Practical No:03 Roll No:16

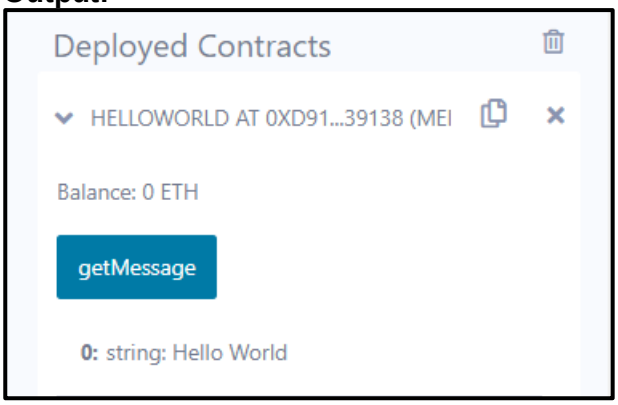
Solidity

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

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract HelloWorld
{
   function getMessage() public view returns(string memory)
   {
      return 'Hello World';
   }
}
```

Output:-

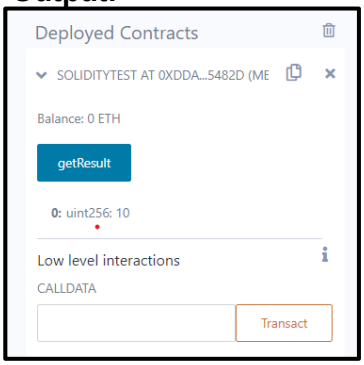


2. **Write a solidity smart contract to demonstrate state variable, local variable and global variable.**

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract SolidityTest {
   uint storedData; // State variable
   constructor() public
   {
      storedData = 10;
   }
   function getResult() public view returns(uint){
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return storedData;
   }
}
```

Output:-



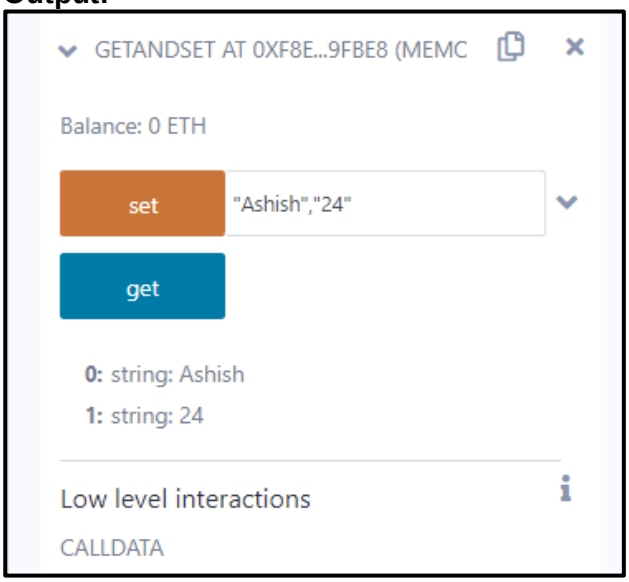
Practical No:03 Roll No:16

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

Code:-

```
pragma solidity >=0.7.0 <0.9.0;

contract GetAndSet{
    string name;
    string  lname;
 function set(string memory newName, string memory lastname) public {
 name = newName;
 lname = lastname;
 }
 function get() public view returns (string memory, string memory) {
 return (name,lname);
 }
}
```

Output:



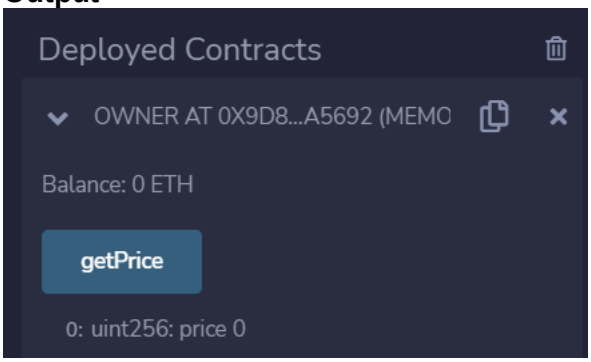
4. **Write a solidity smart contract to demonstrate function modifier.**

Code:-

```
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-

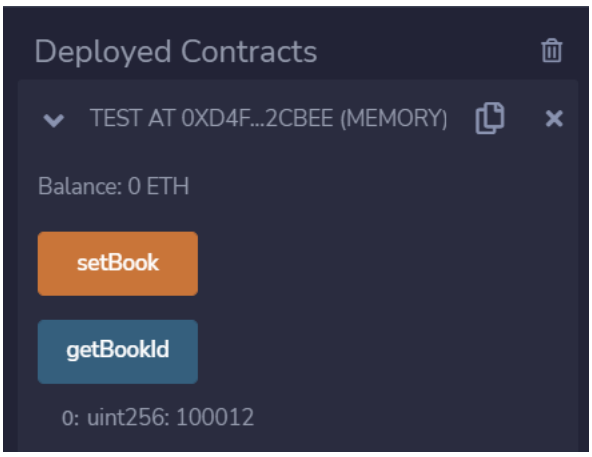


5. **Write a solidity smart contract to demonstrate use of structure.**

Code-

```
 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:

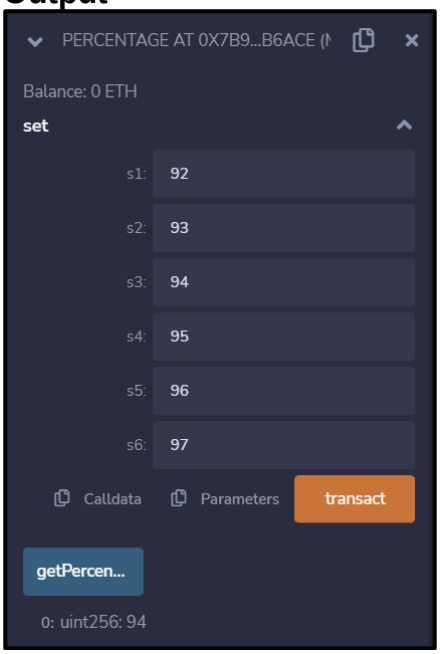


6. **Write a solidity smart contract to calculate percentage of marks obtained by students for six subject in final examination.**

Code-

```
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-

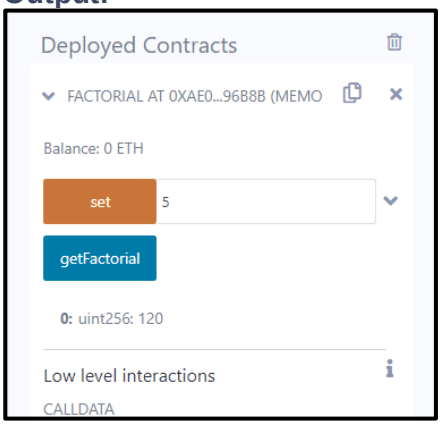


7. **Write a solidity smart contract to find the factorial of entered number.**

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract factorial
{
    uint number;
    function set(uint num) public
    {
        number=num;
    }
    function getFactorial() public view returns (uint)
    {
        uint fact=1;
        for(uint i=2;i<=number;i++)
        {
            fact=fact*i;
        }
        return fact;
    } }
```

Output:-



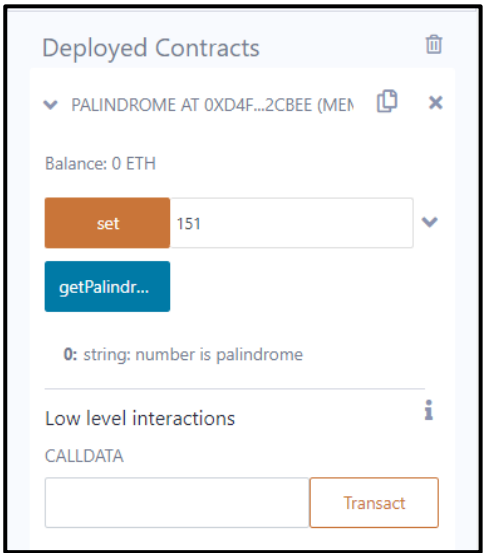
8. **Write a solidity smart contract to check whether entered number is palindrome or not.**

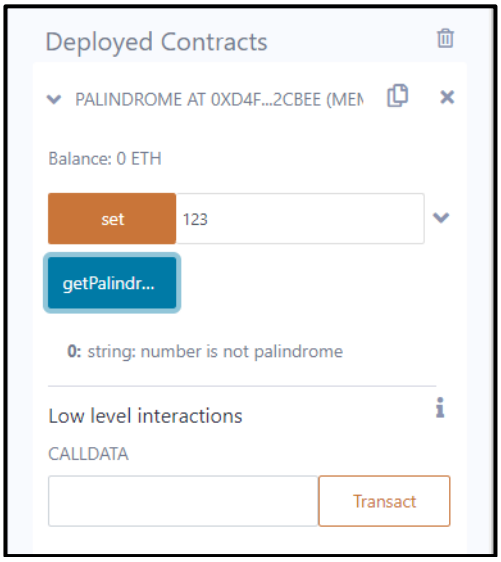
Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract palindrome{
    uint number;
    function set(uint n) public
    {
        number=n;
    }
    function getPalindrome() public view returns (string memory ) {
        uint rev;
        uint n=number;
        uint reverseNumber=0;
        while(n>0){
            rev=n%10;
            reverseNumber=reverseNumber*10+rev;
            n=n/10;
        }
        if(reverseNumber==number)
```

```
            return "number is palindrome";
        else
            return "number is not palindrome";
    }
}
```

Output:-



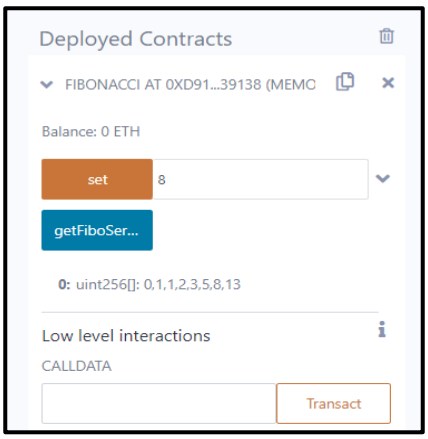


9. **Write a solidity smart contract to generate Fibonacci Series up to given number.**

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
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:-

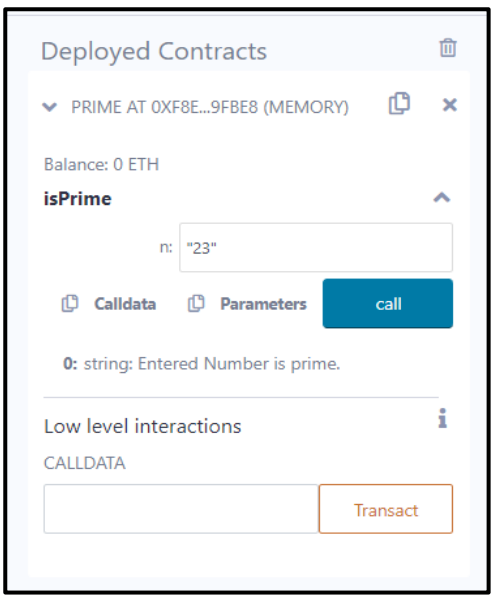


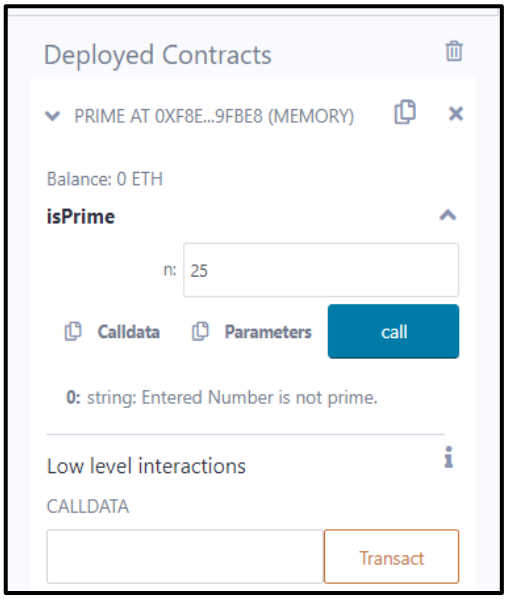
10. Write a solidity smart contract to check whether entered number is prime number or not.

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
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:-



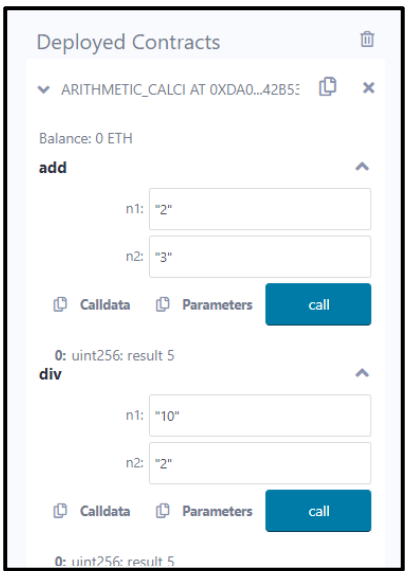


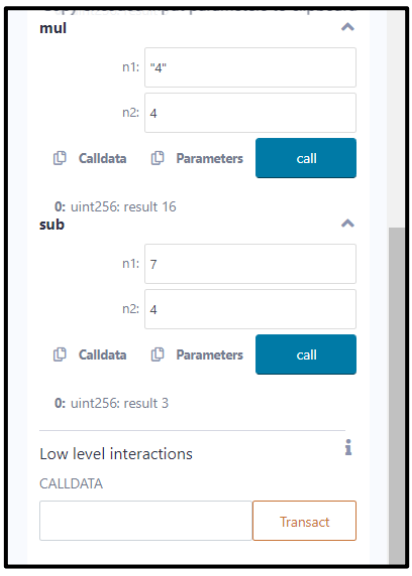
11. Write a solidity smart contract to create arithmetic calculator which includes functions for operations addition, subtraction, multiplication, division etc.

Code:

```
pragma solidity >=0.7.0 <0.9.0;
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;
 }

}
```

Output:-



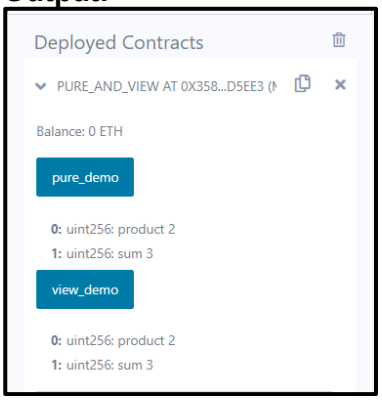


12. Write a solidity smart contract to demonstrate view function and pure function.

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract pure_and_view {
    function view_demo() public view returns(uint product, uint sum)
    {
        uint a = 1; uint b = 2; // local variable
        product = a * b;
        sum = a + b;
    }
    function pure_demo() public pure returns(uint product, uint sum)
    {
        uint a = 1;
        uint b = 2;
        product = a * b;
        sum = a + b;
    }
}
```

Output:-



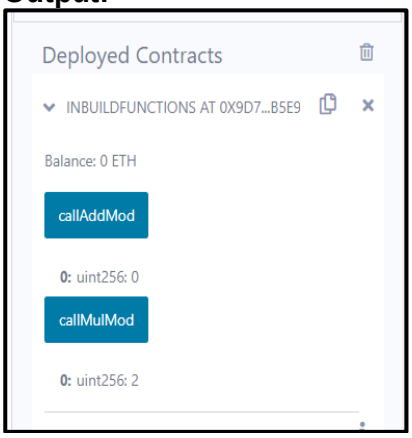
13. Write a solidity smart contract to demonstrate inbuilt mathematical functions.

Code:-

```
pragma solidity >=0.7.0 <0.9.0;
contract inBuildFunctions {
    function callAddMod() public pure returns(uint)
    {
      return addmod(4, 5, 3);
```

```
    }
    function callMulMod() public pure returns(uint)
    {
      return mulmod(4, 5, 3);
    }
}
```

Output:-

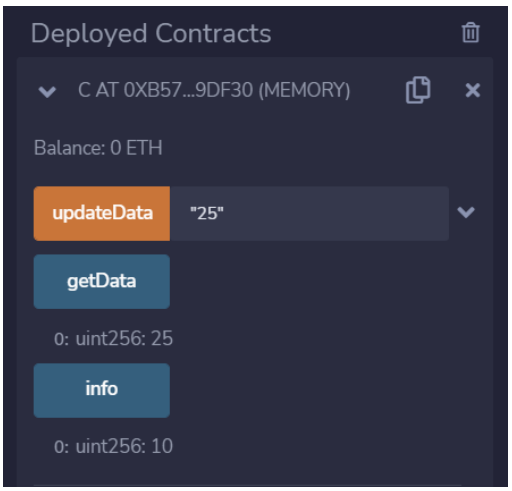


14. Write a solidity smart contract to demonstrate inheritance in contract.

Code:-

```
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-



15. Write a solidity smart contract to demonstrate events.

Code:-

```
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:

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

Code:-

```
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:

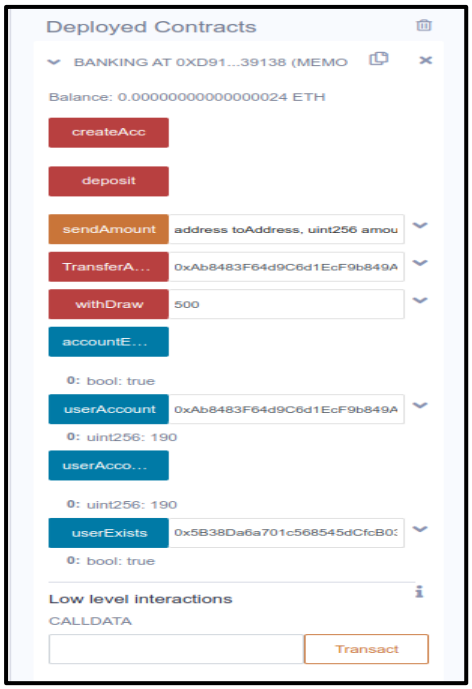
17. **Write a solidity smart contract for Bank Account which provides operations such as check account balance, withdraw amount and deposit amount etc.**

Code:-

```
pragma solidity >=0.7.0 <0.9.0
contract BankApplication {
    mapping(address => uint) public userAccount;
    mapping(address => bool) public userExists;

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

        userAccount[msg.sender] = msg.value;
        userExists[msg.sender] = true;
        return 'account created';
    }

    function deposit() public payable returns (string memory) {
        require (userExists[msg.sender] == true, 'Account does not Exist');
        require (msg.value > 0, 'Value less than zero') ;

        userAccount[msg.sender] += msg.value;

        return 'Amount Deposited';
    }

    function withdraw(uint amount) public payable returns (string memory) {
        require (userExists[msg.sender] == true, 'Account does not Exist');
        require (userAccount[msg.sender] > amount, 'Insufficient Balance');
        require (amount > 0, 'Value less than zero');

        userAccount[msg.sender] -= amount;

        return 'Withdraw Sucessfull';
    }

    function TransferAmount(address payable userAddress, uint amount) public payable
returns (string memory) {
        require(userAccount[msg.sender] > amount, 'Insufficient Balance');
        require(userExists[msg.sender] == true, 'Account does not Exist');
        require(userExists[userAddress] == true, 'Transfer Account not Available');
```

```
        require(amount > 0, 'Value less than zero');

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

        return 'Transfer Successful';
    }

    function SendAmount(address payable toAddress, uint256 amount) public payable
returns (string memory) {
        require(userAccount[msg.sender] > amount, 'Insufficient Balance');
        require(userExists[msg.sender] == true, 'Account does not Exist');
        require(amount > 0, 'Value less than zero');

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

        return 'Transfer Successful';
    }

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

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

Output:



Account created Successfully:

Practical No:03 Roll No:16

Account Exist or not:

Depositing amount to our accout (300):

Checking the account balance:

Practical No:03 Roll No:16

```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Banking.userAccountBalamce() data: 0x98f...e71d6

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Banking.userAccountBalamce() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost          23533 gas (Cost only applies when called by a contract)
input                   0x98f...e71d6
decoded input           {}
decoded output          {
                            "0": "uint256: 300"
                        }
logs                    []
```

Withdrawing amount more than our balance

```
[vm] from: 0x5B3...eddC4 to: Banking.withDraw(uint256) 0xd91...39138 value: 0 wei data: 0x141...001f4 logs: 0 hash: 0xeb2...84569

status                  false Transaction mined but execution failed
transaction hash        0xeb273ee5b9976f24eff6314260a8acb7af5dc1da193b9dd44cbd80a8baf84569
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Banking.withDraw(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                     3000000 gas
transaction cost        26055 gas
execution cost          26055 gas
input                   0x141...001f4
decoded input           {
                            "uint256 amount": "500"
                        }
decoded output          {
                            "error": "Failed to decode output: Error: overflow (fault=\"overflow\", operation=\"toNumber\",
value=\"396387739119734445357598304634811567422170074682075354633153435150806574 6944\", code=NUMERIC_FAULT, version=bignumber/5.5.0)"
                        }
logs                    []
val                     0 wei

transact to Banking.withDraw errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "You donot have enough balanced".
Debug the transaction to get more information.
```

Less than account balance:-

```
[vm] from: 0x5B3...eddC4 to: Banking.withDraw(uint256) 0xd91...39138 value: 0 wei data: 0x141...00014 logs: 0 hash: 0x3f4...71f9d

status                  true Transaction mined and execution succeed
transaction hash        0x3f4ebffe8a0e1ecd626f3f6facd1fa3e577098618f2318a97566231a32e71f9d
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Banking.withDraw(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                     34039 gas
transaction cost        29599 gas
execution cost          29599 gas
input                   0x141...00014
decoded input           {
                            "uint256 amount": "20"
                        }
decoded output          {
                            "0": "string: amount withDraw succeessfully"
                        }
logs                    []
val                     0 wei
```

Amount transfer to another mapping account:-

Practical No:03 Roll No:16

Amount send to another account:-