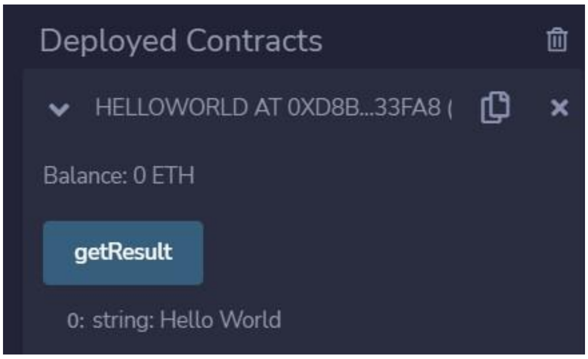
Practical No.3

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

```
pragma solidity ^0.5.0;  contract HelloWorld {

 constructor () public {

 }

 function getResult() public view returns(string memory){

 return 'Hello World';

 }

}
```

Output:



HELLOWORLD AT 0XD8B...33FA8 (

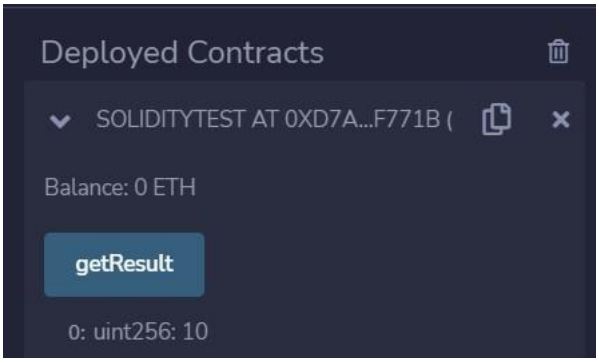
Balance: 0 ETH

0: string: Hello World

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

```
pragma solidity ^0.5.0;  contract SolidityTest {   uint
storedData; // State variable   constructor() public {

 storedData = 10;

 }

 function getResult() public view returns(uint){

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

 return storedData; //access the state variable

 }

}
```

Output:



SOLIDITYTEST AT 0XD7A...F771B (

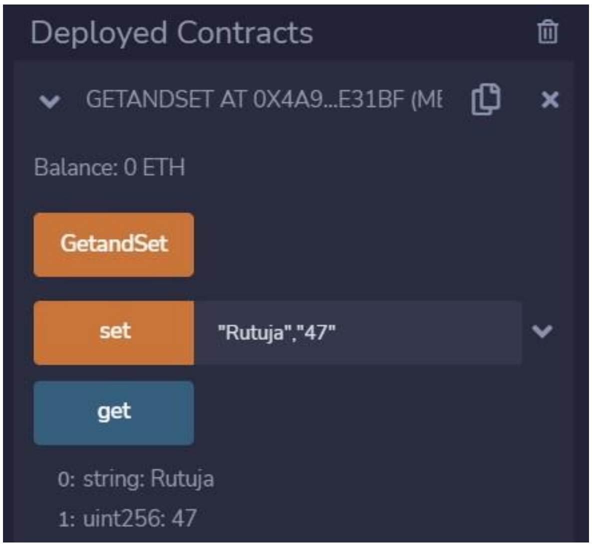
Balance: 0 ETH

0: uint256: 10

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

```
pragma solidity ^0.5.0;  contract
GetAndSet{   string name;   uint age;

 function GetandSet() public {

 }

 function set(string memory newName, uint newAge) public {   name = newName;

 age = newAge;

 }

 function get() public view returns (string memory, uint) {   return
(name,age);

 }

}
```

Output:



Deployed Contracts

GETANDSET AT 0X4A9...E31BF (ME

Balance: 0 ETH

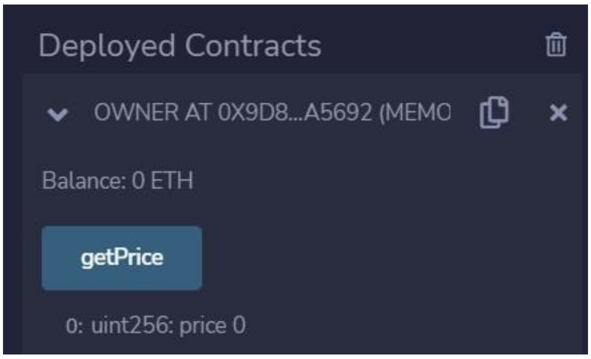
"Rutuja","47"

```
0: string: Rutuja
1: uint256: 47
```

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

```
pragma solidity ^0.5.0;

contract Owner {   address owner;
constructor() public {   owner = msg.sender;

 }

 modifier onlyOwner {

 require(msg.sender == owner);

 }

 modifier costs(uint price) {

 if (msg.value >= price) {

 }

 }
 function getPrice() public view returns(uint price){   return price;
 }
}
contract Register is Owner {
 mapping (address => bool) registeredAddresses;   uint price;
 constructor(uint initialPrice) public { price = initialPrice; }
 function register() public payable costs(price) {
registeredAddresses[msg.sender] = true;
 }
 function changePrice(uint _price) public onlyOwner {   price = _price;
 }
}
```

Output:

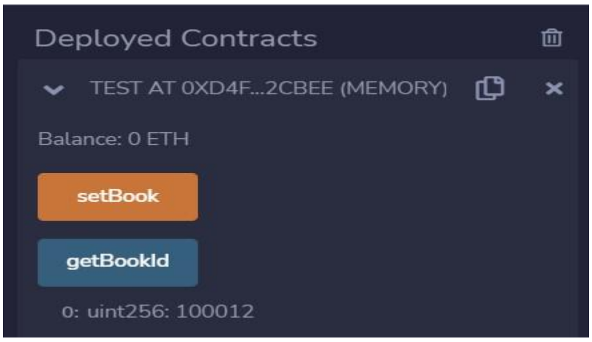


```
0: uint256: price 0
```

5.Write a solidity smart contract to demonstrate use of structure.

```
pragma solidity ^0.5.0;  contract test {
struct Book {   string title;   string author;
 uint book_id;  }
 Book book;
 function setBook() public {
 book = Book('Learn Java', 'TP', 100012);  }
 function getBookId() public view returns (uint) {   return book.book_id;  }  }
```

Output:

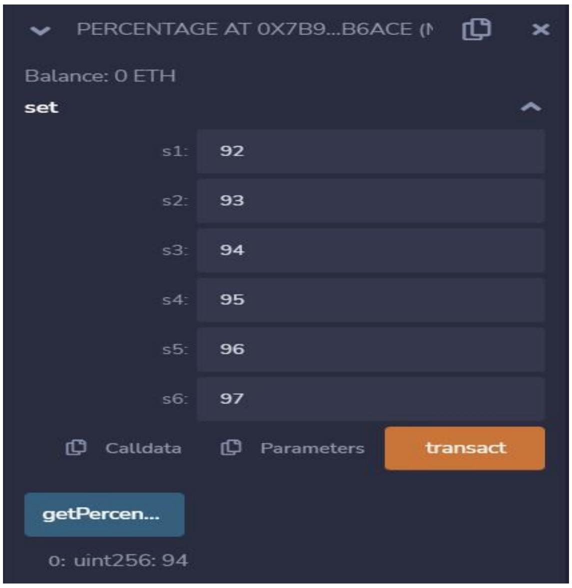


```
0: uint256: 100012
```

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

```
pragma solidity ^0.5.0;  contract percentage{

 uint sub_1;uint sub_2; uint sub_3;uint sub_4;uint sub_5;uint sub_6;uint total=600;   uint marksObtained;

 function set(uint s1,uint s2 ,uint s3,uint s4,uint s5,uint s6) public {   sub_1=s1;
sub_2=s2;   sub_3=s3;   sub_4=s4;   sub_5=s5;   sub_6=s6;

 marksObtained=sub_1+sub_2+sub_3+sub_4+sub_5+sub_6;   marksObtained=marksObtained*100;

 }

 function getPercentage() public view returns (uint) {

 uint percent=marksObtained/total;

 return percent;

 }

}
```

Output:

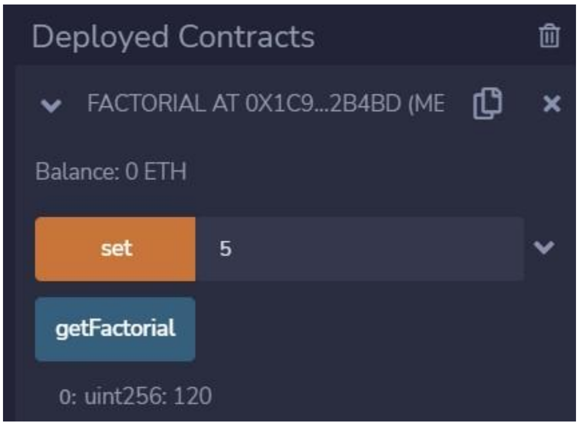


```
s1: 92
s2: 93
s3: 94
s4: 95
s5: 96
s6: 97

0: uint256: 94
```

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

```
pragma solidity ^0.5.0;  contract factorial{   uint
number;   function set(uint n) public {

 number=n;

 }

 function getFactorial() public view returns (uint) {   uint f=1;

 for(uint i=2;i<=number;i++){   f=f*i;

 }

 return f;

 }

}
```

Output:



FACTORIAL AT 0X1C9...2B4BD (ME

Balance: 0 ETH

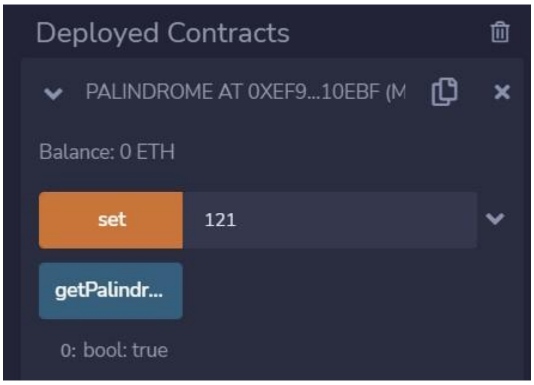
set 5

```
0: uint256: 120
```

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

```
pragma solidity ^0.5.0;  contract palindrome{
uint number;

 function set(uint n) public {

 number=n;

 }

 function getPalindrome() public view returns (bool ) {   uint r;

 uint n=number;   uint reverseNumber=0;

 while(n>0){   r=n%10;

 reverseNumber=reverseNumber*10+r;   n=n/10;

 }

 if(reverseNumber==number){

 return true;

 }

 else

 return false;

 }

}
```

Output:

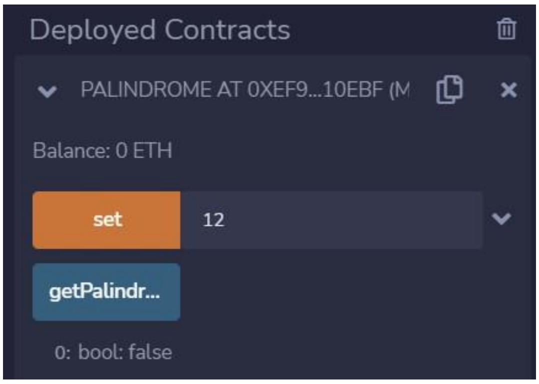


PALINDROME AT 0XEF9...10EBF (M

Balance: 0 ETH

set 121

```
0: bool: true
```



PALINDROME AT 0XEF9...10EBF (M

Balance: 0 ETH

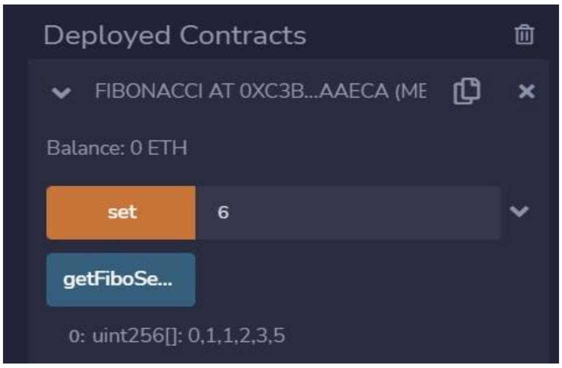
set 12

```
0: bool: false
```

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

```
pragma solidity ^0.5.0;  contract fibonacci{   uint
number_of_terms;   function set (uint n) public {

 number_of_terms=n;

 }

 function getFiboSeries() public view returns (uint[] memory ) {   uint a=0;   uint
b=1;   uint c;

 uint[] memory result=new uint[](number_of_terms);   result[0]=a;
result[1]=b;

 for(uint i=2;i<number_of_terms;i++){   c=a+b;   result[i]=c;
a=b;

 b=c;

 }

 return result;

 }

}
```

Output:

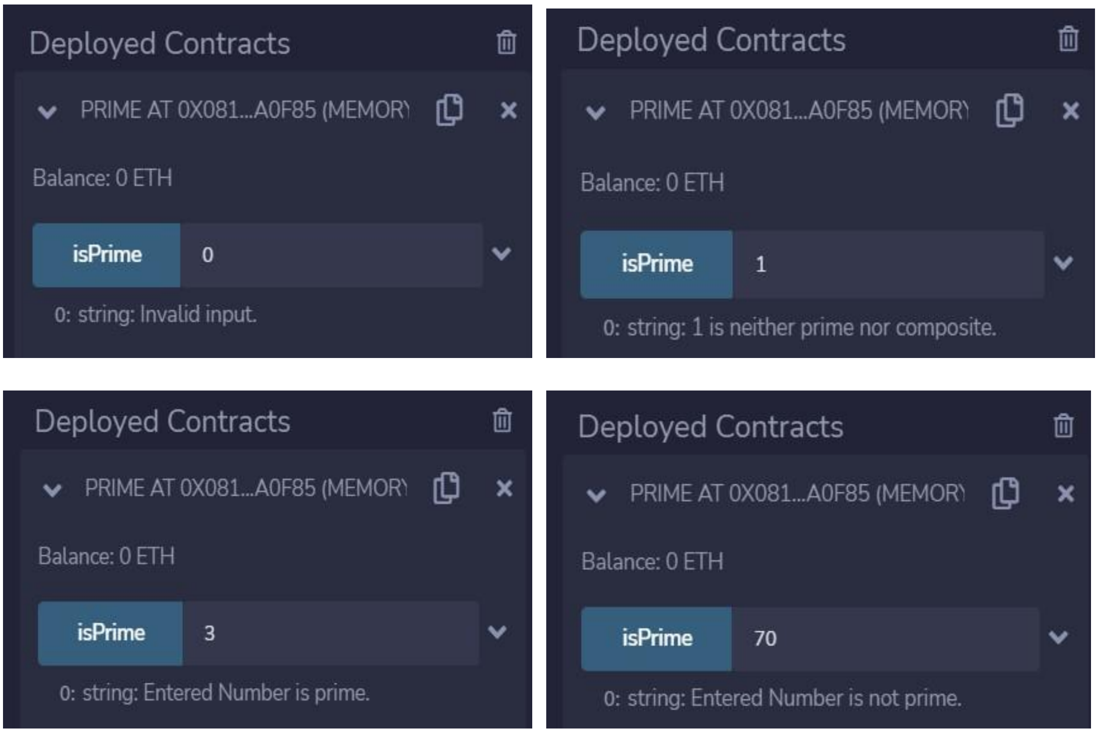


```
0: uint256[]: 0,1,1,2,3,5
```

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

```
pragma solidity ^0.5.0;  contract prime{

 function isPrime(uint n) public view returns (string memory ) {   string
memory message="";   if(n==0){

 return "Invalid input.";

 }

 else if (n==1){

 return "1 is neither prime nor composite.";

 }

 else if(n==2){

 return "Entered Number is prime.";

 }
 else{   bool flag=true;   for(uint
i=2;i<=n/2;i++ ){   if(n%i==0){
flag=false;

 break;

 }

 }

 if(flag){

 return "Entered Number is prime.";

 }

 else{

 return "Entered Number is not prime.";

 }

 }

 } }
```

Output:



```
Deployed Contracts
PRIME AT 0X081...A0F85 (MEMORY
Balance: 0 ETH
isPrime   0
0: string: Invalid input.
```

```
Deployed Contracts
PRIME AT 0X081...A0F85 (MEMORY
Balance: 0 ETH
isPrime   1
0: string: 1 is neither prime nor composite.
```

```
Deployed Contracts
PRIME AT 0X081...A0F85 (MEMORY
Balance: 0 ETH
isPrime   3
0: string: Entered Number is prime.
```

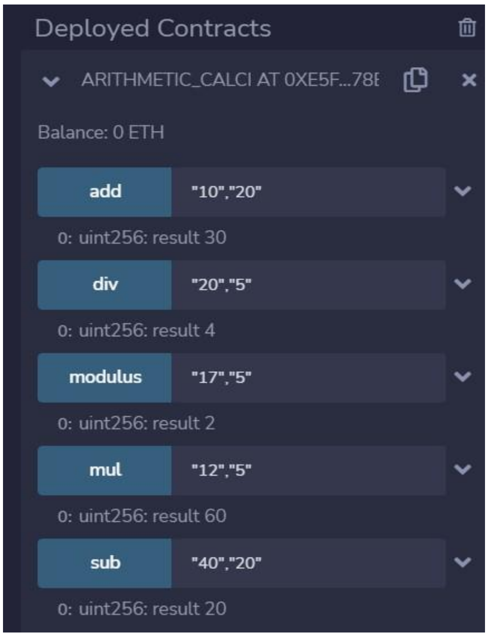
```
Deployed Contracts
PRIME AT 0X081...A0F85 (MEMORY
Balance: 0 ETH
isPrime   70
0: string: Entered Number is not prime.
```

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

```
pragma solidity ^0.5.0;  contract
arithmetic_calci{

 function add(uint n1,uint n2) public view returns (uint result ) {   return
n1+n2;

 }

 function sub(uint n1,uint n2) public view returns (uint result ) {   return n1-
n2;
```

```
 }

 function mul(uint n1,uint n2) public view returns (uint result ) {   return
n1*n2;

 }

 function div(uint n1,uint n2) public view returns (uint result ) {   return
n1/n2;

 }

 function modulus(uint n1,uint n2) public view returns (uint result ) {   return
n1%n2;

 }
```

Output:

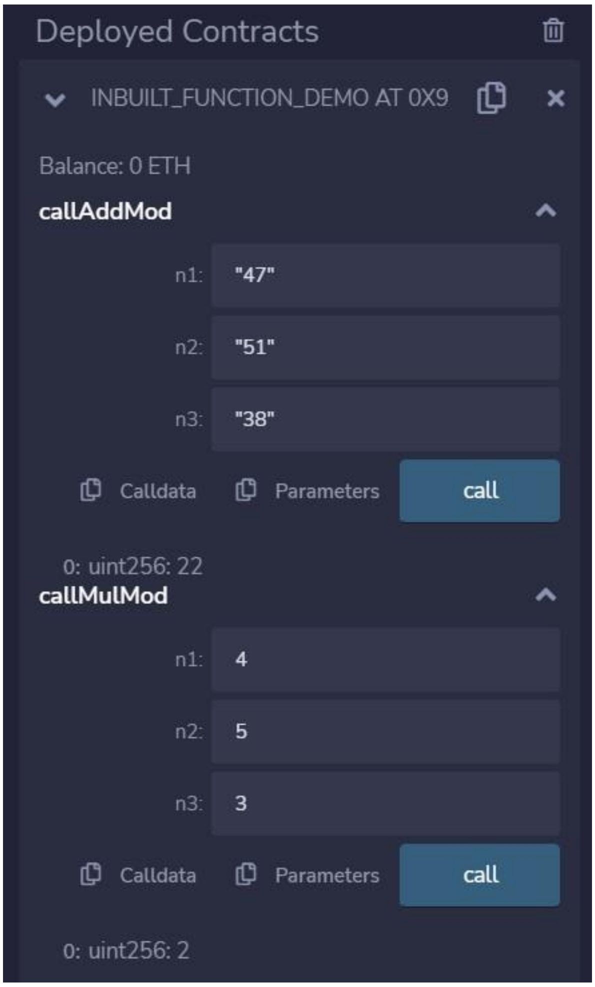


```
Balance: 0 ETH
add      "10","20"
0: uint256: result 30
div      "20","5"
0: uint256: result 4
modulus  "17","5"
0: uint256: result 2
mul      "12","5"
0: uint256: result 60
sub      "40","20"
0: uint256: result 20
```

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

```
pragma solidity ^0.5.0;  contract
inbuilt_function_demo{

 function callAddMod(uint n1,uint n2,uint n3) public pure returns(uint){   return
addmod(n1,n2,n3);

 }

 function callMulMod(uint n1,uint n2,uint n3) public pure returns(uint){   return
mulmod(n1,n2,n3);

 }

}
```

Output:



INBUILT_FUNCTION_DEMO AT 0X9

Balance: 0 ETH

callAddMod

n1: "47"

n2: "51"

n3: "38"

0: uint256: 22

callMulMod

n1: 4

n2: 5

n3: 3

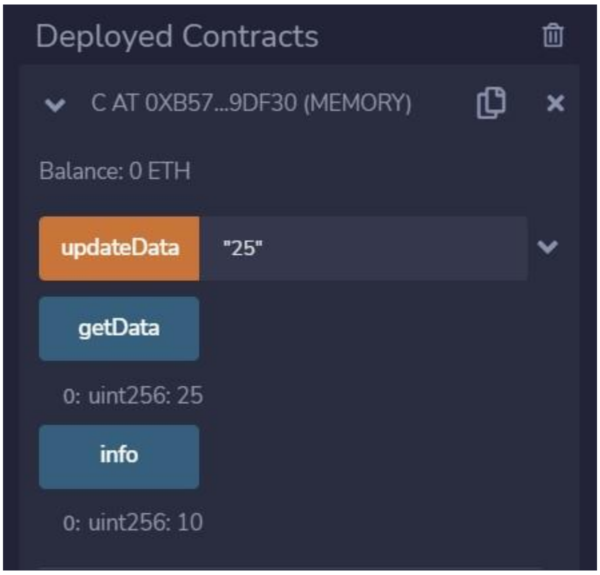
0: uint256: 2

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

```
pragma solidity ^0.5.0;  contract C {

 //private state variable

 uint private data;


 //public state variable   uint public
info;   //constructor   constructor()
public {

 info = 10;

 }

 //private function

 function increment(uint a) private pure returns(uint) { return a + 1; }


 //public function

 function updateData(uint a) public { data = a; }  function getData()
public view returns(uint) { return data; }

 function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}
```

```
//Derived Contract  contract
E is C {   uint private result;
C private c;   constructor()
public {

 c = new C();

 }

 function getComputedResult() public {   result =
compute(3, 5);

 }

 function getResult() public view returns(uint) { return result; }

 function getData() public view returns(uint) { return c.info(); }

}
```

Output:

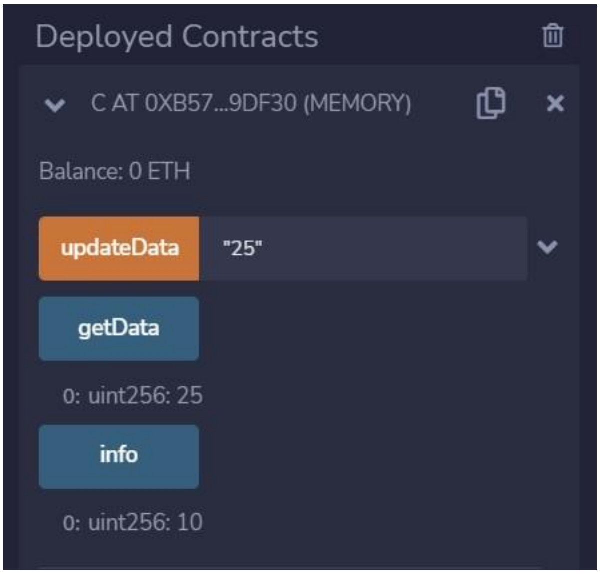


```
Balance: 0 ETH
0: uint256: 25
0: uint256: 10
```

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

```
pragma solidity ^0.5.0;  contract C {

 //private state variable

 uint private data;


 //public state variable   uint public
info;   //constructor   constructor()
public {

 info = 10;

 }

 //private function

 function increment(uint a) private pure returns(uint) { return a + 1; }


 //public function
 function updateData(uint a) public { data = a; }  function getData()
public view returns(uint) { return data; }

 function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}

//Derived Contract  contract
E is C {   uint private result;
C private c;   constructor()
public {

 c = new C();

 }

 function getComputedResult() public {   result =
compute(3, 5);

 }

 function getResult() public view returns(uint) { return result; }

 function getData() public view returns(uint) { return c.info(); }

}
```

Output:



Balance: 0 ETH

"25"

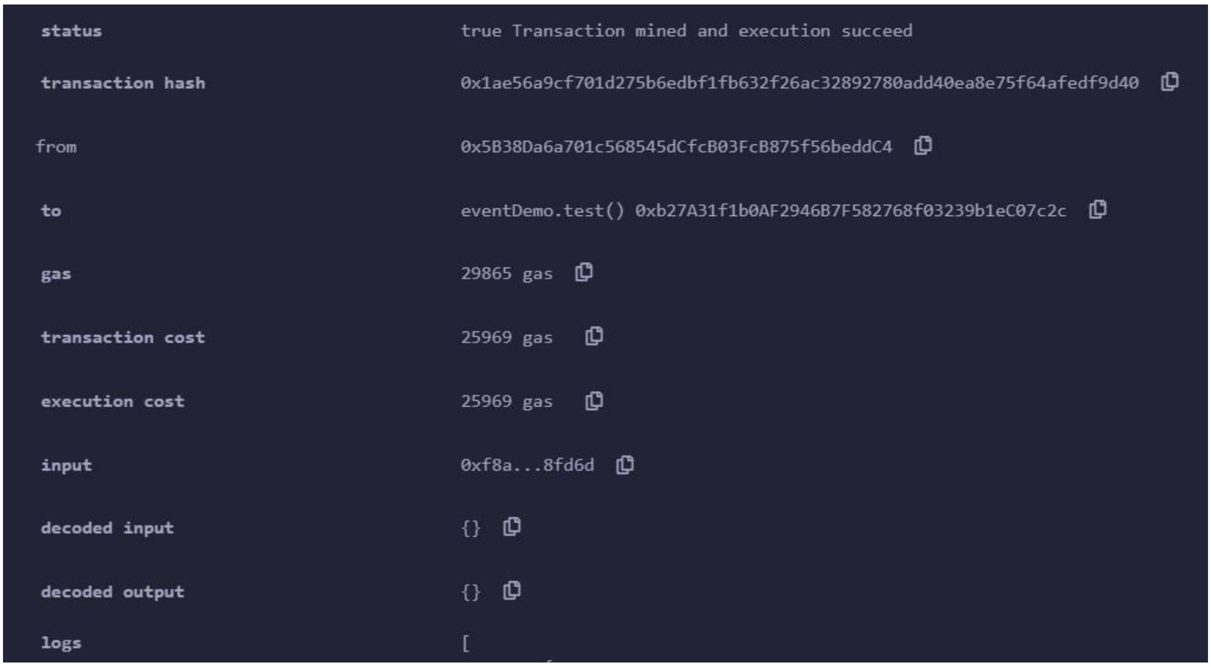
0: uint256: 25

0: uint256: 10

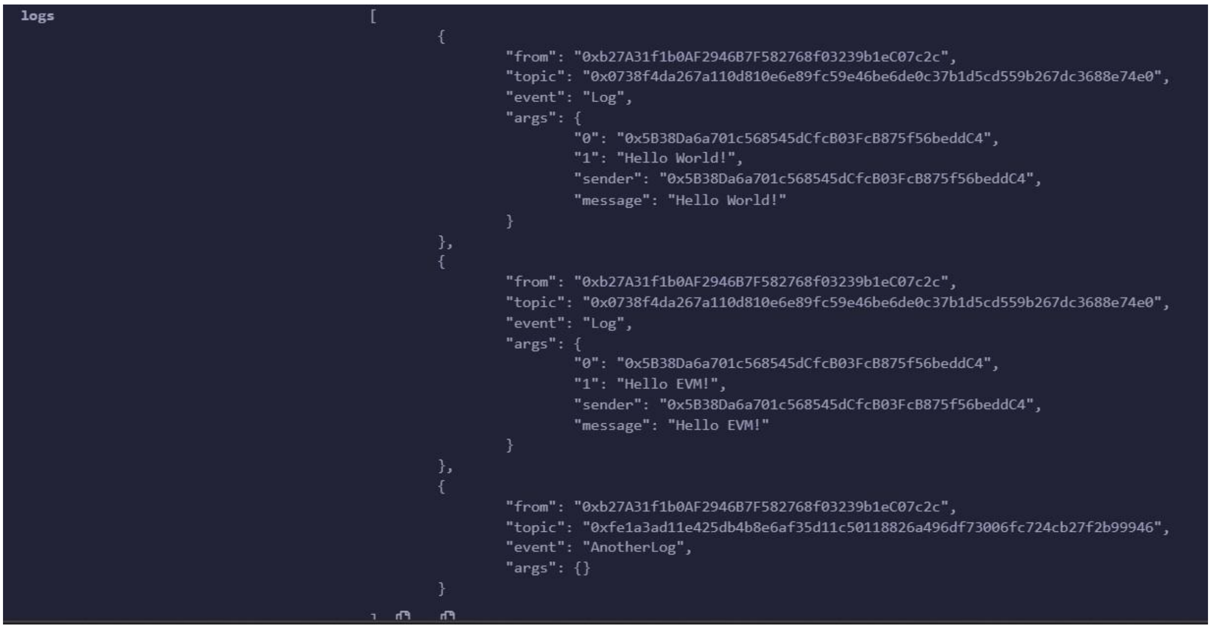
15.Write a solidity smart contract to demonstrate events.

```
pragma solidity ^0.5.0;  contract
eventDemo{


 event Log(address indexed sender, string message);   event
AnotherLog();   function test() public {   emit
Log(msg.sender, "Hello World!");   emit Log(msg.sender,
"Hello EVM!");   emit AnotherLog();

 }

}
```

Output:



```
status              true Transaction mined and execution succeed
transaction hash    0x1ae56a9cf701d275b6edbf1fb632f26ac32892780add40ea8e75f64afedf9d40
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  eventDemo.test() 0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c
gas                 29865 gas
transaction cost    25969 gas
execution cost      25969 gas
input               0xf8a...8fd6d
decoded input       {}
decoded output      {}
logs                [
```

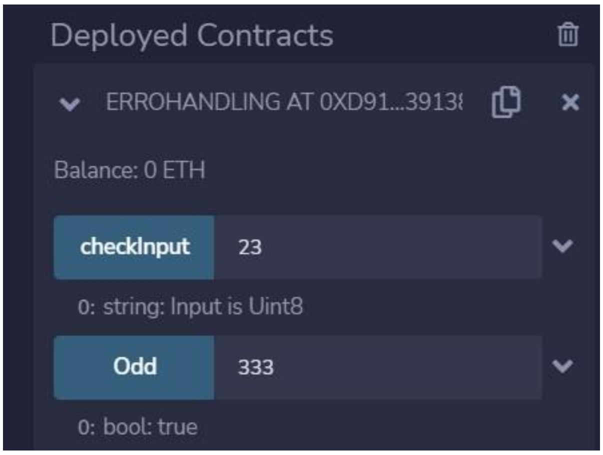


```
logs        [
                {
                        "from": "0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c",
                        "topic": "0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0",
                        "event": "Log",
                        "args": {
                                "0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                "1": "Hello World!",
                                "sender": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                "message": "Hello World!"
                        }
                },
                {
                        "from": "0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c",
                        "topic": "0x0738f4da267a110d810e6e89fc59e46be6de0c37b1d5cd559b267dc3688e74e0",
                        "event": "Log",
                        "args": {
                                "0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                "1": "Hello EVM!",
                                "sender": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                "message": "Hello EVM!"
                        }
                },
                {
                        "from": "0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c",
                        "topic": "0xfe1a3ad11e425db4b8e6af35d11c50118826a496df73006fc724cb27f2b99946",
                        "event": "AnotherLog",
                        "args": {}
                }
            ]
```

16.Write a solidity smart contract to demonstrate error handling.

```
pragma solidity 0.5.0;

contract ErroHandling {

function checkInput(uint _input) public view returns(string memory)

{

require(_input >= 0, "invalid uint8");  require(_input
<= 255, "invalid uint8");  return "Input is Uint8";

}

function Odd(uint _input) public view returns(bool)

{

require(_input % 2 != 0);  return true;  }
}
```

Output:



Deployed Contracts

ERROHANDLING AT 0XD91...3913

Balance: 0 ETH

checkInput 23

0: string: Input is Uint8

Odd 333

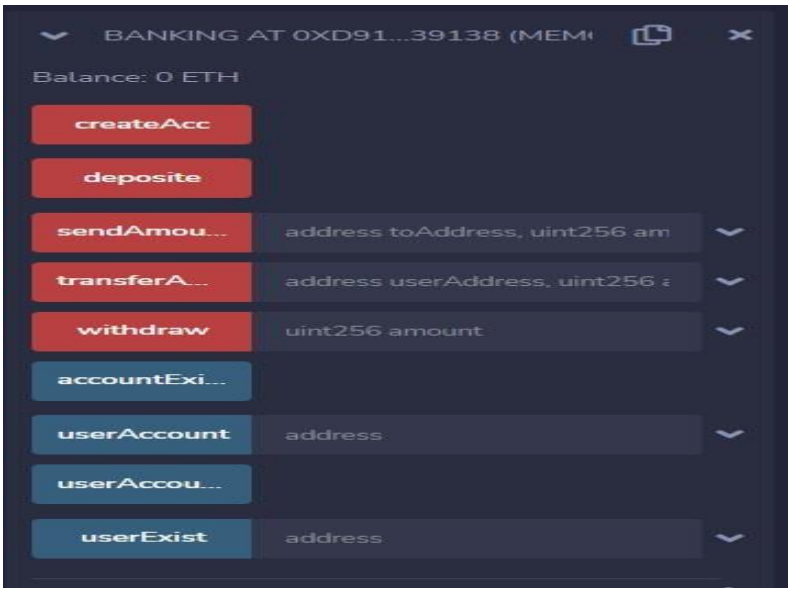
0: bool: true

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

```
pragma solidity ^0.5.0; contract
Banking{

  mapping(address=>uint)public userAccount;

  mapping(address=>bool)public userExist;


  function createAcc() public payable returns(string memory){
require(userExist[msg.sender]==false,'Account Alread Created');
if(msg.value==0){

      userAccount[msg.sender]=0;

    }

    userAccount[msg.sender]=msg.value;     userExist[msg.sender]=true;

    return 'account Created';

  }

  function deposite()public payable returns(string memory){
require(userExist[msg.sender]==true,'Account does not exist!');
require(msg.value>0,'value for deposite is zero');

    userAccount[msg.sender]=userAccount[msg.sender]+msg.value;
return 'Amount deposited successfully!';

  }


 function   withdraw(uint   amount)public   payable   returns(string   memory){
require(userExist[msg.sender]==true,'Account       does       not       exist!');
require(msg.value>0,'deposite value should greater than zero');
    require(msg.value>= amount,'Amount shound be equal to or greater then balance');
userAccount[msg.sender]=userAccount[msg.sender]-amount;     return 'Amount withdraw
successfully!';


   }
```

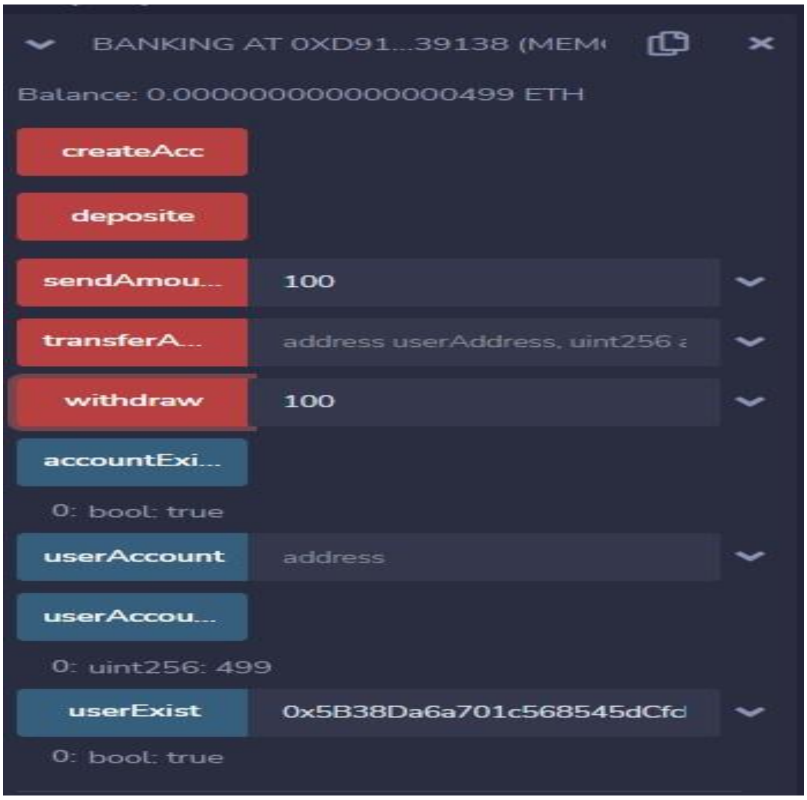
```
    function transferAmount(address payable userAddress,uint amount)public payable
returns(string memory){        require(userAccount [msg.sender]>amount,'insufficent balance
in bank');        require(userExist[msg.sender]==true,'Account does not created');
require(userExist[userAddress]==true,'transfer Amount does not efficient');
require(amount>0,'Enter non zero value for sending');


       userAccount[msg.sender]=userAccount[msg.sender]-amount;

       userAccount[userAddress]=userAccount[userAddress]+amount;
return 'transfer successfully';

    }


   function sendAmount(address payable toAddress , uint256 amount)public payable
returns(string memory){         require(amount>0,'Enter non zero value for withdrawal');
require(userExist[msg.sender]==true,'Account does not created');        require(userAccount
[msg.sender]>amount,'insufficent balance in bank');


        userAccount[msg.sender]=userAccount[msg.sender]-amount;
toAddress.transfer(amount);

        return'transfer successfully';

   }

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

}

   function accountExist()public view returns(bool) {        return
userExist[msg.sender];

}

}
```

Output:

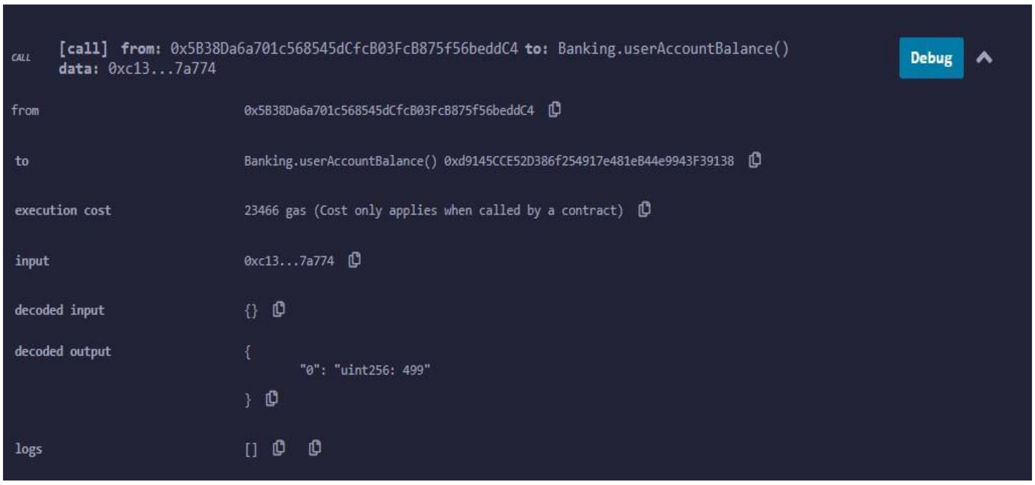


BANKING AT 0XD91...39138 (MEM

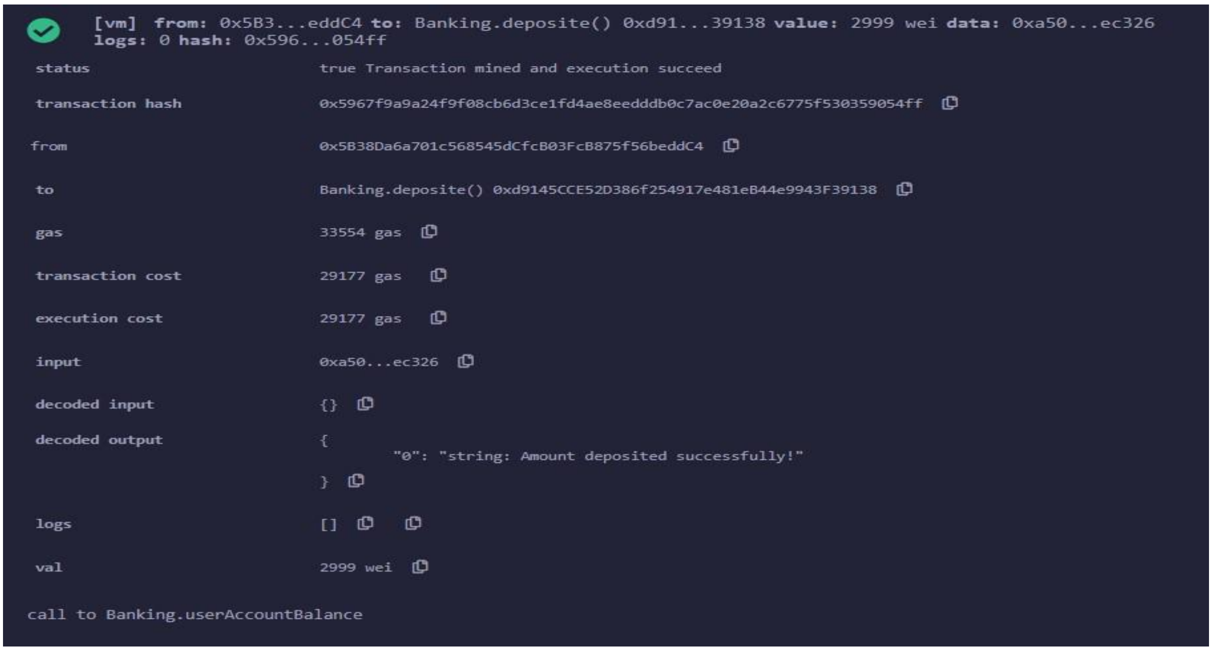
Balance: 0 ETH



```
BANKING AT 0XD91...39138 (MEM(
Balance: 0.000000000000000499 ETH
createAcc
deposite
sendAmou...      100
transferA...     address userAddress, uint256 a
withdraw         100
accountExi...
  0: bool: true
userAccount      address
userAccou...
  0: uint256: 499
userExist        0x5B38Da6a701c568545dCfd
  0: bool: true
```



```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Banking.userAccountBalance()
       data: 0xc13...7a774                                                          Debug

from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                   Banking.userAccountBalance() 0xd9145CCE52D386f254917e481eB44e9943F39138

execution cost       23466 gas (Cost only applies when called by a contract)

input                0xc13...7a774

decoded input        {}

decoded output       {
                         "0": "uint256: 499"
                     }

logs                 []
```



```
[vm] from: 0x5B3...eddC4 to: Banking.deposite() 0xd91...39138 value: 2999 wei data: 0xa50...ec326
logs: 0 hash: 0x596...054ff
status               true Transaction mined and execution succeed
transaction hash     0x5967f9a9a24f9f08cb6d3ce1fd4ae8eedddb0c7ac0e20a2c6775f530359054ff
from                 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                   Banking.deposite() 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                  33554 gas
transaction cost     29177 gas
execution cost       29177 gas
input                0xa50...ec326
decoded input        {}
decoded output       {
                         "0": "string: Amount deposited successfully!"
                     }
logs                 []
val                  2999 wei
call to Banking.userAccountBalance
```