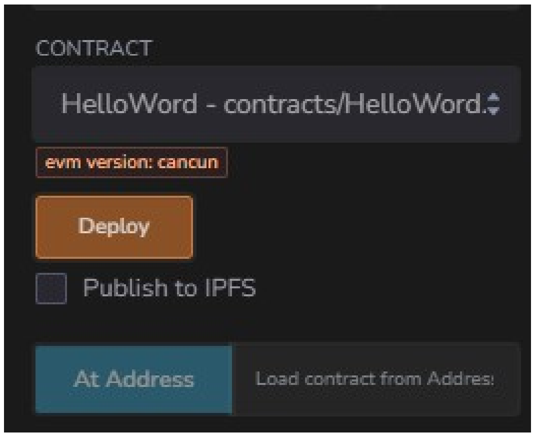
Exercise:

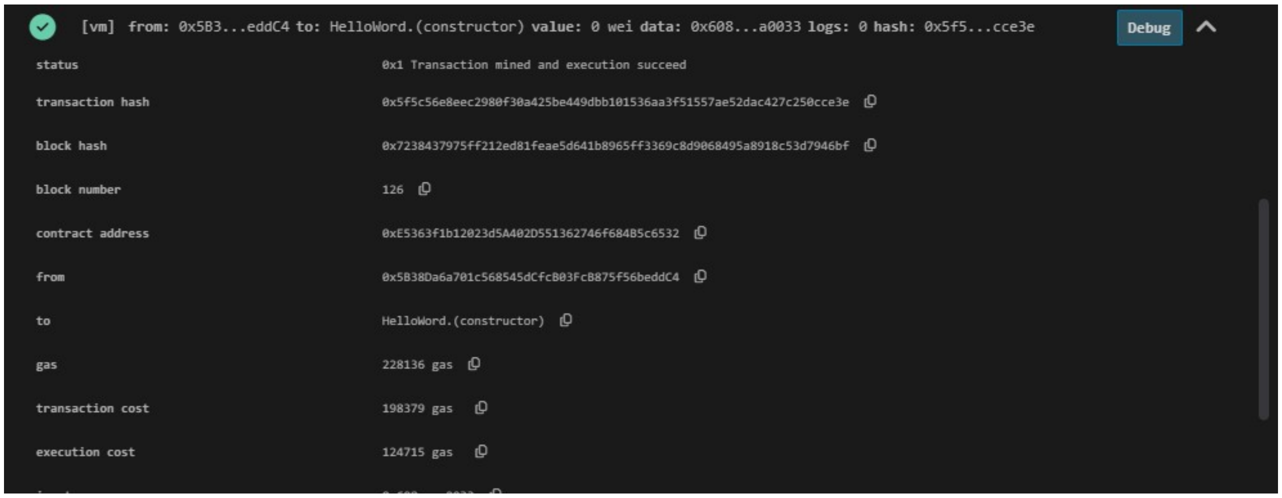
1. Write a solidity smart contract to display hello world message.

Program:

```
//SPDX-License-Identifier: MIT

pragma solidity  >=0.5.0<0.8.27;

contract HelloWord {

 string message = "Hello World";
function get()public view returns (string memory) {
return message;
            }
}
```

Output:





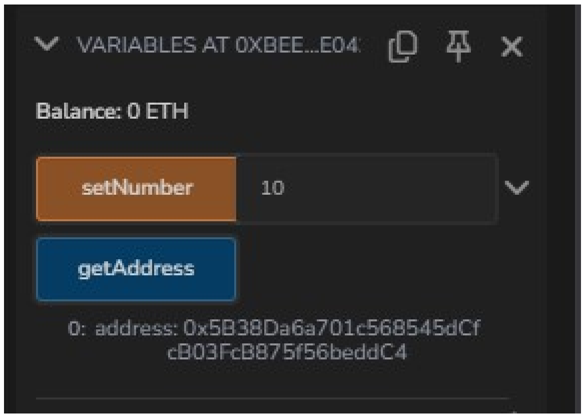
```
[vm] from: 0x5B3...eddC4 to: HelloWord.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x5f5...cce3e
status              0x1 Transaction mined and execution succeed
transaction hash    0x5f5c56e8eec2980f30a425be449dbb101536aa3f51557ae52dac427c250cce3e
block hash          0x7238437975ff212ed81feae5d641b8965ff3369c8d9068495a8918c53d7946bf
block number        126
contract address    0xE5363f1b12023d5A402D551362746f684B5c6532
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  HelloWord.(constructor)
gas                 228136 gas
transaction cost    198379 gas
execution cost      124715 gas
```

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

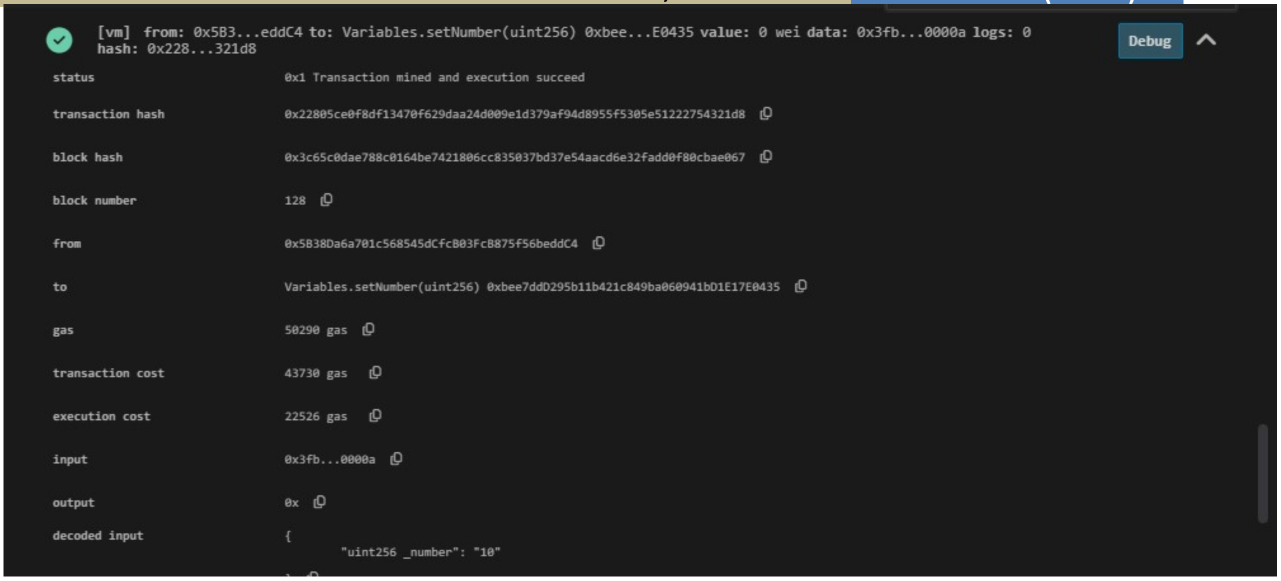
Program:

```
  //SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract Variables{

  //number state Variable
  uint256 number;



  function setNumber(uint256 _number) public {
    //tempNumber localVariable
    uint256 tempNumber = _number;
   number = tempNumber;
  }

  function getAddress() public view returns (address){
   return msg.sender;
  }

}
```

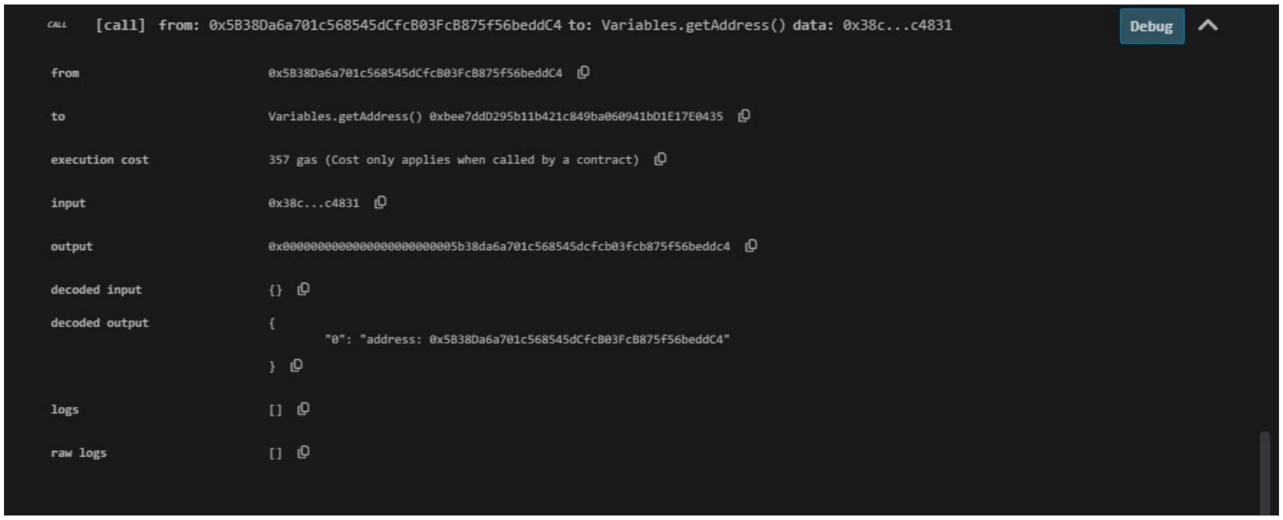
Output:



```
Balance: 0 ETH
setNumber 10
0: address: 0x5B38Da6a701c568545dCf
cB03FcB875f56beddC4
```



```
[vm] from: 0x5B3...eddC4 to: Variables.setNumber(uint256) 0xbee...E0435 value: 0 wei data: 0x3fb...0000a logs: 0
hash: 0x228...321d8
status                  0x1 Transaction mined and execution succeed
transaction hash        0x22805ce0f8df13470f629daa24d009e1d379af94d8955f5305e51222754321d8
block hash              0x3c65c0dae788c0164be7421806cc835037bd37e54aacd6e32fadd0f80cbae067
block number            128
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Variables.setNumber(uint256) 0xbee7ddD295b11b421c849ba060941bD1E17E0435
gas                     50290 gas
transaction cost        43730 gas
execution cost          22526 gas
input                   0x3fb...0000a
output                  0x
decoded input           {
                               "uint256 _number": "10"
```



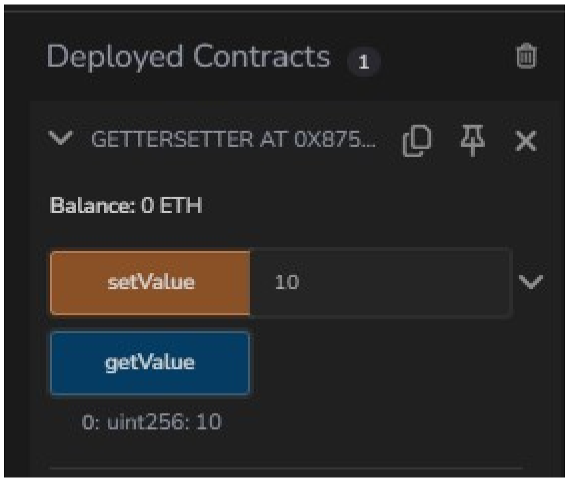
```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Variables.getAddress() data: 0x38c...c4831
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Variables.getAddress() 0xbee7ddD295b11b421c849ba060941bD1E17E0435
execution cost          357 gas (Cost only applies when called by a contract)
input                   0x38c...c4831
output                  0x0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4
decoded input           {}
decoded output          {
                               "0": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                        }
logs                    []
raw logs                []
```

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

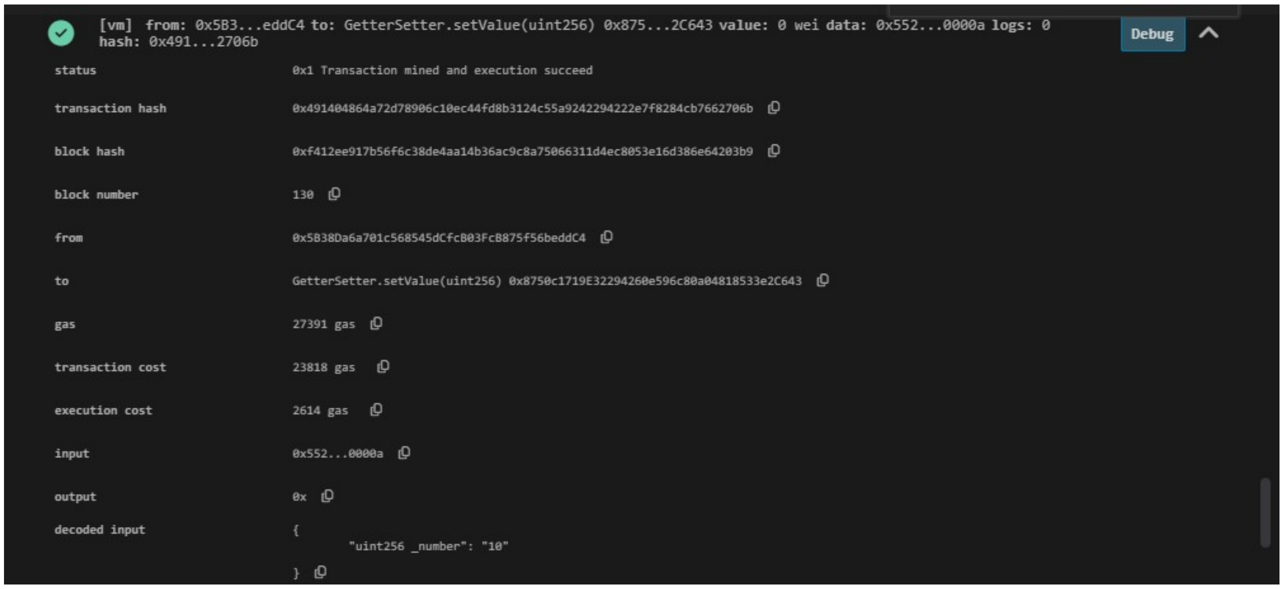
Program:

```
    //SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract GetterSetter{
  uint256 private  number;

  constructor(uint256 _number){
    number =_number;
  }

  function setValue(uint256 _number) public {
    number = _number;
  }

  function getValue() public view returns (uint256 ){
        return number;
    }


}
```

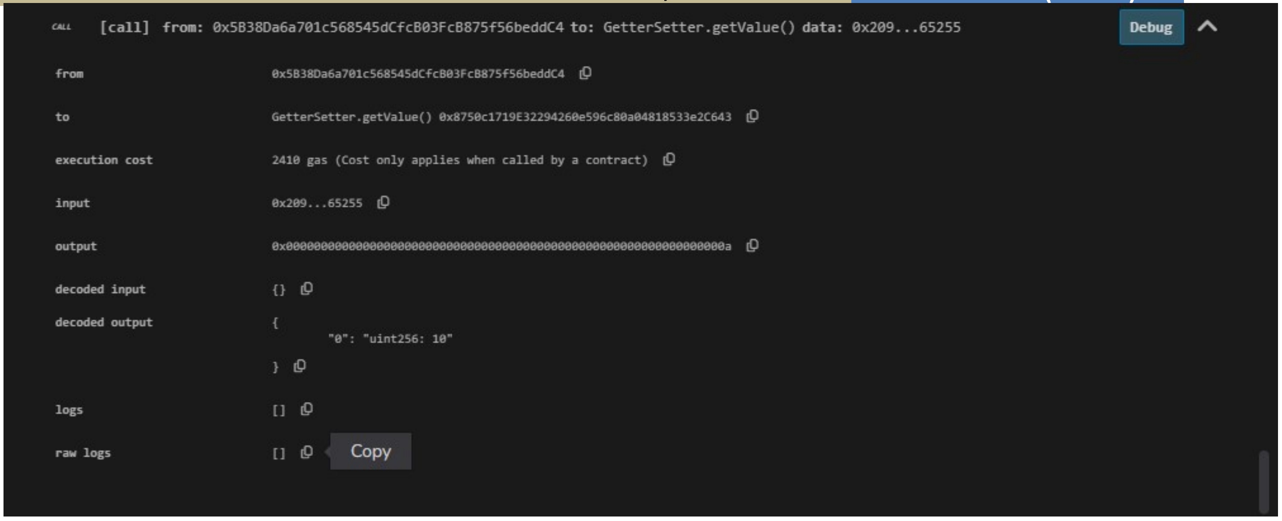
Output:



```
0: uint256: 10
```



```
[vm] from: 0x5B3...eddC4 to: GetterSetter.setValue(uint256) 0x875...2C643 value: 0 wei data: 0x552...0000a logs: 0
hash: 0x491...2706b
status              0x1 Transaction mined and execution succeed
transaction hash    0x491404864a72d78906c10ec44fd8b3124c55a9242294222e7f8284cb7662706b
block hash          0xf412ee917b56f6c38de4aa14b36ac9c8a75066311d4ec8053e16d386e64203b9
block number        130
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  GetterSetter.setValue(uint256) 0x8750c1719E32294260e596c80a04818533e2C643
gas                 27391 gas
transaction cost    23818 gas
execution cost      2614 gas
input               0x552...0000a
output              0x
decoded input       {
                            "uint256 _number": "10"
                    }
```



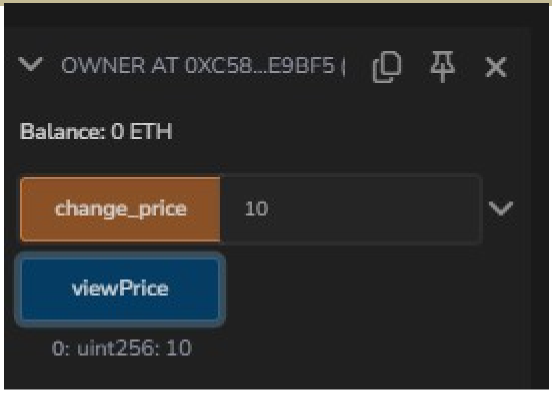
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: GetterSetter.getValue() data: 0x209...65255

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                GetterSetter.getValue() 0x8750c1719E32294260e596c80a04818533e2C643
execution cost    2410 gas (Cost only applies when called by a contract)
input             0x209...65255
output            0x000000000000000000000000000000000000000000000000000000000000000a
decoded input     {}
decoded output    {
                          "0": "uint256: 10"
                  }
logs              []
raw logs          []
```

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

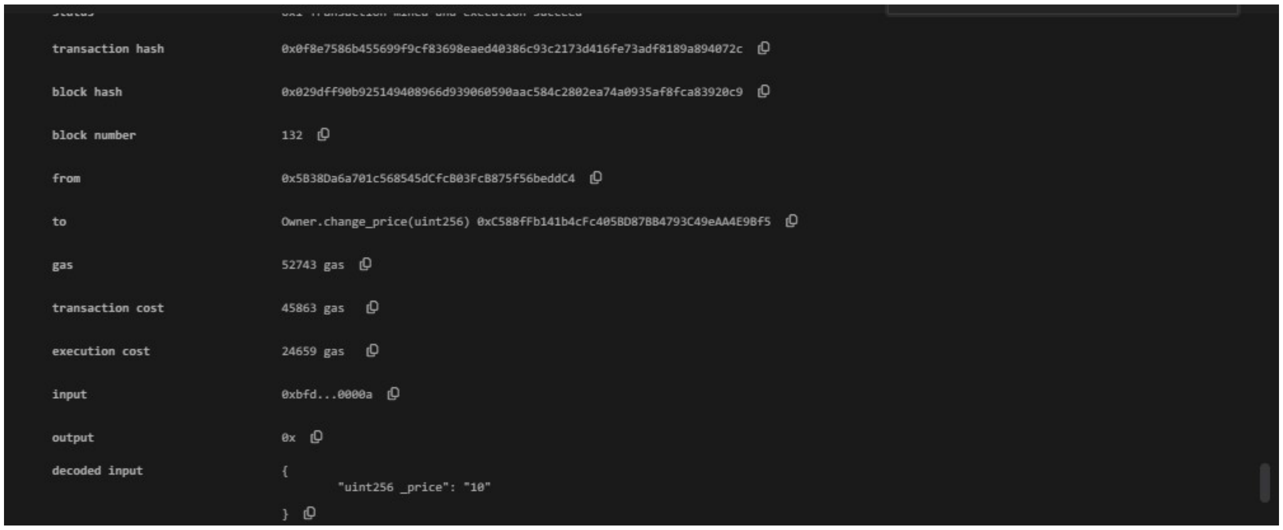
Program:

```
    //SPDX-License-Identifier: MIT

pragma solidity  >=0.5.0<0.8.27;

contract Owner {
  address owner;
  uint price;

  constructor(){
    owner = msg.sender;
  }
  //if function becomes true then and then only it correct then it executes
  modifier onlyOwner{
    require(msg.sender == owner,"Only owner is allowed to modify the price");
    _;
  }

  function change_price(uint _price) public onlyOwner{
    price = _price;
  }

  function viewPrice() public  view returns (uint){
    return price;
  }
}
```

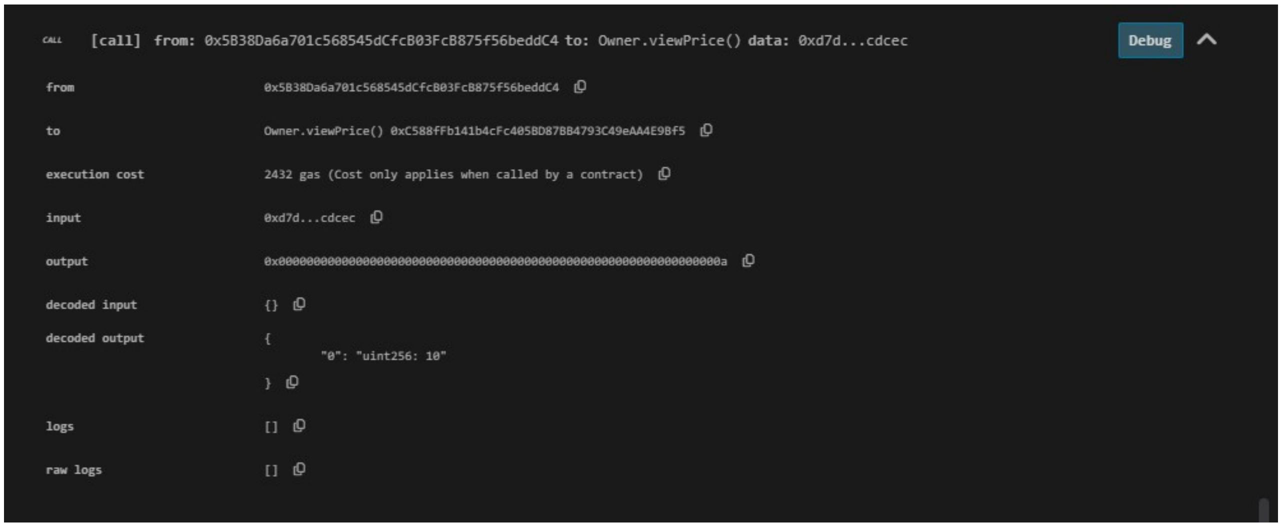
Output:



```
0: uint256: 10
```



```
transaction hash        0x0f8e7586b455699f9cf83698eaed40386c93c2173d416fe73adf8189a894072c

block hash              0x029dff90b925149408966d939060590aac584c2802ea74a0935af8fca83920c9

block number            132

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      Owner.change_price(uint256) 0xC588fFb141b4cFc405BD87BB4793C49eAA4E9Bf5

gas                     52743 gas

transaction cost        45863 gas

execution cost          24659 gas

input                   0xbfd...0000a

output                  0x

decoded input           {
                                "uint256 _price": "10"
                        }
```



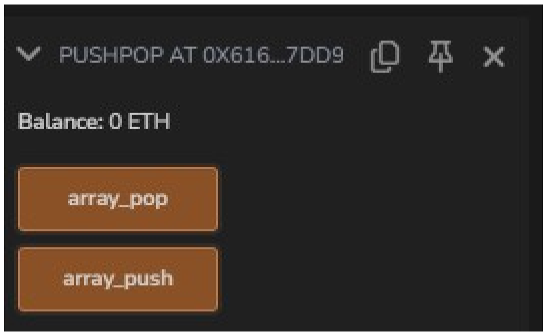
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Owner.viewPrice() data: 0xd7d...cdcec

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      Owner.viewPrice() 0xC588fFb141b4cFc405BD87BB4793C49eAA4E9Bf5

execution cost          2432 gas (Cost only applies when called by a contract)

input                   0xd7d...cdcec

output                  0x000000000000000000000000000000000000000000000000000000000000000a

decoded input           {}

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

logs                    []

raw logs                []
```

5. Write a Solidity program to demonstrate arrays Push operation and Pop operation.
   Program:
   //SPDX-License-Identifier: MIT

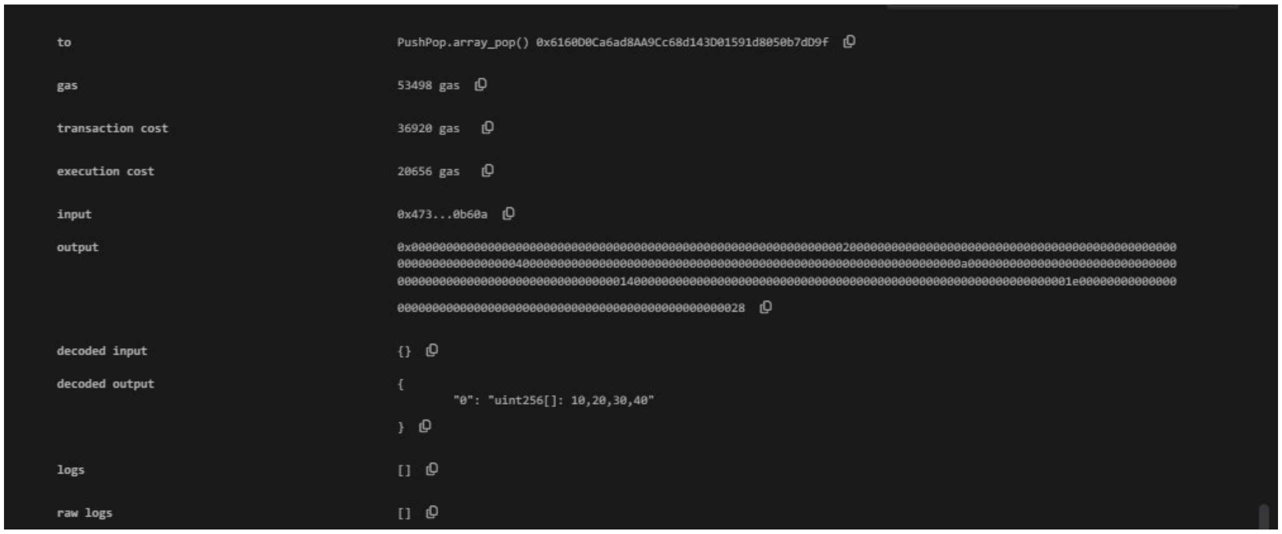
```
pragma solidity  >=0.5.0<0.8.27;

contract PushPop{
   uint[] data=[10,20,30,40,50];
```

```
    function array_push() public  returns (uint[] memory){
        data.push(60);
        data.push(70);
        data.push(80);

        return data;
    }

    function array_pop() public returns  (uint[] memory){
        data.pop();
        return data;
    }
}
```

Output:

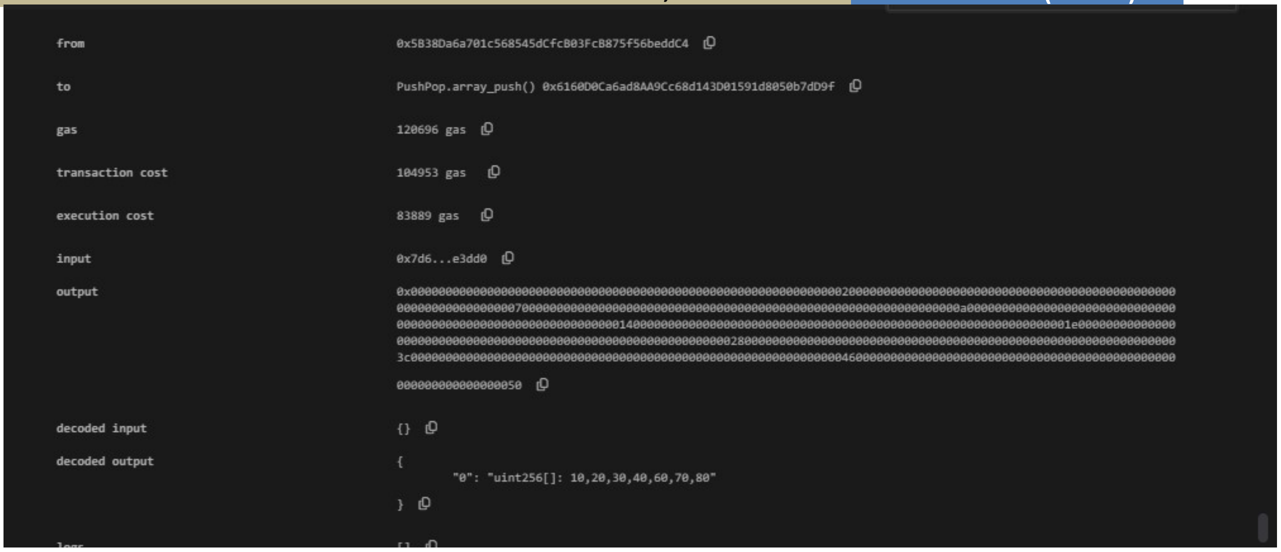


Pop:



```
to                  PushPop.array_pop() 0x6160D0Ca6ad8AA9Cc68d143D01591d8050b7dD9f

gas                 53498 gas

transaction cost    36920 gas

execution cost      20656 gas

input               0x473...0b60a

output              0x00000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000
                    0000000000000000400000000000000000000000000000000000000000000000000000000000000a000000000000000000000000000000
                    00000000000000000000000000000000140000000000000000000000000000000000000000000000000000000000001e00000000000000
                    00000000000000000000000000000000000000000000000028

decoded input       {}

decoded output      {
                           "0": "uint256[]: 10,20,30,40"
                    }

logs                []

raw logs            []
```

Push:

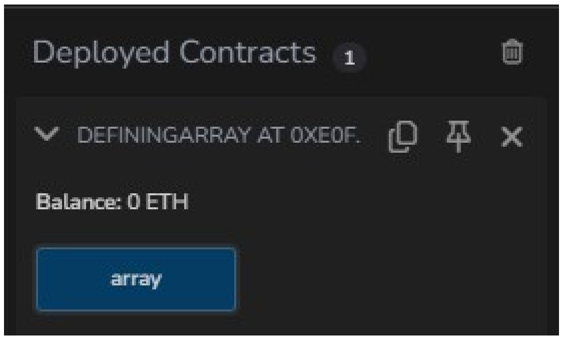


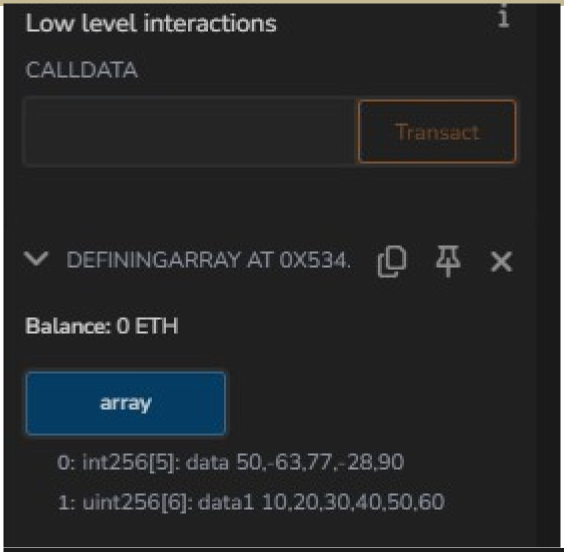
```
from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   PushPop.array_push() 0x6160D0Ca6ad8AA9Cc68d143D01591d8050b7dD9f
gas                  120696 gas
transaction cost     104953 gas
execution cost       83889 gas
input                0x7d6...e3dd0
output               0x00000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000
                     0000000000000000070000000000000000000000000000000000000000000000000000000000000000a000000000000000000000000000000
                     0000000000000000000000000000000014000000000000000000000000000000000000000000000000000000000000001e00000000000000
                     0000000000000000000000000000000000000000000000002800000000000000000000000000000000000000000000000000000000000000
                     3c00000000000000000000000000000000000000000000000000000000000000460000000000000000000000000000000000000000000000
                     000000000000000050
decoded input        {}
decoded output       {
                             "0": "uint256[]: 10,20,30,40,60,70,80"
                     }
logs                 []
```

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

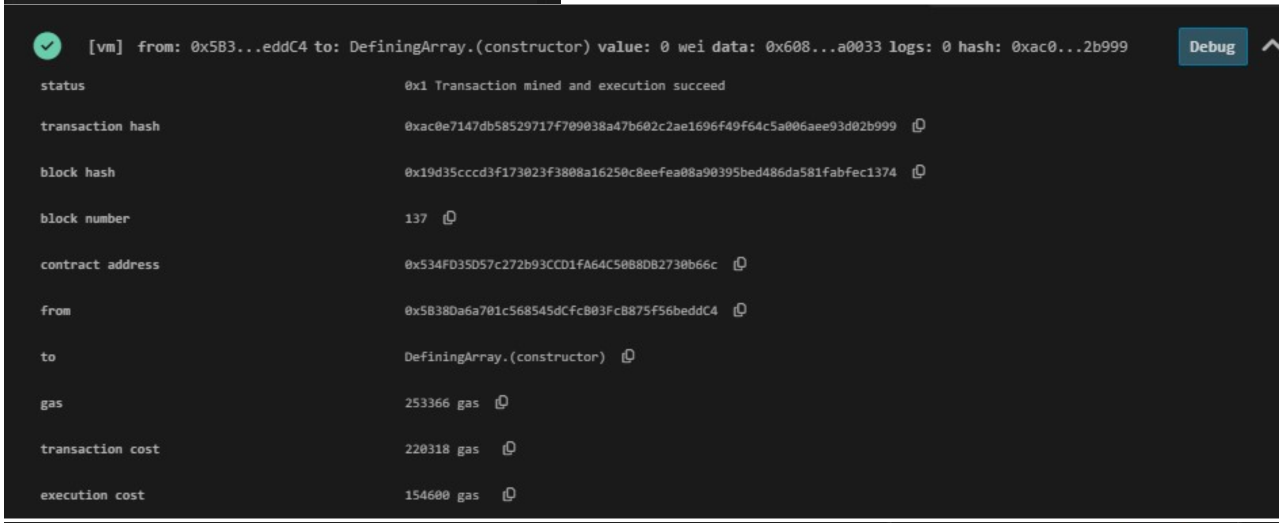
```
// SPDX-License-Identifier: MIT
pragma solidity  >=0.5.0<0.8.27;

contract DefiningArray{
    uint[6] data1;
    function array() pure public returns (int[5] memory data, uint[6] memory data1){
              int[5] memory data =[int(50),-63,77,-28,90];
            data1 =[uint(10),20,30,40,50,60];
            return (data,data1);
    }
}
```

Output:

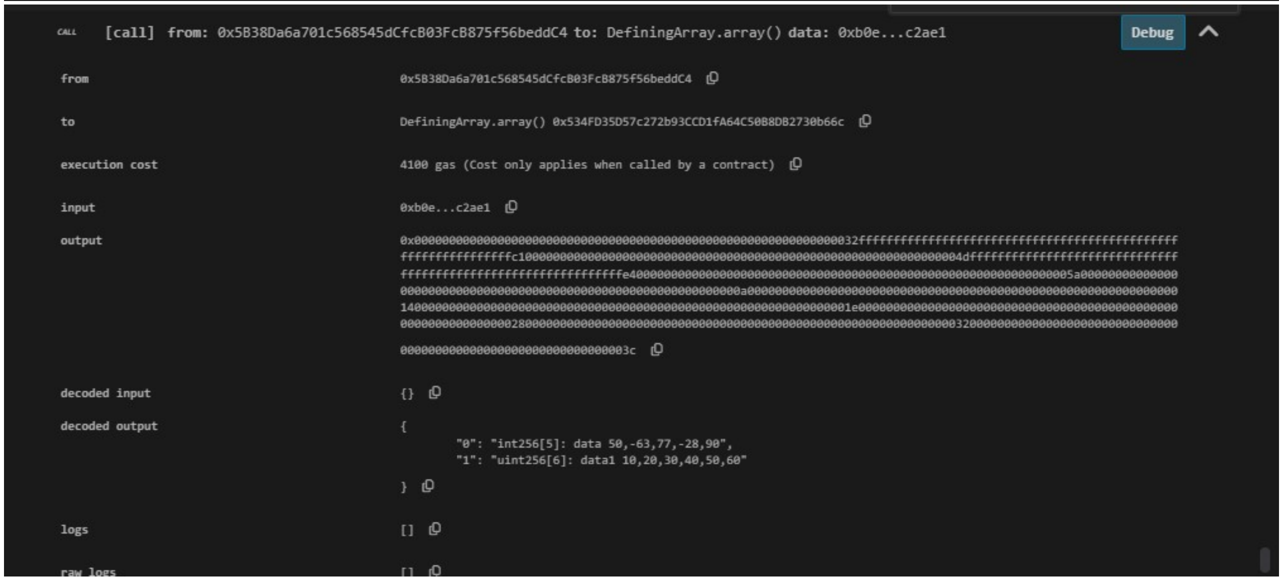




```
0: int256[5]: data 50,-63,77,-28,90
1: uint256[6]: data1 10,20,30,40,50,60
```



```
[vm] from: 0x5B3...eddC4 to: DefiningArray.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xac0...2b999
status                  0x1 Transaction mined and execution succeed
transaction hash        0xac0e7147db58529717f709038a47b602c2ae1696f49f64c5a006aee93d02b999
block hash              0x19d35cccd3f173023f3808a16250c8eefea08a90395bed486da581fabfec1374
block number            137
contract address        0x534FD35D57c272b93CCD1fA64C50B8DB2730b66c
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      DefiningArray.(constructor)
gas                     253366 gas
transaction cost        220318 gas
execution cost          154600 gas
```



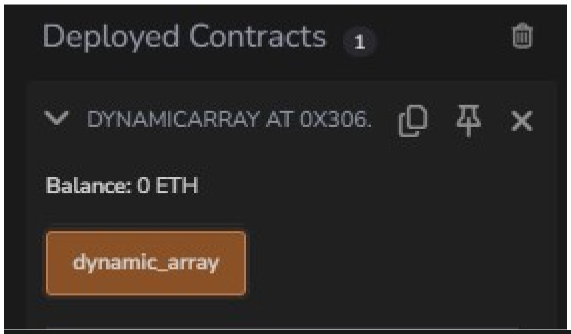
```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: DefiningArray.array() data: 0xb0e...c2ae1
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      DefiningArray.array() 0x534FD35D57c272b93CCD1fA64C50B8DB2730b66c
execution cost          4100 gas (Cost only applies when called by a contract)
input                   0xb0e...c2ae1
output                  0x0000000000000000000000000000000000000000000000000000000000000032ffffffffffffffffffffffffffffffffffffffffffffff
                        ffffffffffffffffc1000000000000000000000000000000000000000000000000000000000000004dffffffffffffffffffffffffffffff
                        ffffffffffffffffffffffffffffffffe4000000000000000000000000000000000000000000000000000000000000005a00000000000000
                        000000000000000000000000000000000000000000000000000a00000000000000000000000000000000000000000000000000000000000000
                        140000000000000000000000000000000000000000000000000000000000000001e0000000000000000000000000000000000000000000000
                        00000000000000002800000000000000000000000000000000000000000000000000000000000000320000000000000000000000000000000
                        000000000000000000000000000000003c
decoded input           {}
decoded output          {
                                "0": "int256[5]: data 50,-63,77,-28,90",
                                "1": "uint256[6]: data1 10,20,30,40,50,60"
                        }
logs                    []
raw logs                []
```

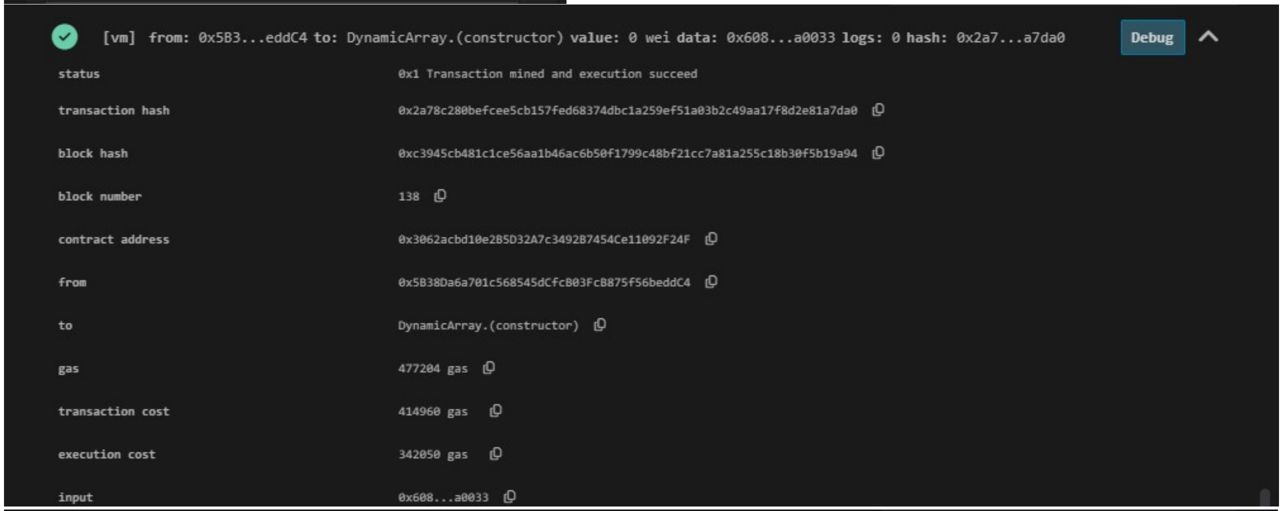
7. Write a Solidity program to demonstrate creating a dynamic array and accessing array element.
Program:
//SPDX-License-Identifier: MIT

pragma solidity >=0.5.0<0.8.27;

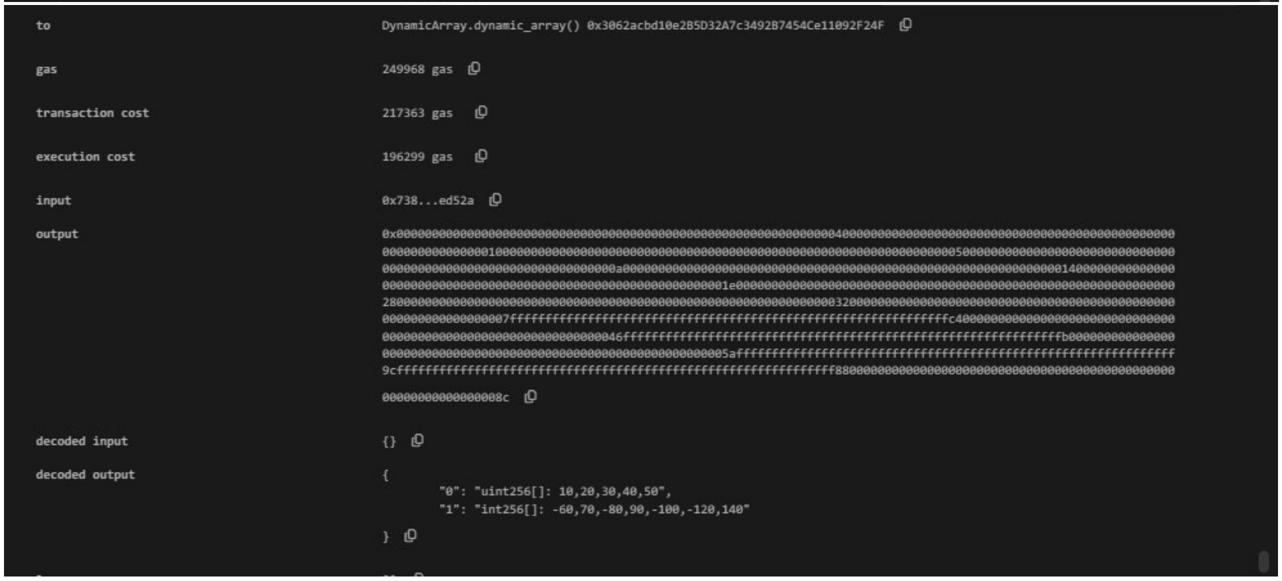
```
contract DynamicArray{
            uint[] data =[10,20,30,40,50];
            int[] data1;
function dynamic_array() public returns(uint[] memory, int[] memory){
             data1= [int(-60), 70,-80,90,-100,-120,140];
            return(data,data1);
            }
}
```

Output:





```
[vm] from: 0x5B3...eddC4 to: DynamicArray.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x2a7...a7da0
status              0x1 Transaction mined and execution succeed
transaction hash    0x2a78c280befcee5cb157fed68374dbc1a259ef51a03b2c49aa17f8d2e81a7da0
block hash          0xc3945cb481c1ce56aa1b46ac6b50f1799c48bf21cc7a81a255c18b30f5b19a94
block number        138
contract address    0x3062acbd10e2B5D32A7c3492B7454Ce11092F24F
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  DynamicArray.(constructor)
gas                 477204 gas
transaction cost    414960 gas
execution cost      342050 gas
input               0x608...a0033
```

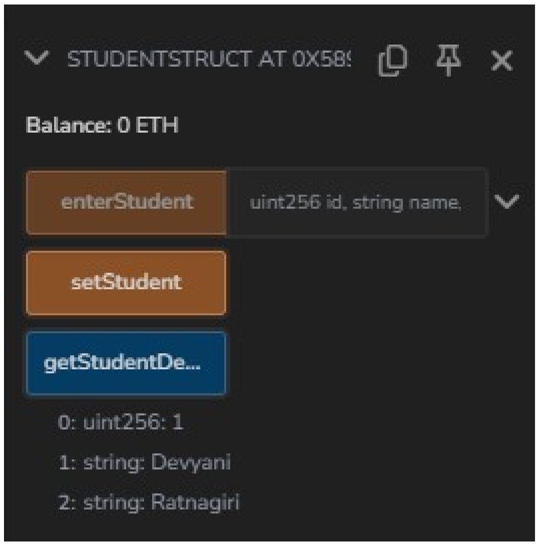


```
to                  DynamicArray.dynamic_array() 0x3062acbd10e2B5D32A7c3492B7454Ce11092F24F
gas                 249968 gas
transaction cost    217363 gas
execution cost      196299 gas
input               0x738...ed52a
output              0x00000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000000000000000000
                    0000000000000001000000000000000000000000000000000000000000000000000000000000000050000000000000000000000000000000
                    000000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000001400000000000000
                    0000000000000000000000000000000000000000000000001e00000000000000000000000000000000000000000000000000000000000000
                    2800000000000000000000000000000000000000000000000000000000000000320000000000000000000000000000000000000000000000
                    00000000000000007ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc4000000000000000000000000000000
                    0000000000000000000000000000000046ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffb000000000000000
                    0000000000000000000000000000000000000000000000005affffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
                    9cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff880000000000000000000000000000000000000000000000
                    000000000000000008c
decoded input       {}
decoded output      {
                            "0": "uint256[]: 10,20,30,40,50",
                            "1": "int256[]: -60,70,-80,90,-100,-120,140"
                    }
```

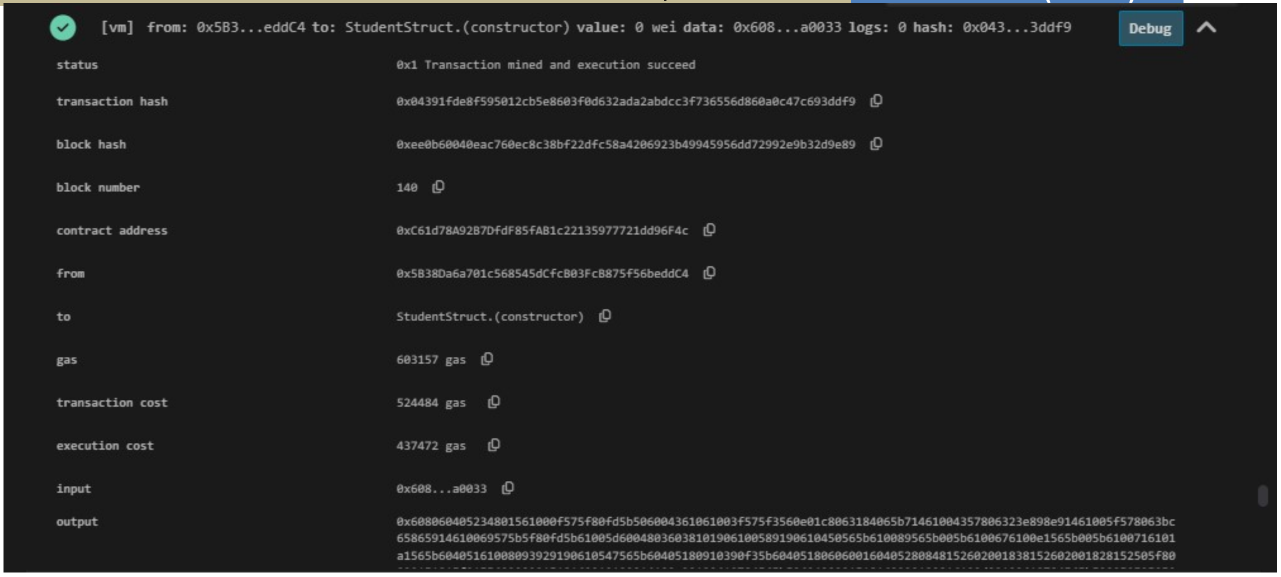
8. Write a solidity smart contract to demonstrate use of structure.
   Program:
   //SPDX-License-Identifier: MIT

```
pragma solidity  >=0.5.0<0.8.27;

contract StudentStruct{

   struct Student{
      uint id;
      string name;
      string add;
   }

   Student s1;
   function setStudent() public {
      s1 =  Student(1,"Devyan","Ratnagiri");
   }

   function enterStudent(uint id,string memory name,string memory add) public {
      s1 = Student(id,name,add);
   }

   function getStudentDetails() public view returns (uint,string memory, string memory){
      return (s1.id,s1.name,s1.add);
   }
}
```

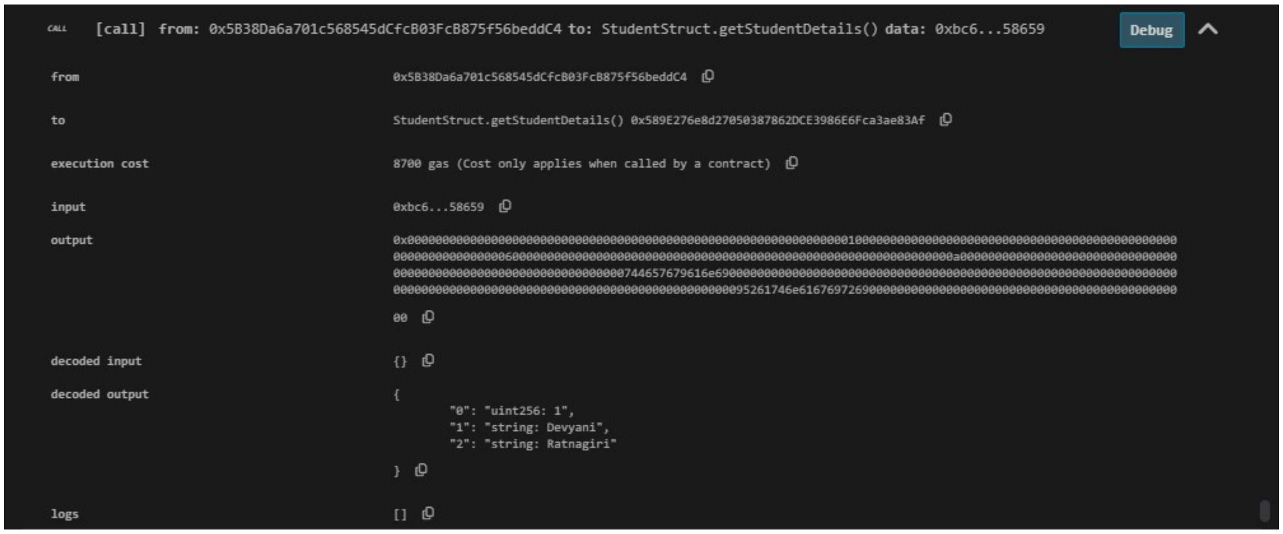
Output:



```
Balance: 0 ETH
0: uint256: 1
1: string: Devyani
2: string: Ratnagiri
```



```
[vm] from: 0x5B3...eddC4 to: StudentStruct.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x043...3ddf9    Debug

status               0x1 Transaction mined and execution succeed
transaction hash     0x04391fde8f595012cb5e8603f0d632ada2abdcc3f736556d860a0c47c693ddf9
block hash           0xee0b60040eac760ec8c38bf22dfc58a4206923b49945956dd72992e9b32d9e89
block number         140
contract address     0xC61d78A92B7DfdF85fAB1c22135977721dd96F4c
from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   StudentStruct.(constructor)
gas                  603157 gas
transaction cost     524484 gas
execution cost       437472 gas
input                0x608...a0033
output               0x608060405234801561000f575f80fd5b506004361061003f575f3560e01c8063184065b71461004357806323e898e91461005f578063bc
                     65865914610069575b5f80fd5b61005d6004803603810190610058919061045056 5b610089565b005b6100676100e1565b005b6100716101
                     a1565b60405161008093929190610547565b60405180910390f35b6040518060600160405280848152602001838152602001828152505f80
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StudentStruct.getStudentDetails() data: 0xbc6...58659    Debug

from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   StudentStruct.getStudentDetails() 0x589E276e8d27050387862DCE3986E6Fca3ae83Af
execution cost       8700 gas (Cost only applies when called by a contract)
input                0xbc6...58659
output               0x000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000
                     0000000000000000600000000000000000000000000000000000000000000000000000000000000a0000000000000000000000000000000
                     0000000000000000000000000000000007446576796160e690000000000000000000000000000000000000000000000000000000000000000
                     000000000000000000000000000000000000000000000000095261746e61676972690000000000000000000000000000000000000000000000
                     00

decoded input        {}
decoded output       {
                             "0": "uint256: 1",
                             "1": "string: Devyani",
                             "2": "string: Ratnagiri"
                     }
logs                 []
```

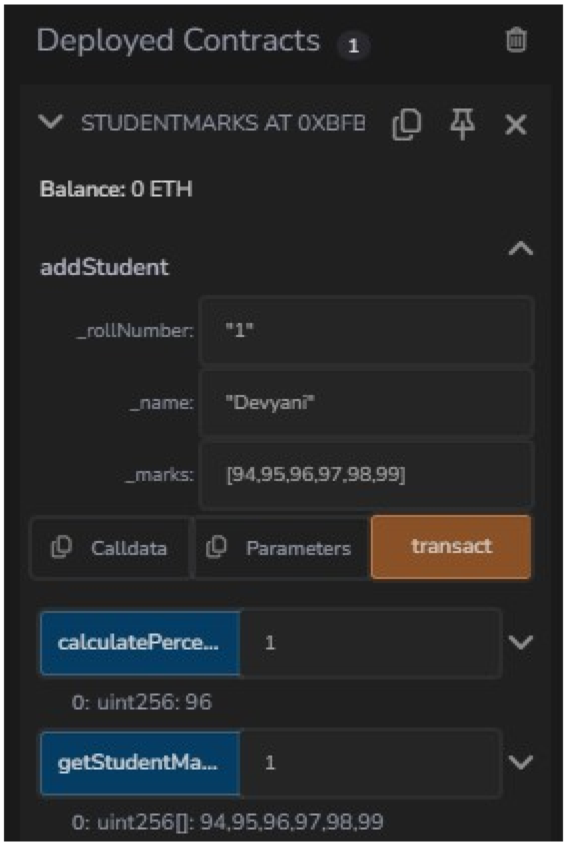
9. Write a solidity smart contract to calculate percentage of marks obtained by students for six subject in final examination.

Program:

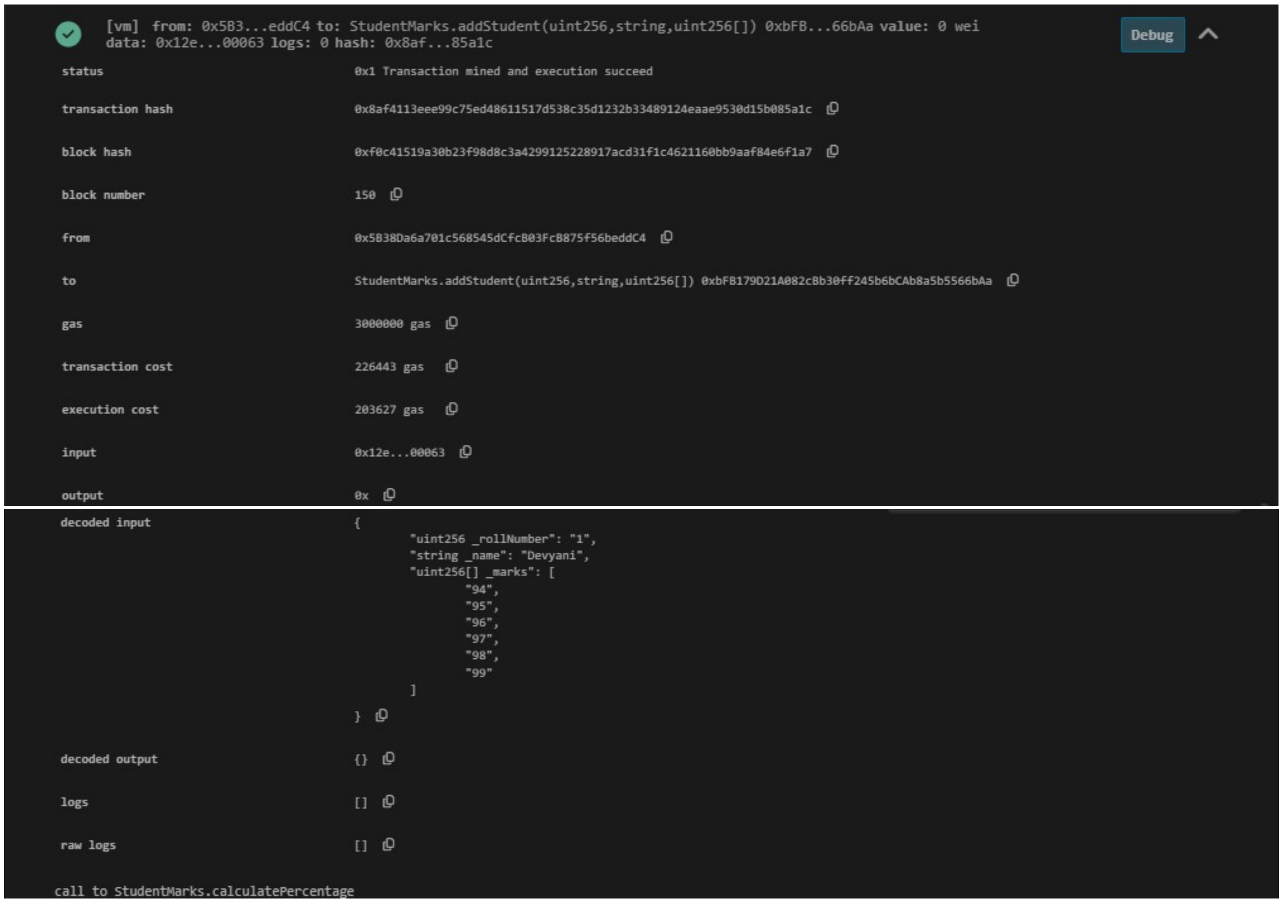
```
//SPDX-License-Identifier: MIT


pragma solidity >=0.5.0 <0.8.27;


contract StudentMarks {
  // Define a struct to store student information
  struct Student {
    string name;
    uint256 rollNumber;
    uint256[] marks;
  }
```

```
    // Mapping to store student data
    mapping(uint256 => Student) public students;

    // Function to add a new student
    function addStudent(uint256 _rollNumber, string memory _name, uint256[] memory _marks) public {
        // Check if the student already exists
        require(students[_rollNumber].rollNumber == 0, "Student already exists");

        // Validate the marks array length
        require(_marks.length == 6, "Invalid number of subjects");

        // Add the student to the mapping
        students[_rollNumber] = Student(_name, _rollNumber, _marks);
    }

    // Function to calculate the percentage of marks for a student
    function calculatePercentage(uint256 _rollNumber) public view returns (uint256) {
        // Get the student's marks
        uint256[] memory marks = students[_rollNumber].marks;

        // Calculate the total marks
        uint256 totalMarks = 0;
        for (uint256 i = 0; i < marks.length; i++) {
            totalMarks += marks[i];
        }

        // Calculate the percentage
        uint256 percentage = (totalMarks / marks.length);

        return percentage;
    }

    // Function to get the student's marks
    function getStudentMarks(uint256 _rollNumber) public view returns (uint256[] memory) {
        return students[_rollNumber].marks;
    }
}
```

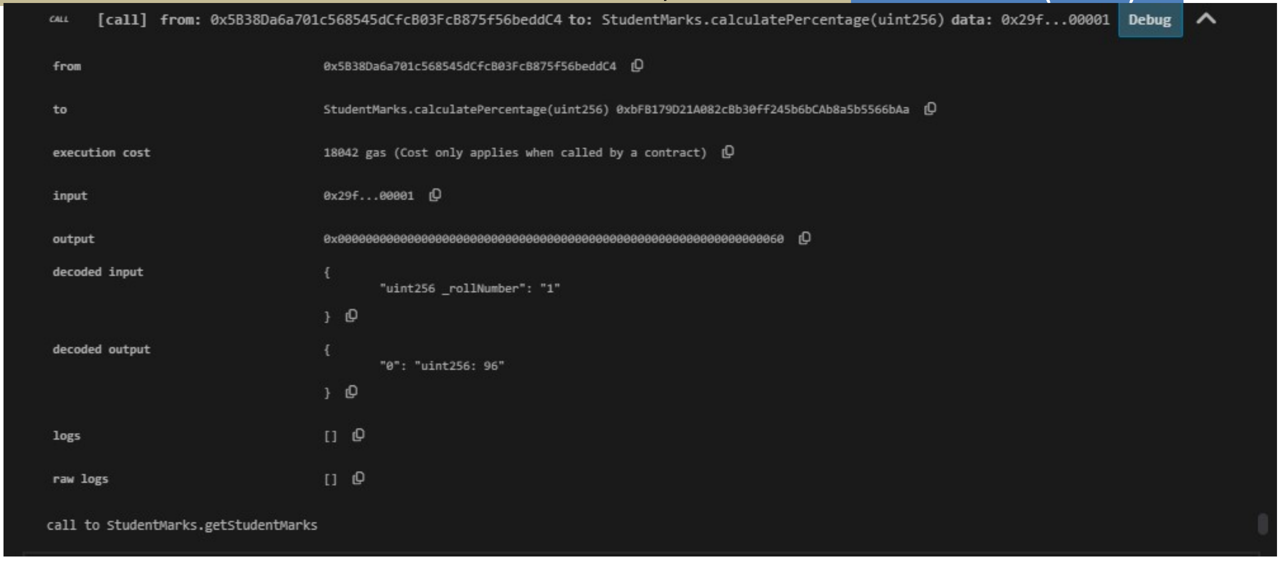
Output:



```
Balance: 0 ETH

addStudent
_rollNumber: "1"
_name: "Devyani"
_marks: [94,95,96,97,98,99]

calculatePerce... 1
0: uint256: 96

getStudentMa... 1
0: uint256[]: 94,95,96,97,98,99
```



```
[vm] from: 0x5B3...eddC4 to: StudentMarks.addStudent(uint256,string,uint256[]) 0xbFB...66bAa value: 0 wei
data: 0x12e...00063 logs: 0 hash: 0x8af...85a1c
status              0x1 Transaction mined and execution succeed
transaction hash    0x8af4113eee99c75ed48611517d538c35d1232b33489124eaae9530d15b085a1c
block hash          0xf0c41519a30b23f98d8c3a4299125228917acd31f1c4621160bb9aaf84e6f1a7
block number        150
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  StudentMarks.addStudent(uint256,string,uint256[]) 0xbFB179D21A082cBb30ff245b6bCAb8a5b5566bAa
gas                 3000000 gas
transaction cost    226443 gas
execution cost      203627 gas
input               0x12e...00063
output              0x
decoded input       {
                            "uint256 _rollNumber": "1",
                            "string _name": "Devyani",
                            "uint256[] _marks": [
                                    "94",
                                    "95",
                                    "96",
                                    "97",
                                    "98",
                                    "99"
                            ]
                    }
decoded output      {}
logs                []
raw logs            []
call to StudentMarks.calculatePercentage
```



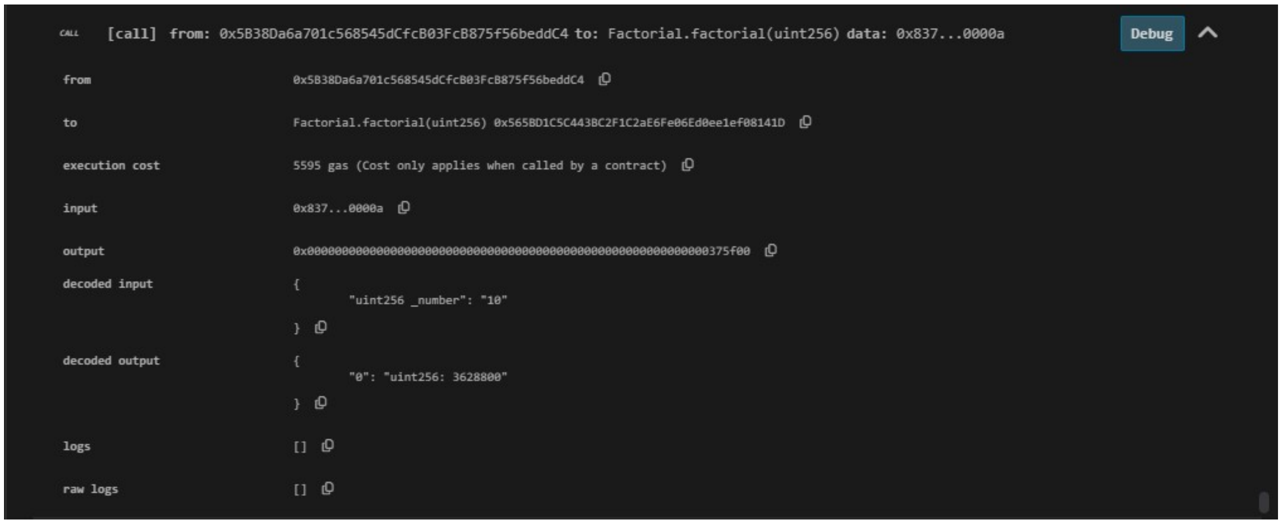
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StudentMarks.calculatePercentage(uint256) data: 0x29f...00001

from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  StudentMarks.calculatePercentage(uint256) 0xbFB179D21A082cBb30ff245b6bCAb8a5b5566bAa
execution cost      18042 gas (Cost only applies when called by a contract)
input               0x29f...00001
output              0x0000000000000000000000000000000000000000000000000000000000000060
decoded input       {
                        "uint256 _rollNumber": "1"
                    }
decoded output      {
                        "0": "uint256: 96"
                    }
logs                []
raw logs            []

call to StudentMarks.getStudentMarks
```

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

Program:

```
    // SPDX-License-Identifier: MIT

    pragma solidity >=0.5.0 <0.8.27;

    contract Factorial{
            function factorial (uint256 _number) public pure returns (uint256){
                    if (_number == 0){
                            return 1;
                    }
                    return (_number * factorial (_number - 1));
            }
    }
```

Output:

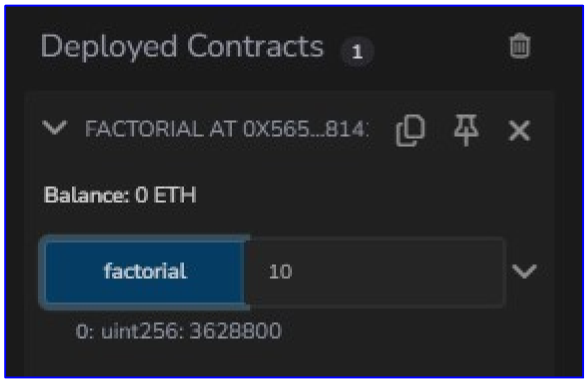


```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Factorial.factorial(uint256) data: 0x837...0000a

from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Factorial.factorial(uint256) 0x565BD1C5C443BC2F1C2aE6Fe06Ed0ee1ef08141D
execution cost      5595 gas (Cost only applies when called by a contract)
input               0x837...0000a
output              0x00000000000000000000000000000000000000000000000000000000000375f00
decoded input       {
                            "uint256 _number": "10"
                    }
decoded output      {
                            "0": "uint256: 3628800"
                    }
logs                []
raw logs            []
```

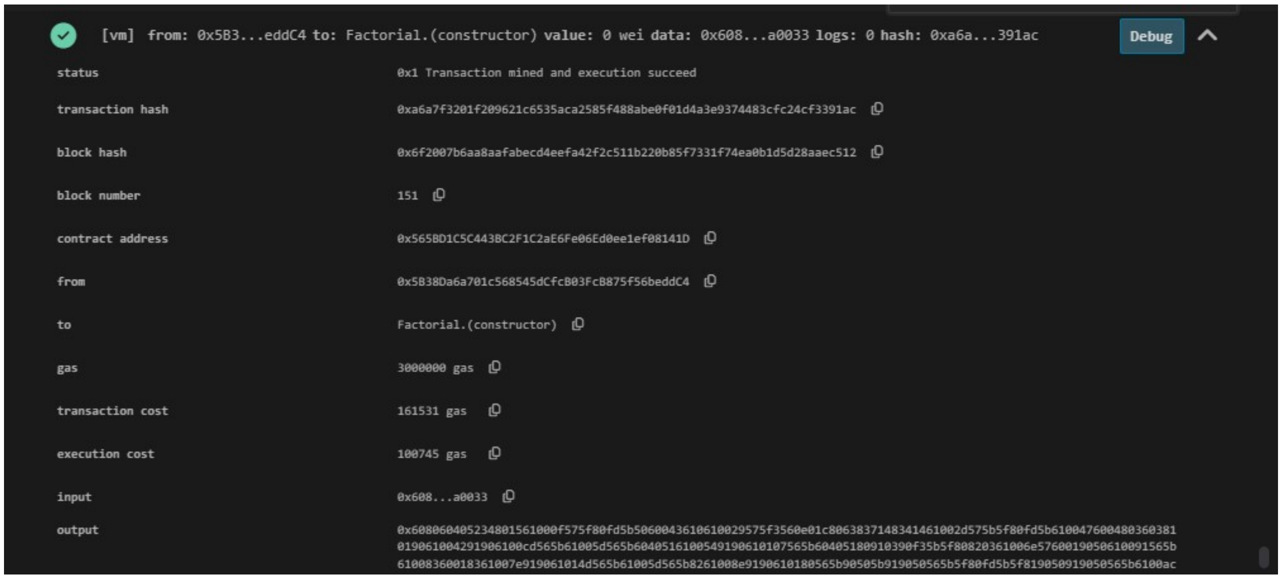
11. Write a solidity smart contract to check whether entered number is palindrome or not.

Program:

//SPDX-License-Identifier: MIT



```
Deployed Contracts 1
FACTORIAL AT 0X565...814
Balance: 0 ETH
factorial  10
0: uint256: 3628800
```

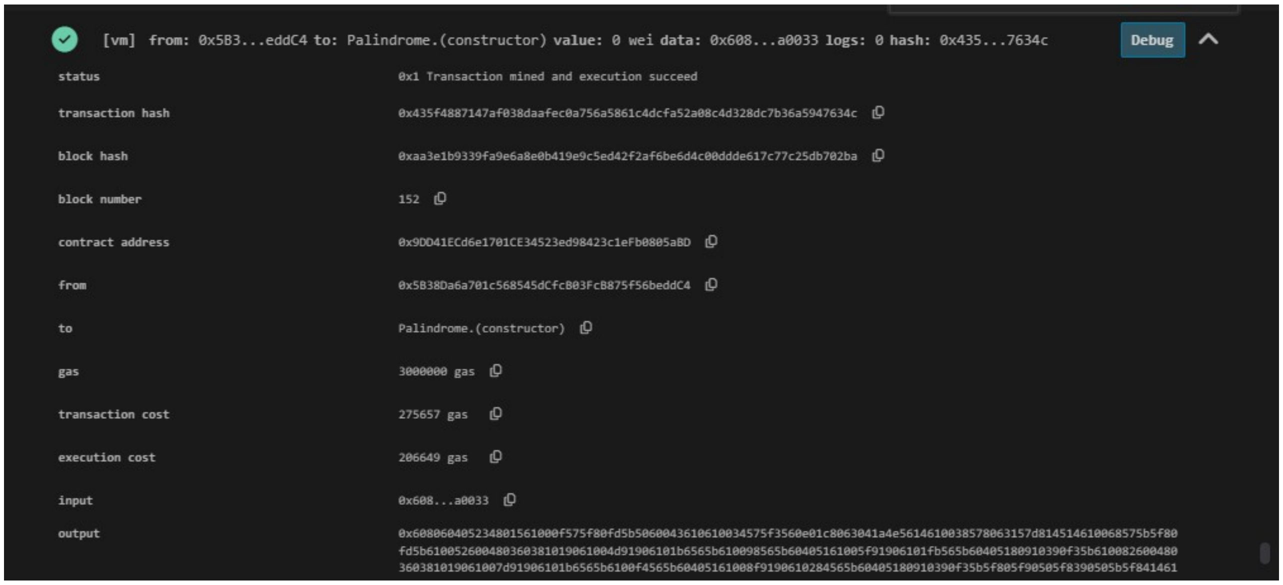


```
[vm] from: 0x5B3...eddC4 to: Factorial.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xa6a...391ac
status              0x1 Transaction mined and execution succeed
transaction hash    0xa6a7f3201f209621c6535aca2585f488abe0f01d4a3e9374483cfc24cf3391ac
block hash          0x6f2007b6aa8aafabecd4eefa42f2c511b220b85f7331f74ea0b1d5d28aaec512
block number        151
contract address    0x565BD1C5C443BC2F1C2aE6Fe06Ed0ee1ef08141D
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Factorial.(constructor)
gas                 3000000 gas
transaction cost    161531 gas
execution cost      100745 gas
input               0x608...a0033
output              0x6080604052348015610000f575f80fd5b5060043610610029575f3560e01c806383714834146100 2d575b5f80fd5b610047600480360381
                    019061004291906100cd565b61005d565b60405161005491906101075 65b60405180910390f35b5f80820361006e576001905061009156 5b
                    6100836001836100 7e919061014d565b61005d565b8261008e91906101805 65b90505b919050565b5f80fd5b5f81905091905056 5b6100ac
```

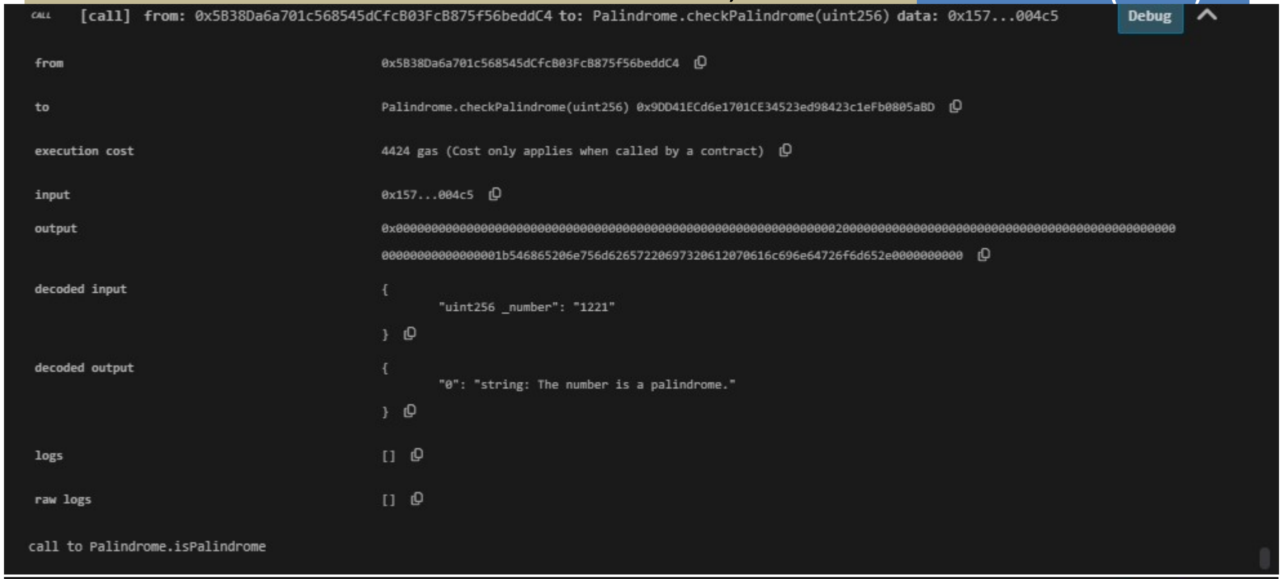
pragma solidity >=0.5.0 <0.8.27;

```
contract Palindrome{
   function isPalindrome(uint256 _number) public pure returns (bool) {
      uint256 reversed = 0;
      uint256 original = _number;

      while (_number != 0) {
         uint256 remainder = _number % 10;
         reversed = reversed * 10 + remainder;
         _number /= 10;
      }
      return original == reversed;
   }

  function checkPalindrome(uint256 _number) public pure returns (string memory) {
      if (isPalindrome(_number)) {
         return ("The number is a palindrome.");
      } else {
         return "The number is not a palindrome.";
      }
   }
}
```

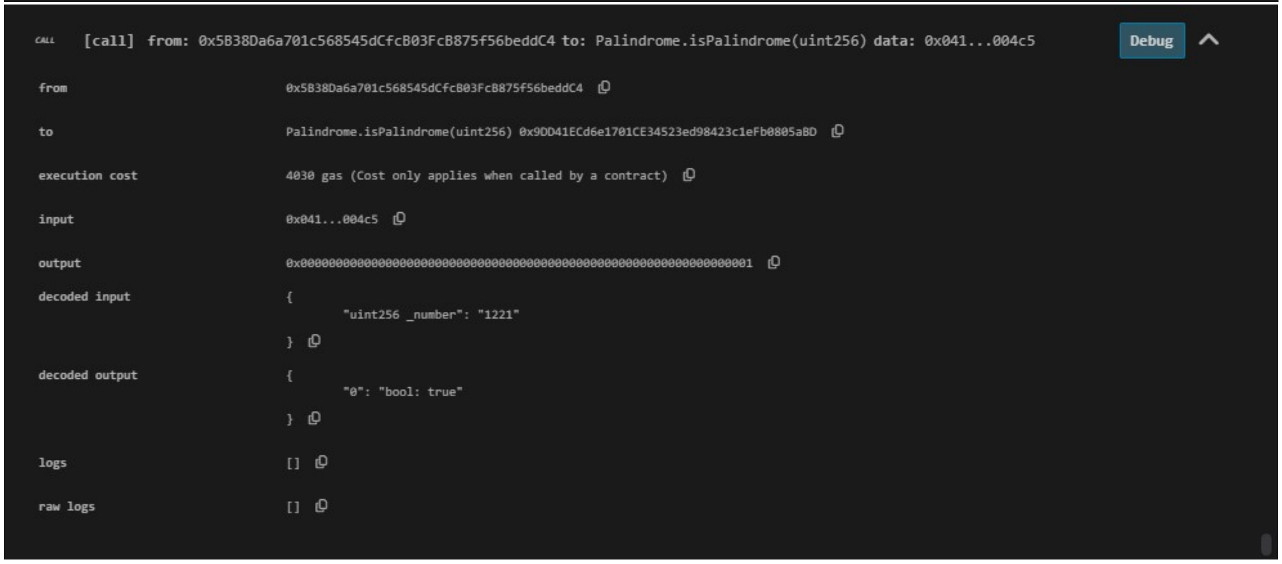
Output:



```
[vm]  from: 0x5B3...eddC4 to: Palindrome.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x435...7634c
status                    0x1 Transaction mined and execution succeed
transaction hash          0x435f4887147af038daafec0a756a5861c4dcfa52a08c4d328dc7b36a5947634c
block hash                0xaa3e1b9339fa9e6a8e0b419e9c5ed42f2af6be6d4c00ddde617c77c25db702ba
block number              152
contract address          0x9DD41ECd6e1701CE34523ed98423c1eFb0805aBD
from                      0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                        Palindrome.(constructor)
gas                       3000000 gas
transaction cost          275657 gas
execution cost            206649 gas
input                     0x608...a0033
output                    0x608060405234801561000f575f80fd5b5060043610610034575f3560e01c8063041a4e561461003857806315 7d814514610068575b5f80
                          fd5b610052600480360381019061004d91906101b6565b610098565b60405161005f91906101fb565b60405180910390f35b610082600480
                          360381019061007d91906101b6565b6100f4565b604051610008f9190610284565b60405180910390f35b5f805f90505f8390505b5f841461
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Palindrome.checkPalindrome(uint256) data: 0x157...004c5   Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Palindrome.checkPalindrome(uint256) 0x9DD41ECd6e1701CE34523ed98423c1eFb0805aBD
execution cost    4424 gas (Cost only applies when called by a contract)
input             0x157...004c5
output            0x00000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000
                  00000000000000001b546865206e756d62657220697320612070616c696e64726f6d652e0000000000
decoded input     {
                        "uint256 _number": "1221"
                  }
decoded output    {
                        "0": "string: The number is a palindrome."
                  }
logs              []
raw logs          []

call to Palindrome.isPalindrome
```



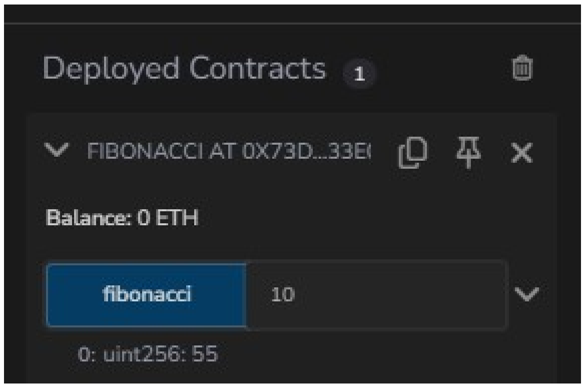
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Palindrome.isPalindrome(uint256) data: 0x041...004c5   Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Palindrome.isPalindrome(uint256) 0x9DD41ECd6e1701CE34523ed98423c1eFb0805aBD
execution cost    4030 gas (Cost only applies when called by a contract)
input             0x041...004c5
output            0x0000000000000000000000000000000000000000000000000000000000000001
decoded input     {
                        "uint256 _number": "1221"
                  }
decoded output    {
                        "0": "bool: true"
                  }
logs              []
raw logs          []
```

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

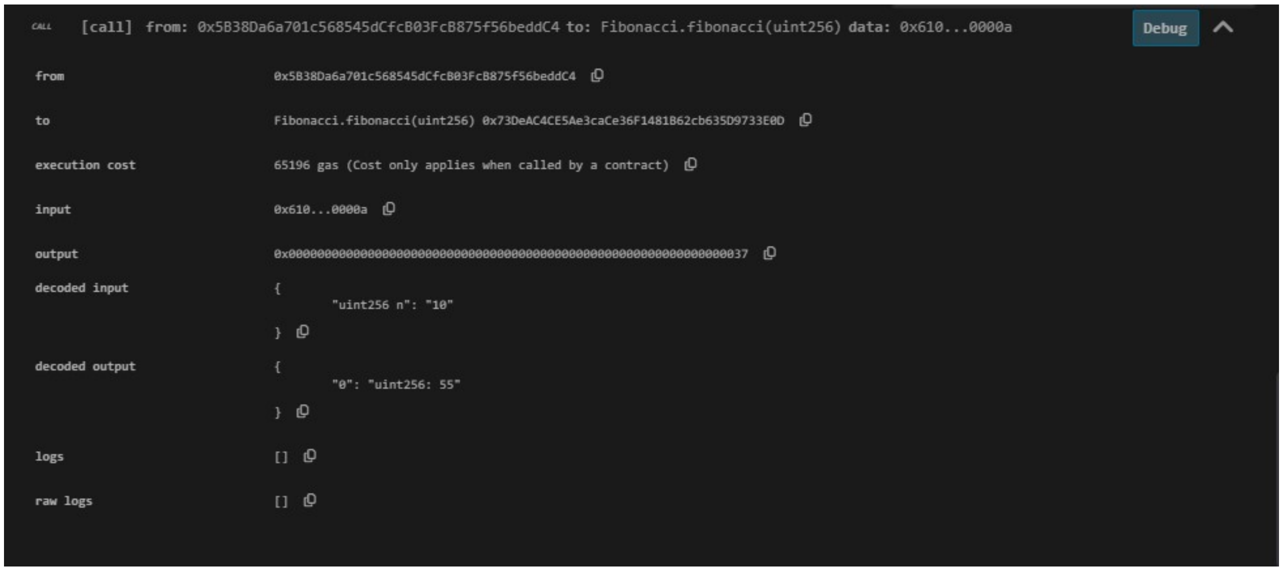
Program:

```
//SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract Fibonacci {
   function fibonacci(uint n) public pure returns (uint){
      if (n == 0){
         return 0;
      }else if (n == 1){
         return 1;
      }else{
         return fibonacci(n -1) +fibonacci(n -2);
      }
   }
}
```

Output:



```
Deployed Contracts 1
FIBONACCI AT 0X73D...33E(
Balance: 0 ETH
fibonacci   10
0: uint256: 55
```



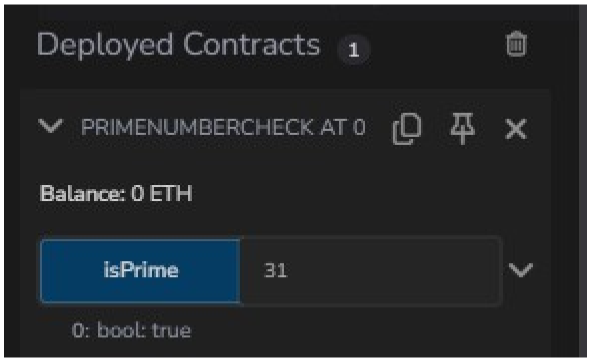
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Fibonacci.fibonacci(uint256) data: 0x610...0000a

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 Fibonacci.fibonacci(uint256) 0x73DeAC4CE5Ae3caCe36F1481B62cb635D9733E0D
execution cost     65196 gas (Cost only applies when called by a contract)
input              0x610...0000a
output             0x0000000000000000000000000000000000000000000000000000000000000037
decoded input      {
                           "uint256 n": "10"
                   }
decoded output     {
                           "0": "uint256: 55"
                   }
logs               []
raw logs           []
```

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

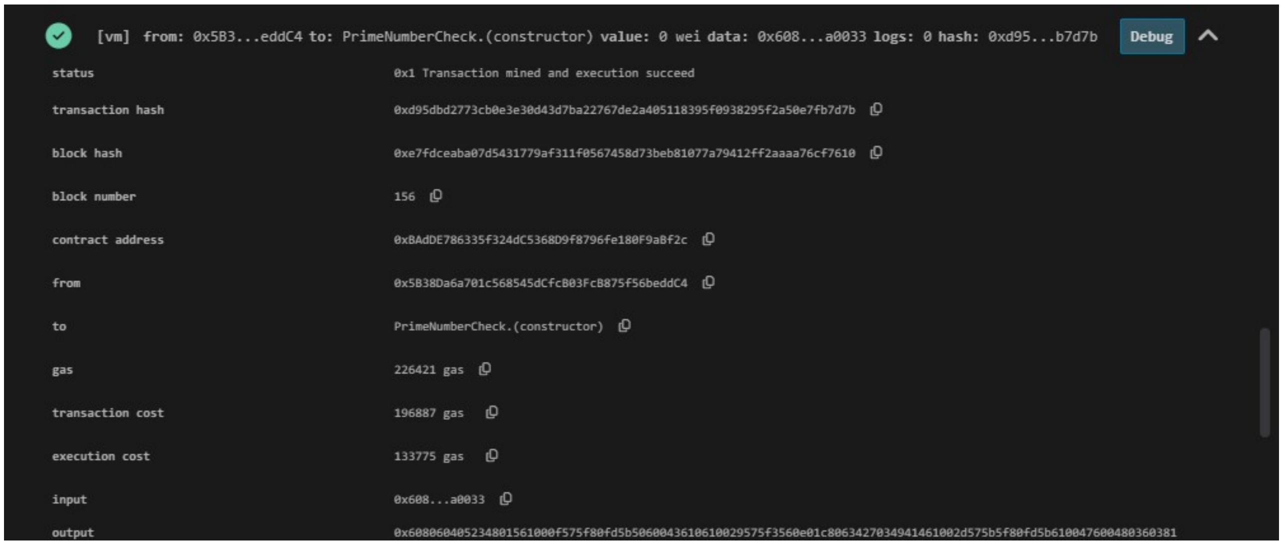
Program:

```
    //SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract PrimeNumberCheck{
  function isPrime(uint256 _number) public pure returns (bool) {
    if (_number <= 1) {
      return false;
    }
    for (uint256 i = 2; i * i <= _number; i++) {
      if (_number % i == 0) {
        return false;
      }
    }
    return true;
  }
}
```

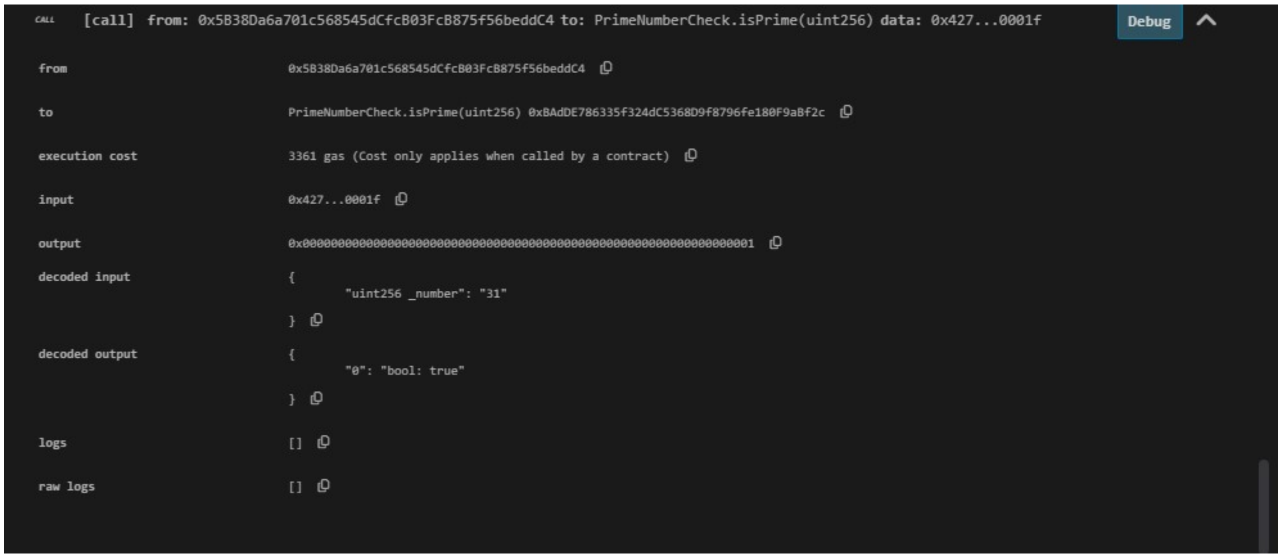
Output:



```
0: bool: true
```



```
[vm] from: 0x5B3...eddC4 to: PrimeNumberCheck.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xd95...b7d7b
status                  0x1 Transaction mined and execution succeed
transaction hash        0xd95dbd2773cb0e3e30d43d7ba22767de2a405118395f0938295f2a50e7fb7d7b
block hash              0xe7fdceaba07d5431779af311f0567458d73beb81077a79412ff2aaaa76cf7610
block number            156
contract address        0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      PrimeNumberCheck.(constructor)
gas                     226421 gas
transaction cost        196887 gas
execution cost          133775 gas
input                   0x608...a0033
output                  0x6080604052348015610 00f575f80fd5b5060043610610029575f3560e01c806342703494146100 2d575b5f80fd5b610047600480360381
```



```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: PrimeNumberCheck.isPrime(uint256) data: 0x427...0001f
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      PrimeNumberCheck.isPrime(uint256) 0xBAdDE786335f324dC5368D9f8796fe180F9aBf2c
execution cost          3361 gas (Cost only applies when called by a contract)
input                   0x427...0001f
output                  0x0000000000000000000000000000000000000000000000000000000000000001
decoded input           {
                                "uint256 _number": "31"
                        }
decoded output          {
                                "0": "bool: true"
                        }
logs                    []
raw logs                []
```

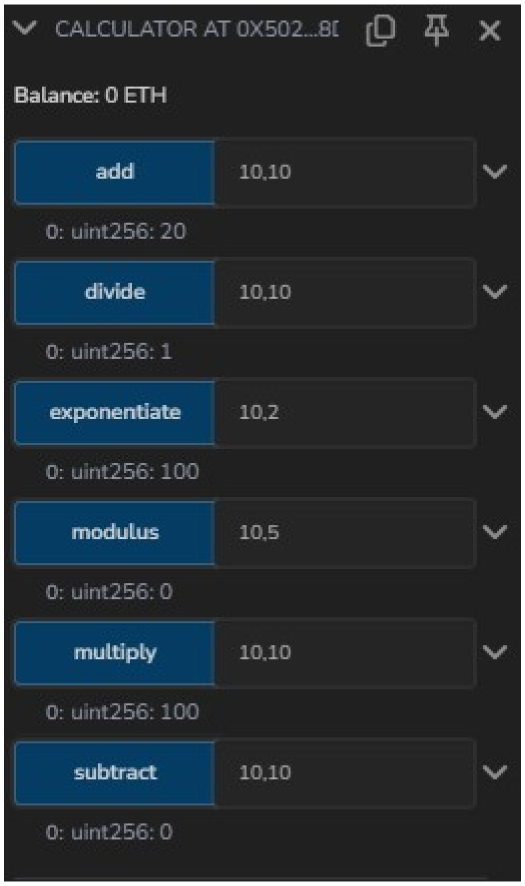
14. Write a solidity smart contract to create arithmetic calculator which includes functions for operations addition, subtraction, multiplication, division etc.

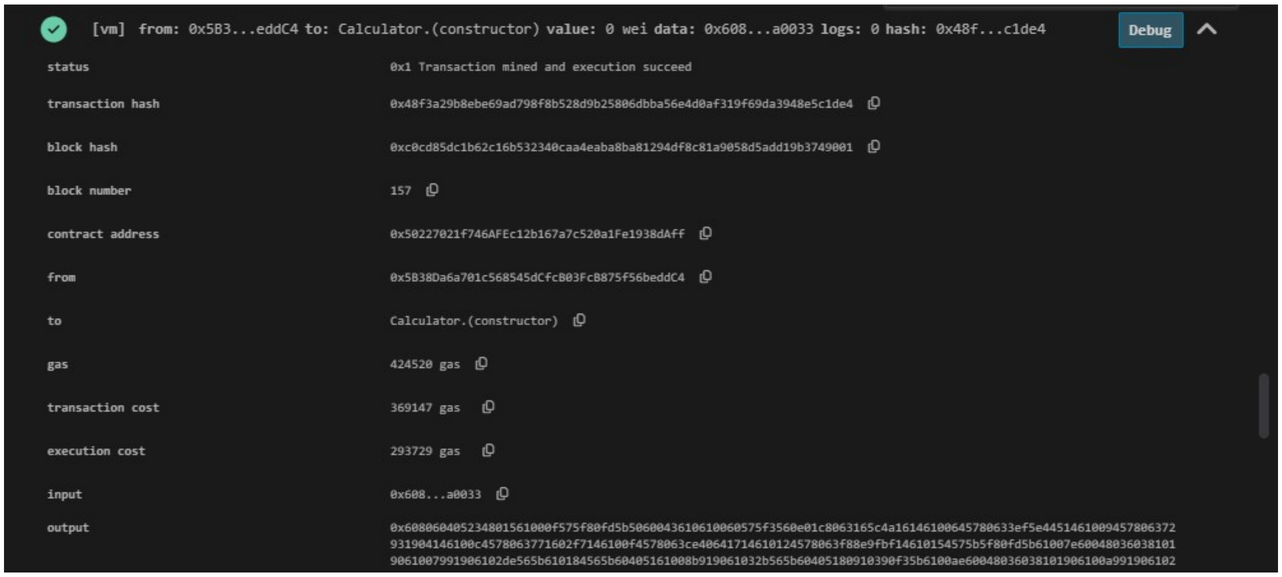
Program:

```
    //SPDX-License-Identifier: MIT


pragma solidity >=0.5.0 <0.8.27;
```

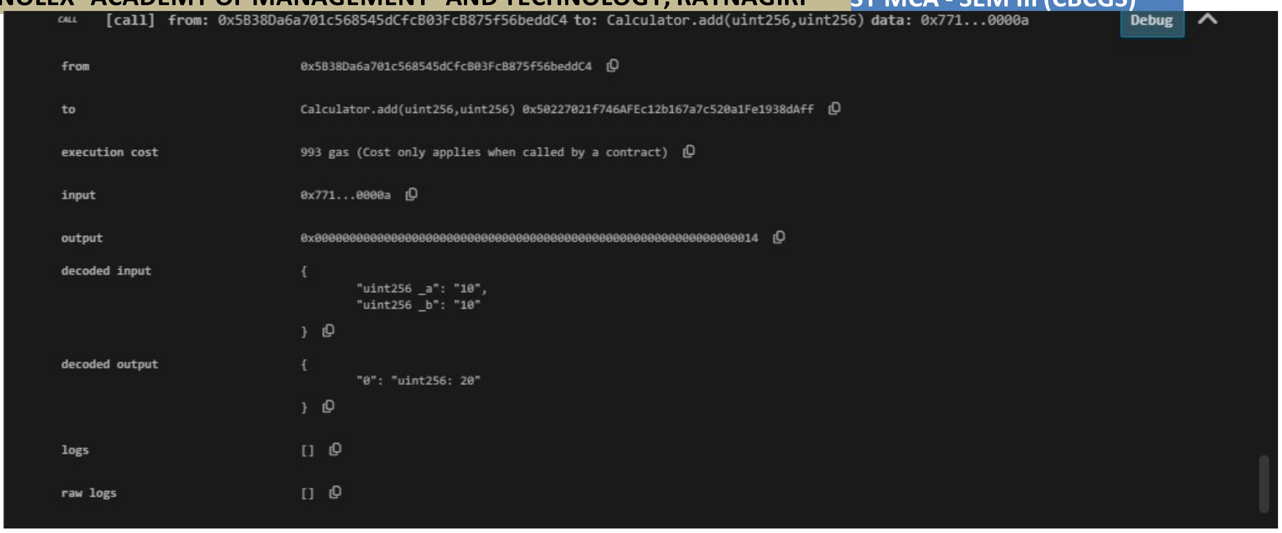
```
contract Calculator{
  function add(uint256 _a, uint256 _b) public pure returns (uint256) {
    return _a + _b;
  }


  function subtract(uint256 _a, uint256 _b) public pure returns (uint256) {
    return _a - _b;
  }


  function multiply(uint256 _a, uint256 _b) public pure returns (uint256) {
    return _a * _b;
  }


  function divide(uint256 _a, uint256 _b) public pure returns (uint256) {
    require(_b != 0, "Division by zero is not allowed");
    return _a / _b;
  }


  function modulus(uint256 _a, uint256 _b) public pure returns (uint256) {
    require(_b != 0, "Modulus by zero is not allowed");
    return _a % _b;
  }


  function exponentiate(uint256 _a, uint256 _b) public pure returns (uint256) {
    uint256 result = 1;
    for (uint256 i = 0; i < _b; i++) {
      result *= _a;
    }
    return result;
  }
}
```

Output:

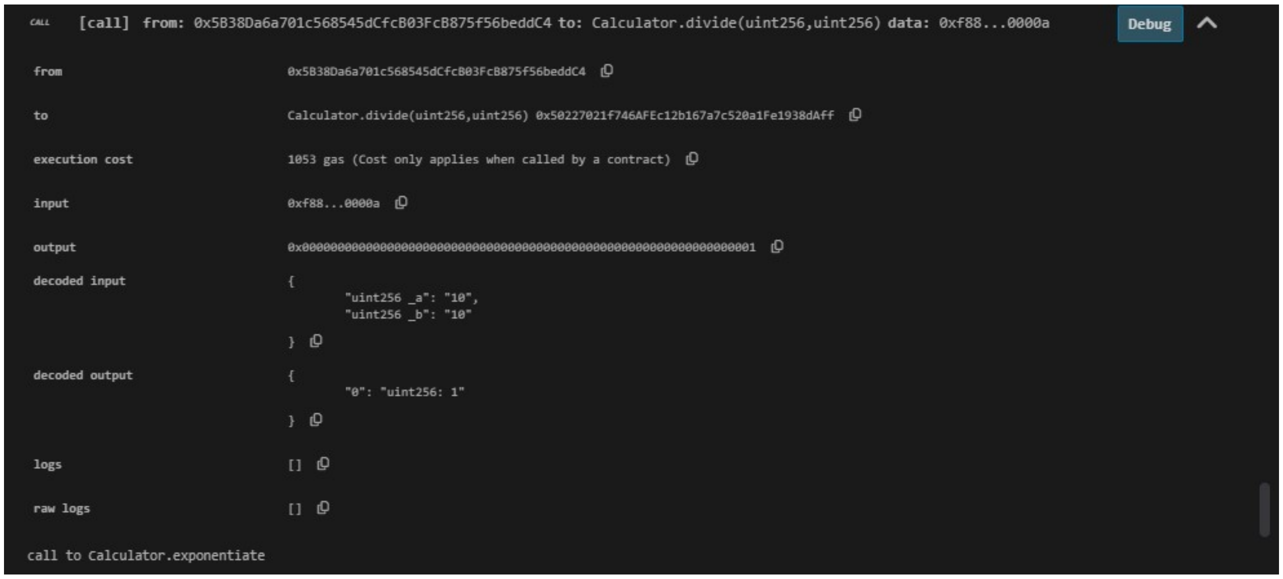




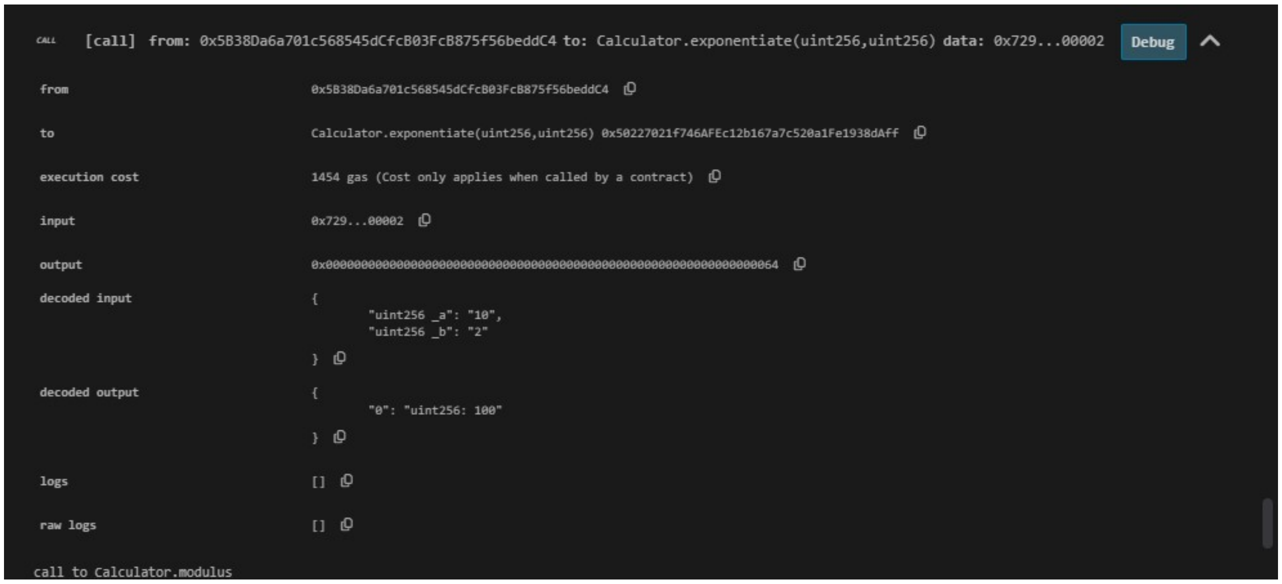
```
[vm] from: 0x5B3...eddC4 to: Calculator.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x48f...c1de4
status                  0x1 Transaction mined and execution succeed
transaction hash        0x48f3a29b8ebe69ad798f8b528d9b25806dbba56e4d0af319f69da3948e5c1de4
block hash              0xc0cd85dc1b62c16b532340caa4eaba8ba81294df8c81a9058d5add19b3749001
block number            157
contract address        0x50227021f746AFEc12b167a7c520a1Fe1938dAff
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Calculator.(constructor)
gas                     424520 gas
transaction cost        369147 gas
execution cost          293729 gas
input                   0x608...a0033
output                  0x608060405234801561000f575f80fd5b506004361061006057 5f3560e01c8063165c4a161461006457806 33ef5e4451461009457806372
                        931904146100c4578063771602f7146100f4578063ce4064171461012457806 3f88e9fbf14610154575b5f80fd5b61007e60048036038101
                        906100799190610 2de565b610184565b60405161008b919061032b565b60405180910390f35b6100ae600480360381019061 00a991906102
```



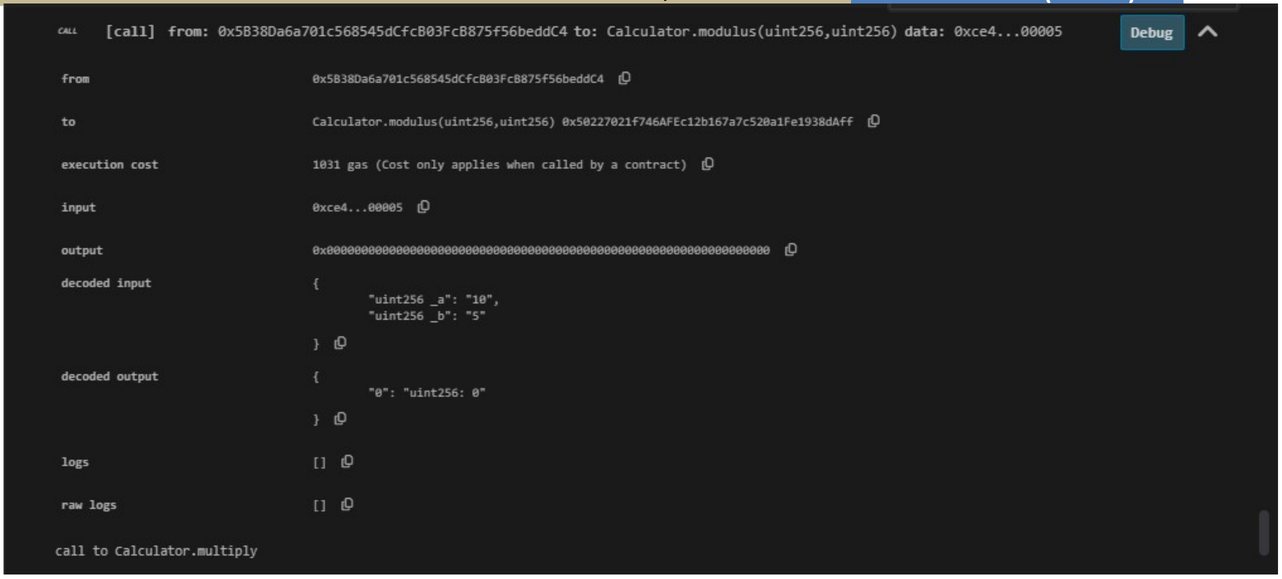
```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.add(uint256,uint256) data: 0x771...0000a   Debug

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 Calculator.add(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost     993 gas (Cost only applies when called by a contract)
input              0x771...0000a
output             0x0000000000000000000000000000000000000000000000000000000000000014
decoded input      {
                           "uint256 _a": "10",
                           "uint256 _b": "10"
                   }
decoded output     {
                           "0": "uint256: 20"
                   }
logs               []
raw logs           []
```



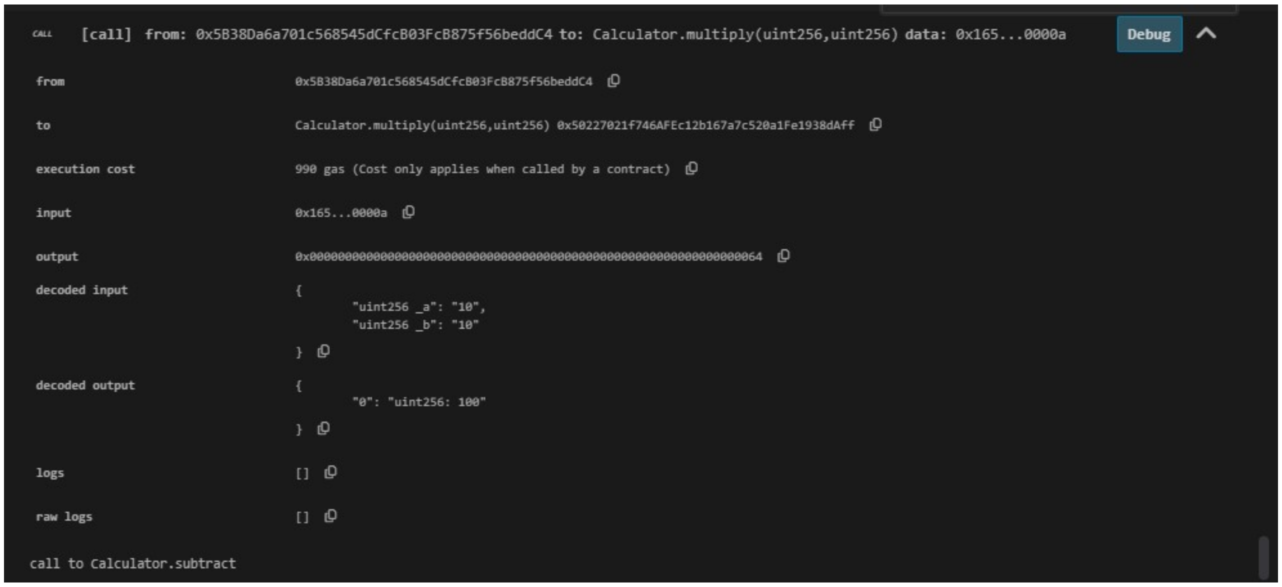
```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.divide(uint256,uint256) data: 0xf88...0000a   Debug

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 Calculator.divide(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost     1053 gas (Cost only applies when called by a contract)
input              0xf88...0000a
output             0x0000000000000000000000000000000000000000000000000000000000000001
decoded input      {
                           "uint256 _a": "10",
                           "uint256 _b": "10"
                   }
decoded output     {
                           "0": "uint256: 1"
                   }
logs               []
raw logs           []

call to Calculator.exponentiate
```



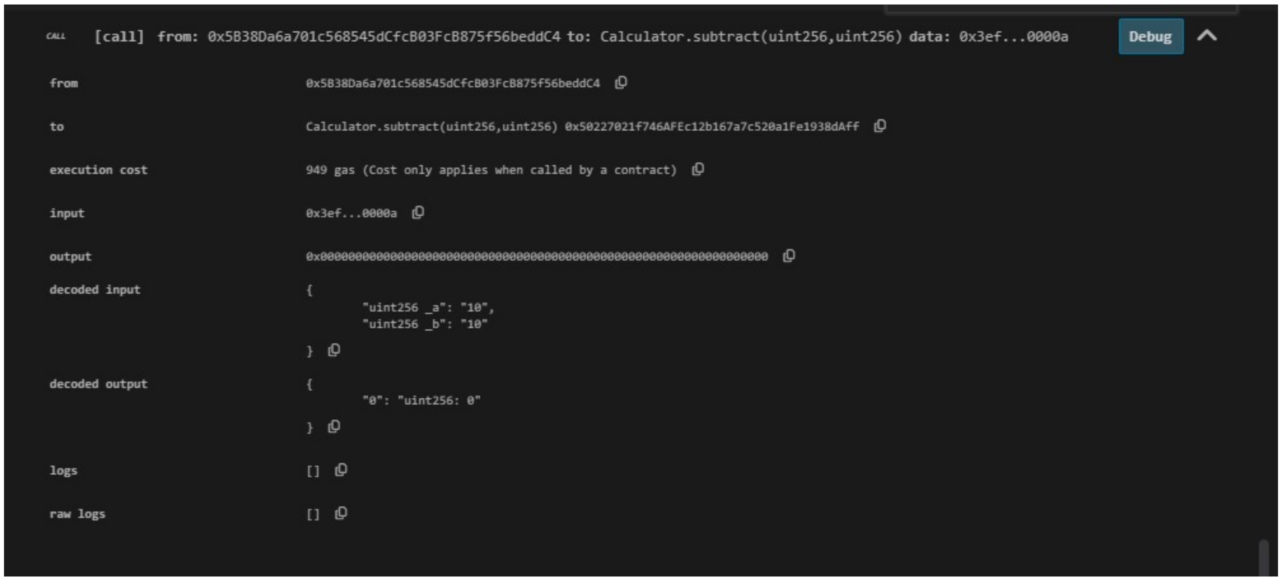
```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.exponentiate(uint256,uint256) data: 0x729...00002   Debug

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 Calculator.exponentiate(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost     1454 gas (Cost only applies when called by a contract)
input              0x729...00002
output             0x0000000000000000000000000000000000000000000000000000000000000064
decoded input      {
                           "uint256 _a": "10",
                           "uint256 _b": "2"
                   }
decoded output     {
                           "0": "uint256: 100"
                   }
logs               []
raw logs           []

call to Calculator.modulus
```



```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.modulus(uint256,uint256) data: 0xce4...00005

from             0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to               Calculator.modulus(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost   1031 gas (Cost only applies when called by a contract)
input            0xce4...00005
output           0x0000000000000000000000000000000000000000000000000000000000000000
decoded input    {
                         "uint256 _a": "10",
                         "uint256 _b": "5"
                 }
decoded output   {
                         "0": "uint256: 0"
                 }
logs             []
raw logs         []

call to Calculator.multiply
```



```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.multiply(uint256,uint256) data: 0x165...0000a

from             0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to               Calculator.multiply(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost   990 gas (Cost only applies when called by a contract)
input            0x165...0000a
output           0x0000000000000000000000000000000000000000000000000000000000000064
decoded input    {
                         "uint256 _a": "10",
                         "uint256 _b": "10"
                 }
decoded output   {
                         "0": "uint256: 100"
                 }
logs             []
raw logs         []

call to Calculator.subtract
```

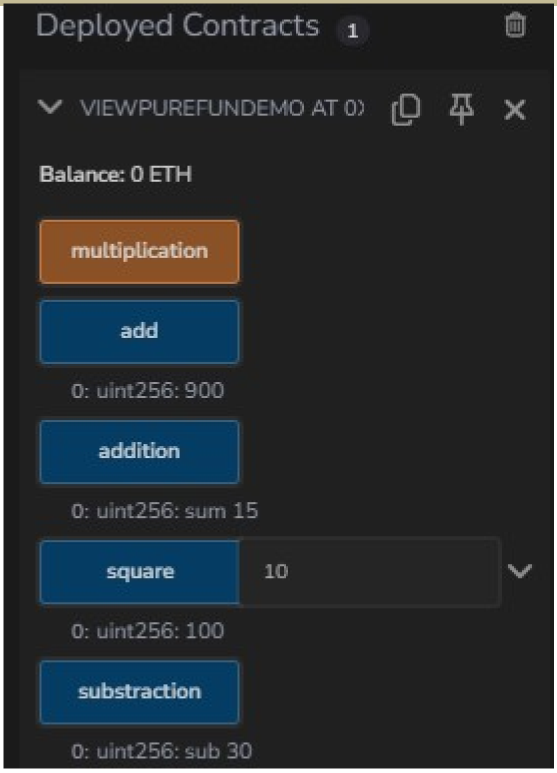


```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Calculator.subtract(uint256,uint256) data: 0x3ef...0000a

from             0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to               Calculator.subtract(uint256,uint256) 0x50227021f746AFEc12b167a7c520a1Fe1938dAff
execution cost   949 gas (Cost only applies when called by a contract)
input            0x3ef...0000a
output           0x0000000000000000000000000000000000000000000000000000000000000000
decoded input    {
                         "uint256 _a": "10",
                         "uint256 _b": "10"
                 }
decoded output   {
                         "0": "uint256: 0"
                 }
logs             []
raw logs         []
```

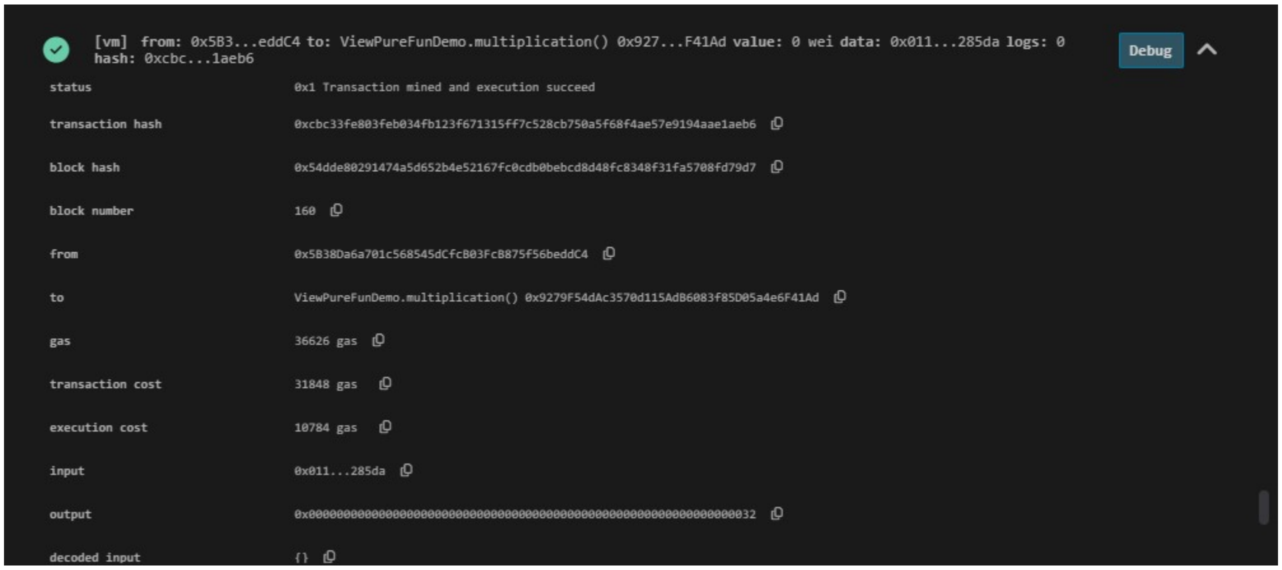
15. Write a solidity smart contract to demonstrate view function and pure function.

Program:

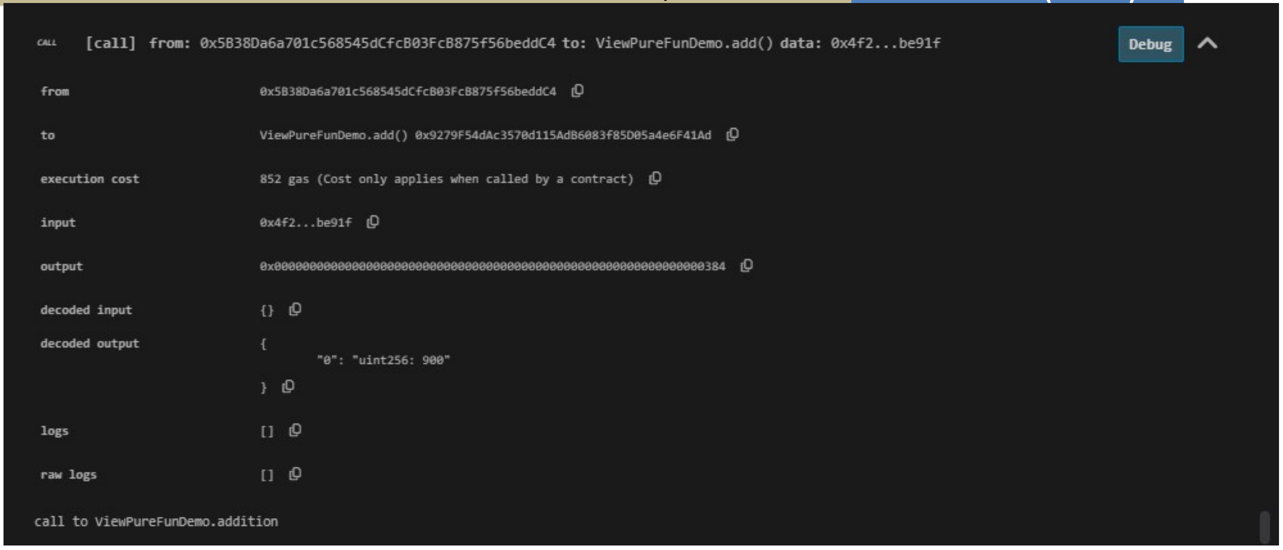
```
// SPDX-License-Identifier: MIT
pragma solidity  >=0.5.0<0.8.27;

contract ViewPureFunDemo{

   uint num1=25;
   uint num2 = 75;

   function multiplication() public returns (uint){
      num1=5;
      num2=10;
      return (num1* num2);
   }

   function addition() public view returns (uint sum){

      sum=num1+num2;
   }

   function substraction ()public pure returns(uint sub){
      uint num1 = 50;
      uint num2 =20;
      sub = num1-num2;
      return  sub;
   }

   function add() public pure returns (uint){
      uint num1 = 10;
      uint num2 =20;
      uint sum = num1+num2;
      return square(sum);

   }

   function square(uint num) public pure returns (uint ){
      num= num*num;
      return num;
   }




}
```
Output:



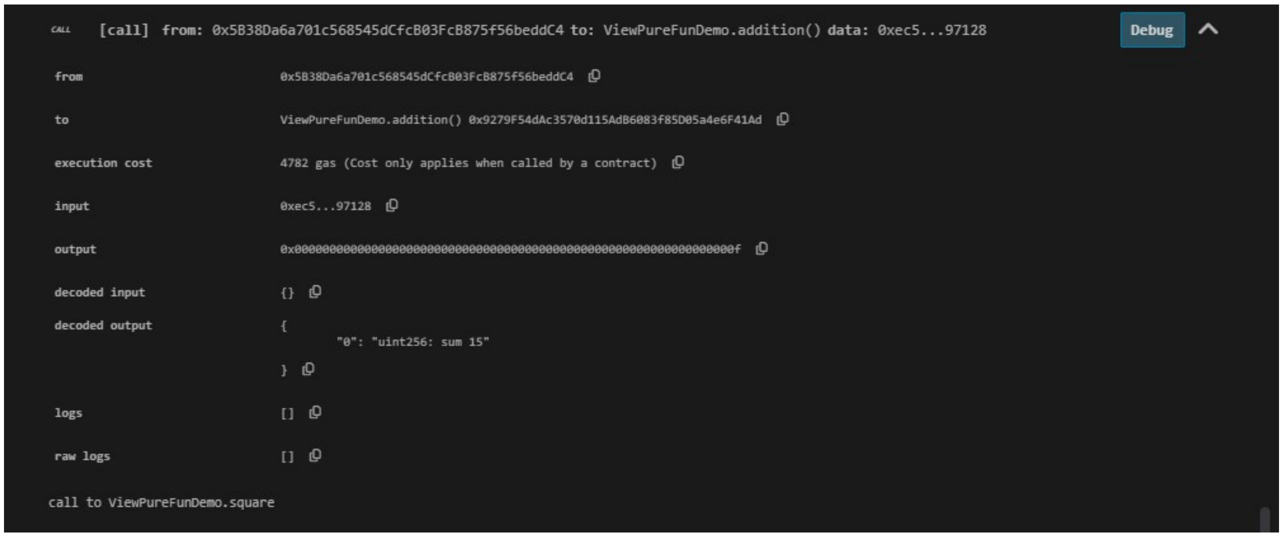
```
Balance: 0 ETH
0: uint256: 900
0: uint256: sum 15
0: uint256: 100
0: uint256: sub 30
```



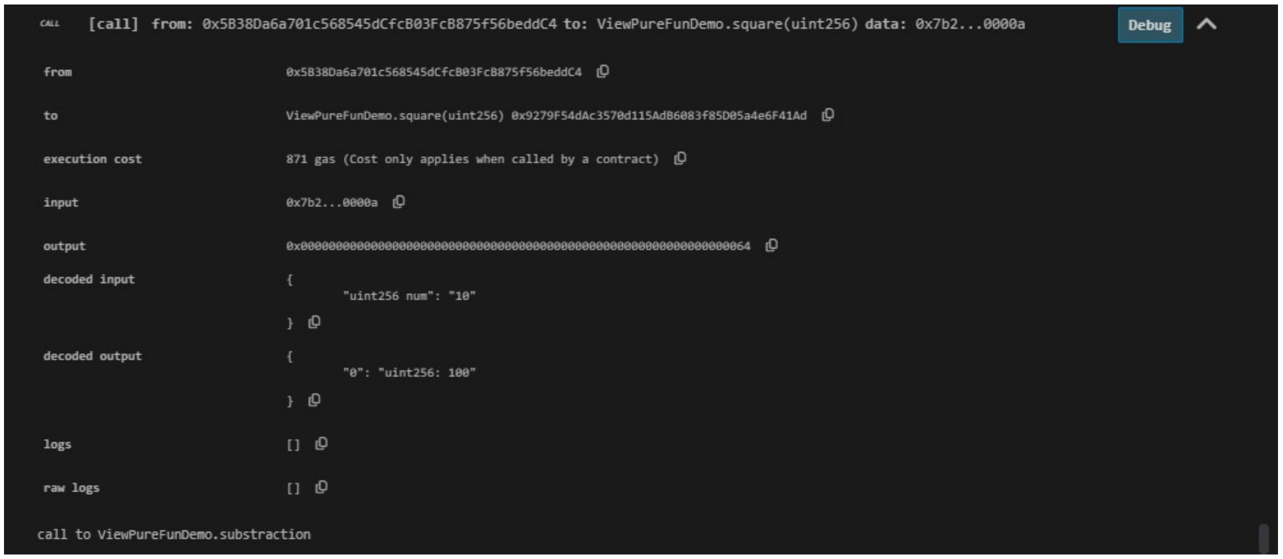
```
[vm] from: 0x5B3...eddC4 to: ViewPureFunDemo.multiplication() 0x927...F41Ad value: 0 wei data: 0x011...285da logs: 0
hash: 0xcbc...1aeb6
status              0x1 Transaction mined and execution succeed
transaction hash    0xcbc33fe803feb034fb123f671315ff7c528cb750a5f68f4ae57e9194aae1aeb6
block hash          0x54dde80291474a5d652b4e52167fc0cdb0bebcd8d48fc8348f31fa5708fd79d7
block number        160
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  ViewPureFunDemo.multiplication() 0x9279F54dAc3570d115AdB6083f85D05a4e6F41Ad
gas                 36626 gas
transaction cost    31848 gas
execution cost      10784 gas
input               0x011...285da
output              0x0000000000000000000000000000000000000000000000000000000000000032
decoded input       {}
```



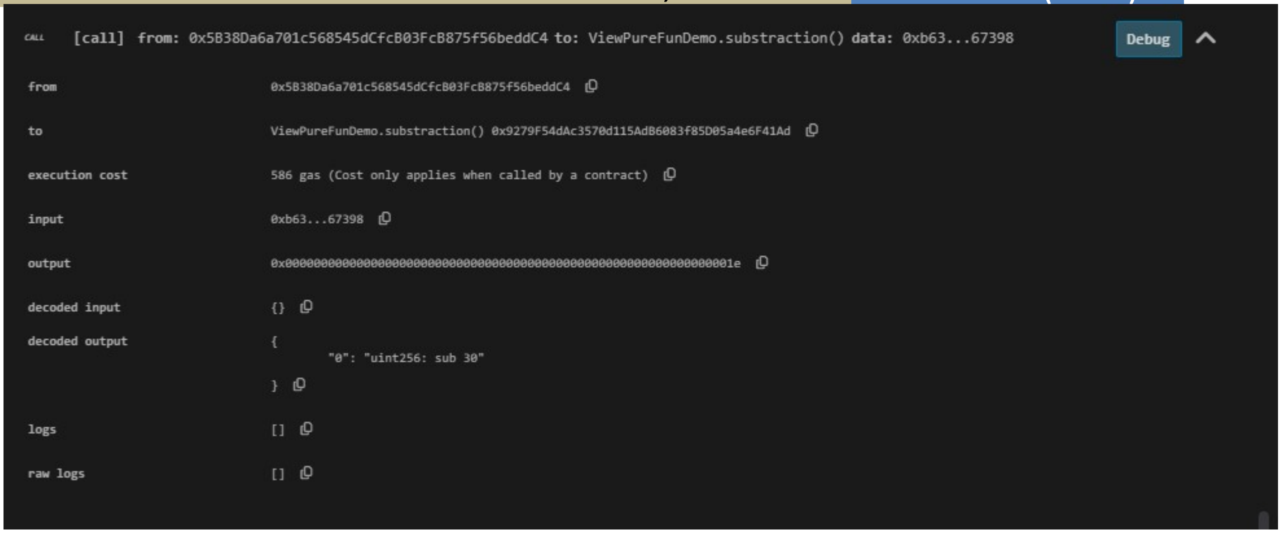
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ViewPureFunDemo.add() data: 0x4f2...be91f   Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                ViewPureFunDemo.add() 0x9279F54dAc3570d115AdB6083f85D05a4e6F41Ad
execution cost    852 gas (Cost only applies when called by a contract)
input             0x4f2...be91f
output            0x0000000000000000000000000000000000000000000000000000000000000384
decoded input     {}
decoded output    {
                          "0": "uint256: 900"
                  }
logs              []
raw logs          []

call to ViewPureFunDemo.addition
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ViewPureFunDemo.addition() data: 0xec5...97128   Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                ViewPureFunDemo.addition() 0x9279F54dAc3570d115AdB6083f85D05a4e6F41Ad
execution cost    4782 gas (Cost only applies when called by a contract)
input             0xec5...97128
output            0x000000000000000000000000000000000000000000000000000000000000000f
decoded input     {}
decoded output    {
                          "0": "uint256: sum 15"
                  }
logs              []
raw logs          []

call to ViewPureFunDemo.square
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ViewPureFunDemo.square(uint256) data: 0x7b2...0000a   Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                ViewPureFunDemo.square(uint256) 0x9279F54dAc3570d115AdB6083f85D05a4e6F41Ad
execution cost    871 gas (Cost only applies when called by a contract)
input             0x7b2...0000a
output            0x0000000000000000000000000000000000000000000000000000000000000064
decoded input     {
                          "uint256 num": "10"
                  }
decoded output    {
                          "0": "uint256: 100"
                  }
logs              []
raw logs          []

call to ViewPureFunDemo.substraction
```



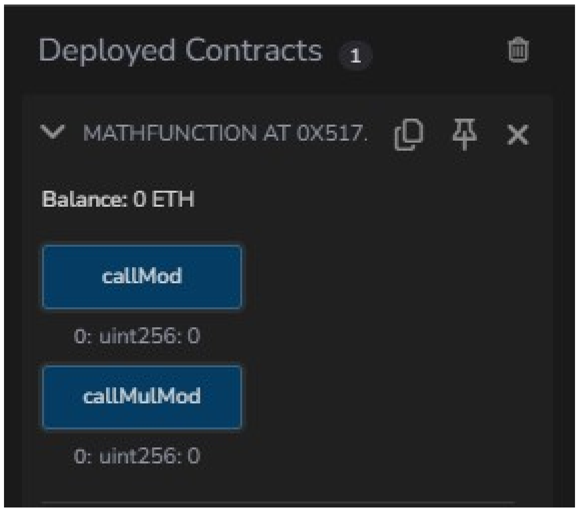
```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ViewPureFunDemo.substraction() data: 0xb63...67398

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                ViewPureFunDemo.substraction() 0x9279F54dAc3570d115AdB6083f85D05a4e6F41Ad
execution cost    586 gas (Cost only applies when called by a contract)
input             0xb63...67398
output            0x000000000000000000000000000000000000000000000000000000000000001e
decoded input     {}
decoded output    {
                      "0": "uint256: sub 30"
                  }
logs              []
raw logs          []
```

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

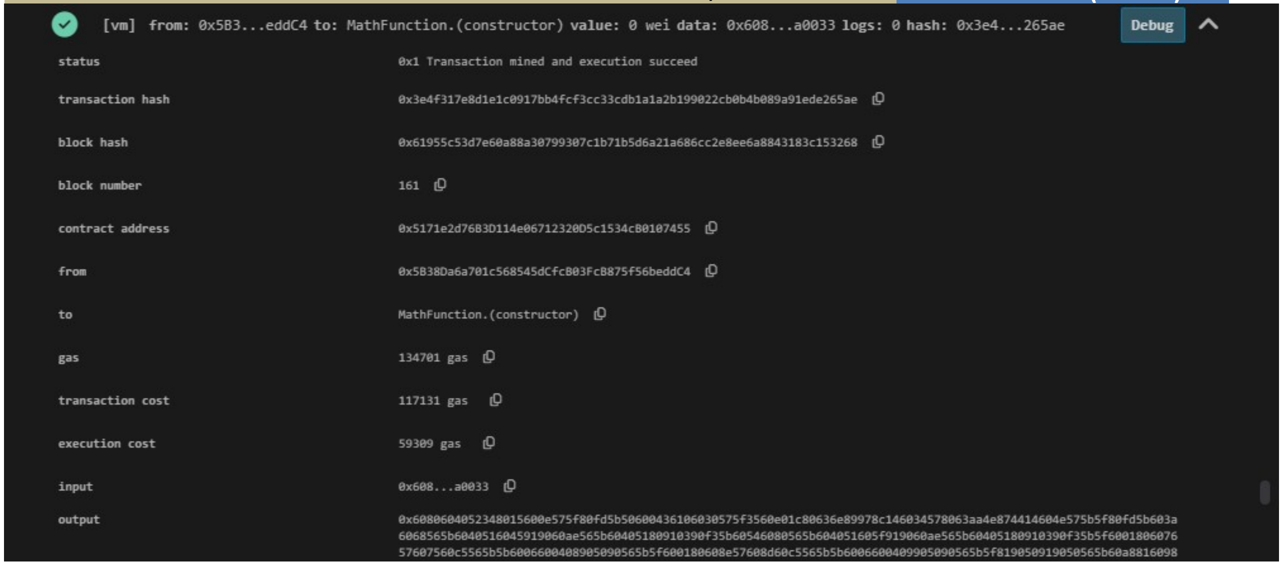
Programs:

```
//SPDX-License-Identifier: MIT

pragma solidity  >=0.5.0<0.8.27;

contract MathFunction{
   function callMod() public pure returns (uint){
      return addmod(4,6,1);
   }

   function callMulMod() public pure returns(uint){
      return mulmod(4, 6, 1);
   }
}
```

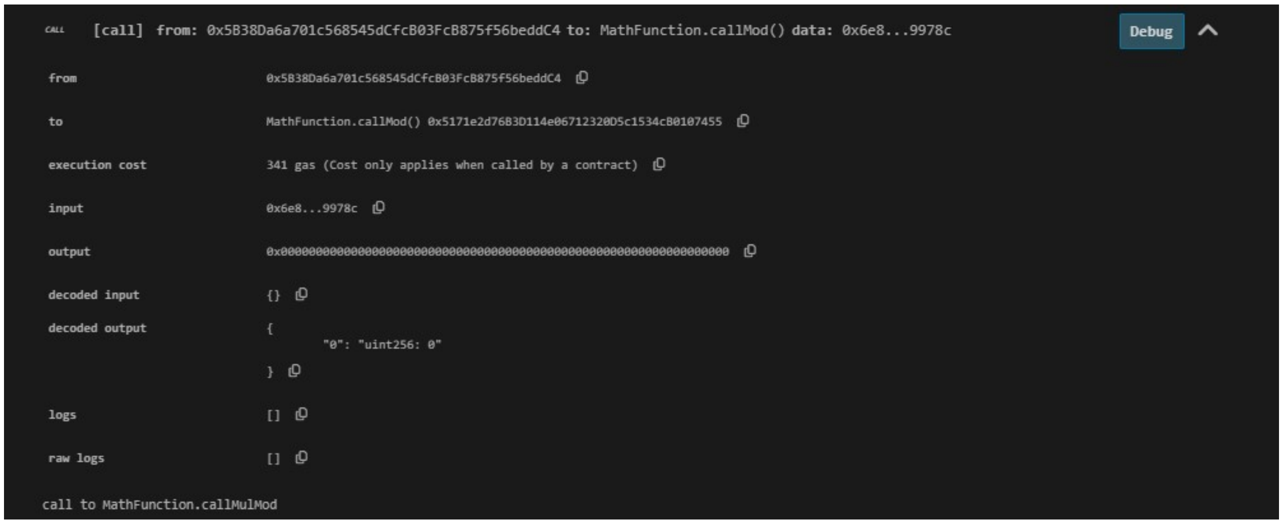
Output:



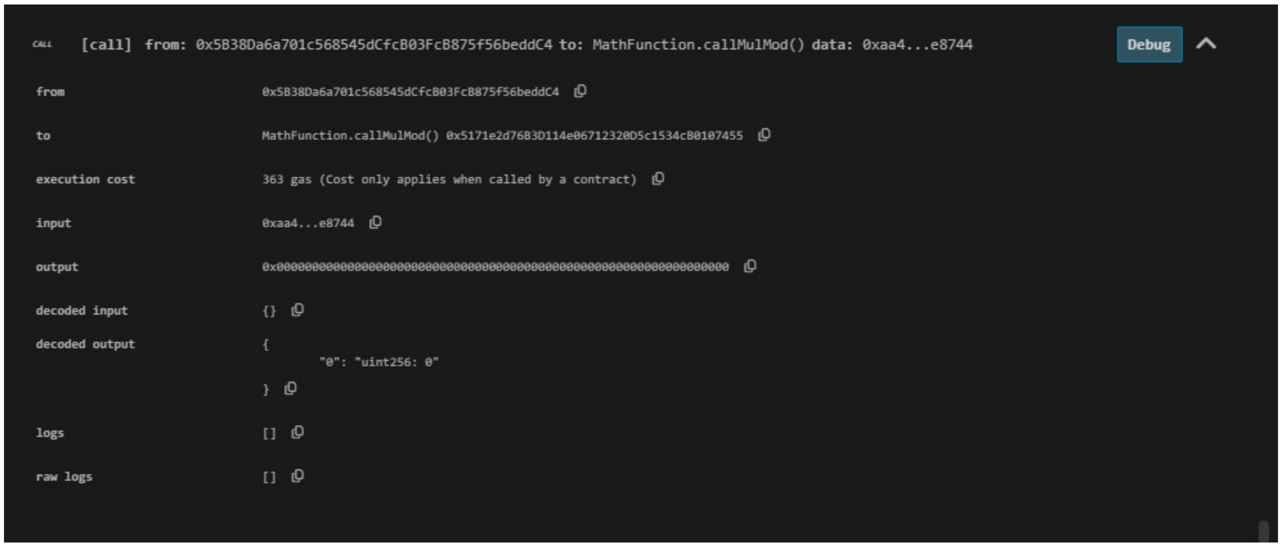
```
Deployed Contracts 1
MATHFUNCTION AT 0X517.
Balance: 0 ETH
callMod
0: uint256: 0
callMulMod
0: uint256: 0
```



```
[vm] from: 0x5B3...eddC4 to: MathFunction.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x3e4...265ae
status              0x1 Transaction mined and execution succeed
transaction hash    0x3e4f317e8d1e1c0917bb4fcf3cc33cdb1a1a2b199022cb0b4b089a91ede265ae
block hash          0x61955c53d7e60a88a30799307c1b71b5d6a21a686cc2e8ee6a8843183c153268
block number        161
contract address    0x5171e2d76B3D114e06712320D5c1534cB0107455
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  MathFunction.(constructor)
gas                 134701 gas
transaction cost    117131 gas
execution cost      59309 gas
input               0x608...a0033
output              0x6080604052348015600e575f80fd5b5060043610603057 5f3560e01c80636e89978c146034578063aa4e874414604e575b5f80fd5b603a
                    6068565b6040516045919060ae565b60405180910390f35b60546080565b604051605f919060ae565b60405180910390f35b5f6001806076
                    57607560c5565b5b60066004089050905 65b5f600180608e57608d60c5565b5b60066004099050905 65b5f8190509190505 65b60a8816098
```



```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MathFunction.callMod() data: 0x6e8...9978c
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                MathFunction.callMod() 0x5171e2d76B3D114e06712320D5c1534cB0107455
execution cost    341 gas (Cost only applies when called by a contract)
input             0x6e8...9978c
output            0x0000000000000000000000000000000000000000000000000000000000000000
decoded input     {}
decoded output    {
                          "0": "uint256: 0"
                  }
logs              []
raw logs          []

call to MathFunction.callMulMod
```



```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MathFunction.callMulMod() data: 0xaa4...e8744
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                MathFunction.callMulMod() 0x5171e2d76B3D114e06712320D5c1534cB0107455
execution cost    363 gas (Cost only applies when called by a contract)
input             0xaa4...e8744
output            0x0000000000000000000000000000000000000000000000000000000000000000
decoded input     {}
decoded output    {
                          "0": "uint256: 0"
                  }
logs              []
raw logs          []
```

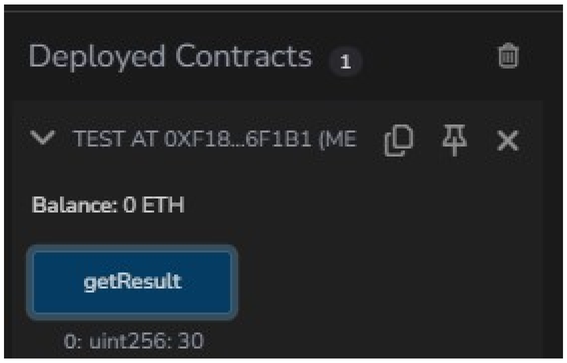
17. Write a solidity smart contract to demonstrate inheritance in contract.

Program:

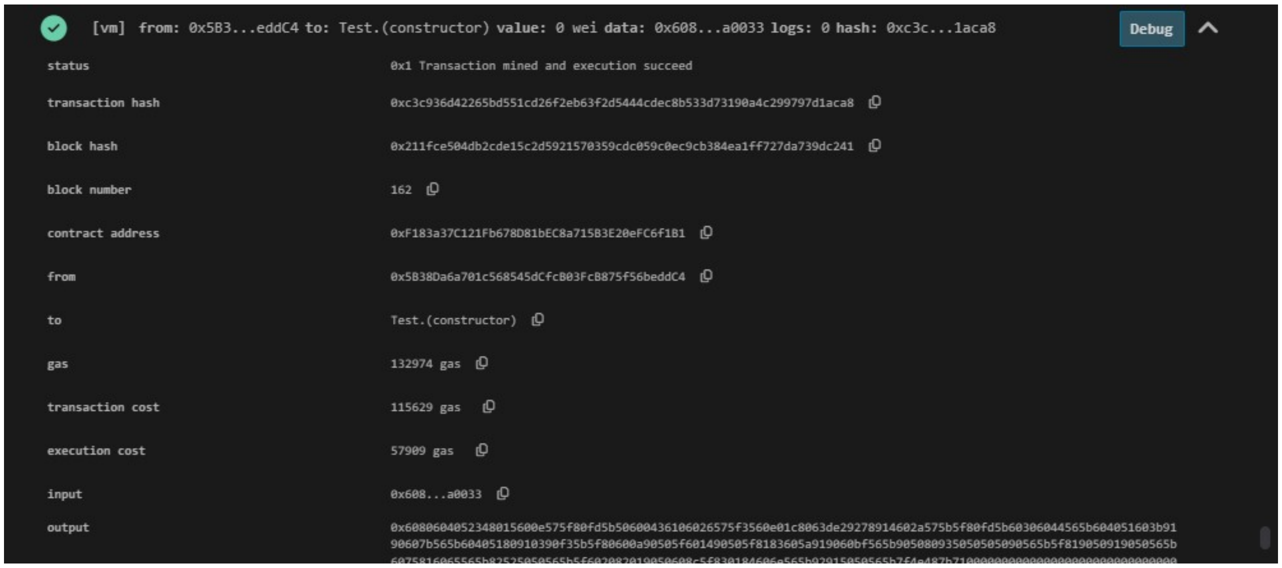
```
//SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;
```

```
interface ICalculator {
   function getResult() external view returns (uint256);
}

contract Test is ICalculator {
   constructor() {}

   function getResult() external pure override returns (uint256) {
      uint256 a = 10;
      uint256 b = 20;
      uint256 result = a + b;
      return result;
   }
}
```

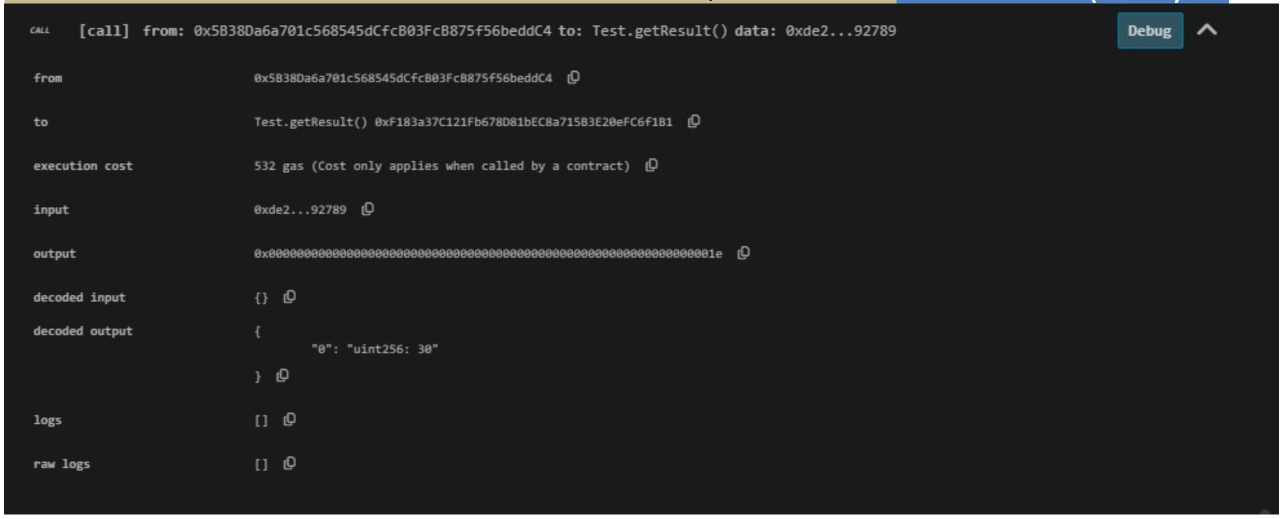
Output:



```
0: uint256: 30
```



```
[vm] from: 0x5B3...eddC4 to: Test.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xc3c...1aca8    Debug
status                 0x1 Transaction mined and execution succeed
transaction hash       0xc3c936d42265bd551cd26f2eb63f2d5444cdec8b533d73190a4c299797d1aca8
block hash             0x211fce504db2cde15c2d5921570359cdc059c0ec9cb384ea1ff727da739dc241
block number           162
contract address       0xF183a37C121Fb678D81bEC8a715B3E20eFC6f1B1
from                   0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                     Test.(constructor)
gas                    132974 gas
transaction cost       115629 gas
execution cost         57909 gas
input                  0x608...a0033
output                 0x6080604052348015600e575f80fd5b50600436106026575f3560e01c8063de29278914602a575b5f80fd5b60306044565b604051603b91
                       90607b565b60405180910390f35b5f80600a90505f601490505f8183605a919060bf565b9050809350505050905b565b5f819050919050565b
```



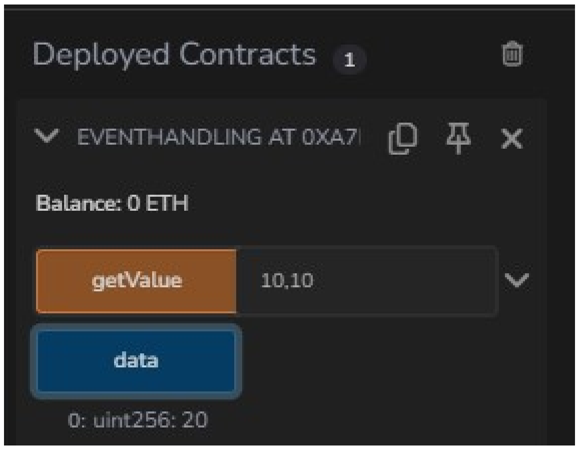
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Test.getResult() data: 0xde2...92789

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Test.getResult() 0xF183a37C121Fb678D81bEC8a715B3E20eFC6f1B1
execution cost    532 gas (Cost only applies when called by a contract)
input             0xde2...92789
output            0x000000000000000000000000000000000000000000000000000000000000001e
decoded input     {}
decoded output    {
                        "0": "uint256: 30"
                  }
logs              []
raw logs          []
```

18. Write a solidity smart contract to demonstrate events.

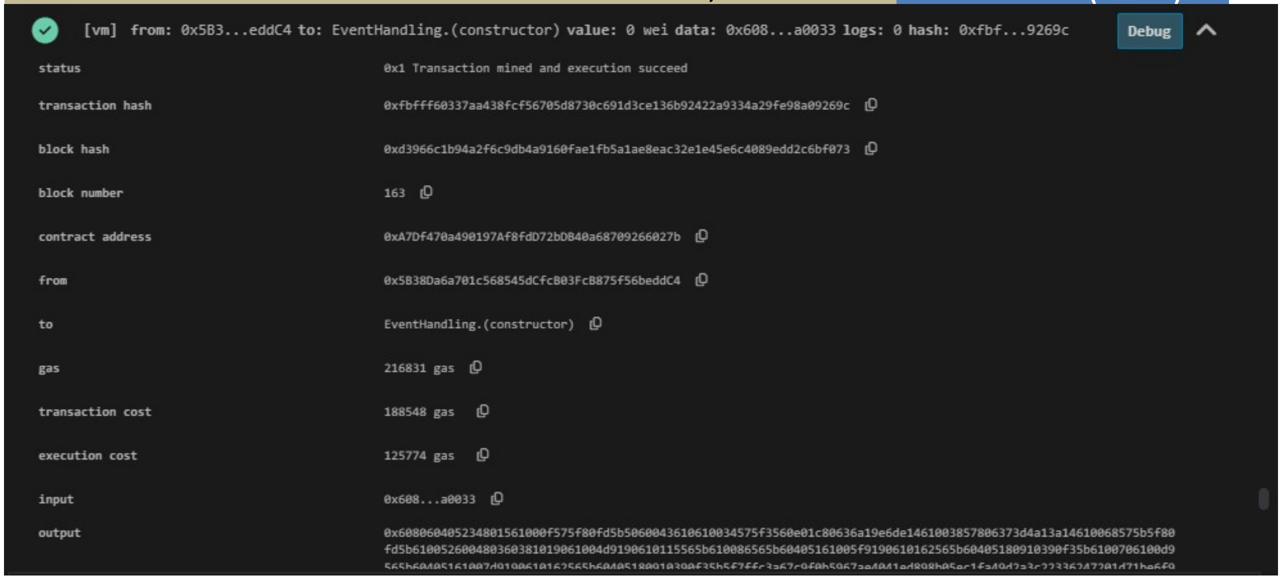
Program:

```
        //SPDX-License-Identifier: MIT

pragma solidity  >=0.5.0<0.8.27;

contract EventHandling{

   uint256 public data = 0;

   event Increment(address owner);

   function getValue(uint a, uint b) public returns(uint256){
      emit Increment(msg.sender);
      data  = a+b;
      return  data;
   }
}
```

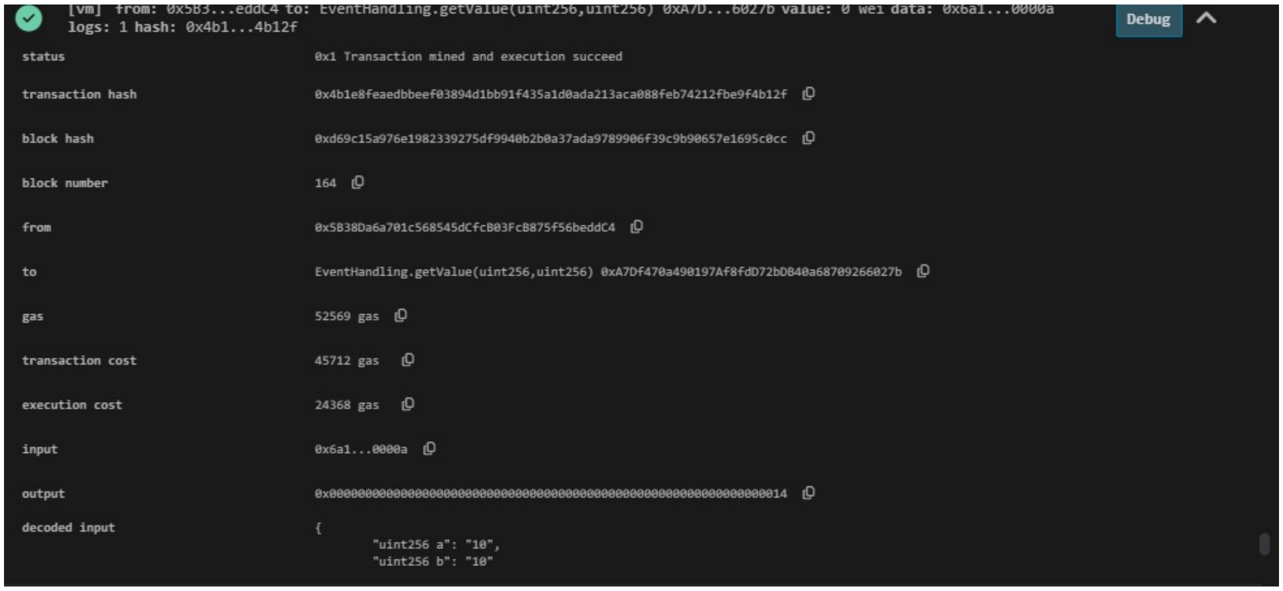
Output:



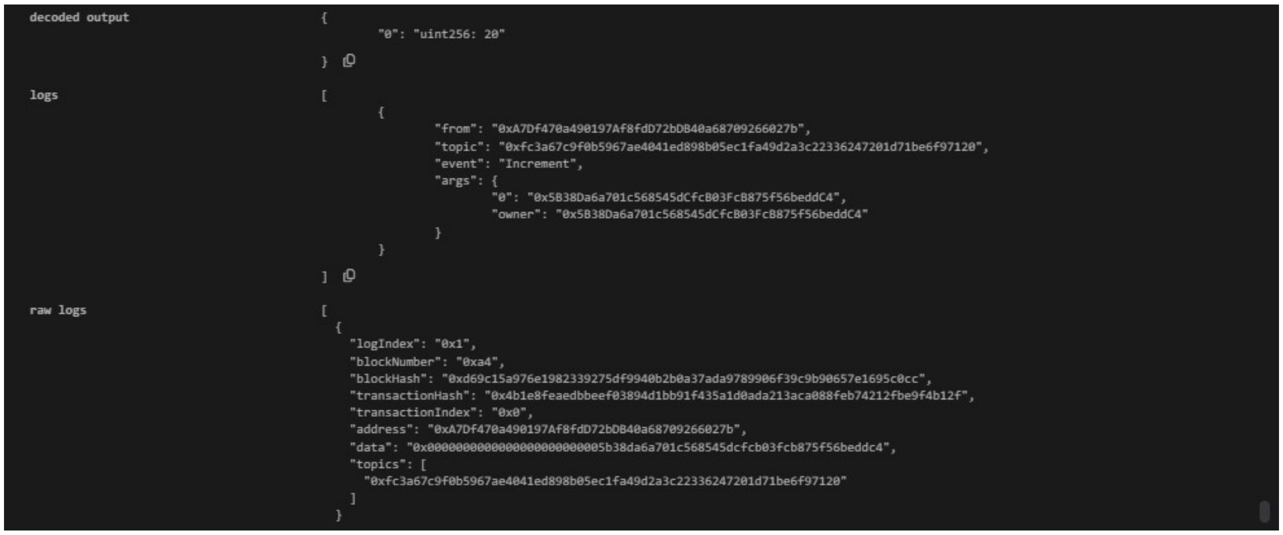
```
Deployed Contracts 1
EVENTHANDLING AT 0XA7
Balance: 0 ETH
getValue  10,10
data
0: uint256: 20
```



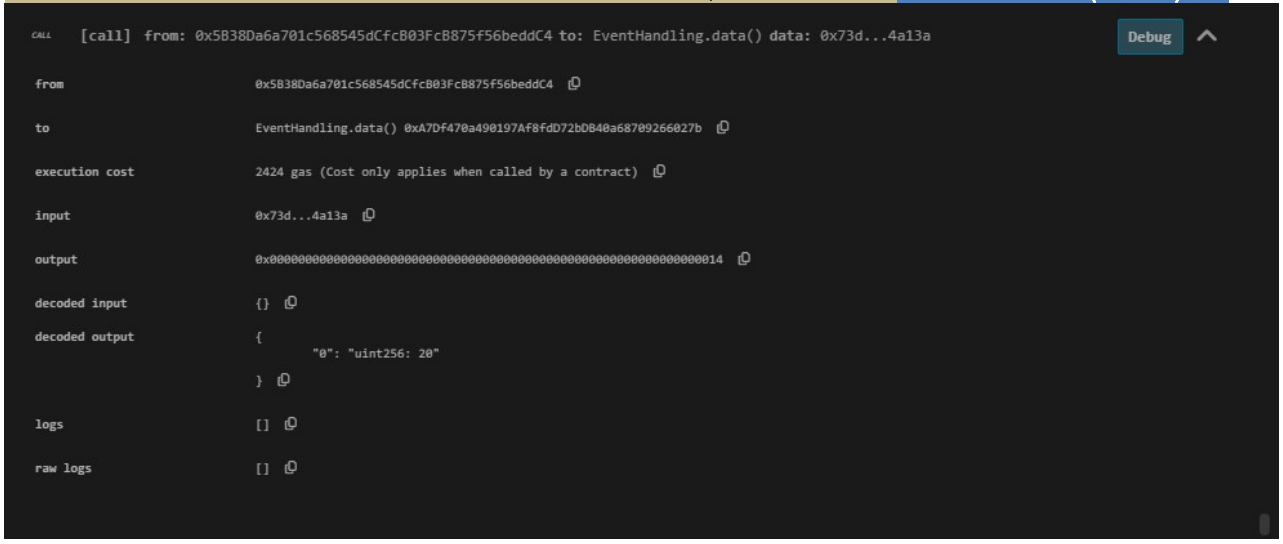
```
[vm] from: 0x5B3...eddC4 to: EventHandling.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xfbf...9269c

status              0x1 Transaction mined and execution succeed
transaction hash    0xfbfff60337aa438fcf56705d8730c691d3ce136b92422a9334a29fe98a09269c
block hash          0xd3966c1b94a2f6c9db4a9160fae1fb5a1ae8eac32e1e45e6c4089edd2c6bf073
block number        163
contract address    0xA7Df470a490197Af8fdD72bDB40a68709266027b
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  EventHandling.(constructor)
gas                 216831 gas
transaction cost    188548 gas
execution cost      125774 gas
input               0x608...a0033
output              0x6080604052348015610 00f575f80fd5b5060043610610034575f3560e01c80636a19e6de1461003857806373d4a13a14610068575b5f80
                    fd5b610052600480360381019061004d9190610115565b610086565b60405161005f9190610162565b60405180910390f35b6100706100d9
                    565b60405161007d9190610162565b60405180910390f35b5f7ffc3a67c9f0b5967ae4041ed898b05ec1fa49d2a3c22336247201d71be6f9
```



```
[vm] from: 0x5B3...eddC4 to: EventHandling.getValue(uint256,uint256) 0xA7D...6027b value: 0 wei data: 0x6a1...0000a
logs: 1 hash: 0x4b1...4b12f

status              0x1 Transaction mined and execution succeed
transaction hash    0x4b1e8feaedbbeef03894d1bb91f435a1d0ada213aca088feb74212fbe9f4b12f
block hash          0xd69c15a976e1982339275df9940b2b0a37ada9789906f39c9b90657e1695c0cc
block number        164
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  EventHandling.getValue(uint256,uint256) 0xA7Df470a490197Af8fdD72bDB40a68709266027b
gas                 52569 gas
transaction cost    45712 gas
execution cost      24368 gas
input               0x6a1...0000a
output              0x0000000000000000000000000000000000000000000000000000000000000014
decoded input       {
                            "uint256 a": "10",
                            "uint256 b": "10"
```



```
decoded output      {
                            "0": "uint256: 20"
                    }
logs                [
                            {
                                    "from": "0xA7Df470a490197Af8fdD72bDB40a68709266027b",
                                    "topic": "0xfc3a67c9f0b5967ae4041ed898b05ec1fa49d2a3c22336247201d71be6f97120",
                                    "event": "Increment",
                                    "args": {
                                            "0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                            "owner": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                                    }
                            }
                    ]
raw logs            [
                      {
                        "logIndex": "0x1",
                        "blockNumber": "0xa4",
                        "blockHash": "0xd69c15a976e1982339275df9940b2b0a37ada9789906f39c9b90657e1695c0cc",
                        "transactionHash": "0x4b1e8feaedbbeef03894d1bb91f435a1d0ada213aca088feb74212fbe9f4b12f",
                        "transactionIndex": "0x0",
                        "address": "0xA7Df470a490197Af8fdD72bDB40a68709266027b",
                        "data": "0x0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4",
                        "topics": [
                          "0xfc3a67c9f0b5967ae4041ed898b05ec1fa49d2a3c22336247201d71be6f97120"
                        ]
                      }
```



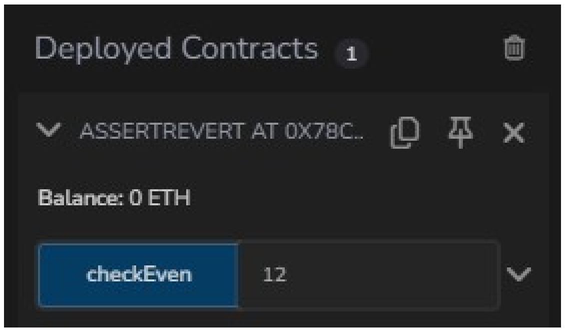
```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: EventHandling.data() data: 0x73d...4a13a    Debug

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                EventHandling.data() 0xA7Df470a490197Af8fdD72bDB40a68709266027b
execution cost    2424 gas (Cost only applies when called by a contract)
input             0x73d...4a13a
output            0x0000000000000000000000000000000000000000000000000000000000000014
decoded input     {}
decoded output    {
                        "0": "uint256: 20"
                  }
logs              []
raw logs          []
```

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

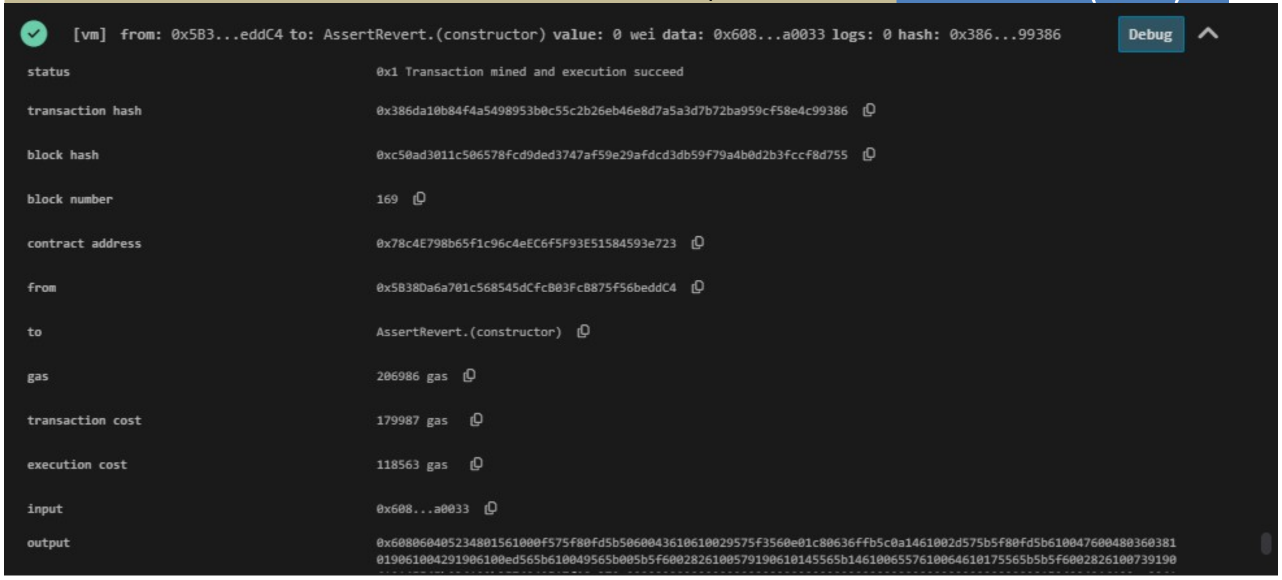
Program:

```
//SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract AssertRevert {
   function checkEven(uint256 _number) public pure {
      assert(_number%2==0);
      if (_number % 2 == 0) {
         revert("Number is Even");
      }
   }
}
```

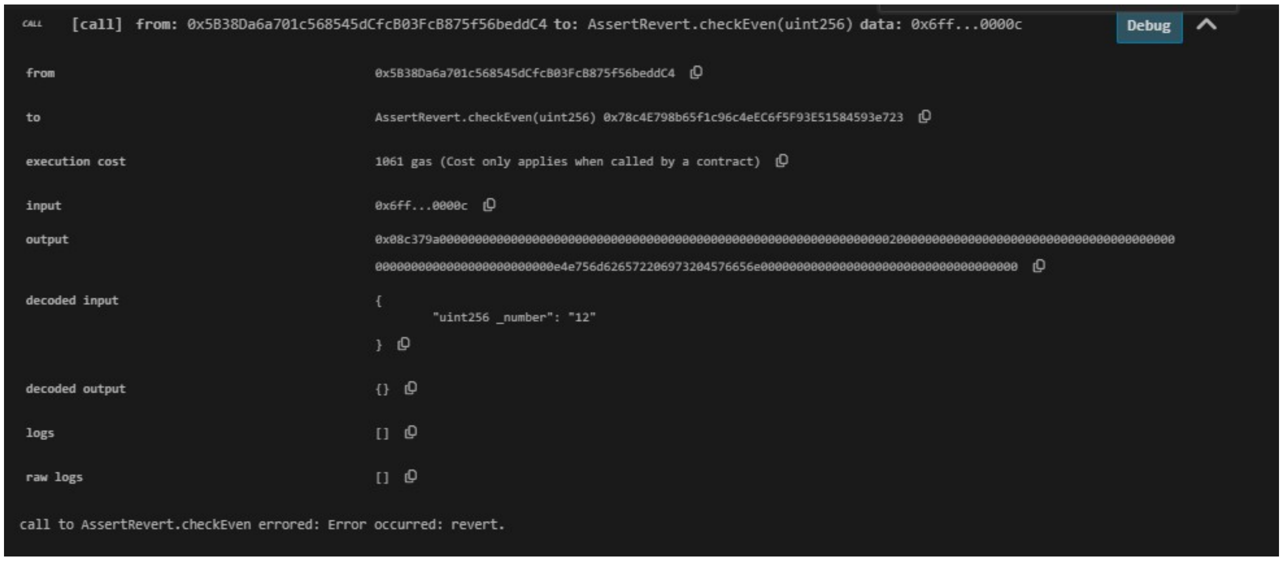
Output:



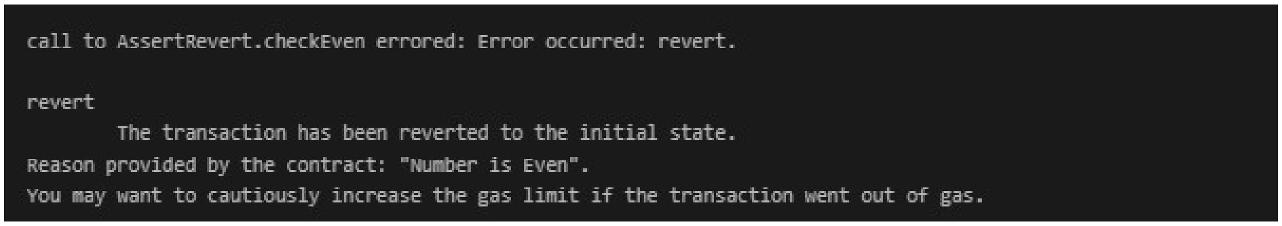
```
Deployed Contracts 1
ASSERTREVERT AT 0X78C...
Balance: 0 ETH
checkEven    12
```



```
[vm] from: 0x5B3...eddC4 to: AssertRevert.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x386...99386   Debug

status               0x1 Transaction mined and execution succeed
transaction hash     0x386da10b84f4a5498953b0c55c2b26eb46e8d7a5a3d7b72ba959cf58e4c99386
block hash           0xc50ad3011c506578fcd9ded3747af59e29afdcd3db59f79a4b0d2b3fccf8d755
block number         169
contract address     0x78c4E798b65f1c96c4eEC6f5F93E51584593e723
from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   AssertRevert.(constructor)
gas                  206986 gas
transaction cost     179987 gas
execution cost       118563 gas
input                0x608...a0033
output               0x6080604052348015610000f575f80fd5b5060043610610029575f3560e01c80636ffb5c0a1461002d575b5f80fd5b610047600480360381
                     019061004291906100ed565b610049565b005b5f6002826100579190610145565b14610065576100646101755565b5b5f6002826100739190
```



```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AssertRevert.checkEven(uint256) data: 0x6ff...0000c   Debug

from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   AssertRevert.checkEven(uint256) 0x78c4E798b65f1c96c4eEC6f5F93E51584593e723
execution cost       1061 gas (Cost only applies when called by a contract)
input                0x6ff...0000c
output               0x08c379a000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000
                     000000000000000000000000e4e756d6265722069732045766656e0000000000000000000000000000000000000
decoded input        {
                             "uint256 _number": "12"
                     }
decoded output       {}
logs                 []
raw logs             []
call to AssertRevert.checkEven errored: Error occurred: revert.
```



```
call to AssertRevert.checkEven errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Number is Even".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

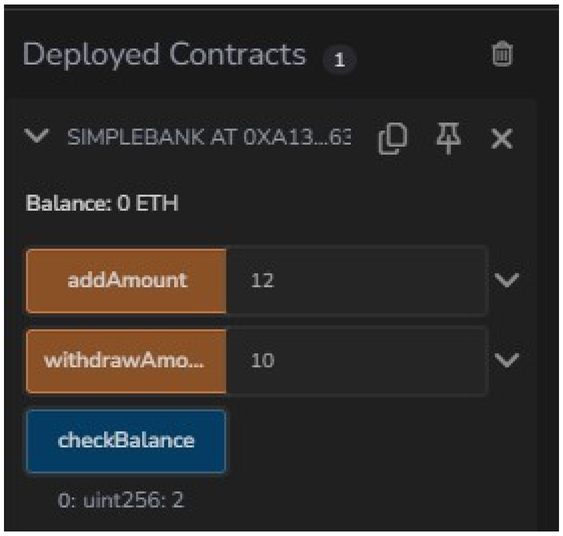
20. Write a solidity smart contract for Bank Account which provides operations such as check account balance, withdraw amount and deposit amount etc.

Program:

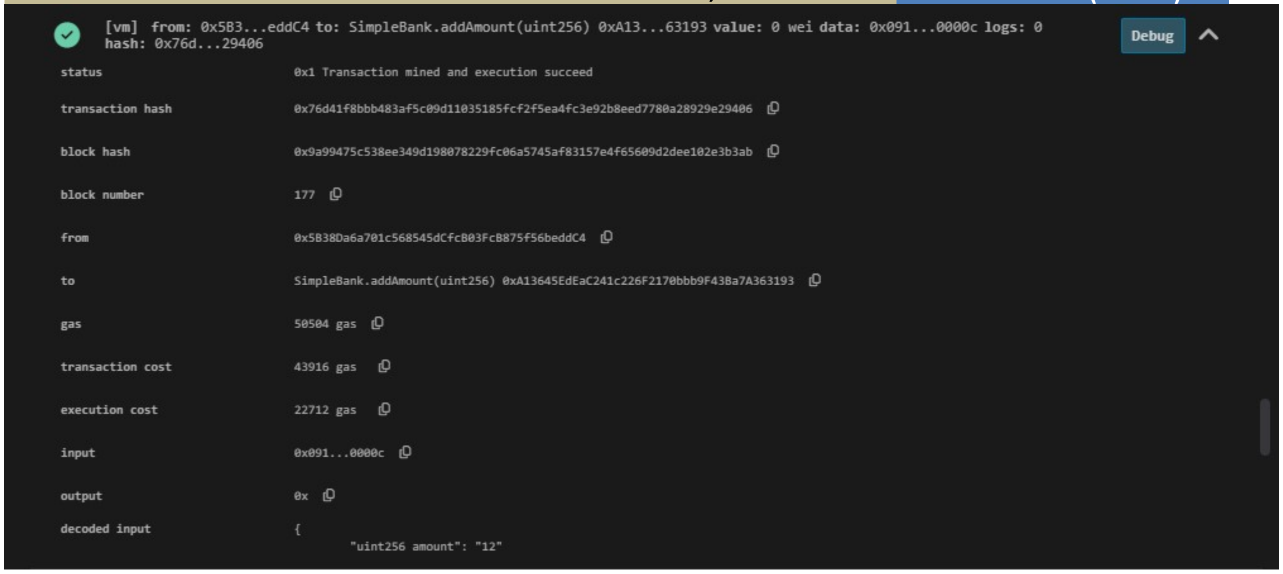
Output:

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;
contract SimpleBank {
  // State variable to store the balance
```

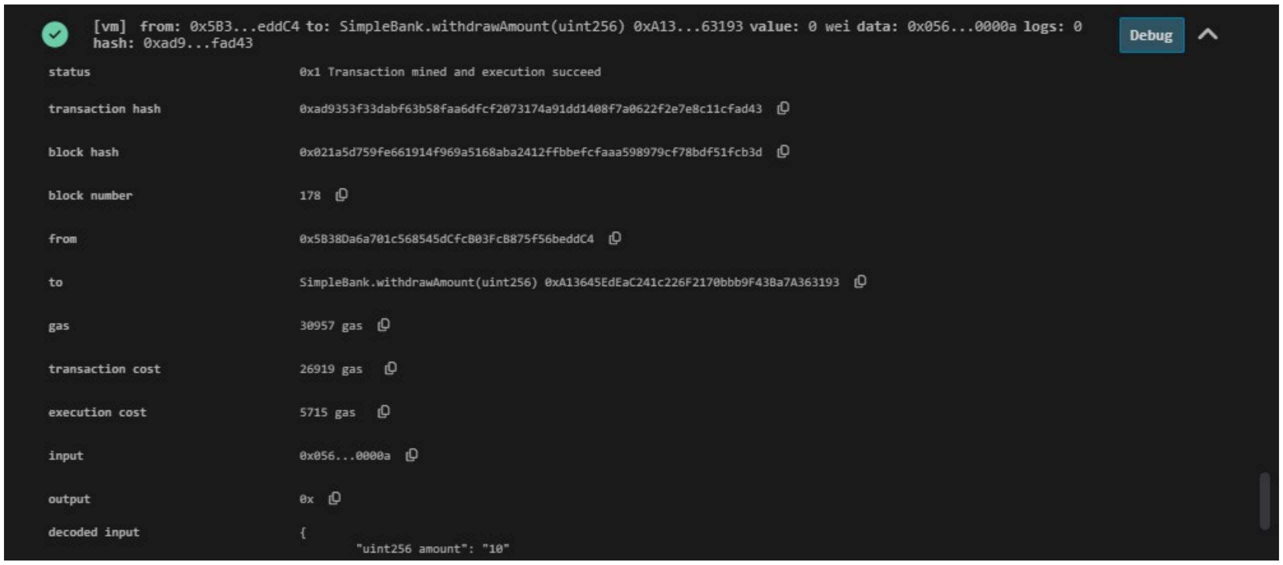
```
    uint256 private balance;
    // Constructor to ini alize balance
    constructor() {
        balance = 0;
    }
    // Func on to add (deposit) amount to the balance
    function addAmount(uint256 amount) public {
        balance += amount;
    }
    // Func on to withdraw amount from the balance
    function withdrawAmount(uint256 amount) public {
        require(amount <= balance, "Insufficient balance");
        balance -= amount;
    }
    // Func on to check the remaining balance
    function checkBalance() public view returns (uint256) {
        return balance;
    }
}
```



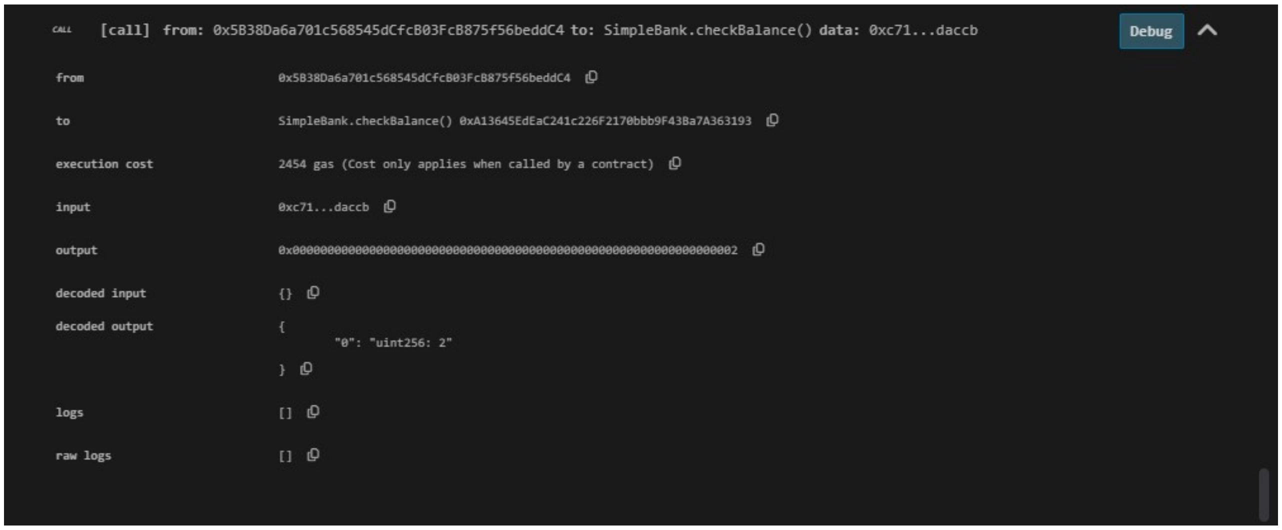
```
Balance: 0 ETH
addAmount 12
withdrawAmo... 10
0: uint256: 2
```



```
[vm] from: 0x5B3...eddC4 to: SimpleBank.addAmount(uint256) 0xA13...63193 value: 0 wei data: 0x091...0000c logs: 0
hash: 0x76d...29406
status              0x1 Transaction mined and execution succeed
transaction hash    0x76d41f8bbb483af5c09d11035185fcf2f5ea4fc3e92b8eed7780a28929e29406
block hash          0x9a99475c538ee349d198078229fc06a5745af83157e4f65609d2dee102e3b3ab
block number        177
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  SimpleBank.addAmount(uint256) 0xA13645EdEaC241c226F2170bbb9F43Ba7A363193
gas                 50504 gas
transaction cost    43916 gas
execution cost      22712 gas
input               0x091...0000c
output              0x
decoded input       {
                            "uint256 amount": "12"
```



```
[vm] from: 0x5B3...eddC4 to: SimpleBank.withdrawAmount(uint256) 0xA13...63193 value: 0 wei data: 0x056...0000a logs: 0
hash: 0xad9...fad43
status              0x1 Transaction mined and execution succeed
transaction hash    0xad9353f33dabf63b58faa6dfcf2073174a91dd1408f7a0622f2e7e8c11cfad43
block hash          0x021a5d759fe661914f969a5168aba2412ffbbefcfaaa598979cf78bdf51fcb3d
block number        178
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  SimpleBank.withdrawAmount(uint256) 0xA13645EdEaC241c226F2170bbb9F43Ba7A363193
gas                 30957 gas
transaction cost    26919 gas
execution cost      5715 gas
input               0x056...0000a
output              0x
decoded input       {
                            "uint256 amount": "10"
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SimpleBank.checkBalance() data: 0xc71...daccb
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  SimpleBank.checkBalance() 0xA13645EdEaC241c226F2170bbb9F43Ba7A363193
execution cost      2454 gas (Cost only applies when called by a contract)
input               0xc71...daccb
output              0x0000000000000000000000000000000000000000000000000000000000000002
decoded input       {}
decoded output      {
                            "0": "uint256: 2"
                    }
logs                []
raw logs            []
```

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

I) Add information of 5 students.

ii) Search for a student using Roll no

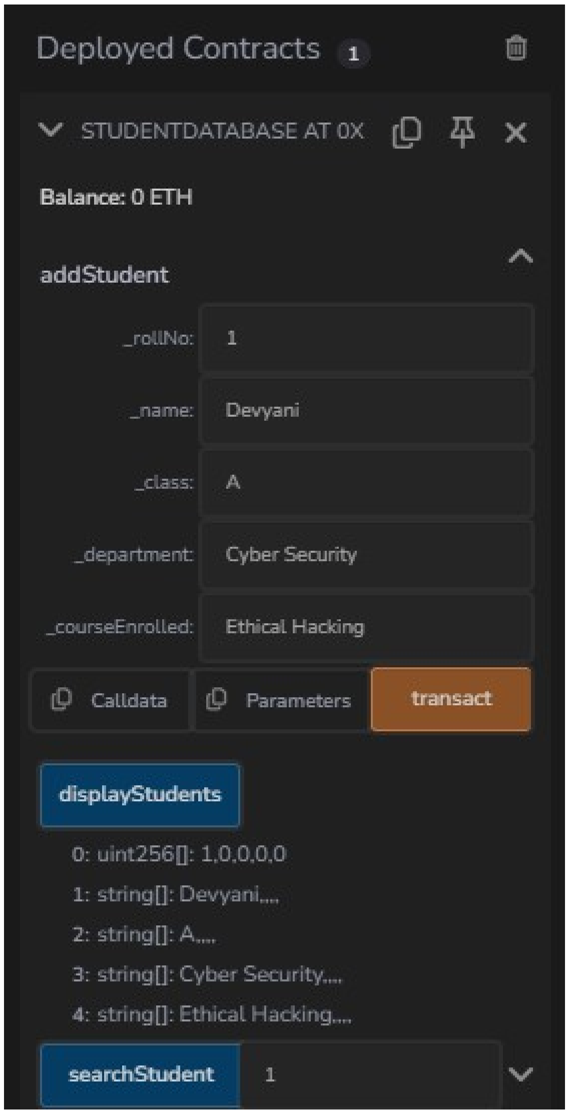
iii) Display all information

Program:

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract StudentDatabase {
   // Define a struct to represent a student
   struct Student {
      uint256 rollNo;
      string name;
      string class_;
      string department;
      string courseEnrolled;
   }

   // Create a mapping to store students by roll number
   mapping(uint256 => Student) public students;

   // Function to add a student
   function addStudent(uint256 _rollNo, string memory _name, string memory _class, string memory _department,
string memory _courseEnrolled) public {
      // Check if the student already exists
      require(students[_rollNo].rollNo == 0, "Student already exists");

      // Create a new student and add it to the mapping
      students[_rollNo] = Student(_rollNo, _name, _class, _department, _courseEnrolled);
   }

   // Function to search for a student by roll number
   function searchStudent(uint256 _rollNo) public view returns (string memory, string memory, string memory, string
memory) {
      // Check if the student exists
      require(students[_rollNo].rollNo != 0, "Student not found");

      // Return the student's information
      return (students[_rollNo].name, students[_rollNo].class_, students[_rollNo].department,
students[_rollNo].courseEnrolled);
   }

   // Function to display all students
   function displayStudents() public view returns (uint256[] memory, string[] memory, string[] memory, string[]
memory, string[] memory) {
      // Create arrays to store the student information
      uint256[] memory rollNos = new uint256[](5);
      string[] memory names = new string[](5);
      string[] memory classes = new string[](5);
      string[] memory departments = new string[](5);
      string[] memory coursesEnrolled = new string[](5);

      // Iterate over the students and add their information to the arrays
      uint256 count = 0;
      for (uint256 i = 1; i <= 5; i++) {
         if (students[i].rollNo != 0) {
            rollNos[count] = students[i].rollNo;
            names[count] = students[i].name;
            classes[count] = students[i].class_;
            departments[count] = students[i].department;
```

```
                coursesEnrolled[count] = students[i].courseEnrolled;
                count++;
            }
        }

        // Return the student information
        return (rollNos, names, classes, departments, coursesEnrolled);
    }
}
```

Output:



Deployed Contracts 1

STUDENTDATABASE AT 0X

Balance: 0 ETH

addStudent

_rollNo: 1

_name: Devyani

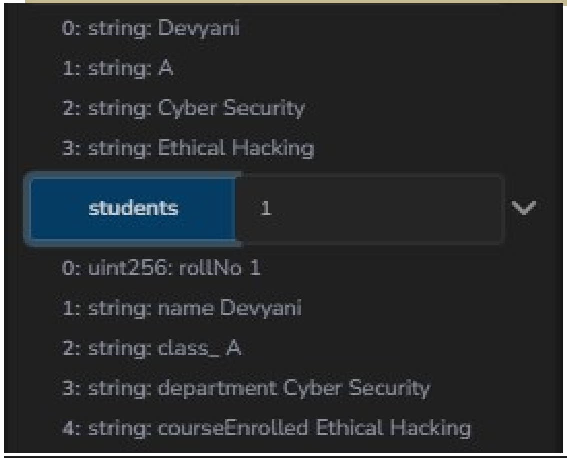
_class: A

_department: Cyber Security

_courseEnrolled: Ethical Hacking

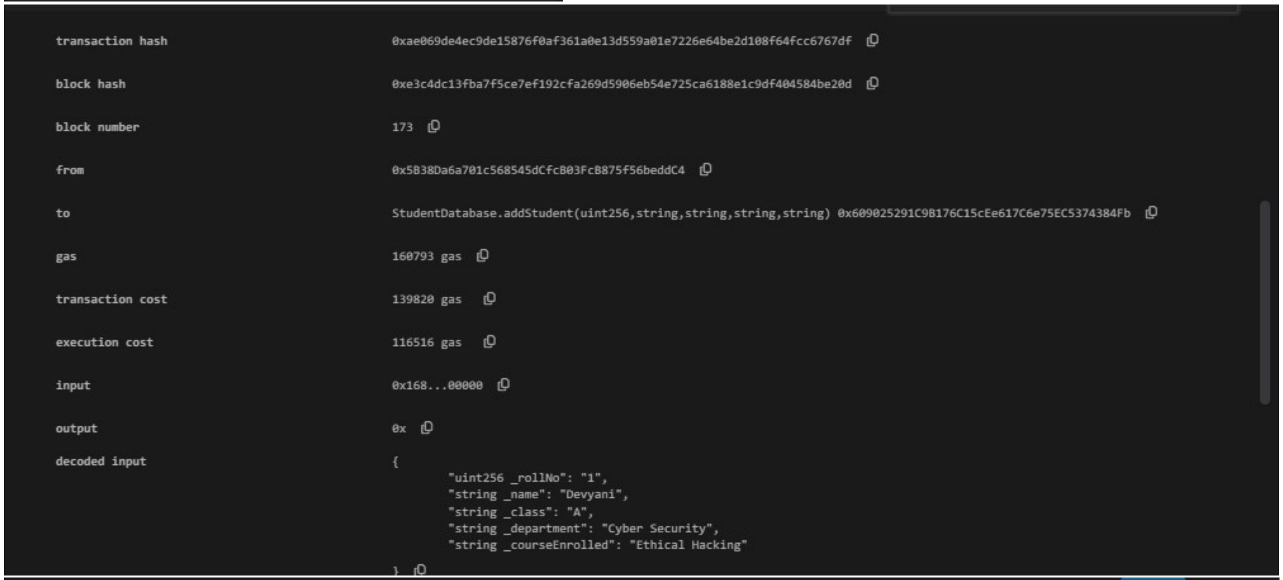
```
0: uint256[]: 1,0,0,0,0
1: string[]: Devyani,,,,
2: string[]: A,,,,
3: string[]: Cyber Security,,,,
4: string[]: Ethical Hacking,,,,
```

1

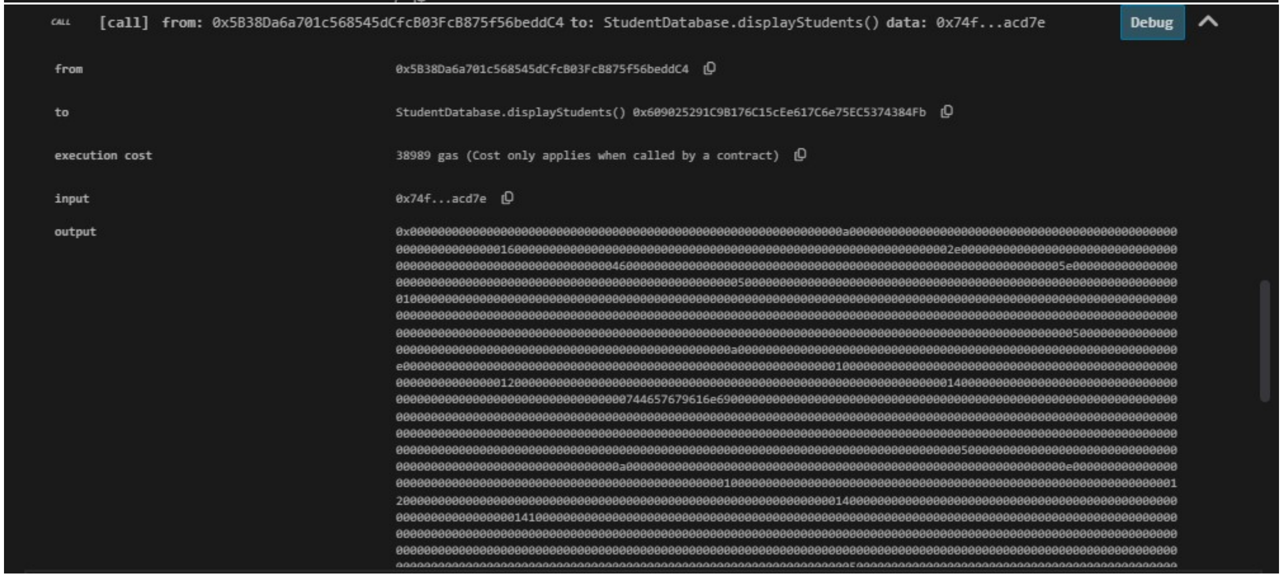


```
0: string: Devyani
1: string: A
2: string: Cyber Security
3: string: Ethical Hacking
```

```
0: uint256: rollNo 1
1: string: name Devyani
2: string: class_ A
3: string: department Cyber Security
4: string: courseEnrolled Ethical Hacking
```

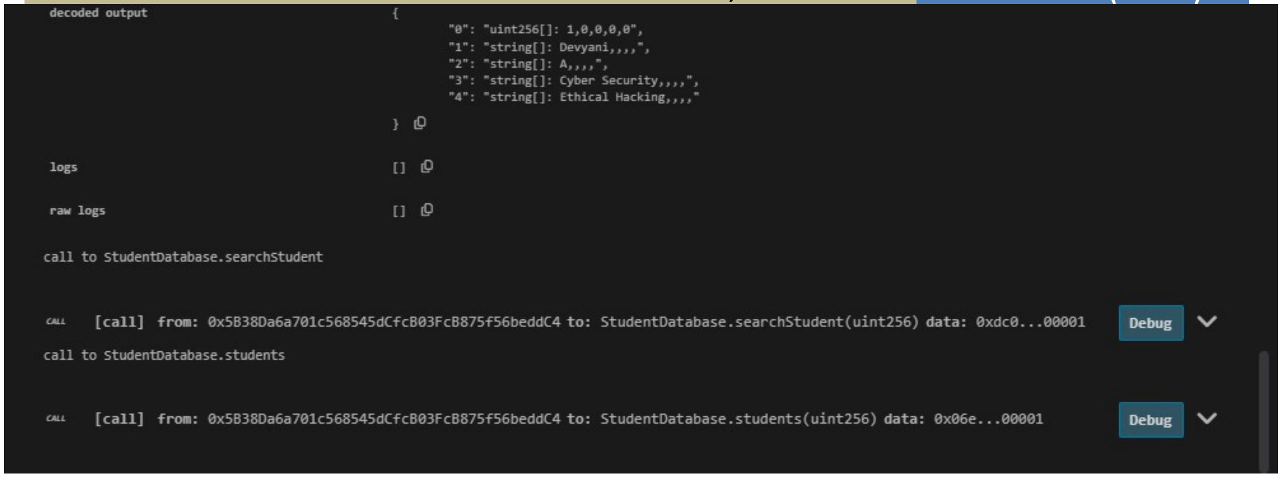


```
transaction hash        0xae069de4ec9de15876f0af361a0e13d559a01e7226e64be2d108f64fcc6767df
block hash              0xe3c4dc13fba7f5ce7ef192cfa269d5906eb54e725ca6188e1c9df404584be20d
block number            173
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      StudentDatabase.addStudent(uint256,string,string,string,string) 0x609025291C9B176C15cEe617C6e75EC5374384Fb
gas                     160793 gas
transaction cost        139820 gas
execution cost          116516 gas
input                   0x168...00000
output                  0x
decoded input           {
                                "uint256 _rollNo": "1",
                                "string _name": "Devyani",
                                "string _class": "A",
                                "string _department": "Cyber Security",
                                "string _courseEnrolled": "Ethical Hacking"
                        }
```



```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StudentDatabase.displayStudents() data: 0x74f...acd7e
```

```
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      StudentDatabase.displayStudents() 0x609025291C9B176C15cEe617C6e75EC5374384Fb
execution cost          38989 gas (Cost only applies when called by a contract)
input                   0x74f...acd7e
output                  0x00000000000000000000000000000000000000000000000000000000000000a00000000000000000000000000000000000000000000000
                        0000000000000001600000000000000000000000000000000000000000000000000000000000002e00000000000000000000000000000000
                        0000000000000000000000000000000460000000000000000000000000000000000000000000000000000000000005e000000000000000
                        0000000000000000000000000000000000000000000000000500000000000000000000000000000000000000000000000000000000000000
                        0100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000500000000000000
                        000000000000000000000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000000
                        e000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000
                        0000000000000001200000000000000000000000000000000000000000000000000000000000001400000000000000000000000000000000
                        000000000000000000000000000000000744657679616e69000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000005000000000000000000000000000000
                        00000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000000e000000000000000
                        0000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000001
                        2000000000000000000000000000000000000000000000000000000000000001400000000000000000000000000000000000000000000000
                        0000000000000000014100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
                        0000000000000000000000000000000000000000000000000000000000000000050000000000000000000000000000000000000000000000
```



```
decoded output                    {
                                          "0": "uint256[]: 1,0,0,0,0",
                                          "1": "string[]: Devyani,,,,",
                                          "2": "string[]: A,,,,",
                                          "3": "string[]: Cyber Security,,,,",
                                          "4": "string[]: Ethical Hacking,,,,"
                                  }

logs                              []

raw logs                          []

call to StudentDatabase.searchStudent

CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StudentDatabase.searchStudent(uint256) data: 0xdc0...00001
call to StudentDatabase.students

CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: StudentDatabase.students(uint256) data: 0x06e...00001
```

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

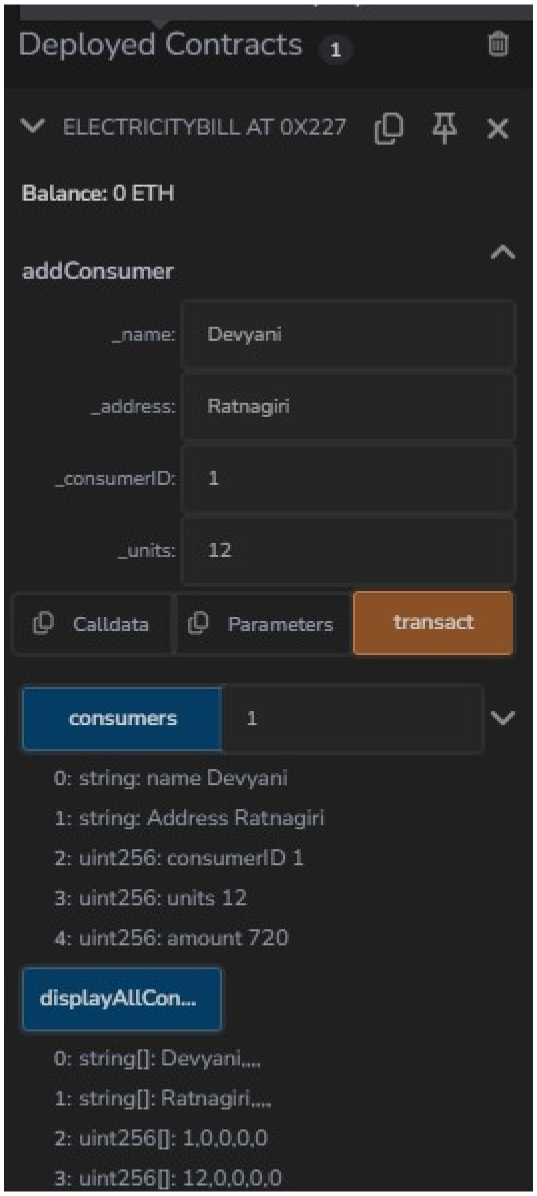
For first 50 units Rs. 0.50/unit. For next 100 units Rs. 0.75/unit. For next 100 units Rs. 1.20/unit. For unit above 250 Rs. 50/unit. An additional surcharge of 20% is added to the bill. Display the information of 5 such consumers along with their units consumed and amount.

Program:

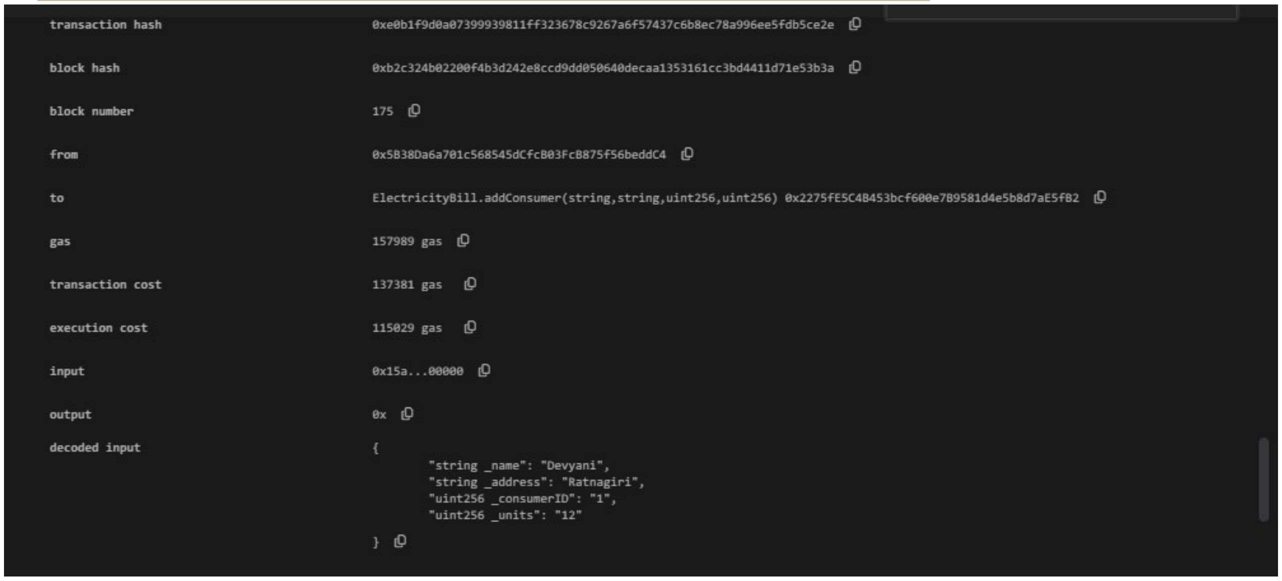
```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract ElectricityBill {
    // Define a struct to represent a consumer
    struct Consumer {
        string name;
        string Address;
        uint256 consumerID;
        uint256 units;
        uint256 amount;
    }

    // Create a mapping to store consumers by consumer ID
    mapping(uint256 => Consumer) public consumers;

    // Function to calculate the electricity bill
    function calculateBill(uint256 _units) internal pure returns (uint256) {
        uint256 bill;

        // Calculate the bill for the first 50 units
        if (_units <= 50) {
            bill = _units * 50; // Rs. 0.50/unit
        } else {
            bill = 50 * 50; // Rs. 0.50/unit for the first 50 units

            // Calculate the bill for the next 100 units
            uint256 remainingUnits = _units - 50;
            if (remainingUnits <= 100) {
                bill += remainingUnits * 75; // Rs. 0.75/unit
            } else {
                bill += 100 * 75; // Rs. 0.75/unit for the next 100 units
                // Calculate the bill for the next 100 units
                remainingUnits -= 100;
                if (remainingUnits <= 100) {
                    bill += remainingUnits * 120; // Rs. 1.20/unit
                } else {
                    bill += 100 * 120; // Rs. 1.20/unit for the next 100 units

                    // Calculate the bill for units above 250
                    remainingUnits -= 100;
                    bill += remainingUnits * 1500; // Rs. 15.00/unit
                }
            }
        }

        // Add a 20% surcharge to the bill
        bill += (bill * 20) / 100;

        return bill;
    }

    // Function to add a consumer
    function addConsumer(string memory _name, string memory _address, uint256 _consumerID,
uint256 _units) public {
        // Check if the consumer already exists
        require(consumers[_consumerID].consumerID == 0, "Consumer already exists");

        // Calculate the electricity bill
        uint256 amount = calculateBill(_units);

        // Create a new consumer and add it to the mapping
        consumers[_consumerID] = Consumer(_name, _address, _consumerID, _units, amount);
    }

    // Function to display consumer information
    function displayConsumer(uint256 _consumerID) public view returns (string memory, string
memory, uint256, uint256, uint256) {
        // Check if the consumer exists
        require(consumers[_consumerID].consumerID != 0, "Consumer not found");

        // Return the consumer information
        return (consumers[_consumerID].name, consumers[_consumerID].Address,
consumers[_consumerID].consumerID, consumers[_consumerID].units,
consumers[_consumerID].amount);
    }

    // Function to display all consumers
    function displayAllConsumers() public view returns (string[] memory, string[] memory, uint256[]
memory, uint256[] memory, uint256[] memory) {
        // Create arrays to store the consumer information
        string[] memory names = new string[](5);
        string[] memory addresses = new string[](5);
        uint256[] memory consumerIDs = new uint256[](5);
        uint256[] memory units = new uint256[](5);
        uint256[] memory amounts = new uint256[](5);

        // Iterate over the consumers and add their information to the arrays
        uint256 count = 0;
```

```
            for (uint256 i = 1; i <= 5; i++) {
                if (consumers[i].consumerID != 0) {
                    names[count] = consumers[i].name;
                    addresses[count] = consumers[i].Address;
                    consumerIDs[count] = consumers[i].consumerID;
                    units[count] = consumers[i].units;
                    amounts[count] = consumers[i].amount;
                    count++;
                }
            }

            // Return the consumer information
            return (names, addresses, consumerIDs, units, amounts);
        }
    }
```

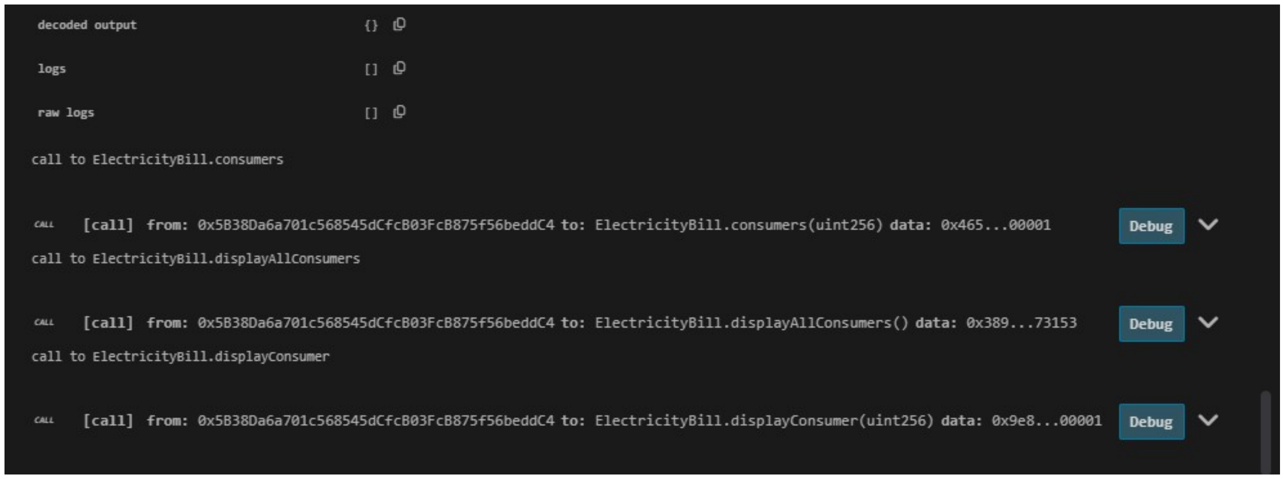
Output:



```
Balance: 0 ETH
_name: Devyani
_address: Ratnagiri
_consumerID: 1
_units: 12
consumers 1
0: string: name Devyani
1: string: Address Ratnagiri
2: uint256: consumerID 1
3: uint256: units 12
4: uint256: amount 720
0: string[]: Devyani,,,,
1: string[]: Ratnagiri,,,,
2: uint256[]: 1,0,0,0,0
3: uint256[]: 12,0,0,0,0
```



```
transaction hash        0xe0b1f9d0a07399939811ff323678c9267a6f57437c6b8ec78a996ee5fdb5ce2e
block hash              0xb2c324b02200f4b3d242e8ccd9dd050640decaa1353161cc3bd4411d71e53b3a
block number            175
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      ElectricityBill.addConsumer(string,string,uint256,uint256) 0x2275fE5C4B453bcf600e7B9581d4e5b8d7aE5fB2
gas                     157989 gas
transaction cost        137381 gas
execution cost          115029 gas
input                   0x15a...00000
output                  0x
decoded input           {
                                "string _name": "Devyani",
                                "string _address": "Ratnagiri",
                                "uint256 _consumerID": "1",
                                "uint256 _units": "12"
                        }
```



```
decoded output          {}
logs                    []
raw logs                []

call to ElectricityBill.consumers

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ElectricityBill.consumers(uint256) data: 0x465...00001
call to ElectricityBill.displayAllConsumers

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ElectricityBill.displayAllConsumers() data: 0x389...73153
call to ElectricityBill.displayConsumer

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ElectricityBill.displayConsumer(uint256) data: 0x9e8...00001
```

Conclusion: Implemented a smart contracts using solidity in Ethereum.