**Blockchain Technology**

**Lab Assignment No. : 03**

**Code:**

// SPDX-License-Identifier: Unlicensed

pragma solidity ^0.6.0;

contract MyBank

{

    mapping(address=> uint ) private \_balances;

    address public owner;

    event LogDepositeMade(address accountHoder, uint amount );

    constructor () public

     {

         owner=msg.sender;

         emit LogDepositeMade(msg.sender, 1000);

     }

        function deposite() public payable  returns (uint)

        {

            require ((\_balances[msg.sender] + msg.value) >  \_balances[msg.sender] && msg.sender!=address(0));

            \_balances[msg.sender] += msg.value;

            emit LogDepositeMade(msg.sender , msg.value);

            return \_balances[msg.sender];

        }

        function withdraw (uint withdrawAmount) public  returns (uint)

        {

                require (\_balances[msg.sender] >= withdrawAmount);

                require(msg.sender!=address(0));

                require (\_balances[msg.sender] > 0);

                \_balances[msg.sender]-= withdrawAmount;

                msg.sender.transfer(withdrawAmount);

                emit LogDepositeMade(msg.sender , withdrawAmount);

                return \_balances[msg.sender];

        }

        function viewBalance() public view returns (uint)

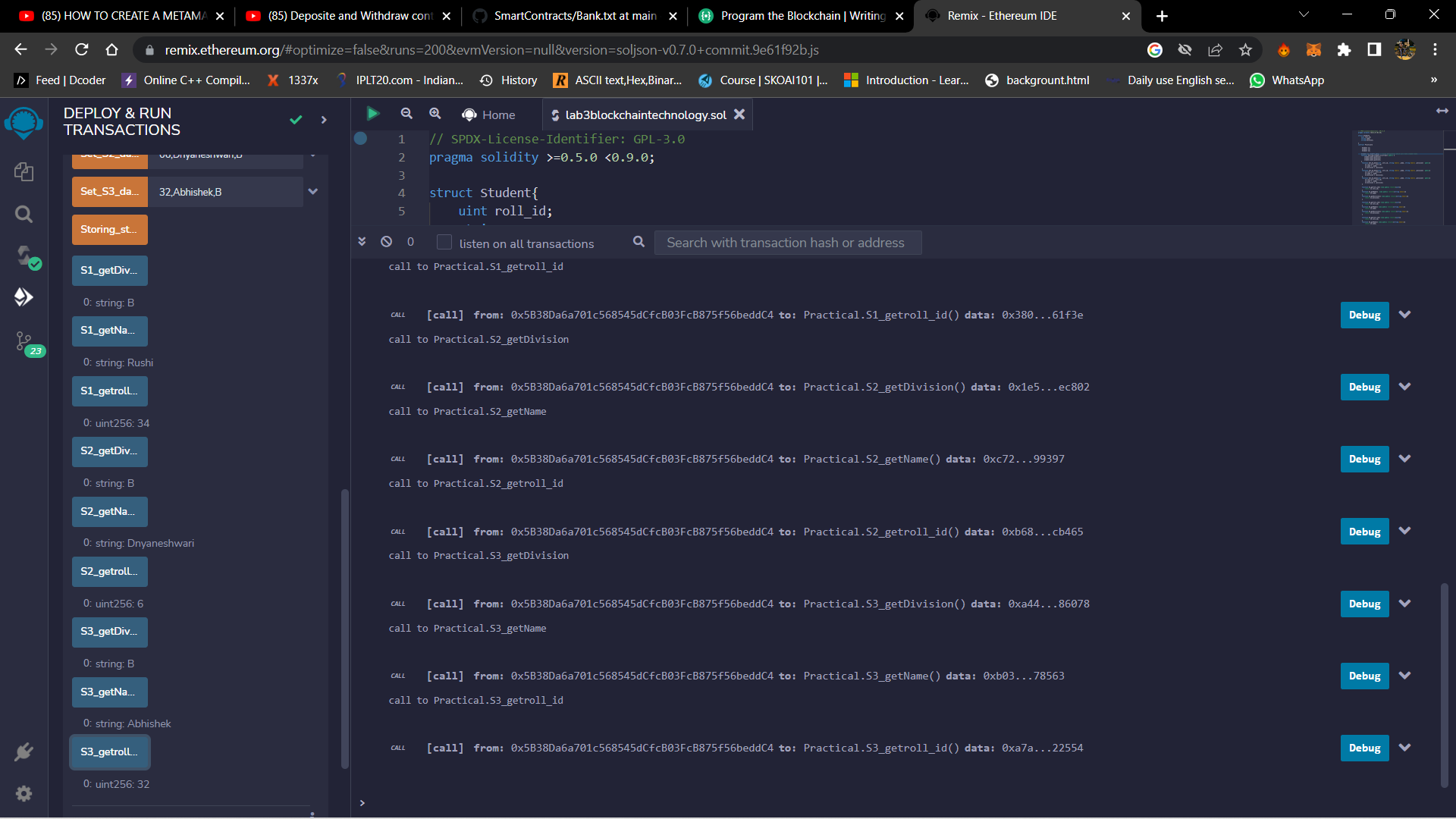
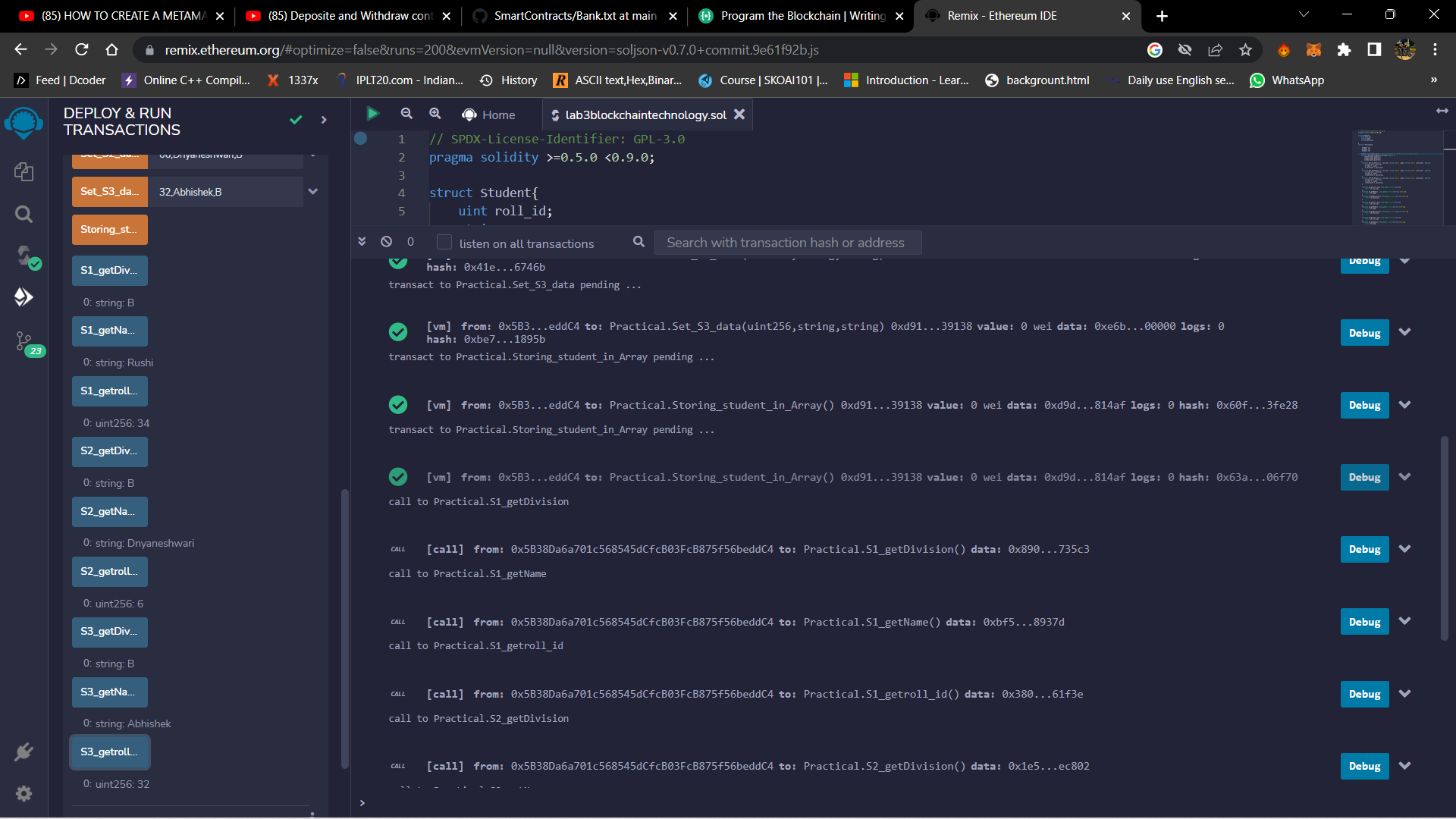
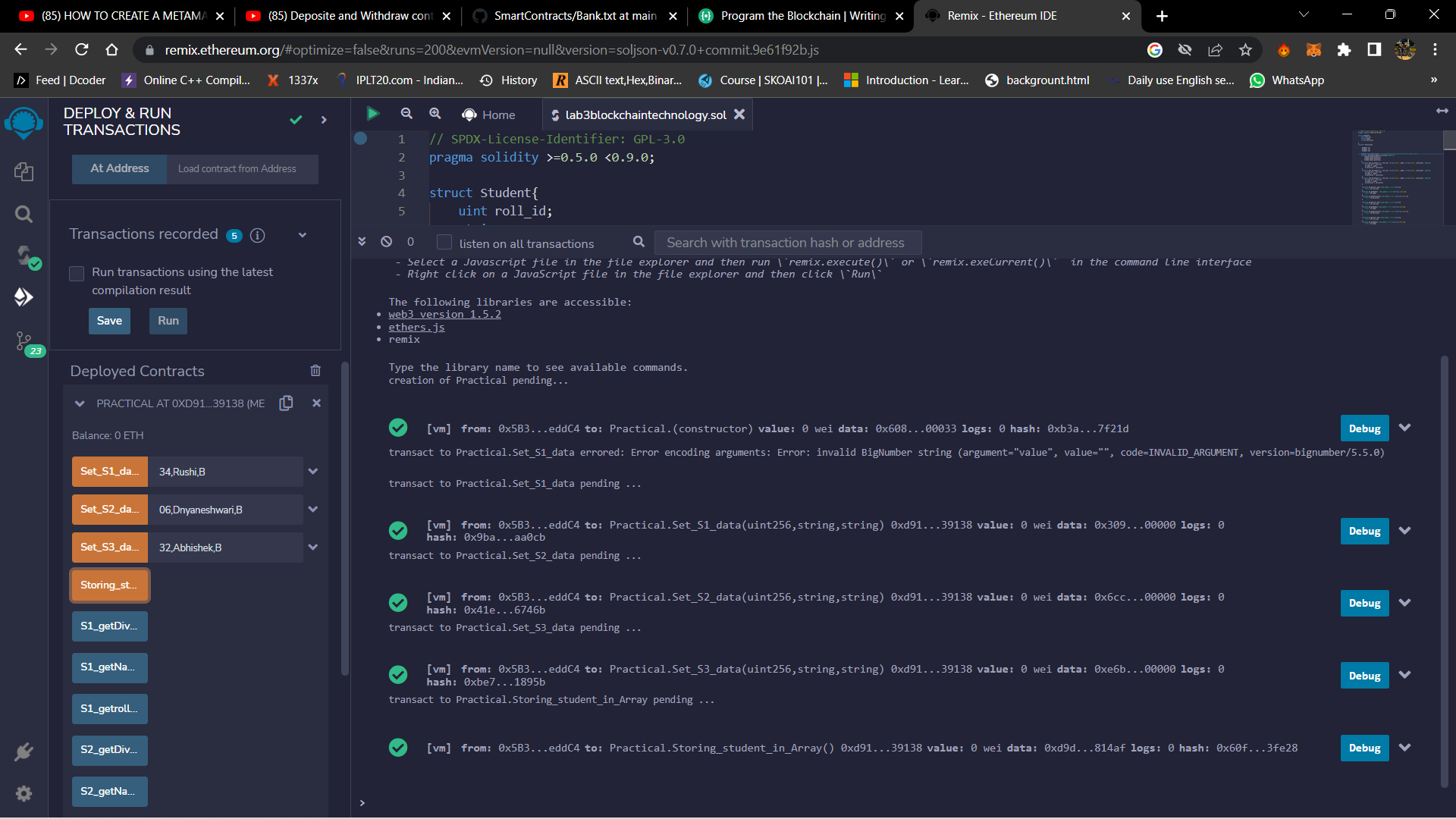
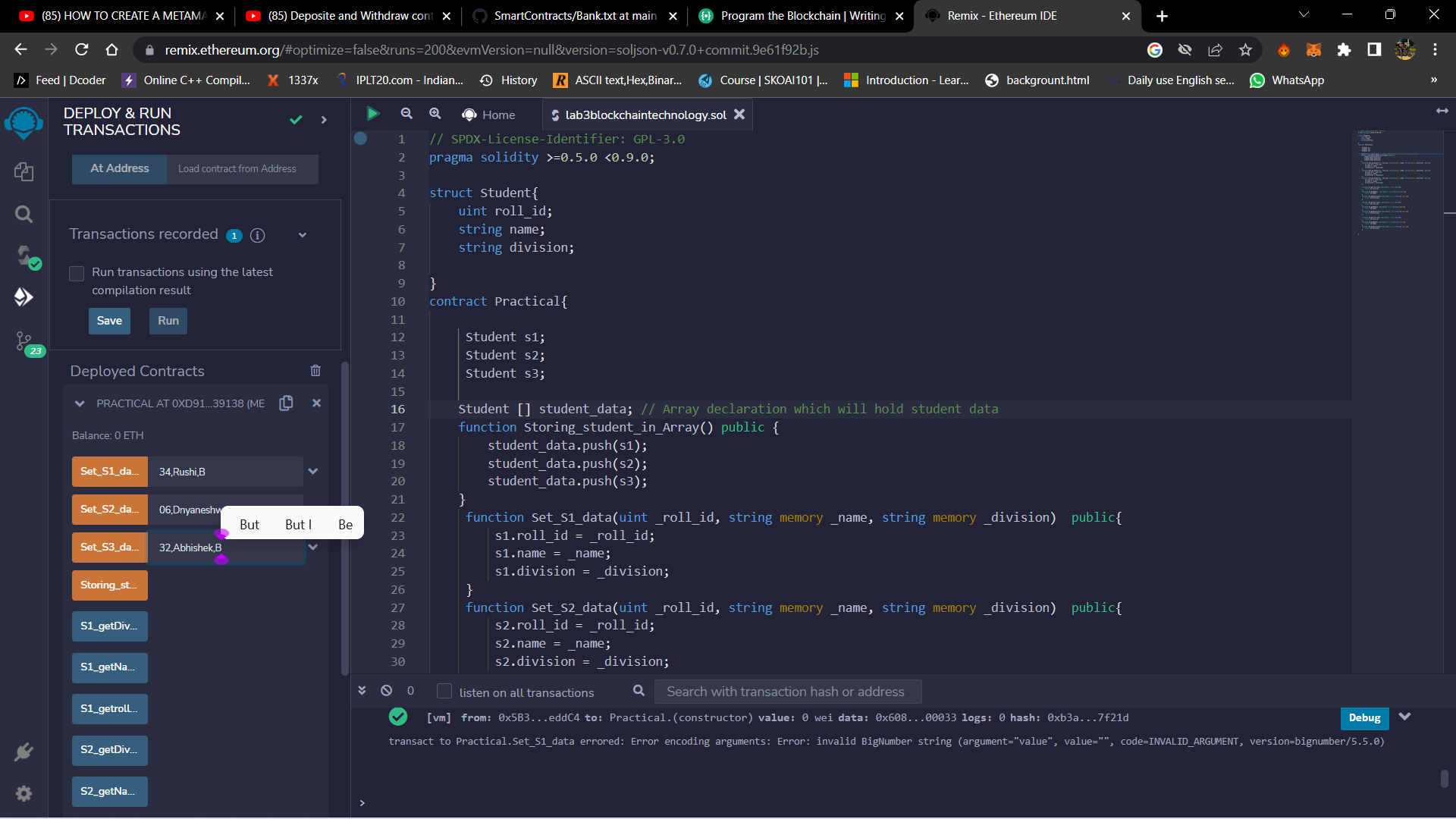
        {

            return \_balances[msg.sender];

        }

}

**Output:**



**Blockchain Technology**

**Lab Assignment No. : 04**

**Code:**

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.9.0;

struct Student{

    uint roll\_id;

    string name;

    string division;

}

contract Practical{

     Student s1;

     Student s2;

     Student s3;

    Student [] student\_data; // Array declaration which will hold student data

    function Storing\_student\_in\_Array() public {

        student\_data.push(s1);

        student\_data.push(s2);

        student\_data.push(s3);

    }

     function Set\_S1\_data(uint \_roll\_id, string memory \_name, string memory \_division)  public{

         s1.roll\_id = \_roll\_id;

         s1.name = \_name;

         s1.division = \_division;

     }

     function Set\_S2\_data(uint \_roll\_id, string memory \_name, string memory \_division)  public{

         s2.roll\_id = \_roll\_id;

         s2.name = \_name;

         s2.division = \_division;

     }

     function Set\_S3\_data(uint \_roll\_id, string memory \_name, string memory \_division)  public{

         s3.roll\_id = \_roll\_id;

         s3.name = \_name;

         s3.division = \_division;

     }

     function S1\_getroll\_id() view public returns(uint){

         return s1.roll\_id;

     }

     function S1\_getName()  view public returns(string memory){

         return s1.name;

     }

     function S1\_getDivision() view public returns(string memory){

         return s1.division;

     }

     function S2\_getroll\_id() view public returns(uint){

         return s2.roll\_id;

     }

     function S2\_getName() view public returns(string memory){

         return s2.name;

     }

     function S2\_getDivision() view public returns(string memory){

         return s2.division;

     }

     function S3\_getroll\_id() view public returns(uint){

         return s3.roll\_id;

     }

     function S3\_getName() view public returns(string memory){

         return s3.name;

     }

     function S3\_getDivision() view public returns(string memory){

         return s3.division;

     }

}

**Output:**

