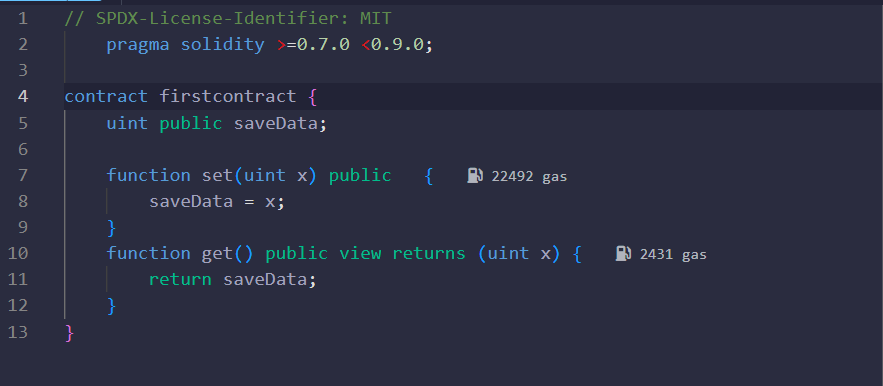
**Học về Solidity**

**Bài 1: Solidity và chương trình cơ bản**

**1.** Solidity là ngôn ngữ lập trình chuyên dụng để phát triển smart contract (hợp đồng thông minh) trên blockchain Ethereum và các mạng tương thích với Ethereum Virtual Machine (EVM). Đây là một ngôn ngữ lập trình bậc cao, hướng đối tượng với cú pháp tương tự JavaScript, C++ và Python, nhưng được thiết kế riêng để xử lý logic blockchain một cách an toàn và hiệu quả.

IDE web : https://remix.ethereum.org/

**2. Chương trình cơ bản (set giá trị và get giá trị):**



**Giải thích code:**

1. **pragma solidity >=0.7.0 <0.9.0;**  
   Dòng này khai báo phiên bản solidity được sử dụng từ phiên bản 0.7.0 đến dưới phiên bản 0.9.0
2. **contract firstcontract {**  
   khai báo, định nghĩa 1 contract (giống như class trông OOP) với tên là “firstcontract”
3. **uint public saveData;**Khai báo 1 biến tên là saveData với kiểu dữ liệu là uint(unsigned interger 256-bit, giá trị nguyên >=0, trong solidity có thể ghi uint8 từ 0 ->255(2^8),… ), public: “**Visibility modifier”** giá trị này được công khai.
4. **function set(uint x) public{  
    saveData = x;  
   }**Tạo hàm set, hàm này cho phép có thể thay đổi giá trị của biến saveData và public thì ai cũng có thể thay đổi giá trị.
5. **function get() public view returns (uint x){  
    return saveData;  
   }**Tạohàm get, hàm này cho phép lấy giá trị của biến saveData hay trả về giá trị của saveData mà không làm thay đổi trạng thái(có thể nói dễ hiểu là giá trị) của contract.

**Giải thích sâu:**

1. Trong Solidity, **function** được định nghĩa bằng từ khóa **function** theo sau là **tên function**, **danh sách tham số, modifier** và **kiểu trả về**. Cú pháp tổng quát là: **function function\_name(parameter\_list) visibility state\_mutability returns(return\_type) {**

**// code logic**

**}**

1. **Return và returns.**

**Vai trò khác nhau hoàn toàn:**

* **returns:** Khai báo function sẽ trả về gì (trong signature)
* **return:** Thực hiện việc trả về giá trị (trong body)

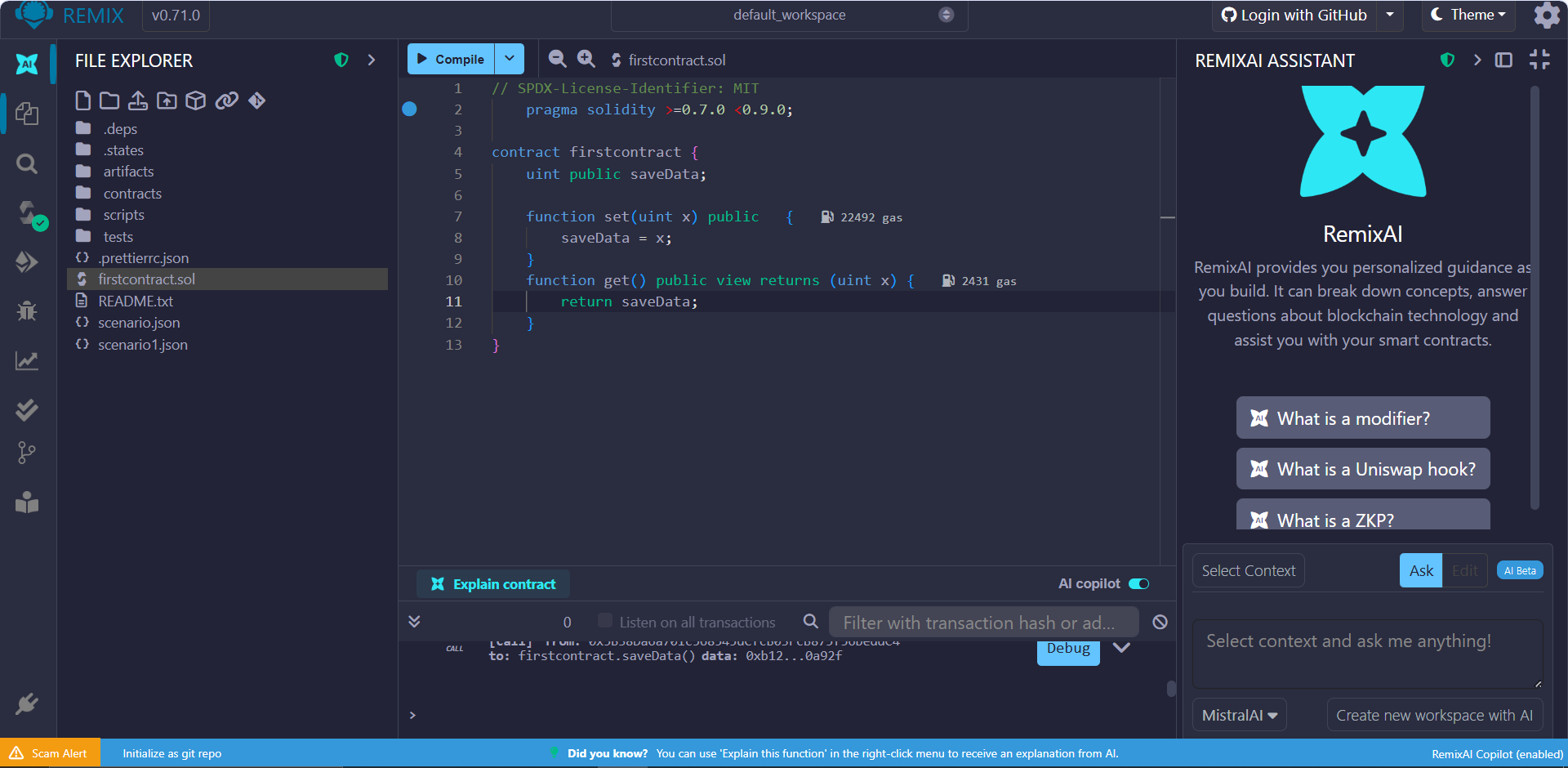
**Vị trí khác nhau**:

* **returns:** Nằm **ngoài** dấu {} của function
* **return:** Nằm **bên trong** dấu {} của function body

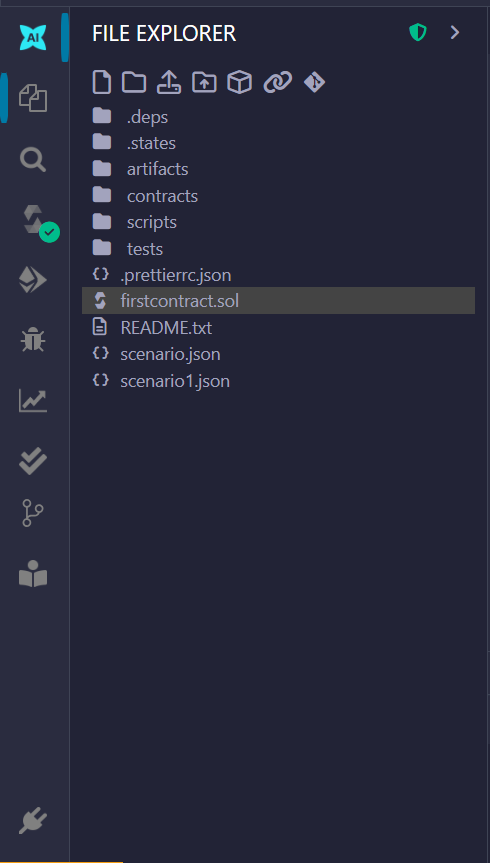
**returns** là **khai báo kiểu** (declaration), còn **return** là **câu lệnh thực thi** (statement). Cả hai đều cần thiết - returns để compiler biết function trả về gì, return để thực sự trả về giá trị đó.

**3. IDE remix.ethereum (Remix IDE)** là 1 **môi trường phát triển tích hợp** và **chạy trực tiếp trên nền tảng trình duyệt web**, chuyên dùng để viết và biên dịch các **smart contract** trên nền tảng **Ethereum**

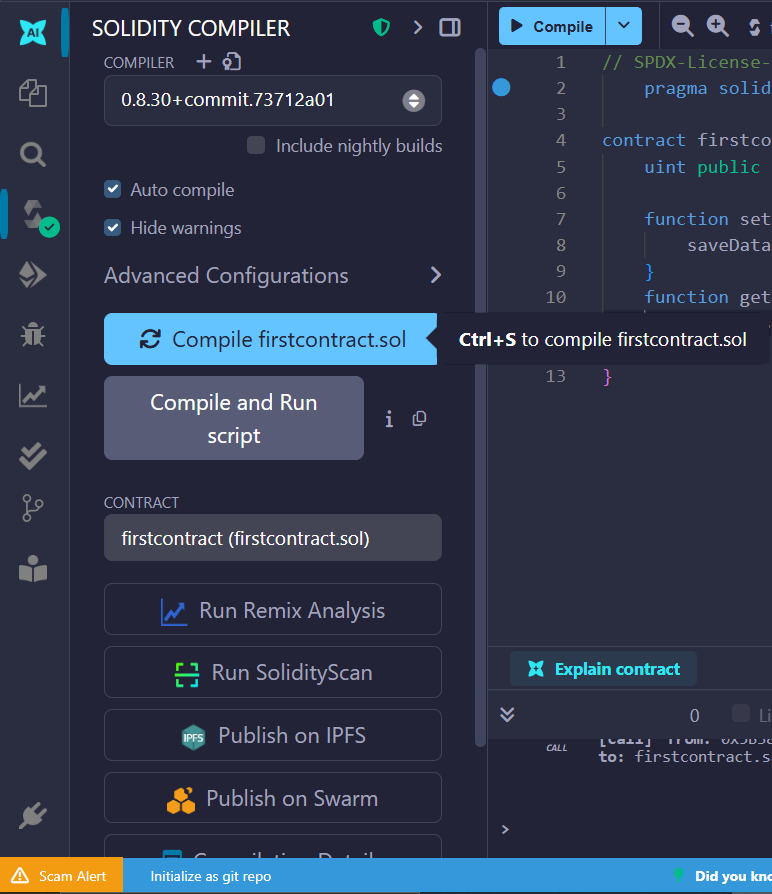
**Giao diện làm việc của IDE:**

****

**Thanh Sidebar:**

****

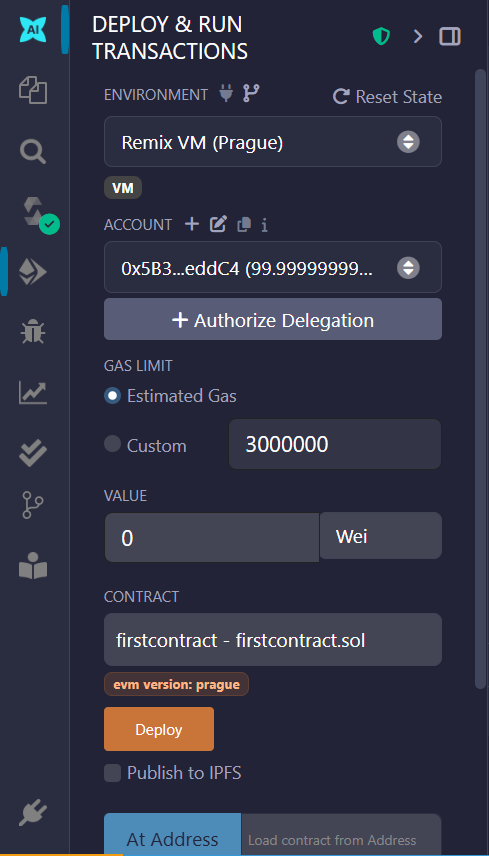
**Compile contract:**

****

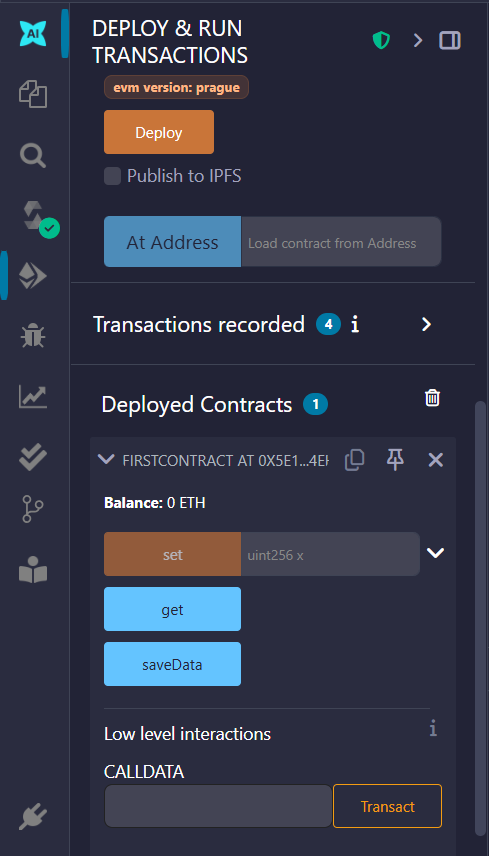
Solidity Compiler có nhiệm vụ **biên dịch** (compile) code Solidity thành bytecode, không liên quan đến việc đẩy lên blockchain. Cụ thể:

* **Input**: Code Solidity (.sol files) mà con người viết
* **Output**: Bytecode (mã máy) mà Ethereum Virtual Machine hiểu được
* **Kèm theo**: ABI (Application Binary Interface) để tương tác với contract

**Deploy & run transactions:**

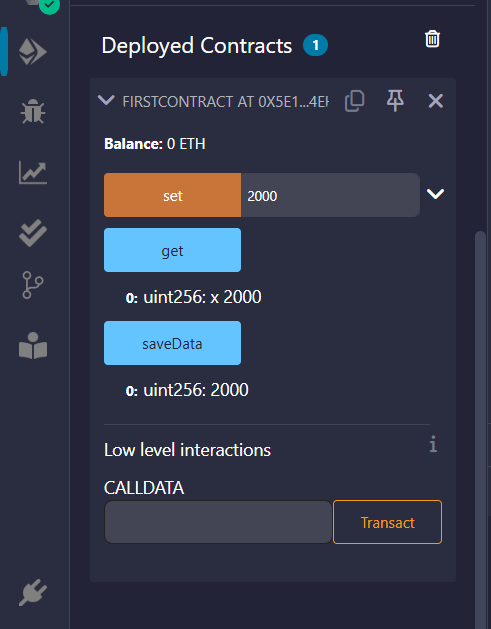
****

Chọn **Environment**(môi trường) và **account**(tài khoản) để triển khai contract lên blockchain, có sẳn hơn **10 tài khoản** với khoản **100ETH cho mỗi tài khoản. Deploy** để gửi **trainsaction** lên blockchain và sẽ nhận **contract address** sau khi deploy thành công**.**



At address(truy cập vào contract đã có). Kết nối với contract đã được deploy lên trước đó bằng address

**Tương tác với Deploy contract:**



**Button xanh(blue):** **View/Pure function** – chỉ đọc, lấy dữ liệu và không tốn **gas**(**tài nguyên** của các smart contract trên Ethereum, giúp vận hành hiệu quả và tránh bị lạm dụng hệ thống).

**Button cam(orange):** **Non-payable functions** – thay đổi **state**(trạng thái), tốn **gas.**

**Button đỏ(red):** **Payable functions** – có thể gửi **ETH** kèm theo.

**Bài 2: Tạo token đầu tiên**

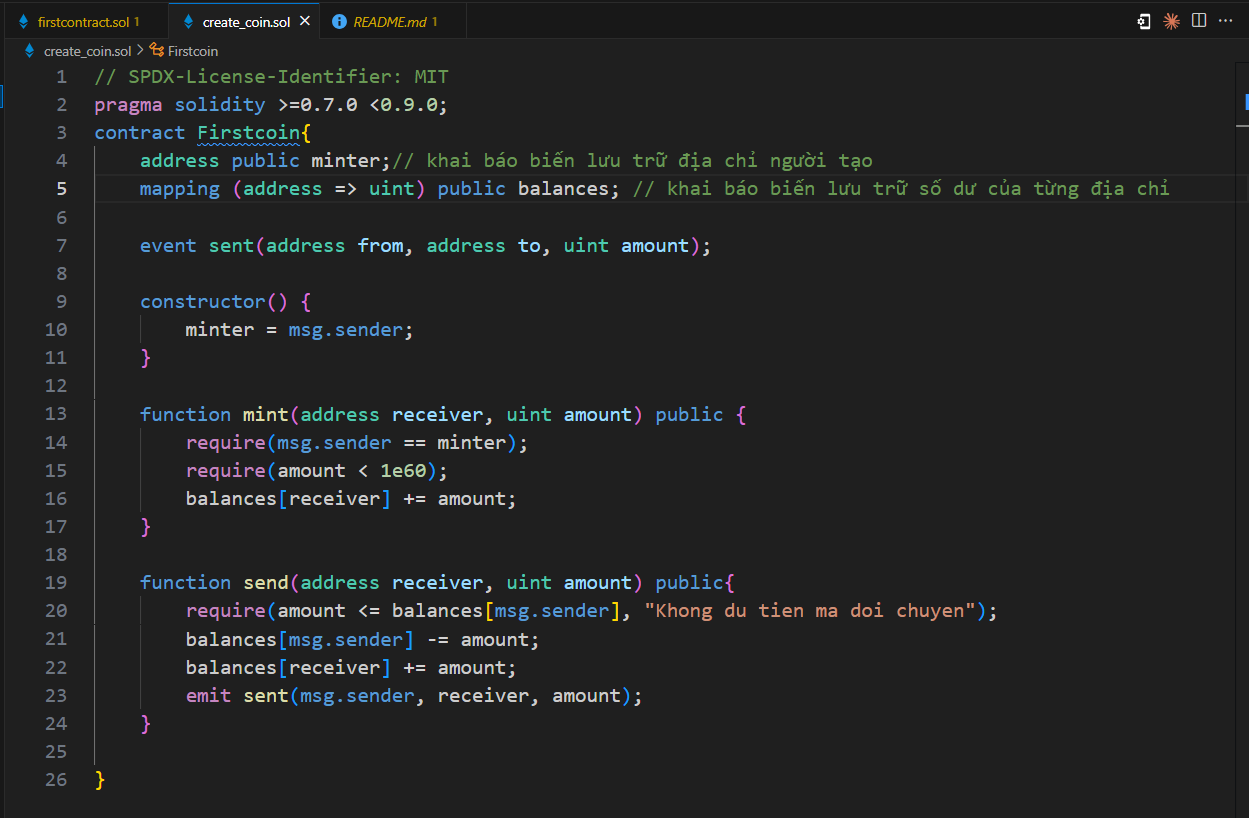
1. Muốn giao dịch token thì cần phải:

1. **Create a token/Coin**
2. **Create amount token/Coin**

* Minter(người tạo)
* Suply(lượng cung)
* Balance(Ví, số dư)

1. **Sent Token**
   * Receiver(Người nhận)
     1. If Amount > Balance then không đủ tiền để chuyển
     2. Balance sender -= amount;
     3. Balance receiver += amount;
   * Amount(Khối lượng)

2. Chương trình Khởi tạo coin, chuyển coin:



**Giải thích code:**

1. **Pragma solidity >=0.7.0 <0.9.0** : Khai báo phiên bản solidity được sử dụng trong chương trình này.
2. **contract Firstcoin{} :** Tạo 1 **contract**(giống class trong OOP) với tên là **Firstcoin**
3. **address public minter;** : khai báo biến **minter** với kiểu dữ liệu là **address**(kiểu dữ liệu riêng của solidity) ở trạng thái **public.**
4. **mapping (address => uint) public balances;** : Khai báo biến **balances** kiểm tra số **uint**(kiểu dữ liệu trả về theo câu lệnh trên) đã được lưu ở địa chỉ với kiểu dữ liệu **mapping**(kiểu dữ liệu ánh xạ)ở trạng thái **public**. Chuyên sâu hơn : sẽ giống hàm

function balances(address \_addr) public view returns (uint) {

return balances[\_addr];

}

1. **event sent( address from, address to, uint amount); :** Khai báo sự kiện sent với các tham số from(địa chỉ gửi), to(địa chỉ nhận) và amount(số lượng sent). Mỗi khi ai đó gửi token thì ta muốn ghi lại **ai gửi, ai nhận, gửi bao nhiêu**.
2. **contructor(){**

**minter = msg.sender;**

**} :**khởi tạo 1 **contructor**, hàm này đặc biệt chỉ chạy **1 lần duy nhất** khi contract được deploy. Ở đây biến **minter** sẽ lưu địa chỉ từ **msg.sender**(biến môi trường trong solidity, ở đây là địa chỉ đã thực hiện hành động deploy contract). Khi chạy thì **minter** sẽ **cố định** cho đến khi bị thay đổi bởi code khác

1. **function mint(address receiver, unit amount) public{**

**require(msg.sender == minter);**

**require(amount <1e60);**

**balances[receiver] += amount;**

**}** : Khởi tạo 1 **function** có tên là **mint** để tạo thêm token/coin mới, có 2 tham số đưa vào đó là **receiver**(địa chỉ của người nhận **amount** coin này) và **amount**(số lượng coin muốn tạo ra), đầu tiên sẽ kiểm tra yêu cầu là người tạo coin phải là deployer của dự án này hay chỉ có thể **minter** mới có quyền dùng **function** này, tiếp theo là đảm bảo **amount**(số lượng) coin tạo ra phải bé hơn **1e60** tránh tình trạng overflow hoặc tạo vô hạn coin, cuối cùng là sẽ cộng thêm vào địa chỉ của người này một giá trị = số **amount**, nếu **receiver** chưa có gì thì mặc định sẽ là **0**.

1. **function sent(address receiver, uint amount) public {**

**require(amount <= balances[msg.sender],”Khong du tien!”);**

**balances[msg.sender] -= amount;**

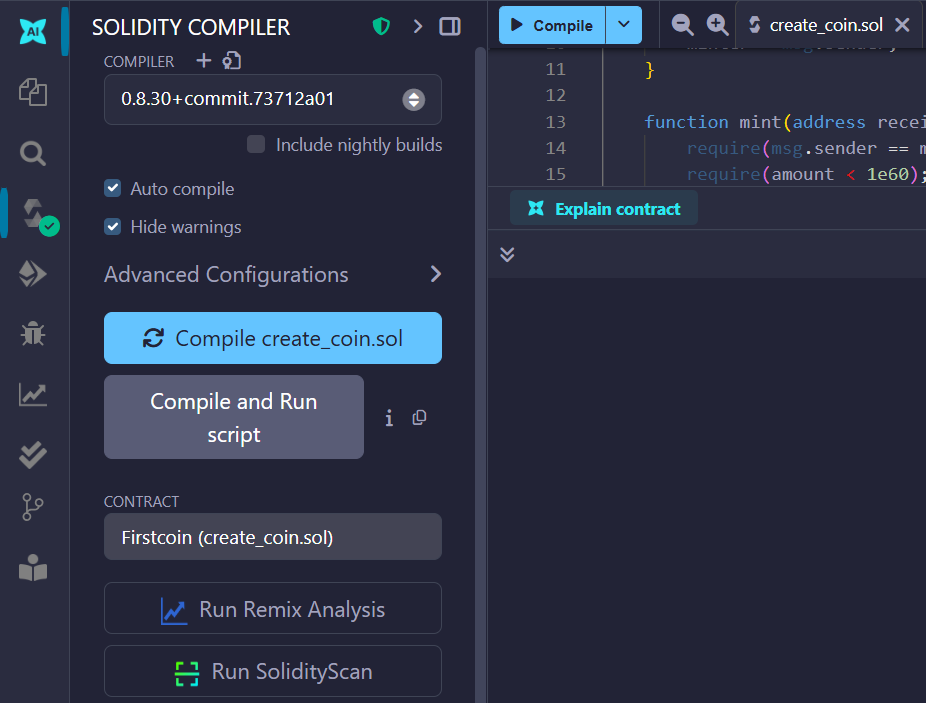
**balances[receiver] += amount;**

**emit sent(msg.sender, receiver, amount);**

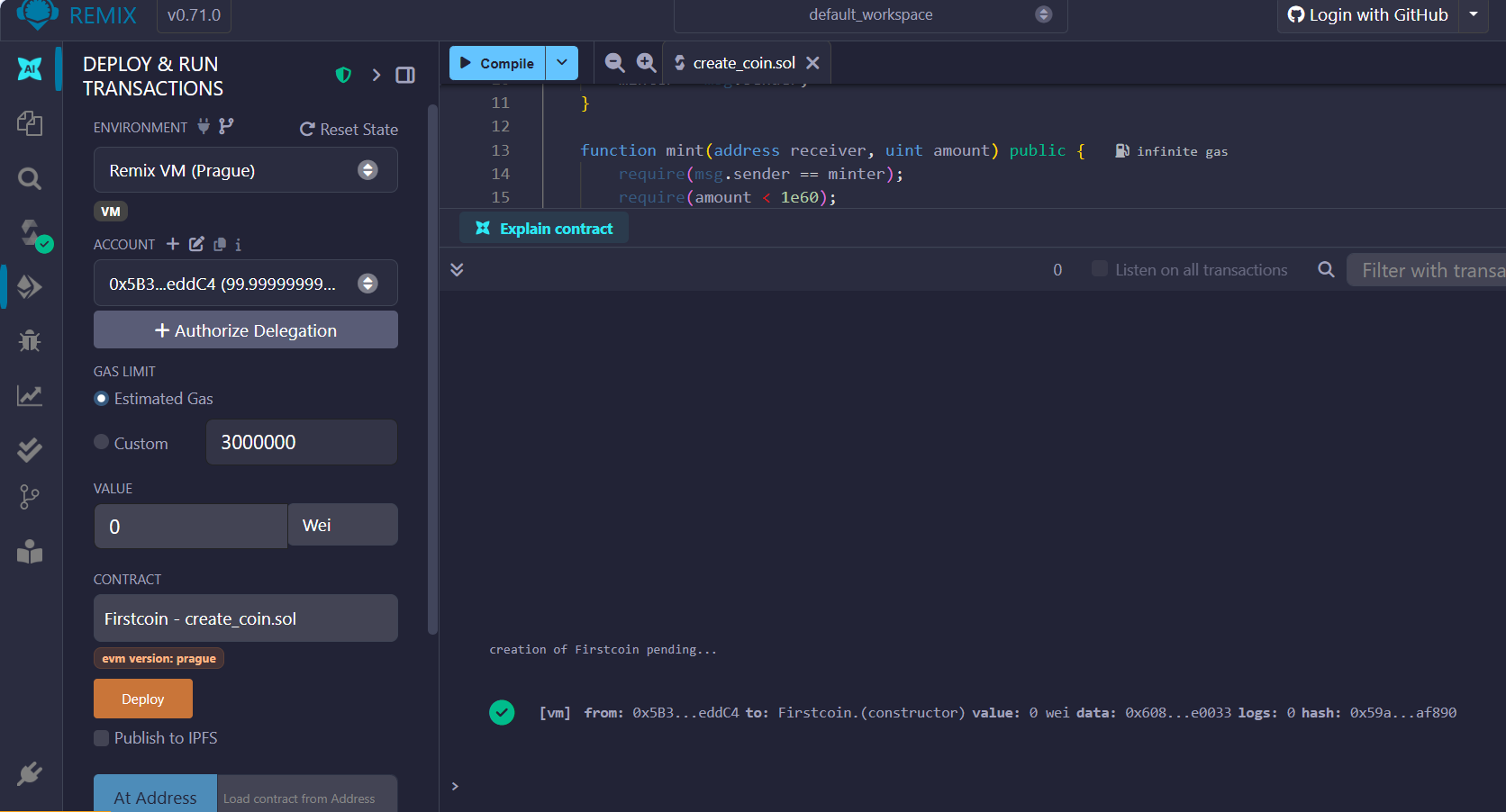
**}:** Khởi tạo 1 **function** có tên là **sent** để gửi token/coin đến địa chỉ của người nhận, có 2 tham số ở đây đấy là **receiver**(địa chỉ người nhận) và **amount**(ở đây là số lượng token/coin được gửi đi) và ở trạng thái **public**. Đầu tiên sẽ kiểm tra **amount**(sô lượng chuyển) phải thấp hơn hoặc bằng số dư trong cái **balances** của người **msg.sender**(ở đây là người gọi hàm), nếu không thõa mãn điều kiện trên sẽ đưa ra dòng “Khong du tien!”. Tiếp theo sẽ là lấy giá trị coin trong ví của người gửi ở đây là **msg.sender** trừ đi **amount** và đồng thời cộng vào cái **balances** của receiver 1 lượng **amount** giống vậy. Cuối cùng sẽ gọi **even** **sent** đã được khai báo từ trước đó. **emit** không chạy code mà chỉ chương trình chạy hay ghi dấu lại cái function đã phát sinh thêm một **event** và đưa nội dung vào **logs** của **transaction**(giao dịch) hay **transaction receipt logs**, ghi chứng 1 **transaction** đã phát sinh **event.**

3. Chạy chương trình trên Remix:

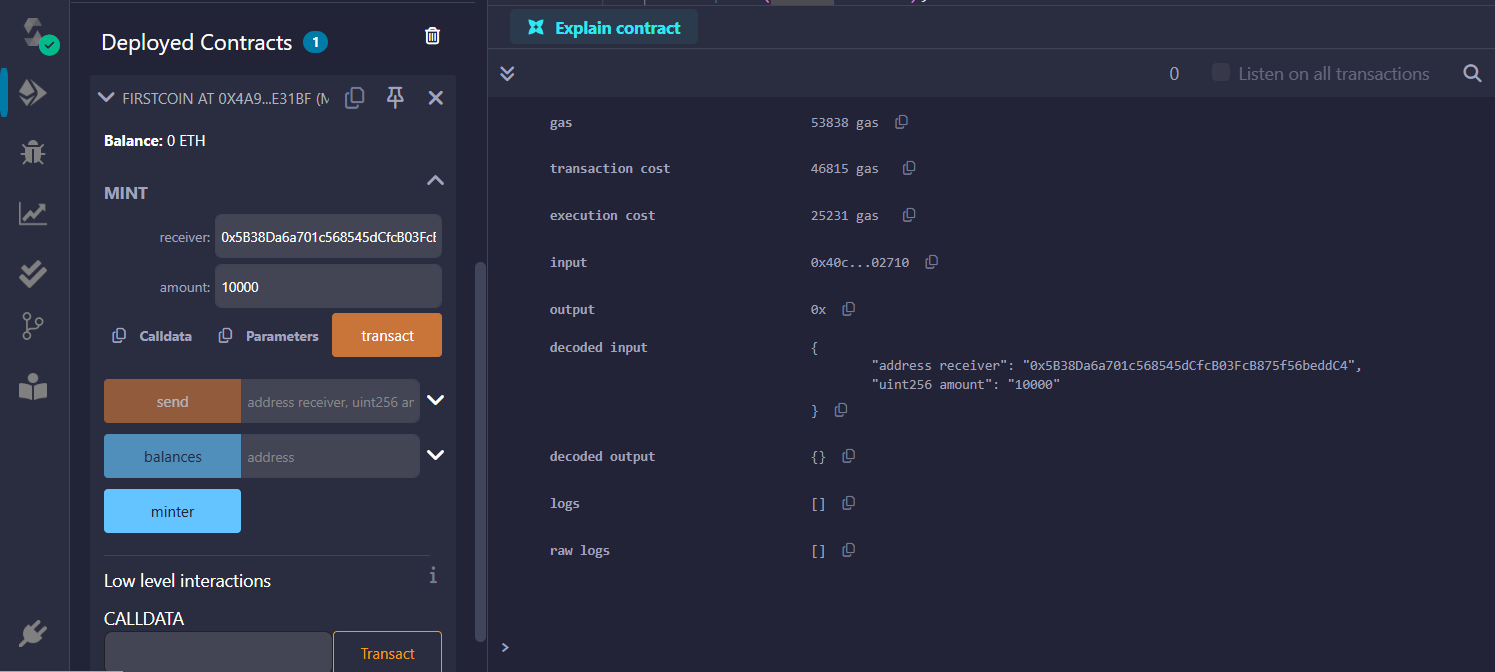
1. **Compile file Solidity**



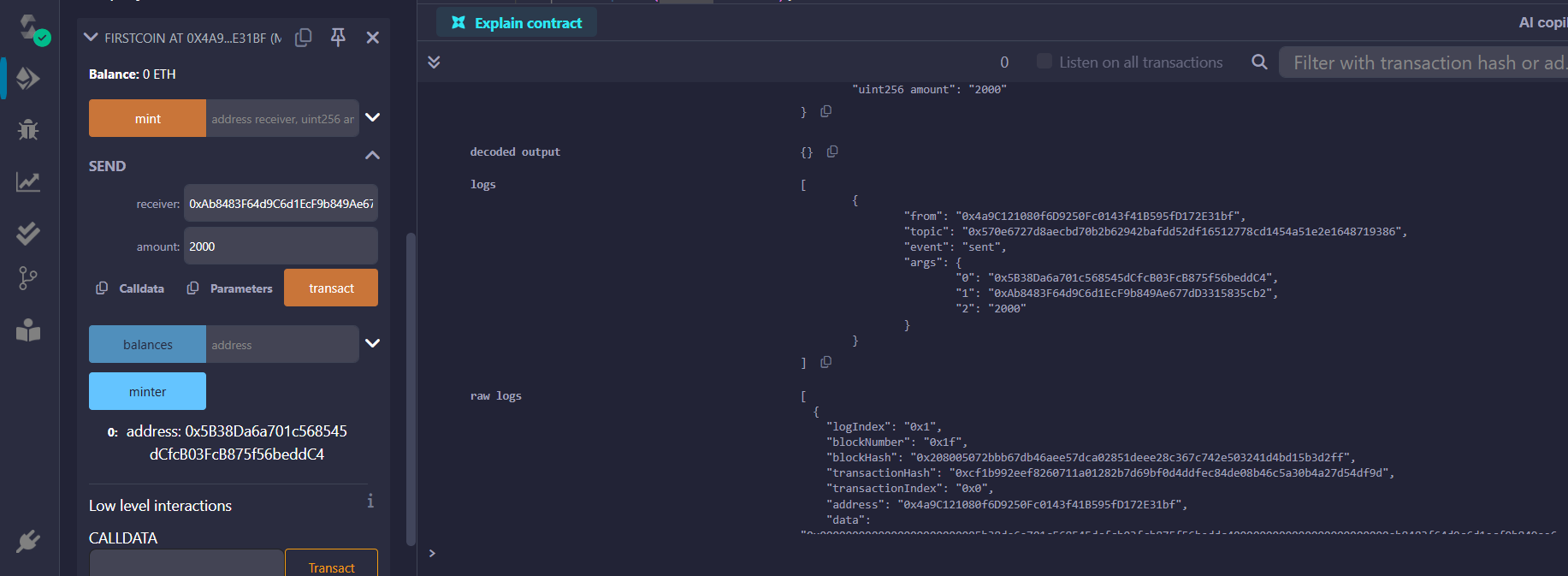
1. **Chọn địa chỉ account để làm địa chỉ minter và deploy smartcontract. Sẽ nhận được trong log trong cửa sổ console**



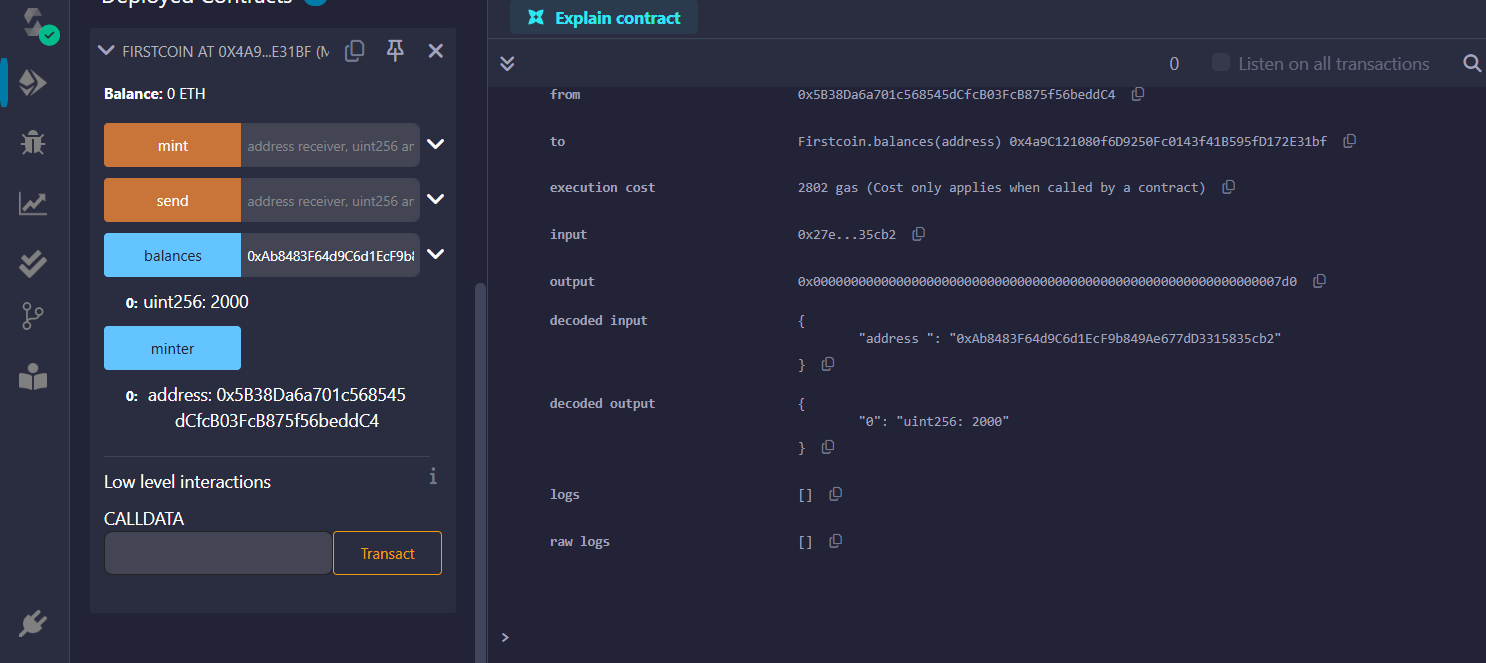
1. **Ở phần deployed contracts chọn vào phần mint để tạo coin. Sẽ ghi địa chỉ receiver(ví người nhận) và amount(số lượng coin muốn tạo) và chọn transact để thực hiện. Logs sẽ được ghi lại và hiện ở cửa sỗ console.**



1. **Tiêp theo là phần send sẽ chọn receiver người nhận và amount(số lượng coin trong giao dịch) và chọn transact để thực hiện. thông tin sẽ được hiện ở cửa sổ console và có thêm event.**

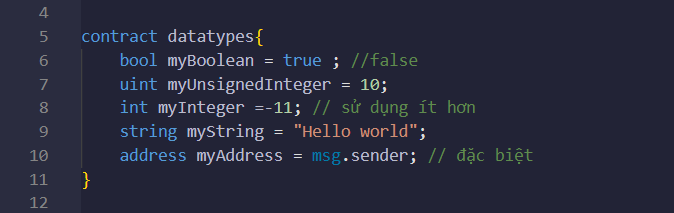


1. **Phần balances sẽ kiểm tra số dư hay giá trị uint trong địa chỉ được nhập. Và minter sẽ trả về địa chỉ của người đã deploy cái contract này**



**Bài 3: Các kiểu dữ liệu trong Solidity**

1. **Các kiểu dữ liệu phổ biến**

****

* **bool:** kiểu dữ liệu đúng sai (true/false), điều chỉnh logic
* **uint:** kiểu số nguyên không âm >= 0
* **int:** kiểu số nguyên
* **string:** kiểu chuỗi ký tự dùng để lưu text
* **address:** địa chỉ Ethereum, kiểu đặc biệt của Solidity dùng để lưu trữ địa chỉ của tài khoản trong hợp đồng

1. **Code ví dụ sử dụng các data types**



Giải thích code:

1. Đầu tiên khởi tạo 1 contract có tên là Game
2. Khai báo biến countPlayer để đếm số lượng có trong game
3. mapping sẽ ánh xạ dữ liệu từ address của player có thể lấy được tất cả các thông tin có trong player đó và khai báo cho biến playes
4. enum sẽ khai báo biến Level có 3 giá trị là Beginner, Intermediate và Avande
5. dùng struct để tạo 1 đối tượng là Player có 5 thông tin là addressPlayer, fullName, myLevel, age và sex.
6. function addPlayer(), hàm này dùng để thêm mới 1 player vào trong chương trình. Sẽ lấy địa chỉ đang gọi hàm làm địa chỉ cho player đó thông qua msg.sender và level được gán ban đầu là Beginner. Biến đếm countPlayer sẽ được cộng thêm 1.
7. function getPlayerLevel(), hàm này là hàm get/call(lấy giá trị) giá trị nên sẽ có returns về kiểu giá trị trả về. Hàm này cần tham số addressPlayer để xác định player cần xem và chỉ lấy thuộc tính myLevel từ Player. Hàm này chỉ lấy 1 giá trị thay vì lấy toàn bộ thuộc tính của Player. Trả về giá trị theo Index của biến đã được khai báo bằng enum ở trên.