Hands-on Workshop Exercise

https://github.com/TcsInnovationEvent

PS: Make sure you have cloned/downloaded the HelloEthereum respository

1. Create a folder HandsOnLab
2. In a terminal window, CD to the folder you created
3. Execute > truffle init
4. If you are using windows, rename truffle.json to truffle-config.json

#1 Calculator Contract

Code a contract that would expose mathematical operations as contract functions. The contract should be initialized with a number and all subsequent operations will change the state of the contract that is the number held in the contract storage.

1. In your project run > truffle create contract Calculator

2. Add a storage variable number of type int public number

3. Add the following functions
   - addToNumber(int num) subtractFromNumber(int num)
   - multiplyWith(int num) divideBy(int num)

4. Open the remix editor http://remix.ethereum.org

5. Copy/Paste your contract code

6. Deploy the contract to ROPSTEN

7. Execute various functions and validate the results

#2 Voting

Contracts can be used for managing the Voting process. Voting on Blockchain will be transparent to all as a result there will be more trust in the system. In this exercise you will create a very simple voting contract. The contract will hold a “Question” and anyone can invoke the vote function to respond with Yes/No.

1. In your project run > truffle create contract Voting

2. Copy & Paste the code from the GitHub Repo/HelloEthereum/contracts/VotingSkeleton

3. Code all of the functions
   - Constructor should take a string that would initialize the contract with a question
   - Contract will store to state variables, a yes count and a no count
   - For the time being assume – *everyone is allowed to vote more than once*

4. In a terminal window launch testrpc

5. Update the 2_deploy_contracts.json + run > truffle migrate

6. Create a test file > truffle create test Voting + Copy/Paste code for testing from the HelloEthereum project

7. Run the test > truffle test ./test/voting.js

*Solution in the repository

#3 Launch ERC20 Token

Exercise: ICO launches a token that investors buy from the token owner. An ICO needs a solid business idea that would appeal to the potential investors. We will leave the business idea to you but in this exercise you would learn how to create a token that is ERC20 compliant and can be used for launching your ICO campaign.

*Solution in the repository

ERC20 is a standard for Creating Tokens

- Implement a set of functions

- Tokens can be managed using the tools such as Wallet & MetaMask

  - Check your token balance
  - Transfer your token to others
  - Approve others to spend your tokens

ERC20 Functions

```
/// total amount of tokens
/// Compiler generates a function totalSupply()
uint256 public totalSupply;

/// digits after decimal points
/// Compiler generates a function decimals()
uint256 public decimals;

/// symbol for your token
/// Compiler generates a function symbol()
string public  symbol;

/// description for your token
/// Compiler generates a function description()
string public  description;

/// returns balance of the _owner
/// @param _owner The address from which the balance will be retrieved
/// @return The balance
function balanceOf(address _owner) constant returns (uint256 balance);

/// transfers specified number of tokens from=msg.sender to=_to
/// @notice send `_value` token to `_to` from `msg.sender`
/// @param _to The address of the recipient
/// @param _value The amount of token to be transferred
/// @return Whether the transfer was successful or not
function transfer(address _to, uint256 _value) returns (bool success);
```

```
/// this requires implementation of the allowance & approve
/// @notice send `_value` token to `_to` from `_from` on the condition it is approved by `_from`
/// @param _from The address of the sender
/// @param _to The address of the recipient
/// @param _value The amount of token to be transferred
/// @return Whether the transfer was successful or not
function transferFrom(address _from, address _to, uint256 _value) returns (bool success);

/// msg.sender approves _spender for spending _value of his tokens
/// @notice `msg.sender` approves `_spender` to spend `_value` tokens
/// @param _spender The address of the account able to transfer the tokens
/// @param _value The amount of tokens to be approved for transfer
/// @return Whether the approval was successful or not
function approve(address _spender, uint256 _value) returns (bool success);

/// checks the max _spender can spend _owner tokens
/// @param _owner The address of the account owning tokens
/// @param _spender The address of the account able to transfer the tokens
/// @return Amount of remaining tokens allowed to spent
function allowance(address _owner, address _spender) constant returns (uint256 remaining);

/// emit the events for transfer and transferFrom
event Transfer(address indexed _from, address indexed _to, uint256 _value);

/// emit from approve event
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

Part-1 ERC20 Functions

1. Create a truffle project

2. Create a contract **> truffle create contract MyERC20Token**

3. Copy/paste code from **ERC20Standard.sol** to **MyERC20Token.sol**

4. Change the code to reflect:
   - totalSupply, name, symbol, decimals
   - Create a mapping to store balances

5. Code just the following functions in version-1 of your token

```
function balanceOf(address _owner) constant returns (uint256 balance);
function transfer(address _to, uint256 _value) returns (bool success);
```

6. Return false or 0 from the following functions

```
function transferFrom(address _from, address _to, uint256 _value) returns (bool success);
function approve(address _spender, uint256 _value) returns (bool success);
function allowance(address _owner, address _spender) constant returns (uint256 remaining);
```

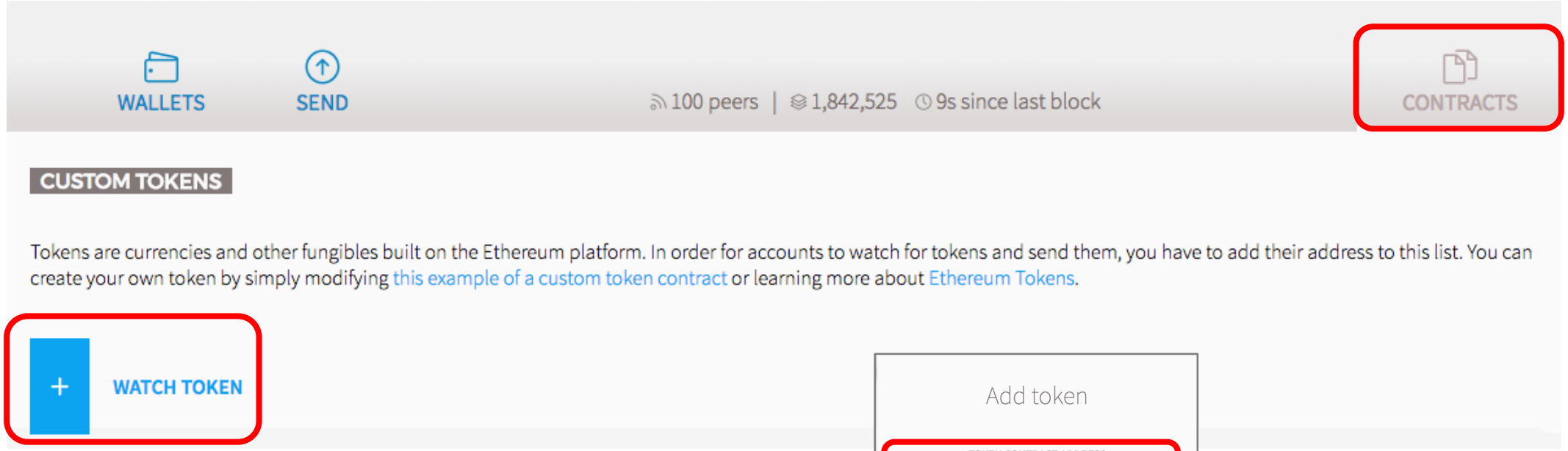
Part-2 ERC20 Functions

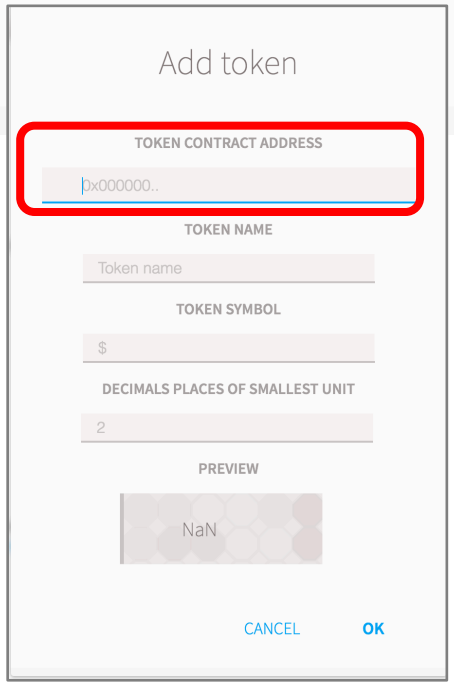
1. Create a truffle test file > truffle create test token_test
2. Copy/paste code from HelloEthereum/token_test.js to your test case file
3. Review the test case code & make changes as needed e.g., adjust the total_supply
4. Launch TestRPC
5. Make the change to 2_deploy_contract.js to deploy your contract
6. > truffle migrate
7. Run test > truffle test ./test/your_test_file_name.js

Part-3 Deploy using Remix

1. Open Remix
2. Create a new contract file – copy your token code and paste it in remix
3. Copy the address of the contract

Part-4 Track your token in the wallet





1. Paste your token address & confirm details
2. Change the decimals = 0
3. You should see the balance

Part-4 Share address with anyone interested in your ICO

1. Send your token contract address to friends & ask them to add to their wallet
2. Get friend's account information & send some tokens
3. Do they see a balance? Yes – test is good

CONGRATULATIONS!!!

On the launch of your ICO ☺