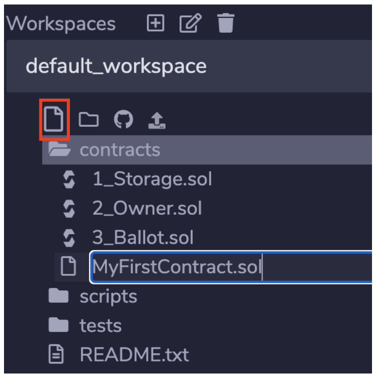
Smart Contract Developer Bootcamp: Weekend Track Day 1 Exercises

Exercise 1: My First Smart Contract

In this exercise, you'll create a simple smart contract that stores and retrieves a value, then you'll deploy it to the Kovan testnet and interact with it using Remix.

1. Open http://remix.ethereum.org/
2. If prompted, create a new default workspace
3. Expand the contracts folder, and press the new file button. Call the file MyFirstContract.sol



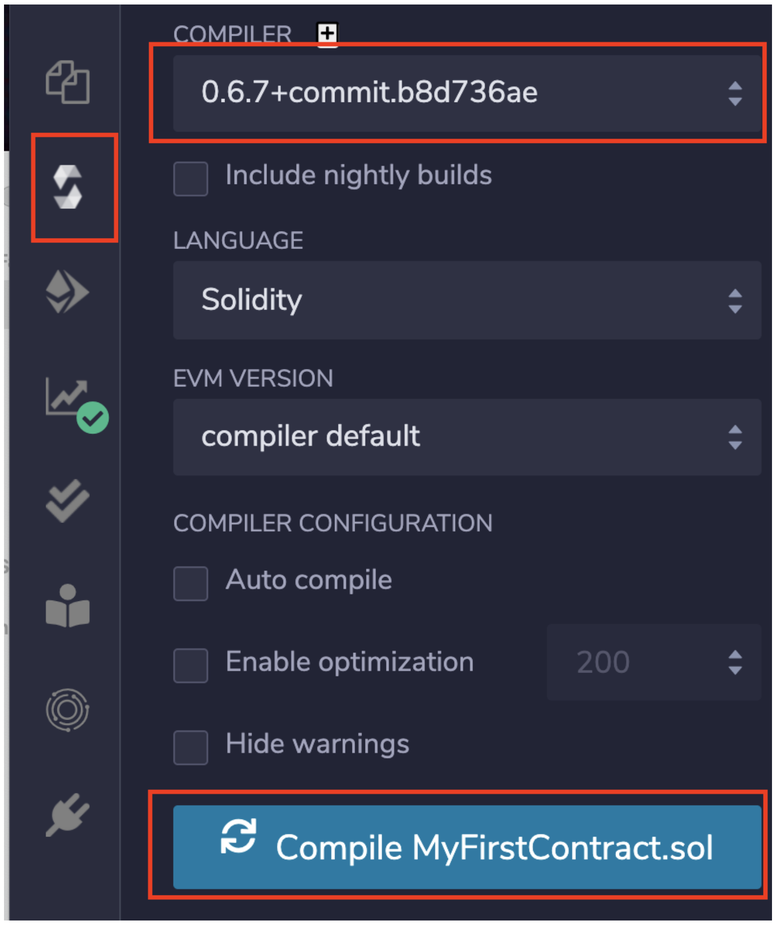
4. Select which version of the compiler we will be using by entering the following line into the new file:

```
pragma solidity ^0.6.7;
```

5. Underneath, enter in the contract source code:

```
contract MyFirstContract {

    uint256 number;


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


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

6. Your contract is now ready to be compiled. Press on the Solidity Compiler menu option on the left hand side menu, change the compiler version in the compiler dropdown so that it matches the compiler specified in your code, then press on the blue Compile button:



COMPILER

0.6.7+commit.b8d736ae

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

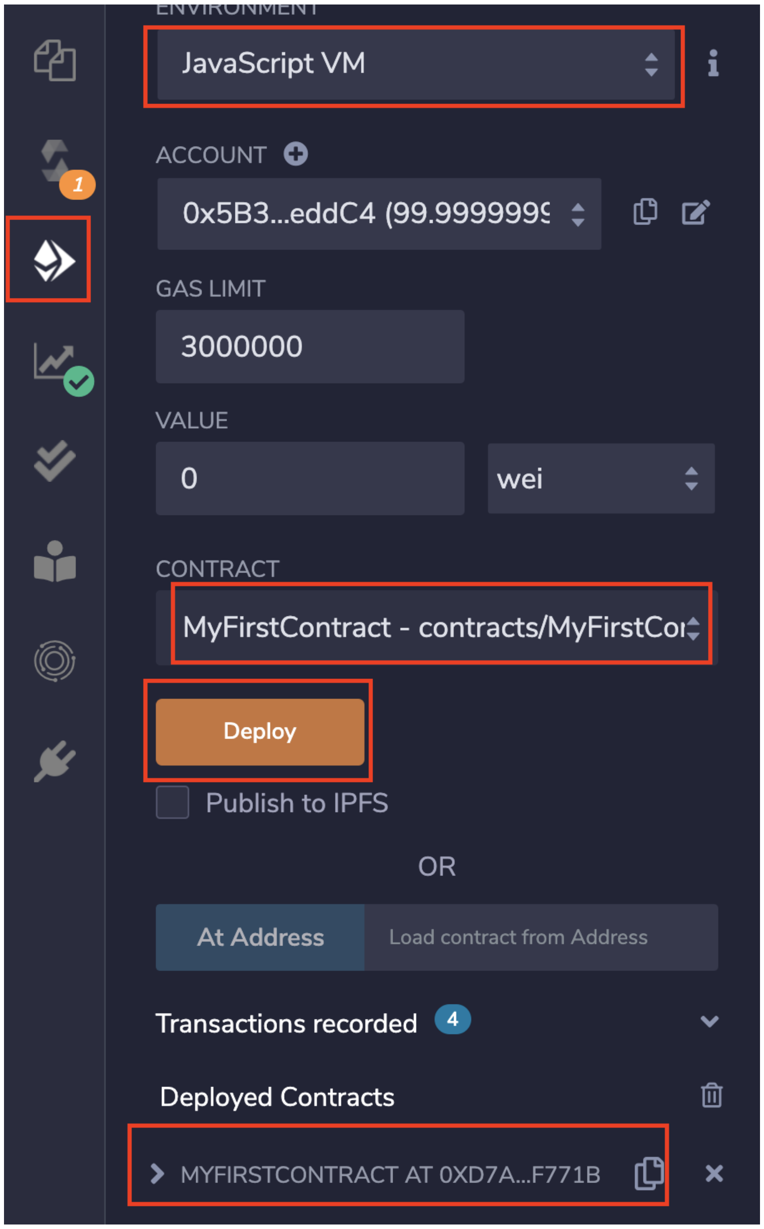
COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

7. Choose the ‘Deploy and Run Transactions’ button on the left hand side menu. Ensure Environment is set to ‘JavaScript VM’, and that the selected contract is ‘MyFirstContract’, then press the Deploy button. You should then see your contract come up under the ‘Deployed Contracts’ section further below


ENVIRONMENT

JavaScript VM

ACCOUNT

0x5B3...eddC4 (99.9999999

GAS LIMIT

3000000

VALUE

0 wei

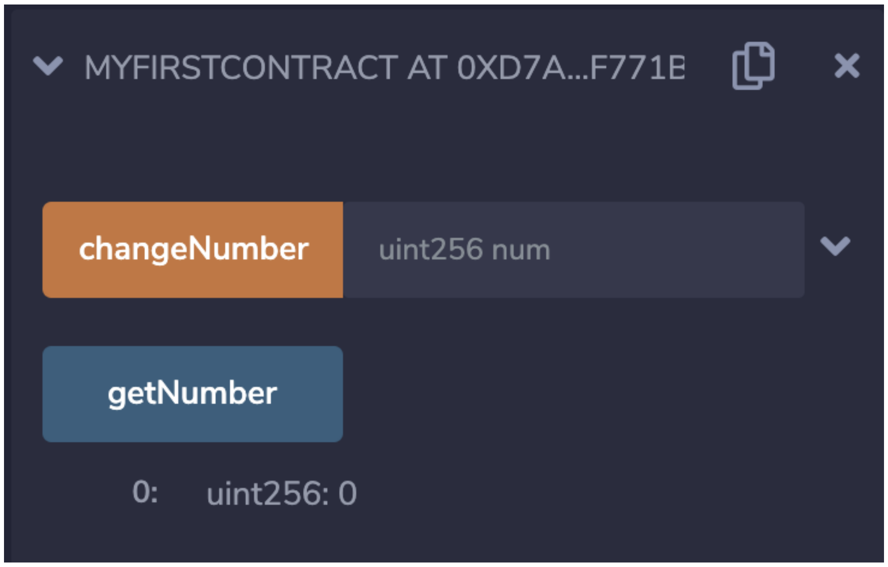
CONTRACT

MyFirstContract - contracts/MyFirstCo

MYFIRSTCONTRACT AT 0XD7A...F771B

8. Expand the deployed contract by selecting the > sign next to it. You can now see all of the functions that you can use to interact with the contract.

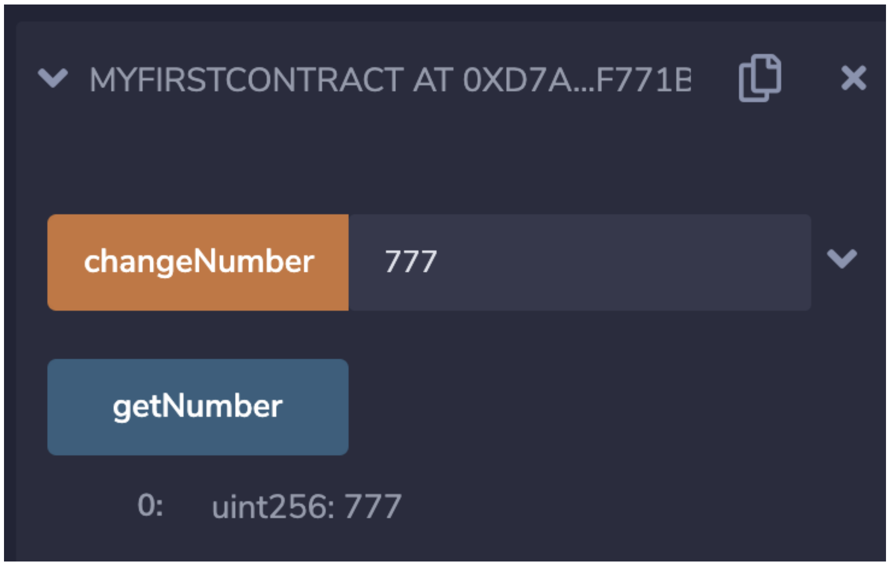
9. Press the ‘getNumber’ function to read the state of the contract, and return the value of the number variable. Because we haven’t set it yet, it should simply return 0



```
0:    uint256: 0
```

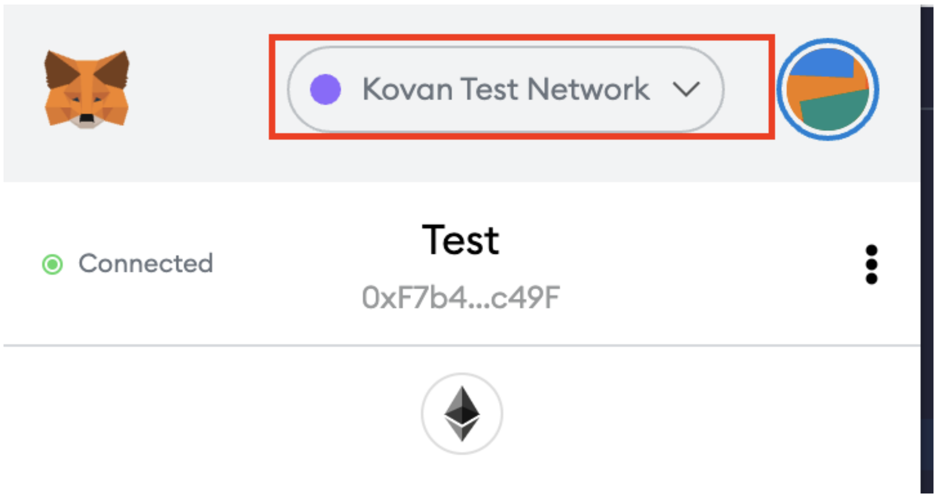
10. Enter a value into the ‘changeNumber’ function input parameter, then press the ‘changeNumber’ button to execute the function.

11. Execute the ‘getNumber’ function again. You should now see the contract state has been updated.

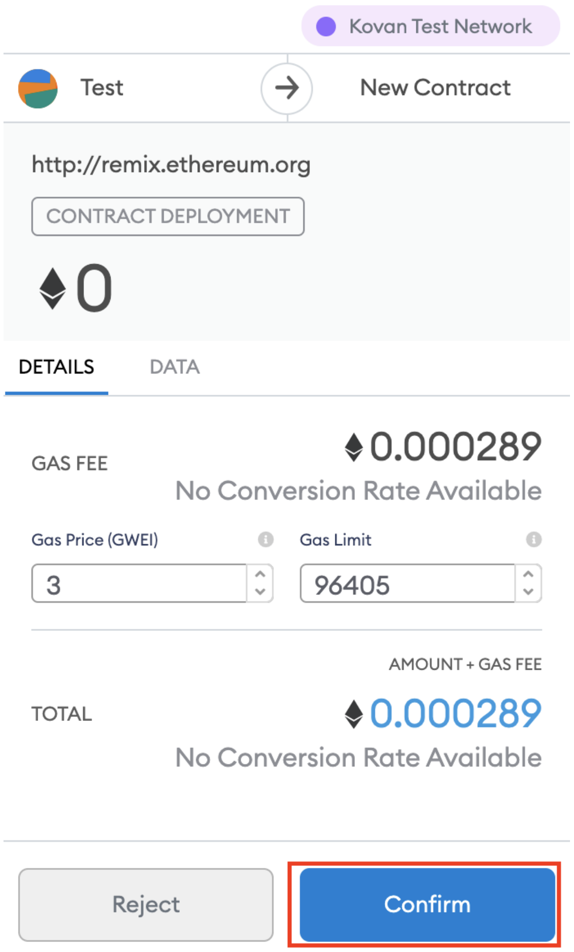


```
0:    uint256: 777
```

12. Now let's try this again, but instead of working on the local Remix VM network, we’ll deploy our contract to the public Kovan testnet. Change the ‘Environment’ dropdown to ‘Injected Web3’. If you’re not signed into your Metamask wallet, then sign in before continuing, and ensure the selected network is the ‘Kovan Test Network’, and that you have some ETH in your wallet. If you don’t have any ETH, raise your hand for help, and an instructor will help get you some.



13. Because we already compiled our contract and we haven't changed it, we don't need to compile it again. Press the orange Deploy button to deploy your already compiled contract. Metamask will popup asking you to confirm the transaction. Press the blue 'Confirm' button to execute the transaction, and deploy your contract to the Kovan network. After a few seconds, you should see your deployed contract in the 'Deployed Contracts' section.


Kovan Test Network

Test → New Contract

http://remix.ethereum.org

CONTRACT DEPLOYMENT

0

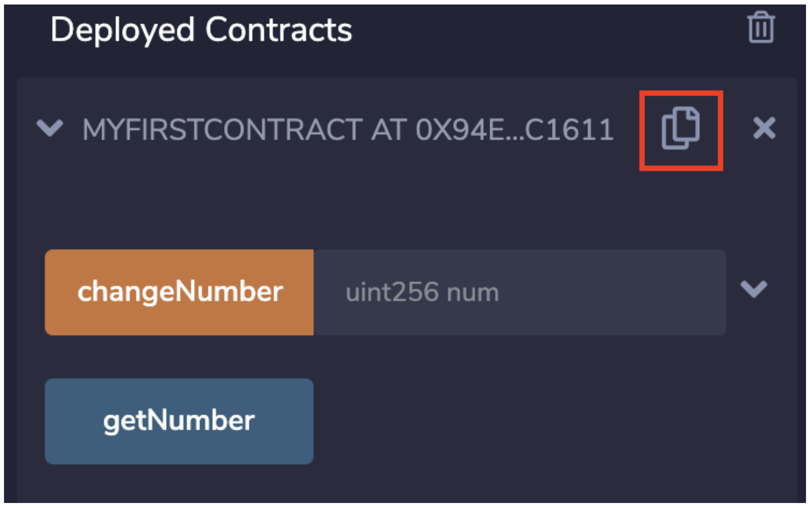
DETAILS DATA

GAS FEE 0.000289
No Conversion Rate Available

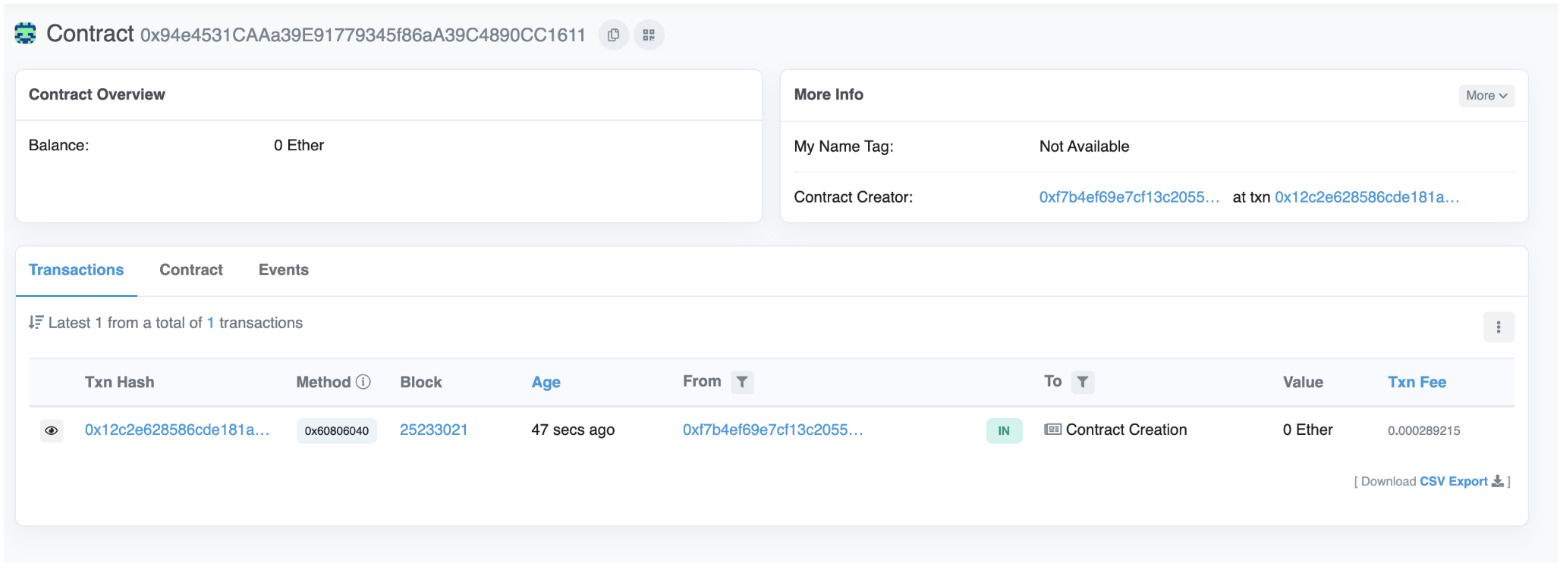
Gas Price (GWEI): 3
Gas Limit: 96405

AMOUNT + GAS FEE
TOTAL 0.000289
No Conversion Rate Available

14. Press the ‘copy’ contract address button to copy the contracts address to your clipboard, then head to https://kovan.etherscan.io/, here is where you can find your deployed contract on the Kovan network, and any associated transactions with it. Paste your contract address in the search field, and then press search. You should find your deployed contract, with 1 transaction (contract creation).


Deployed Contracts

MYFIRSTCONTRACT AT 0X94E...C1611



Contract 0x94e4531CAAa39E91779345f86aA39C4890CC1611

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: 0xf7b4ef69e7cf13c2055… at txn 0x12c2e628586cde181a…

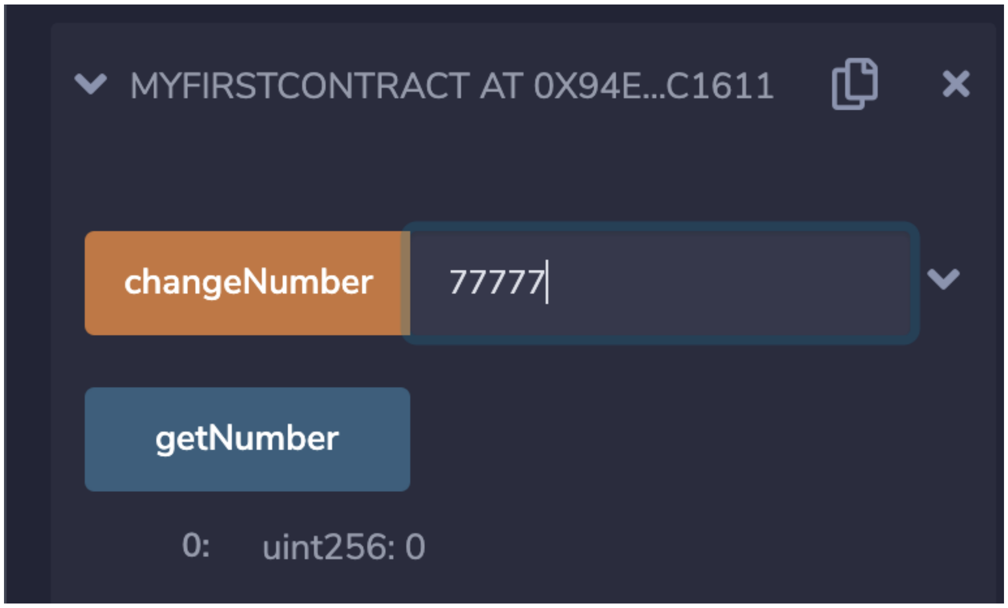
Transactions Contract Events

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

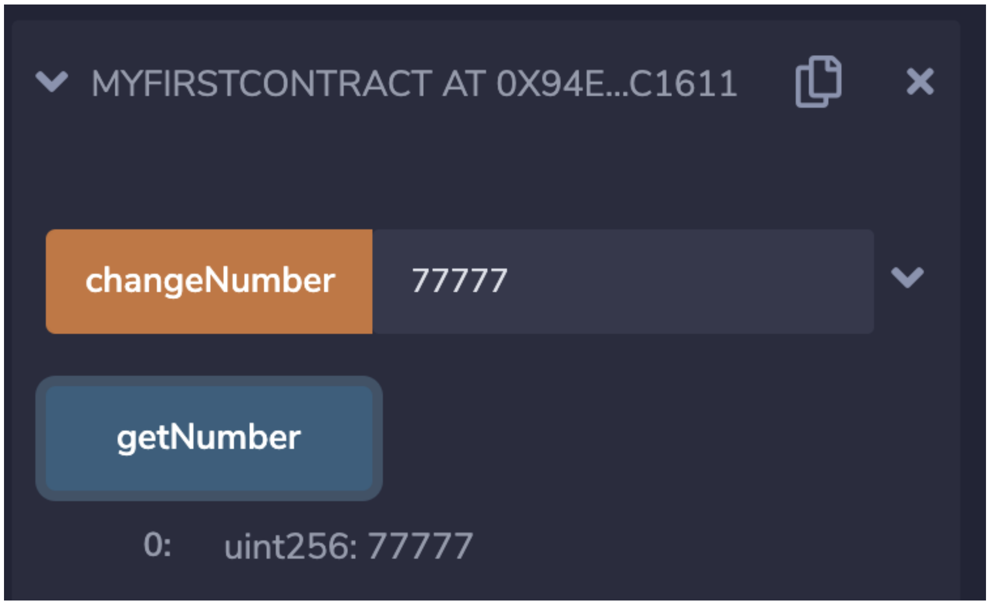
[Download CSV Export]

15. Once again, execute the ‘getNumber’ function and verify that the result returned is 0. Then enter a number into the ‘changeNumber’ function parameter, and execute the ‘changeNumber’ function to update the contract state. Once again, Metamask will pop up and ask you to confirm the transaction.

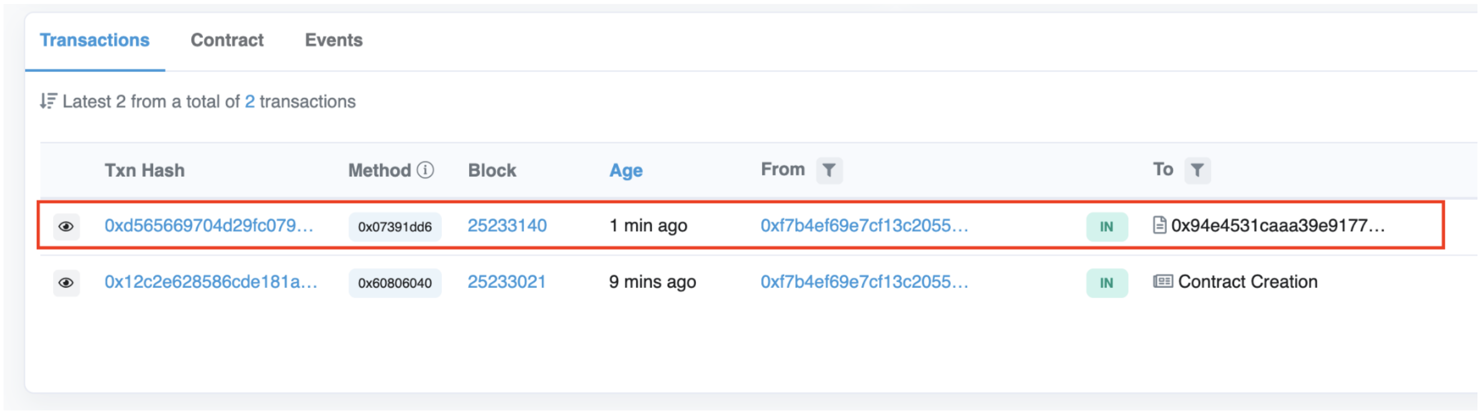


```
0:    uint256: 0
```

18. Give the transaction a few seconds to be included in a block, then once again execute the ‘getNumber’ function. You should now see the contract state has been updated

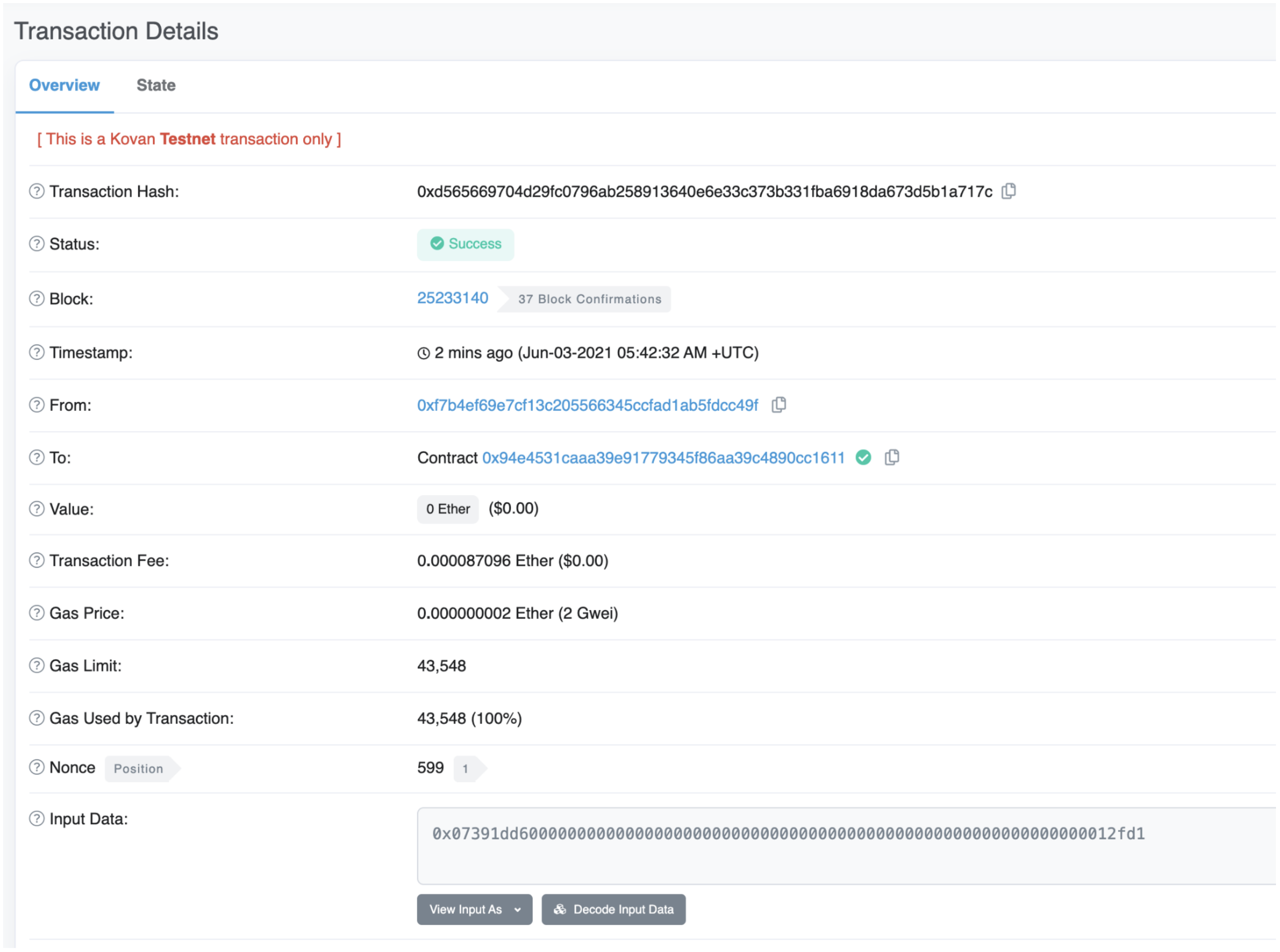


19. If you go back to Etherscan and refresh the page with your contract, you should see your new transaction listed. If you press on the ‘Txn Hash’ link, you will see the details of the transaction.



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

Overview State

[This is a Kovan **Testnet** transaction only]

Transaction Hash:	0xd565669704d29fc0796ab258913640e6e33c373b331fba6918da673d5b1a717c
Status:	Success
Block:	25233140 37 Block Confirmations
Timestamp:	2 mins ago (Jun-03-2021 05:42:32 AM +UTC)
From:	0xf7b4ef69e7cf13c205566345ccfad1ab5fdcc49f
To:	Contract 0x94e4531caaa39e91779345f86aa39c4890cc1611
Value:	0 Ether ($0.00)
Transaction Fee:	0.000087096 Ether ($0.00)
Gas Price:	0.000000002 Ether (2 Gwei)
Gas Limit:	43,548
Gas Used by Transaction:	43,548 (100%)
Nonce Position	599 1
Input Data:	0x07391dd60000000000000000000000000000000000000000000000000000000000012fd1

Congratulations, you've successfully written your first smart contract that reads and writes to the blockchain, and deployed it to a live network!

Bonus Exercise:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Modify your smart contract so that instead of storing the passed in number, it increments the currently stored number by the passed in *_num* param. To do this, you should initially set the state of the *number* variable to be 0

```
uint256 number = 0;
```

You can increment the stored number variable with the following syntax

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

2. Create a new function called ‘getNumberMultiplied’, that takes in a parameter called _*num*, and then returns an integer of the result of multiplying the _num parameter with the currently stored *number* parameter. The function should be defined as a view function similar to the getNumber() function, because we are not modifying the state of the contract.

3. Create a new function called ‘addNumbers’ that takes in two uint parameters, _*num1* and_*num2*, and then stores the result of adding the two numbers into the number variable.

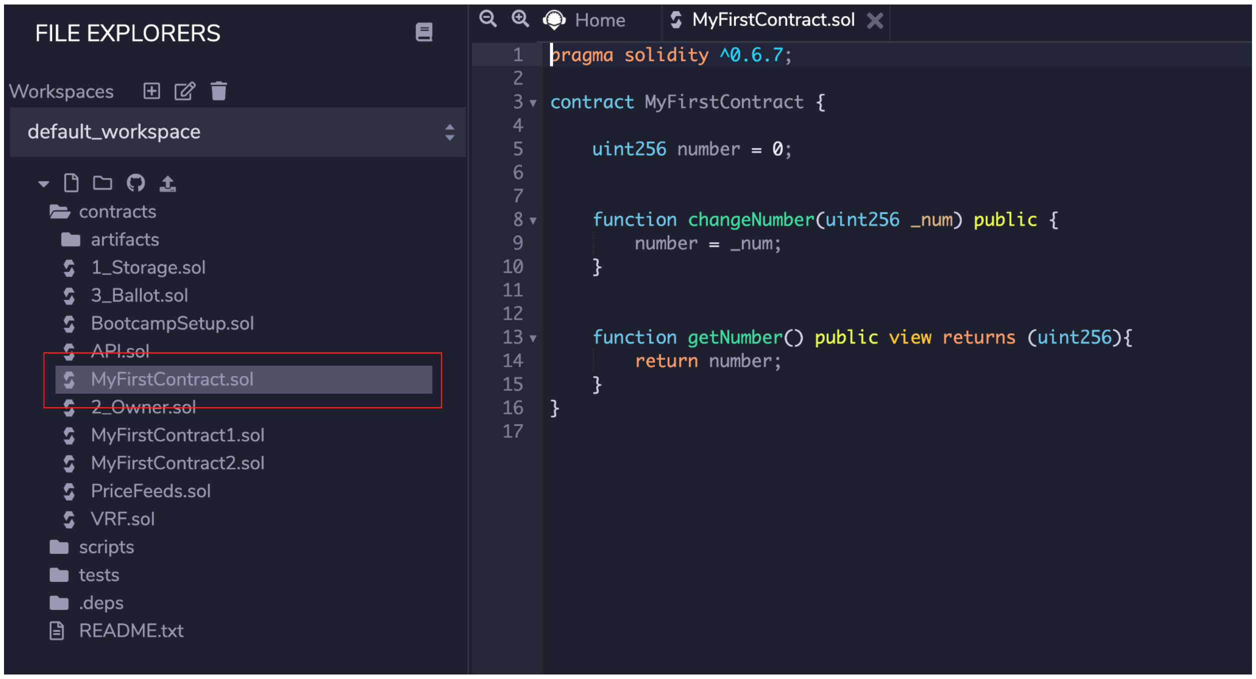
Exercise 2: Arrays and Mappings

In this exercise, we’re going to modify our MyFirstSmartContract smart contract from Day 1 of the bootcamp, and we’re going to add an array and a Mapping to it!

Adding an Array

For this part of the exercise, you’ll create a dynamic array of names in your smart contract, stored as strings.

1. Open Remix, you should be able to find your completed smart contract from the last session in your explorer



```
pragma solidity ^0.6.7;

contract MyFirstContract {

    uint256 number = 0;


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


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

2. Next you need to add the new array to your smart contract. Create a new dynamic array of strings called 'names'. You can add it under the existing 'number' variable.

```
string[] names;
```

3. Add a function to 'push' new values to the array. It should take in a parameter called _name

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

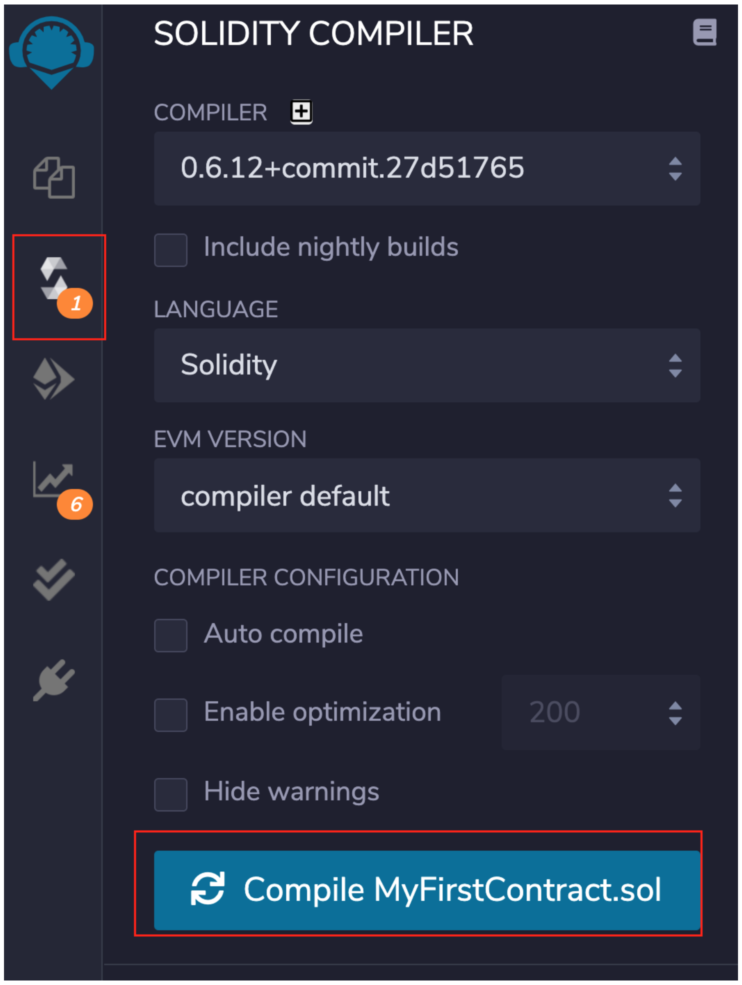
4. Add another function to get a name stored in the array, given an index parameter passed in. It should return the name in the array at the given index. Because you are reading the contract state, this can be a view function.

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

5. Your completed code should now look like this

```
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. You’re now ready to test your contract! Compile and re-deploy it. You can ignore any compile time warnings (text in orange). For this exercise, we’ll work in the local browser based EVM environment, so choose a ‘JavaScript VM’ environment:



SOLIDITY COMPILER

COMPILER

0.6.12+commit.27d51765

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

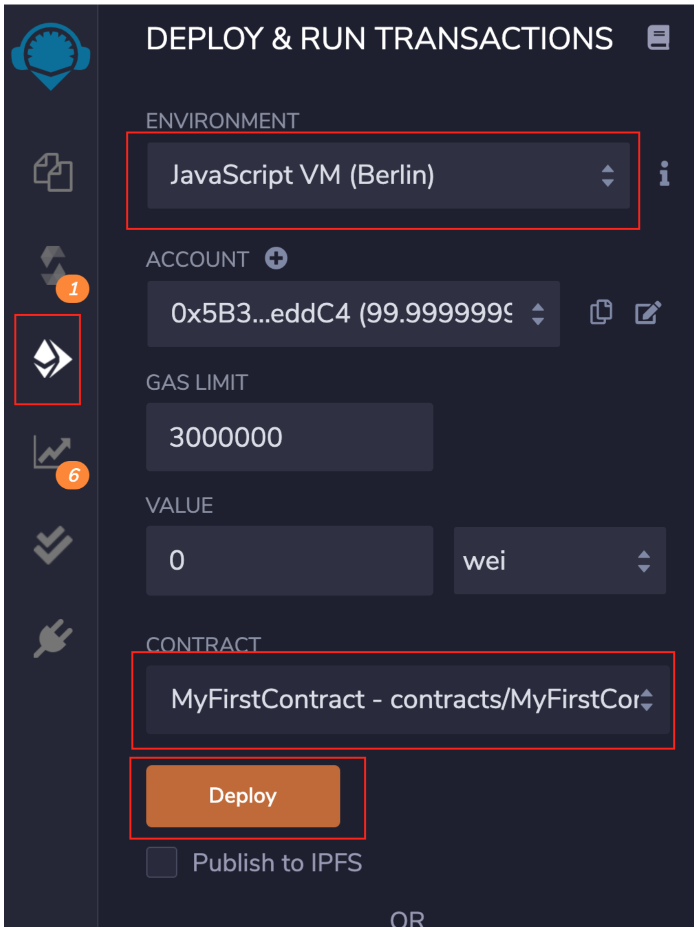
compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM (Berlin)

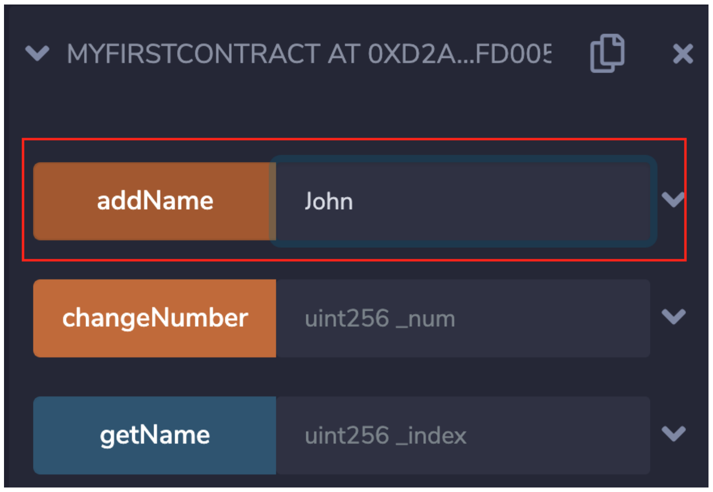
ACCOUNT
0x5B3...eddC4 (99.9999999

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
MyFirstContract - contracts/MyFirstCor

7. Once deployed, pass in a parameter to the ‘addName’ function, and execute it to add a name to the names array in the smart contract

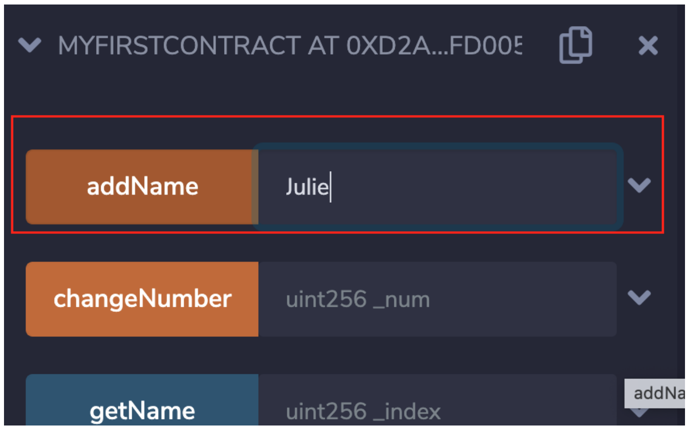

MYFIRSTCONTRACT AT 0XD2A...FD005

addName John

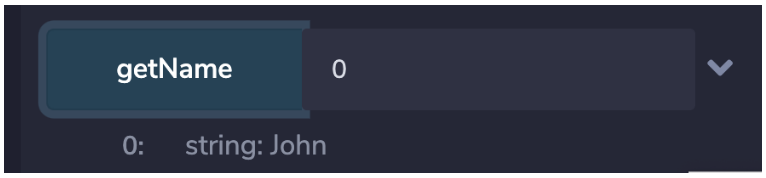
changeNumber uint256 _num

getName uint256 _index

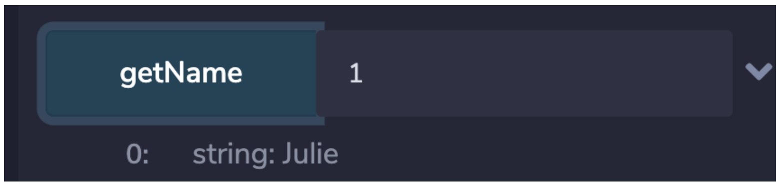
8. Add in another name of your choice to append it to the names array



9. Now call the ‘getName’ function, passing in an index of 0 to see the first value stored in the array. Repeat the step after this, passing in an index of 1 to see the second value.



```
0:    string: John
```



```
0:    string: Julie
```

Congratulations, you’ve successfully created a dynamic array in your smart contract, and populated it with data.

Adding a Mapping

For this part of the exercise, you’ll create a mapping of names->mobile numbers in your smart contract

1. First you need to add the new mapping to your smart contract. Create a new mapping of names to integers (phone numbers) as follows:

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

2. Create a function to add values to the mapping. It should take in two parameters, called _name and _mobileNumber

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

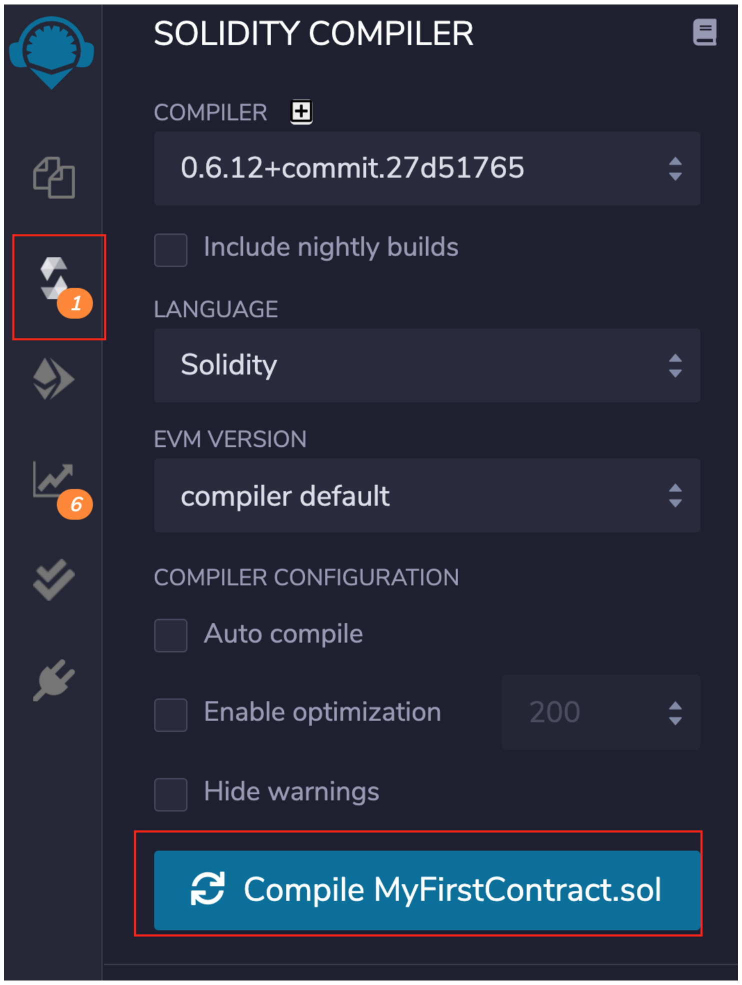
3. Add another function to get a phone number from the mapping for a given name. It should take a _name parameter, and return a uint

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

4. Your completed code should now look like this

```
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. You’re now ready to test your contract! Compile and re-deploy it. You can ignore any compile time warnings (text in orange). For this exercise, once again we’ll work in the local browser based EVM environment, so choose a ‘JavaScript VM’ environment:



SOLIDITY COMPILER

COMPILER

0.6.12+commit.27d51765

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

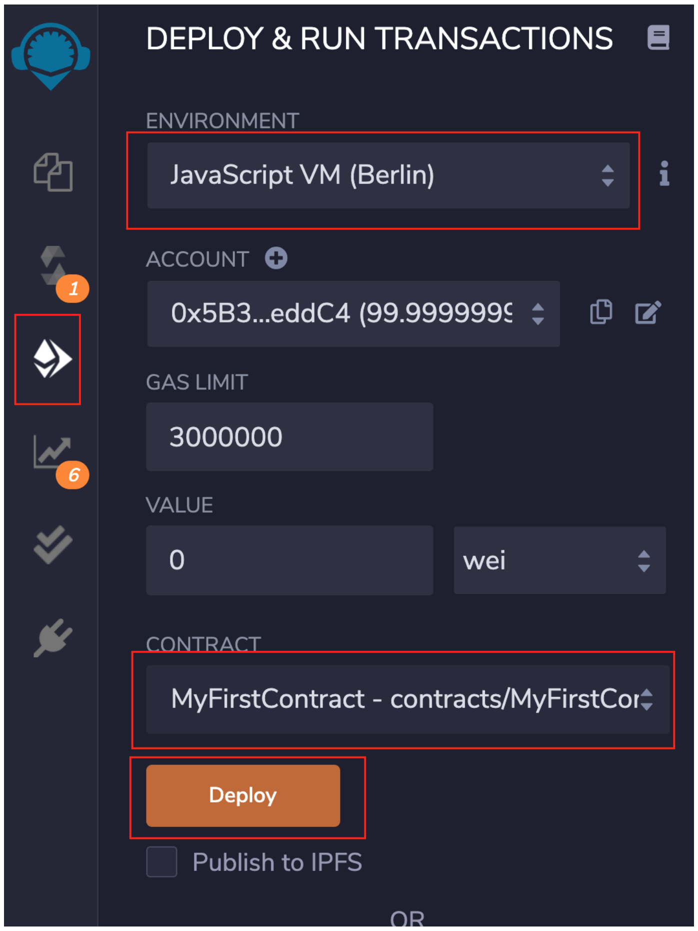
compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM (Berlin)

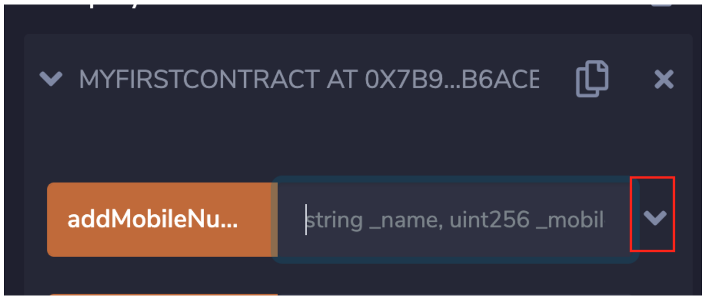
ACCOUNT: 0x5B3...eddC4 (99.9999999

GAS LIMIT: 3000000

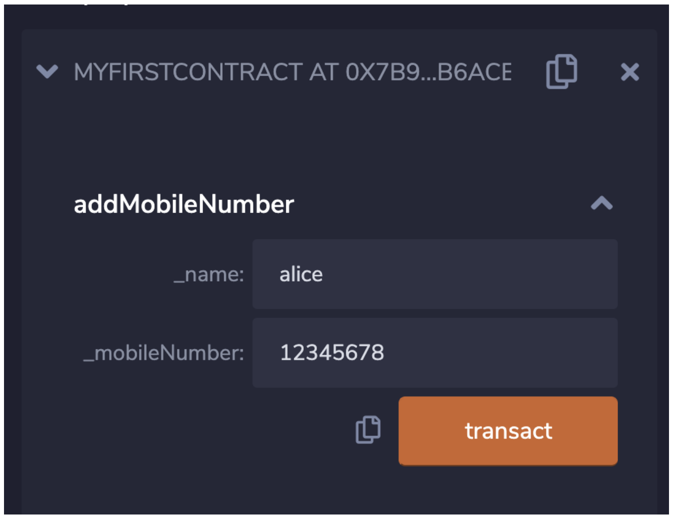
VALUE: 0 wei

CONTRACT: MyFirstContract - contracts/MyFirstCor

11. Once deployed, expand the ‘addMobileNumber’ dropdown button, so you can see both parameters. Enter in some values and execute the function

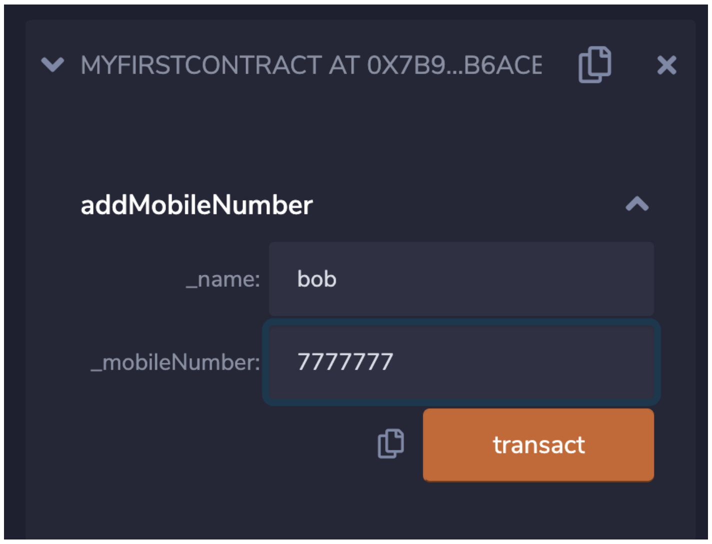


MYFIRSTCONTRACT AT 0X7B9...B6ACE



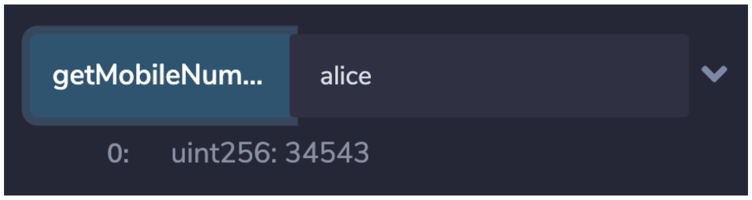
_name: alice
_mobileNumber: 12345678

12. Add in another name and number of your choice, and add it to the mapping



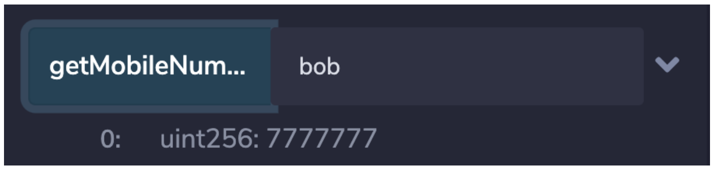
_name: bob
_mobileNumber: 7777777

13. Now call the 'getMobileNumber' function, passing in one of the names entered into the mapping in the previous steps. You should see it return the stored number against the name. Repeat the step for the other name entered



alice

```
0:    uint256: 34543
```



```
0:    uint256: 7777777
```

Congratulations, you've successfully created a mapping in your smart contract, and populated and interacted with it!

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Create a function in your smart contract that returns the length of the names array, call it getNamesLength(). The return type should be uint, and the way to get the length of an array is with the following syntax

```
return names.length;
```

2. Create a function in your smart contract that returns the entire names array as a string. You'll need to add the ABIEncoderV2 pragma statement for this at the top of your contract file under your import. ie:

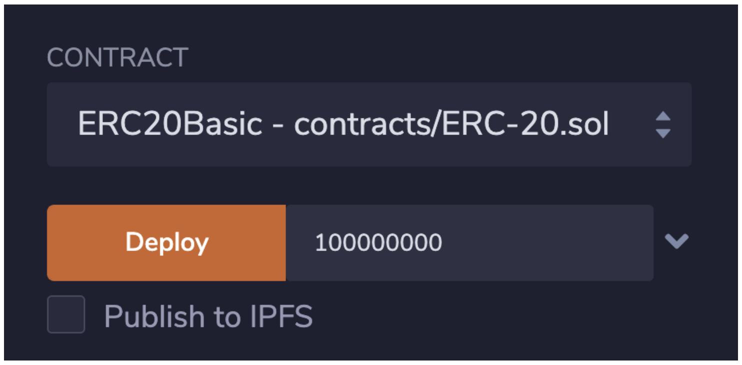
```
pragma solidity ^0.6.7;
pragma experimental ABIEncoderV2;
```

   Call the function getNames. It should be a view function, with a return type of the following:

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

3. Create a new smart contract in Remix (in a new file) called 'ERC-20.sol'. Paste in the complete code example of a basic ERC-20 contract from the Ethereum developer tutorials page. Modify the name and symbol parameters in the constructor of the ERC20Basic contract, choose any values you wish

4. Deploy the ERC20Basic contract to your local JavaScript VM network. You need to specify the total number of tokens in the contract when you deploy. In this example, we've chosen 100 million. Make sure you have the right contract selected to deploy.



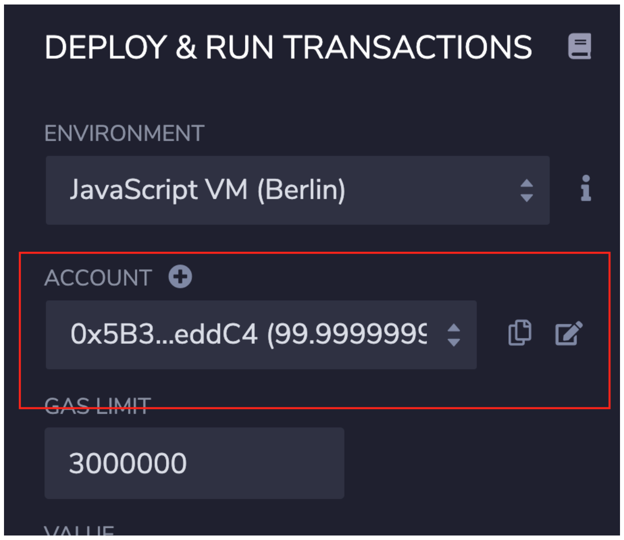
CONTRACT

ERC20Basic - contracts/ERC-20.sol

Deploy 100000000

Publish to IPFS

5. Once deployed, copy your currently selected account in Remix by pressing the copy button next to the value



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM (Berlin)

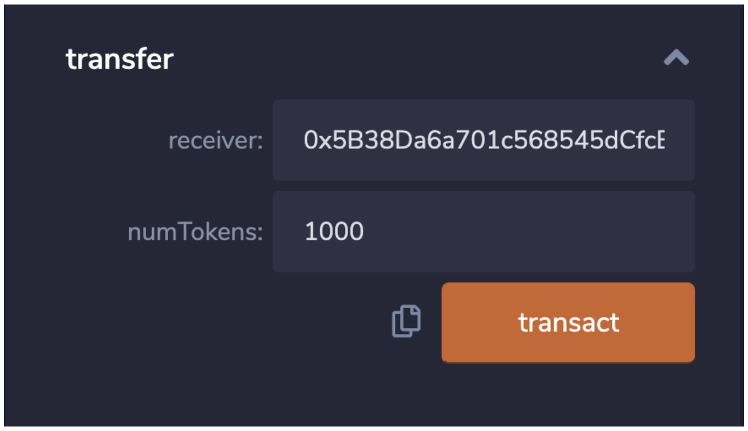
ACCOUNT

0x5B3...eddC4 (99.9999999

GAS LIMIT

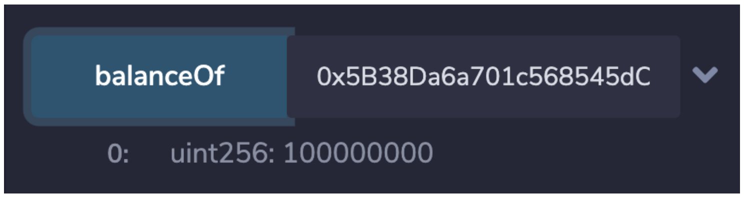
3000000

6. Call the transfer function in your deployed contract to mint yourself some tokens, passing in your copied wallet address, and a value for the token amount. Then call the 'balanceOf' function, once again passing in your copied wallet address, to see your balance


transfer

receiver: 0x5B38Da6a701c568545dCfcE

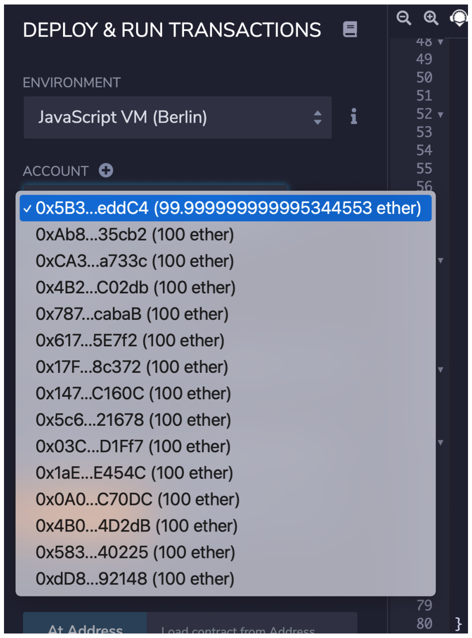
numTokens: 1000



0x5B38Da6a701c568545dC

0: uint256: 100000000

7. Continue to play with the token contract and see how the other functions like ‘transferFrom’ etc works. You can switch your currently active wallet address around using the ‘Account’ dropdown in Remix, so you can send token amounts between your wallet accounts


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM (Berlin)

ACCOUNT

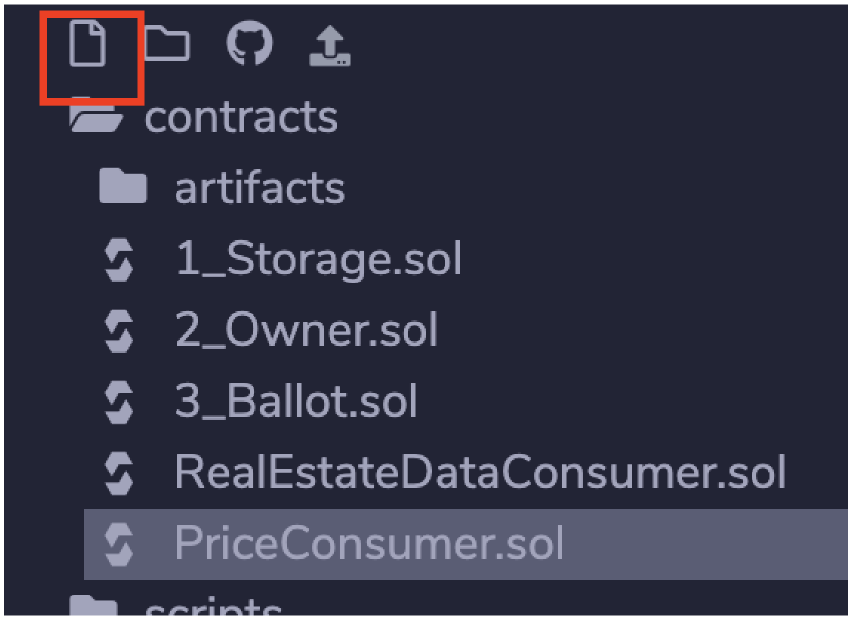
```
✓ 0x5B3...eddC4 (99.999999999995344553 ether)
0xAb8...35cb2 (100 ether)
0xCA3...a733c (100 ether)
0x4B2...C02db (100 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x147...C160C (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (100 ether)
0xdD8...92148 (100 ether)
```

Congratulations, you’ve built, deployed and interacted with a token contract.

Exercise 3: Price Feeds

In this exercise, you're going to create a simple smart contract that looks up the current price of Ethereum using Chainlink Price Feeds. Firstly, open http://remix.ethereum.org/

1. Expand the contracts folder, and press the new file button. Call the file PriceConsumer.sol



2. Select which version of the compiler we will be using by entering the following line into the new file:

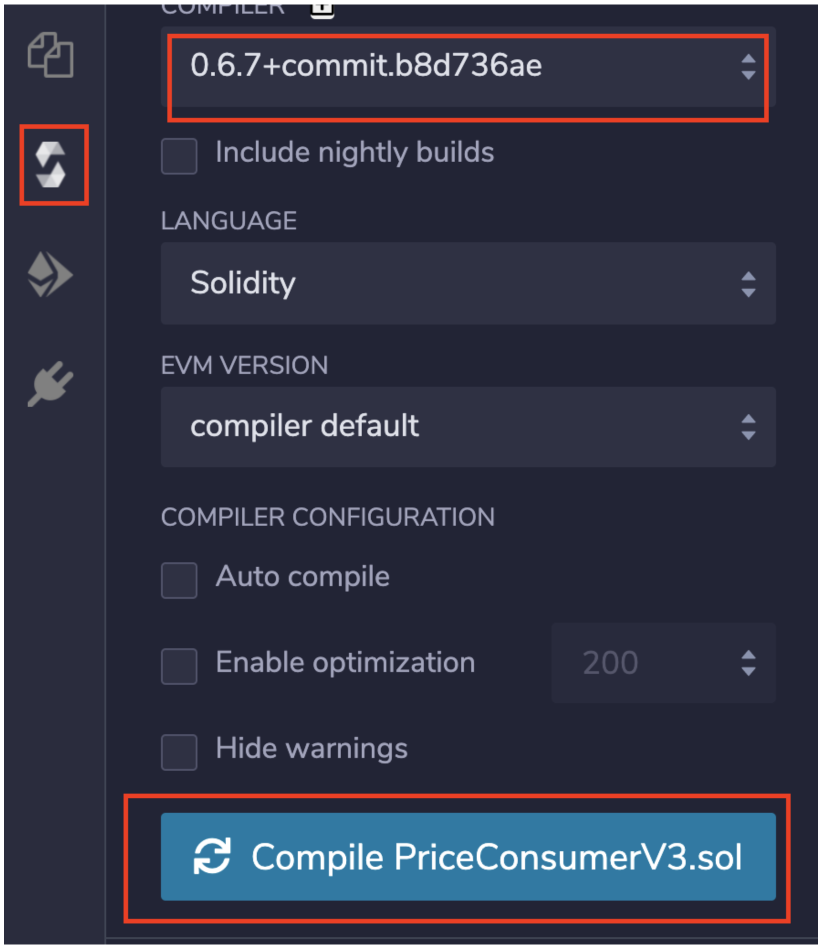
```
pragma solidity ^0.6.7;
```

3. Underneath, enter in the contract source code. In our first example we will be using the ETH/USD price feed contract **0x9326BFA02ADD2366b30bacB125260Af641031331**, which we've taken from the Kovan section of the Ethereum Price Feeds page of the Chainlink documentation

```
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. Your contract is now ready to be compiled. Press on the Solidity Compiler menu option on the left hand side menu, change the compiler version in the compiler dropdown so that it matches the compiler specified in your code, then press on the blue Compile button:


COMPILER

0.6.7+commit.b8d736ae

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

COMPILER CONFIGURATION

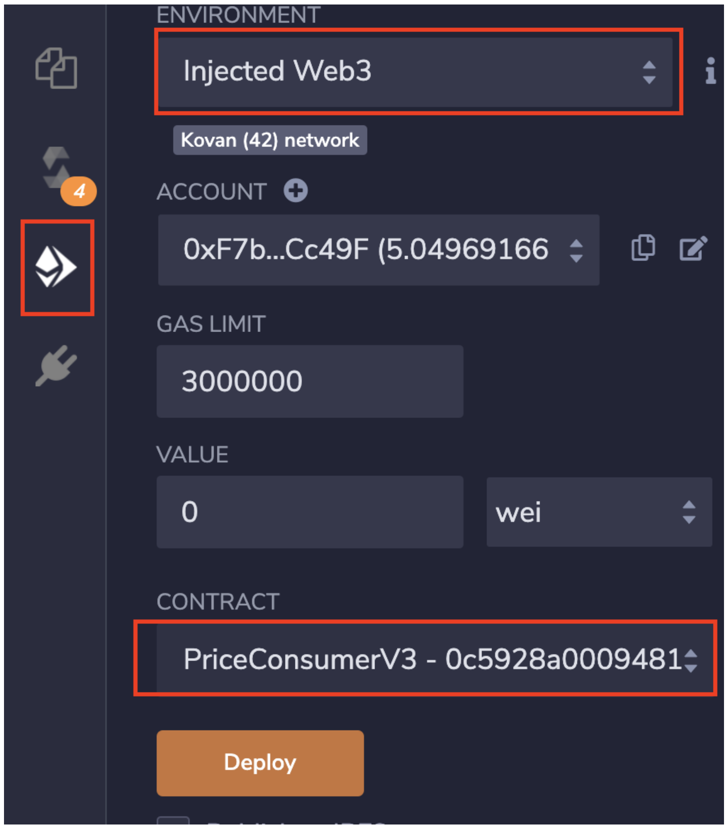
Auto compile

Enable optimization 200

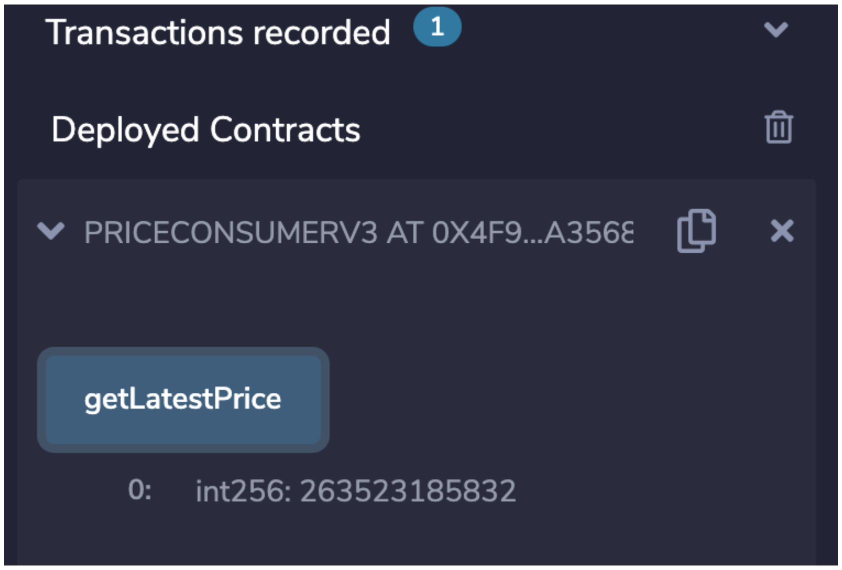
Hide warnings

Compile PriceConsumerV3.sol

5. Choose the ‘Deploy and Run Transactions’ button on the left hand side menu. Because you're going to integrate to the Chainlink Oracle network, we need to connect to the Kovan public testnet. Ensure Environment is set to ‘Injected Web3’, and that the selected contract is ‘PriceConsumer.sol’, then press the Deploy button. Metamask will popup asking you to confirm the transaction. Press the blue ‘Confirm’ button to execute the transaction, and deploy your contract to the Kovan network. After a few seconds, you should see your deployed contract in the ‘Deployed Contracts’ section in Remix.


ENVIRONMENT
Injected Web3
Kovan (42) network
ACCOUNT
0xF7b...Cc49F (5.04969166
GAS LIMIT
3000000
VALUE
0
wei
CONTRACT
PriceConsumerV3 - 0c5928a0009481

6. Expand the deployed contract by pressing on the “>” icon next to the contract name in the Deployed Contracts section. You should see the available functions that can be executed. Press on the ‘getLatestPrice’ button to lookup the price of ETH from the Kovan ETH/USD price feed.


Transactions recorded 1
Deployed Contracts
PRICECONSUMERV3 AT 0X4F9...A3568

```
0:    int256: 263523185832
```

7. The returned result is the current market price of ETH, multiplied by 10^8. You can compare it to the Mainnet ETH/USD Price Feed by going to https://data.chain.link/eth-usd and looking at the current price.

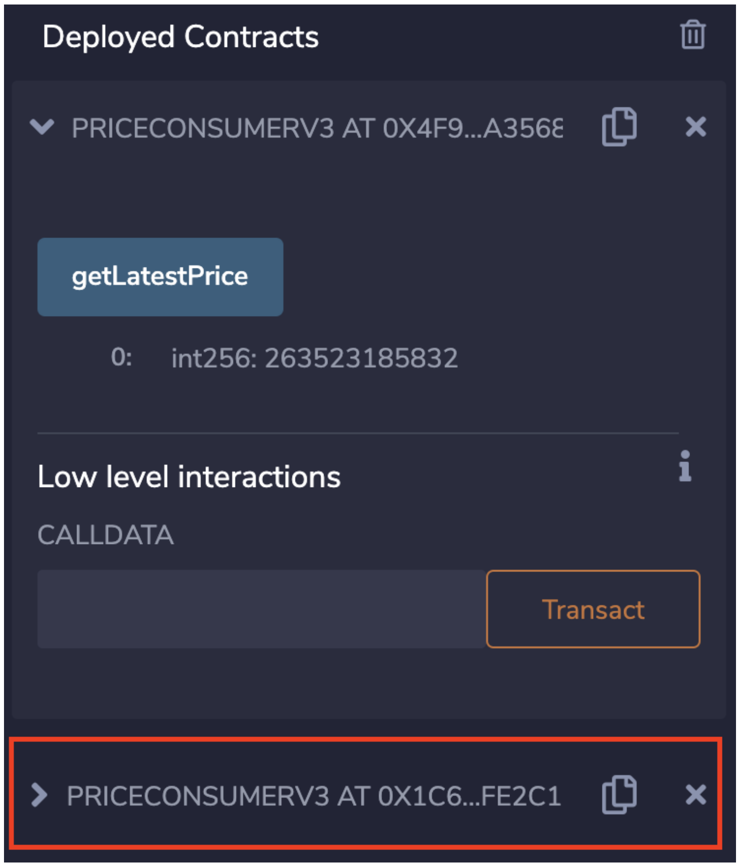
8. Go to the Ethereum Price Feeds page in the Chainlink documentation, and navigate down to the 'Kovan Testnet' section. Pick another pair, and copy the Proxy address. In our example, we've chosen the LINK/USD pair **0x396c5E36DD0a0F5a5D33dae44368D4193f69a1F0**

9. Go back to your contract, and in the constructor, replace the address of the ETH/USD price feed address, with the one that you copied in the previous step.

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

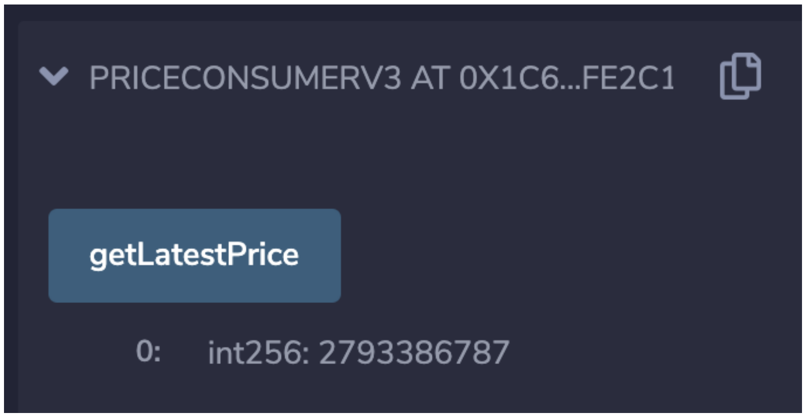
10. Go back to the Solidity Compiler tab on Remix, and press the blue 'Compile PriceConsumer.sol' button to re-compile your contract.

11. Go to the 'Deploy & Run Transactions' tab and re-deploy your contract, accepting the transaction in Metamask. Once deployed, you should see a second deployed contract under the 'Deployed Contracts' section.



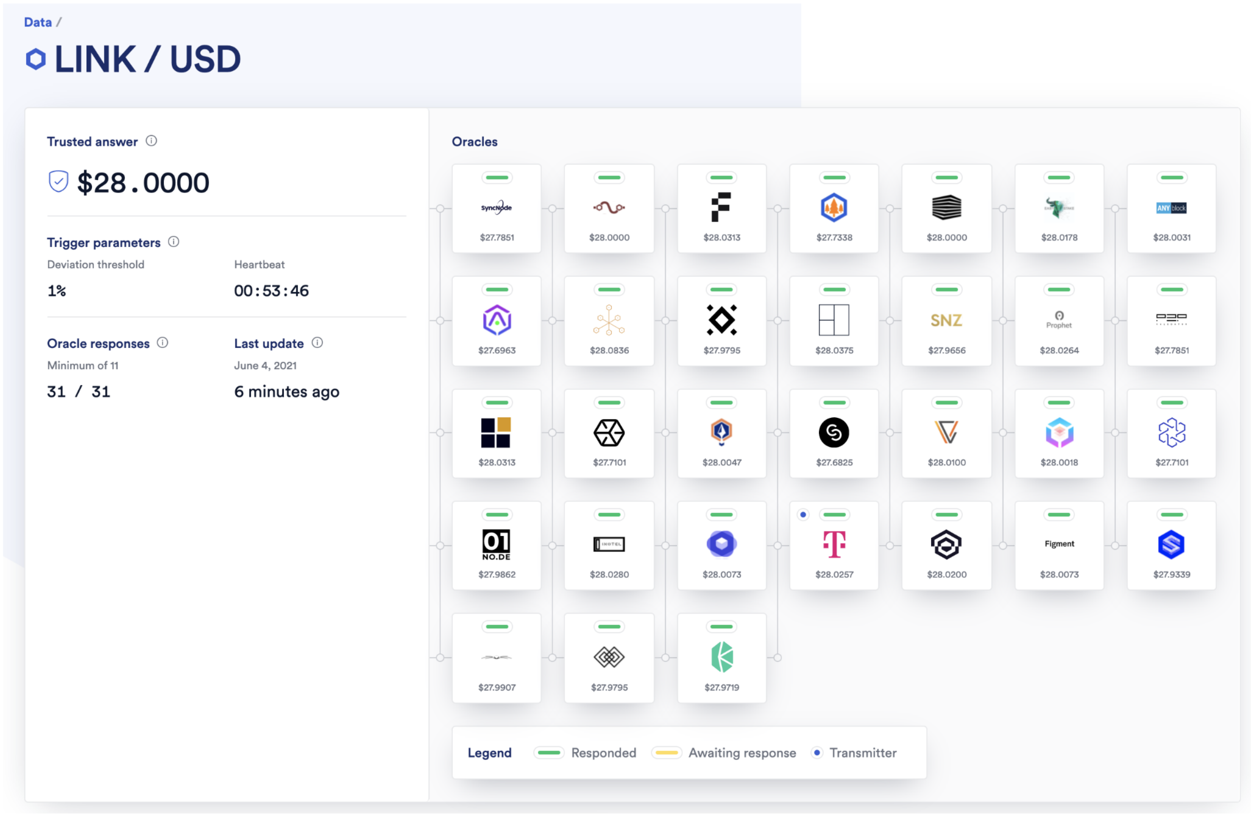
```
0:    int256: 263523185832
```

12. Expand the deployed contract once again, and execute the ‘getLatestPrice’ function. You should see the returned current price in the price feed contract that you selected.



```
0:    int256: 2793386787
```

13. Head to https://data.chain.link/ and look for the equivalent price feed running on Mainnet.



Data /

LINK / USD

Trusted answer

$28.0000

Trigger parameters

Deviation threshold: 1%

Heartbeat: 00:53:46

Oracle responses

Minimum of 11

31 / 31

Last update

June 4, 2021

6 minutes ago

Oracles

$27.7851	$28.0000	$28.0313	$27.7338	$28.0000	$28.0178	$28.0031
$27.6963	$28.0836	$27.9795	$28.0375	$27.9656	$28.0264	$27.7851
$28.0313	$27.7101	$28.0047	$27.6825	$28.0100	$28.0018	$27.7101
$27.9862	$28.0280	$28.0073	$28.0257	$28.0200	$28.0073	$27.9339
$27.9907	$27.9795	$27.9719				

Legend: Responded · Awaiting response · ● Transmitter

Congratulations, you've successfully written and deployed a contract that uses Chainlink Price Feeds to obtain external price data!

Bonus Exercise:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

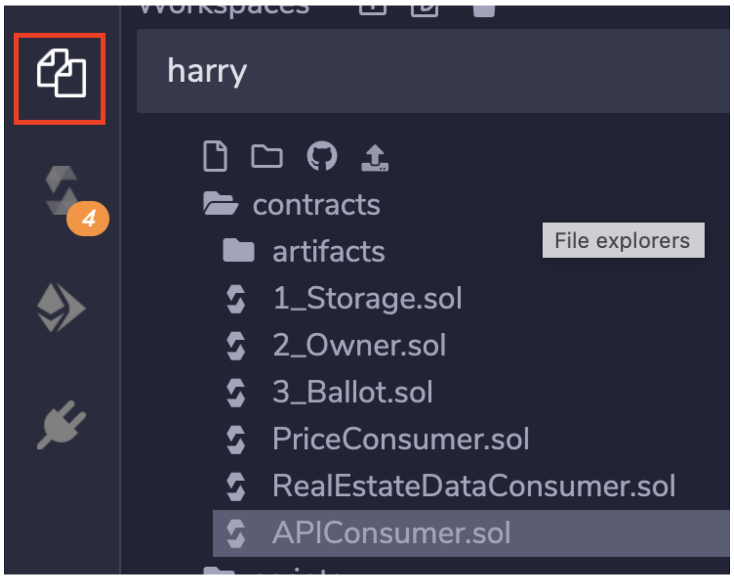
1. Modify the Price Feed contract to get a different price feed that you pick from the Kovan Price Feed addresses in the Chainlink Documentation. Then deploy and test your contract. Compare the value to the mainnet feed value at https://data.chain.link

2. Create a new function called 'getTimestamp' that returns a uint which is the timeStamp of the current round (hint: it can be accessed via the priceFeed.latestRoundData() call)

Exercise 4: Any-API

In this exercise, you're going to create a simple smart contract that performs a request for external data, using Chainlink Any-API. Firstly, open http://remix.ethereum.org/

1. Expand the contracts folder, and press the new file button. Call the file APIConsumer.sol



2. Select which version of the compiler we will be using by entering the following line into the new file:

```
pragma solidity ^0.6.7;
```

3. Go to the following URL in your browser, to see the json output of the API we will be accessing. Search for the term "**VOLUME24HOUR**" to see the exact value we will be looking to get into our smart contract

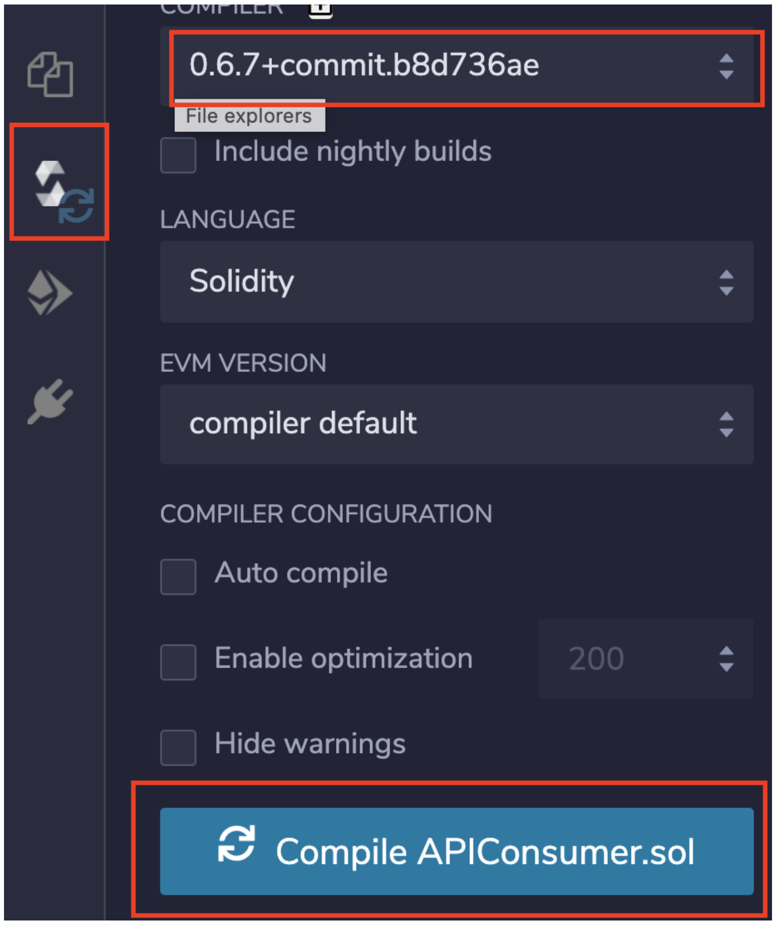
```
https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH
&tsyms=USD
```

4. Back in Remix, underneath your pragma command, enter in the contract source code.

```
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. Your contract is now ready to be compiled. Press on the Solidity Compiler menu option on the left hand side menu, change the compiler version in the compiler dropdown so that it matches the compiler specified in your code, then press on the blue Compile button:


COMPILER

0.6.7+commit.b8d736ae

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

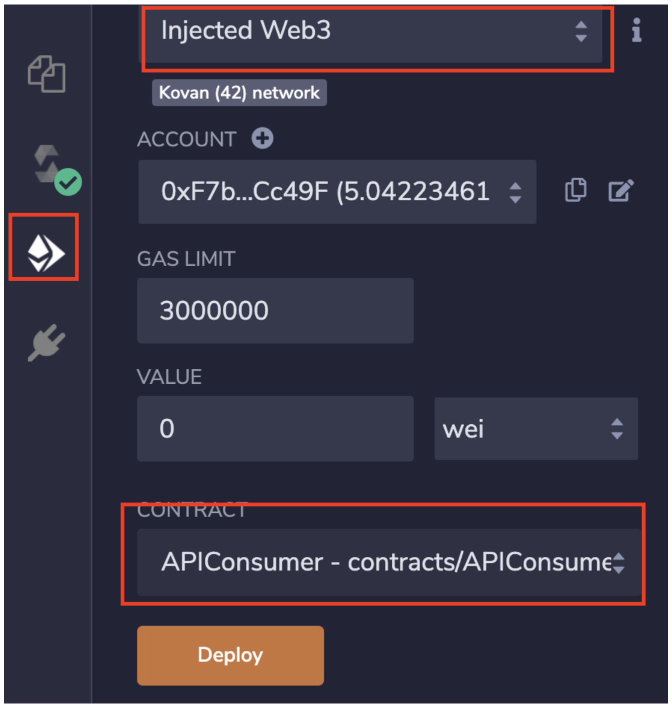
COMPILER CONFIGURATION

Auto compile

Enable optimization 200

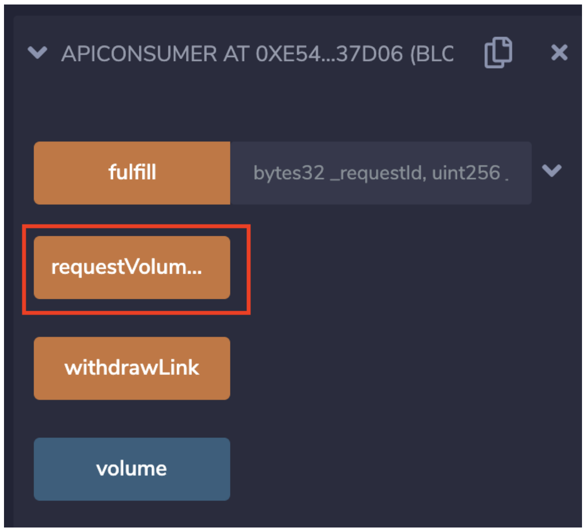
Hide warnings

6. Choose the ‘Deploy and Run Transactions’ button on the left hand side menu. Because you're going to integrate to the Chainlink Oracle network, we need to connect to the Kovan public testnet. Ensure Environment is set to ‘Injected Web3’, and that the selected contract is APIConsumer.sol’, then press the Deploy button. Metamask will popup asking you to confirm the transaction. Press the blue ‘Confirm’ button to execute the transaction, and deploy your contract to the Kovan network. After a few seconds, you should see your deployed contract in the ‘Deployed Contracts’ section in Remix.

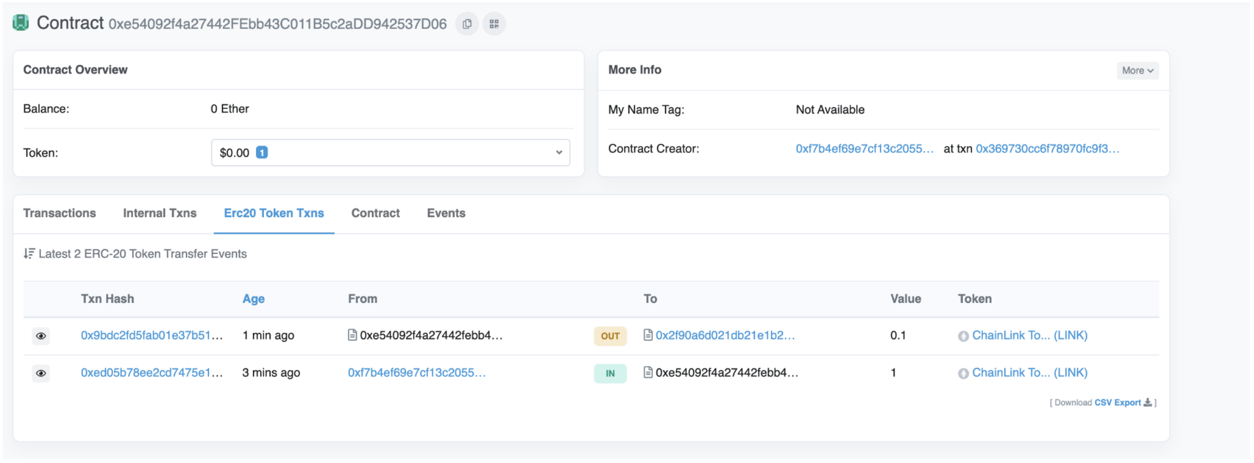


7. Fund your contract with 1 link token. If you don't have the LINK token added to your MetaMask wallet yet, press the 'Add Token' button at the bottom, then enter in the LINK token contract address **0xa36085F69e2889c224210F603D836748e7dC0088,** then press save.

8. Once your contract has been funded with LINK, you can now initiate the external data request. Expand the deployed contract by pressing on the ">" icon next to the contract name in the Deployed Contracts section. You should see the available functions that can be executed. Press on the 'requestVolumeData' button, then confirm the transaction when MetaMask pops up.


APICONSUMER AT 0XE54...37D06 (BLC
fulfill
bytes32 _requestId, uint256
requestVolum...
withdrawLink
volume

9. After a few seconds, your transaction should be approved. While you wait for the chainlink node to fulfill the request, goto https://kovan.etherscan.io/ and search for your deployed contract address (you copied it to the clipboard when you funded your contract with LINK). Once you find your deployed contract on Etherscan, go-to the 'ERC-20 Token Txns' tab. You should see 2 transactions, one incoming transfer of 1 LINK from when you funded your contract, and 0.1 LINK transferred out of the contract for your external data request.



Contract 0xe54092f4a27442FEbb43C011B5c2aDD942537D06

Contract Overview

Balance:	0 Ether
Token:	$0.00 1

More Info

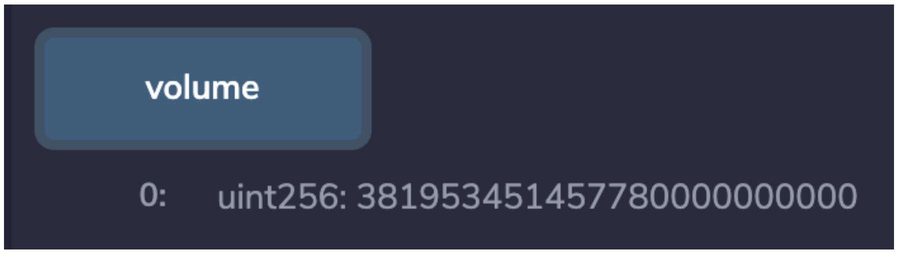
My Name Tag:	Not Available
Contract Creator:	0xf7b4ef69e7cf13c2055… at txn 0x369730cc6f78970fc9f3…

Latest 2 ERC-20 Token Transfer Events

	Txn Hash	Age	From		To	Value	Token
👁	0x9bdc2fd5fab01e37b51…	1 min ago	0xe54092f4a27442febb4…	OUT	0x2f90a6d021db21e1b2…	0.1	ChainLink To... (LINK)
👁	0xed05b78ee2cd7475e1…	3 mins ago	0xf7b4ef69e7cf13c2055…	IN	0xe54092f4a27442febb4…	1	ChainLink To... (LINK)

[Download CSV Export]

10. Go back to your deployed contract in Remix. We will now check to see the returned result from the external data request. Press the 'volume' button to execute the function that returns the value of the public 'volume' variable. You should now have a result that matches the value that you saw in your web browser in step 19 above. The value has been purposely multiplied by 10**18 to avoid any issues with math operations etc that may be done on the result.



```
0:    uint256: 381953451457780000000000
```

Congratulations, you've successfully performed a request for external data, and pulled in data from a public API into your smart contract!

Bonus Exercise:

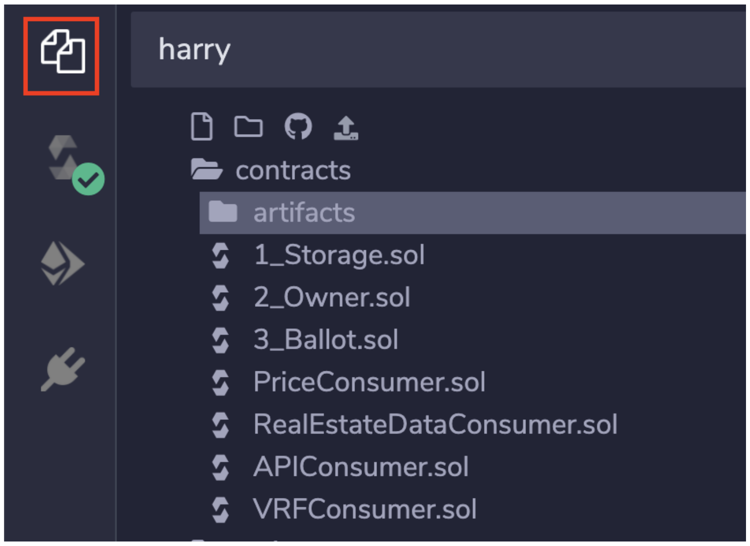
You can attempt to complete these exercises if you've completed the main exercise ahead of schedule

1. Modify your contract to get the **price** data from the cryptocompare API instead of the volume. Rename all variables that mention volume to reflect the change in the data being brought in
2. Change the API being accessed to the following, to get the BTC/USD price. *https://api.cryptowat.ch/markets/kraken/btcusd/price*. You'll need to modify the 'path' parameter sent in the request as well to match the JSON path of the 'price' field if you paste the URL above into a browser window

Exercise 5: VRF

In this exercise, you're going to create a simple smart contract that performs a request for randomness using Chainlink VRF. Firstly, open http://remix.ethereum.org/

11. Expand the contracts folder, and press the new file button. Call the file VRFConsumer.sol



12. Select which version of the compiler we will be using by entering the following line into the new file:

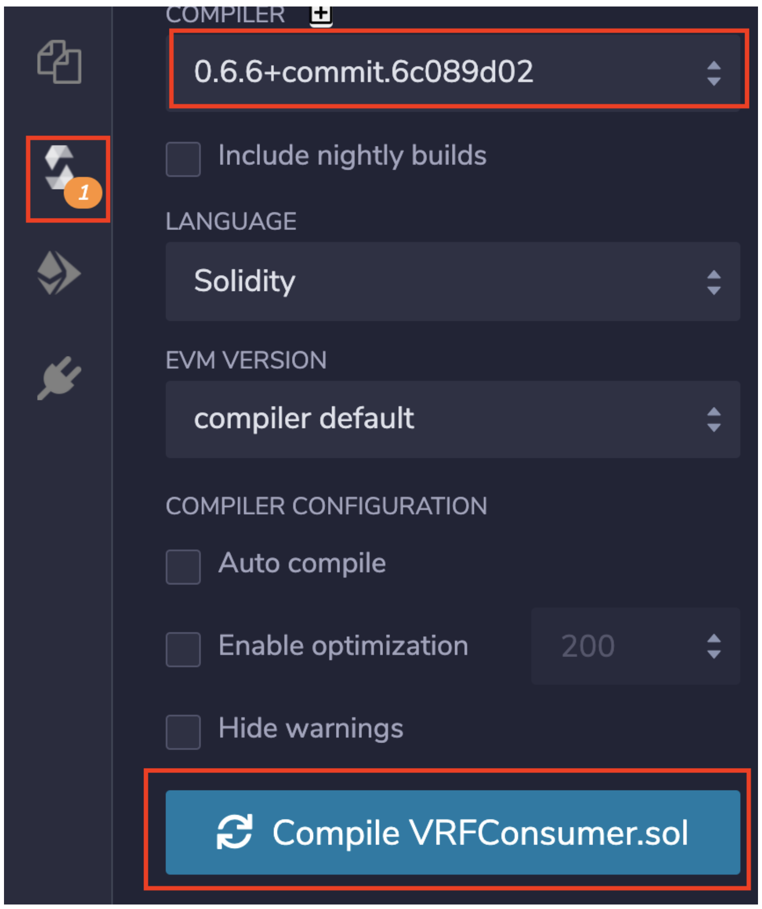
```
pragma solidity ^0.6.7;
```

13. Underneath, enter in the contract source code

```
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. Your contract is now ready to be compiled. Press on the Solidity Compiler menu option on the left hand side menu, change the compiler version in the compiler dropdown so that it matches the compiler specified in your code, then press on the blue Compile button:


COMPILER

0.6.6+commit.6c089d02

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

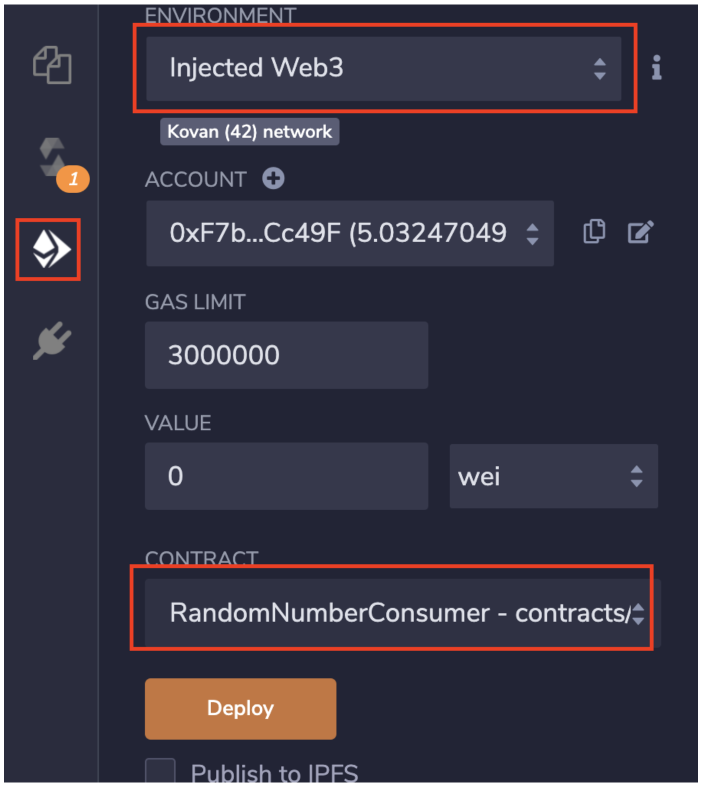
COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

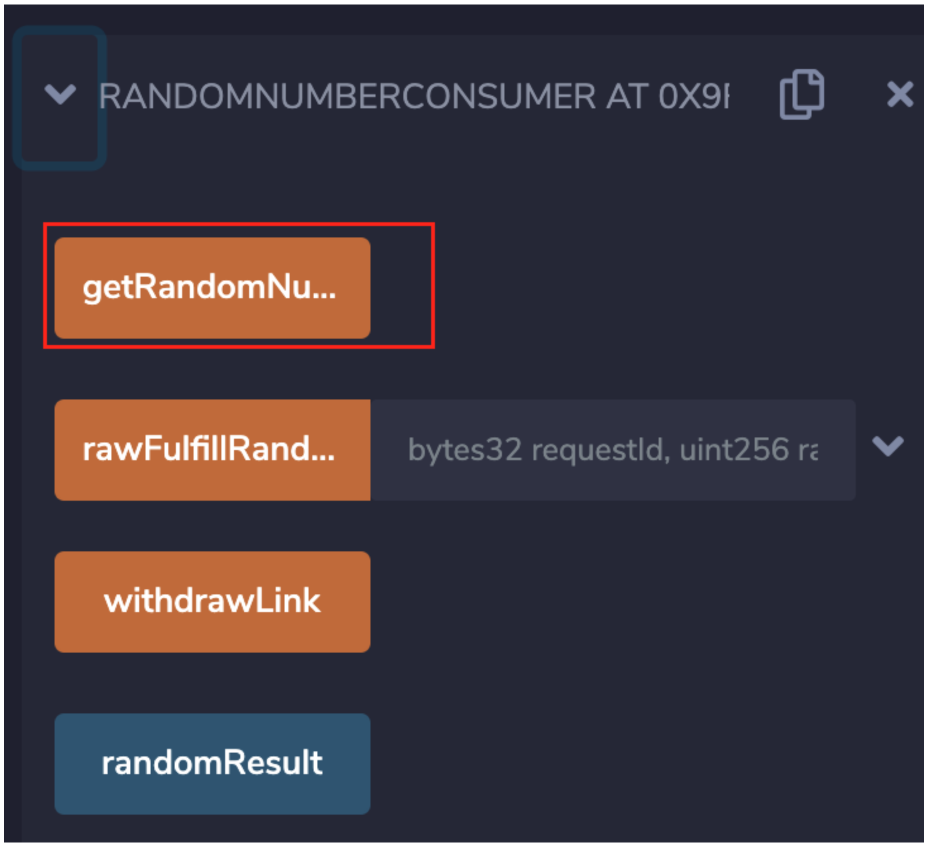
15. Choose the ‘Deploy and Run Transactions’ button on the left hand side menu. Because you're going to integrate to the Chainlink Oracle network, we need to connect to the Kovan public testnet. Ensure Environment is set to ‘Injected Web3’, and that the selected contract is RandomNumberConsumer’, then press the Deploy button. Metamask will popup asking you to confirm the transaction. Press the blue ‘Confirm’ button to execute the transaction, and deploy your contract to the Kovan network. After a few seconds, you should see your deployed contract in the ‘Deployed Contracts’ section in Remix.



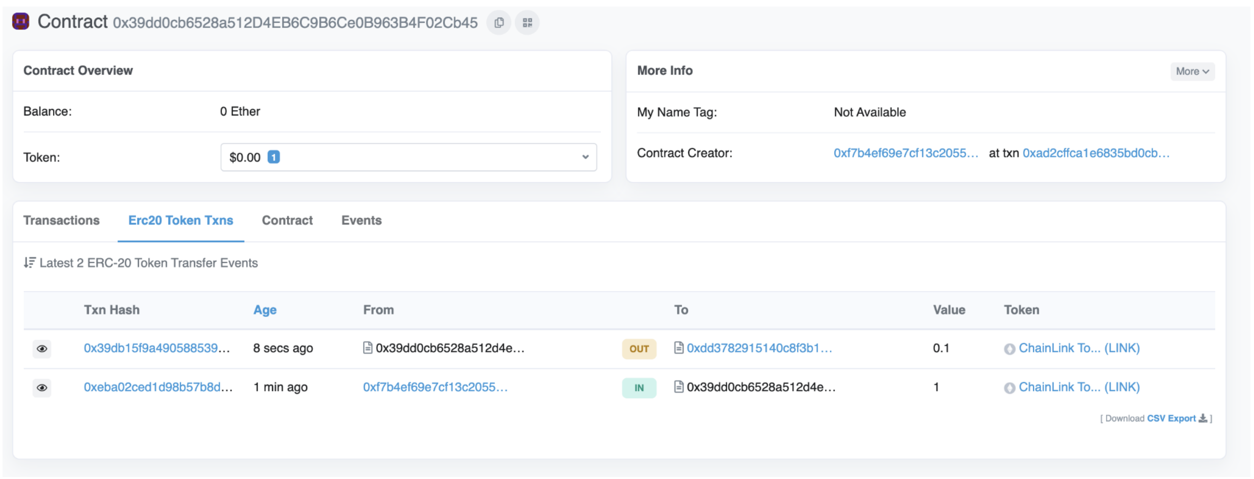
ENVIRONMENT
Injected Web3
Kovan (42) network
ACCOUNT
0xF7b...Cc49F (5.03247049
GAS LIMIT
3000000
VALUE
0 wei
CONTRACT
RandomNumberConsumer - contracts/

16. Fund your contract with 1 link token.

17. Once your contract has been funded with LINK, you can now initiate the request for randomness. Expand the deployed contract by pressing on the “>” icon next to the contract name in the Deployed Contracts section. You should see the available functions that can be executed. Press on the getRandomNumber button, and confirm the transaction when MetaMask pops up.



18. After a few seconds, your transaction should be approved. While you wait for the chainlink node to fulfill the request, goto https://kovan.etherscan.io/ and search for your deployed contract address (you copied it to the clipboard when you funded your contract with LINK). Once you find your deployed contract on Etherscan, go-to the 'ERC-20 Token Txns' tab. You should see 2 transactions, one incoming transfer of 1 LINK from when you funded your contract, and 0.1 LINK transferred out of the contract for your external data request.


Contract 0x39dd0cb6528a512D4EB6C9B6Ce0B963B4F02Cb45

Contract Overview

Balance:	0 Ether
Token:	$0.00 1

More Info

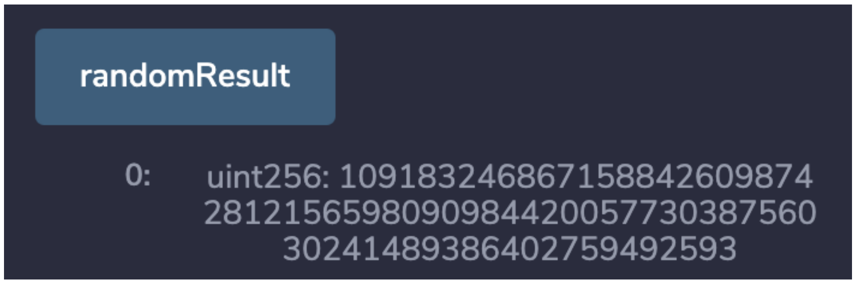
My Name Tag:	Not Available
Contract Creator:	0xf7b4ef69e7cf13c2055… at txn 0xad2cffca1e6835bd0cb…

Latest 2 ERC-20 Token Transfer Events

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

[Download CSV Export]

19. Go back to your deployed contract in Remix. We will now check to see the returned random number. Press the randomResult button to execute the function that returns the value of the public 'randomResult' variable. You should now have a verified random number in your smart contract.



```
0:    uint256: 109183246867158842609874
      281215659809098442005773038756O
      302414893864027594925 93
```

20. Take note of your random number, and repeat step 17. When you receive a new result, you should see that we get a different random number.

Congratulations, you've successfully performed a request for randomness in a smart contract!

Bonus Exercise:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule

1. Modify your contract so that the random number stored is always a number between 1 and 100. To do this, you can use the modulus operator as defined in the code snippet below. Re-deploy and test your contract to see what random numbers it now returns.

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