A Workshop on

Blockchain Technology

Lecture 2

Hands on Coding With Solidity

Smart Contract and Solidity

Lecture 2

Hands on Coding With Solidity

What is smart contract?

- The code defines the mechanisms of the transaction
- It is the final arbiter of the terms, no human intervention.
- Transaction type: If something happens, then something else happens.

Key Features

Self-verifying

Self-executing

Tamperproof

Solidity

Object-oriented

Contract-oriented

High-level language

Influenced by C++, Python, and JavaScript

Target Ethereum Virtual Machine (EVM)

Before Coding...

Lecture 2

Hands on Coding With Solidity

Storage and Memory

Storage: where all the contract state variables reside. Every contract has its own storage and it is persistent between function calls

Memory: hold temporary values and gets erased between (external) function calls and is cheaper to use

Value Types

Lecture 2

Hands on Coding With Solidity

Value Types : Boolean

Keyword: Bool

The possible values are constants i.e., true or false

For examples:

bool isRegistered; // to check whether voter has registered or not

Value Types : Integer

Keyword: int/uint (uint8 to uint256 (unsigned of 8 up to 256 bits) and int8 to int256)

Signed and unsigned integers of various sizes.

Example:

```
contract MySample{

uint256 UnsignedInt =50;

}
```

In the above statement, we have created a uint called UnsignedInt & set it to 50.

Value Types : String

Solidity supports String literal using both double quote (") and single quote ('). It provides string as a data type to declare a variable of type String.

```
contract SolidityString {

   string data = "String";

}
```

Value Types : Address

Keyword: address

Holds a 20-byte value (size of an Ethereum address). Address types also have members and serve as a base for all contracts.

Members : balance and transfer

Address Ram;

Ram.balance; // give balance give balance of Ram address.

payable(Ram).transfer(10);// transfers 10 amount to Ram address.

Value Types : Address

Keyword: address

Holds a 20-byte value (size of an Ethereum address). Address types also have members and serve as a base for all contracts.

Members : balance and transfer

Address Ram;

Ram.balance; // give balance give balance of Ram address.

payable(Ram).transfer(10);// transfers 10 amount to Ram address.

Reference Types

Lecture 2

Hands on Coding With Solidity

Struct

Solidity provides a way to define new types in the form of structs. Structs are custom defined types that can group several variables.

```
Struct Fruit{
String name;

uint256 price;

}
```

Array

Arrays in Solidity can have a compile-time fixed size or they can be dynamic.

Collection of similar objects

uint256[] naturalNum;

Array of structs like:

Fruit [] fruits;

Mapping

Mappings can be seen as hash tables which are virtually initialized such that every possible key exists and is mapped to a value whose byte-representation is all zeros: a type's default value.

Similar to Dictionary in Python.

Mappings are declared as:

Mapping(_Keytype => _ValueType)

Control Structures

if, else, while, do, for, break, continue, return, ? :,

Similar to semantics known from C or JavaScript.

Functions

Here is how a function is declared in Solidity.

Syntax:

```
function function_name(parameter_list) scope returns(return_type) {
       // block of code
}
function getTotal(uint256 qty, uint256 amount) public view returns (uint256 total){
return qty*amount;
}
```

You would call this function like:

```
sampleFunc("Shashank", 10000);
```

Functions Modifiers

It is used to easily change the behavior of the functions.

Condition to be checked before function call.

Visibility Specifier

Lecture 2

Hands on Coding With Solidity

Variable Visibility

Public : Allow other contracts to read their values. The compiler automatically generates a getter function for them.

Private : Default visibility level. These variables can only be accessed from within their contract and from derived contracts.

Internal : Similar to internal variables but they are not visible to derived contracts.

Function Visibility

Public : Accessible by external account transactions or contracts. They are also visible by other functions within the contract.

Private : Accessible to only inside the smart contract.

Internal : Accessible to derived smart contract also.

External : Accessible externally only, not inside the smart contract.

Function Behaviour

constant or view: These functions won’t modify any state variable values.

pure: Don’t have any side effects. They don’t read nor write any variables in storage.

payable: Payable functions accept incoming payments.

Predefined Global Variables

Lecture 2

Hands on Coding With Solidity

Global Variables

block.chainid (uint): current chain id

block.coinbase (address payable): current block miner's address

block.difficulty (uint): current block difficulty

block.gaslimit (uint): current block gaslimit

block.number (uint): current block number

block.timestamp (uint): current block timestamp as seconds since unix epoch

Global Variables

gasleft() returns (uint256): remaining gas

msg.value (uint): number of wei sent with the message

msg.data (bytes calldata): complete calldata

msg.sender (address): sender of the message (current call)

msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)

tx.gasprice (uint): gas price of the transaction

Requirements

Node Installation

Hardhat Boilerplate Setup

Metamask Chrome Extension

The Real Game
Coding

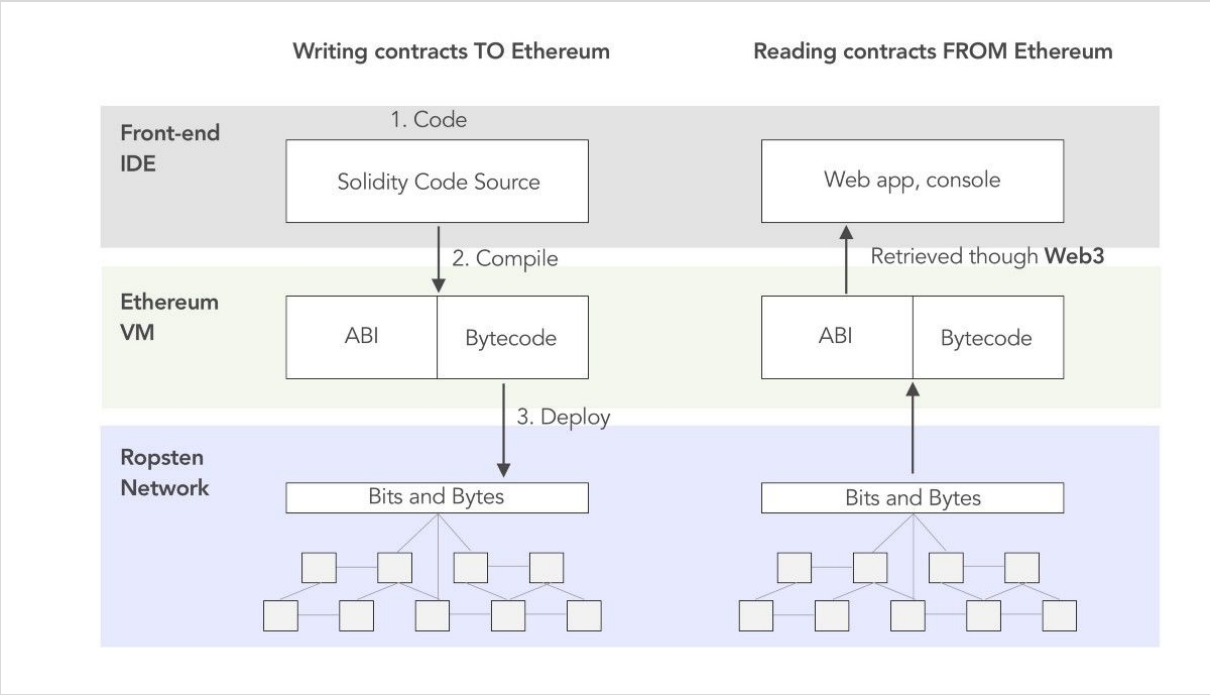
Lecture 2

Hands on Coding With Solidity

Contract ABI and Bytecode

Lecture 2

Hands on Coding With Solidity


Writing contracts TO Ethereum
Reading contracts FROM Ethereum
Front-end
IDE
1. Code
Solidity Code Source
Web app, console
2. Compile
Retrieved though Web3
Ethereum
VM
ABI
Bytecode
ABI
Bytecode
3. Deploy
Ropsten
Network
Bits and Bytes
Bits and Bytes

Time for Assignment

Lecture 2

Hands on Coding With Solidity

Preparation for next class

Basic Frontend Development(HTML, CSS)

(React would be preferable)

Saturday and Sunday for Smart Contract Development Practice

Thank you!

Lecture 2

Hands on Coding With Solidity