Practical No. 03

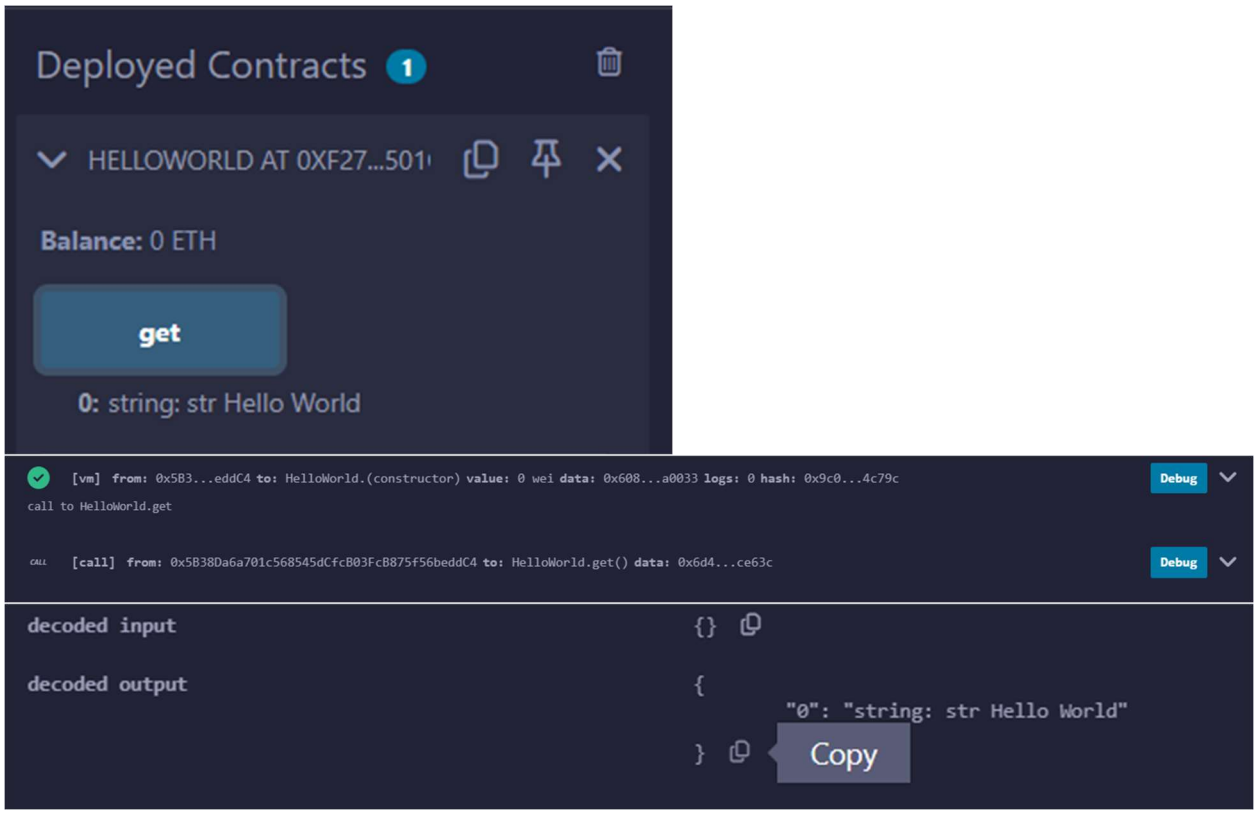
Solidity Programming

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

Code :

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


contract HelloWorld {
   function get()public pure returns (string memory str) {
      str = "Hello World";
   }
}
```

Output :

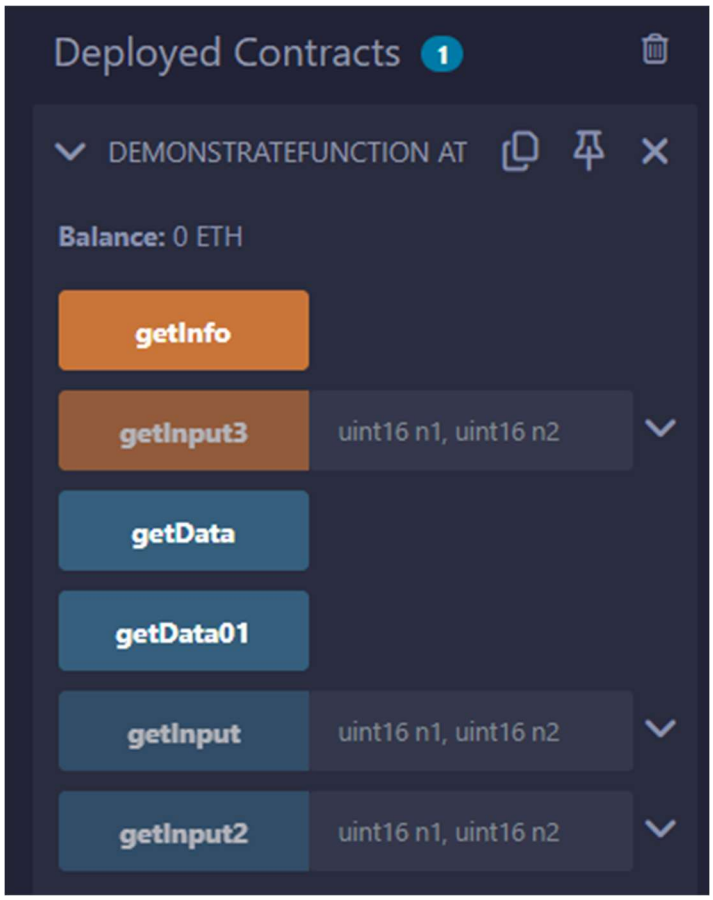


Q. 2 Write a solidity smart contract to demonstrate state variable, local variable and global variable.

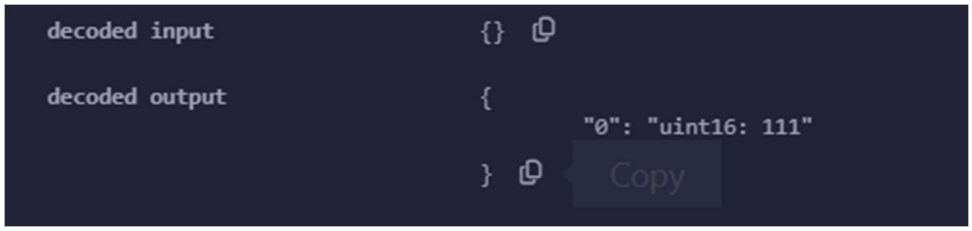
Code:

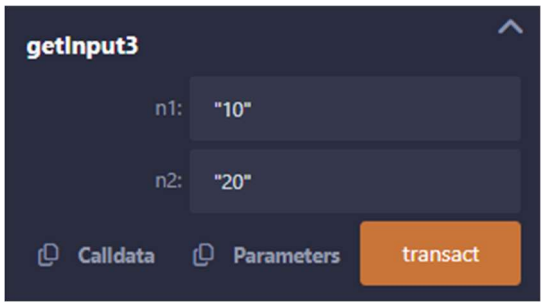
```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.8.27;
contract DemonstrateFunction{
   uint16 num = 10; // state variable
   function getInput3(uint16 n1, uint16 n2) public returns (uint16){
      num = 100;
      return n1 + n2 + num;
   }
   function getInput2(uint16 n1, uint16 n2) public view returns (uint16){
      return n1 + n2 + num;
   }
   function getInput(uint16 n1, uint16 n2) public pure returns (uint16){
      return n1 + n2;
   }
   function getInfo() public returns (uint16){
      return num += 11;
   }
   function getData() public view returns (uint16){
      return num;
   }
   function getData01() public pure returns (uint16){
      return 111;
   }
}
```

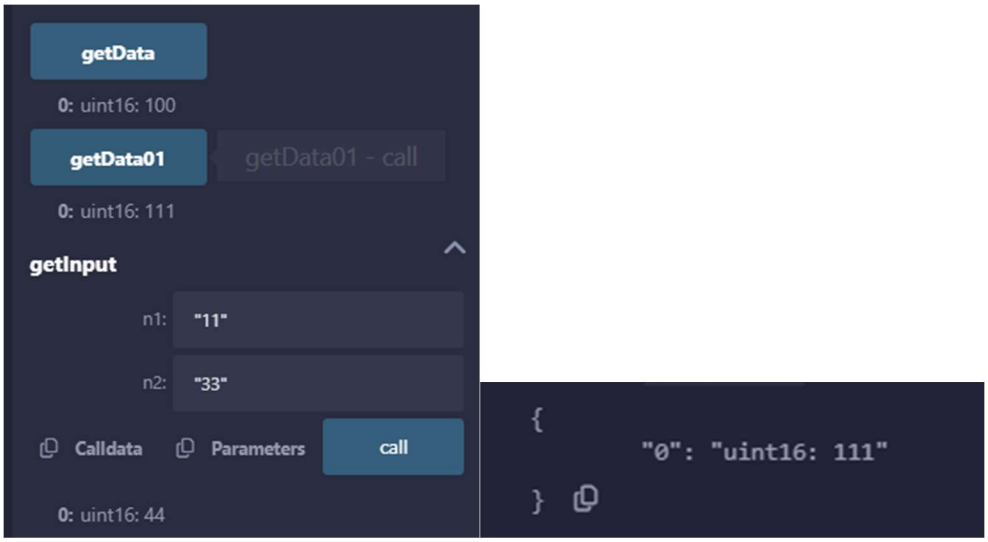
Output:


Deployed Contracts 1
DEMONSTRATEFUNCTION AT
Balance: 0 ETH
getInfo
getInput3
uint16 n1, uint16 n2
getData
getData01
getInput
uint16 n1, uint16 n2
getInput2
uint16 n1, uint16 n2

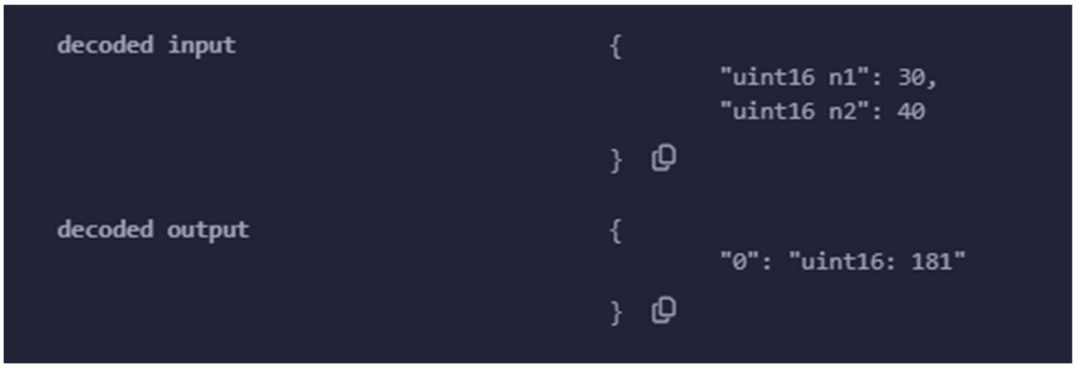

Balance: 0 ETH
getInfo


decoded input
{}
decoded output
{
"0": "uint16: 111"
}
Copy


getInput3
n1:
"10"
n2:
"20"
Calldata
Parameters
transact


getData
0: uint16: 100
getData01
getData01 - call
0: uint16: 111
getInput
n1: "11"
n2: "33"
Calldata
Parameters
call
0: uint16: 44
{
"0": "uint16: 111"
}


{
"uint16 n1": 11,
"uint16 n2": 33
}
{
"0": "uint16: 44"
}
getInput2
n1: "30"
n2: "40"
Calldata
Parameters
call
0: uint16: 181

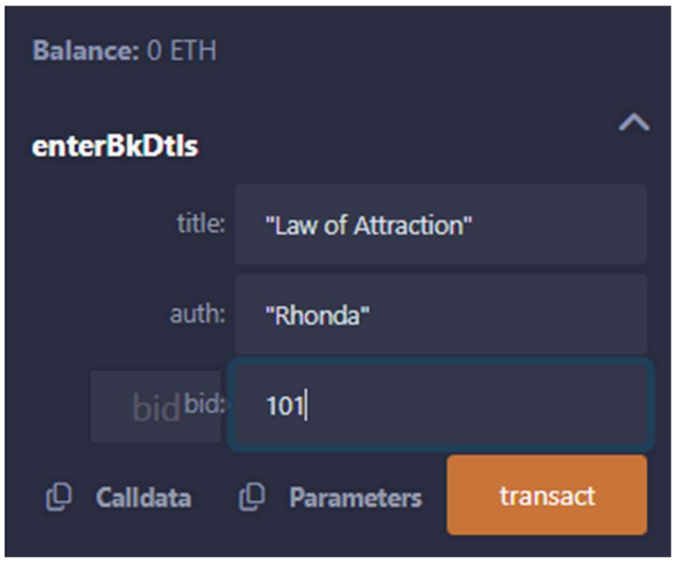

decoded input
{
"uint16 n1": 30,
"uint16 n2": 40
}
decoded output
{
"0": "uint16: 181"
}

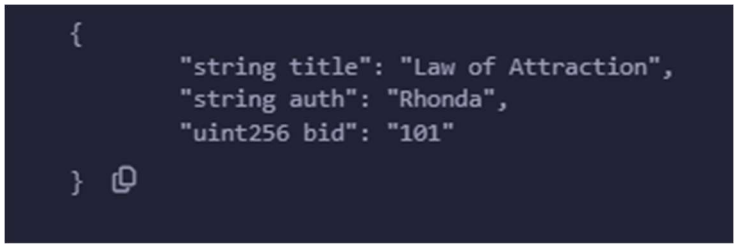
Q.3 Write a solidity smart contract to demonstrate getter and setter methods.

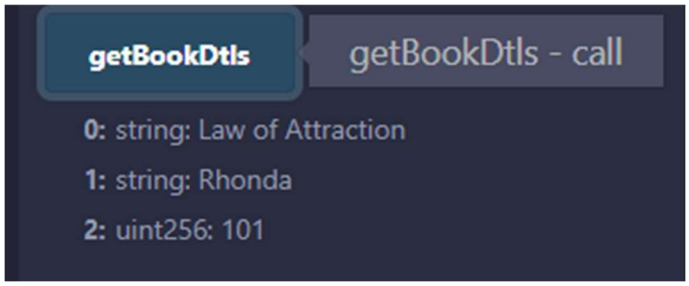
Code :

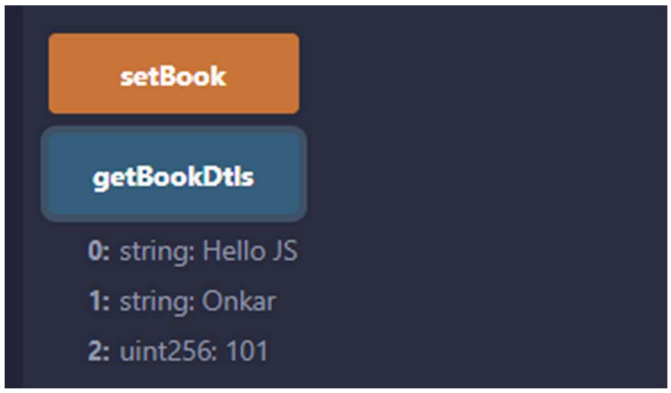
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract StructureDemo1{
   // structure like C program
   struct Book{
      string title;
      string author;
      uint book_id;
   }
   // variable object created
   Book b1;
   // setter method
   function setBook() public {
       b1=Book('Hello JS', 'Onkar', 101);
    }
   // add values to it
   function enterBkDtls(string memory title, string memory auth, uint bid) public {
      b1=Book(title, auth, bid);
   }
   // Getter method
   function getBookDtls() public view returns(string memory,string memory,uint) {
      return (b1.title,b1.author,b1.book_id);
   }
}
```

Output :


Balance: 0 ETH
enterBkDtls
title:
"Law of Attraction"
auth:
"Rhonda"
bid
bid:
101
Calldata
Parameters
transact


{
"string title": "Law of Attraction",
"string auth": "Rhonda",
"uint256 bid": "101"
}


getBookDtls
getBookDtls - call
0: string: Law of Attraction
1: string: Rhonda
2: uint256: 101

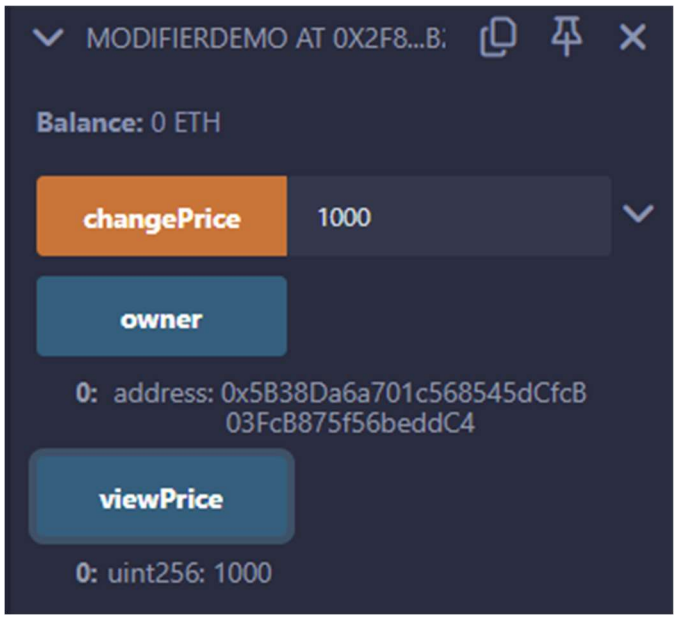

setBook
getBookDtls
0: string: Hello JS
1: string: Onkar
2: uint256: 101

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

Code:

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

pragma solidity >= 0.5.0 < 0.8.27;

contract ModifierDemo {

   // We will use the modifier to limit the function changePrice to only the owner of the
contract

   address public owner;

   uint price;

   constructor() {

      owner = msg.sender;

   }

   modifier onlyOwner{

      require(msg.sender == owner, 'Only Owner is allowed to modify the price!');

      _;   // Asterisk is used to indicate that this function will be executed even if there is an
exception.

      // This will allow us to do any other modification.

   }

   function changePrice(uint _price) public onlyOwner{

      price=_price;

   }

   function viewPrice() public view returns (uint){

      return price;

   }

}
```

Output:


MODIFIERDEMO AT 0X2F8...B:
Balance: 0 ETH
changePrice
1000
owner
0: address: 0x5B38Da6a701c568545dCfcB
03FcB875f56beddC4
viewPrice
0: uint256: 1000

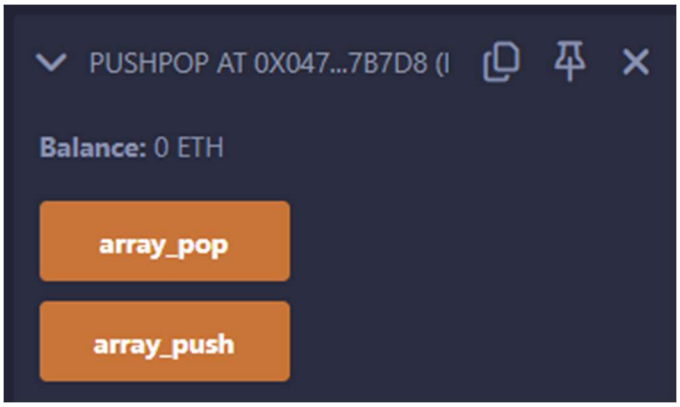

{
"0": "uint256: 1000"
}

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

Code:

```
// Push pop Array
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
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:



Push Button Click:


```
{
        "0": "uint256[]: 10,20,30,40,50,60,70,80"
}
```


Pop Button Click:

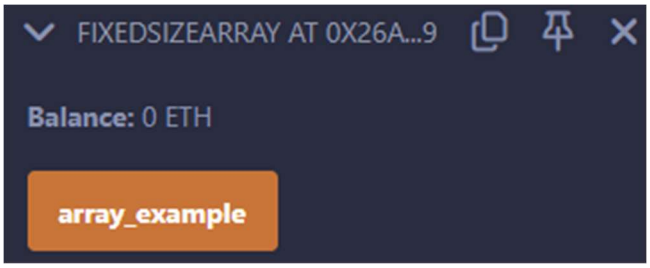

```
{
        "0": "uint256[]: 10,20,30,40,50,60,70"
}
```

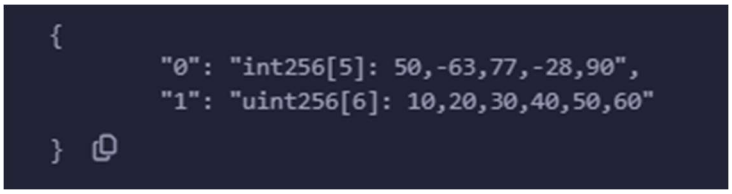
Q.6 Write a Solidity program to demonstrate creating a fixed-size array and access array element.

Code:

```
// Demonstrate Array
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.8.27;
// create a Contract
contract FixedSizeArray {
   // Declaring state variable of Array
   uint[6] data1;
   //Defining the functions
   function array_example() public returns (int[5] memory, uint[6] memory){
      int[5] memory data = [int(50), -63, 77, -28, 90];
      // local variable type
      data1 = [uint(10), 20, 30, 40, 50, 60];
      return (data, data1); // Returning the values
   }
}
```

Output:



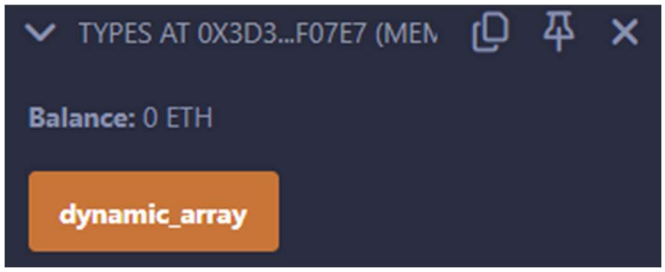


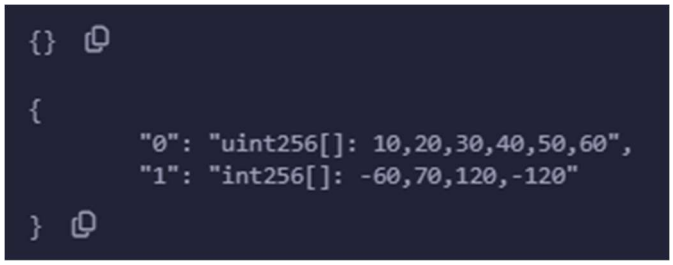
Q.7 Write a Solidity program to demonstrate creating a dynamic array and accessing array element.

Code:

```
// Dynamic Smart Contract
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.8.27;
contract Types{
   // dynamic array
   int[] data1;
   // static array
   uint[] data = [10, 20, 30, 40, 50, 60];
   function dynamic_array() public returns (uint[] memory, int[] memory) {
      data1 = [int(-60), 70, 120, -120];
      return (data, data1);
   }
}
```

Output:

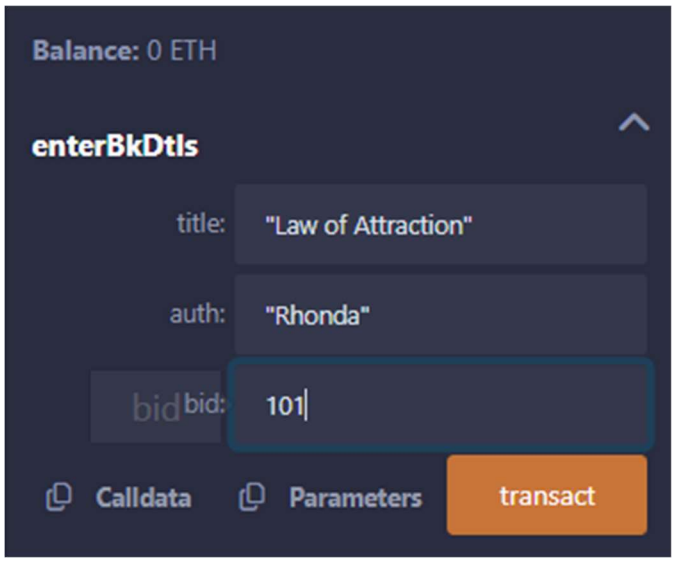


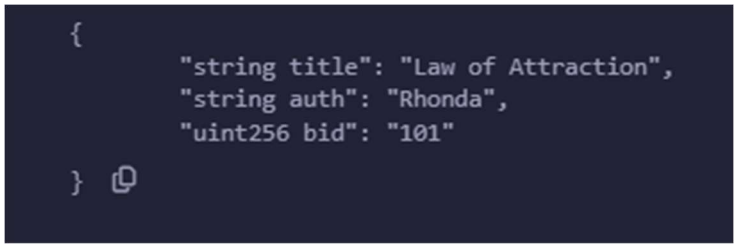


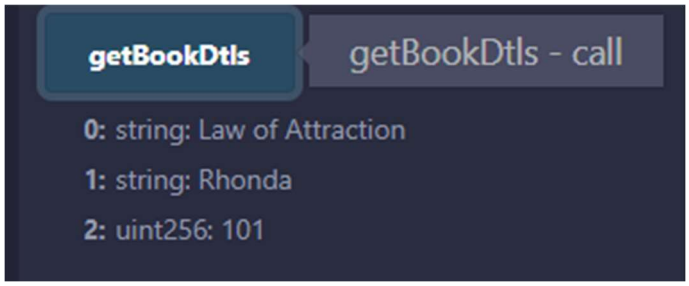
Q.8 Write a solidity smart contract to demonstrate use of structure.

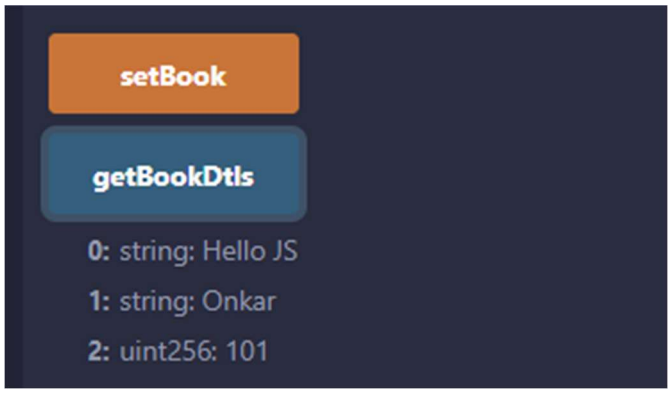
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract StructureDemo1{
   // structure like C program
   struct Book{
      string title;
      string author;
      uint book_id;
   }
   // variable object created
   Book b1;
   // setter method
   function setBook() public {
       b1=Book('Hello JS', 'Onkar', 101);
    }
   // add values to it
   function enterBkDtls(string memory title, string memory auth, uint bid) public {
      b1=Book(title, auth, bid);
   }
   // Getter method
   function getBookDtls() public view returns(string memory,string memory,uint) {
      return (b1.title,b1.author,b1.book_id);
   }
}
```

Output :


Balance: 0 ETH
enterBkDtls
title:
"Law of Attraction"
auth:
"Rhonda"
bid
bid:
101
Calldata
Parameters
transact


{
"string title": "Law of Attraction",
"string auth": "Rhonda",
"uint256 bid": "101"
}


getBookDtls
getBookDtls - call
0: string: Law of Attraction
1: string: Rhonda
2: uint256: 101

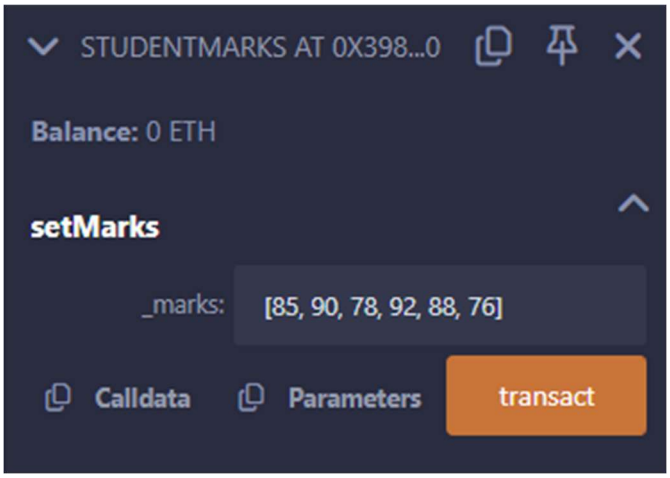

setBook
getBookDtls
0: string: Hello JS
1: string: Onkar
2: uint256: 101

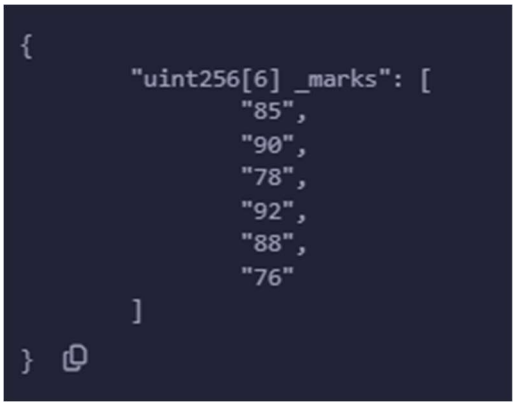
Q.9 Write a solidity smart contract to calculate percentage of marks obtained by students for six subjects in final examination.

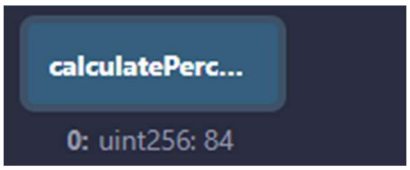
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract StudentMarks {
    struct Student {
        uint256[6] marks; // Array to hold marks for six subjects
    }
    mapping(address => Student) private students;
    // Function to set marks for a student
    function setMarks(uint256[6] memory _marks) public {
        require(_marks.length == 6, "Must provide marks for six subjects");
        students[msg.sender].marks = _marks;
    }
    // Function to calculate the percentage of marks obtained
    function calculatePercentage() public view returns (uint256) {
        Student storage student = students[msg.sender];
        uint256 totalMarks = 0;
        uint256 maxMarks = 600; // Assuming each subject is out of 100
        for (uint256 i = 0; i < 6; i++) {
            totalMarks += student.marks[i];
        }
        // Calculate percentage
        return (totalMarks * 100) / maxMarks;
    }
}
```

Output:


STUDENTMARKS AT 0X398...0
Balance: 0 ETH
setMarks
_marks:
[85, 90, 78, 92, 88, 76]
Calldata
Parameters
transact


{
"uint256[6] _marks": [
"85",
"90",
"78",
"92",
"88",
"76"
]
}


calculatePerc...
0: uint256: 84

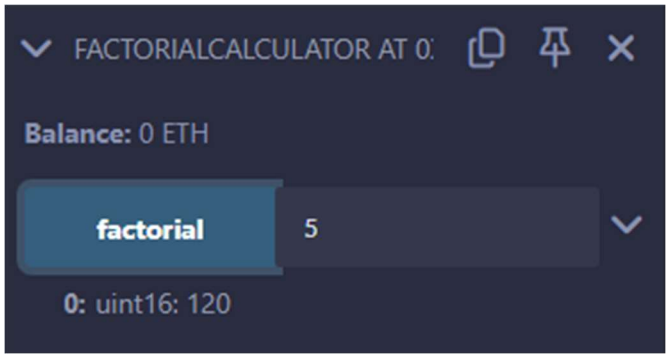

{
"0": "uint256: 84"
}

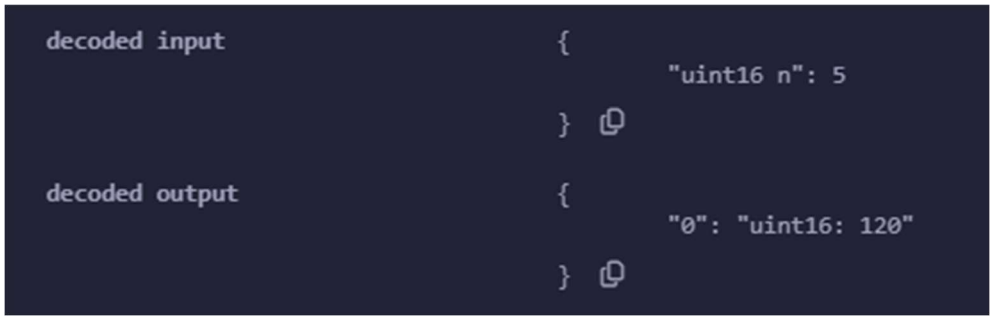
Q.10 Write a solidity smart contract to find the factorial of entered number.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract FactorialCalculator {
   // Function to calculate factorial of a number
   function factorial(uint16 n) public pure returns (uint16) {
      require(n >= 0, "Input must be a non-negative integer");
      if (n == 0) {
         return 1; // Base case: 0! = 1
      } else {
         return n * factorial(n - 1); // Recursive case
      }
   }
}
```

Output:



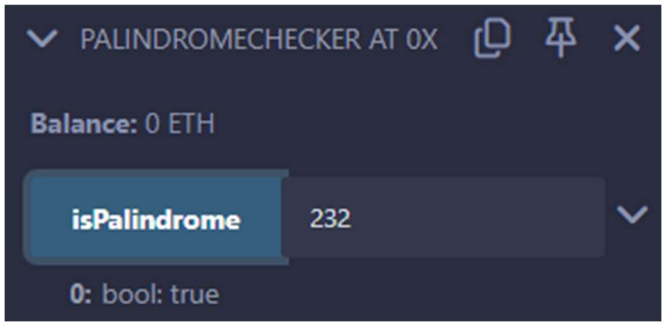


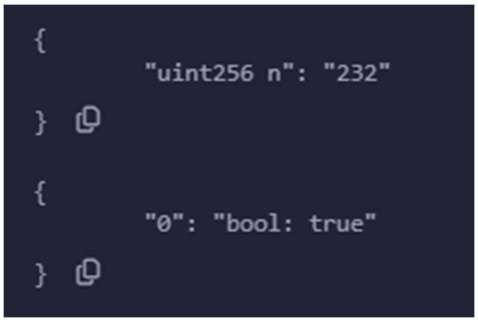
Q.11 Write a solidity smart contract to check whether entered number is palindrome or not.

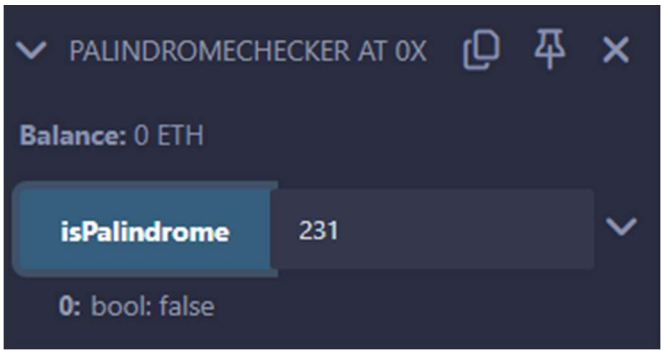
Code:

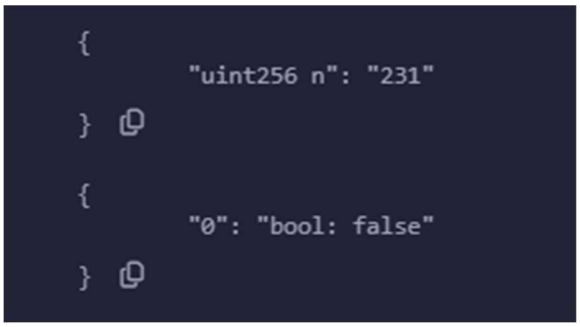
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract PalindromeChecker {
   // Function to check if a number is a palindrome
   function isPalindrome(uint256 n) public pure returns (bool) {
      uint256 reversed = reverse(n);
      return n == reversed;
   }
   // Function to reverse a number
   function reverse(uint256 n) internal pure returns (uint256) {
      uint256 reversed = 0;
      while (n > 0) {
         reversed = reversed * 10 + n % 10;
         n /= 10;
      }
      return reversed;
   }
}
```

Output:




{
"uint256 n": "232"
}
{
"0": "bool: true"
}


PALINDROMECHECKER AT 0X
Balance: 0 ETH
isPalindrome
231
0: bool: false

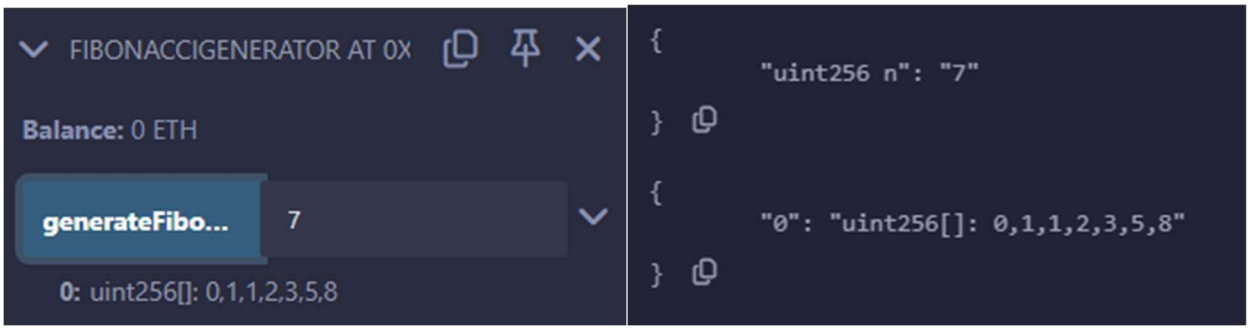

{
"uint256 n": "231"
}
{
"0": "bool: false"
}

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

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract FibonacciGenerator {
    // Function to generate Fibonacci series up to a given number
    function generateFibonacci(uint256 n) public pure returns (uint256[] memory) {
        require(n > 0, "Input must be a positive integer");
        uint256[] memory fibSequence = new uint256[](n);
        fibSequence[0] = 0;
        if (n == 1) {
            return fibSequence;
        }
        fibSequence[1] = 1;
        for (uint256 i = 2; i < n; i++) {
            fibSequence[i] = fibSequence[i - 1] + fibSequence[i - 2];
        }
        return fibSequence;
    }
}
```

Output:

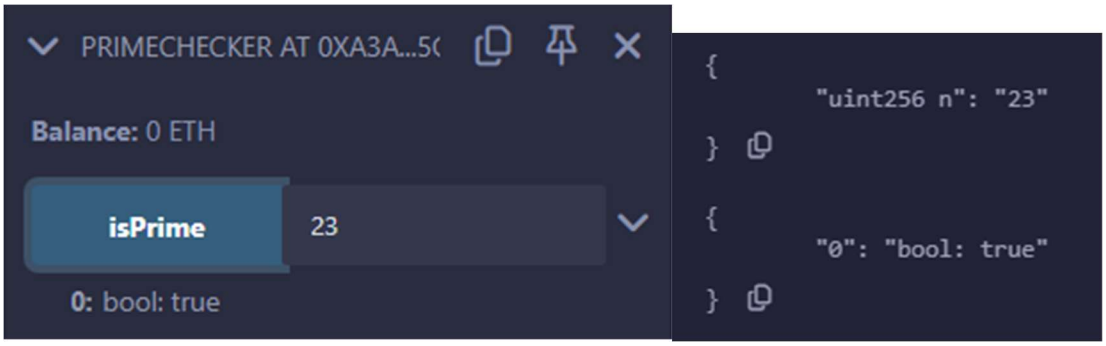


Q.13 Write a solidity smart contract to check whether entered number is prime number or not.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract PrimeChecker {
    // Function to check if a number is prime
    function isPrime(uint256 n) public pure returns (bool) {
        require(n > 0, "Input must be a positive integer");
        if (n == 1) {
            return false; // 1 is not a prime number
        }
        for (uint256 i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                return false; // n is divisible by i, so it's not prime
            }
        }
        return true; // n is prime
    }
}
```

Output:



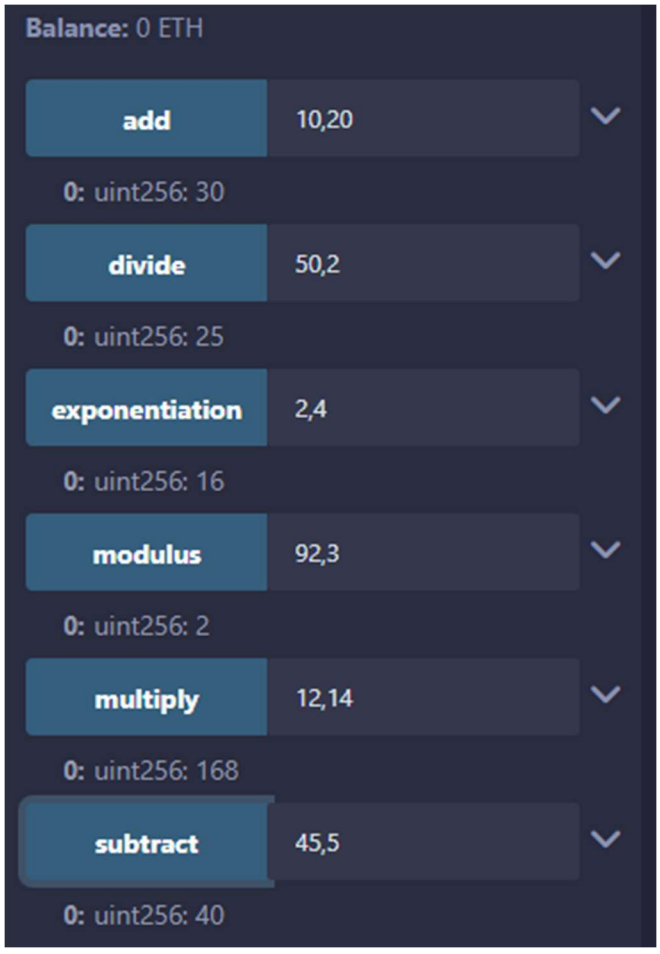
Q.14 Write a solidity smart contract to create arithmetic calculator which includes functions for operations addition, subtraction, multiplication, division etc.

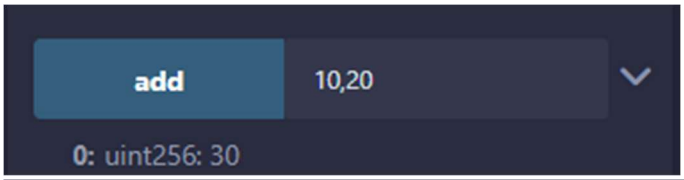
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract ArithmeticCalculator {
    // Function to add two numbers
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }
    // Function to subtract two numbers
    function subtract(uint256 a, uint256 b) public pure returns (uint256) {
        require(b <= a, "Subtraction result would be negative");
        return a - b;
    }
    // Function to multiply two numbers
    function multiply(uint256 a, uint256 b) public pure returns (uint256) {
        return a * b;
    }
    // Function to divide two numbers
    function divide(uint256 a, uint256 b) public pure returns (uint256) {
        require(b > 0, "Division by zero is not allowed");
        return a / b;
    }
    // Function to calculate modulus of two numbers
    function modulus(uint256 a, uint256 b) public pure returns (uint256) {
        require(b > 0, "Modulus by zero is not allowed");
```

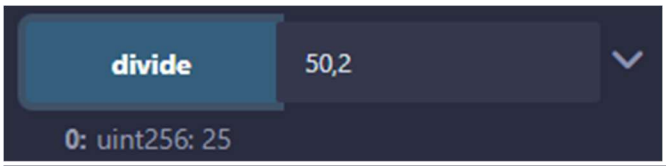
```
        return a % b;
    }
    // Function to calculate exponentiation of two numbers
    function exponentiation(uint256 base, uint256 exponent) public pure returns (uint256) {
        uint256 result = 1;
        for (uint256 i = 0; i < exponent; i++) {
            result *= base;
        }
        return result;
    }
}
```

Output:

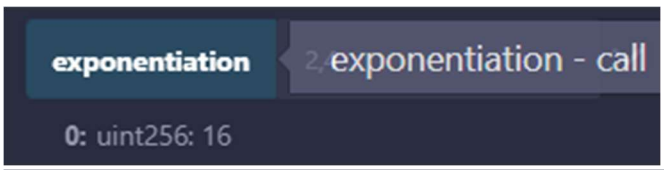




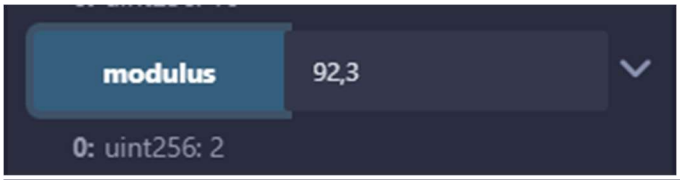

```
decoded input
{
    "uint256 a": "10",
    "uint256 b": "20"
}
decoded output
{
    "0": "uint256: 30"
}
```



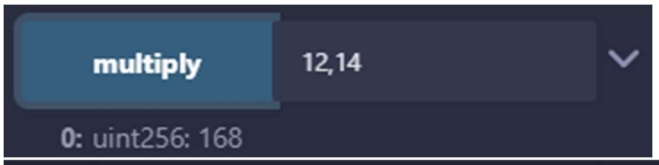

```
decoded input
{
    "uint256 a": "50",
    "uint256 b": "2"
}
decoded output
{
    "0": "uint256: 25"
}
```



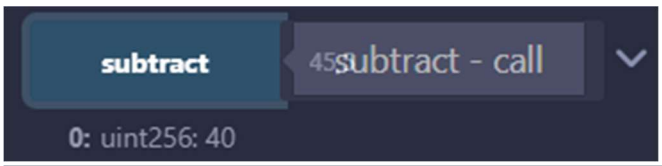

```
decoded input
{
    "uint256 base": "2",
    "uint256 exponent": "4"
}
decoded output
{
    "0": "uint256: 16"
}
```




```
decoded input
{
    "uint256 a": "92",
    "uint256 b": "3"
}

decoded output
{
    "0": "uint256: 2"
}
```




```
decoded input
{
    "uint256 a": "12",
    "uint256 b": "14"
}

decoded output
{
    "0": "uint256: 168"
}
```



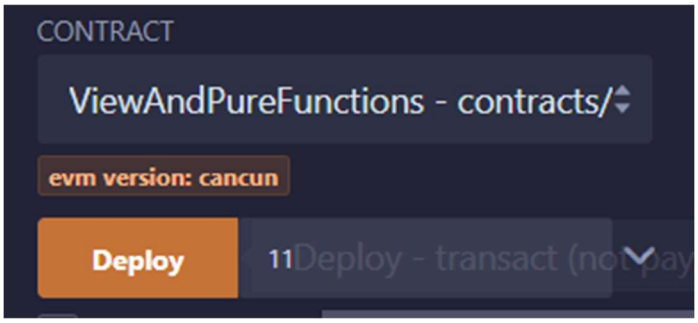

```
decoded input
{
    "uint256 a": "45",
    "uint256 b": "5"
}

decoded output
{
    "0": "uint256: 40"
}
```

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

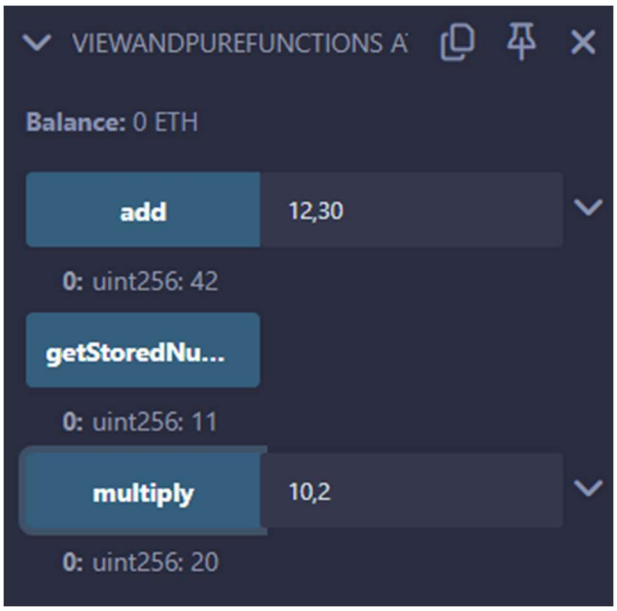
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract ViewAndPureFunctions {
    // State variable to store a number
    uint256 private storedNumber;

    // Constructor to initialize the stored number
    constructor(uint256 initialNumber) {
        storedNumber = initialNumber;
    }

    // View function to get the stored number
    function getStoredNumber() public view returns (uint256) {
        return storedNumber;
    }
    // Pure function to add two numbers
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }
    // Pure function to multiply two numbers
    function multiply(uint256 a, uint256 b) public pure returns (uint256) {
        return a * b;
    }
}
```

Output:

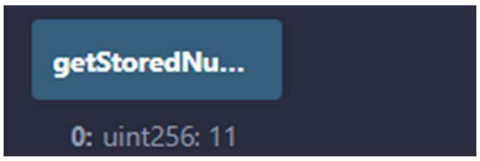

CONTRACT
ViewAndPureFunctions - contracts/
evm version: cancun
Deploy
11
Deploy - transact (not pay


decoded input
{
"uint256 initialNumber": "11"
}

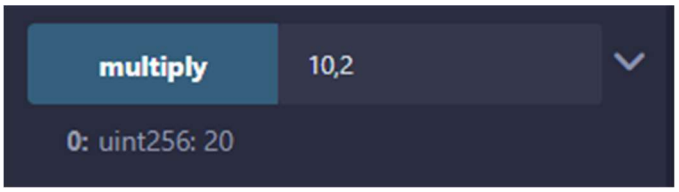

VIEWANDPUREFUNCTIONS A
Balance: 0 ETH
add
12,30
0: uint256: 42
getStoredNu...
0: uint256: 11
multiply
10,2
0: uint256: 20


VIEWANDPUREFUNCTIONS A
Balance: 0 ETH
add
12,30
0: uint256: 42

```
decoded input
{
        "uint256 a": "12",
        "uint256 b": "30"
}
decoded output
{
        "0": "uint256: 42"
}
```




```
decoded input
{}
decoded output
{
        "0": "uint256: 11"
}
```




```
decoded input
{
        "uint256 a": "10",
        "uint256 b": "2"
}
decoded output
{
        "0": "uint256: 20"
}
```

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

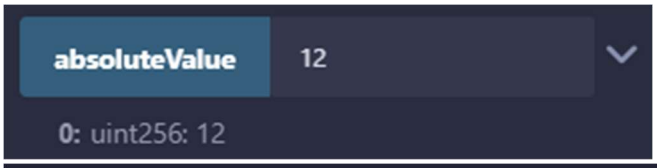
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract MathematicalFunctions {
   // Function to demonstrate addition
   function add(uint256 a, uint256 b) public pure returns (uint256) {
      return a + b;
   }
   // Function to demonstrate subtraction
   function subtract(uint256 a, uint256 b) public pure returns (uint256) {
      return a - b;
   }
   // Function to demonstrate multiplication
   function multiply(uint256 a, uint256 b) public pure returns (uint256) {
      return a * b;
   }
   // Function to demonstrate division
   function divide(uint256 a, uint256 b) public pure returns (uint256) {
      require(b > 0, "Division by zero is not allowed");
      return a / b;
   }
   // Function to demonstrate modulus
   function modulus(uint256 a, uint256 b) public pure returns (uint256) {
      require(b > 0, "Modulus by zero is not allowed");
      return a % b;
   }
```

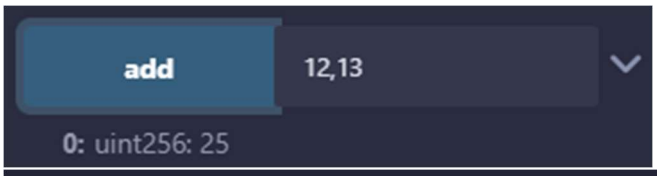
```
    // Function to demonstrate exponentiation
    function exponentiation(uint256 base, uint256 exponent) public pure returns (uint256) {
        uint256 result = 1;
        for (uint256 i = 0; i < exponent; i++) {
            result *= base;
        }
        return result;
    }
    // Function to demonstrate absolute value
    function absoluteValue(int256 a) public pure returns (uint256) {
        return a >= 0 ? uint256(a) : uint256(-a);
    }
    // Function to demonstrate minimum value
    function minimum(uint256 a, uint256 b) public pure returns (uint256) {
        return a < b ? a : b;
    }
    // Function to demonstrate maximum value
    function maximum(uint256 a, uint256 b) public pure returns (uint256) {
        return a > b ? a : b;
    }
}
```

Output:

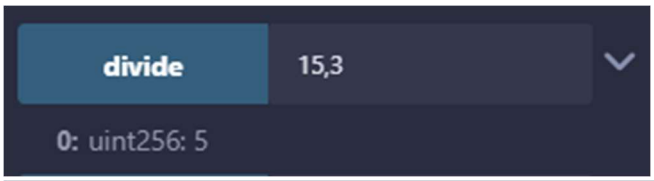

MATHEMATICALFUNCTIONS
Balance: 0 ETH
absoluteValue
12
0: uint256: 12
add
12,13
0: uint256: 25
divide
15,3
0: uint256: 5
exponentiation
2,3
0: uint256: 8
maximum
14,45
0: uint256: 45
minimum
14,43
0: uint256: 14
modulus
150,7
0: uint256: 3
multiply
12,3
0: uint256: 36
subtract
130,87
0: uint256: 43



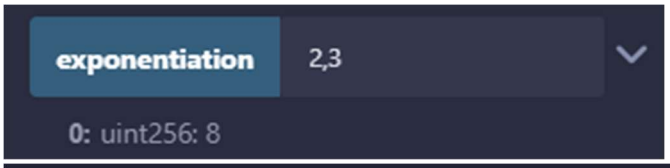

```
decoded input
{
    "int256 a": "12"
}

decoded output
{
    "0": "uint256: 12"
}
```



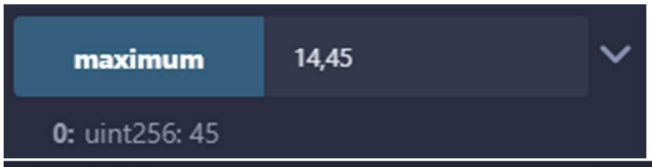

```
decoded input
{
    "uint256 a": "12",
    "uint256 b": "13"
}

decoded output
{
    "0": "uint256: 25"
}
```



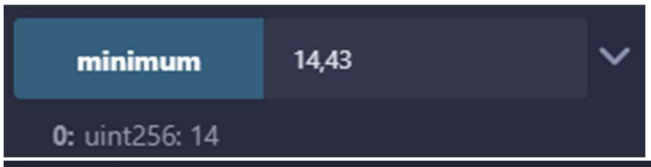

```
decoded input
{
    "uint256 a": "15",
    "uint256 b": "3"
}

decoded output
{
    "0": "uint256: 5"
}
```



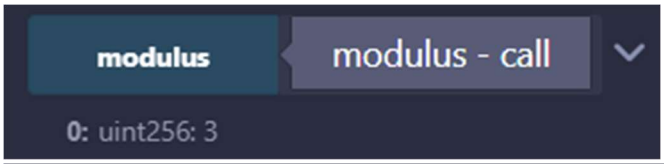

```
decoded input
{
    "uint256 base": "2",
    "uint256 exponent": "3"
}

decoded output
{
    "0": "uint256: 8"
}
```



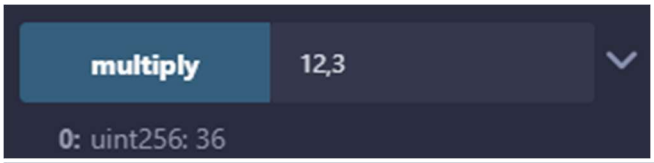

```
decoded input
{
    "uint256 a": "14",
    "uint256 b": "45"
}

decoded output
{
    "0": "uint256: 45"
}
```



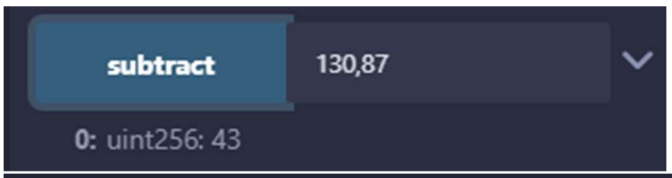

```
decoded input
{
    "uint256 a": "14",
    "uint256 b": "43"
}

decoded output
{
    "0": "uint256: 14"
}
```




```
decoded input
{
    "uint256 a": "150",
    "uint256 b": "7"
}
decoded output
{
    "0": "uint256: 3"
}
```




```
decoded input
{
    "uint256 a": "12",
    "uint256 b": "3"
}
decoded output
{
    "0": "uint256: 36"
}
```




```
decoded input
{
    "uint256 a": "130",
    "uint256 b": "87"
}
decoded output
{
    "0": "uint256: 43"
}
```

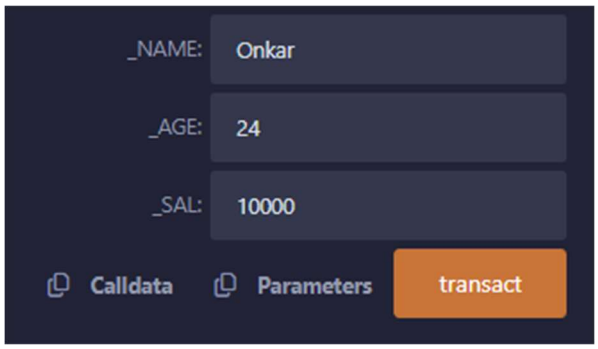
Q.17 Write a solidity smart contract to demonstrate inheritance in contract.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

// Base contract
contract Animal {
    // State variable to store the name of the animal
    string private name;
    // Constructor to initialize the name of the animal
    constructor(string memory _name) {
        name = _name;
    }
    // Function to get the name of the animal
    function getName() public view returns (string memory) {
        return name;
    }
    // Function to make a sound
    function makeSound() public virtual returns (string memory) {
        return "The animal makes a sound";
    }
}

// Derived contract that inherits from Animal
contract Dog is Animal {
    // Constructor to initialize the name of the dog
    constructor(string memory _name) Animal(_name) {}
```

```
    // Override the makeSound function to make a dog sound
    function makeSound() public override returns (string memory) {
        return "The dog barks";
    }
    // Function to wag the tail
    function wagTail() public pure returns (string memory) {
        return "The dog wags its tail";
    }
}

// Derived contract that inherits from Animal
contract Cat is Animal {
    // Constructor to initialize the name of the cat
    constructor(string memory _name) Animal(_name) {}
    // Override the makeSound function to make a cat sound
    function makeSound() public override returns (string memory) {
        return "The cat meows";
    }
    // Function to scratch
    function scratch() public pure returns (string memory) {
        return "The cat scratches";
    }
}
```

Output:


_NAME:
Onkar
_AGE:
24
_SAL:
10000
Calldata
Parameters
transact

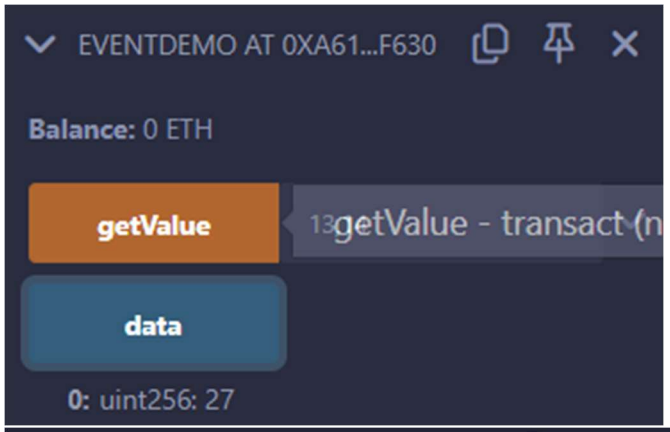

decoded input
{
"string _name": "Onkar",
"string _age": "24",
"string _sal": "10000"
}

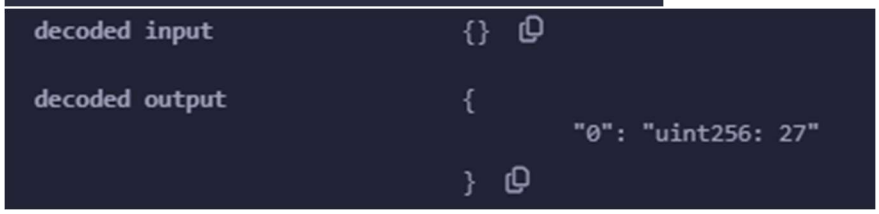
Q.18 Write a solidity smart contract to demonstrate events.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract EventDemo{
   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:



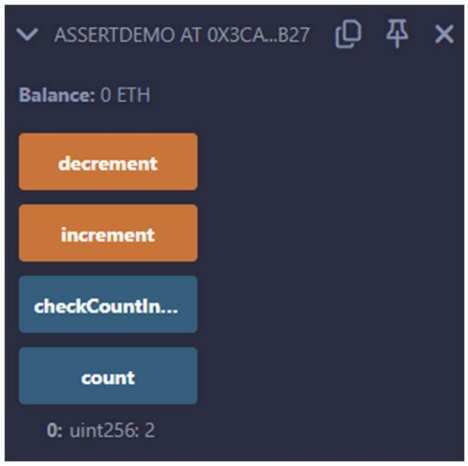


Q.19 Write a solidity smart contract to demonstrate assert statement and revert statement.

Code:

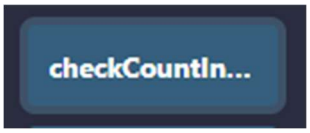
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract AssertDemo {
   uint public count;
   function increment() public returns (uint) {
      count += 1;
      return count;
   }
   function decrement() public returns (uint) {
      require(count > 0, "Count must be greater than zero");
      count -= 1;
      return count;
   }
   function checkCountInVarient() public view {
      assert(count >= 1);
   }
}
```

Output:




decrement
decoded output
{
"0": "uint256: 396387739119734445357598304
}
logs
[]
raw logs
[]
transact to AssertDemo.decrement errored: Error occurred: revert.
revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Count must be greater than zero".
You may want to cautiously increase the gas limit if the transaction went out of gas.


increment
decoded output
{
"0": "uint256: 1"
}


checkCountIn...

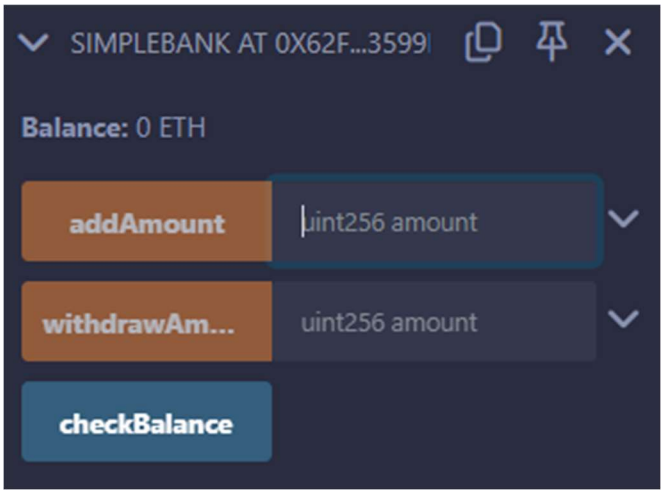

count
0: uint256: 1
{
"0": "uint256: 1"
}

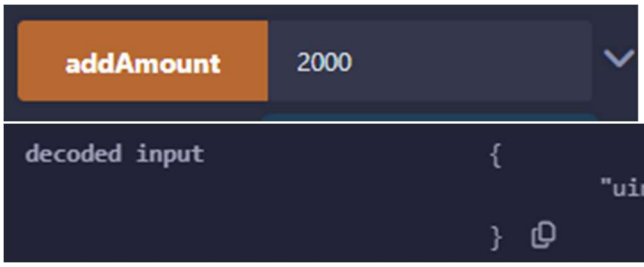
Q.20 Write a solidity smart contract for Bank Account which provides operations such as check account balance, withdraw amount and deposit amount etc.

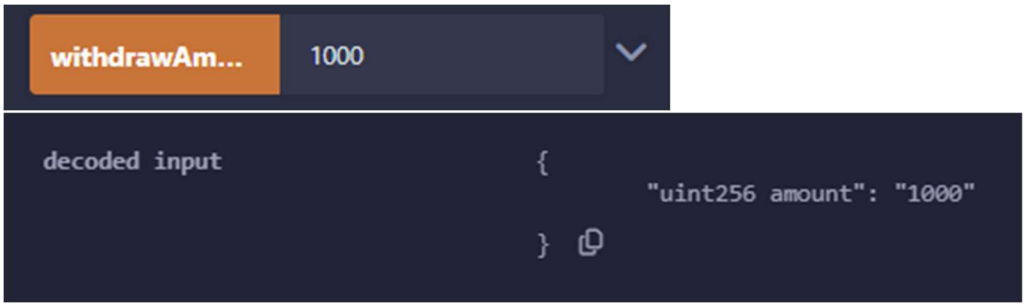
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract SimpleBank {
   // State variable to store the balance
   uint256 private balance;
   // Constructor to initialize balance
   constructor() {
      balance = 0;
   }
   // Function to add (deposit) amount to the balance
   function addAmount(uint256 amount) public {
      balance += amount;
   }
   // Function to withdraw amount from the balance
   function withdrawAmount(uint256 amount) public {
      require(amount <= balance, "Insufficient balance");
      balance -= amount;
   }
   // Function to check the remaining balance
   function checkBalance() public view returns (uint256) {
      return balance;
   }
}
```

Output:


SIMPLEBANK AT 0X62F...3599
Balance: 0 ETH
addAmount
uint256 amount
withdrawAm...
uint256 amount
checkBalance


addAmount
2000
decoded input
{
"uint256 amount": "2000"
}


withdrawAm...
1000
decoded input
{
"uint256 amount": "1000"
}


checkBalance
0: uint256: 1000
decoded output
{
"0": "uint256: 1000"
}

Q.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

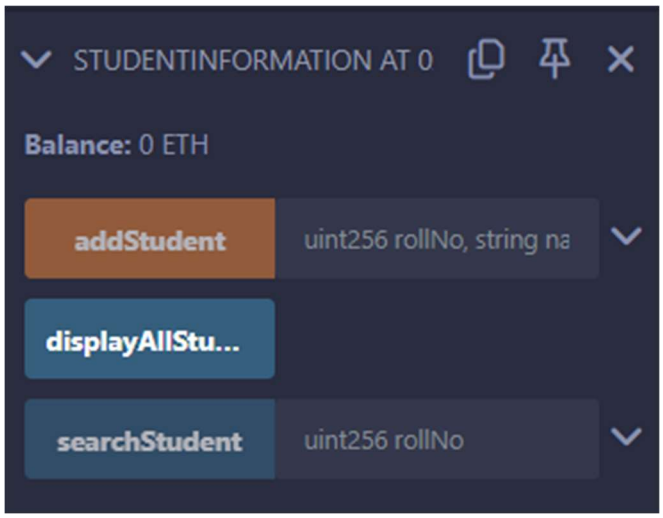
iii) Display all information

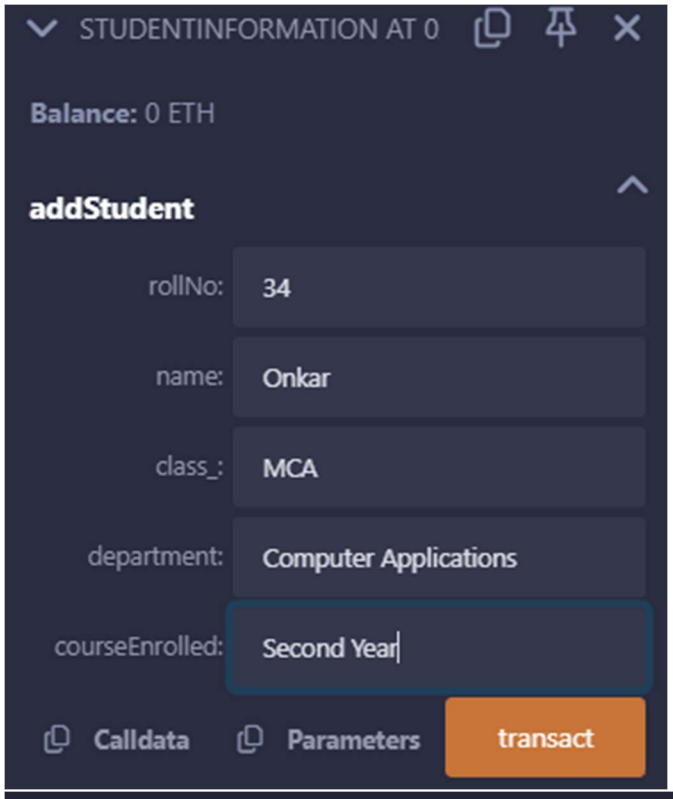
Code:

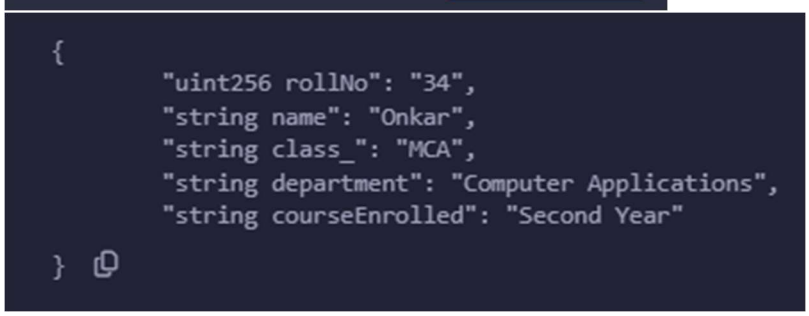
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
contract StudentInformation {
   // Struct to represent a student
   struct Student {
      uint256 rollNo;
      string name;
      string class_;
      string department;
      string courseEnrolled;
   }
   // Mapping to store students by roll number
   mapping(uint256 => Student) private students;
   // Function to add a new student
   function addStudent(uint256 rollNo, string memory name, string memory class_, string memory department, string memory courseEnrolled) public {
      require(rollNo > 0, "Roll number must be greater than zero");
      require(bytes(name).length > 0, "Name cannot be empty");
      require(bytes(class_).length > 0, "Class cannot be empty");
      require(bytes(department).length > 0, "Department cannot be empty");
      require(bytes(courseEnrolled).length > 0, "Course enrolled cannot be empty");
      students[rollNo] = Student(rollNo, name, class_, department, courseEnrolled);
```

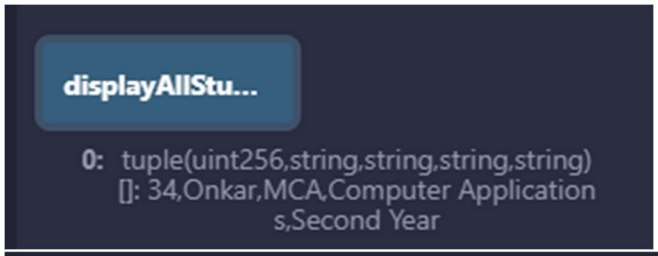
```
    }
    // Function to search for a student by roll number
    function searchStudent(uint256 rollNo) public view returns (Student memory) {
        require(students[rollNo].rollNo > 0, "Student not found");
        return students[rollNo];
    }
    // Function to display all student information
    function displayAllStudents() public view returns (Student[] memory) {
        uint256 count = 0;
        for (uint256 i = 1; i <= 100; i++) {
            if (students[i].rollNo > 0) {
                count++;
            }
        }
        Student[] memory allStudents = new Student[](count);
        uint256 index = 0;
        for (uint256 i = 1; i <= 100; i++) {
            if (students[i].rollNo > 0) {
                allStudents[index] = students[i];
                index++;
            }
        }
        return allStudents;
    }
}
```

Output:

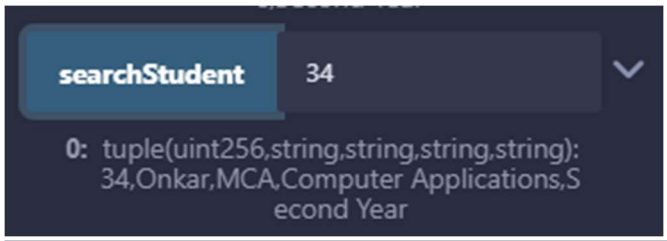

STUDENTINFORMATION AT 0
Balance: 0 ETH
addStudent
uint256 rollNo, string na
displayAllStu...
searchStudent
uint256 rollNo


STUDENTINFORMATION AT 0
Balance: 0 ETH
addStudent
rollNo:
34
name:
Onkar
class_:
MCA
department:
Computer Applications
courseEnrolled:
Second Year
Calldata
Parameters
transact


{
"uint256 rollNo": "34",
"string name": "Onkar",
"string class_": "MCA",
"string department": "Computer Applications",
"string courseEnrolled": "Second Year"
}




```
{
        "0": "tuple(uint256,string,string,string,string)[]: 34,Onkar,MCA,Computer Applications,Second Year"
}
```


```
decoded input
{
        "uint256 rollNo": "34"
}

decoded output
{
        "0": "tuple(uint256,string,string,string,string): 34,Onkar,MCA,Computer Applications,Second Year"
}
```

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

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

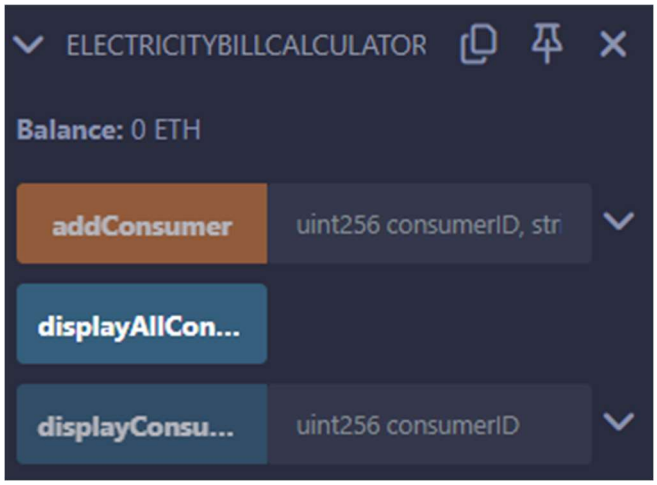
Code:

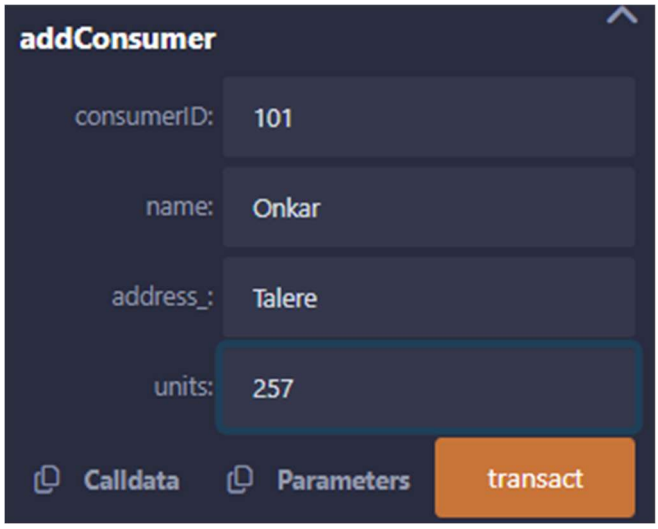
```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.19;


contract ElectricityBillCalculator {

  // Struct to represent a consumer

  struct Consumer {

    string name;

    string address_;

    uint256 consumerID;

    uint256 units;

    uint256 amount;

  }


  // Mapping to store consumers by consumer ID

  mapping(uint256 => Consumer) private consumers;


  // Function to add a new consumer

  function addConsumer(uint256 consumerID, string memory name, string memory address_, uint256 units) public {

    require(consumerID > 0, "Consumer ID must be greater than zero");

    require(bytes(name).length > 0, "Name cannot be empty");

    require(bytes(address_).length > 0, "Address cannot be empty");
```

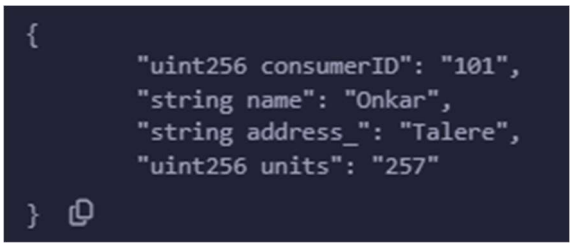
```
        require(units >= 0, "Units cannot be negative");

        uint256 amount = calculateBill(units);
        consumers[consumerID] = Consumer(name, address_, consumerID, units, amount);
    }

    // Function to calculate the total electricity bill
    function calculateBill(uint256 units) internal pure returns (uint256) {
        uint256 amount;
        if (units <= 50) {
            amount = units * 50; // Rs. 0.50/unit
        } else if (units <= 150) {
            amount = 50 * 50 + (units - 50) * 75; // Rs. 0.50/unit for first 50 units, Rs. 0.75/unit for next 100 units
        } else if (units <= 250) {
            amount = 50 * 50 + 100 * 75 + (units - 150) * 120; // Rs. 0.50/unit for first 50 units, Rs. 0.75/unit for next 100 units, Rs. 1.20/unit for next 100 units
        } else {
            amount = 50 * 50 + 100 * 75 + 100 * 120 + (units - 250) * 150; // Rs. 0.50/unit for first 50 units, Rs. 0.75/unit for next 100 units, Rs. 1.20/unit for next 100 units, Rs. 1.50/unit for units above 250
        }

        // Additional surcharge of 20%
        amount = amount + (amount * 20 / 100);
        return amount;
    }
```

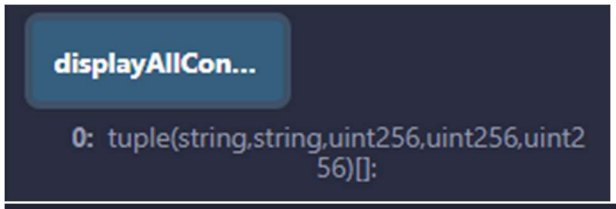
```
    // Function to display the information of a consumer
    function displayConsumer(uint256 consumerID) public view returns (Consumer memory) {
        require(consumers[consumerID].consumerID > 0, "Consumer not found");
        return consumers[consumerID];
    }

    // Function to display the information of all consumers
    function displayAllConsumers() public view returns (Consumer[] memory) {
        uint256 count = 0;
        for (uint256 i = 1; i <= 100; i++) {
            if (consumers[i].consumerID > 0) {
                count++;
            }
        }

        Consumer[] memory allConsumers = new Consumer[](count);
        uint256 index = 0;
        for (uint256 i = 1; i <= 100; i++) {
            if (consumers[i].consumerID > 0) {
                allConsumers[index] = consumers[i];
                index++;
            }
        }

        return allConsumers;
    }
}
```

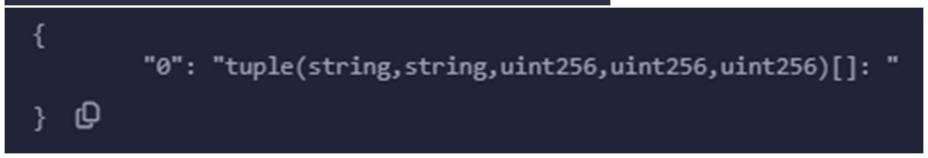
Output:

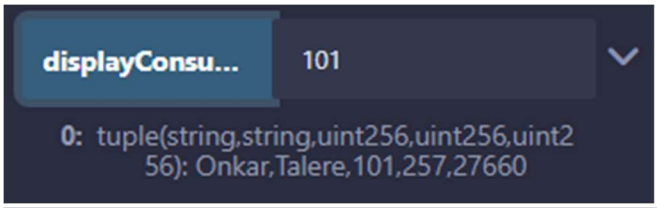

ELECTRICITYBILLCALCULATOR
Balance: 0 ETH
addConsumer
uint256 consumerID, str
displayAllCon...
displayConsu...
uint256 consumerID

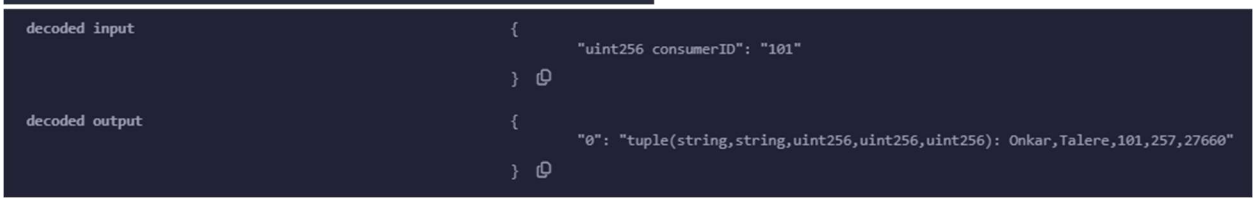

addConsumer
consumerID:
101
name:
Onkar
address_:
Talere
units:
257
Calldata
Parameters
transact

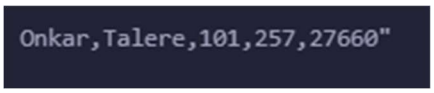

{
"uint256 consumerID": "101",
"string name": "Onkar",
"string address_": "Talere",
"uint256 units": "257"
}


displayAllCon...
0: tuple(string,string,uint256,uint256,uint256)[]:


{
"0": "tuple(string,string,uint256,uint256,uint256)[]: "
}


displayConsu...
101
0: tuple(string,string,uint256,uint256,uint256): Onkar,Talere,101,257,27660


decoded input
{
"uint256 consumerID": "101"
}
decoded output
{
"0": "tuple(string,string,uint256,uint256,uint256): Onkar,Talere,101,257,27660"
}


Onkar,Talere,101,257,27660"