



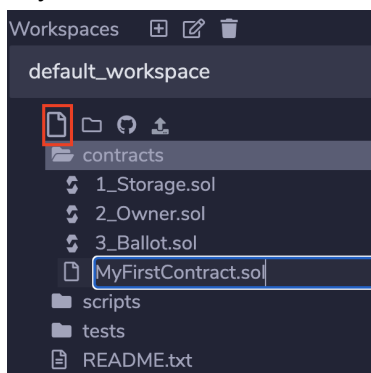
Curso de capacitación para desarrolladores de contratos inteligentes: Ejercicios día 1

Ejercicio 1: Mi primer contrato inteligente	2
Ejercicio bonus	11
Ejercicio 2: Arrays and Mappings	12
Añadir un Array	12
Añadiendo un Mapping	17
Ejercicio Bonus	23
Ejercicio 3: Price Feeds	26
Ejercicio Bonus	32
Ejercicio 4: Any-API	33
Ejercicio Bonus	39
Ejercicio 5: VRF	40
Ejercicio Bonus	46

Ejercicio 1: Mi primer contrato inteligente

En este ejercicio, crearás un contrato inteligente simple que almacene y recupere un valor, luego lo desplegarás en la red de pruebas de Kovan e interactuarás con él usando Remix.

1. Abrir <http://remix.ethereum.org/>
2. Si se le pide, cree un nuevo espacio de trabajo por defecto
3. Despliegue la carpeta de contratos y pulse el botón de nuevo archivo. Llame al archivo "MyFirstContract.sol"



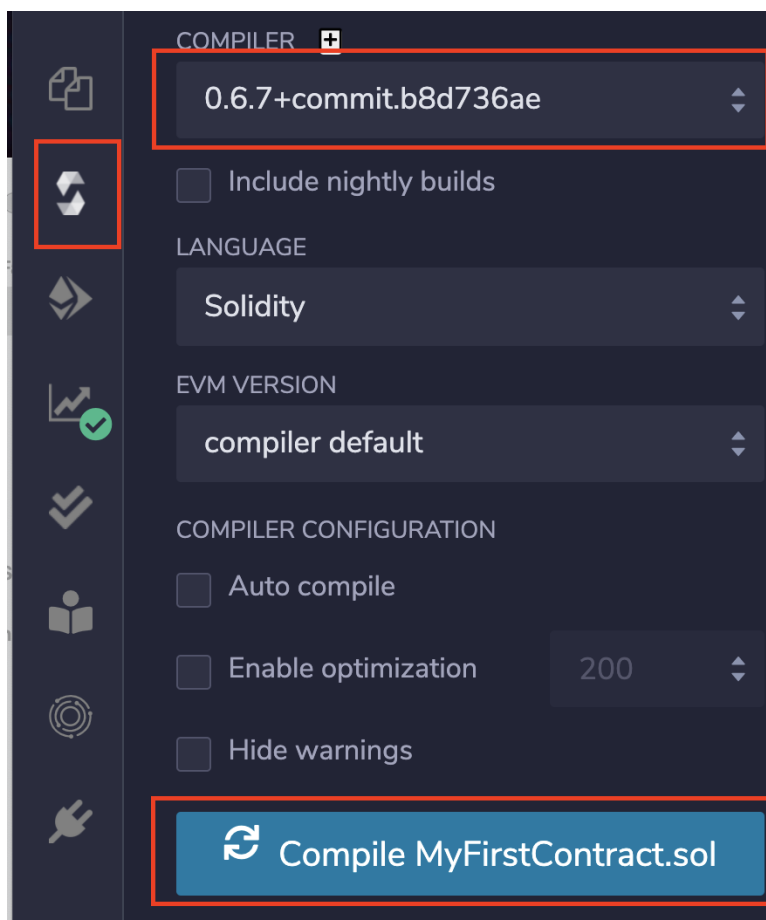
4. Seleccione qué versión del compilador vamos a utilizar introduciendo la siguiente línea en el nuevo archivo:

```
pragma solidity ^0.6.7;
```

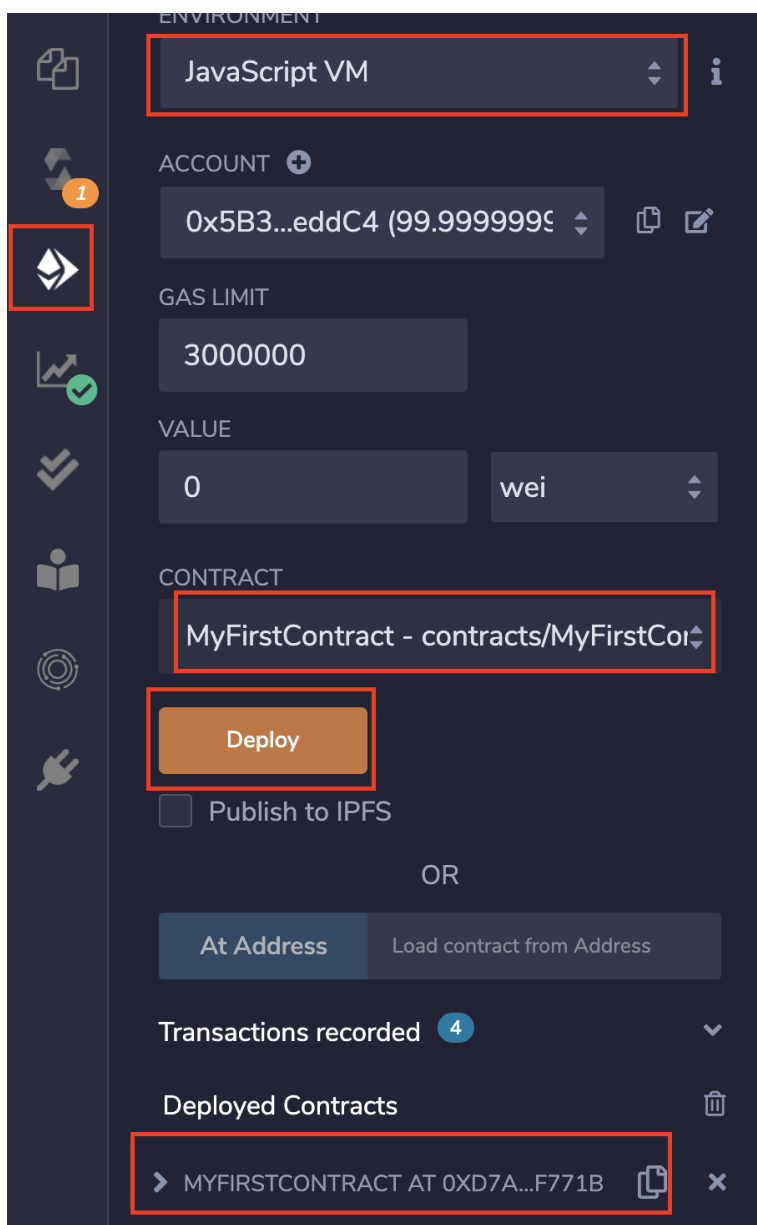
5. Debajo, introduzca el código fuente del contrato:

```
contract MyFirstContract {  
  
    uint256 number;  
  
    function changeNumber(uint256 _num) public {  
        number = _num;  
    }  
  
    function getNumber() public view returns (uint256){  
        return number;  
    }  
}
```

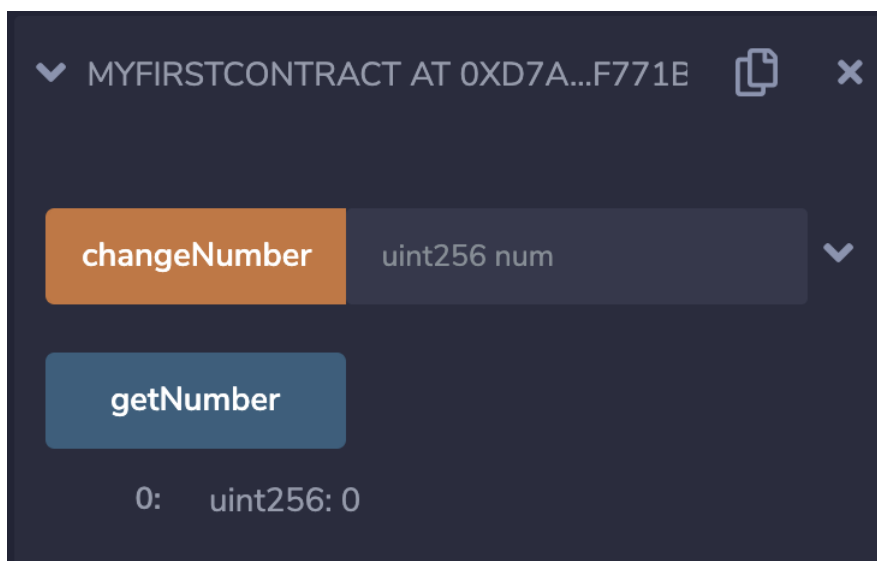
6. Su contrato está ahora listo para ser compilado. Pulse la opción de menú Solidity Compiler en el menú de la izquierda, cambie la versión del compilador en el desplegable del compilador para que coincida con el compilador especificado en su código, y luego pulse el botón azul "Compile":



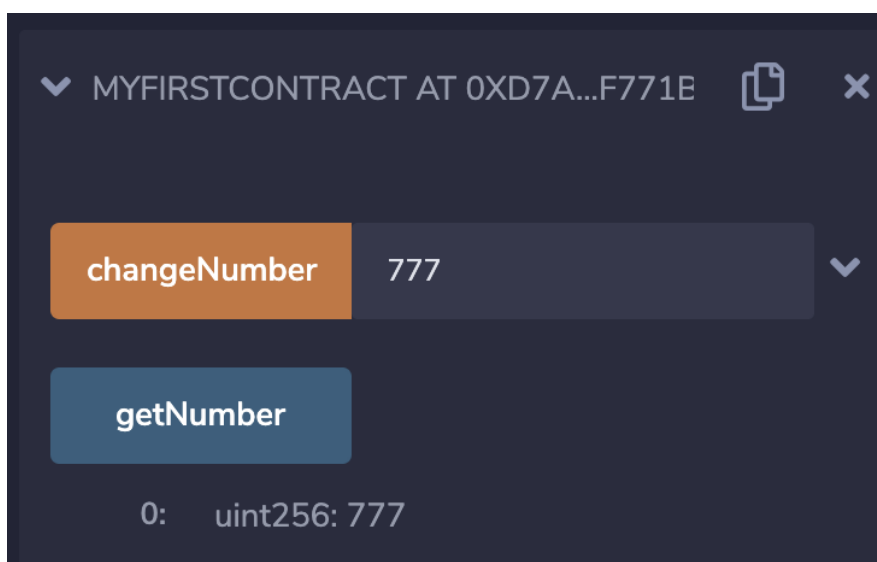
Elija el botón "Desplegar y ejecutar transacciones" en el menú de la izquierda. Asegúrese de que el Entorno está configurado como 'JavaScript VM', y que el contrato seleccionado es 'MyFirstContract', entonces pulse el botón de Despliegue. Debería ver su contrato más abajo en la sección "Deployed Contracts".



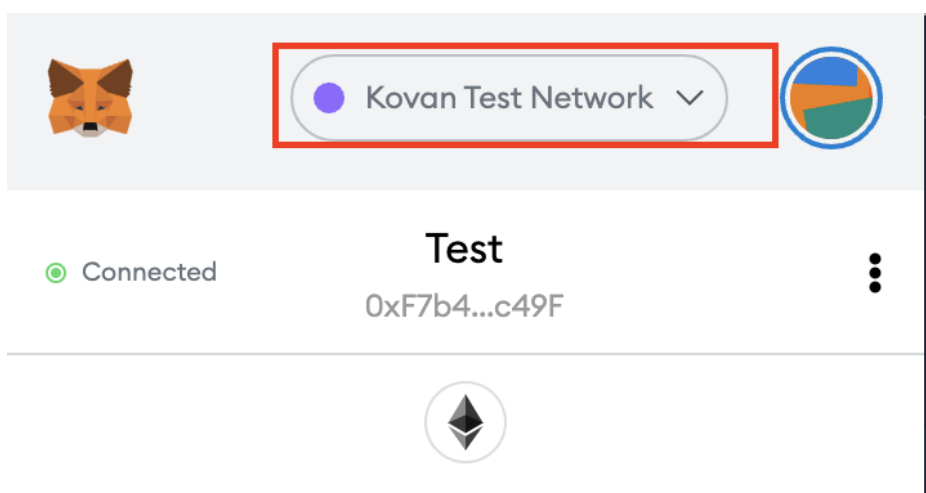
7. Despliegue el contrato desplegado seleccionando el signo > junto a él. Ahora puede ver todas las funciones que puede utilizar para interactuar con el contrato.
8. Pulsa la función 'getNumber' para leer el estado del contrato, y devuelve el valor de la variable número. Como aún no la hemos configurado, debería devolver simplemente 0



9. Introduzca un valor en el parámetro de entrada de la función 'changeNumber' y pulse el botón 'changeNumber' para ejecutar la función.
10. Ejecute de nuevo la función "getNumber". Ahora debería ver que el estado del contrato se ha actualizado.



11. Ahora vamos a intentarlo de nuevo, pero en lugar de trabajar en la red local de Remix VM, desplegaremos nuestro contrato en la red pública de pruebas de Kovan. Cambia el desplegable de "Entorno" a "Web3 inyectada". Si no has iniciado sesión en tu monedero Metamask, hazlo antes de continuar y asegúrate de que la red seleccionada es la "Red de pruebas Kovan" y de que tienes algo de ETH en tu monedero. Si no tienes ETH, levanta la mano para pedir ayuda y un instructor te ayudará a conseguirla.



12. Como ya hemos compilado nuestro contrato y no lo hemos cambiado, no necesitamos compilarlo de nuevo. Pulse el botón naranja de Despliegue para desplegar su contrato ya compilado. Metamask aparecerá pidiéndole que confirme la transacción. Pulsa el botón azul "Confirmar" para ejecutar la transacción y desplegar tu contrato en la red de Kovan. Después de unos segundos, debería ver su contrato desplegado en la sección "Contratos desplegados".

Kovan Test Network

Test → New Contract

http://remix.ethereum.org

CONTRACT DEPLOYMENT

0

DETAILS DATA

GAS FEE 0.000289
No Conversion Rate Available

Gas Price (GWEI) 3 Gas Limit 96405

AMOUNT + GAS FEE

TOTAL 0.000289
No Conversion Rate Available

Reject Confirm

13. Pulsa el botón de "copiar" la dirección del contrato en tu portapapeles, luego dirígete a <https://kovan.etherscan.io/>, aquí es donde puedes encontrar tu contrato desplegado en la red de Kovan, y cualquier transacción asociada a él. Pegue la dirección de su contrato en el campo de búsqueda y pulse buscar. Debería encontrar su contrato desplegado, con 1 transacción (creación de contrato).

Deployed Contracts

MYFIRSTCONTRACT AT 0X94E...C1611

changeNumber uint256 num

getNumber

Contract 0x94e4531CAaA39E91779345f86aA39C4890CC1611

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

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

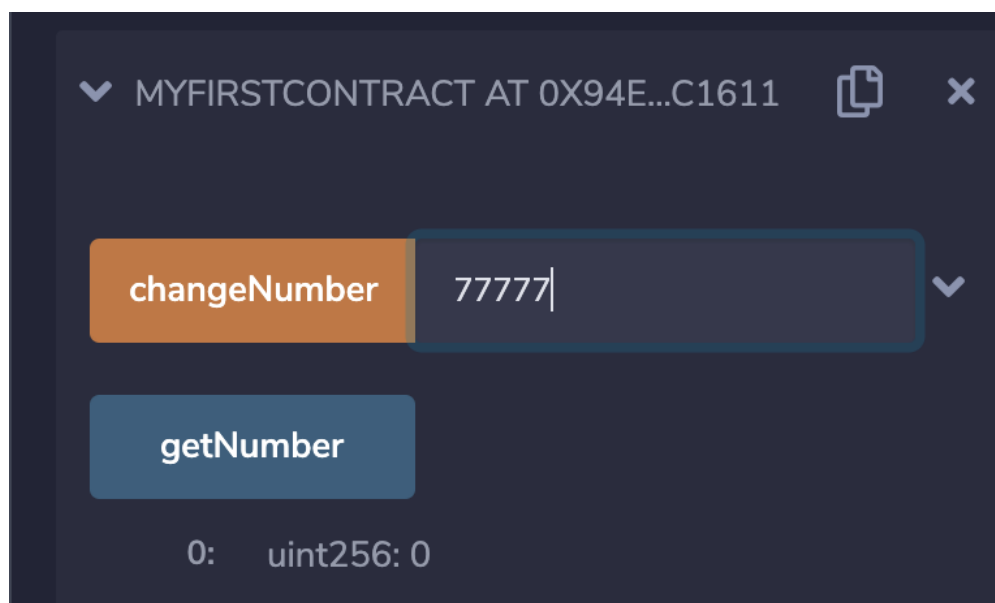
Transactions Contract Events

Latest 1 from a total of 1 transactions

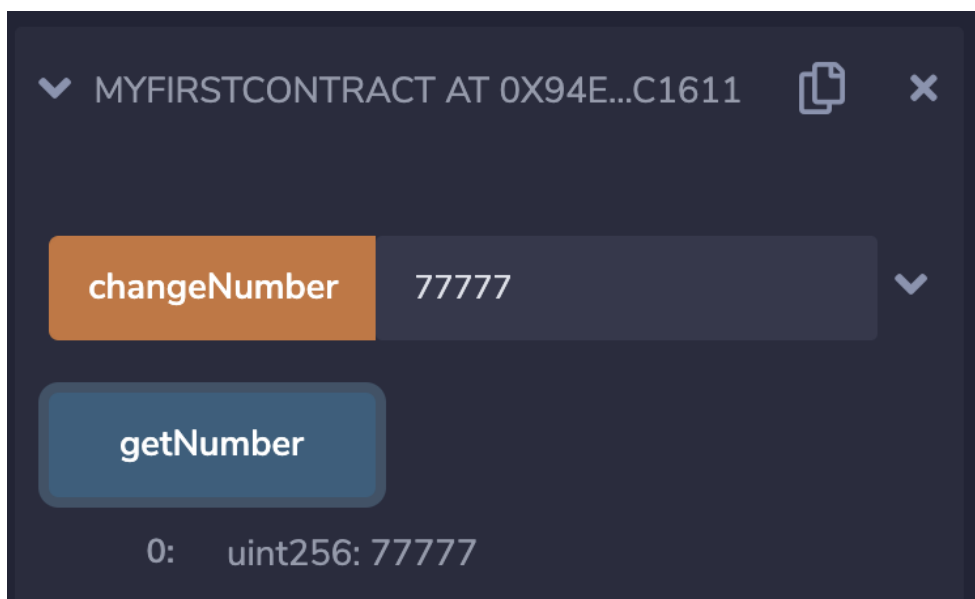
Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x12c2e628586cde181a...	0x60806040	25233021	47 secs ago	0xf7b4ef69e7cf13c2055...	IN Contract Creation	0 Ether	0.000289215

[Download CSV Export]

14. Una vez más, ejecute la función 'getNumber' y compruebe que el resultado devuelto es 0. A continuación, introduzca un número en el parámetro de la función 'changeNumber' y ejecute la función 'changeNumber' para actualizar el estado del contrato. Una vez más, Metamask aparecerá y le pedirá que confirme la transacción.



15. Dale a la transacción unos segundos para que se incluya en un bloque, y luego ejecuta de nuevo la función 'getNumber'. Ahora debería ver que el estado del contrato se ha actualizado



19. Si vuelve a Etherscan y actualiza la página con su contrato, debería ver su nueva transacción en la lista. Si pulsa el enlace 'Txn Hash', verá los detalles de la transacción.

Transactions Contract Events						
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

Ejercicio bonus

1. Modifica tu contrato inteligente para que en lugar de almacenar el número pasado, incremente el número actualmente almacenado por el parámetro `_num` pasado. Para ello, debes establecer inicialmente el estado de la variable `number` como 0

```
uint256 number = 0;
```

Puede incrementar la variable numérica almacenada con la siguiente sintaxis

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

2. Crea una nueva función llamada 'getNumberMultiplied', que toma un parámetro llamado `_num`, y luego devuelve un entero del resultado de multiplicar el parámetro `_num` con el parámetro número actualmente almacenado. La función debe ser definida como una función de vista similar a la función `getNumber()`, porque no estamos modificando el estado del contrato.
3. Crea una nueva función llamada 'addNumbers' que toma dos parámetros uint, `_num1` y `_num2`, y luego almacena el resultado de sumar los dos números en la variable `number`.

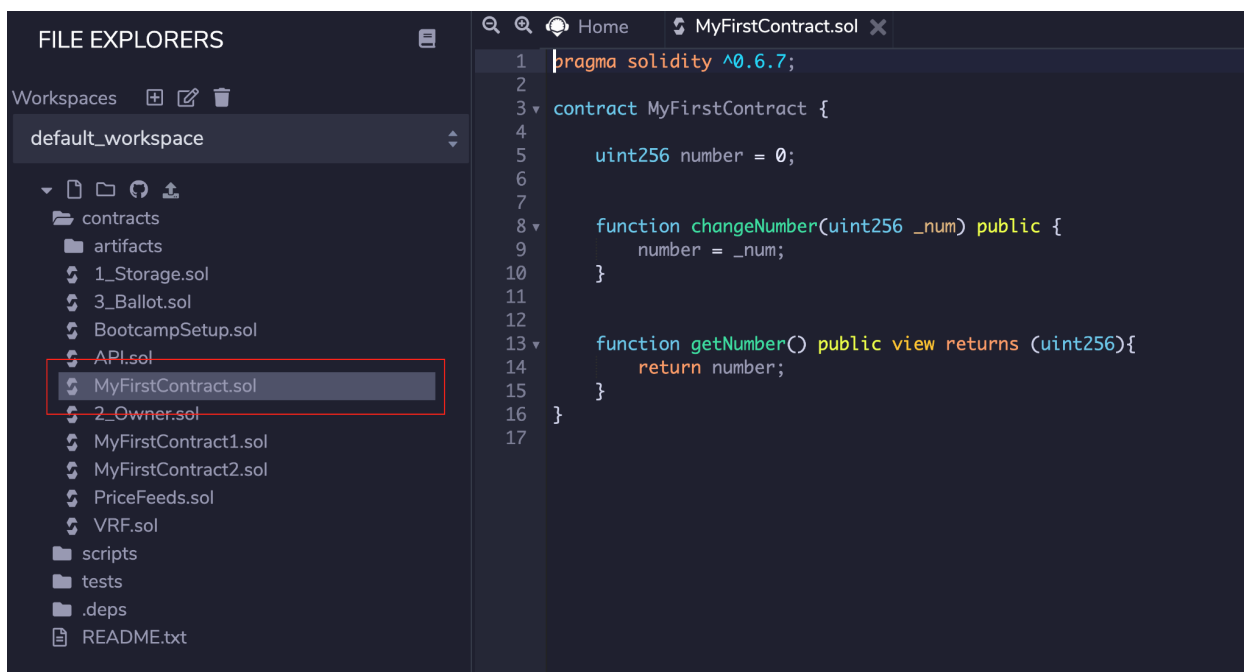
Ejercicio 2: Arrays and Mappings

En este ejercicio, vamos a modificar nuestro contrato inteligente `MyFirstSmartContract` del Día 1 del bootcamp, y vamos a añadirle un array y un Mapping.

Añadir un Array

Para esta parte del ejercicio, crearás una dynamic array de nombres en tu contrato inteligente, almacenados como strings.

1. Abre [Remix](#), deberías ser capaz de encontrar tu contrato inteligente completado de la última sesión en tu explorador



2. A continuación, tienes que añadir la nueva array a tu contrato inteligente. Crea una nueva dynamic array de strings llamada 'names'. Puedes añadirlo bajo la variable existente 'number'.

```
string[] names;
```

3. Añade una función para 'push' nuevos valores al array. Debe tomar un parámetro llamado `_name`

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

4. Añade otra función para obtener un nombre almacenado en el array, dado un parámetro de índice pasado. Debería devolver el nombre en el array en el índice dado. Como está leyendo el estado del contrato, puede ser una función de la vista.

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

5. Tu código completado debería tener ahora el siguiente aspecto

```
pragma solidity ^0.6.7;

contract MyFirstContract {

    uint256 number = 0;

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

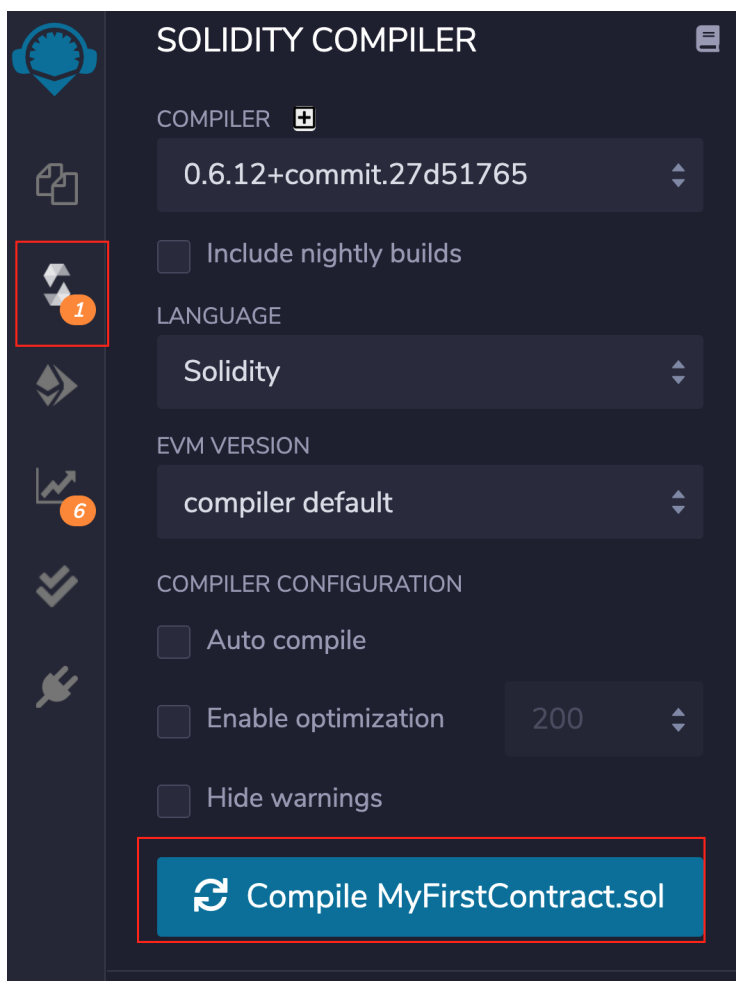
    function addName(string memory _name) public {
        names.push(_name);
    }

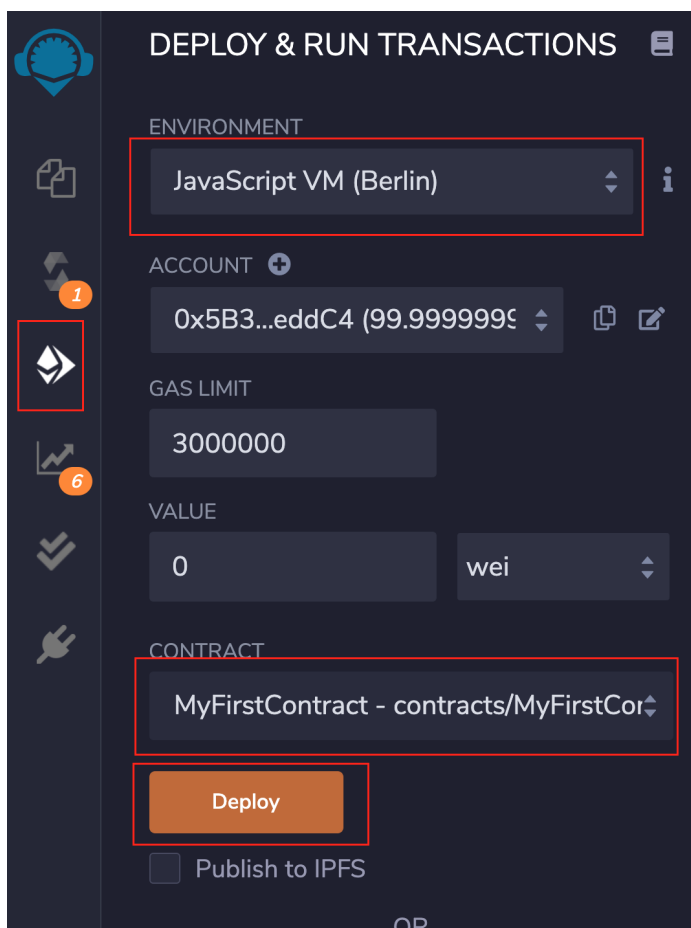
    function getName(uint _index) public view returns (string memory) {
        return names[_index];
    }

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

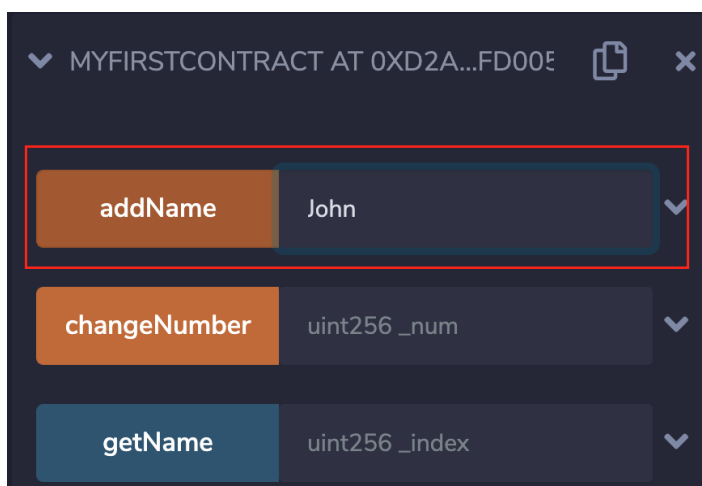
    function getNumber() public view returns (uint256){
        return number;
    }
}
```

6. Ya estás listo para probar tu contrato. Compila y vuelve a re-desplegar. Puedes ignorar cualquier advertencia en tiempo de compilación (texto en naranja). Para este ejercicio, trabajaremos en el entorno EVM basado en el navegador local, así que elige un entorno 'JavaScript VM':

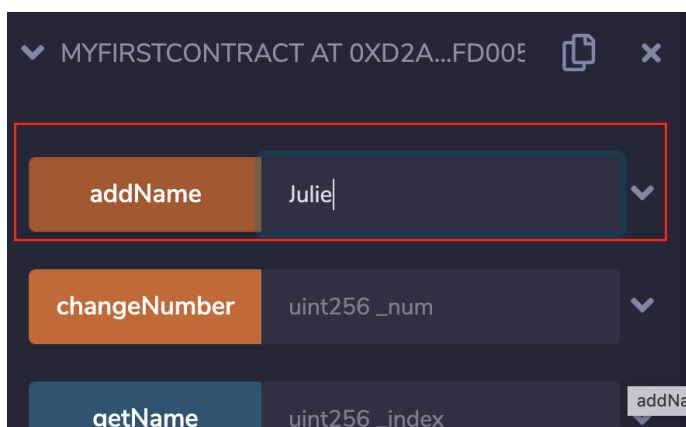




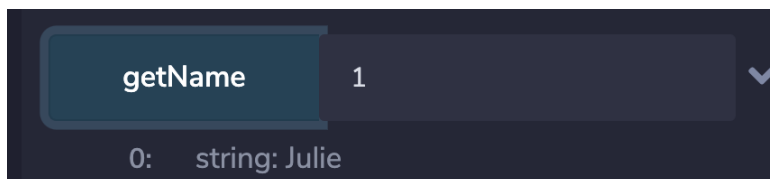
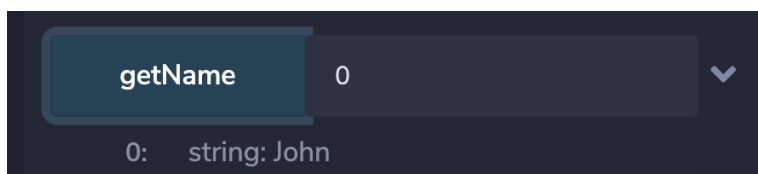
7. Una vez desplegado, pasa un parámetro a la función 'addName', y ejecútala para añadir un nombre al array de nombres en el contrato inteligente



8. Añade otro nombre de tu elección para añadirlo al array de names



- Now call the 'getName' function, pasando un índice de 0 para ver el primer valor almacenado en la matriz. Repite el paso después de esto, pasando un índice de 1 para ver el segundo valor.



Felicitaciones, has creado con éxito un array dinámico en tu contrato inteligente, y lo has rellenado con datos.

Añadiendo un Mapping

Para esta parte del ejercicio, crearás un mapeo de names-> mobile numbers en tu contrato inteligente

- Primero tienes que añadir el nuevo mapeo a tu contrato inteligente. Crea un nuevo mapping de nombres a enteros (numeros de telefono) como sigue:

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

2. Crea una función para añadir valores al mapping. Debe tomar dos parámetros, llamados `_name` y `_mobileNumber`

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

3. Añade otra función para obtener un número de teléfono del mapping para un nombre dado. Debería tomar el parámetro `_name` parameter, y devolver un uint

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

4. Su código completado debería tener ahora el siguiente aspecto

```
pragma solidity ^0.6.7;

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);
    }

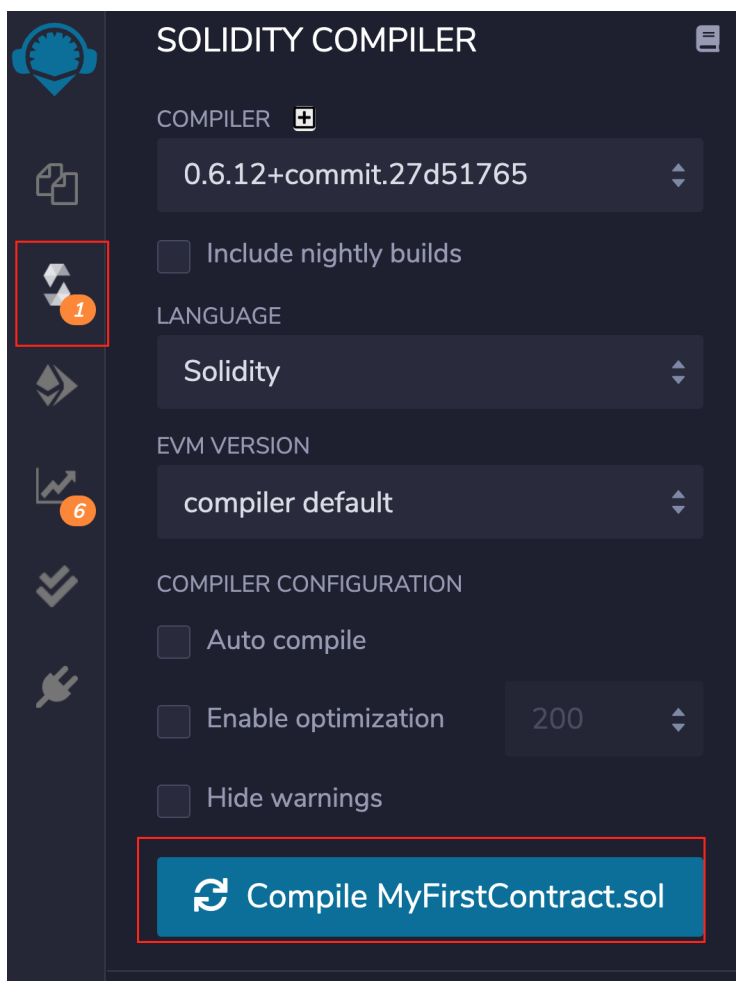
    function getName(uint _index) public view returns (string memory) {
        return names[_index];
    }

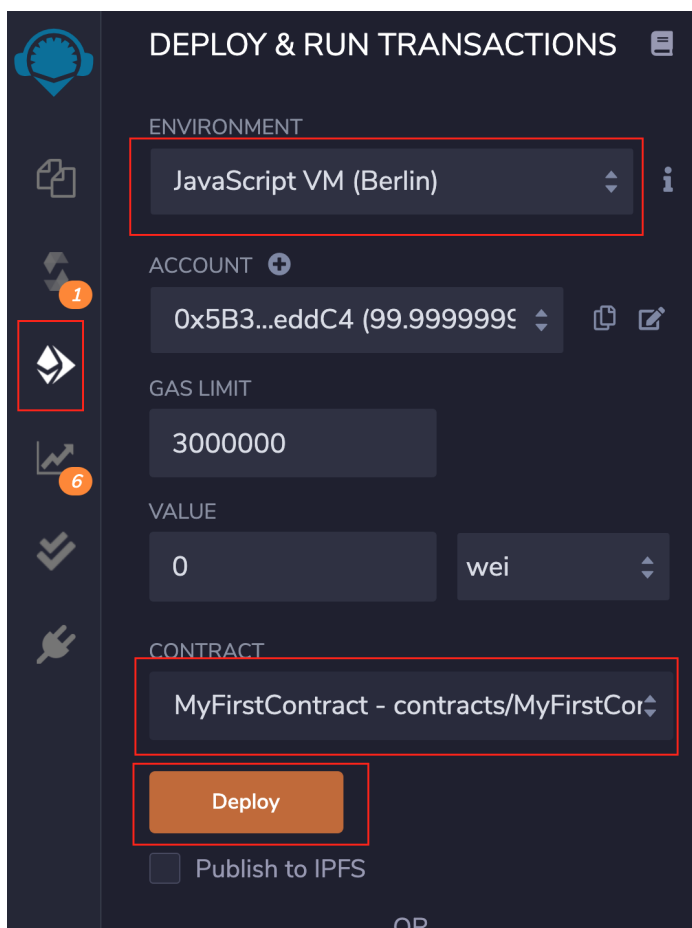
    function changeNumber(uint256 _num) public {
        number = _num;
    }

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

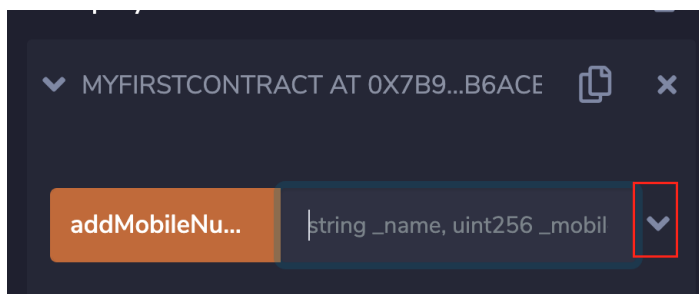
```

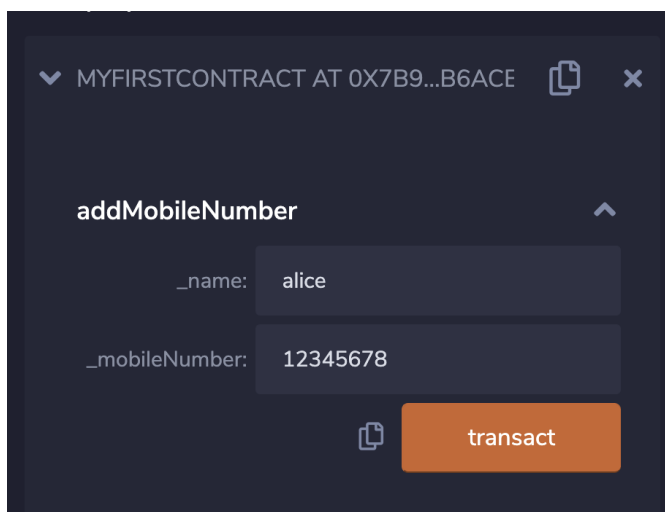
10. Ya estás listo para probar tu contrato. Compila y vuelve a desplegarlo. Puedes ignorar cualquier advertencia en tiempo de compilación (texto en naranja). Para este ejercicio, una vez más trabajaremos en el entorno local de EVM basado en el navegador, así que elige un entorno 'JavaScript VM':





11. Una vez desplegado, expande el botón desplegable 'addMobileNumber', para que puedas ver ambos parámetros. Introduzca algunos valores y ejecute la función





▼ MYFIRSTCONTRACT AT 0X7B9...B6ACE

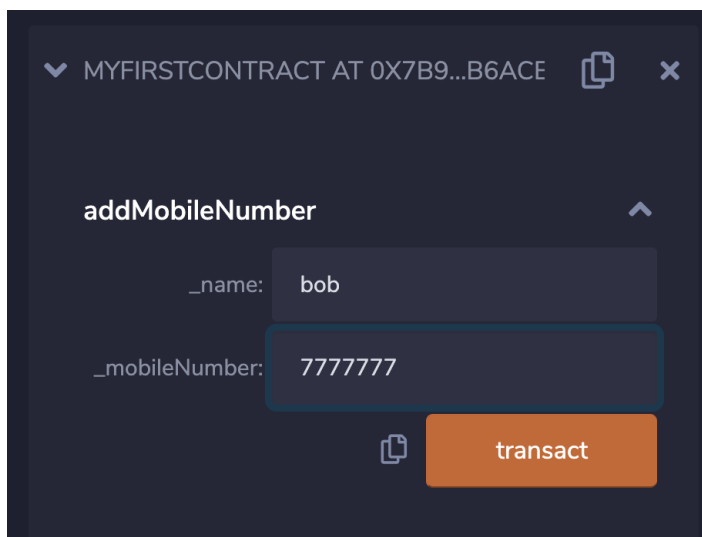
addMobileNumber

_name: alice

_mobileNumber: 12345678

transact

12. Añade otro nombre y número de tu elección, y añádelo al mapping



▼ MYFIRSTCONTRACT AT 0X7B9...B6ACE

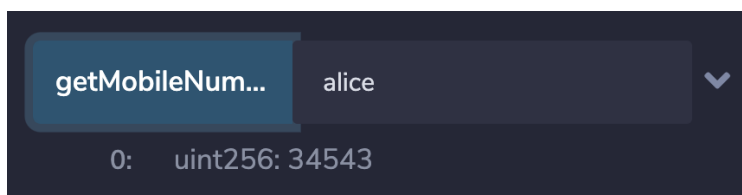
addMobileNumber

_name: bob

_mobileNumber: 7777777

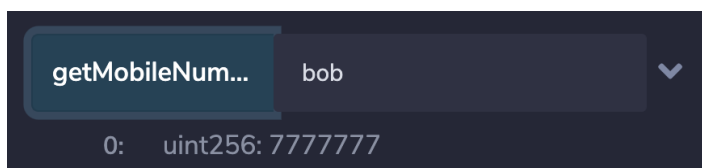
transact

13. Ahora llama a la función 'getMobileNumber', pasando uno de los nombres introducidos en la asignación en los pasos anteriores. Debería ver que devuelve el número almacenado con el nombre. Repita el paso para el otro nombre introducido



getMobileNum... alice

0: uint256: 34543



Felicidades, has creado con éxito un mapeo en tu contrato inteligente, y has poblado e interactuado con él.

Ejercicio Bonus

Puedes intentar completar estos ejercicios si has completado el ejercicio principal antes de lo previsto:

1. Crea una función en tu contrato inteligente que devuelva la longitud del array de nombres, llámala `getNamesLength()`. El tipo de retorno debe ser `uint`, y la forma de obtener la longitud de un array es con la siguiente sintaxis

```
return names.length;
```

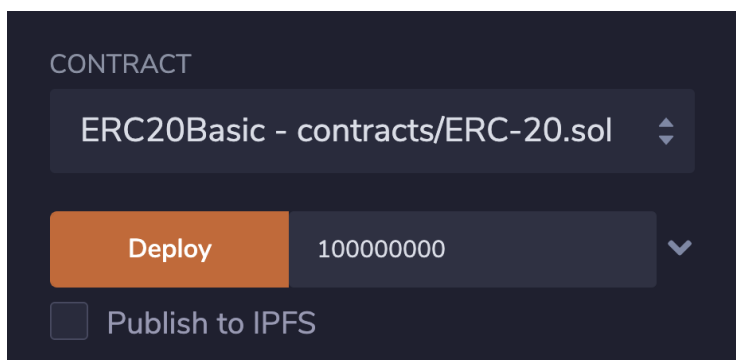
2. Crea una función en tu contrato inteligente que devuelva todo el array de nombres como una cadena. Tendrás que añadir la declaración `pragma ABIEncoderV2` para esto en la parte superior de tu archivo de contrato bajo tu importación. ie:

```
pragma solidity ^0.6.7;
pragma experimental ABIEncoderV2;
```

Llame a la función `getNames`. Debe ser una función de vista, con un tipo de retorno de lo siguiente:

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

3. Crea un nuevo Contrato en Remix (en un nuevo archivo) llamado 'ERC-20.sol'. Pegue el ejemplo de código completo de un contrato ERC-20 básico de la página de [Tutoriales para desarrolladores de Ethereum](#). Modifique los parámetros de nombre y símbolo en el constructor del contrato `ERC20Basic`, elija los valores que desee
4. Despliega el contrato `ERC20Basic` en su red local de VM de JavaScript. Debe especificar el número total de tokens en el contrato cuando lo despliegue. En este ejemplo, hemos elegido 100 millones. Asegúrese de que tiene el contrato correcto seleccionado para desplegar.



5. Una vez desplegado, copie su cuenta actualmente seleccionada en Remix pulsando el botón de copia junto al valor



6. Llama a la función de transferencia en tu contrato desplegado para acuñar algunos tokens, pasando la dirección de tu cartera copiada, y un valor para la cantidad de tokens. A continuación, llama a la función 'balanceOf', pasando de nuevo la dirección de tu cartera copiada, para ver tu saldo

balanceOf 0x5B38Da6a701c568545dC
0: uint256: 100000000

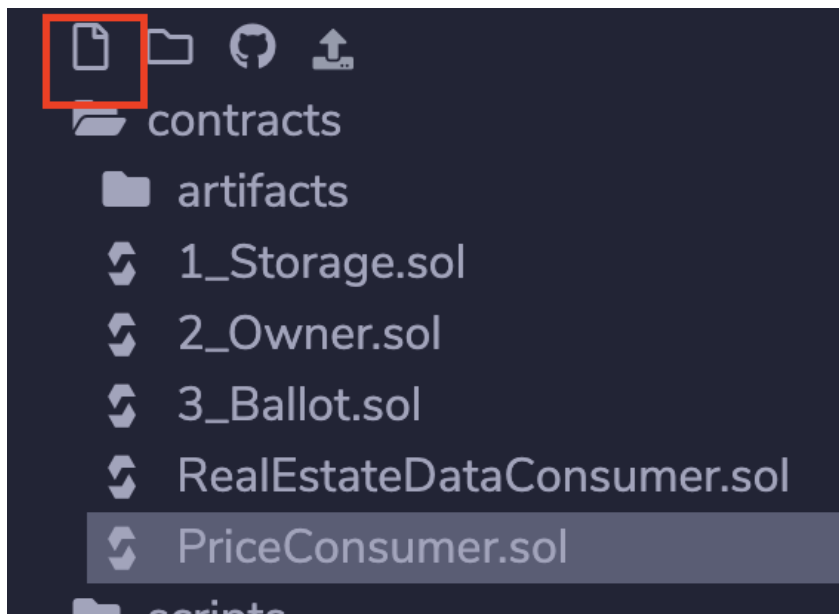
7. Continúa jugando con el contrato de tokens y comprueba cómo funcionan las demás funciones, como 'transferFrom', etc. Puedes cambiar la dirección de tu monedero activo utilizando el menú desplegable "Account" en Remix, para poder enviar cantidades de tokens entre tus cuentas de monedero

Felicidades, has construido, desplegado e interactuado con un contrato de tokens.

Ejercicio 3: Price Feeds

En este ejercicio, vas a crear un simple contrato inteligente que busque el precio actual de Ethereum usando Chainlink Price Feeds. En primer lugar, abre <http://remix.ethereum.org/>

1. Expande la carpeta de contratos, y pulsa el botón de nuevo archivo. Llama al archivo PriceConsumer.sol



2. Seleccione qué versión del compilador vamos a utilizar introduciendo la siguiente línea en el nuevo archivo:

```
pragma solidity ^0.6.7;
```

3. Debajo, introduzca el código fuente del contrato. En nuestro primer ejemplo utilizaremos el contrato ETH/USD price feed **0x9326BFA02ADD2366b30bacB125260Af641031331**, que hemos tomado de la sección Kovan de la página [Ethereum Price Feeds](#) de la documentación de Chainlink

```
import
"@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface
.sol";

contract PriceConsumerV3 {

    AggregatorV3Interface internal priceFeed;

    /**
```

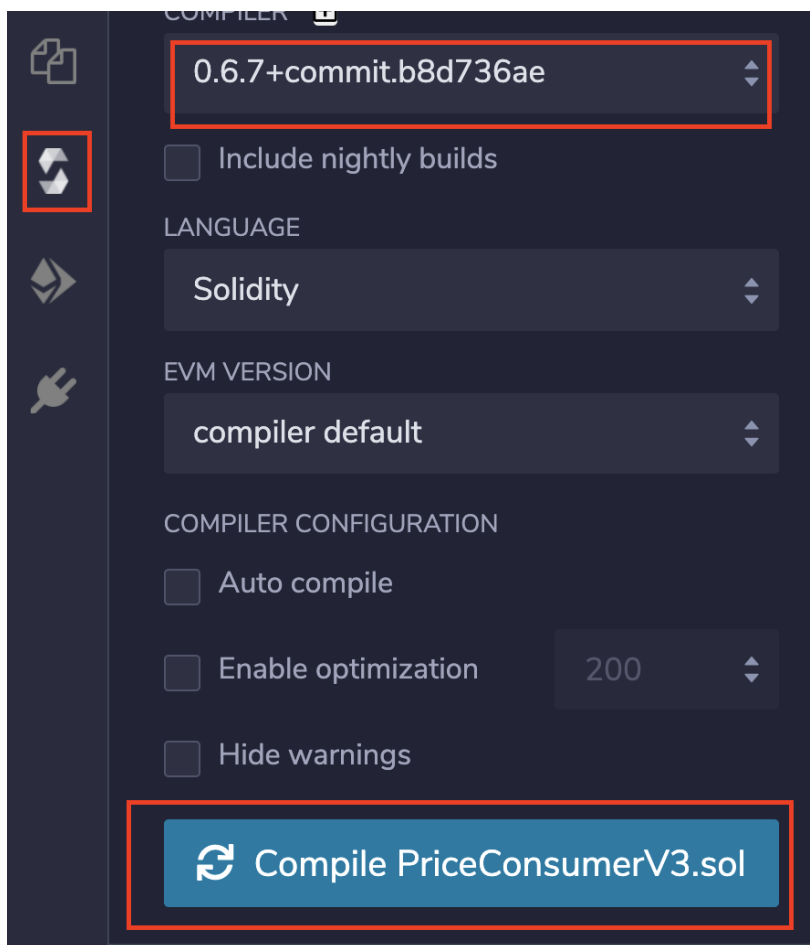
```

    * Network: Kovan
    * Aggregator: ETH/USD
    * Address: 0x9326BFA02ADD2366b30bacB125260Af641031331
    */
    constructor() public {
        priceFeed =
AggregatorV3Interface(0x9326BFA02ADD2366b30bacB125260Af64103133
1);
    }

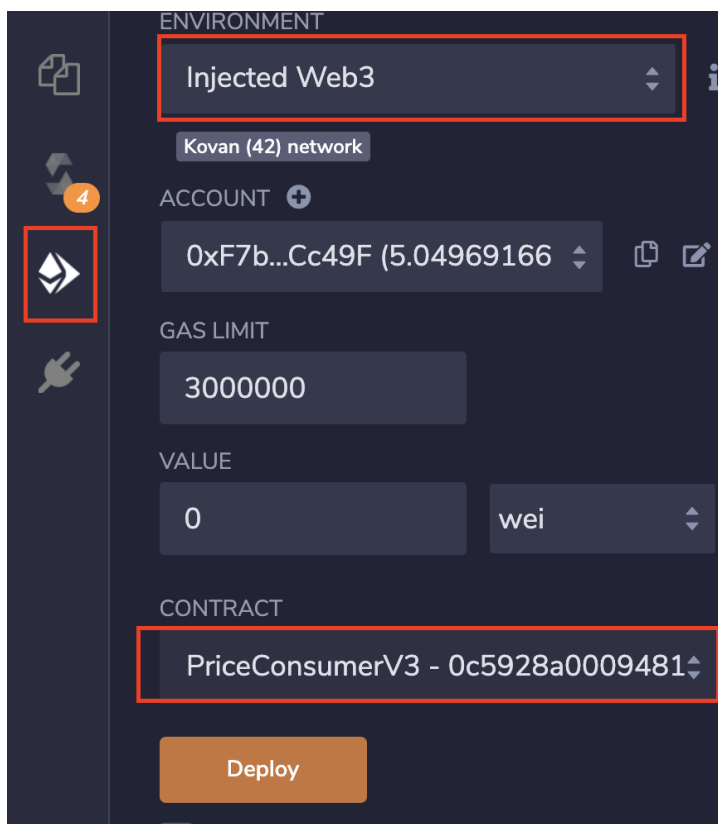
    /**
    * Returns the latest price
    */
    function getLatestPrice() public view returns (int) {
        (
            uint80 roundID,
            int price,
            uint startedAt,
            uint timeStamp,
            uint80 answeredInRound
        ) = priceFeed.latestRoundData();
       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

```

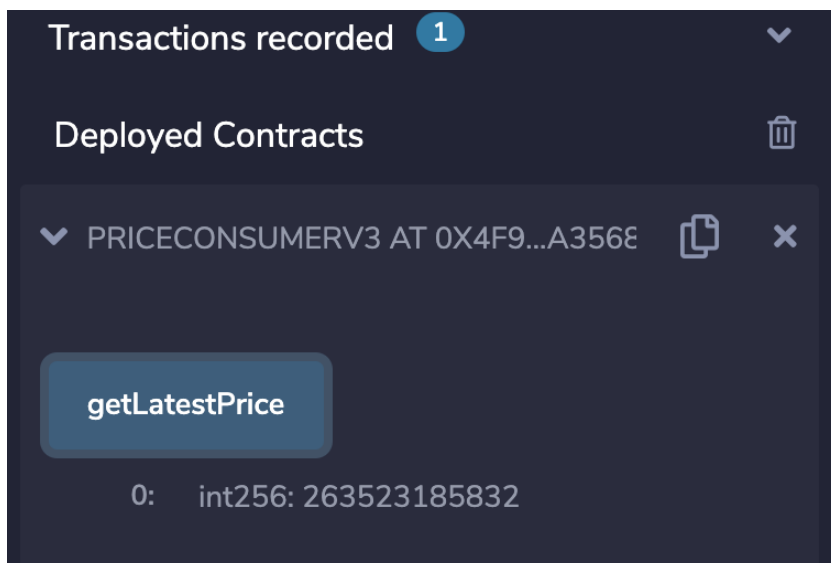
4. Su contrato está ahora listo para ser compilado. Pulse la opción de menú Solidity Compiler en el menú de la izquierda, cambie la versión del compilador en el desplegable del compilador para que coincida con el compilador especificado en su código, y luego pulse el botón azul Compile:



5. Elija el botón 'Deploy and Run Transactions' en el menú de la izquierda. Como se va a integrar a la red Chainlink Oracle, necesitamos conectarnos a la red pública de pruebas de Kovan. Asegúrese de que el Entorno está configurado como 'Injected Web3', y que el contrato seleccionado es 'PriceConsumer.sol', luego pulse el botón Deploy. Aparecerá una ventana emergente en la que Metamask le pedirá que confirme la operación. Pulse el botón azul "Confirmar" para ejecutar la transacción y deployar su contrato en la red de Kovan. Después de unos segundos, debería ver su contrato desplegado en la sección 'Deployed Contracts' en Remix.



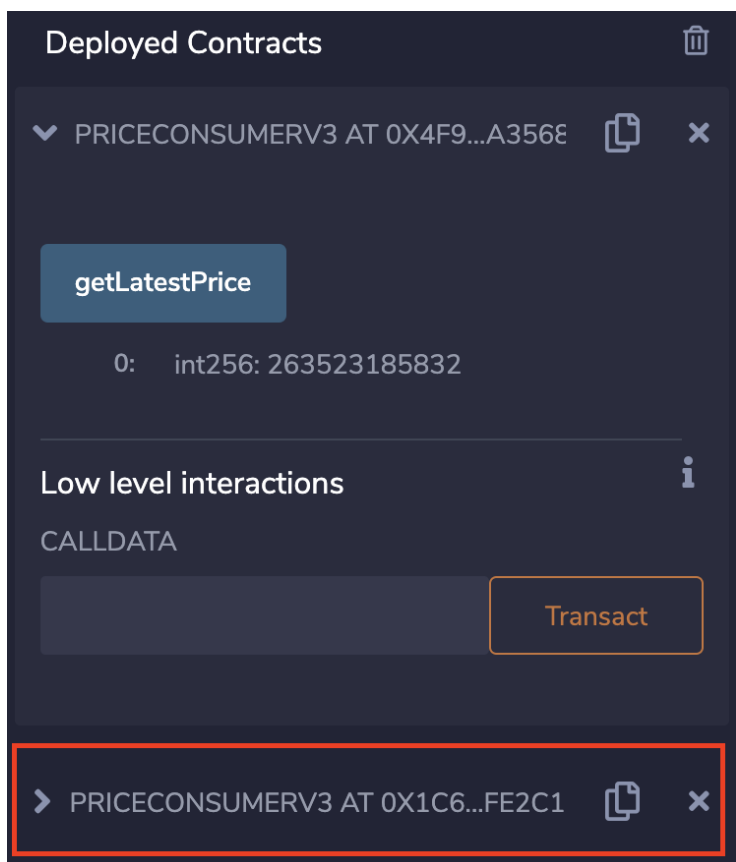
- Despliegue el contrato desplegado pulsando el icono ">" junto al nombre del contrato en la sección de Contratos desplegados. Debería ver las funciones disponibles que se pueden ejecutar. Pulse el botón "getLatestPrice" para buscar el precio de ETH en el feed de precios de Kovan ETH/USD.



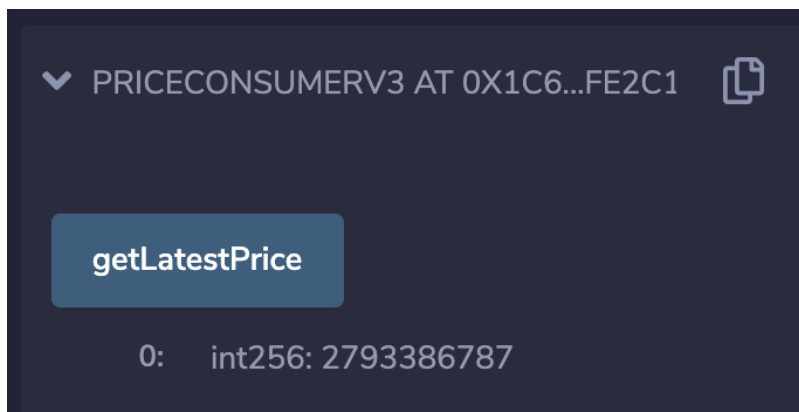
7. El resultado devuelto es el precio de mercado actual de ETH, multiplicado por 10^8 . Puedes compararlo con el feed de precios de ETH/USD de Mainnet yendo a <https://data.chain.link/eth-usd> y mirando el precio actual.
8. Vaya a la página de Ethereum Price Feeds en la documentación de Chainlink, y navegue hasta la sección 'Kovan Testnet'. Elija otro par y copie la dirección del proxy. En nuestro ejemplo, hemos elegido el par LINK/USD
0x396c5E36DD0a0F5a5D33dae44368D4193f69a1F0
9. Vuelve a tu contrato, y en el constructor, sustituye la dirección del feed de precios ETH/USD, por la que has copiado en el paso anterior.

```
priceFeed =  
AggregatorV3Interface(0x396c5E36DD0a0F5a5D33dae44368D4193f69a1F  
0);
```

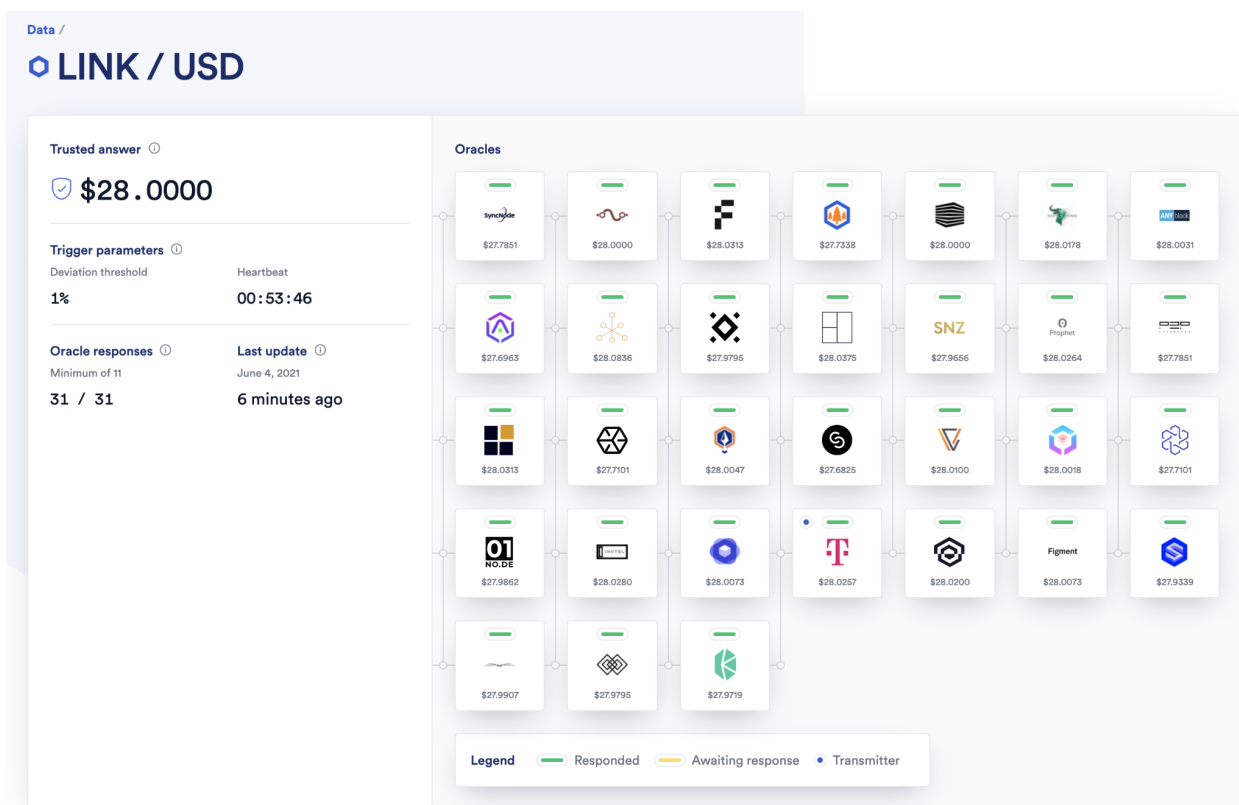
10. Vuelva a la pestaña Solidity Compiler en Remix, y pulse el botón azul 'Compile PriceConsumer.sol' para volver a compilar su contrato.
11. Vaya a la pestaña 'Deploy & Run Transactions' y vuelva a desplegar su contrato, aceptando la transacción en Metamask. Una vez desplegado, debería ver un segundo contrato desplegado en la sección 'Deployed Contracts'.



12. Despliegue el contrato desplegado una vez más y ejecute la función "getLatestPrice". Debería ver el precio actual devuelto en el contrato de alimentación de precios que ha seleccionado.



13. Dirijase a <https://data.chain.link/> y busque la fuente de precios equivalente que se ejecuta en Mainnet.



Felicidades, ha escrito y desplegado con éxito un contrato que utiliza las fuentes de precios de Chainlink para obtener datos de precios externos.

Ejercicio Bonus

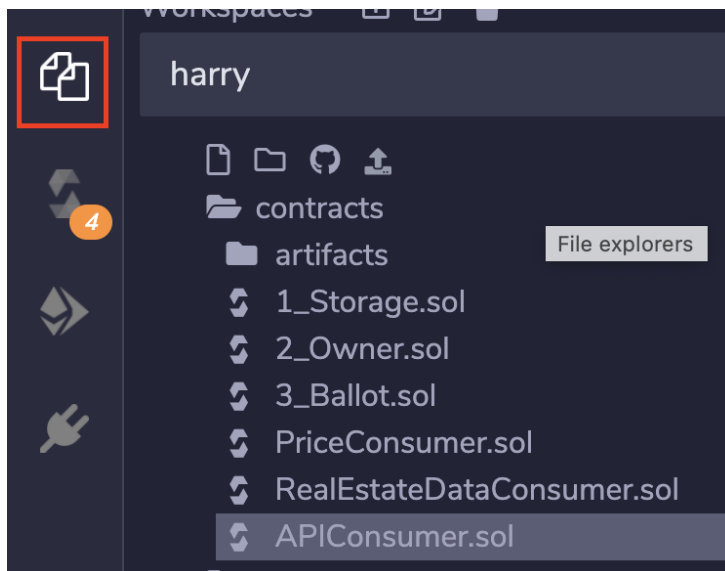
Puede intentar completar estos ejercicios si ha completado el ejercicio principal antes de lo previsto:

1. Modifique el contrato del Price Feed para obtener una alimentación de precios diferente que elija de las direcciones de Price Feed en la [Documentación de Chainlink](#). A continuación, despliegue y pruebe su contrato. Compare el valor con el valor de la Feed de la red principal en <https://data.chainlink>.
2. Cree una nueva función llamada 'getTimestamp' que devuelva un uint que es el timeStamps de la ronda actual (pista: se puede acceder a través de la llamada `priceFeed.latestRoundData()`)

Ejercicio 4: Any-API

En este ejercicio, vas a crear un contrato inteligente simple que realiza una solicitud de datos externos, utilizando Chainlink Any-API. En primer lugar, abre <http://remix.ethereum.org/>

1. Expande la carpeta de contratos, y pulsa el botón new file. Llamar al archivo APIConsumer.sol



2. Seleccione qué versión del compilador vamos a utilizar introduciendo la siguiente línea en el nuevo archivo:

```
pragma solidity ^0.6.7;
```

3. Ve a la siguiente URL en tu navegador, para ver la salida json de la API a la que vamos a acceder. Busque el término "**VOLUME24HOUR**" para ver el valor exacto que vamos a buscar en nuestro contrato inteligente

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

4. De vuelta en Remix, debajo de tu comando pragma, introduce el código fuente del contrato.

```
import "@chainlink/contracts/src/v0.6/ChainlinkClient.sol";

contract APIConsumer is ChainlinkClient {
```

```
uint256 public volume;

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

/**
 * Network: Kovan
 * Chainlink - 0x2f90A6D021db21e1B2A077c5a37B3C7E75D15b7e
 * Chainlink - 29fa9aa13bf1468788b7cc4a500a45b8
 * Fee: 0.1 LINK
 */
constructor() public {
    setPublicChainlinkToken();
    oracle = 0x2f90A6D021db21e1B2A077c5a37B3C7E75D15b7e;
    jobId = "29fa9aa13bf1468788b7cc4a500a45b8";
    fee = 0.1 * 10 ** 18; // 0.1 LINK
}

/**
 * Create a Chainlink request to retrieve API response,
find the target
 * data, then multiply by 1000000000000000000 (to remove
decimal places from data).

*****
*****
          *                               STOP!
*
          *   THIS FUNCTION WILL FAIL IF THIS CONTRACT DOES
NOT OWN LINK      *
          *
-----
*
          *   Learn how to obtain testnet LINK and fund this
contract:              *
          *   -----
https://docs.chain.link/docs/acquire-link -----
*
          *   ----
https://docs.chain.link/docs/fund-your-contract -----
*
          *
*
```

```

*****
*****/
    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=ET
H&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)
    {

```

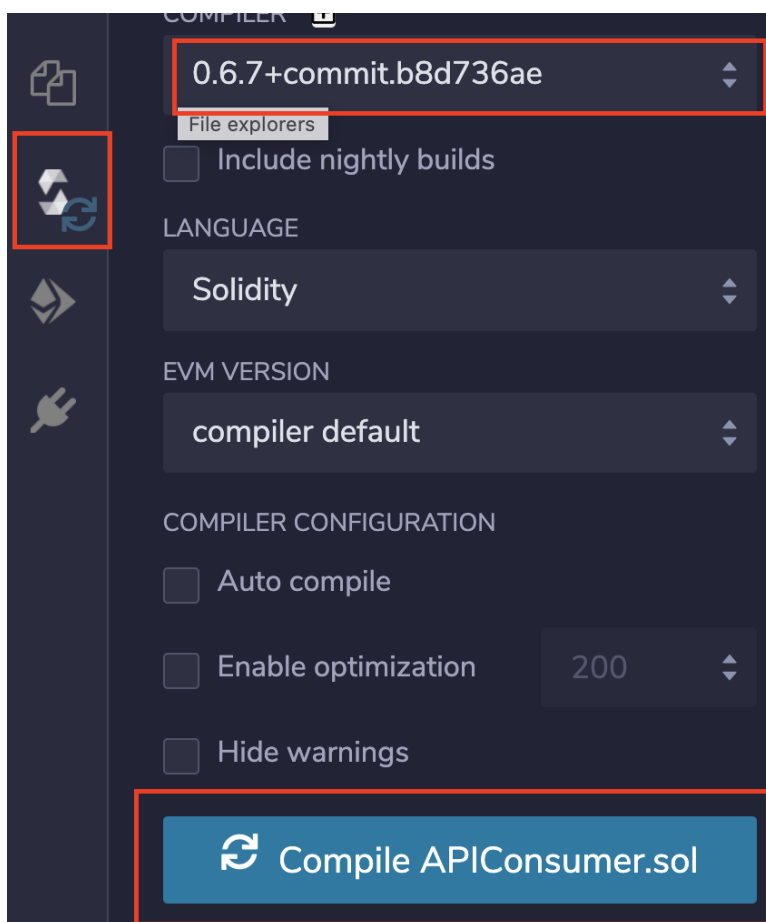
```

        volume = _volume;
    }

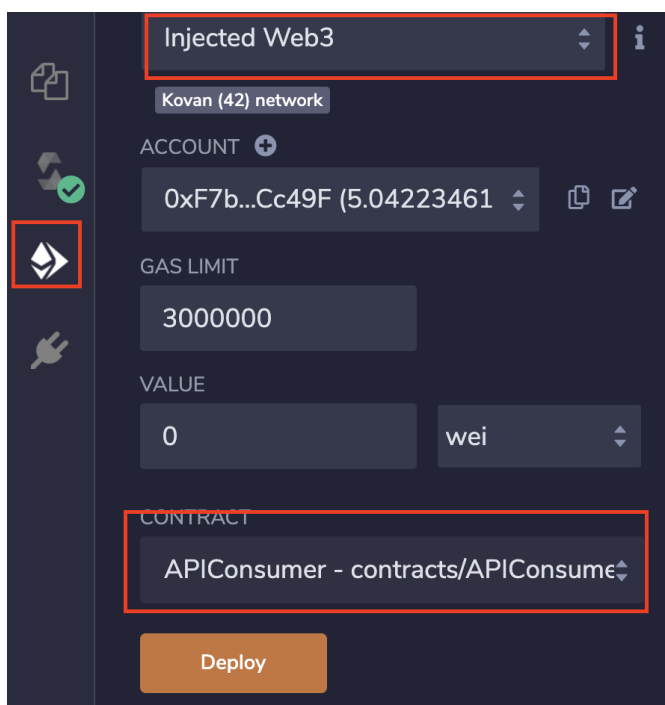
    /**
     * Withdraw LINK from this contract
     *
     * NOTE: DO NOT USE THIS IN PRODUCTION AS IT CAN BE CALLED
    BY ANY ADDRESS.
     * THIS IS PURELY FOR EXAMPLE PURPOSES ONLY.
     */
    function withdrawLink() external {
        LinkTokenInterface linkToken =
    LinkTokenInterface(chainlinkTokenAddress());
        require(linkToken.transfer(msg.sender,
    linkToken.balanceOf(address(this))), "Unable to transfer");
    }
}

```

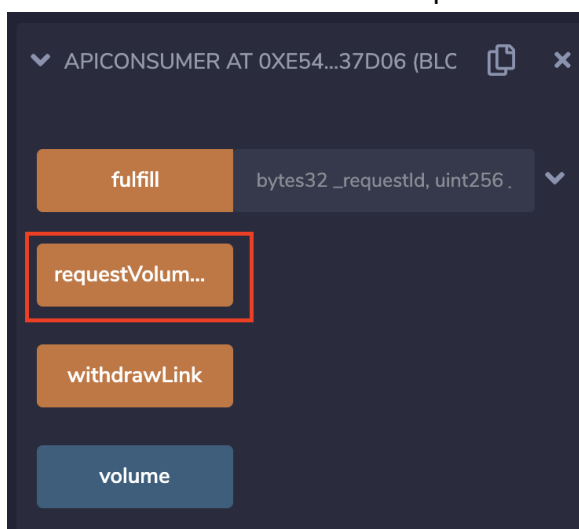
5. Su contrato está ahora listo para ser compilado. Pulse la opción de menú Solidity Compiler en el menú de la izquierda, cambie la versión del compilador en el desplegable del compilador para que coincida con el compilador especificado en su código, y luego pulse el botón azul Compile:



6. Elija el botón 'Deploy and Run Transactions' en el menú de la izquierda. Como se va a integrar a la red Chainlink Oracle, necesitamos conectarnos a la red pública de pruebas de Kovan. Asegúrese de que el Entorno está configurado como 'Injected Web3', y que el contrato seleccionado es APIConsumer.sol', luego pulse el botón Deploy. Aparecerá una ventana emergente en la que se le pedirá que confirme la operación. Pulse el botón azul "Confirmar" para ejecutar la transacción y desplegar su contrato en la red de Kovan. Después de unos segundos, debería ver su contrato desplegado en la sección "Deployed Contracts" en Remix.



7. [Obtenga 1 token LINK de fondo en su contrato](#). Si aún no tienes el token LINK añadido a tu cartera MetaMask, pulsa el botón "Añadir token" en la parte inferior, luego introduce la dirección del contrato con el token LINK **0xa36085F69e2889c224210F603D836748e7dC0088**, y luego pulsa guardar.
8. Una vez que su contrato ha sido financiado con LINK, ahora puede iniciar la solicitud de datos externos. Despliegue el contrato desplegado pulsando el icono ">" junto al nombre del contrato en la sección Deployed Contracts. Debería ver las funciones disponibles que se pueden ejecutar. Pulse sobre el botón "requestVolumeData" y confirme la transacción cuando aparezca MetaMask.



9. Después de unos segundos, su transacción debería ser aprobada. Mientras esperas a que el nodo chainlink cumpla con la solicitud, ve a <https://kovan.etherscan.io/> y busca la dirección de tu contrato desplegado (la copiaste en el portapapeles cuando financiaste tu contrato con LINK). Una vez que encuentres tu contrato desplegado en Etherscan, ve a la pestaña 'ERC-20 Token Txns'. Deberías ver 2 transacciones, una transferencia entrante de 1 LINK de cuando financiaste tu contrato, y 0,1 LINK transferido fuera del contrato para tu solicitud de datos externa.

Contract 0xe54092f4a27442FEbb43C011B5c2aDD942537D06

Contract Overview

Balance: 0 Ether

Token: \$0.00

More Info

My Name Tag: Not Available

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

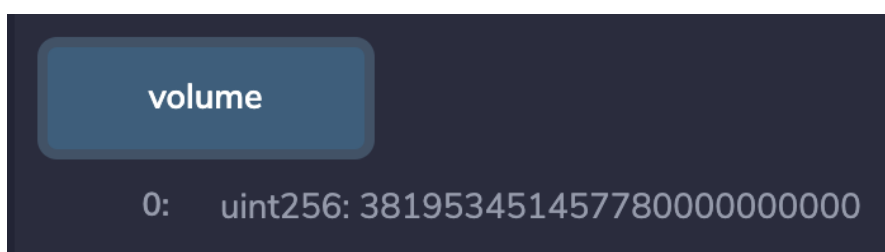
ERC20 Token Txns

Latest 2 ERC-20 Token Transfer Events

Txn Hash	Age	From	To	Value	Token
0x9bdc2fd5fab01e37b51...	1 min ago	0xe54092f4a27442febb4...	0x2f90a6d021db21e1b2...	0.1	ChainLink To... (LINK)
0xed05b78ee2cd7475e1...	3 mins ago	0x7b4ef69e7cf13c2055...	0xe54092f4a27442febb4...	1	ChainLink To... (LINK)

[Download CSV Export]

10. Vuelva a su contrato desplegado en Remix. Ahora comprobaremos el resultado devuelto de la solicitud de datos externa. Pulse el botón 'volumen' para ejecutar la función que devuelve el valor de la variable pública 'volumen'. Ahora debería tener un resultado que coincide con el valor que vio en su navegador web en el paso 19 anterior. El valor se ha multiplicado a propósito por 10^{18} para evitar cualquier problema con las operaciones matemáticas, etc., que puedan realizarse en el resultado.



Felicidades, has realizado con éxito una solicitud de datos externos y has introducido datos de una API pública en tu contrato inteligente.

Ejercicio Bonus

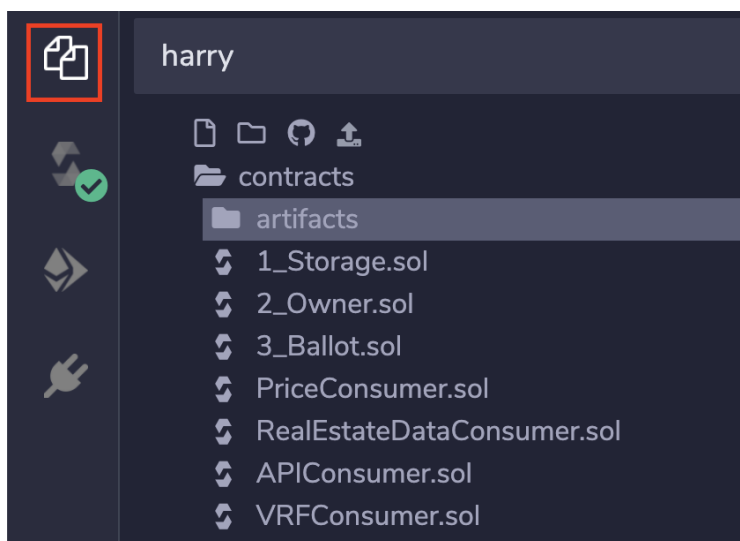
Puedes intentar completar estos ejercicios si has completado el ejercicio principal antes de lo previsto

1. Modifica tu contrato para obtener los datos del **precio** de la API de cryptocompare en lugar del volumen. Cambie el nombre de todas las variables que mencionan el volumen para reflejar el cambio en los datos que se traen
2. Cambie la API a la que se accede por la siguiente, para obtener el precio BTC/USD. <https://api.cryptowat.ch/markets/kraken/btcusd/price> Tendrás que modificar también el parámetro 'path' enviado en la petición para que coincida con la ruta JSON del campo 'price' si pegas la URL anterior en una ventana del navegador

Ejercicio 5: VRF

En este ejercicio, vas a crear un contrato inteligente simple que realiza una solicitud de aleatoriedad utilizando Chainlink VRF. En primer lugar, abre <http://remix.ethereum.org/>

11. Expande la carpeta de contratos, y pulsa el botón new fil. Llama al archivo VRFConsumer.sol



12. Selecciona qué versión del compilador vamos a utilizar introduciendo la siguiente línea en el nuevo archivo:

```
pragma solidity ^0.6.7;
```

13. Debajo, introduce el código fuente del contrato

```
import "@chainlink/contracts/src/v0.6/VRFConsumerBase.sol";

/**
```



```

* THIS IS AN EXAMPLE CONTRACT WHICH USES HARDCODED VALUES FOR
CLARITY.
* PLEASE DO NOT USE THIS CODE IN PRODUCTION.
*/
contract RandomNumberConsumer is VRFConsumerBase {

    bytes32 internal keyHash;
    uint256 internal fee;

    uint256 public randomResult;

    /**
     * Constructor inherits VRFConsumerBase
     *
     * Network: Kovan
     * Chainlink VRF Coordinator address:
0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9
     * LINK token address:
0xa36085F69e2889c224210F603D836748e7dC0088
     * Key Hash:
0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b64
1f4
     */
    constructor()
        VRFConsumerBase(
            0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9, // VRF
Coordinator
            0xa36085F69e2889c224210F603D836748e7dC0088 // LINK
Token
        ) public
    {
        keyHash =
0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b64
1f4;
        fee = 0.1 * 10 ** 18; // 0.1 LINK (Varies by network)
    }

    /**
     * Requests randomness
     */
    function getRandomNumber() public returns (bytes32
requestId) {
        require(LINK.balanceOf(address(this)) >= fee, "Not
enough LINK - fill contract with faucet");
        return requestRandomness(keyHash, fee);
    }
}

```

```

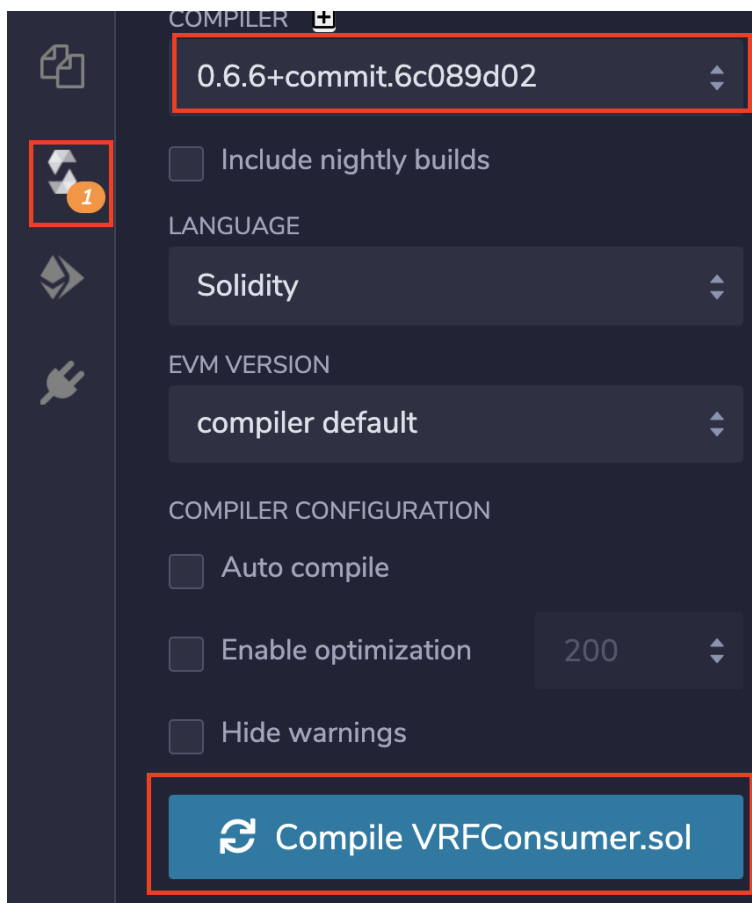
    }

    /**
     * Callback function used by VRF Coordinator
     */
    function fulfillRandomness(bytes32 requestId, uint256
randomness) internal override {
        randomResult = randomness;
    }

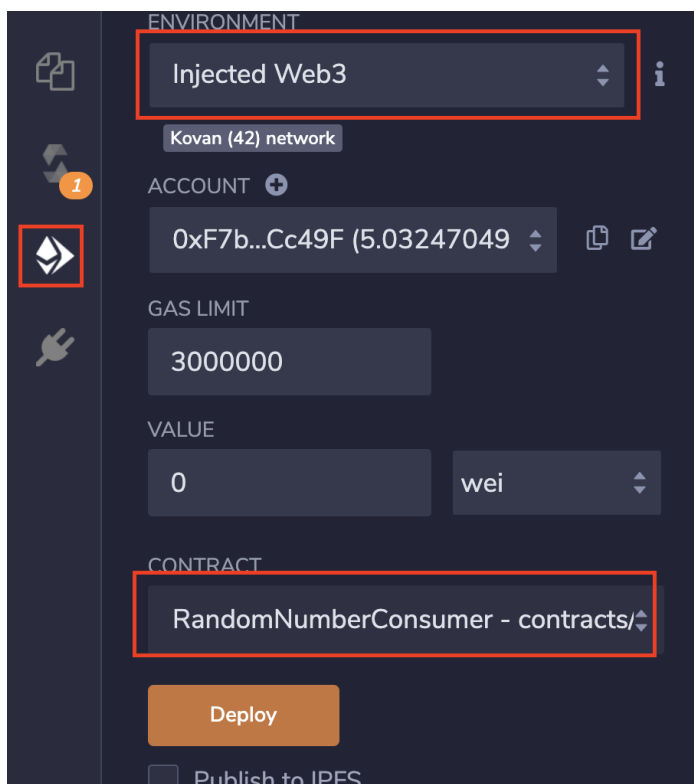
    // function withdrawLink() external {} - Implement a
withdraw function to avoid locking your LINK in the contract
}

```

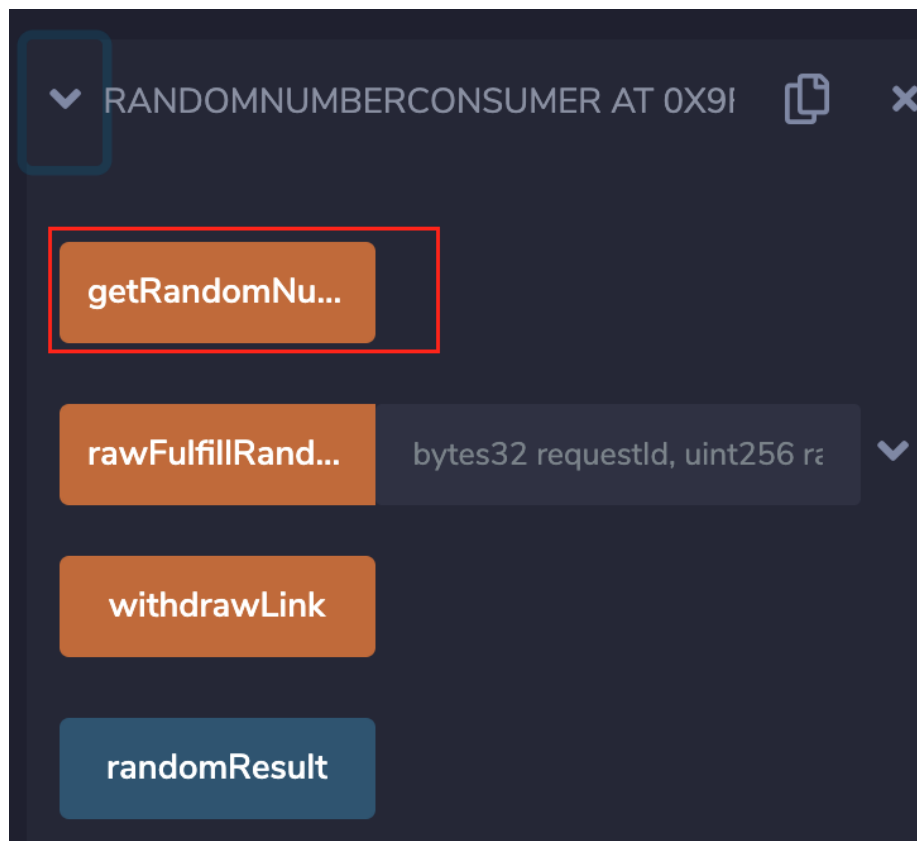
14. Su contrato está ahora listo para ser compilado. Pulse la opción de menú Solidity Compiler en el menú de la izquierda, cambie la versión del compilador en el desplegable del compilador para que coincida con el compilador especificado en su código, luego pulse el botón azul Compile:



15. Elija el botón 'Deploy and Run Transactions' en el menú de la izquierda. Como se va a integrar a la red Chainlink Oracle, necesitamos conectarnos a la red pública de pruebas de Kovan. Asegúrese de que el Entorno está configurado como 'Injected Web3', y que el contrato seleccionado es 'RandomNumberConsumer', luego pulse el botón Deploy. Aparecerá una ventana emergente en la que se le pedirá que confirme la operación. Pulse el botón azul "Confirmar" para ejecutar la transacción y desplegar su contrato en la red de Kovan. Después de unos segundos, debería ver su contrato desplegado en la sección 'Deployed Contracts' en Remix.



16. [Obtenga 1 token LINK de fondo en su contrato.](#)
17. Una vez que su contrato ha sido financiado con LINK, ahora puede iniciar la solicitud de aleatoriedad. Amplíe el contrato desplegado pulsando el icono ">" junto al nombre del contrato en la sección de Contratos Desplegados. Debería ver las funciones disponibles que se pueden ejecutar. Pulse sobre el botón getRandomNumber, y confirme la transacción cuando aparezca MetaMask.



18. Después de unos segundos, su transacción debería ser aprobada. Mientras esperas a que el nodo chainlink cumpla con la solicitud, ve a <https://kovan.etherscan.io/> y busca la dirección de tu contrato desplegado (la copiaste en el portapapeles cuando financiaste tu contrato con LINK). Una vez que encuentres tu contrato desplegado en Etherscan, ve a la pestaña 'ERC-20 Token Txns'. Deberías ver 2 transacciones, una transferencia entrante de 1 LINK de cuando financiaste tu contrato, y 0,1 LINK transferido fuera del contrato para tu solicitud de datos externa.

Contract 0x39dd0cb6528a512D4EB6C9B6Ce0B963B4F02Cb45

Contract Overview

Balance: 0 Ether

Token: \$0.00

More Info

My Name Tag: Not Available

Contract Creator: 0xf7b4ef69e7cf13c2055... at txn 0xad2cfc1e6835bd0cb...

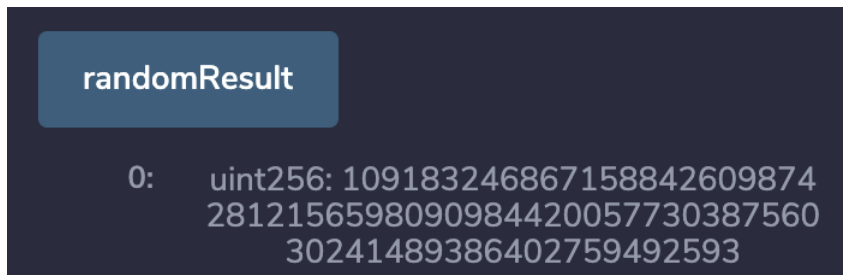
Transactions **ERC20 Token Txns** Contract Events

Latest 2 ERC-20 Token Transfer Events

Txn Hash	Age	From	To	Value	Token
0x39db15f9a490588539...	8 secs ago	0x39dd0cb6528a512d4e...	0xdd3782915140c8f3b1...	0.1	ChainLink To... (LINK)
0xeba02ced1d98b57b8d...	1 min ago	0xf7b4ef69e7cf13c2055...	0x39dd0cb6528a512d4e...	1	ChainLink To... (LINK)

[\[Download CSV Export\]](#)

19. Vuelva a su contrato desplegado en Remix. Ahora comprobaremos el número aleatorio devuelto. Pulse el botón `randomResult` para ejecutar la función que devuelve el valor de la variable pública `'randomResult'`. Ahora deberías tener un número aleatorio verificado en tu contrato inteligente.



20. Toma nota de tu número aleatorio y repite el paso 17. Cuando recibas un nuevo resultado, deberías ver que obtenemos un número aleatorio diferente.

Felicidades, ¡has realizado con éxito una petición de aleatoriedad en un contrato inteligente!

Ejercicio Bonus

Puedes intentar completar estos ejercicios si has completado el ejercicio principal antes de tiempo

1. Modifica tu contrato para que el número aleatorio almacenado sea siempre un número entre 1 y 100. Para ello, puedes utilizar el operador módulo como se define en el fragmento de código siguiente. Vuelve a desplegar y prueba tu contrato para ver qué números aleatorios devuelve ahora.

```
randomness.mod(100).add(1)
```