Ethereum
Blockchain Development
Week 15

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. Coding requires practice, so you will need to practice these and other exercises outside this lesson as independent study.

Each exercise starts on a new page.

```
//____-License-_________: GPL2.0
pragma ________ ^0._.0_

_________ Test{
  uint ______ age_
  ______ public name_

  function setAge(uint x) ______{
    age = _;
  }
  function setName(string x) public {
    name = x;
    }
  function getAge() public ____ returns {
    return ___;
  }
  function get name() public ____ returns {
    return ____ ;
  }

}
```

Figure 1: Listing for `ex1.sol`, type this in the Remix environment

1 Exercise 1

This is a simple exercise to test the environment. Create a new environment in the remix IDE and add a new file, `ex1.sol`, and type the following code in fig. 1. Compile and deploy this contract and test if it is running correctly.

Correct the code in Fig.1 and complete the underscore lines. The code should have:

- two state variables of data type `uint` & `string`
- Create two functions that are able to set the values of respective state variable.
- Create two functions that returns the values of the state variables.

```
//____-License-__________: GPL-3.0
pragma solidity _0._.0_
________ ex2{
    ____ x;
    ____ y;
    ________ setXY(uint _, uint _) ______ {
        x = a;
        y = b;
    }
    ________ modY() public  _______ (uint){
        x += y;
        y *= 10;
        return _;
    }
}
```

Figure 2: Listing for `ex2.sol`, type this in the Remix environment

2 Exercise 2

Create a new file, `ex2.sol`, in the existing environment in Remix. Complete the code in fig. 2 and compile and deploy this contract. Then test and see if this contract is running correctly.

Add two functions called `getX` and `getY` that will be able to return the value of state variables `x` and `y`, respectively. Recompile and re-deploy the contract and test if it is successful.

Write two functions called `setX` and `setY` that are able to change the values of the state variable.

```
//SPDX-_______-Identifier: ___
______ solidity ^0.8.0;
_________ ex3{
    ____[2] x = [10, 20];
    ____[2] y = [1, 2];
   _________ ModXY() ______ _______ (uint){
        x = y;
        y[1] = 7;
        ______ x[1];
    }
}
```

Figure 3: Listing for `ex3.sol`, type this in the Remix environment

3 Exercise 3

Create a new file, `ex3.sol`, in the existing environment in Remix. Complete the code in fig. 3 and compile and deploy this contract. Then test and see if this contract is running correctly.

Take care to study this function. It is assigns state variable y to state variable x. In other languages, this may be done by reference. In Solidity this is done by value, since they are state variables. To test this the function makes a change to y and returns x, to see if x remains isolated and independent.

```
__SPDX-_______-Identifier: ___
______ solidity _______
_______  ex4_

  uint ______ stateUint=10;

  ______on getUint__ ______ _______ (uint)_
    uint localUint =  20;
    stateUint = localUint;
    localUint = 2;
    ______ stateUint;
  _
_
```

Figure 4: Listing for `ex4.sol`, type this in the Remix environment

4 Exercise 4

Create a new file, `ex4.sol`, in the existing environment in Remix. Type in the code in fig. 4 and compile and deploy this contract. Then test and see if this contract is running correctly.

4.1 Rule 6 & 7

This is clearly an implementation of Rule 6 in the lecture on storage rules. The principle here is that when a state variable is assigned a local variable they both remain indepedent. Rule 7 is the the converse of Rule 6, and a local variable is assigned a state variable. Write a function that demonstrates this rule, compile, deploy and test it.

4.2 Events

Write an event that commits the values of the `uint` parameter to the blockchain. Name this event `display`. Emit the value of the local variable in the newly created function above using the `emit` instruction.

Compile, deploy and test it and thus demonstrate that the value is written to the blokchain and that it is independent of the state variable value and memory location.

```
__SPDX-_______-Identifier: ___
______ solidity _______
_______  ex4_

  uint ______ stateUint=10;

  ______on getUint__ ______ _______ (uint)_
    uint localUint =  20;
    stateUint = localUint;
    localUint = 2;
    ______ stateUint;
  _
_
```

Figure 5: Listing for `ex5.sol`, type this in the Remix environment

5 Exercise 5

Create a new file, `ex5.sol`, in the existing environment in Remix. Type in the code in fig. 5 and compile and deploy this contract. Then test and see if this contract is running correctly.

6 Exercise 6

Create a new file, `ex6.sol`, in the existing environment in Remix.

Create a constant unsigned integer state variable, named `size`, and assign it the value 5.

Create an unsigned integer array of size, 5 (use variable above), as a state variable named, `sa`.

On the same declaration line populate the array with the following values: 10, 20, 30, 40, 50.

Write an event, name `display`, that will pass an unsigned integer as a parameter.

Write a function that calculates the sum of all the integers in the array and emits the sum in the event `display`.

Then test and see if this contract is running correctly.

Reading

- Chapter 3 in [**?**]
- Chapter 1 in [**?**]