



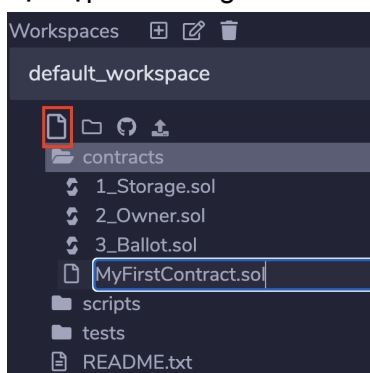
Bootcamp lập trình Smart Contract: Bài tập về nhà Buổi 1

Bài tập 1: Tạo Smart Contract đầu tiên	2
Bài tập:	11
Bài tập 2: Array/Mảng và Map/Ánh Xạ	12
Cách dùng Array trong smart contract	12
Cách dùng Map trong smart contract	17
Bài tập:	23
Bài tập 3: Nguồn cấp giá	26
Bài tập:	32
Bài tập 4: Any-API	33
Bài tập:	39
Bài tập 5: VRF	40
Bài tập:	46

Bài tập 1: Tạo Smart Contract

Trong bài tập này, bạn sẽ tạo một Smart Contract đơn giản để lưu trữ và truy xuất một giá trị, sau đó bạn sẽ triển khai nó vào mạng thử nghiệm Goerli và tương tác với nó bằng cách sử dụng Remix.

1. Mở <http://remix.ethereum.org/>
2. Nếu remix yêu cầu hãy lựa chọn “default workspace”
3. Tạo tệp mới trong contracts folder. Đặt tên là MyFirstContract.sol



4. Chọn phiên bản compiler:

```
pragma solidity ^0.8.7;
```

5. Bên dưới, nhập code của smart contract. Nếu như remix báo “**SPDX Identifier warning**” thì cứ tiếp tục, đó là warning không phải là chương trình báo lỗi.

```
pragma solidity ^0.8.7;

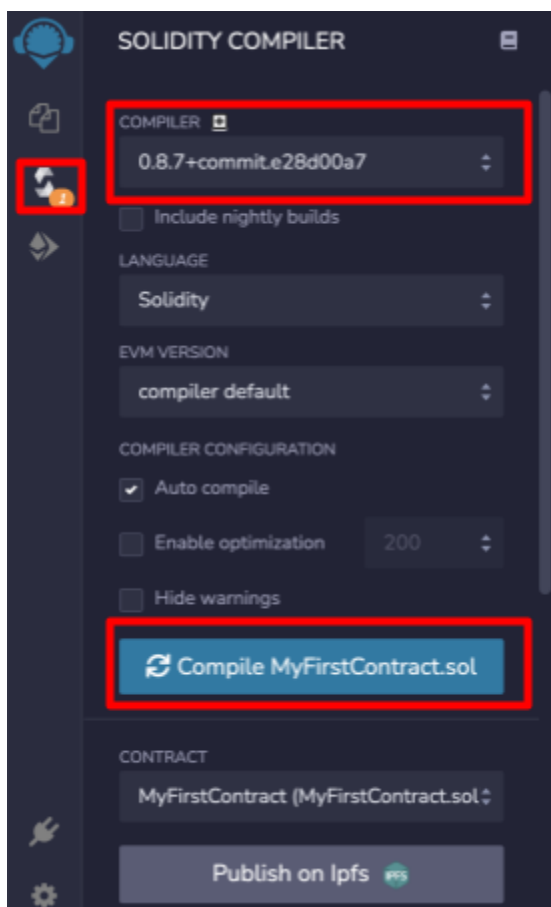
contract MyFirstContract {

    uint256 number;

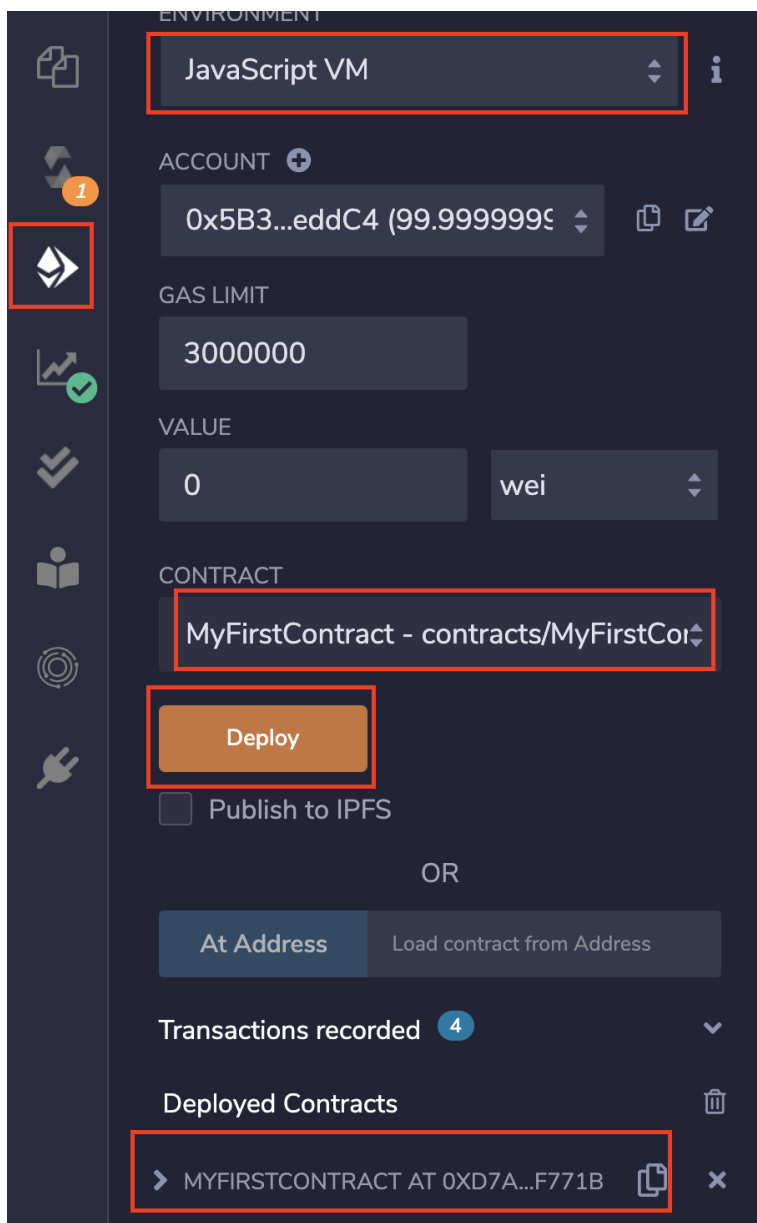
    function changeNumber(uint256 _num) public {
        number = _num;
    }

    function getNumber() public view returns (uint256){
        return number;
    }
}
```

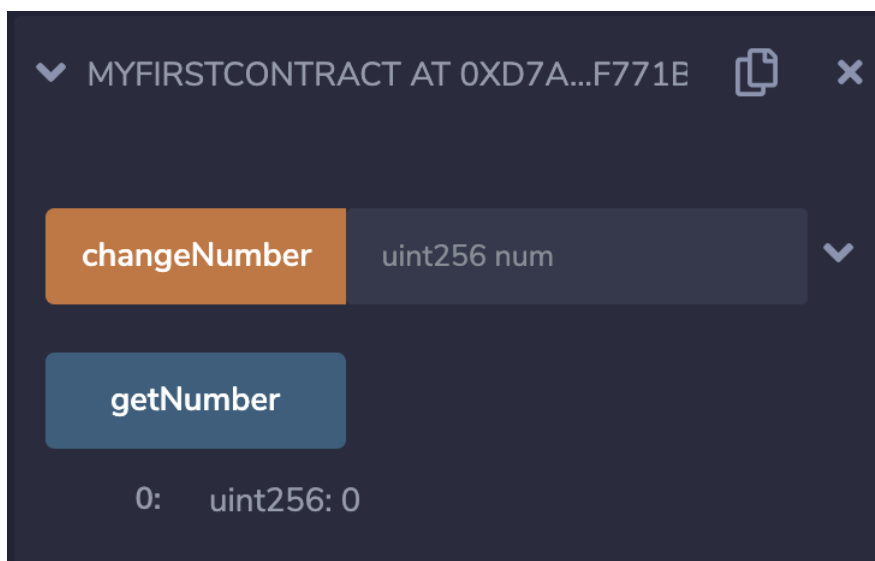
6. Hợp đồng của bạn đã sẵn sàng. Nhấn vào tùy chọn **Solidity Compiler** trên menu bên trái, thay đổi version của **Compiler** trong trình đơn thả xuống của trình biên dịch để nó khớp với trình biên dịch được chỉ định trong mã của bạn, sau đó nhấn vào nút **Compile** màu xanh lam:



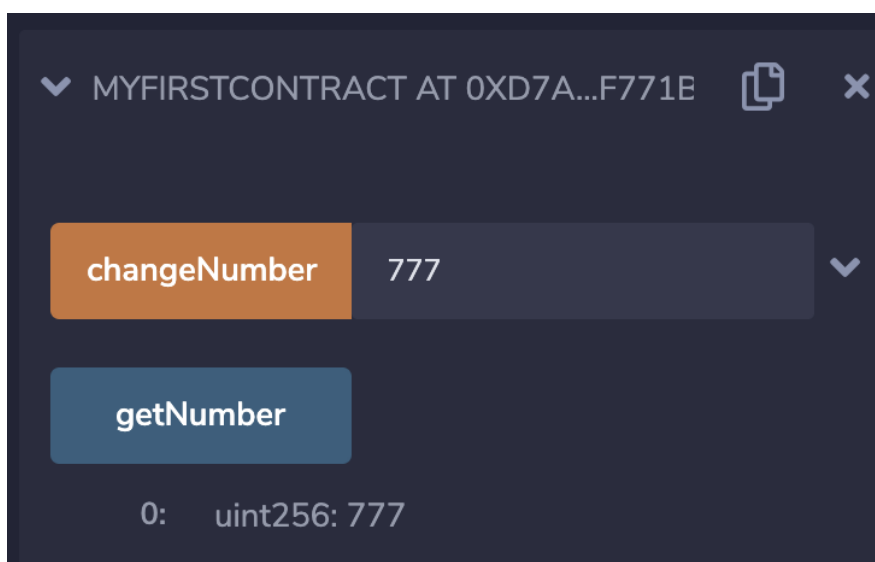
- Click vào '**Deploy and Run Transactions**' trên menu bên trái. Đảm bảo **Environment** được đặt thành '**JavaScript VM**' và **Contract** đã chọn là '**MyFirstContract**', sau đó nhấn nút **Deploy**. Sau đó, bạn sẽ thấy hợp đồng của mình xuất hiện trong phần '**Deployed Contracts**' bên dưới



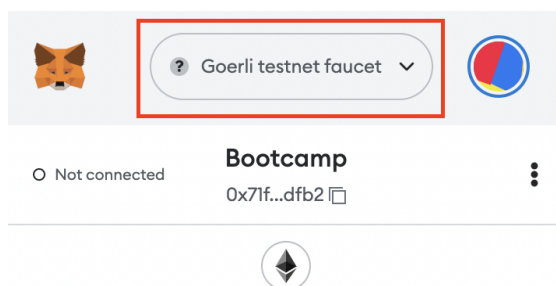
8. Expand hợp đồng đã triển khai bằng cách chọn dấu > bên cạnh hợp đồng đó. Bây giờ bạn có thể thấy tất cả các chức năng mà bạn có thể sử dụng để tương tác với hợp đồng.
9. Nhấn hàm '**getNumber**' để đọc trạng thái của hợp đồng và trả về giá trị của biến số. Bởi vì chúng ta chưa đặt nó, nên nó sẽ trả về 0



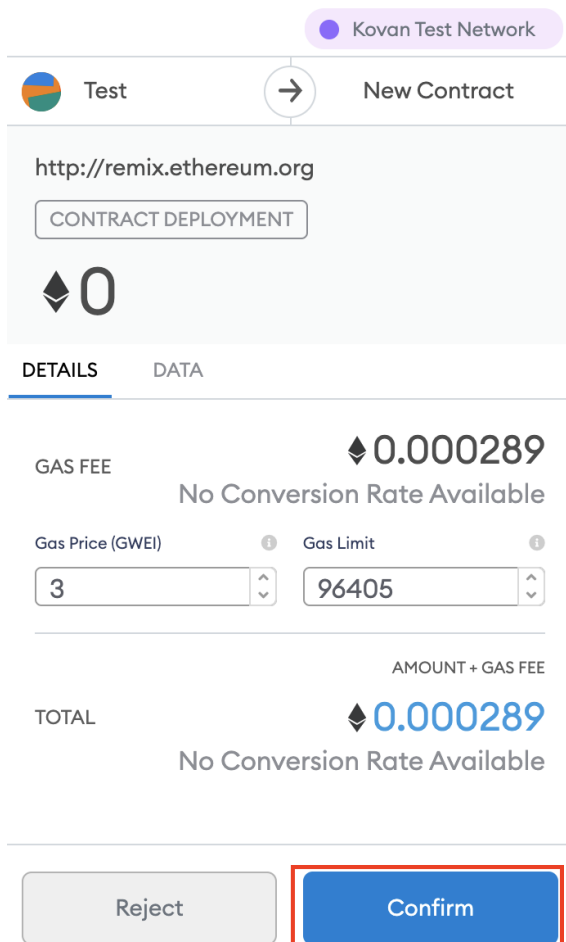
10. Nhập một giá trị giá trị số nguyên vào ô '**changeNumber**' (số nguyên uint256), sau đó nhấn nút '**changeNumber**' để thực thi chức năng.
11. Thực hiện lại chức năng '**getNumber**'. Bạn sẽ thấy contract state đã thay đổi so với lúc trước.



12. Tiếp theo, thay vì làm việc trên mạng máy ảo của **Remix**, chúng ta sẽ deploy contract của mình cho **Goerli testnet**. Thay đổi '**Environment**' thành '**Injected Web3**' như hình phía dưới. Nếu bạn chưa đăng nhập vào ví **Metamask** của mình, thì hãy đăng nhập trước khi tiếp tục và đảm bảo mạng được chọn là '**Goerli Test Network**' đồng thời bạn có một số **ETH** trong ví của mình. Nếu bạn không có bất kỳ **ETH** nào, hãy đặt câu hỏi và sẽ có người giúp đỡ bạn.



13. Bởi vì chúng ta đã compile smart contract rồi nên chúng ta không cần phải làm lại lần nữa. Nhấn nút **Deploy** màu cam để deploy smart contract sau khi đã được compile. **Metamask** sẽ bật lên yêu cầu bạn xác nhận giao dịch. Nhấn nút **'Confirm'** màu xanh lam để thực hiện giao dịch và triển khai hợp đồng của bạn vào mạng **Goerli**. Sau vài giây, bạn sẽ thấy hợp đồng đã triển khai của mình trong phần **'Deployed Contracts'**.



14. Nhấn vào biểu tượng **'copy'** tại contract address để copy address, sau đó truy cập <https://goerli.etherscan.io/>, đây là nơi bạn có thể tìm thấy hợp đồng đã deploy của mình trên mạng **Goerli** và mọi transactions liên quan với nó. Paste smart contract address

của bạn vào hộp tìm kiếm, sau đó nhấn search. Bạn nên tìm hợp đồng đã triển khai của mình, với 1 transaction (khởi tạo smart contract).



Contract 0x94e4531CAAa39E91779345f86aA39C4890CC1611

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: 0xf7b4ef69e7cf13c2055... at txn 0x12c2e628586cde181a...

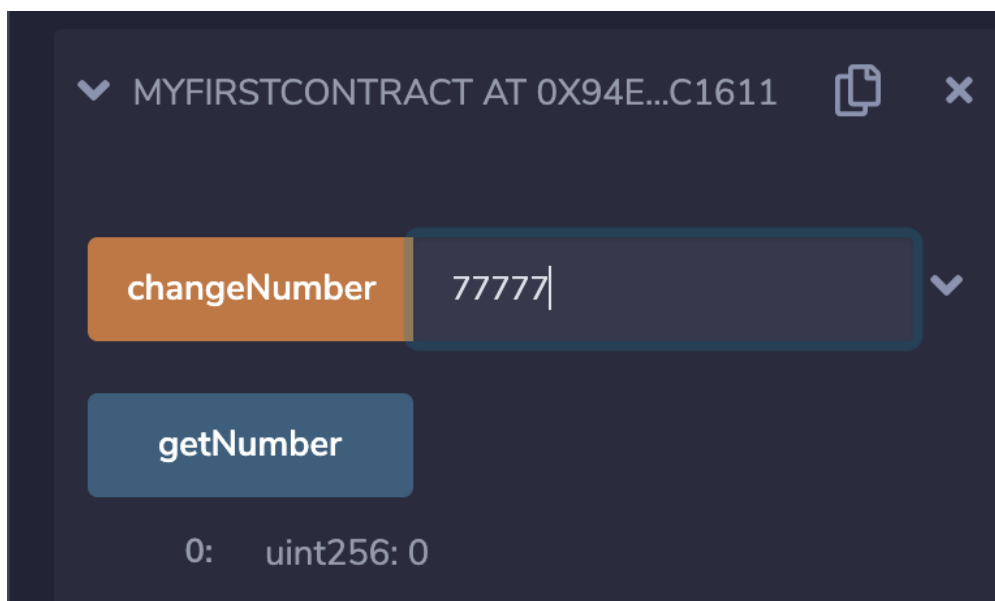
Transactions | Contract | Events

Latest 1 from a total of 1 transactions

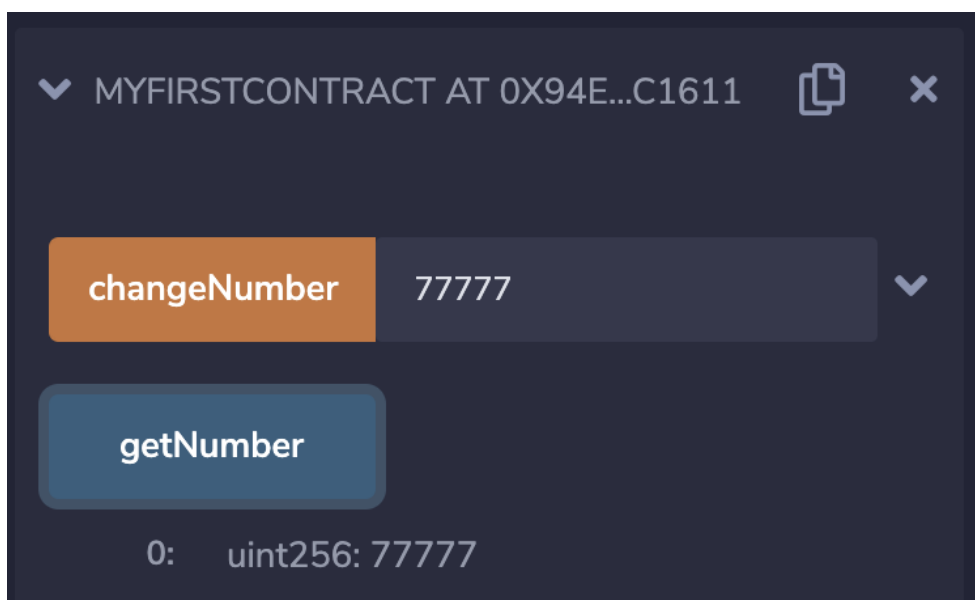
Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x12c2e628586cde181a...	0x60806040	25233021	47 secs ago	0xf7b4ef69e7cf13c2055...	IN Contract Creation	0 Ether	0.000289215

[Download CSV Export]





- Thực thi hàm **'getNumber'** và xác minh rằng kết quả trả về là 0. Sau đó, nhập một số vào tham số hàm **'changeNumber'** và thực thi hàm **'changeNumber'** để cập nhật state của smart contract. **Metamask** sẽ bật lên và yêu cầu bạn xác nhận giao dịch.



18. Chờ vài giây để giao dịch được đưa vào một **Block**, sau đó thực hiện lại chức năng **'getNumber'**. Bây giờ bạn sẽ thấy state của smart contract đã được cập nhật



19. Nếu bạn quay lại **Etherscan** và refresh trang hợp đồng của mình, bạn sẽ thấy giao dịch mới của mình. Nếu bạn nhấn vào **'Txn Hash'**, bạn sẽ thấy thông tin của giao dịch.

Transactions						
Latest 2 from a total of 2 transactions						
Txn Hash	Method ⓘ	Block	Age	From ⌵	To ⌵	
 0xd565669704d29fc079...	0x07391dd6	25233140	1 min ago	0xf7b4ef69e7cf13c2055...	IN	 0x94e4531caaa39e9177...
 0x12c2e628586cde181a...	0x60806040	25233021	9 mins ago	0xf7b4ef69e7cf13c2055...	IN	 Contract Creation

Transaction Details

[illegible]

Xin chúc mừng, bạn đã tạo thành công **Smart Contract** đầu tiên, với chức năng là read và write vào **blockchain**, đồng thời deploy nó vào **live network**!

Bài tập:

Bạn có thể cố gắng hoàn thành các bài tập này nếu bạn đã hoàn thành bài tập chính trước thời hạn:

1. Sửa đổi **Smart Contract** của bạn để thay vì lưu trữ số được truyền vào, nó sẽ tăng **number** được lưu trữ hiện tại bằng tham số **_num** được truyền vào. Để làm điều này, ban đầu bạn nên đặt trạng thái của biến số là 0

```
uint256 number = 0;
```

Bạn có thể tăng biến **number** được lưu trữ theo cú pháp sau

```
number = number + _num;
```

2. Tạo một hàm mới gọi là **'getNumberMultiplied'**, nhận tham số có tên **_num** và sau đó trả về một số nguyên là kết quả của phép nhân tham số **_num** với tham số **number** hiện được lưu trữ. Hàm này phải được định nghĩa là một hàm tương tự như hàm **getNumber()**, bởi vì chúng ta không sửa đổi trạng thái của hợp đồng.
3. Tạo một hàm mới gọi là **'addNumbers'** nhận hai tham số **uint**, **_num1** và **_num2**, sau đó lưu trữ kết quả của việc cộng hai số vào biến số.

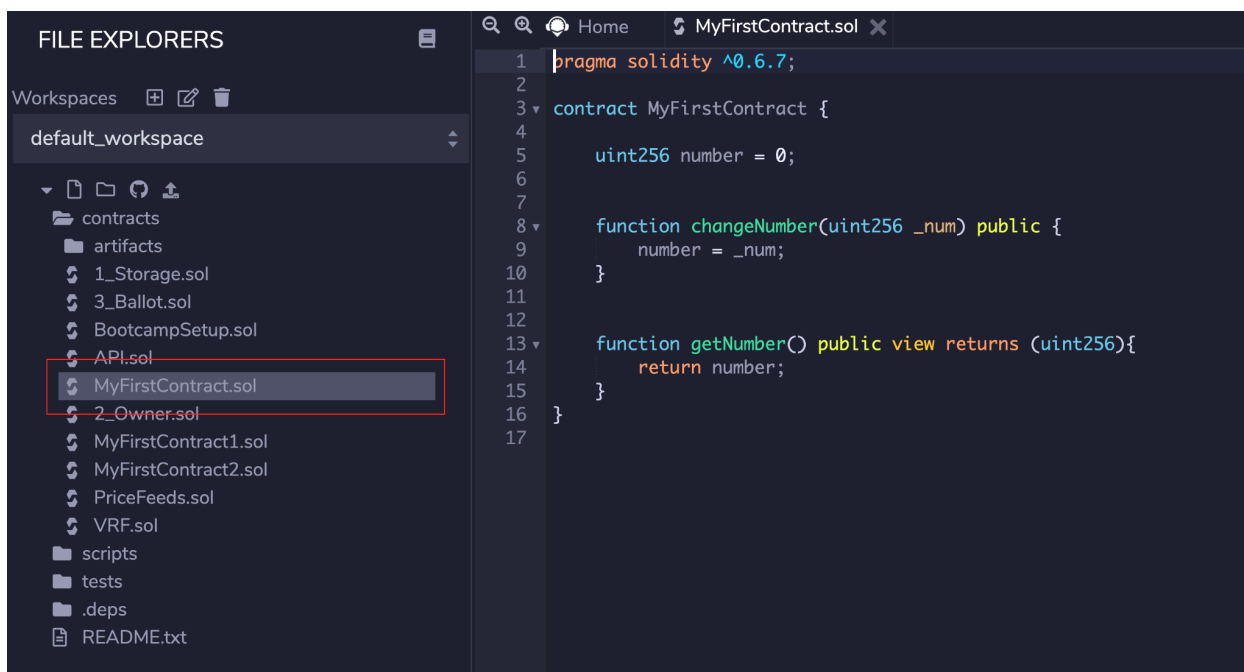
Bài tập 2: Mảng và ánh xạ

In this exercise, we're going to modify our MyFirstSmartContract smart contract and we're going to add an array and a Mapping to it! If you have extra time after completing exercise 1 feel free to complete these as well!

Thêm một mảng

Đối với phần này của bài tập, bạn sẽ tạo một mảng động **names** trong **Smart Contract** của mình, được lưu trữ dưới dạng chuỗi.

1. Mở [Remix](#), bạn sẽ có thể tìm thấy **Smart Contract** đã hoàn thành của mình từ phiên cuối cùng trong trình **Explorers** của bạn



2. Tiếp theo, bạn cần thêm mảng mới vào **Smart Contract** của mình. Tạo một mảng động kiểu **strings** mới được gọi là **'names'**. Bạn có thể thêm nó dưới biến **'number'** hiện có.

```
string[] names;
```

3. Thêm một chức năng để 'push' các giá trị mới vào mảng. Nó sẽ nhận một tham số gọi là **_name**

```
function addName(string memory _name) public {
    names.push(_name);
}
```

4. Thêm một hàm khác để lấy **name** được lưu trữ trong mảng, được cung cấp một tham số **index** được truyền vào. Hàm này sẽ trả về **name** trong mảng tại **index** đã cho. Bởi vì bạn đang đọc trạng thái hợp đồng, đây có thể là một chức năng **view**.

```
function getName(uint _index) public view returns (string memory) {
    return names[_index];
}
```

5. Source Code đầy đủ của bạn bây giờ sẽ trông như thế này

```
pragma solidity ^0.8.7;
```

```

contract MyFirstContract {

    uint256 number = 0;

    //Dynamic array (variable size)
    string[] names;

    function addName(string memory _name) public {
        names.push(_name);
    }

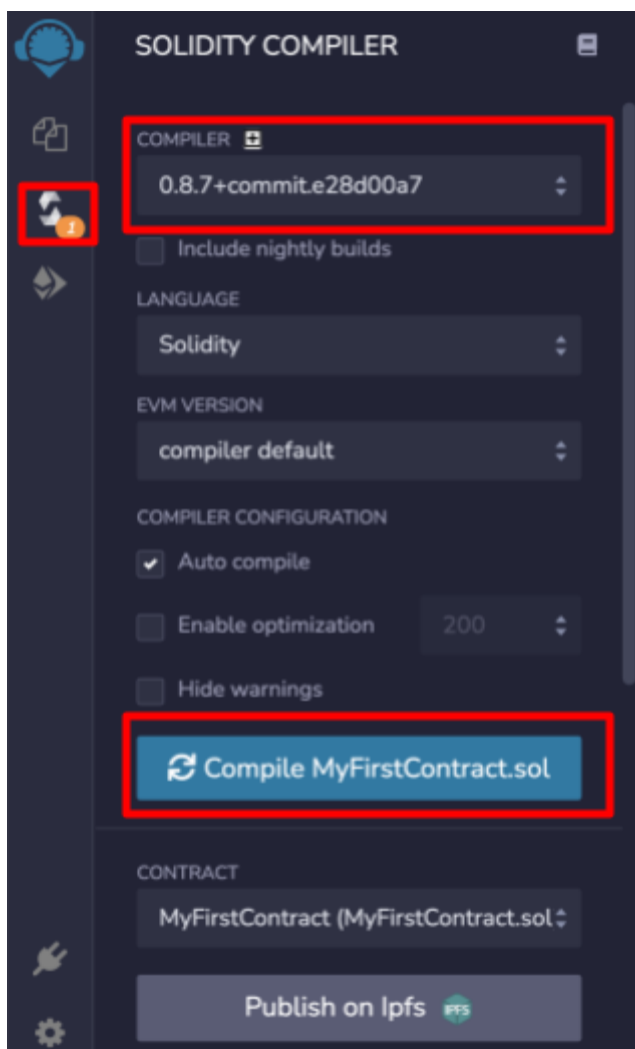
    function getName(uint _index) public view returns (string memory) {
        return names[_index];
    }

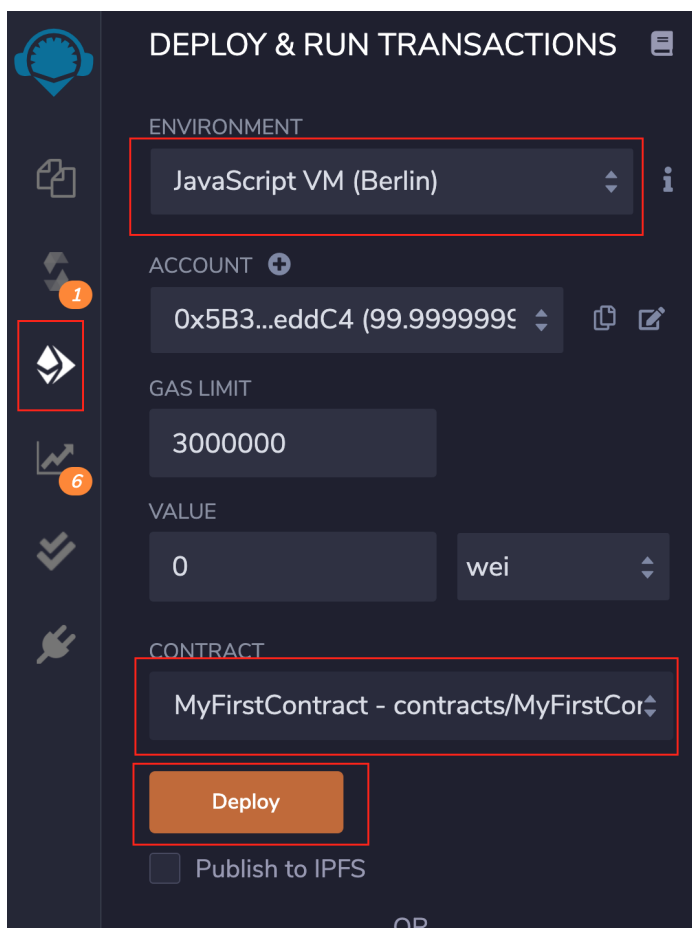
    function changeNumber(uint256 _num) public {
        number = _num;
    }

    function getNumber() public view returns (uint256){
        return number;
    }
}

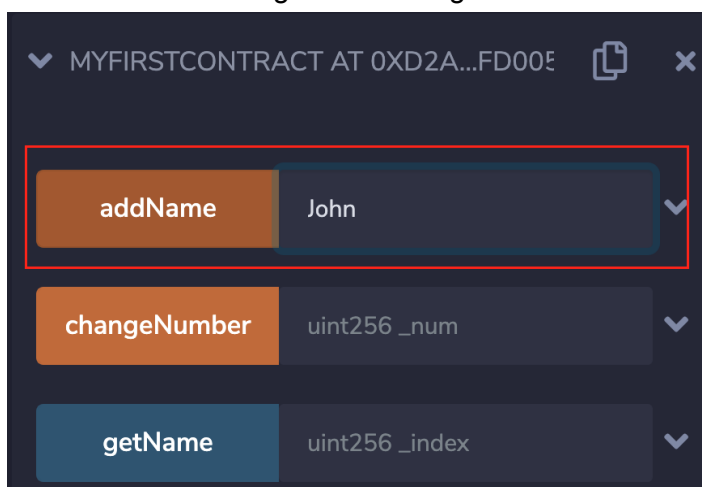
```

6. Bây giờ bạn đã sẵn sàng để kiểm tra hợp đồng của mình! Biên dịch và triển khai lại nó. Bạn có thể bỏ qua mọi cảnh báo về thời gian biên dịch (chữ màu cam). Đối với bài tập này, chúng ta sẽ làm việc trong môi trường **EVM** dựa trên trình duyệt cục bộ, vì vậy hãy chọn môi trường **'JavaScript VM'**:

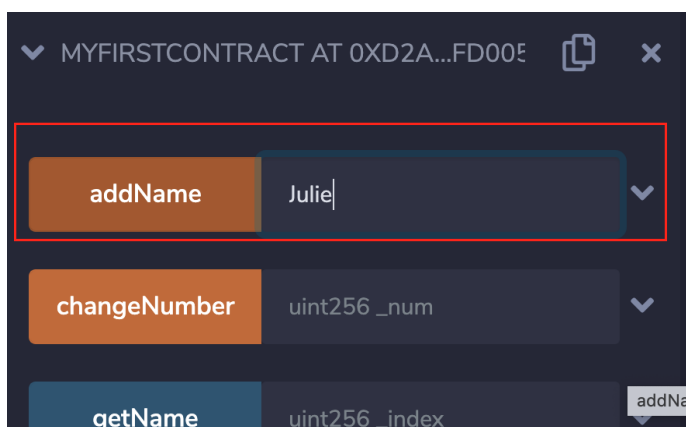




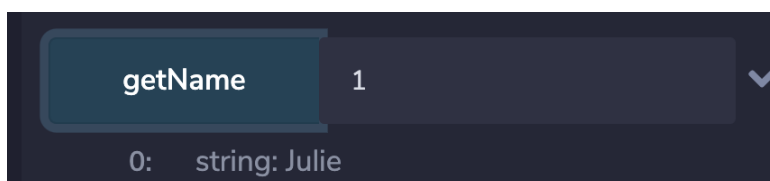
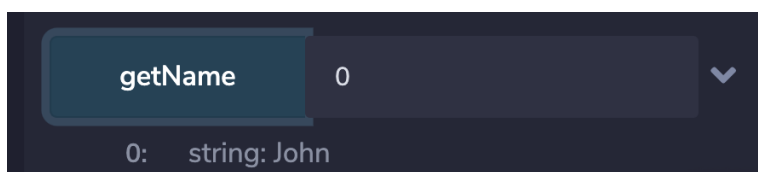
7. Sau khi được triển khai, hãy truyền một tham số cho hàm '**addName**' và thực thi nó để thêm **name** vào mảng **names** trong **Smart Contract**



8. Thêm một **name** khác mà bạn chọn để thêm nó vào mảng **names**



9. Bây giờ, hãy gọi hàm '**getName**', truyền vào **index 0** để xem giá trị đầu tiên được lưu trữ trong mảng. Lặp lại bước sau đó, chuyển vào chỉ số 1 để xem giá trị thứ hai.



Xin chúc mừng, bạn đã tạo thành công một mảng động trong Smart Contract của mình và điền dữ liệu vào đó.

Thêm ánh xạ

Đối với phần này của bài tập, bạn sẽ tạo ánh xạ **names->mobile numbers** trong **Smart Contract** của mình

1. Trước tiên, bạn cần thêm ánh xạ mới vào **Smart Contract** của mình. Tạo một ánh xạ mới của **name** thành số nguyên (**mobile number**) như sau:

```
mapping (string => uint) public phoneNumbers;
```


2. Tạo một hàm để thêm các giá trị vào ánh xạ. Nó sẽ nhận hai tham số, được gọi là ***_name*** và ***_mobileNumber***

```
function addMobileNumber(string memory _name, uint _mobileNumber) public {
    phoneNumbers[_name] = _mobileNumber;
}
```

3. Thêm một chức năng khác để lấy ***mobile number*** từ ánh xạ cho một ***name*** nhất định. Nó sẽ nhận một tham số ***_name*** và trả về một ***uint***

```
function getMobileNumber(string memory _name) public view returns (uint) {
    return phoneNumbers[_name];
}
```

4. Mã nguồn của bạn bây giờ sẽ trông như thế này

```
pragma solidity ^0.8.7;

contract MyFirstContract {

    uint256 number = 0;

    //Dynamic array (variable size)
    string[] names;
    mapping (string => uint) public phoneNumbers;

    function addMobileNumber(string memory _name, uint _mobileNumber) public {
        phoneNumbers[_name] = _mobileNumber;
    }

    function getMobileNumber(string memory _name) public view returns (uint) {
        return phoneNumbers[_name];
    }

    function addName(string memory _name) public {
        names.push(_name);
    }

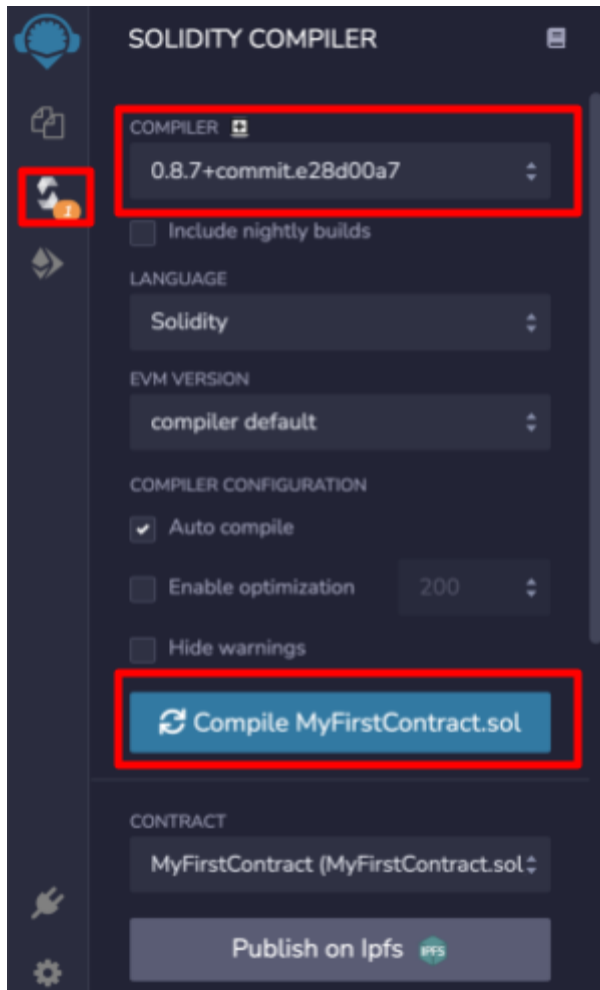
    function getName(uint _index) public view returns (string memory) {
```

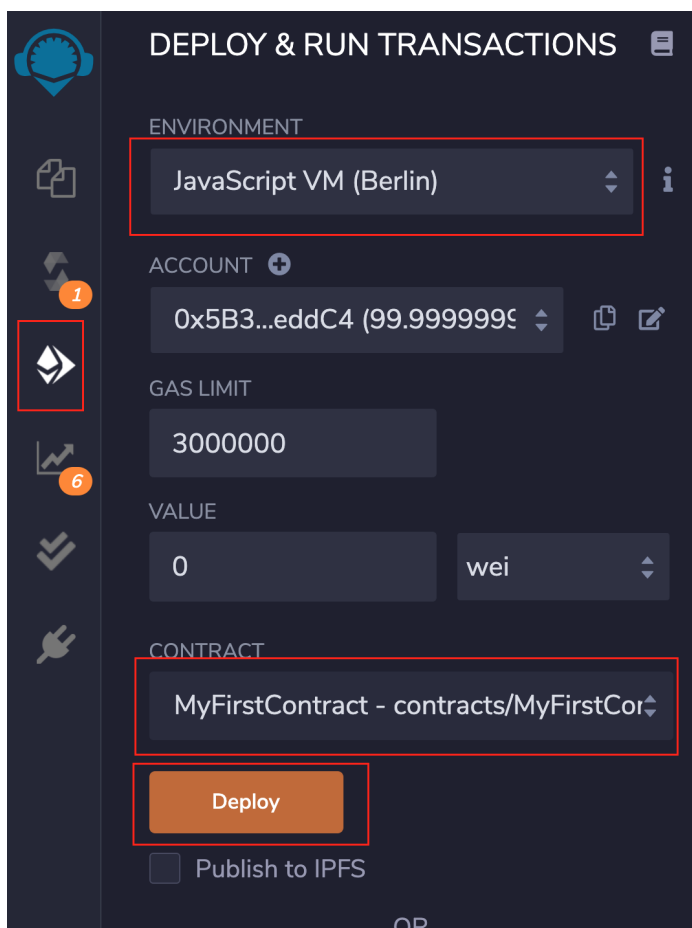
```
        return names[_index];
    }


    function changeNumber(uint256 _num) public {
        number = _num;
    }

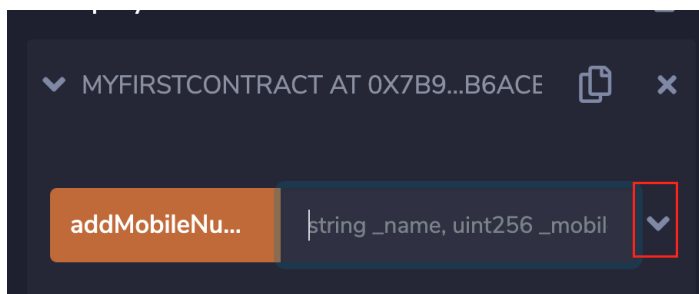
    function getNumber() public view returns (uint256){
        return number;
    }
}
```

10. Bây giờ bạn đã sẵn sàng để kiểm tra hợp đồng của mình! Biên dịch và triển khai lại nó. Bạn có thể bỏ qua mọi cảnh báo về thời gian biên dịch (văn bản màu cam). Đối với bài tập này, một lần nữa chúng ta sẽ làm việc trong môi trường **EVM** dựa trên trình duyệt cục bộ, vì vậy hãy chọn môi trường **'JavaScript VM'**:





11. Sau khi triển khai, hãy nhấn vào biểu tượng  tại '**addMobileNumber**' để bạn có thể xem cả hai tham số. Nhập vào một số giá trị và thực hiện chức năng



▼ MYFIRSTCONTRACT AT 0X7B9...B6ACE

addMobileNumber

_name:

_mobileNumber:

transact

12. Thêm vào một **name** và **mobile number** khác vào ánh xạ

▼ MYFIRSTCONTRACT AT 0X7B9...B6ACE

addMobileNumber

_name:

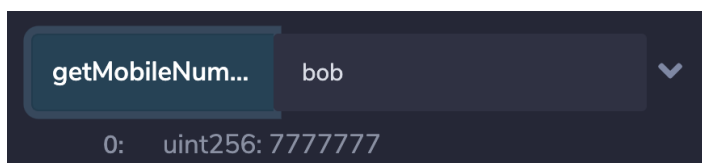
_mobileNumber:

transact

13. Bây giờ, hãy gọi hàm '**getMobileNumber**', chuyển một trong các **name** đã nhập vào ánh xạ ở các bước trước. Bạn sẽ thấy nó trả về **number** đã lưu so với **name**. Lặp lại bước cho **name** khác đã nhập

getMobileNum...

0: uint256: 34543



Xin chúc mừng, bạn đã tạo thành công một ánh xạ trong Smart Contract của mình, đồng thời điền và tương tác với nó!

Bài tập:

Bạn có thể làm các bài tập này nếu như bạn đã hoàn thành bài tập chính trước thời hạn:

1. Thay đổi Smart Contract của bạn để trả về độ dài của mảng **names**, gọi hàm đó là **getNamesLength()**. Kiểu trả về phải là **uint** và cách để lấy độ dài của một mảng là theo cú pháp sau

```
return names.length;
```

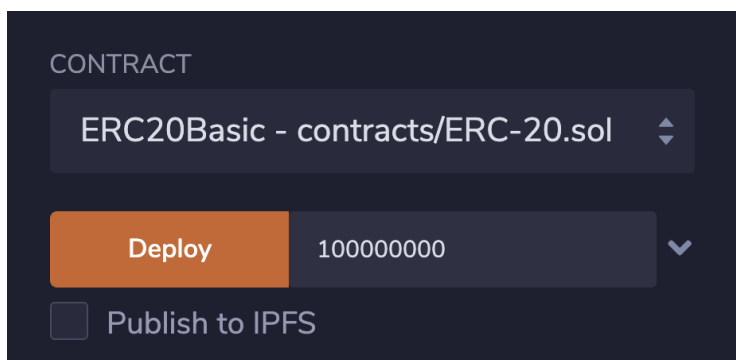
2. Tạo một hàm trong Smart Contract của bạn để trả về toàn bộ mảng **names** dưới dạng một string. Bạn sẽ cần thêm câu lệnh pragma ABIEncoderV2 ở phía dưới câu lệnh "pragma solidity ^***". I E:

```
pragma solidity ^0.8.7;
pragma solidity experimental ABIEncoderV2;
```

Gọi hàm **getNames**. Nó phải là một hàm **view**, với kiểu trả về như sau:

```
returns (string[] memory)
```

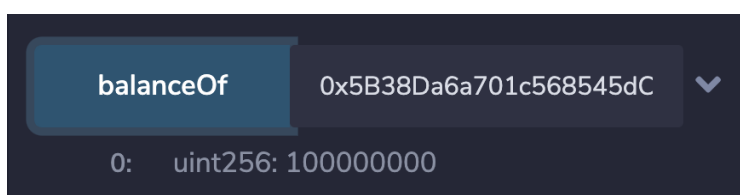
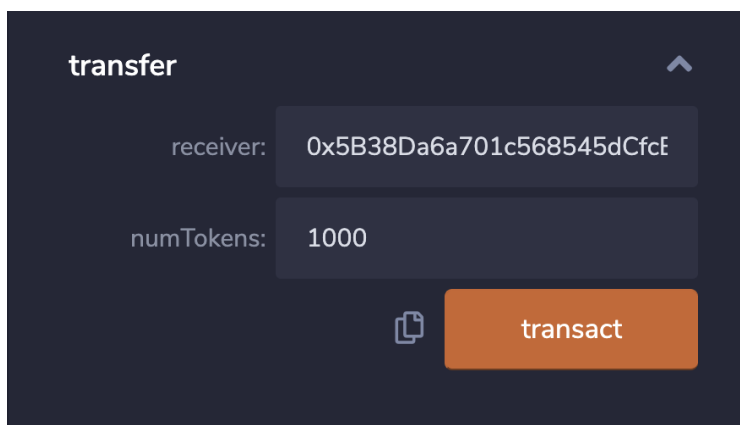
3. Tạo một Smart Contract mới trong Remix (trong một tệp mới) có tên 'ERC-20.sol'. Dán vào ví dụ mã hoàn chỉnh của hợp đồng ERC-20 cơ bản từ trang [hướng dẫn lập trình Ethereum](#). Sửa đổi các tham số **name** và **symbol** trong hàm tạo của hợp đồng ERC20Basic, chọn bất kỳ giá trị nào bạn muốn
4. Triển khai hợp đồng ERC20Basic tại local JavaScript VM network. Bạn cần chỉ định tổng số **tokens** trong smart contract khi bạn deploy. Trong ví dụ này, chúng ta đã chọn **100 triệu**. Đảm bảo rằng bạn đã chọn đúng hợp đồng để deploy.



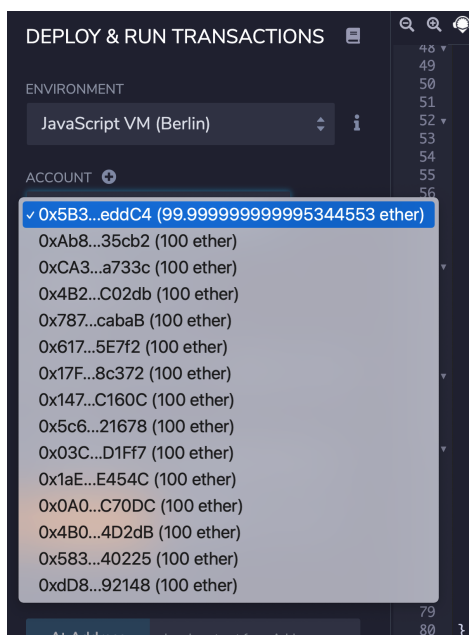
- Sau khi triển khai, hãy sao chép **Account** hiện được chọn của bạn trong Remix bằng cách nhấn biểu tượng sao chép bên phải.



- Gọi chức năng **transfer** trong hợp đồng đã deploy của bạn để mint **tokens**, paste address **Account** của bạn và số lượng **token**. Sau đó, gọi hàm **'balanceOf'**, paste address account của bạn vào để xem số lượng token hiện có.



7. Tiếp tục tìm hiểu các tính năng của token và xem các chức năng khác như 'transferFrom' hoạt động như thế nào. Bạn có thể chuyển đổi giữa các address ví của mình trong **Account** của bạn bằng cách sử dụng menu thả xuống '**Account**' trong Remix, để bạn có thể gửi số lượng token giữa các tài khoản ví của mình

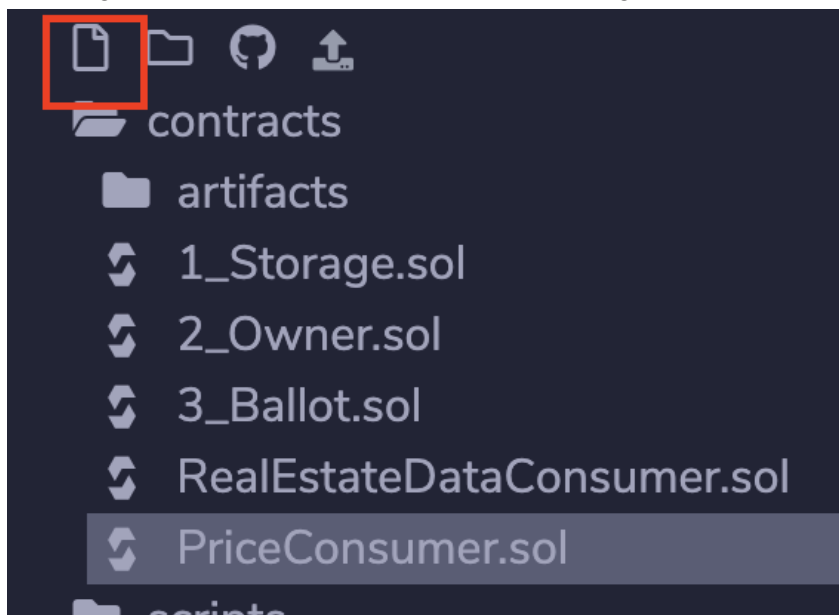


Xin chúc mừng, bạn đã xây dựng, triển khai và tương tác với token smart contract.

Bài tập 3: Price Feeds

Trong bài tập này, bạn sẽ tạo một Smart Contract đơn giản để tra cứu giá hiện tại của Ethereum bằng cách sử dụng Chainlink Price Feeds. Đầu tiên, mở <http://remix.ethereum.org/>

1. Mở rộng thư mục **contracts** và nhấn biểu tượng tệp mới. Gọi tệp PriceConsumer.sol



2. Chọn phiên bản compiler:

```
pragma solidity ^0.8.7;
```

3. Bên dưới, nhập code smart contract. Trong ví dụ đầu tiên của chúng ta, chúng ta sử dụng hợp đồng price feed ETH/USD **0xD4a33860578De61DBAbDc8BFdb98FD742fA7028e** mà chúng ta đã lấy từ phần Goerli của trang [Ethereum Price Feeds](#) của tài liệu Chainlink

```
import
"@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract PriceConsumerV3 {

    AggregatorV3Interface internal priceFeed;

    /**
     * Network: Goerli
```

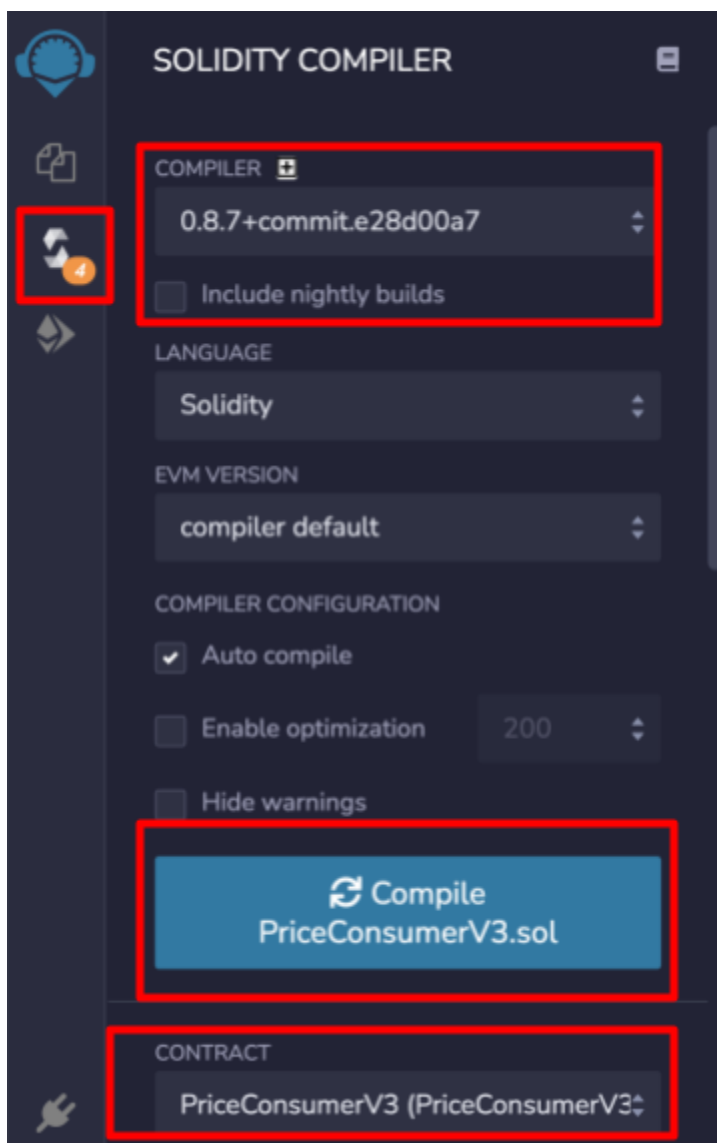
```

* Aggregator: ETH/USD
* Address: 0xD4a33860578De61Dc8BfDb98FD742fA7028e
*/
constructor() public {
    priceFeed =
AggregatorV3Interface(0xD4a33860578De61Dc8BfDb98FD742fA7028e);
}

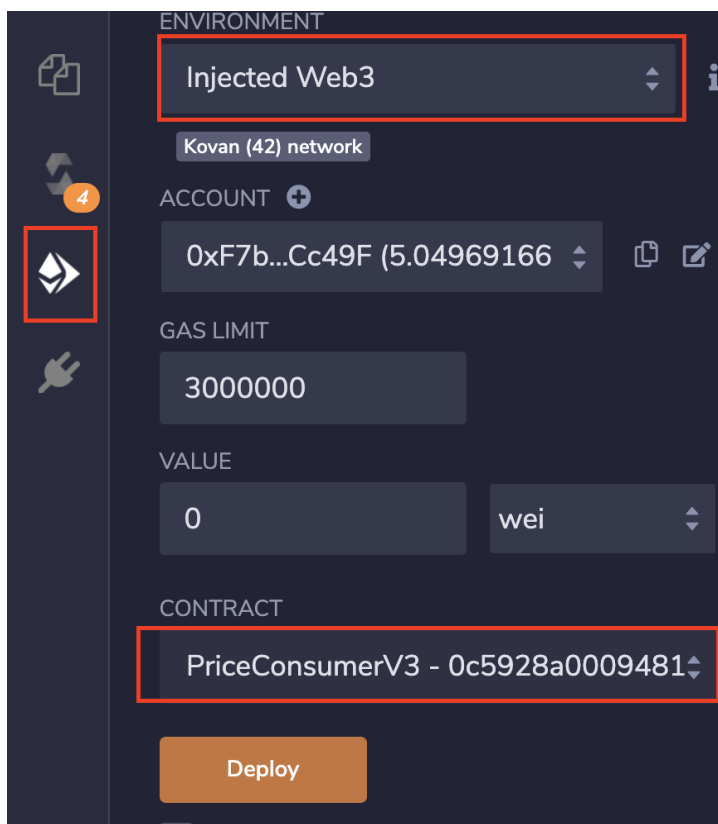
/**
* Returns the latest price
*/
function getLatestPrice() public view returns (int) {
    (
        uint80 roundID,
        int price,
        uint startedAt,
        uint timeStamp,
        uint80 answeredInRound
    ) = priceFeed.latestRoundData();
    return price;
}
}

```

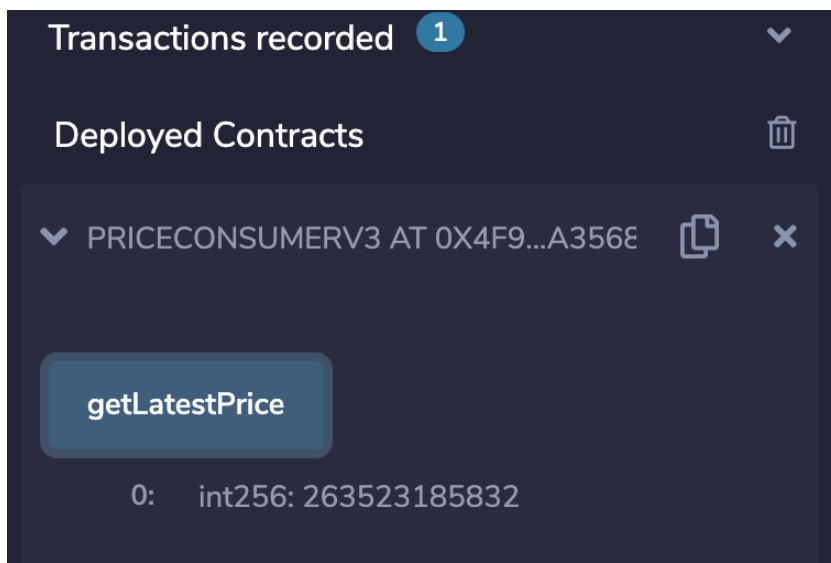
4. Hợp đồng của bạn hiện đã sẵn sàng để compile. Nhấn vào tùy chọn Solidity Compiler trên menu bên trái, thay đổi phiên bản **Compiler** trong trình đơn thả xuống của trình biên dịch để nó khớp với trình biên dịch được chỉ định trong code của bạn, sau đó nhấn vào nút Compile màu xanh lam:



5. Chọn nút '**Deploy and Run Transactions**' trên menu bên trái. Bởi vì smart contract sẽ integrate vào mạng Chainlink Oracle, chúng ta cần kết nối với mạng thử nghiệm của Goerli. Đảm bảo **Environment** được đặt thành '**Injected Web3**' và hợp đồng đã chọn là '**PriceConsumer.sol**', sau đó nhấn nút **Deploy**. Metamask sẽ bật lên yêu cầu bạn xác nhận giao dịch. Nhấn nút '**Confirm**' màu xanh lam để thực hiện giao dịch và triển khai hợp đồng của bạn vào mạng Goerli. Sau vài giây, bạn sẽ thấy hợp đồng của mình trong phần '**Deployed Contracts**' trong Remix.



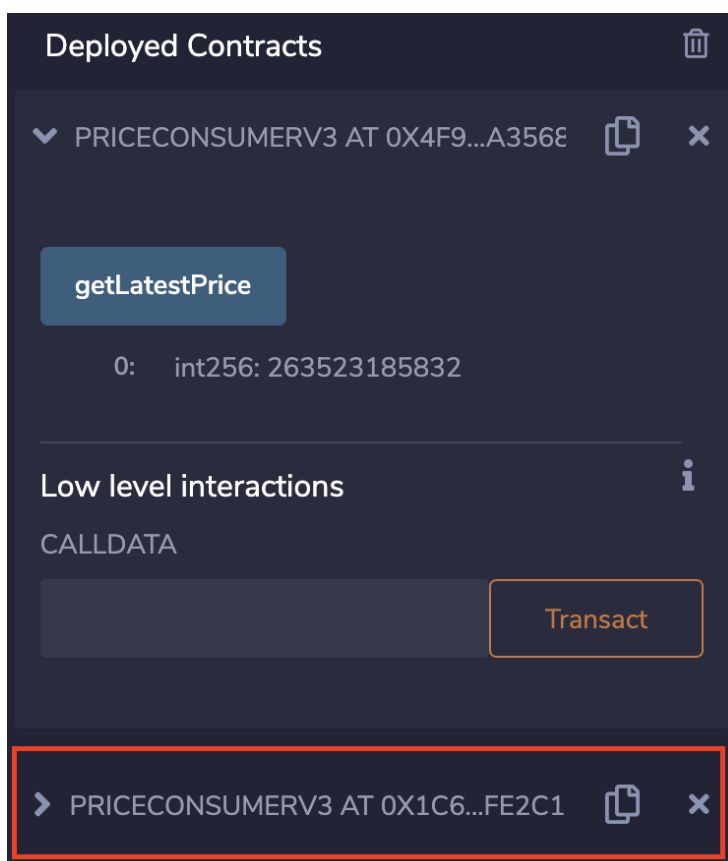
6. Mở rộng hợp đồng đã triển khai bằng cách nhấn vào biểu tượng ">" bên cạnh tên smart contract trong phần **Deployed Contracts**. Bạn sẽ thấy các chức năng của smart contract. Nhấn vào nút '**getLatestPrice**' để tra cứu giá **ETH** từ ETH/USD price feed của Goerli.



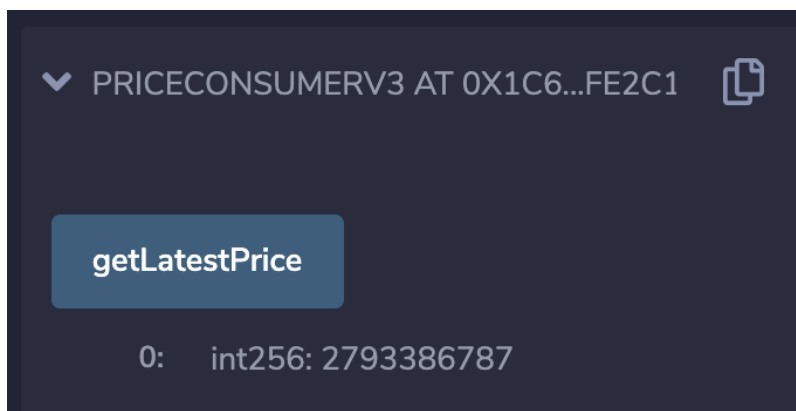
7. Kết quả trả về là giá thị trường hiện tại của ETH, nhân với 10^8 . Bạn có thể so sánh nó với ETH/USD Price Feed của Mainnet bằng cách truy cập <https://data.chain.link/eth-usd> và xem giá hiện tại.
8. Truy cập trang Ethereum Price Feeds trong tài liệu Chainlink và điều hướng xuống phần 'Goerli Testnet'. Chọn một cặp khác và sao chép địa chỉ Proxy. Trong ví dụ này chúng ta sẽ sử dụng LINK/USD **0x48731cF7e84dc94C5f84577882c14Be11a5B7456**
9. Quay lại hợp đồng của bạn và trong hàm constructor, thay thế địa chỉ của ETH/USD price feed bằng địa chỉ mà bạn đã sao chép ở bước trước.

```
priceFeed =
AggregatorV3Interface(0x48731cF7e84dc94C5f84577882c14Be11a5B7456);
```

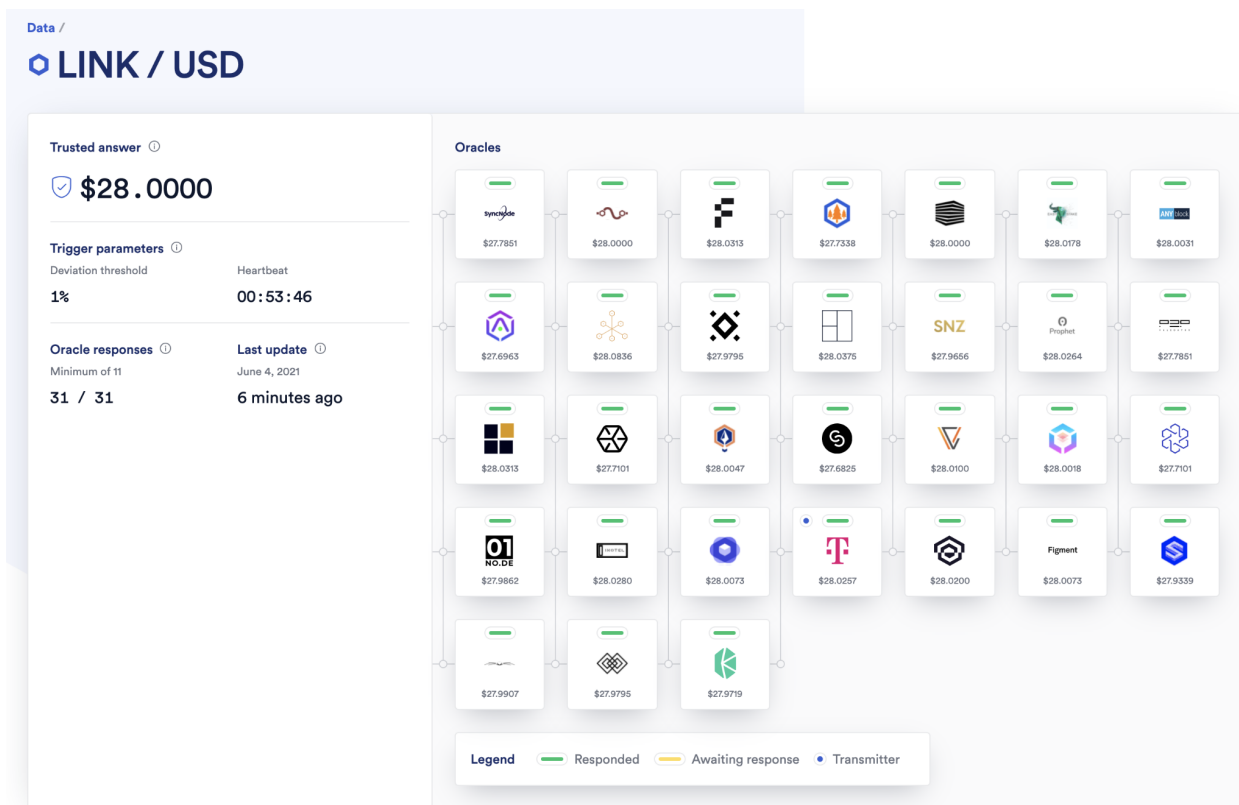
10. Quay lại tab Solidity Compiler trên Remix và nhấn nút 'Compile PriceConsumer.sol' màu xanh lam để biên dịch lại smart contract của bạn.
11. Chuyển đến tab 'Deploy & Run Transactions' và deploy lại smart contract của bạn, chấp nhận giao dịch trong Metamask. Sau khi deploy, bạn sẽ thấy smart contract thứ hai được deploy trong phần 'Deployed Contracts'.



12. Mở rộng hợp đồng đã triển khai và thực hiện chức năng **'getLatestPrice'**. Hàm sẽ trả về giá hiện tại trong hợp đồng **price feed** mà bạn đã chọn.



13. Truy cập <https://data.chain.link/> và tìm price feed tương đương đang chạy trên Mainnet.



Xin chúc mừng, bạn đã viết và triển khai thành công một hợp đồng sử dụng Chain Link Price Feeds để lấy dữ liệu giá bên ngoài!

Bài tập:

Bạn có thể cố gắng hoàn thành các bài tập này nếu bạn đã hoàn thành bài tập chính trước thời hạn:

1. Chỉnh sửa smart contract Price Feed để nhận price feed khác mà bạn chọn từ các địa chỉ Price Feed của Goerli trong [Tài liệu Chainlink](#). Sau đó triển khai và kiểm tra smart contract của bạn. So sánh giá trị với giá trị mainnet feed tại <https://data.chain.link>
2. Tạo một hàm mới gọi là 'getTimestamp' trả về một uint là timeStamp của round hiện tại (gợi ý: có thể truy cập hàm này thông qua lệnh gọi priceFeed.latestRoundData())

Bài tập 3: VRF v2

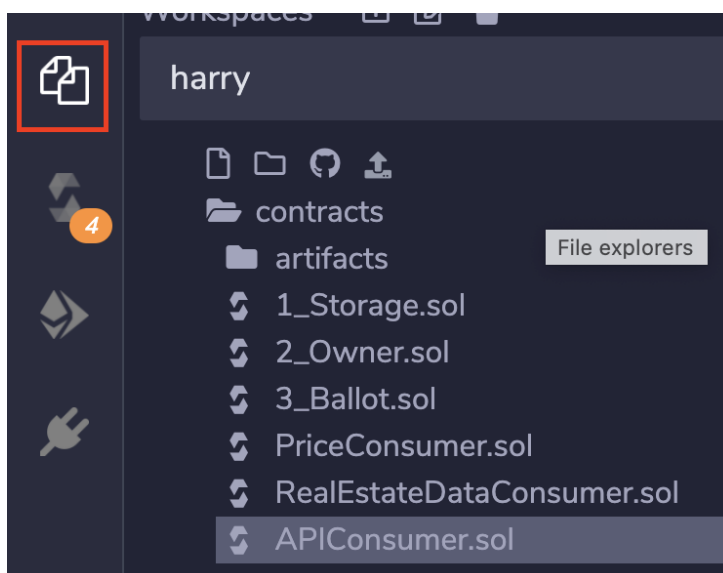
In this exercise, you're going to create a simple smart contract that performs a request for randomness using Chainlink VRF.

This exercise is detailed in the tutorial [get a random number](#)

Bài tập 4: Any-API

Trong bài tập này, bạn sẽ tạo một Smart Contract đơn giản thực hiện yêu cầu về dữ liệu external, sử dụng Chainlink Any-API. Đầu tiên, mở <http://remix.ethereum.org/>. This exercise uses the Goerli network, so make sure you have Goerli ETH and LINK in you

1. Mở rộng thư mục hợp đồng và nhấn biểu tượng tệp mới. Tạo tệp APIConsumer.sol



2. Chọn phiên bản trình biên dịch mà chúng ta sẽ sử dụng bằng cách nhập dòng sau vào tệp mới:

```
pragma solidity ^0.8.7;
```

3. Truy cập URL sau trong trình duyệt của bạn để xem đầu ra json của API mà chúng ta sẽ truy cập. Tìm kiếm cụm từ “ **VOLUME24HOUR** ” để xem giá trị chính xác mà chúng ta sẽ tìm kiếm để đưa vào Smart Contract của mình

```
https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD
```

4. Quay lại Remix, bên dưới lệnh pragma của bạn, hãy viết mã nguồn hợp đồng như dưới đây

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";

/**
 * Request testnet LINK and ETH here: https://faucets.chain.link/
 * Find information on LINK Token Contracts and get the latest ETH
and LINK faucets here:
https://docs.chain.link/docs/link-token-contracts/
 */
```



```

/**
 * THIS IS AN EXAMPLE CONTRACT WHICH USES HARDCODED VALUES FOR
 * CLARITY.
 * PLEASE DO NOT USE THIS CODE IN PRODUCTION.
 */
contract APIConsumer is ChainlinkClient {
    using Chainlink for Chainlink.Request;

    uint256 public volume;

    address private oracle;
    bytes32 private jobId;
    uint256 private fee;

    /**
     * Network: Goerli
     * Oracle: 0xc57B33452b4F7BB189bB5AfaE9cc4aBa1f7a4FD8 (Chainlink
    Devrel
     * Node)
     * Job ID: d5270d1c311941d0b08bead21fea7747
     * Fee: 0.1 LINK
     */
    constructor() {
        setPublicChainlinkToken();
        oracle = 0xc57B33452b4F7BB189bB5AfaE9cc4aBa1f7a4FD8;
        jobId = "d5270d1c311941d0b08bead21fea7747";
        fee = 0.1 * 10 ** 18; // (Varies by network and job)
    }

    /**
     * Create a Chainlink request to retrieve API response, find the
    target
     * data, then multiply by 1000000000000000000 (to remove decimal
    places from data).
     */
    function requestVolumeData() public returns (bytes32 requestId)
    {
        Chainlink.Request memory request =

```

```

buildChainlinkRequest(jobId, address(this), this.fulfill.selector);

    // Set the URL to perform the GET request on
    request.add("get",
"https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD");

    // Set the path to find the desired data in the API response,
where the response format is:
    // {"RAW":
    //   {"ETH":
    //     {"USD":
    //       {
    //         "VOLUME24HOUR": xxx.xxx,
    //       }
    //     }
    //   }
    // }
    request.add("path", "RAW.ETH.USD.VOLUME24HOUR");

    // Multiply the result by 1000000000000000000 to remove
decimals
    int timesAmount = 10**18;
    request.addInt("times", timesAmount);

    // Sends the request
    return sendChainlinkRequestTo(oracle, request, fee);
}

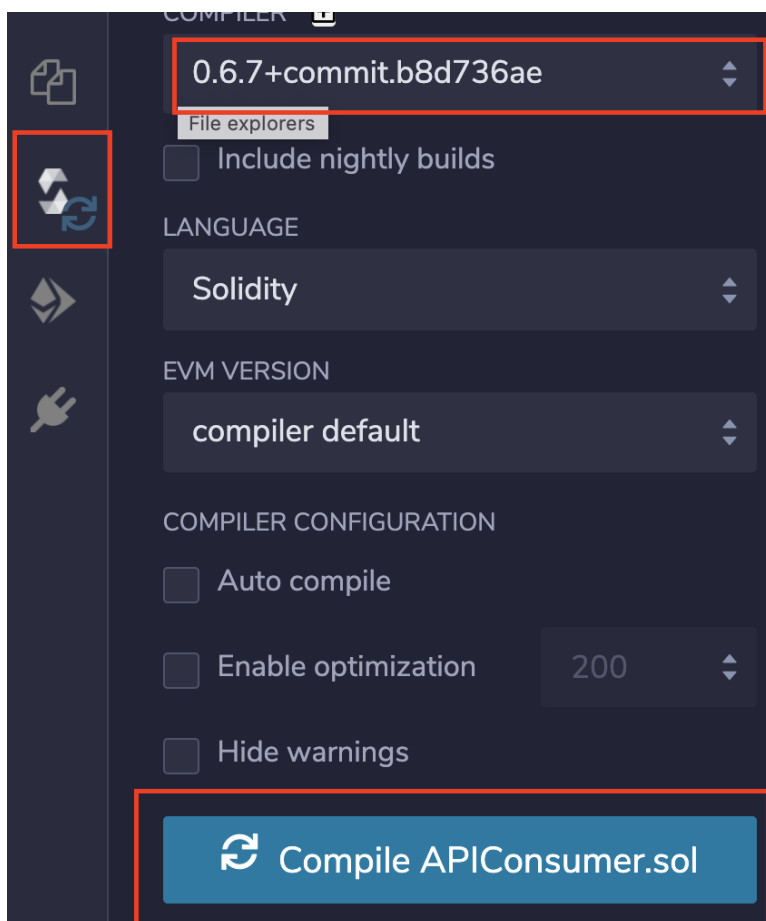
/**
 * Receive the response in the form of uint256
 */
function fulfill(bytes32 _requestId, uint256 _volume) public
recordChainlinkFulfillment(_requestId)
{
    volume = _volume;
}

// function withdrawLink() external {} - Implement a withdraw

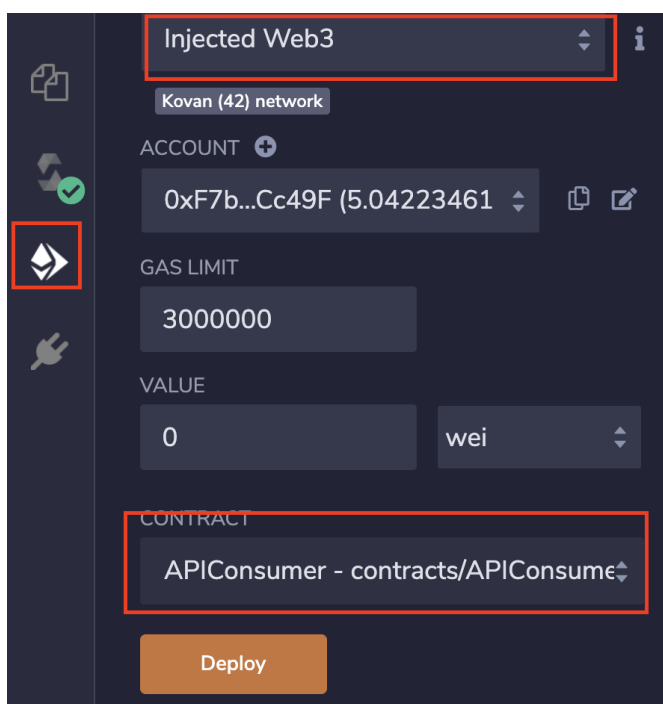
```

```
function to avoid locking your LINK in the contract
}
```

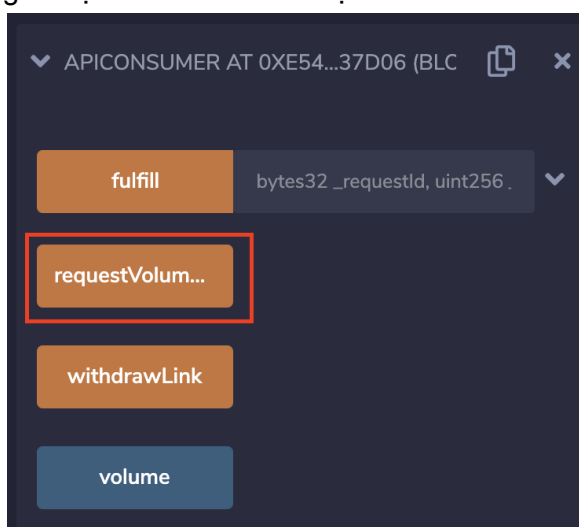
5. Hợp đồng của bạn hiện đã sẵn sàng để compile. Nhấn vào tùy chọn menu Solidity Compiler trên menu bên trái, thay đổi phiên bản Compiler trong trình đơn thả xuống của trình biên dịch để nó khớp với trình biên dịch được chỉ định trong mã của bạn, sau đó nhấn vào nút Compile màu xanh lam:



6. Chọn nút 'Deploy and Run Transactions' trên menu bên trái. Bởi vì bạn sẽ tích hợp vào mạng Chainlink Oracle, chúng ta cần kết nối với mạng thử nghiệm Goerli. Đảm bảo Environment được đặt thành 'Injected Web3' và hợp đồng đã chọn là APIConsumer.sol', sau đó nhấn nút Deploy. Metamask sẽ bật lên yêu cầu bạn xác nhận giao dịch. Nhấn nút 'Confirm' màu xanh lam để thực hiện giao dịch và triển khai hợp đồng của bạn vào mạng Goerli. Sau vài giây, bạn sẽ thấy hợp đồng đã triển khai của mình trong phần 'Deployed Contracts' trong Remix.



7. [Cấp tiền cho hợp đồng của bạn bằng 1 LINK](#) token. Nếu bạn chưa thêm token LINK vào ví MetaMask của mình, hãy nhấn nút 'Add Token' ở dưới cùng, sau đó nhập địa chỉ hợp đồng của token LINK `0x326C977E6efc84E512bB9C30f76E30c160eD06FB`, sau đó nhấn lưu.
8. Khi hợp đồng của bạn đã nhận được LINK, bây giờ bạn có thể bắt đầu yêu cầu dữ liệu bên ngoài (external data). Mở rộng hợp đồng đã triển khai bằng cách nhấn vào biểu tượng ">" bên cạnh tên hợp đồng trong phần Deployed Contracts. Bạn sẽ thấy các chức năng có sẵn có thể được thực thi. Nhấn vào nút 'requestVolumeData', sau đó xác nhận giao dịch khi MetaMask bật lên.



9. Sau vài giây, giao dịch của bạn sẽ được phê duyệt. Trong khi bạn đợi chainlink thực hiện yêu cầu, hãy truy cập <https://goerli.etherscan.io/> và tìm kiếm địa chỉ hợp đồng đã triển khai của bạn (bạn đã sao chép địa chỉ đó vào khay nhớ tạm khi nạp tiền cho hợp đồng của mình bằng LINK). Sau khi bạn tìm thấy hợp đồng đã triển khai của mình trên Etherscan, hãy chuyển đến tab 'ERC-20 Token Txns'. Bạn sẽ thấy 2 giao dịch, một lần chuyển đến 1 LINK từ khi bạn tài trợ cho hợp đồng của mình và 0.1 LINK được chuyển ra ngoài hợp đồng cho yêu cầu dữ liệu bên ngoài (external data) của bạn.

Contract Overview

Balance: 0 Ether

Token: \$0.00

More Info

My Name Tag: Not Available

Contract Creator: 0x7b4ef69e7cf13c2055... at txn 0x369730cc6f78970fc9f3...

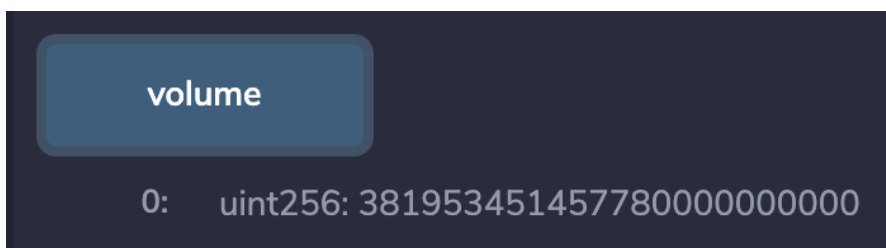
Transactions Internal Txns **Erc20 Token Txns** Contract Events

Latest 2 ERC-20 Token Transfer Events

Txn Hash	Age	From	To	Value	Token
0x3bdc2fd5fab01e37b51...	1 min ago	0xe54092f4a27442febb43c011b5c2aDD942537D06	0x2f90a6d021db21e1b2...	0.1	ChainLink To... (LINK)
0xed05b78ee2cd7475e1...	3 mins ago	0x7b4ef69e7cf13c2055...	0xe54092f4a27442febb43c011b5c2aDD942537D06	1	ChainLink To... (LINK)

[Download CSV Export]

10. Quay lại hợp đồng đã triển khai của bạn trong Remix. Bây giờ chúng ta sẽ kiểm tra xem kết quả trả về từ yêu cầu dữ liệu bên ngoài (external data). Nhấn nút 'volume' để thực thi hàm trả về giá trị của biến 'volume' công khai. Bây giờ, bạn sẽ có kết quả khớp với giá trị mà bạn đã thấy trong trình duyệt web của mình ở bước 19 ở trên. Giá trị đã được cố ý nhân với 10^{18} để tránh bất kỳ vấn đề nào với các phép toán, v.v. có thể xảy ra trên kết quả.



Xin chúc mừng, bạn đã thực hiện thành công yêu cầu dữ liệu bên ngoài (external data) và lấy dữ liệu từ API công khai vào Smart Contract của mình!

Bài tập:

Bạn có thể cố gắng hoàn thành các bài tập này nếu bạn đã hoàn thành bài tập chính trước giờ

1. Điều chỉnh hợp đồng của bạn để lấy dữ liệu **giá (price)** từ API cryptocompare thay vì từ volume. Đổi tên tất cả các biến liên quan đến volume để phản ánh sự thay đổi trong dữ liệu được đưa vào
2. Thay đổi API đang được truy cập thành thông tin sau để nhận giá (price) BTC/USD.
<https://api.cryptowat.ch/markets/kraken/btcusd/price>. Bạn cũng cần sửa đổi tham số "path" được gửi trong yêu cầu để khớp với đường dẫn JSON của trường "price" nếu bạn dán URL ở trên vào cửa sổ trình duyệt