smerf.net

Ethereum Blockchain Development Week 18

Introduction

The intention of this lab is to look at functions and how variables are stored. All the exercises are completed in Remix IDE.

Code Completion

Writing code in a new language can be a steep learning curve. The approach here is to provide some code with underscores (_) that you are required to complete. These underscores are there to help you. By completing these exercises you will be improving your skills and knowledge of Solidity.

Each exercise starts on a new page. The red numbers in the right-hand margin are estimated minutes you should spend on each exercise. T

1 Network using MetaMask and Ganache

15-20

As shown in the lecture the aim is to send ether between two accounts and explore the limitations. The objectives in this exercise are to set up Ganache and export the addresses to a Wallet. Complete the following:

1. Open VM and run Ganache
2. Open MetaMask Wallet
3. Add a Ganache account to the metamask Walllet. Repeat this for all accounts. Label these accounts A1, A2, ..., A10.
4. Send 10 Ether between A1 and A2 and answer the following:
   (a) What is the value of the block number?
   (b) What is the value of the TX hash?
   (c) What is the value of Gas used to complete the account?
   (d) Confirm the amount sent in Wei and convert it to Ether.
   (e) What is the value of the block hash?
   (f) What is the timestamp for the block?
5. Repeat the above, with different amounts of ether and between different accounts. Study the results and answer the following:
   (a) Are the block numbers consecutive? Would they be in a live network?
   (b) Is the amount of Gas the same, regardless of the amount transferred? Is this the same in a live network?
   (c) Are the timestamps, TX hashes, block hashes the same?

2 Payroll

The overall objective of this to create a payroll that is owned by an EOA, delivered on a Contract Address, and pays employees in Ether when the transaction is initiated by the owner.

There are some access issues that are considered and are resolved using access modifiers.

Each subsection adds an important part of the contract and starts on a new page.

2.1 Access Modifiers

15–20

This exercise is to be completed in the VM provided and is to be deployed on a private network setup using Remix-ide desktop and Ganache. The contract will be owned by the first account in ganache. The other accounts will form the employees.

To start this exercise create a contract in the remix-ide editor as shown in Fig. 2.1

Once completed continue to the next section.

Code Completion

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Payroll {
    address[] public employees;
    mapping (address => uint) accounts;
    address public owner;
    uint public balance;
 bool init = false;

    //EVENTS
    event paid(address employee, uint amount, uint timestamp);

    //ACCESS MODIFIERS
    modifier ownerOnly(){
            require(msg.sender == owner, "Insufficient access: owner only");
            _;
    }

    //CONSTRUCTOR
    constructor() payable {
            owner = msg.sender;
            balance = msg.sender.balance;
            payable(address(this)).transfer(msg.value);
    }
```

Figure 2.1: Solidity listing of payroll functions and state variables.

2.2 `require`

10–15

Add the following functions displayed in Fig. 2.2 to the contract. Then complete the next section.

Code Completion

```
//TRANSFER FUNDS, AMOUNT, TO AN ADDRESS
function payAmountTo( address payable _to, uint _amount) payable public ownerOnly {
    require(_amount < balance, "Insufficient Funds");
_to.transfer(_amount);
}
```

Figure 2.2: Solidity listing of payroll functions.

2.3 Payroll

15–20

The objective is to pay all the employees 10,000 Wei from the owner account using a for loop. This is much like a payroll system.

Your task is to complete the contract and transfer 10,000 Wei using the function in fig.2.3.

Add two functions to get and return the owner's balance and address.

Finally, compile and deploy this on a ganache private network.

Code Completion

```
    //PAY EMPLOYEES
    function payEmployees() payable public ownerOnly{
        address payable _to;
        uint _amount=10000; //set this to 10,000 Wei
        for (uint8 i=0; i<employees.length; i++){
           _to = payable(employees[i]);
           payAmountTo(_to, _amount);
           accounts[_to]+=_amount;
           balance-=_amount;
           emit paid(employees[i], _amount, block.timestamp);
        }
    }

    fallback() external payable{}

    receive() external payable{}
```

Figure 2.3: Solidity listing of payroll, receive and fallback functions.

Code Completion

```
    function initialise() public ownerOnly {
    requite(!init, "Already initialised");
        employees.push(0x5D54265716A05582dd039251D9321AC3ce221a1a);
        accounts[employees[0]]=employees[0].balance;
        employees.push(0xAa128cb783639f52184c4A5c2C5Fd623f18C2049);
        accounts[employees[1]]=employees[1].balance;
        employees.push(0x15bdBe61bE27125451885815385 8B9fd8cFCE555);
        accounts[employees[2]]=employees[2].balance;
    init=true;
    }
 }
```

Figure 2.4: Solidity listing of initialise function.

2.4 Initialise

15–20

Sometimes as testers, it may be necessary to speed things up. This can be done by creating an initialise function that can enter the addresses automatically upon being invoked.

Complete the code in Fig. 2.4, compile, deploy and answer why the require function is needed?

3 Truffle

20–25

These are the steps to follow to complete a unit test.

3.1 Setup

Complete the following in the VM provided:

- open a terminal for command line.
- make a directory, `mkdir truffleProject`
- change to this directory, `cd truffleProject`
- type `truffle init`
- type `code .`

Code Completion

```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract mfc{
    uint8 sum;
    function add(uint8 _x, uint8 _y) public returns (uint8){
        require( _x + _y <256);
        return uint8(_x + _y);
    }
}
```

Figure 3.1: Solidity listing of `mfc` contract.

3.2 Contract

In the contracts folder create a new file, `mfc.sol`. Write this as displayed in Fig. 3.1.

Code Completion

```
networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache-cli, geth or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
   development: {
    host: "127.0.0.1",     // Localhost (default: none)
    port: 7545,            // Standard Ethereum port (default: none)
    network_id: "*",       // Any network (default: none)
   },
```

Figure 3.2: Solidity listing of `mfc` contract.

3.3 Configuation

In the `truffle-config.js` file, make the following changes as indicated in Fig. 3.2.

Code Completion

```
const mfc = artifacts.require("mfc");
module.exports = function (deployer){
    deployer.deploy(mfc);
}
```

Figure 3.3: Solidity listing of `2_mfc.js`.

3.4 Migrations

In the migrations folder create a new file `2_mfc.js` and type the code in Fig. 3.3.

Code Completion

```
const mfc = artifacts.require("mfc");
module.exports = function (deployer){
    deployer.deploy(mfc);
}
```

Figure 3.4: Solidity listing of `test-mfc` contract.

3.5 Assert

Create a new file in the `test` directory, named, `test-mfc.sol` and complete the code in Fig. 3.4

3.6 Testing

Finally, test by the following instructions:

- Open a terminal
- make sure you are in the `truffleProject` directory created earlier
- type `truffle compile`
- type `truffle test`

4 Reading

Read Ch. 8 in [**?**]

References

[1] Ritesh Modi. *Solidity Programming Essentials*. Packt, 2018.