Steps to create Simple Cryptocurrency Smart Contract and Deploy it on Ethereum Network

Introduction

Ethereum, launched in 2015, is an open-source, blockchain-based, decentralized software platform used for its own cryptocurrency, Ether. It enables Smart Contracts and Distributed Applications to be built and run without any downtime, fraud, control or interference from a third party. Ethereum works by creating smart contracts, implementing it and deploying on the network. We will talk a more about smart contract in the later in the module, but let's first understand Ethereum platform and how it works.

Ethereum : Blockchain platform for developing Decentralized Applications

In simple words, blockchain is a digital record of transactions. Ethereum is also one of the blockchain platforms that can execute an arbitrary code so that you can perform any program on Ethereum. Like every blockchain has their cryptocurrency, Ethereum has the cryptocurrency - Ether (ETH).

The reason companies or manufacturers are shifting to Ethereum is because it can be used to create successfully working decentralized application. There is no need for users to give any kind of personal details, no financial services are needed to mediate any transactions since it a peer to peer network. No government or single organization has control over it and hence it is decentralized.

Creating one's own cryptocurrencies

Using Ethereum we can create our own custom token that can be used a new currency. These tokens can be created with the Ethereum platform use a standard coin API to be compatible with any Ethereum blockchain wallet. This is what we will be doing today.

Smart Contracts

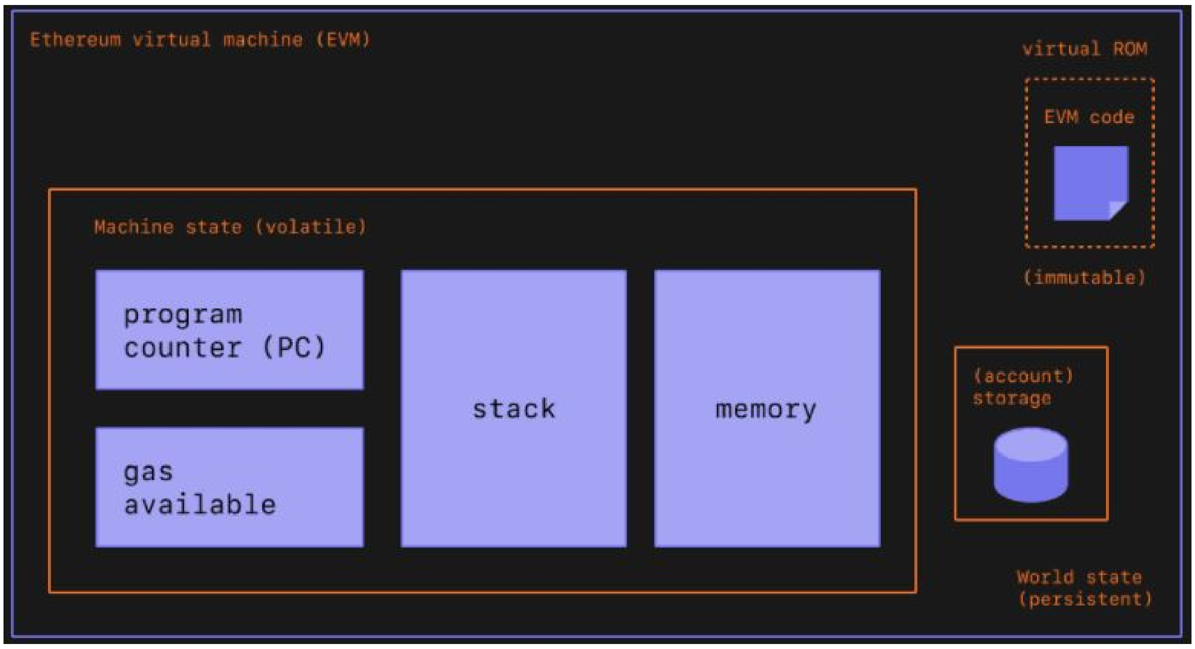
A Smart Contract is lines of arbitrary code, data and functions that runs on Ethereum network. These functions and lines of code decide what should happen at each transaction on the network. They are not controlled by any user and can not be changed mid-transaction. Once deployed on the network, the network recognizes the code and run it as it has been programmed. They are like open APIs and as we know, on the Ethereum network, everything is open to the users. Thus, one smart contract can call other smart contract and use them.

Smart contracts are written in high level languages like Solidity, Golang, Python. This stand-alone script is mostly written Solidity, compiled into JSON and deployed on a particular address on the blockchain. Anyone with will to program a smart contract can write a smart contract and deploy it on the network. All that is needed is knowledge of a good language and tool in their hands.

Ethereum Virtual Machine (EVM)

EVM is like the one single entity maintained by thousands of connected computers (the peer-to-peer network) running an Ethereum Client. The aim of an EVM is to provide runtime environment for the smart contracts. EVM are responsible for the ultimate execution of the smart contracts. Smart contracts are written in high-level languages, which means that machine code is completely isolated from the network. The network has many nodes and each of these nodes runs an EVM instance which permits them to agree on execution of the same instructions.

EVM is implemented in various languages like C++, Java, JavaScript, Python, Ruby and many others. For execution of every instruction there is an execution cost called Gas units. This gas is paid in Ethers, hence a transaction would not take place successfully unless the user has enough Ether reserved that they are willing to pay as gas.



Gas

The gas is the measurement unit used for assigning fees to each transaction with a smart contract. The complexity of the computation during the transaction decides how much gas will be required to run the smart contracts. That is, smart contracts are deployed to the decentralized database for a fee proportional to the containing code's storage size.

Transaction fee = Total Gas Used * gas price

Cryptocurrency

Cryptocurrency is a digital currency or token that can be used as the medium to facilitate transactions of either goods or services. Think of them as the tickets you buy at the arcade game, without those tickets, you can not play the game no matter how much real cash (dollar) you are carrying. This ticket is the token we are talking about here. Today top news at every site includes at least one news on cryptocurrency. In this module we will build out own simple token/cryptocurrency that you can use in your own game perhaps you are developing or in your business later on.

Requirements

Before we start coding the smart contract and deploying it on the network, there are a few requirements we need to meet.

Choosing the Network to work on

We know the Ethereum network, but that is the main network where all the transactions are running. This is what most people are familiar with, but what everyone does not know is that there are other alternate options. There are various test networks running like Ropsten, Kovan and even the localhost network of your computer. We are going to use Ropsten Test Network today to deploy our smart contract. You can use any other network of your own choice if you wish to.

The advantages and reasons of using a test network instead of the main network is that working on the main network requires real Ether, that is real money like . Of course, you can test it on the real network, there is no one stopping you from that, but when we are still in the testing stage, it is advised to save money and use test networks and test Ethers. Test network only needs test ethers and thus it can be done free of charge. So every time our token will be created, the gas fee will be paid by these test ethers. How to get a test ether? We will look at that later.

Wallet

We all have physical wallets that store our physical cash or the cards which are the substitutes of cash. Similarly, there are digital wallets that are needed to store the digital cash. Now that we have our network decided, we need a digital wallet for connecting into the network. I would recommend Metamask for this task.

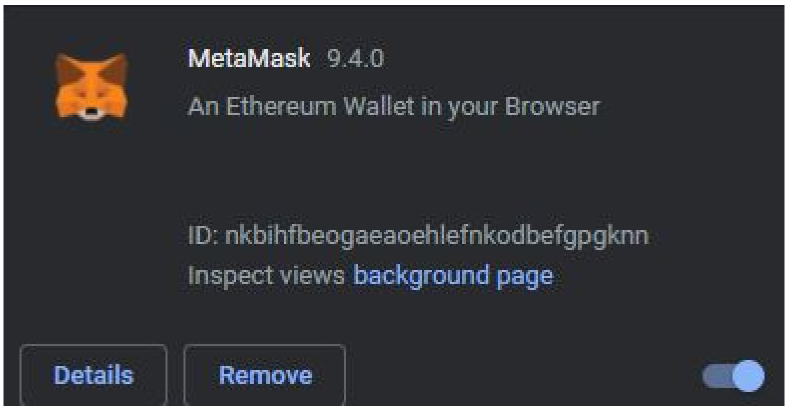
Metamask

Metamask is a digital wallet that runs as an extension to your browser. You can create multiple number of accounts in multiple networks. Each account can have different amount of ethers depending on what account is used how. Metamask is a widely used wallet that helps users connect and interact with smart contracts and the dApps on the web without the need of installing any kind of blockchain or software on the computer. Metamask extension is available for the google chrome browser and thus chrome would be our choice of browser. Let's download and add metamask as our extension.

Step 1 : Installing and downloading metamask

Go to this site and download metamask for chrome. You can check if metamask is added as your extension by opening your extensions for the browser by clicking on the extensions symbol. It looks like this :

Now click on the search button at the top right corner (it looks like a little magnifying glass) and type in metamask over there and search. The extension will display in the result and the symbol is a fox for metamask.

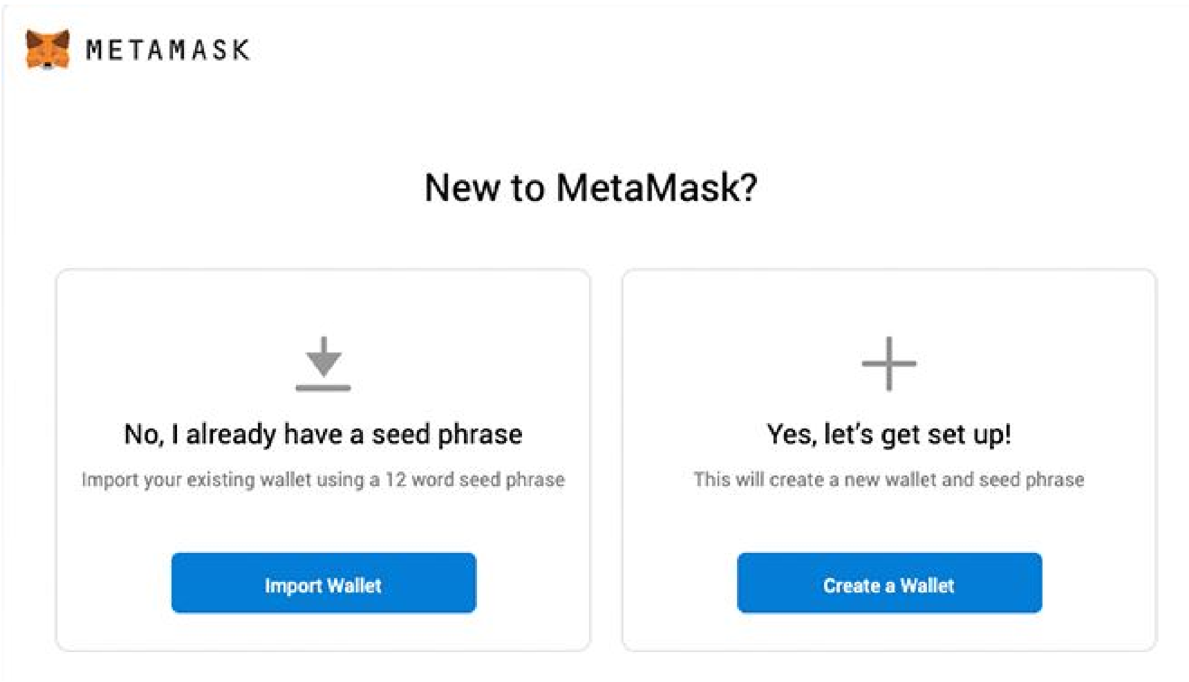

MetaMask 9.4.0

An Ethereum Wallet in your Browser

ID: nkbihfbeogaeaoehlefnkodbefgpgknn

Inspect views background page

Once you have downloaded and added as your extension, open it, and you should see this. Click on the option "Yes, let's get set up!".


METAMASK

New to MetaMask?

No, I already have a seed phrase

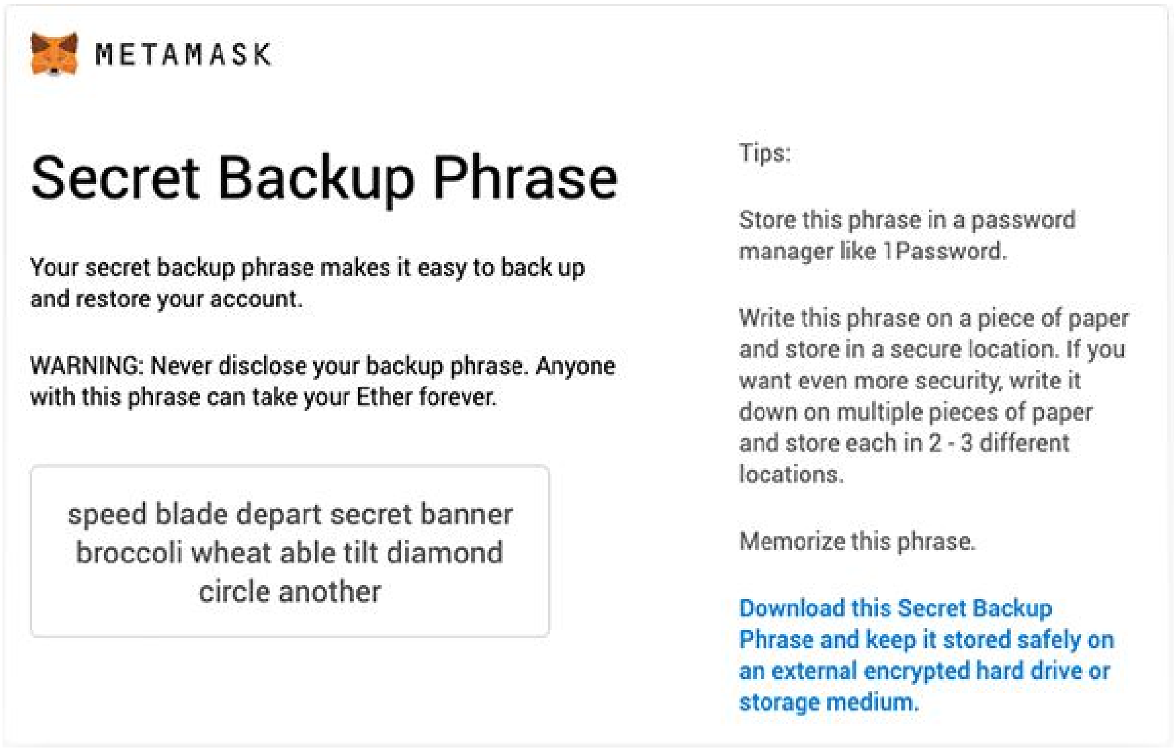
Import your existing wallet using a 12 word seed phrase

Yes, let's get set up!

This will create a new wallet and seed phrase

Step 2 : Logging into metamask

After installing metamask for your chrome, enable it. Once it installed, click on its icon and it should open up in the new tab of the browser. Click on "Create Wallet" and check the "I agree" to proceed further. You will now be asked to create new password. Create a strong password and try not to forget it. Once the password is created, you will be sent a secret phrase for backing up and restoring the account. This is a private phrase meant for you and only you, so do not disclose it to anyone and save it with yourself. This will help you log back into account if you lose your password.



METAMASK

Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

speed blade depart secret banner broccoli wheat able tilt diamond circle another

Tips:

Store this phrase in a password manager like 1Password.

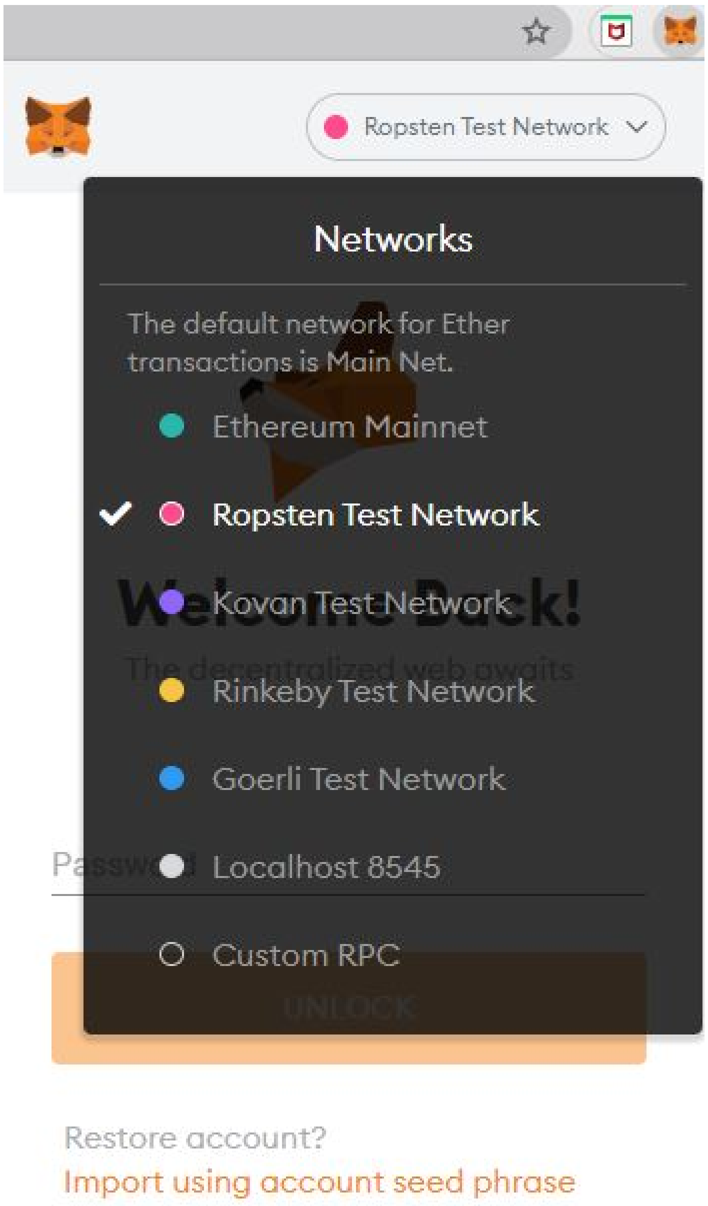
Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down on multiple pieces of paper and store each in 2 - 3 different locations.

Memorize this phrase.

Download this Secret Backup Phrase and keep it stored safely on an external encrypted hard drive or storage medium.

Step 3 : Selecting the network

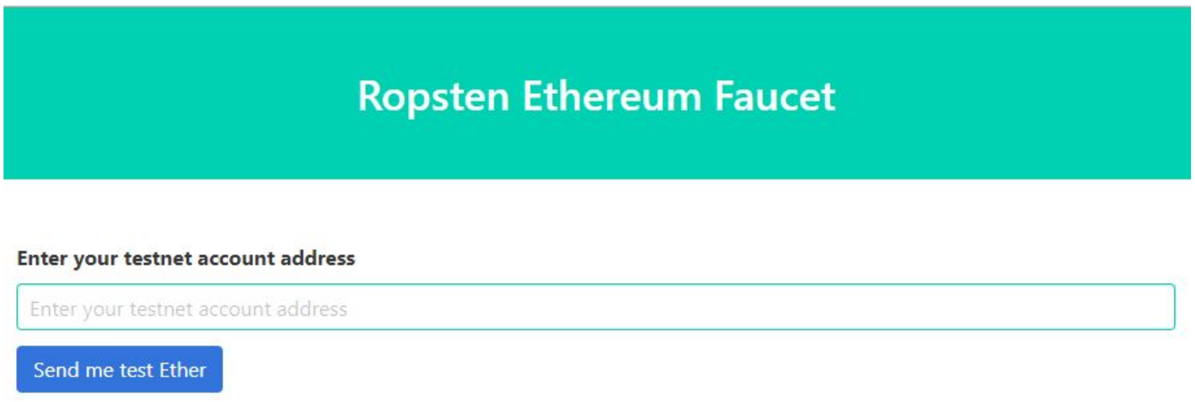
Now select the test network you wish to work on. As discussed earlier, there are various networks - main Ethereum Network, Ropsten Test Network, Kovan Test Network, Rinkeby Test Network, Goerli Test Network and the localhost. I am going to work on Ropsten network.



Step 4 : Adding dummy Ethers

There are different ways to add dummy ethers depending on the ether. One way is to directly click on the Faucet refers to fake test ETH.

Another easy way is to search 'metamask faucet' and clicking on the first link - this link. When you click on the metamask extension, a small pop-up opens, and in the center you should see "Account 1" and a few characters (mix of number and letters). This is your public key for the metamask account, the one you give to others for transferring ethers. Click on it, this should copy your account to the clipboard. Paste the account address on the space available on the website and click 'send me test ether'. Bear in mind, you can only request test ethers once in 24 hours and will be blocked for spamming reasons. This is a super easy method and you will receive 1 ETH.


Ropsten Ethereum Faucet
Enter your testnet account address
Enter your testnet account address
Send me test Ether

Token Information

Now that we have our wallet and network all set, let's decide on how our token would be. There are a few important details about the token that we need to think about before we start.

- **Token Symbol** : This is an identifier for the token, for example, the token for Ether is ETH. You can use 3-5 alphanumeric characters.
- **Token Name** : This is the name of your token, like Ether.
- **Decimal** : How many decimals do you want your token to be able to break down to. For instance, 0 decimal means either the token exists or does not exist. Similarly, "2" decimals means you can fraction the token to smallest size of 0.01.
- **Token Supply** : What is the supply of token for every creation? How many tokens do we want to be created in total at every deployment? Depending on the decimals, the value we enter in the code may differ. For example, if we wish to create 1000 tokens at every deployment, for 0 decimals the total supply will be 1000, but for 2 decimals it will be 100000.
- **Owner Account** : This is the account in the same network as the token which will receive all the token upon creation. This account will be paying the GAS fee. This will be the account we will be working with.

Code

Finally, the code, the smart contract. I have attached the link to my code underneath from where you can download the code. We still need to update the code for personalization and deploy it, so we are not done yet.

Get your code here :

https://github.com/PunyajaMish/ICOIN.git

Remix IDE

Deciding on which tool to use is also important step of the procedure. The best and recommended tool is to use the **Remix IDE**.

http://remix.ethereum.org/

Remix is an open-source web and desktop application. You do not need to install remix and can directly work on the web version. Compiling the smart contract as well as deploying is made easier when using Remix IDE as you will see ahead.

If you do want to download the code, you can simply copy it by clicking on 'Raw' then simply using the ctrl+c and ctrl+v commands for copying and pasting.

Solidity

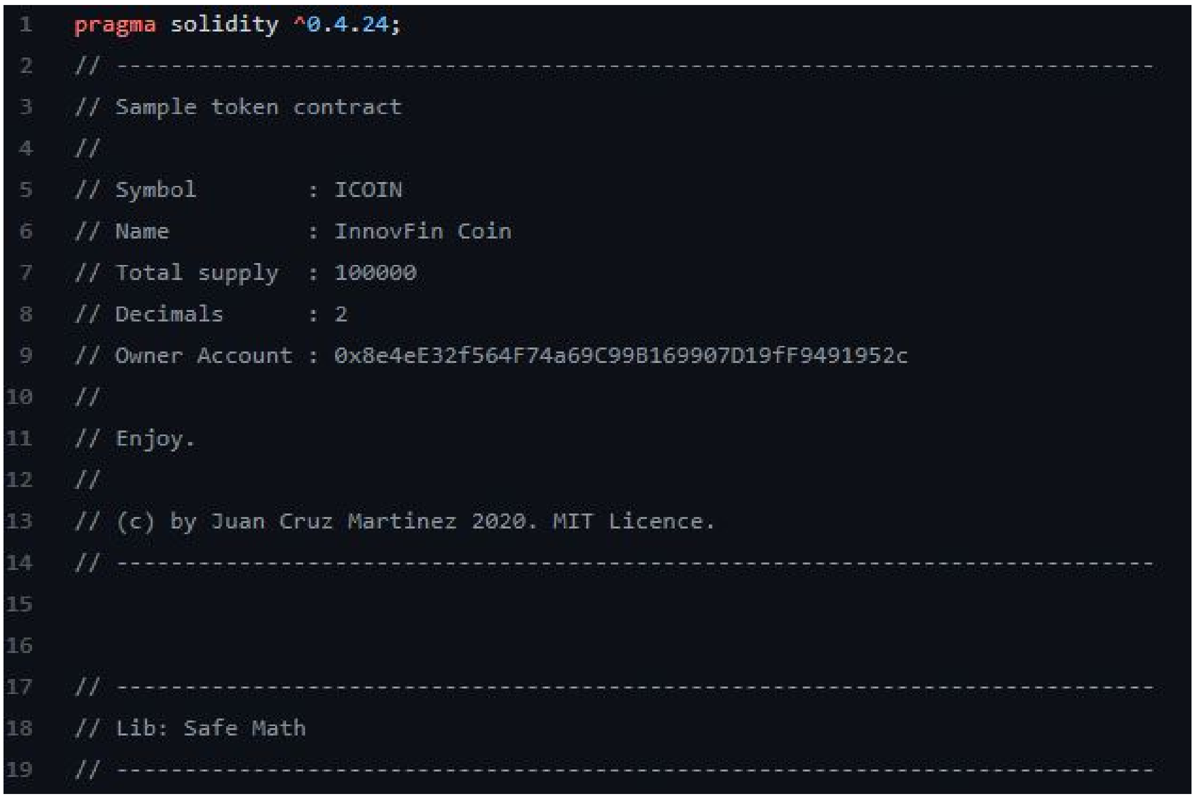
The code is in language **Solidity**. It is a JavaScript-similar language that is developed specifically for creating smart contracts. This is influenced by C++, Python and JavaScript.

It can easily support libraries, inheritance and complex user-defined types. Solidity compiler converts code into the EVM bytecode which is then sent onto the Ethereum network as a deployment transaction.

Personalizing the Code

The first thing after you have downloaded the code, is to update the code and change the token information to your token information. An easy, fast and recommended way to do this is use a text or code editor and use the 'find and replace' option.

The first line of code determines the version of solidity being used. We are using Solidity version 0.4.24. Of course you can change the version to any higher version but that may require you to change a few code lines because of the updates that come with higher versions. For the sake of simplicity, I would say stick with this version.



```
pragma solidity ^0.4.24;
// ----------------------------------------------------------------------------
// Sample token contract
//
// Symbol        : ICOIN
// Name          : InnovFin Coin
// Total supply  : 100000
// Decimals      : 2
// Owner Account : 0x8e4eE32f564F74a69C99B169907D19fF9491952c
//
// Enjoy.
//
// (c) by Juan Cruz Martinez 2020. MIT Licence.
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// Lib: Safe Math
// ----------------------------------------------------------------------------
```

The commented lines of code is where I have written down the token information. You may change these information based on your like. Again, use the find and replace option if you do not wish to look through the entire code and risk the chance of missing a few places. One good tool for editing that will help for this is *VSCode Extension*.

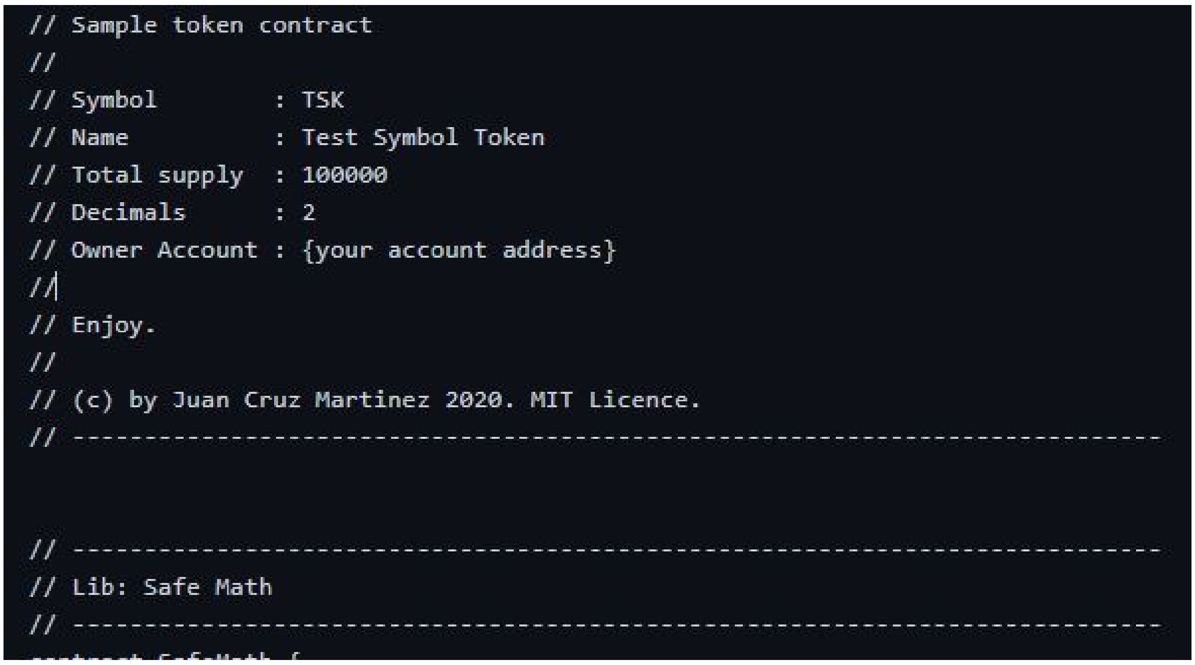
Now, looking at the token information, the symbol for my token is ICOIN. Let's say you wish to change the token symbol to TSK . Now using VSCode, press ctrl + F and type in ICOIN and press Enter. Now click on the small arrow beside the find black, it will open a new blank for you to type in what you wish to replace it with - type in TSK.

Similarly, for token name, change InnovFin Coin to Test Symbol Token.

I have taken decimal as 2 and the total supply I wish to have is 1000 ICOINS, hence for 2 decimals, 100000. You can change it if you want, or let it be the same.

Change the owner account to your own account address in the similar way.

Awesome, now your information should look like :



```
// Sample token contract
//
// Symbol        : TSK
// Name          : Test Symbol Token
// Total supply  : 100000
// Decimals      : 2
// Owner Account : {your account address}
//
// Enjoy.
//
// (c) by Juan Cruz Martinez 2020. MIT Licence.
// ----------------------------------------------------------------------------



// ----------------------------------------------------------------------------
// Lib: Safe Math
// ----------------------------------------------------------------------------
```

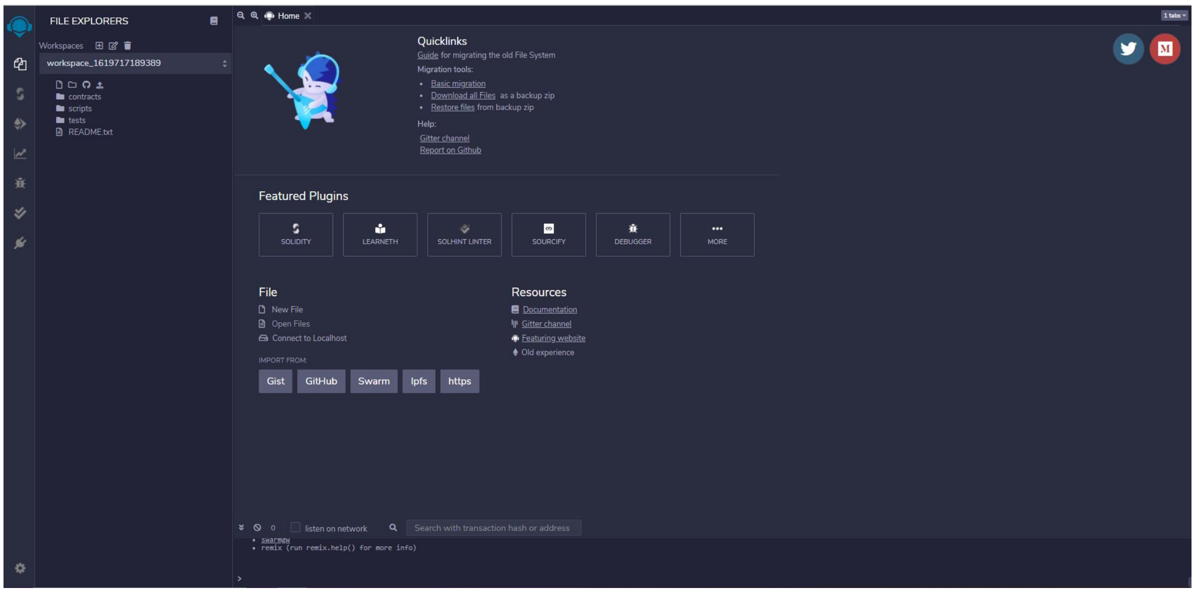
Deployment

Time for deploying our smart contract.

I have used Remix IDE for this purpose because as discussed, it allows easy deployment. You can either install Remix or just click on the following link :

http://remix.ethereum.org/

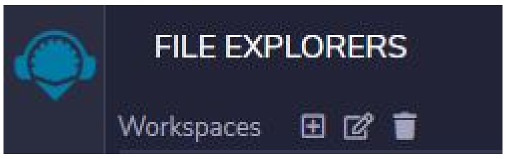
This is how your page should look.



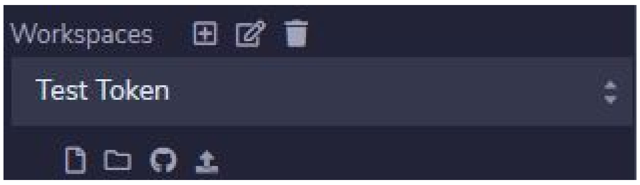
```
• swarmgw
• remix (run remix.help() for more info)
```

Adding code

You can either create a new workspace or start working on the default workspace. If you want to create a new workspace, click on the plus button beside the Word Workspaces. Name it "Test Token" or any other name you wish to name it as.



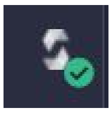
Leave the default files as it is, we do need them but they will not interfere either so we are fine. Click on the contracts folder and then create a new file by clicking on the button at the very left under the test token name. Name is 'tsk.sol'. Paste your code in there.



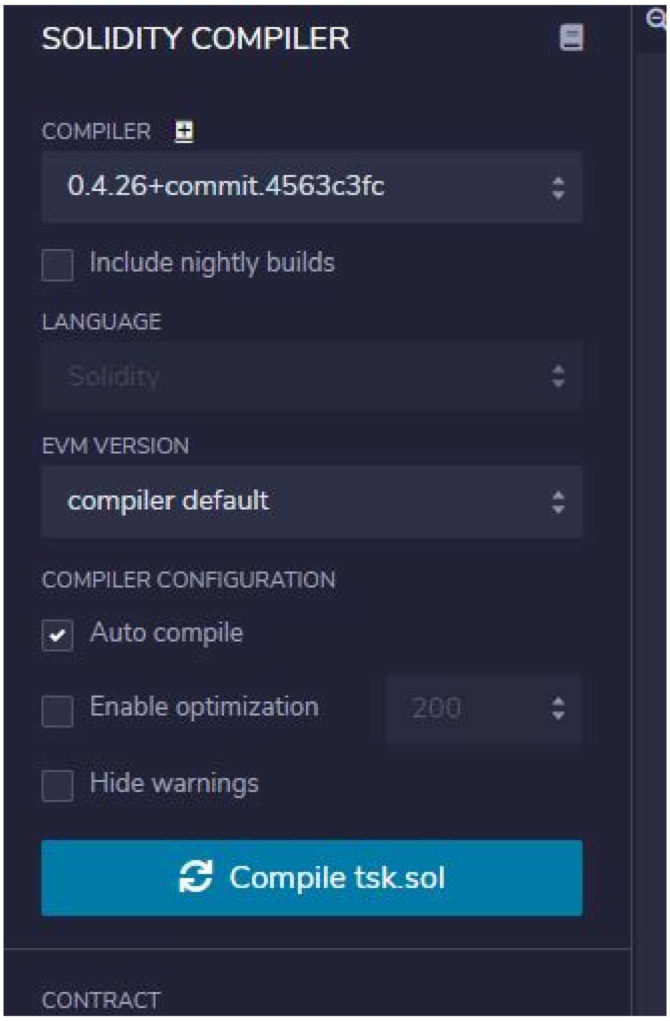
Once you paste the code, a new folder is automatically generated under the name artifacts, which contains the JSON files for this smart contract. We do not need to edit or delete them. Let them be.

Compiling the code

Time to compile the code. There are a lot of symbol at the very left of Remix in a column. The symbol with either a green check mark or red cross is the compile button. This should be the second button under the files in which we currently are.



Once in compile, you should see errors if you have made any mistake till now. Let's hope you also get a green check mark.


SOLIDITY COMPILER

COMPILER: 0.4.26+commit.4563c3fc

[ ] Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION

[✓] Auto compile

[ ] Enable optimization 200

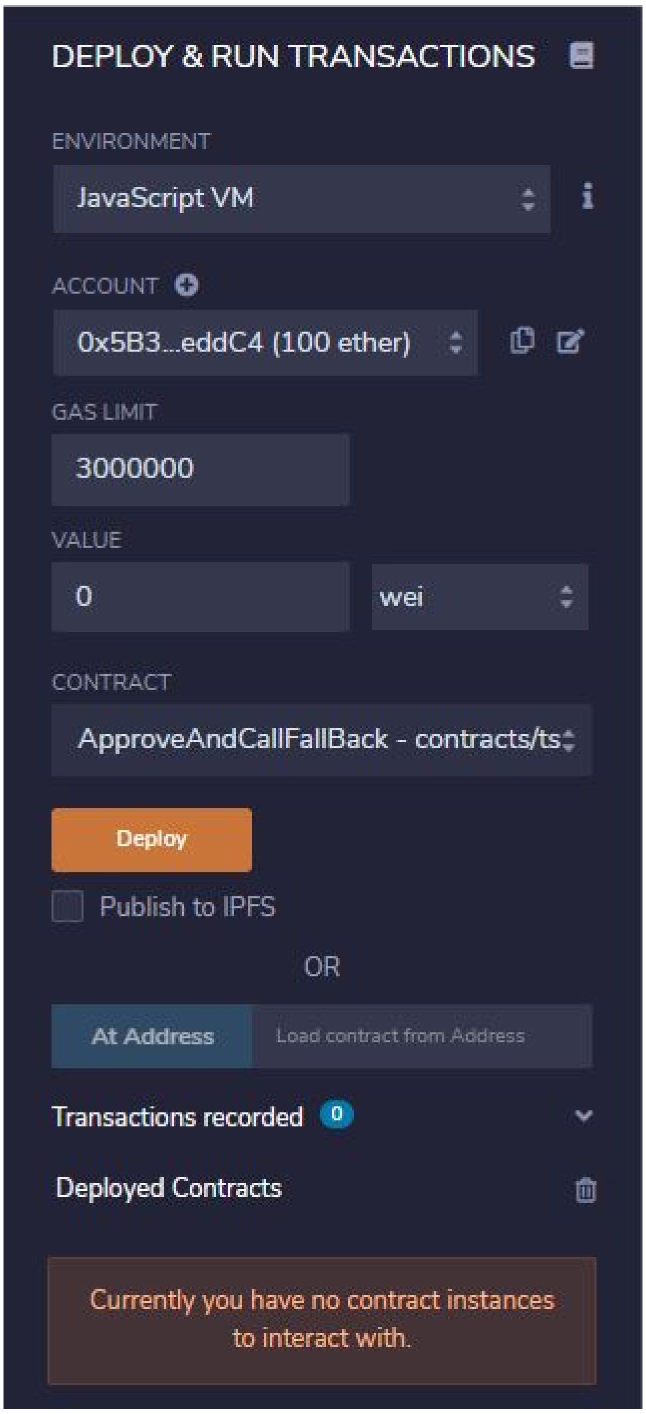
[ ] Hide warnings

As you can see, the compiler version is already set to the nearby version of our solidity version to ensure proper compiling. Now click on the compile tsk.sol and if it shows any errors then fix those errors. If you have followed all steps properly, it should not show any errors.

Deployment

It is time now that we deploy our smart contract. Click on this symbol.

Once in there, you should see this. Now these are important steps, so pay attention.


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0x5B3...eddC4 (100 ether)

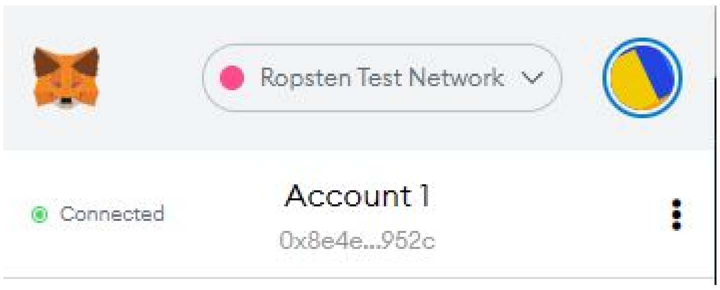
GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
ApproveAndCallFallBack - contracts/ts

Currently you have no contract instances to interact with.

The **Environment** is by default set to "JavaScript VM". However, since we are trying to connect to the web extension of metamask. Click on the dropdown and change it to "Injected Web3". The moment you do this, there will be a metamask pop up asking you to confirm if you want to connect with this site. Click confirm. And now your sign should change from 'not connected' to 'connected' with a green dot. If you have previously connected to the network, this pop up will not come and automatically connect without confirmation.



Next change the **CONTRACT** to your contract tsk.sol. So that we are deploying the right contract. If you do not do this then basically, you are not executing your smart contract.

We are ready now.

Hit **Deploy**. Hit **Confirm**

The deployment process may take time form 2-3 minutes to 10 minutes on the main network. It usually depends on the status of the network at that time. Usually Ropsten should not take more than 2-3 minutes Sometimes, it may even work as fast as in 20 seconds.

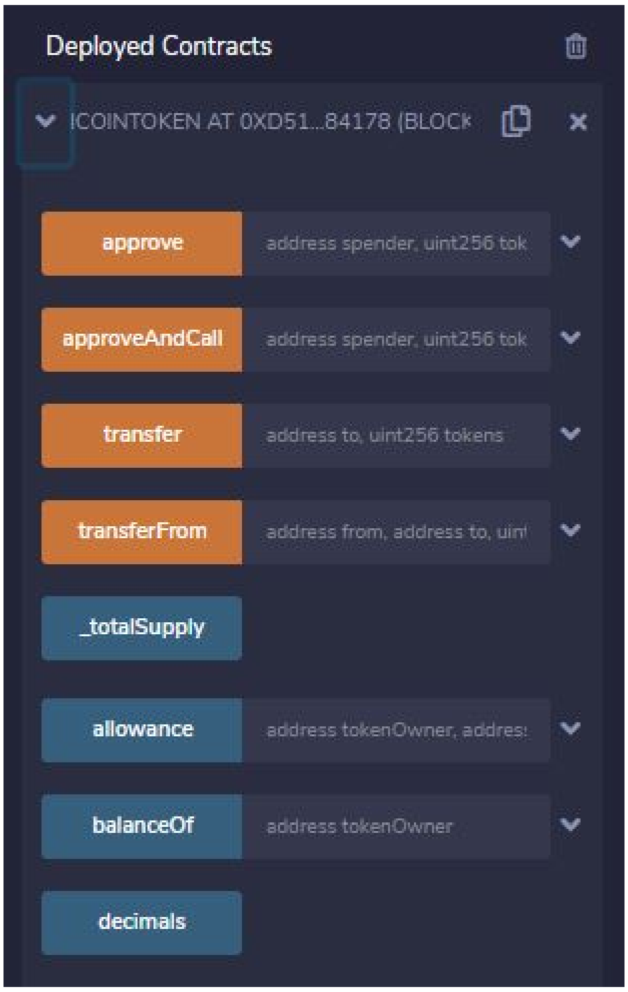
After the process is completed you should get a confirmation on the screen with the information of your deployed contract including the address it just got saved in.

There you go, your token is not live. But you can't see it on your metamask account yet. Why is that?

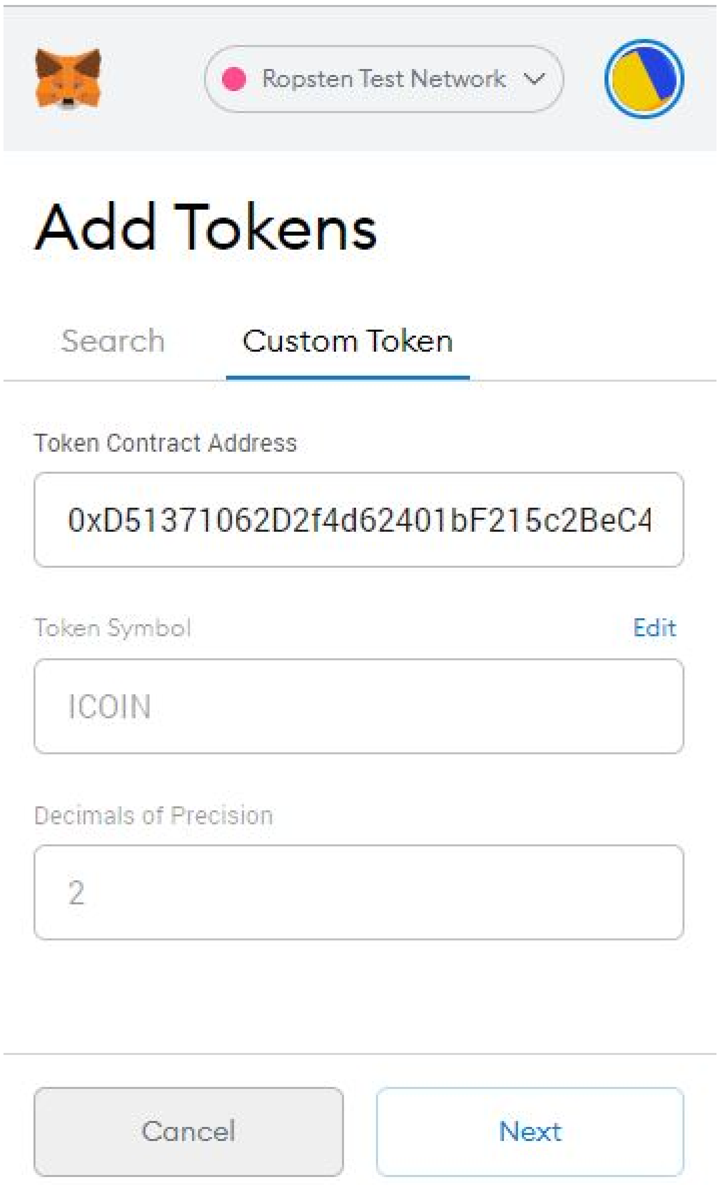
Metamask configuration and retrieving token

Right now, the token exists on the Ethereum network and can be used for transactions but we can't access it yet. How do we do that? The problem is right now your wallet is not aware of this token. To make it aware, we need to add this token as custom token.

When the deployment is done, click on the copy symbol - this copies the address where the token is residing.


Deployed Contracts
ICOINTOKEN AT 0XD51...84178 (BLOCK
approve
address spender, uint256 tok
approveAndCall
address spender, uint256 tok
transfer
address to, uint256 tokens
transferFrom
address from, address to, uin
_totalSupply
allowance
address tokenOwner, addres
balanceOf
address tokenOwner
decimals

Now open your metamask, and click on Add token below in the Assets. Choose custom token as your option and then paste the address over there. The other fields are auto-filled from the address. Now click 'Next' and then 'Add tokens'. Now you will have 1000 tsk tokens in your asset.


Ropsten Test Network

Add Tokens

Search | Custom Token

Token Contract Address: 0xD51371062D2f4d62401bF215c2BeC4

Token Symbol: ICOIN

Decimals of Precision: 2

Conclusion

There, you are all done!

Today we learned how to build and deploy our own custom token and cryptocurrency by following the ERC20 interface. The process is super easy and this is one of the easy smart contract projects any newcomer to blockchain can start with.

I hope you enjoyed this tutorial and learnt something new!

Thank you!