

# BÁO CÁO THỰC HÀNH

Môn học: Blockchain – Nền tảng, ứng dụng và bảo mật

Lab 2: Remix, solidity và Smart Contract

GVHD: Đoàn Minh Trung

## 1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT547.P11.ANTT

STT	Họ và tên	MSSV	Email
1	Lê Huy Hiệp	21522067	21522067@gm.uit.edu.vn
2	Nguyễn Thanh Tuấn	21522756	21522756@gm.uit.edu.vn
3	Nguyễn Việt Khang	21522198	21522198@gm.uit.edu.vn

## 2. NỘI DUNG THỰC HIỆN:<sup>1</sup>

STT	Công việc	Kết quả tự đánh giá
1	Bài tập 1	100%
2	Bài tập 2	100%
3	Bài tập 3	100%
4	Bài tập 4	100%
5	Bài tập 5	100%
6	Bài tập 6	100%
7	Bài tập 7	100%
8	Bài tập 8	100%

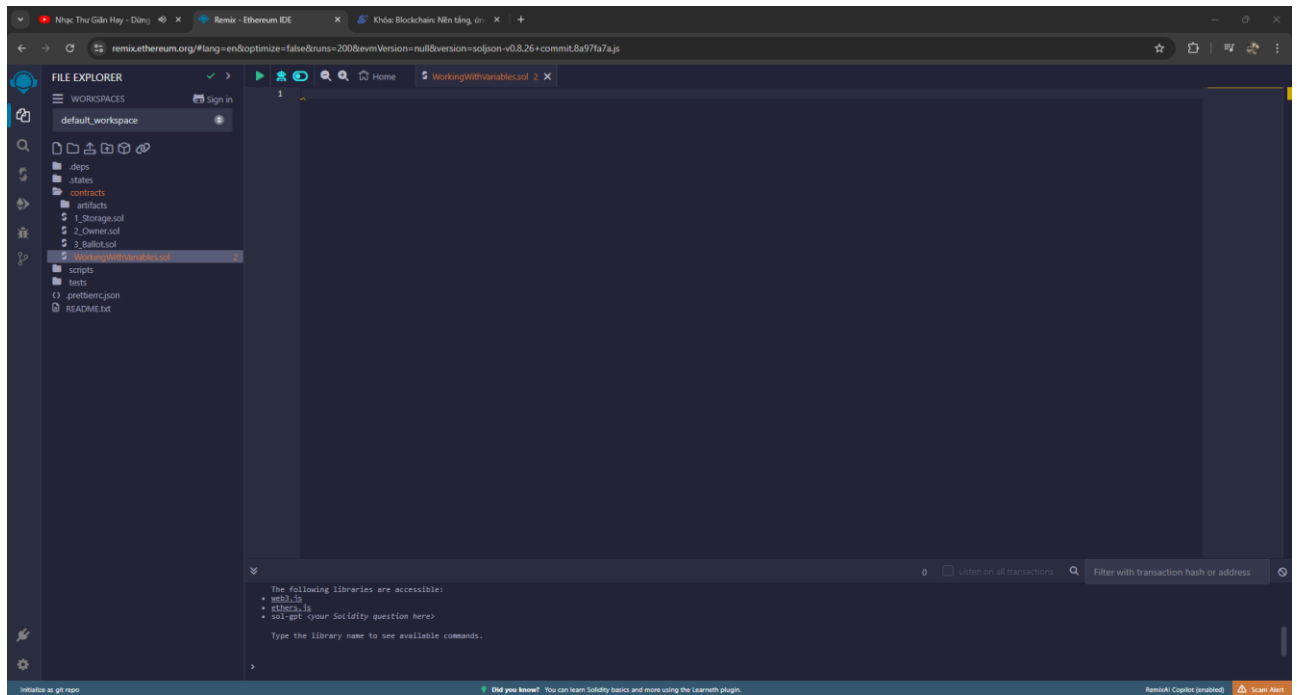
Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

<sup>1</sup> Ghi nội dung công việc, các kịch bản trong bài Thực hành

# BÁO CÁO CHI TIẾT

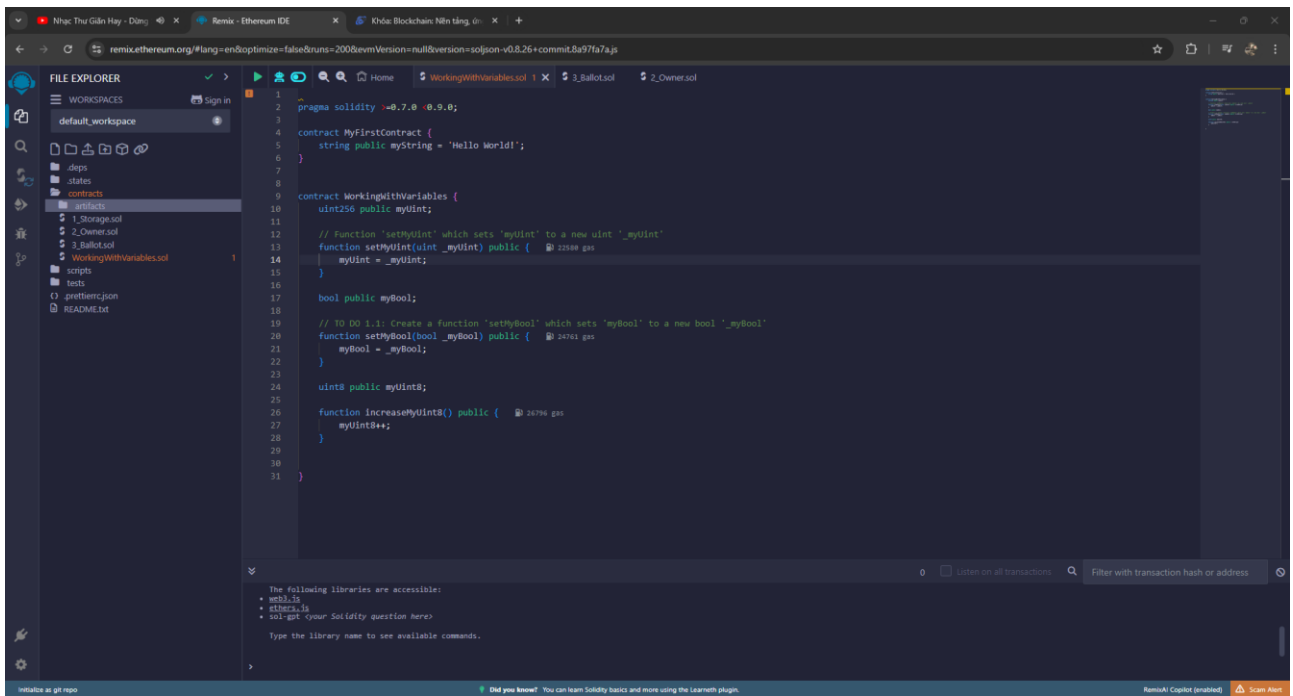
## Bài tập 1:

Tạo một tập tin mới có tên WorkingWithVariables.sol.



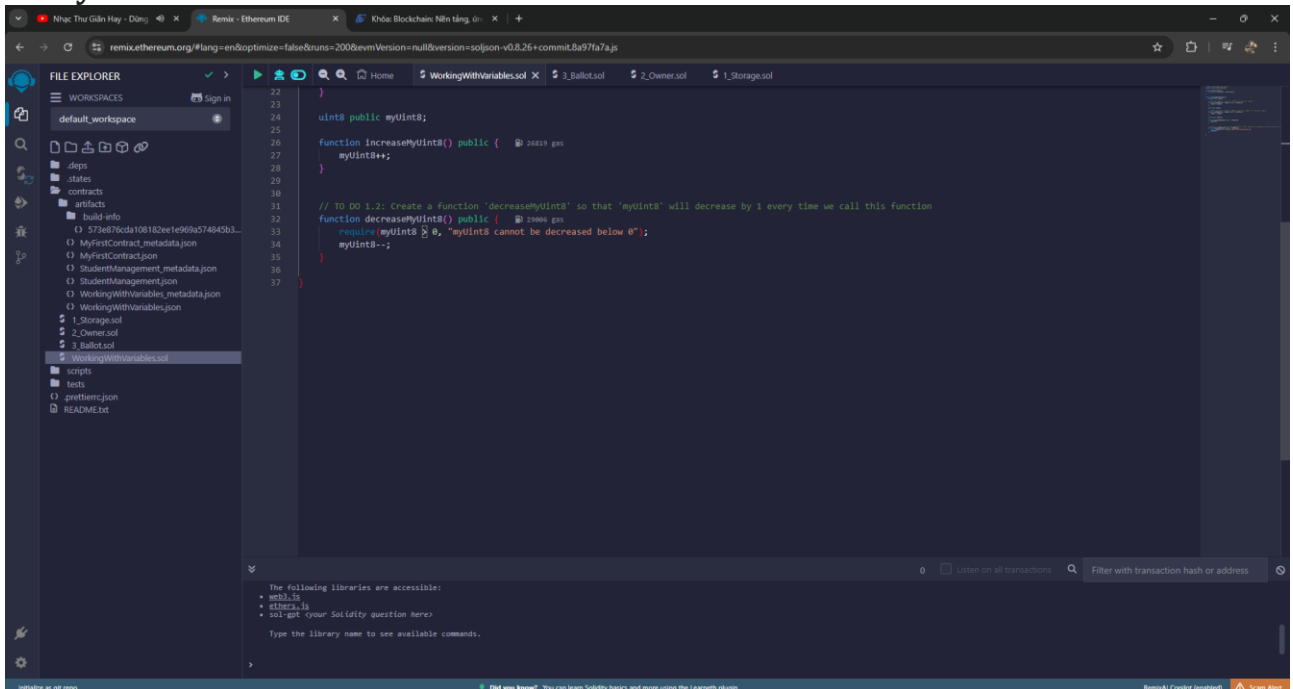
Thực hiện các yêu cầu 1.1, 1.2, 1.3, 1.4

### 1.1 Create a function 'setMyBool' which set 'Mybool' to a new bool '\_myBool'



=> SetMybool là một hàm dùng để gán giá trị boolean mới cho myBool bằng tham số đầu vào \_myBool.

### 1.2 Create a function 'decreaseMyUint8' so that 'myUint8' will decrease by 1 every time we call this function.



=> decreaseMyUint8 giảm giá trị của biến myUint8 xuống 1 mỗi lần được gọi, với yêu cầu là giá trị không được nhỏ hơn 0.

### 1.3 Create a function 'setAddress' which set 'myAddress' to a new address '\_myAddress'

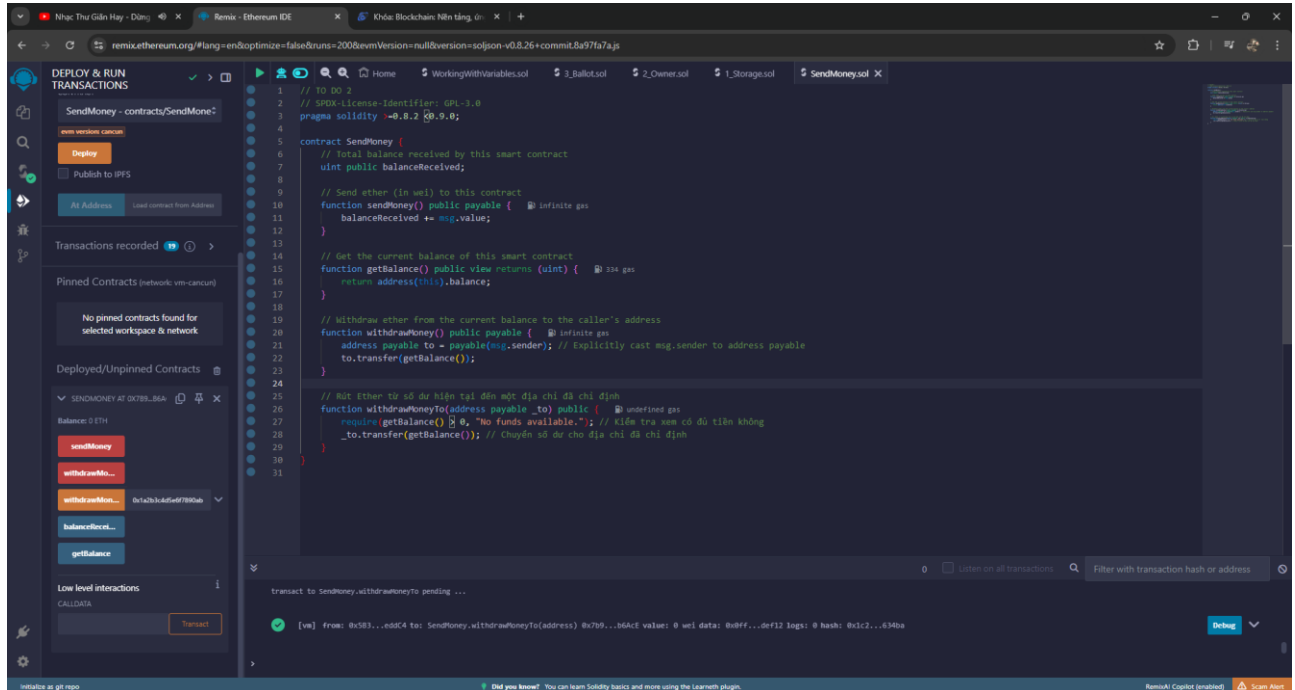
=> Tạo biến `_myAddress` để lưu địa chỉ mới và hàm `setAddress` cập nhật `myAddress` với một địa chỉ mới được cung cấp qua `_myAddress`.

### 1.4 Return to the balance of 'myAddress'

=> `getBalance` trả về số dư Ether liên kết với địa chỉ được lưu trong `myAddress`.

## Bài tập 2:

### 2.1 Transfer current balance to selected address `_to.transfer(this.getBalance());`



Function `WithdrawMoneyTo` nhận một tham số `_To`, là địa chỉ chỉ định sẽ chuyển tiền đến. Tham số này bắt buộc phải khai báo bằng `payable` để có thể nhận ether.

Trước tiên, nó sẽ kiểm tra số dư của smart contract, nếu đủ => chuyển tiền cho địa chỉ `_to`.

## Bài tập 3:

### 3.1 We want the 'owner' of this contract is the 'msg.sender' owner = msg.sender

Để địa chỉ của người gọi là owner, ta chỉ cần viết constructor () cho owner = msg.sender;

```

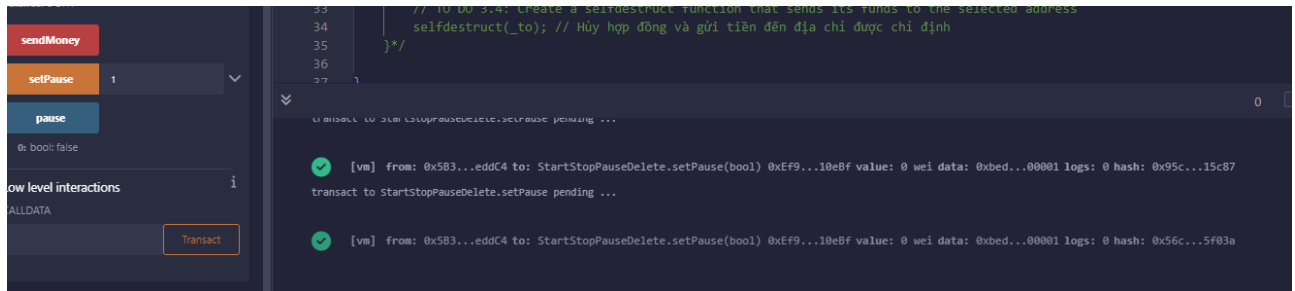
// Constructor để khởi tạo owner
constructor() {
    // TO DO 3.1: We want the 'owner' of this contract is the 'msg.sender'
    owner = msg.sender; // Đặt owner là địa chỉ của người gọi hàm
}

```

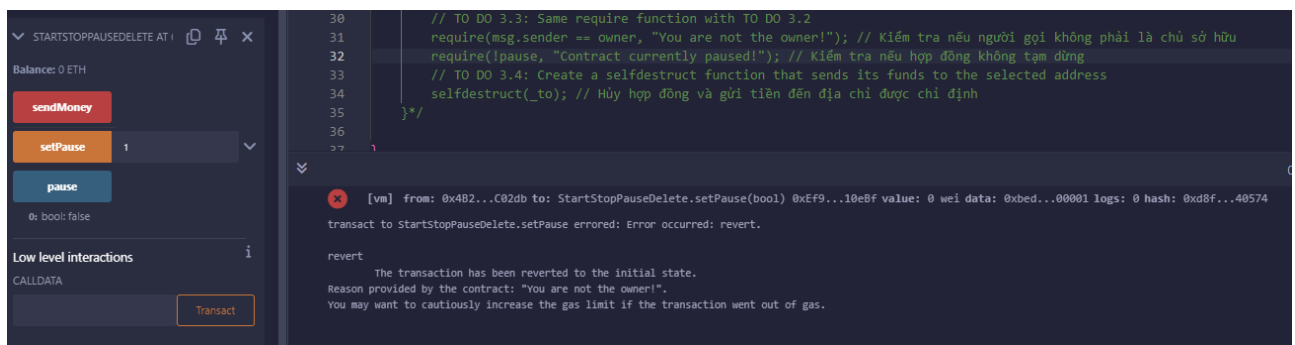
### 3.2 Create require function that, if the 'msg.sender' is not the 'owner', the error message shows "You are not the owner!"

```
// Chỉ chủ sở hữu có thể tạm dừng hợp đồng
function setPause(bool _pause) public {
    // TODO 3.2: Create require function that, if the 'msg.sender' is not the 'owner', the error message shows "You are not the owner!"
    require(msg.sender == owner, "You are not the owner!"); // Kiểm tra nếu người gọi không phải là chủ sở hữu
    pause = _pause; // Đặt trạng thái tạm dừng
}
```

Trường hợp là owner, khi thực hiện gọi hàm setpause



Ngược lại nếu không là owner, khi gọi hàm setpause



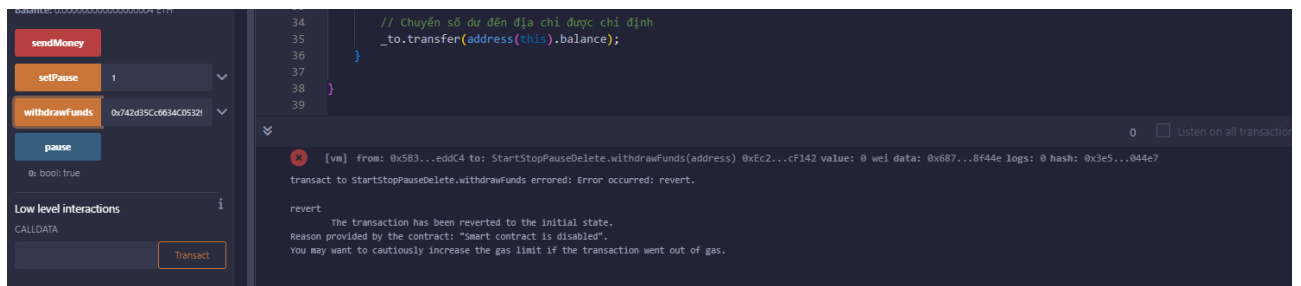
### 3.3 Same require function with TO DO 3.2 require(msg.sender == owner, 'You are not the owner!');

### 3.4 Create a selfdestruct function that sends its funds to the selected address

```
// Rút ether đến địa chỉ được chỉ định
function withdrawFunds(address payable _to) public {
    require(msg.sender == owner, "You are not the owner!"); // Kiểm tra nếu không phải là chủ sở hữu
    require(!pause, "Smart contract is disabled"); // Kiểm tra nếu hợp đồng không bị tạm dừng

    // Chuyển số dư đến địa chỉ được chỉ định
    _to.transfer(address(this).balance);
}
```

Khi pause = 1 => hàm dừng và không thực hiện rút ether



Khi pause = 0 => rút ether nếu người gọi hàm là owner và pause ở trạng thái false.



## Bài tập 4

### 4.1 Create a struct 'Payment' with two uint which are 'amount' and 'timestamp'

```
// Create a struct 'Balance' with two variables and one mapping
struct Balance {
    uint totalBalance;// tổng số tiền
    uint numPayments;// số lần thanh toán
    mapping(uint => Payment) payments;// mapping để lưu chi tiết thanh toán cho từng lần thanh toán
}
```

Nếu như ta muốn lưu trữ mỗi lần thanh toán riêng biệt, ta cần dùng mapping để lưu trữ.

### 4.2 Create a public mapping called 'balanceReceived' with key type address to value type 'Balance'

Tạo một mapping public với kiểu khóa là address và kiểu giá trị là balance

```
// TO DO 4.2: Create a public mapping called 'balanceReceived' with key type address to value type 'Balance'
mapping(address => Balance) public balanceReceived;
```

Function Sendmoney cho phép người dùng gửi ether vào smart contract. Mỗi lần người dùng gửi tiền, thông tin về giao dịch sẽ được ghi lại ( số tiền và thời gian )

```
// Send money from different addresses and create separated 'payment' in every transaction
function sendMoney() public payable {    ⚡ infinite gas
    balanceReceived[msg.sender].totalBalance += msg.value;
    Payment memory payment = Payment(msg.value, block.timestamp); // Using block.timestamp for current time
    balanceReceived[msg.sender].payments[balanceReceived[msg.sender].numPayments] = payment;
    balanceReceived[msg.sender].numPayments++;
}
```

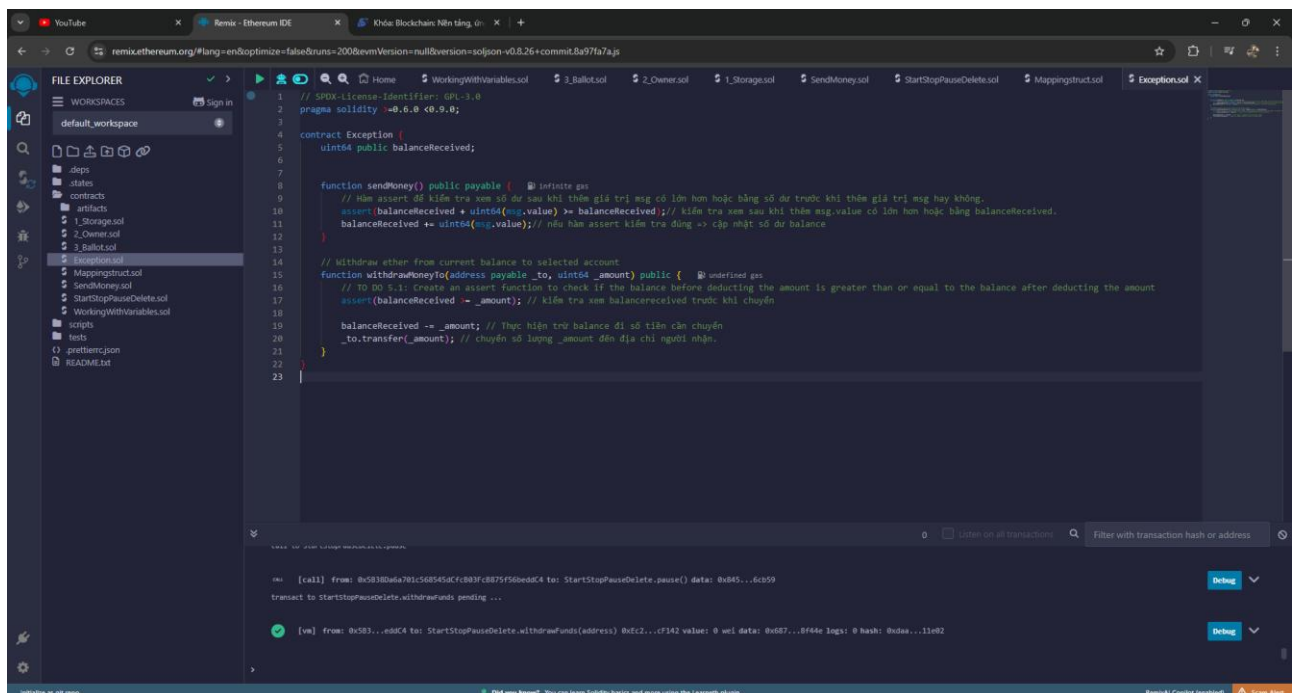
Function WithdrawAllmoney cho phép người dùng rút toàn bộ số tiền họ đã gửi vào smart contract.

```
// Withdraw all money from selected address
function withdrawAllMoney(address payable _to) public {    ⚡ undefined gas
    uint balanceToSend = balanceReceived[msg.sender].totalBalance;
    balanceReceived[msg.sender].totalBalance = 0;
    _to.transfer(balanceToSend);
}
```

### Bài tập 5:

#### 5.1 Create an assert function to check if the balance before deducting the amount is greater than or equal to the balance after deducting the amount

Assert thường được dùng để kiểm tra các điều kiện luôn đúng. Nếu sai thì có thể là lỗi logic hoặc lỗi lập trình. Áp dụng assert vào yêu cầu trên.





Dòng kiểm tra assert: `assert(balanceReceived >= _amount);`

Dòng này kiểm tra xem số dư hiện tại (`balanceReceived`) có đủ để thực hiện việc rút tiền không.

- Nếu số dư (`balanceReceived`) nhỏ hơn số tiền muốn rút (`_amount`), hàm `assert` sẽ gây ra lỗi và làm ngưng thực hiện giao dịch.

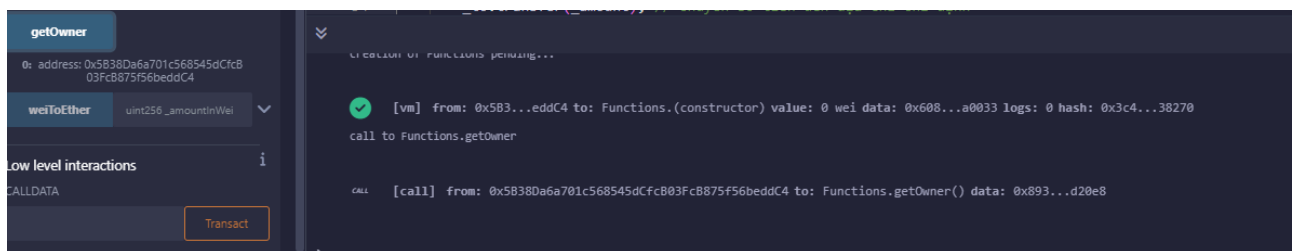
=> Điều này giúp bảo vệ hợp đồng khỏi việc cố gắng rút nhiều tiền hơn số dư hiện có, ngăn ngừa lỗi tài chính không mong muốn.

## Bài tập 6

### 6.1 : Create a public and view function 'getOwner' which returns address, and the return value is 'owner'

```
// TO DO 6.1: Create a public and view function 'getOwner' which returns address, and the return value is 'owner'
function getOwner() public view returns (address) { 2493 gas
    return owner; // Trả về địa chỉ của chủ sở hữu
}
```

sau khi gọi hàm `getOwner`, ta có

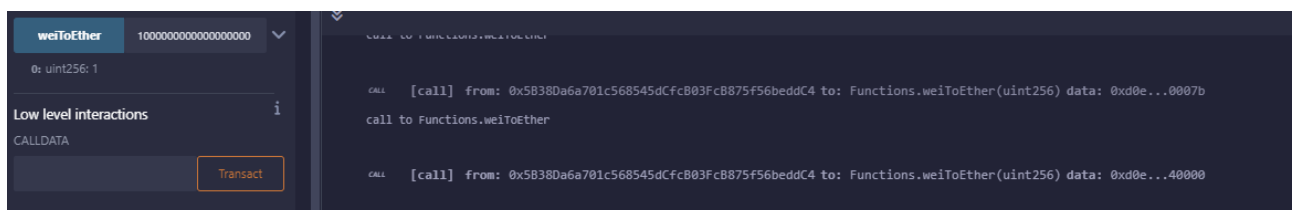


### 6.2: Create a public and pure function 'weiToEther' which uses uint \_amountInWei and returns uint, and the return value is '\_amountInWei / 1 ether'

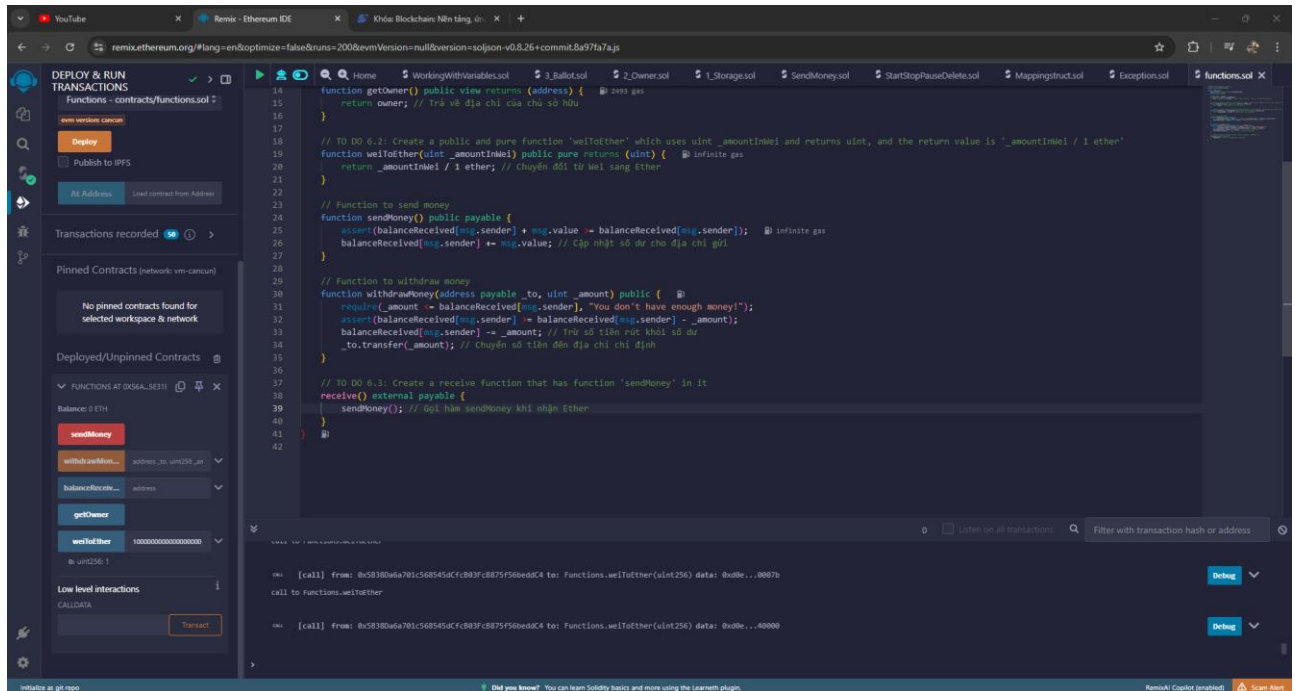
```
// TO DO 6.2: Create a public and pure function 'weiToEther' which uses uint _amountInWei and returns uint, and the return value is '_amountInWei / 1 ether'
function weiToEther(uint _amountInWei) public pure returns (uint) { Infinite gas
    return _amountInWei / 1 ether; // Chuyển đổi từ Wei sang Ether
}
```

Hàm pure chỉ ra rằng nó không đọc hoặc ghi vào trạng thái của hợp đồng. Nó chỉ thực hiện một phép toán và trả về kết quả.

Giả sử ta gọi hàm `weiToEther ( 1000000000000000000 )`, kết quả sẽ là 1 vì 1 ether =  $10^{18}$  wei



### 6.3 Create a receive function that has function 'sendMoney' in it



Hàm sendmoney là một hàm đặc biệt trong solidity, cho phép hợp đồng nhận ether mà không cần một hàm cụ thể nào khác được gọi.

Giả sử một người dùng gửi 1 Ether đến hợp đồng. Khi giao dịch được thực hiện:

- Hợp đồng sẽ gọi hàm nhận.
- Hàm nhận sẽ gọi sendMoney, ghi lại số lượng Ether đã nhận vào trạng thái của hợp đồng.

## Bài tập 7

**7.1 Create a modifier function 'onlyOwner' which has a require function in it. In the require function, we are going to check if msg.sender is the owner.**

```
contract Owned {
    address public owner;

    // Constructor function
    constructor() {
        owner = msg.sender; // Thiết lập người sở hữu hợp đồng là người đã tạo hợp đồng
    }

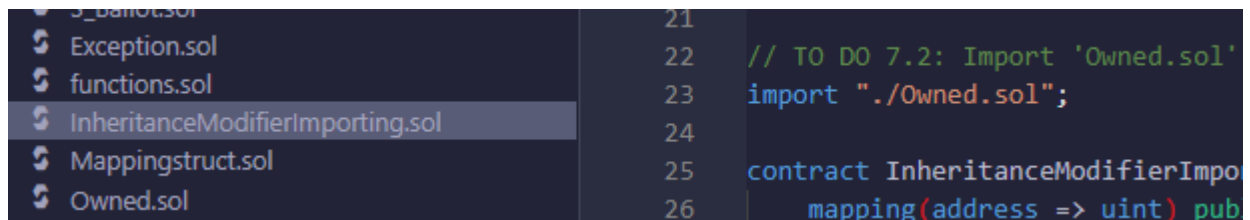
    // TO DO 7.1: Create a modifier function 'onlyOwner' which has a require function in it.
    modifier onlyOwner() {
        require(msg.sender == owner, "You are not the owner"); // Kiểm tra xem msg.sender có phải là owner không
        _; // Tiếp tục thực hiện hàm nếu kiểm tra thành công
    }
}
```

Modifier này sẽ kiểm tra xem địa chỉ người gọi hàm (msg.sender) có phải là chủ sở hữu (owner) của hợp đồng hay không.

Nếu không phải, hàm sẽ không cho phép thực hiện chức năng mà nó điều chỉnh.

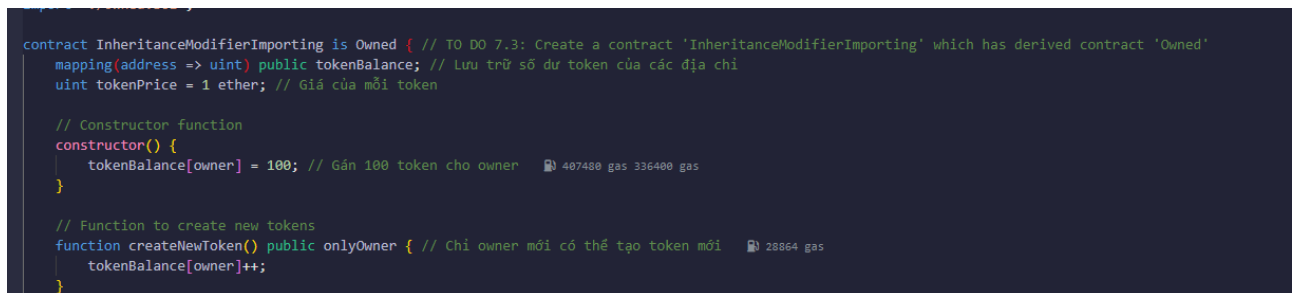
## 7.2 Import owned.sol

Tạo thêm file owned.sol nằm cùng thư mục với file contract hiện tại, import nó vào file hiện tại.



```
21
22 // TO DO 7.2: Import 'Owned.sol'
23 import "./Owned.sol";
24
25 contract InheritanceModifierImporting {
26     mapping(address => uint) public
```

## 7.3 Create a contract 'InheritanceModifierImporting' which has derived contract 'Owned'



```
contract InheritanceModifierImporting is Owned { // TO DO 7.3: Create a contract 'InheritanceModifierImporting' which has derived contract 'Owned'
    mapping(address => uint) public tokenBalance; // Lưu trữ số dư token của các địa chỉ
    uint tokenPrice = 1 ether; // Giá của mỗi token

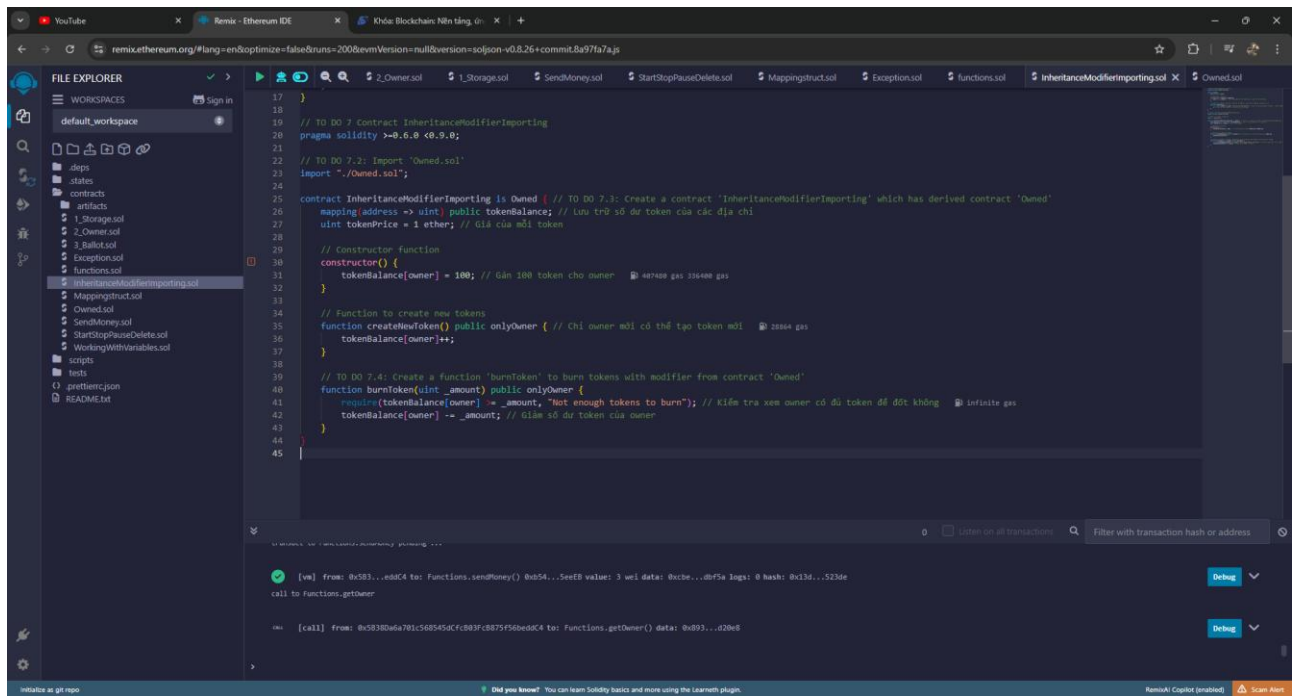
    // Constructor function
    constructor() {
        tokenBalance[owner] = 100; // Gán 100 token cho owner 407480 gas 336400 gas
    }

    // Function to create new tokens
    function createNewToken() public onlyOwner { // Chỉ owner mới có thể tạo token mới 28864 gas
        tokenBalance[owner]++;
    }
}
```

Tạo một hợp đồng mới có tên là InheritanceModifierImporting, hợp đồng này sẽ kế thừa từ hợp đồng Owned.

=> InheritanceModifierImporting sẽ có quyền truy cập vào tất cả các thuộc tính và phương thức công khai (public) và nội bộ (internal) của hợp đồng Owned, bao gồm cả modifier onlyOwner.

## 7.4 Create a function 'burnToken' to burn tokens with modifier from contract 'Owned'



Tạo ra hàm `burnToken` trong hợp đồng `InheritanceModifierImporting`.  
Hàm này sẽ được bảo vệ bằng cách sử dụng modifier từ `onlyOwner` và `Owned`, có nghĩa là chỉ có Owner mới được phép thực hiện hành động này.

## Bài tập 8

### 8.1 Create an event which has 3 variables: `_from`, `_to` and `_amount` event `TokensSent(address _from, address _to, uint _amount);`

Tạo một event với 3 biến `_from`, `_to` và `_amount`

```
// TO DO 8.1: Tạo một event với 3 biến: _from, _to và _amount
event TokensSent(address indexed _from, address indexed _to, uint _amount);
```

**Event** được sử dụng để định nghĩa một sự kiện trong Solidity. Sự kiện là một phần quan trọng của giao thức Ethereum, cho phép các hợp đồng thông minh ghi lại thông tin và phát thông báo về những thay đổi trạng thái.

Các biến trạng thái:

**address \_from:** Biến này chứa địa chỉ của người gửi token (người thực hiện giao dịch).

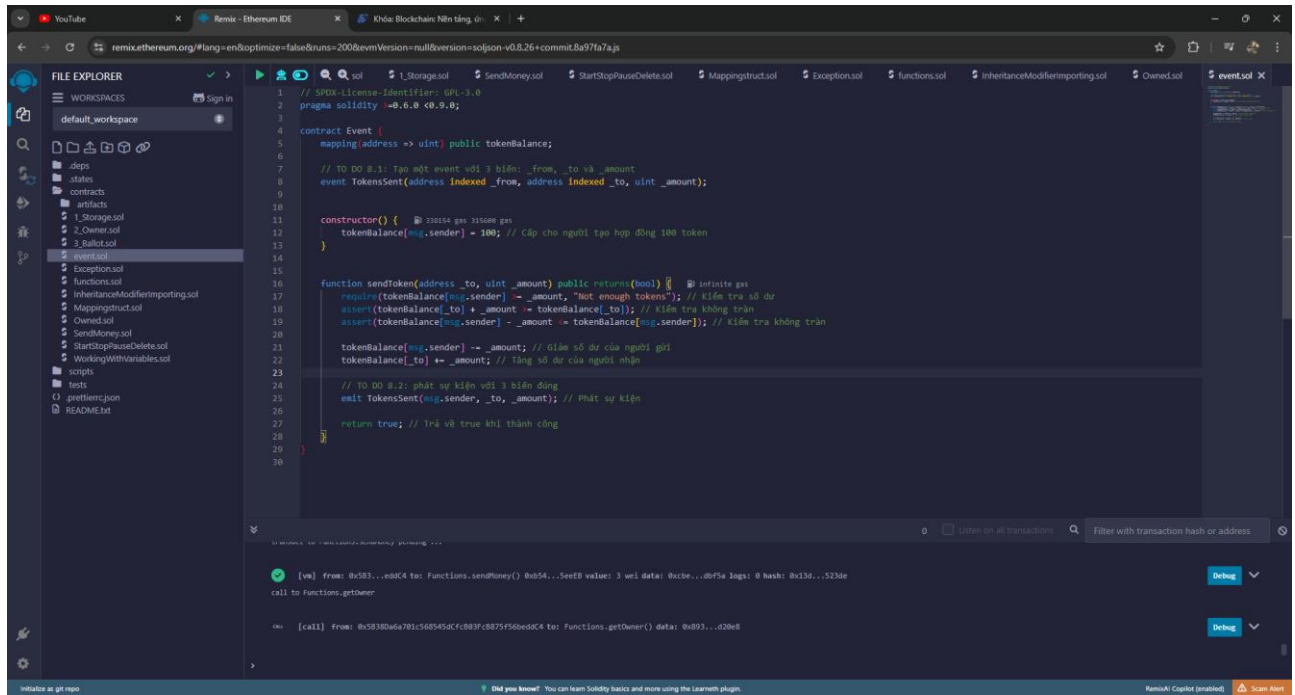
**address \_to:** Biến này chứa địa chỉ của người nhận token.

**uint \_amount:** Biến này chứa số lượng token được chuyển từ người gửi đến người nhận.

## 8.2 emit the event with 3 right variables

Emit được sử dụng để phát ra sự kiện trong solidity. Khi một sự kiện được phát ra, nó sẽ được ghi lại trong nhật ký ( logs ) của giao dịch trên blockchain.

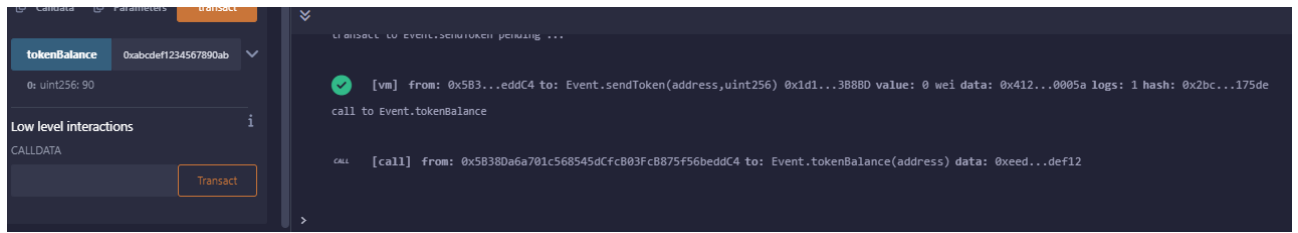
⇒ Theo dõi các sự kiện xảy ra trong smart contract.



Thực hiện gửi token đến địa chỉ ví: 0xabcd1234567890abcdef1234567890abcdef12

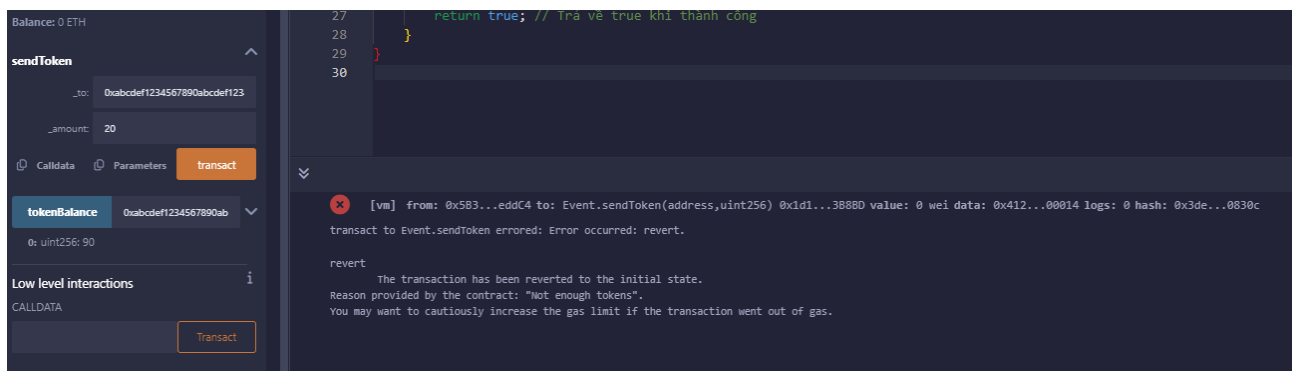


Kiểm tra tokenBalance của địa chỉ trên.



Giá trị là 90 khớp với giá trị đã chuyển từ Owner.

Sau khi chuyển 90, nếu ta thực hiện chuyển thêm 20 thì sẽ không thành công vì số dư ban đầu cung cấp cho owner là 100 token.



**HẾT**