:

```
// 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:





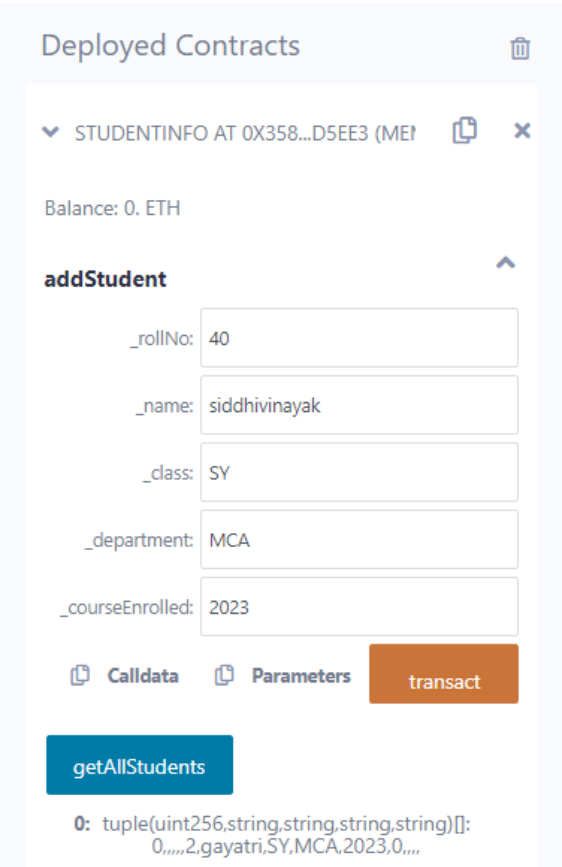
21) Write a program in solidity to create a structured student with Roll no, Name,Class, Department, Course enrolled as variables.

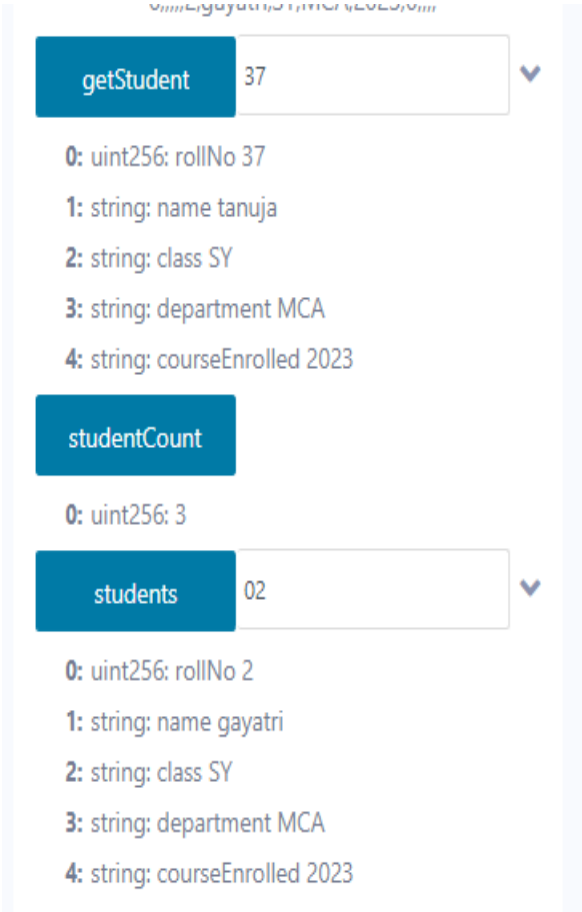
1)Add information of 5 students

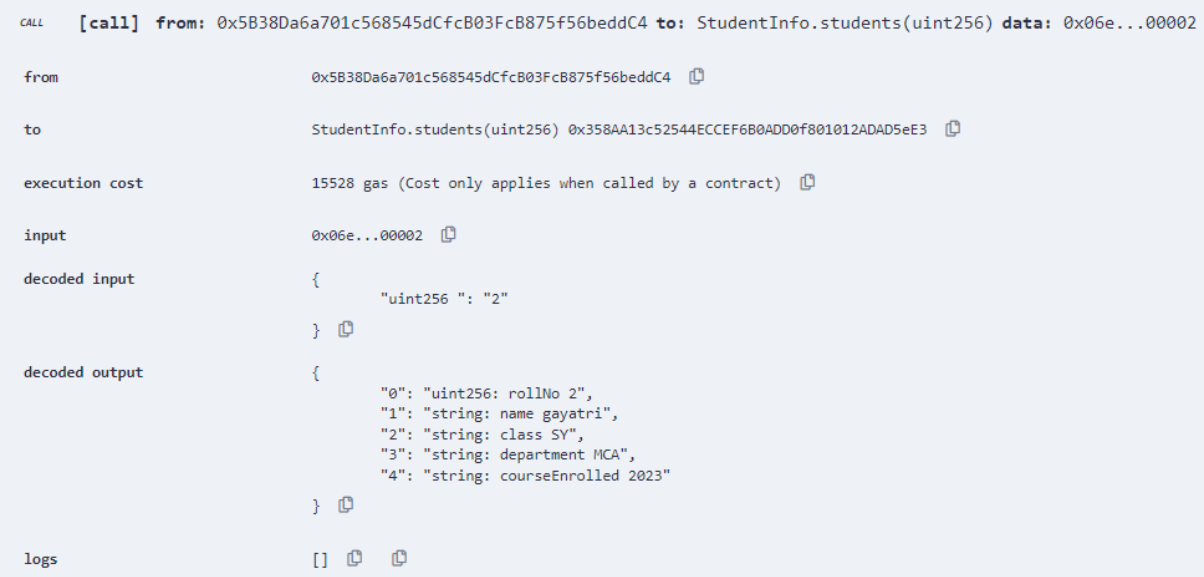
2)Search for a student using Roll no
3)Display all Information
Program:

```
// 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:



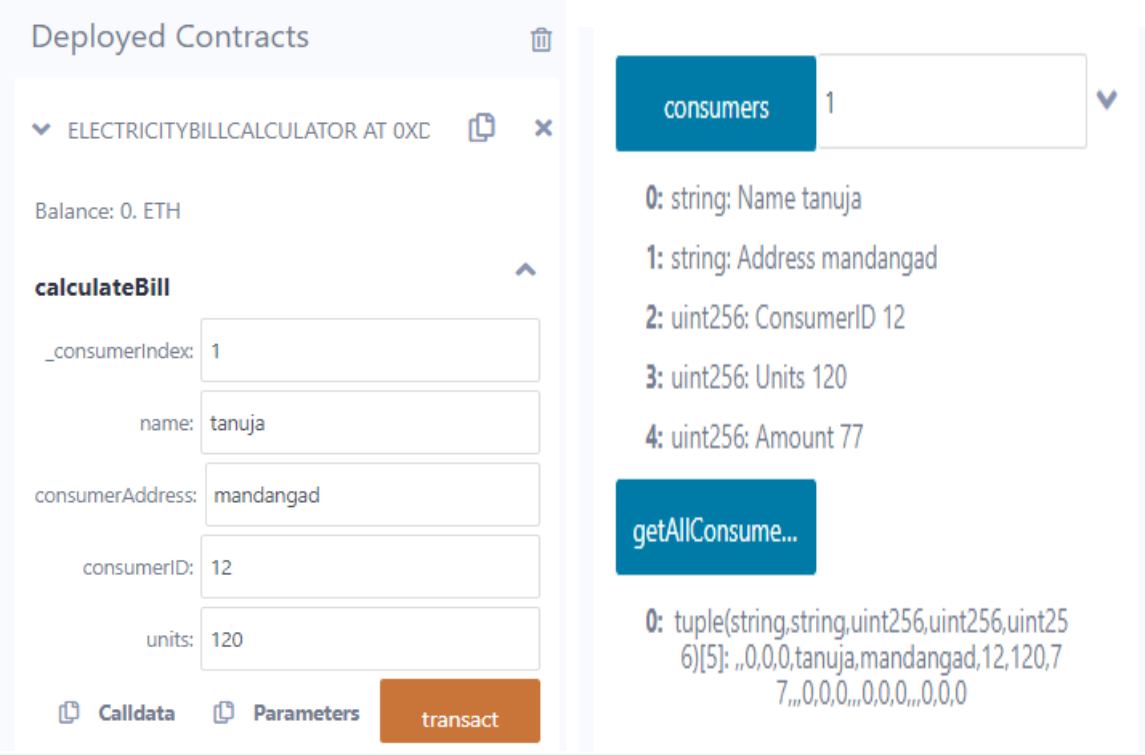




22) Create a structure Consumer with Name , Address, Consumer ID, Units and Amount as members. Write a program in solidity to calculate the total electricity bill according to the given condition:
For first 50 units Rs. 0.50/unit. For next 100 units Rs. 0.75/unit. For next 100 units Rs. 1.20/unit. For 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:




```
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() 0xd9145CCE52D386f254917e481eB44e9943F39138
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,,,0,0,0"
                }
logs            []
```