

|  |
| --- |
| Solid Hands-on in Solidity |
| Solidity Lab  By  Sriram |

Solid Hands-on in Solidity

pragma solidity ^0.4.0**;**

//Source files can (and should) be annotated with a so-called version pragma to reject being compiled with future compiler versions that might introduce incompatible change

import “filename”**;**

//This statement imports all global symbols from “filename” (and symbols imported there) into the

current global scope

//import \* as symbolName from “filename”;

//Creates a new global symbol symbolName whose members are all the global symbols from “filename”

## General Value Types

Bool: The possible values are constants i.e., true and false int / uint: Signed and unsigned integers of various sizes

Keywords uint8 to uint256 in steps of 8 (unsigned of 8 up to 256 bits) and int8 to int256. uint and int are aliases for uint256 and int256, respectively

## Address

pragma solidity **^0.4.0;**

address x= 0x123**;**

address myAddress = this;

if (x.balance< 10 && myAddress.balance >=10) x.transfer(10);

**Arrays**

**Allocating Memory Arrays**

pragma solidity ^0.4.0; contract C {

function f(uint len) {

uint[] memory a = new uint[](7); bytes memory b = new bytes(len);

// Here we have a.length == 7 and b.length == len

a[6] = 8;

}

}

1 Page 1

Solid Hands-on in Solidity



## Assigning Storage Variable to Memory Arrays

pragma solidity ^0.4.0;

contract C {

uint[] x; // the data location of x is storage

// the data location of memoryArray is memory function f(uint[] memoryArray) {

x = memoryArray; // works, copies the whole array to storage

var y = x; // works, assigns a pointer, data location of y is storage y[7]; // fine, returns the 8th element

y.length = 2; // fine, modifies x through y

delete x; // fine, clears the array, also modifies y

// The following does not work; it would need to create a new temporary /

// unnamed array in storage, but storage is "statically" allocated:

// y = memoryArray;

// This does not work either, since it would "reset" the pointer, but

there

}

// is no sensible location it could point to.

// delete y;

g(x); // calls g, handing over a reference to x

h(x); // calls h and creates an independent, temporary copy in memory

function g(uint[] storage storageArray) internal {} function h(uint[] memoryArray) {}

}

## Array Literals/ Inline Arrays

pragma solidity ^0.4.0; contract C {

function f() { g([uint(1), 2, 3]);

}

function g(uint[3] \_data) {

// ...

} }

2

Solid Hands-on in Solidity



## Mappings

Mapping( \_Keytype => \_ValueType )

pragma solidity ^0.4.0; contract MappingExample {

mapping(address => uint) public balances; function update(uint newBalance) {

balances[msg.sender] = newBalance;

}

}

contract MappingUser {

function f() returns (uint) {

MappingExample m = new MappingExample(); m.update(100);

return m.balances(this);

}

}

## Enums



pragma solidity ^0.4.0; contract test {

enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }

ActionChoices choice;

ActionChoices constant defaultChoice = ActionChoices.GoStraight; function setGoStraight() {

choice = ActionChoices.GoStraight;

}

// Since enum types are not part of the ABI, the signature of "getChoice"

// will automatically be changed to "getChoice() returns (uint8)"

// for all matters external to Solidity.

The integer type used is just

// large enough to hold all enum values, i.e. if you have more

values,

// `uint16` will be used and so on.

function getChoice() returns (ActionChoices) { return choice;

}

function getDefaultChoice() returns (uint) { return uint(defaultChoice);

}

}

3

Solid Hands-on in Solidity



## Structs

pragma solidity ^0.4.0; contract Ballot {

struct Voter { // Struct

uint weight1, weight2, weight3; bool voted;

address delegate1, delegate2, delegate3, delegate4; uint vote1, vote2, vote3, vote4, vote5;

uint height1, height2, height3 //structs can only have 16 members, exceeding which the following error might occur [stack too deep]

} }

**Operators in Solidity**

pragma solidity ^0.4.0**;** contract operators {

// Arithmetic Operators

// +,-,\*,/, %, \*\*

// Incremental Operators

// a++, a--, a+=1, a=a+1,++a,--a; a=10;

a= a++; //here, output will be 10, because the value is first

returned and then then increment is done

a=++a;

//Logical Operators

!, &&, ||, ==, !=

isOwner = true && false;

var orValue= 0x02 | 0x01; // output would be 0x03

//Bitwise Operators~,>>, <<; function Operators() {

// Initialize state variables here}}

## Input Parameters

pragma solidity ^0.4.0**;**

contract Simple {

function taker(uint \_a, uint \_b){

// do something with a and b

} }

## Output Parameters

pragma solidity ^0.4.0**;** contract Simple {

function arithmetics(uint \_a, uint \_b) returns (uint o\_sum, uint 0\_product){

O\_sum = \_a + \_b;

O\_product = \_a\* \_b;

}

4

Solid Hands-on in Solidity



## Function calls & return types



pragma solidity ^0.4.0**;** contract FunctionCall {

// Constructor calls are also a function calls and are defined like this function FunctionCall(uint param1) {

// Initialize state variables here

}

// you can create a contract object with a name & then use it inside the function calls like this

Miner m;

function setMiner(address addr) {

m = Miner(addr); // type casted the addr to Miner type and stored in m

}//function setMiner(Miner \_m) { m = \_m; } is also correct

// Now you can use the Miner's function which is info to sent

// some ether with optionally specifying the gas like this function callMinerInfo() {

1. info.value(10). gas (800) ();

}

//function can also be called as json object as parameters

// below function can be called by using the json object as shown in demo function below

function someFunction(uint key, uint value) {

// Do something}

function demoFunction() {

// named arguments someFunction({value: 2, key: 3}); }

//also note that variable names are optional in parameters & in returns function someFunction2(uint key, uint) returns (uint) {

return key; // Do something}} contract Miner {

//The modifier payable has to be used for info,

// because otherwise, we would not be able to

//send Ether to it in the call m.info.value(10). gas (800) () function info () payable returns (uint ret) {

return 42;}}

## Function Modifier

pragma solidity ^0.4.0**;** contract FunctionModifiers{

address public creator; function FunctionModifiers(){

creator= msg.sender; } Modifier onlyCreator(){ if(msg.sender!=creator){

throw;

}

\_; // resumes the function execution wherever the access modifier is used} function killContract() onlyCreator{ //this function will not execute if an exception occurs

self-destruct(creator); }}

5

Solid Hands-on in Solidity



## Fallback Function

pragma solidity ^0.4.0; contract Test {

// This function is called for all messages sent to

// this contract (there is no other function).

// Sending Ether to this contract will cause an exception,

// because the fallback function does not have the "payable"

// modifier.

function () {x = 1; } uint x;

}

// This contract keeps all Ether sent to it with no way

// to get it back. contract Sink {

function () payable {}

}

contract Caller {

function callTest(Test test) { test.call(0xabcdef01); // hash does not exist

// results in test.x becoming == 1.

// The following will not compile, but even

// if someone sends ether to that contract,

// the transaction will fail and reject the

// Ether.

//test.send(2 ether);

}

## Inheritance

pragma solidity ^0.4.0**;** contract owned {

address owner; function owned () {

owner = msg.sender;

}}

contract mortal is owned{ //’is’ keyword is used for Inheritance

function kill () {

self-destruct(owner);}}

contract User is owned, mortal //Multiple Inheritance

{

String public UserName; function User (String \_name) { UserName = \_name;

}

}

6

Solid Hands-on in Solidity



## Abstract Contracts

Contract functions can lack an implementation as in the following example**:**

pragma solidity **^0.4.0;**

contract Feline {

function **utterance() returns (bytes32)**;

}

Such contracts cannot be compiled, but they can be used as base contracts:

pragma solidity **^0.4.0;**

contract Feline {

function **utterance () returns (bytes32)**;

}

contract Cat is feline {

function utterance() returns (bytes32) { return

“miaow;

}

## Interfaces

pragma solidity **^0.4.0;**

interface Token {

function transfer (address recipient, uint amount);

}

**Events**

pragma solidity ^0.4.0; contract SimpleAuction {

event HighestBidIncreased(address bidder, uint amount); // Event function bid() payable {

// ...

HighestBidIncreased(msg.sender, msg.value); // Triggering

event

} }

7

Solid Hands-on in Solidity



## Events Example:



pragma solidity ^0.4.0; contract ClientReceipt {

event Deposit (

address indexed \_from, bytes32 indexed \_id, uint \_value

);

function deposit (bytes32 \_id) payable {

// Any call to this function (even deeply nested) can

// be detected from the JavaScript API by filtering

// for `Deposit` to be called.

Deposit (msg.sender, \_id, msg.value);

}

}

pragma solidity ^0.4.0;

var abi = /\* abi as generated by the compiler \*/; var ClientReceipt = web3.eth.contract(abi);

var clientReceipt = ClientReceipt.at (0x123 /\* address \*/); var event = clientReceipt.Deposit();

// watch for changes

event.watch(function(error, result){

// result will contain various information

// including the argumets given to the Deposit call. if (! error)

console.log(result);

});

// Or pass a callback to start watching immediately

var event = clientReceipt.Deposit(function (error, result) { if (! error)

console.log(result);});

8

Solid Hands-on in Solidity



## Creating Contracts via new

pragma solidity **^0.4.0;**

contract D {

uint x;

function D (uint a) payable {

x=a;

} }

contract C {

D d= new D (4); // will be executed as part of C’s constructor

function created (uint arg) { D newD =new D(arg);

}

function createAndEndowD(uint arg, uint amount){ D newD= (new D).value(amount)(arg);}}

**Libraries**



pragma solidity ^0.4.0**;** library Set {

struct Data {mapping (uint => bool) flags;} function insert (Data storage self, uint value)

returns (bool)

{

if (self.flags[value]) return false;

self.flags[value]= true; return true;}

function remove (Data storage self, uint value) returns (bool)

{

if (! self.flags[value])

return false; // not there self.flags[value] = false;

return true;

}

function contains (Data storage self, uint value) returns (bool)

{

return self.flags[value]; }} contract C {

set.Data knownValues;

function register (uint value) {

// The library functions can be called without a

// specific instance of the library, since the

// "instance" will be the current contract. require (Set.insert(knownValues, value));}

// In this contract, we can also directly access knownValues.flags, if

we want

}

9

Solid Hands-on in Solidity

# Creating Smart Contracts using Ethereum Remix Browser

Follow the steps to create Smart Contract on Ethereum: Step 1: Creating “**testrpc.sh**” file.

**Example:** cat>testrpc.sh

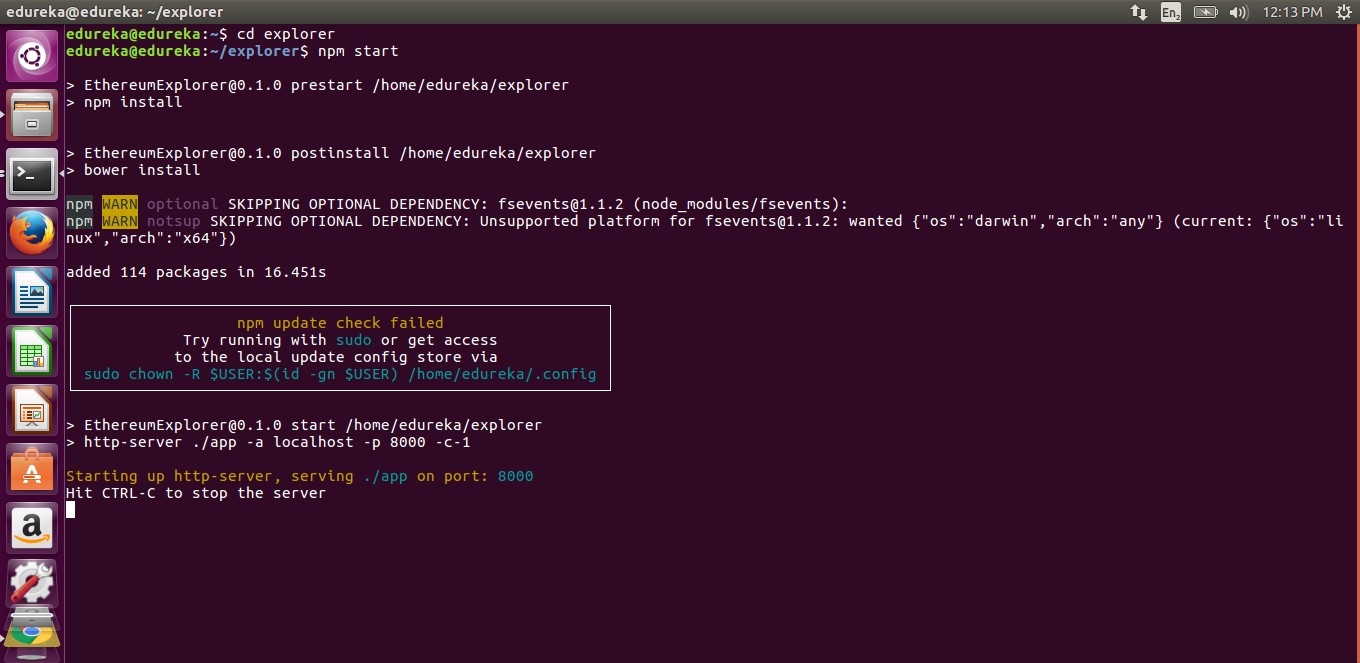
Step 2: Execute” testrpc” file.

**Execute:** ./testrpc.sh

Step 3: Change the Directory to “explorer”

**Execute:** cd explorer Step 4: Start Block Explorer

**Execute:** npm start



This command will launch the block explorer



This command will install all the packages and will launch the block explorer at port 8000 Step 5: View Block Explorer

**Type:** localhost:8000 in the browser tab and hit enter

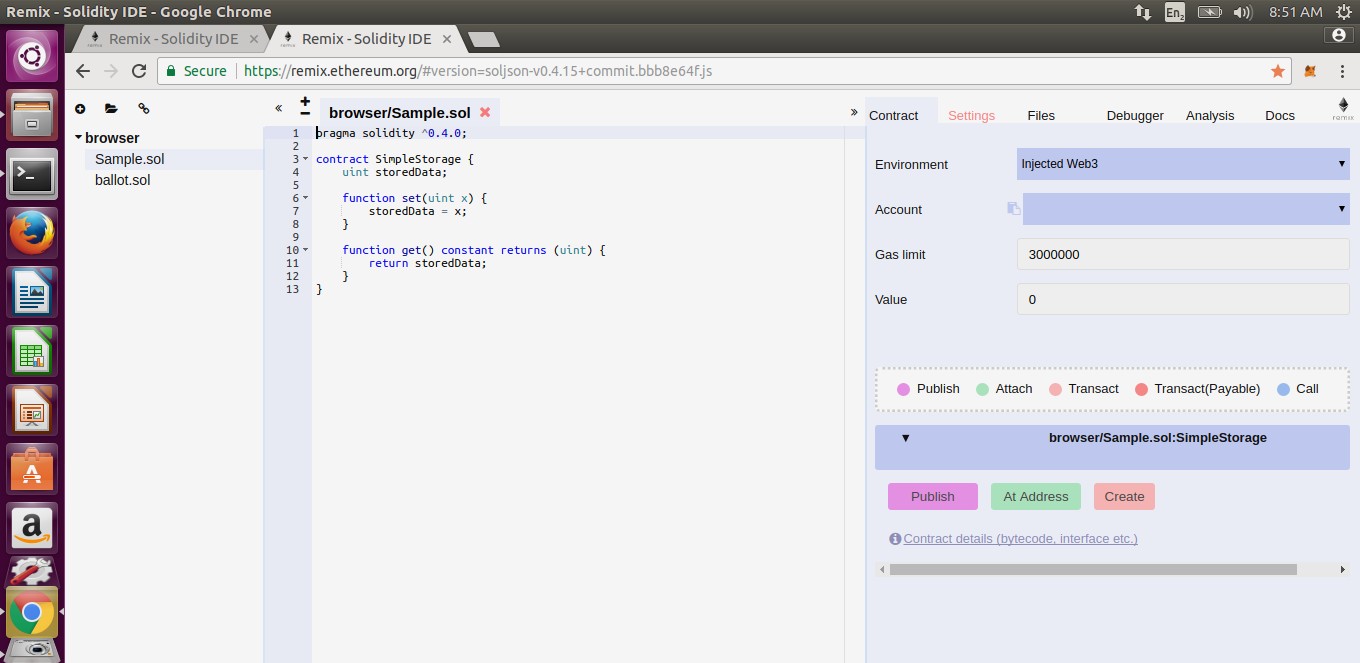
10

Solid Hands-on in Solidity



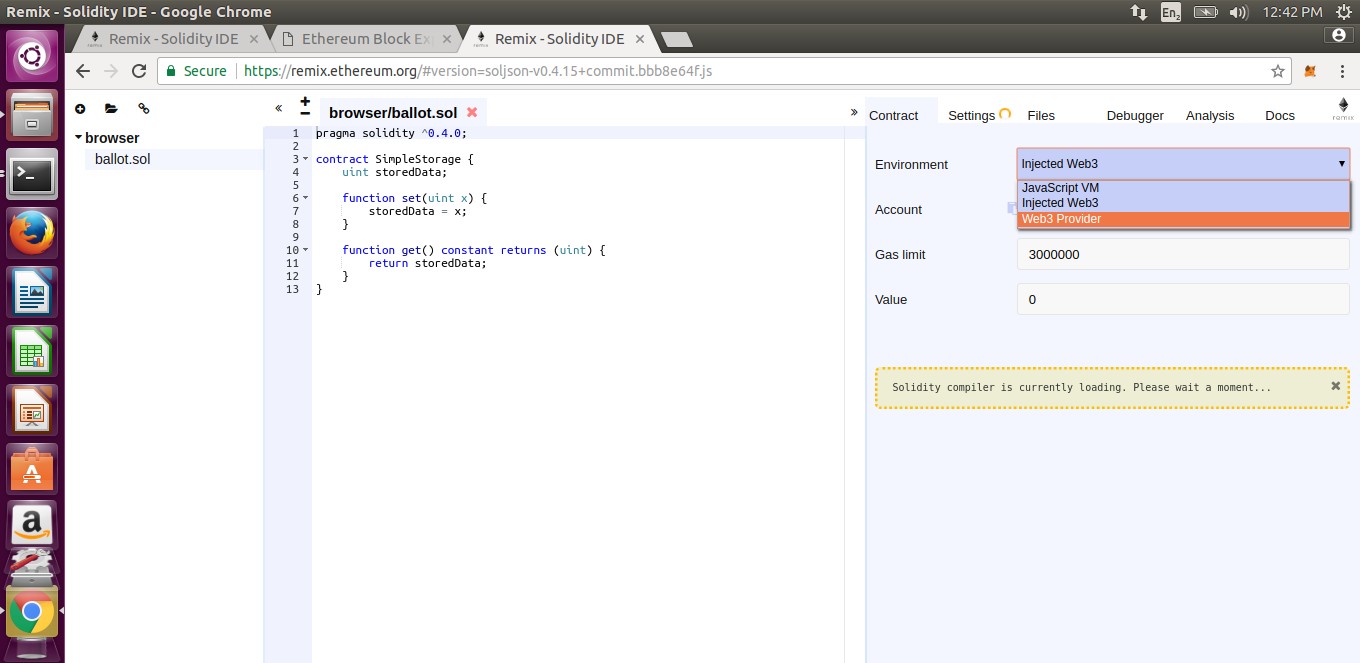
Step 6: Open Remix browser

* + Open a new tab in your browser and search “Remix Ethereum”
  + Click on the first link that pops up



Step 7: Select the Environment

1. Click on the drop-down icon near in front of the “**Environment**”
   * + Three options will be seen
     + Select “**web3 provider**” option from the list

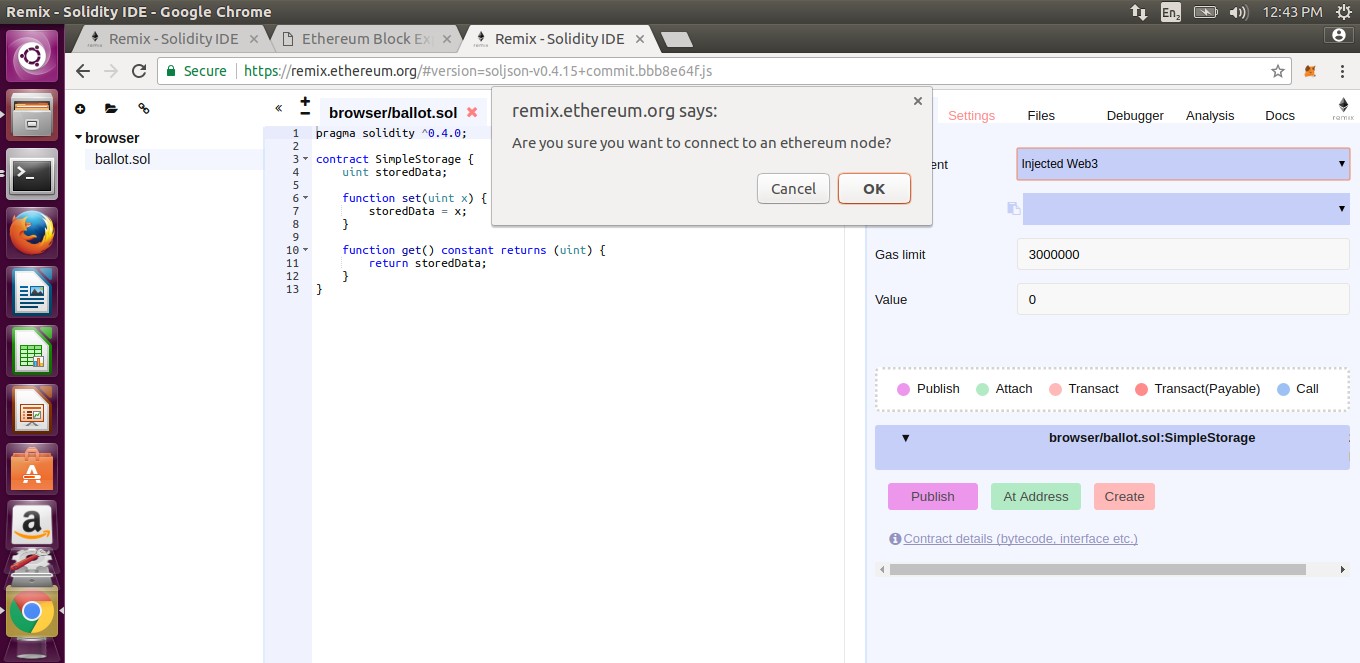


11

Solid Hands-on in Solidity

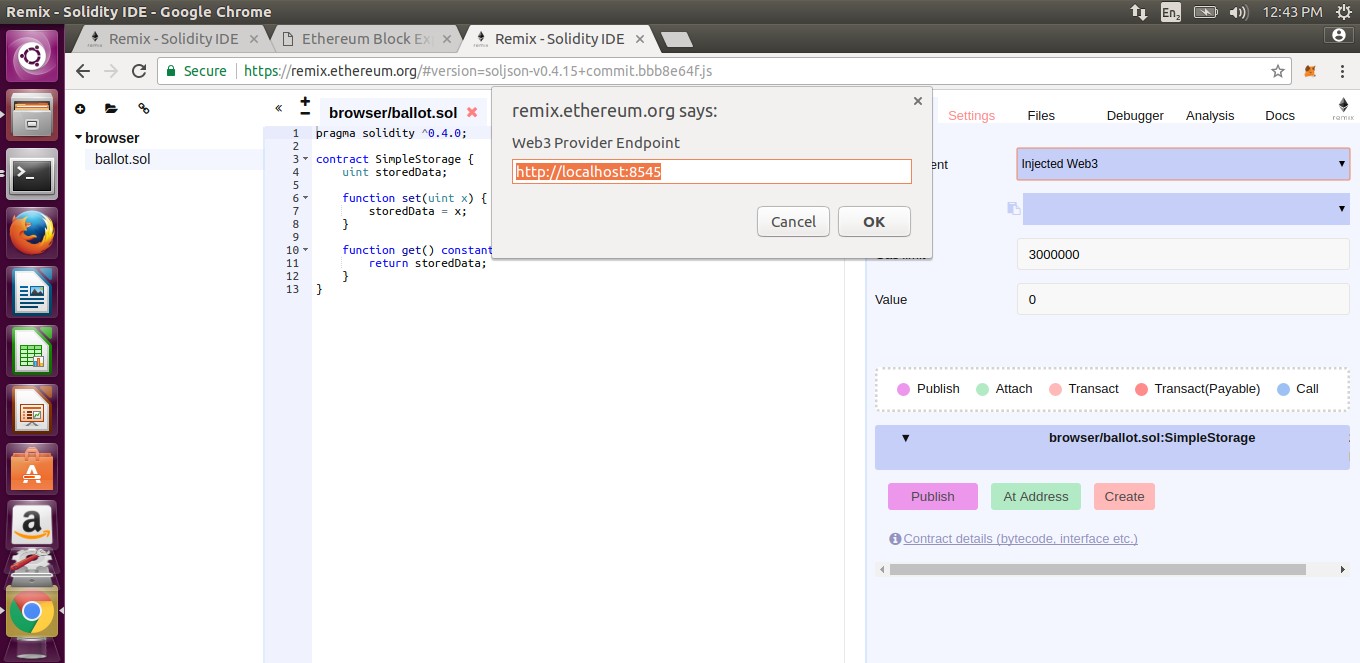


Step 8: Confirm the selected option Click ok on the pop up box



Step 9: Selecting Web3 Provider Endpoint

## Next, It will ask for the web3 Provider Endpoint, now this can be any address, like the Ethereum Blockchain server, or the address of the different machine you wish to connect, or it can be the localhost



So here, just click ok button on the pop.

12

Solid Hands-on in Solidity

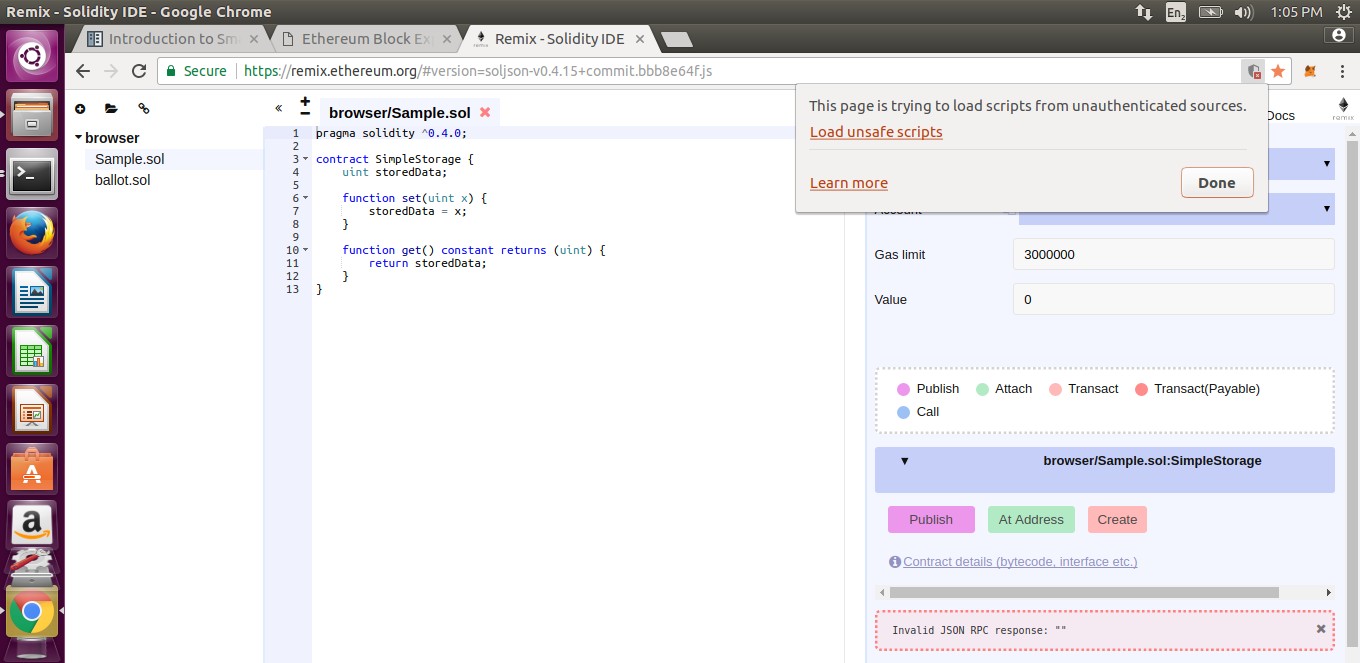


Step 10: The Browser Restricts the Connection. On clicking the ok button (in previous slide) , the window shown below will pop up, connection to the local host is restricted by the browser

Step 11: Load Unsafe Script.

As, we have seen, the browser is restricting the connection, it is because of the unsafe script. Select the **“load unsafe scripts option “**

This will now connect you to the local host port 8545



Step 12: Again, we need to set the “Environment”

Click on the load unsafe script to resolve the issue

## The connection with port is established as you can see the account balances in the window shown

**below, however we need to setup the “environment” again**



Click on the box and select the

“web3 provider” option

13

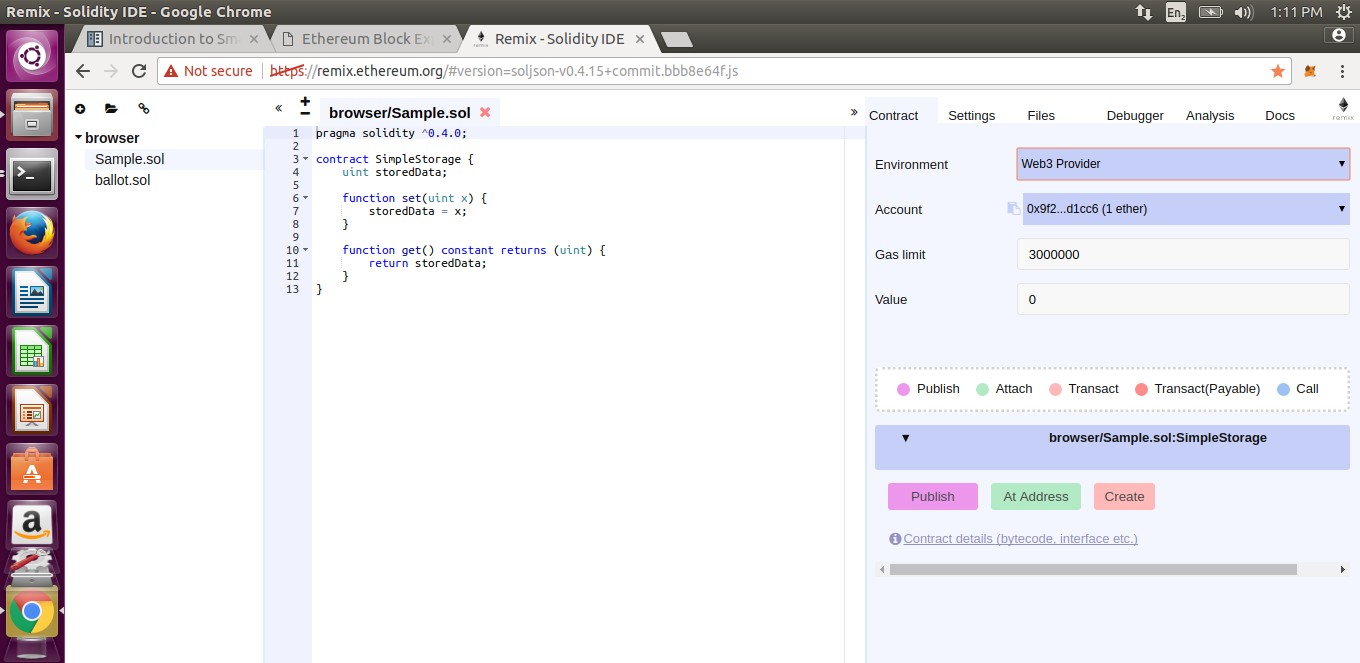
Solid Hands-on in Solidity



## When you select the “load unsafe script” (previously),The page reloads and the environment gets changed. Now to change it to “web3 provider”, just click on the dropdown box and select the “web3 provider” option as done previously.

Step 13: The connection is now successful.

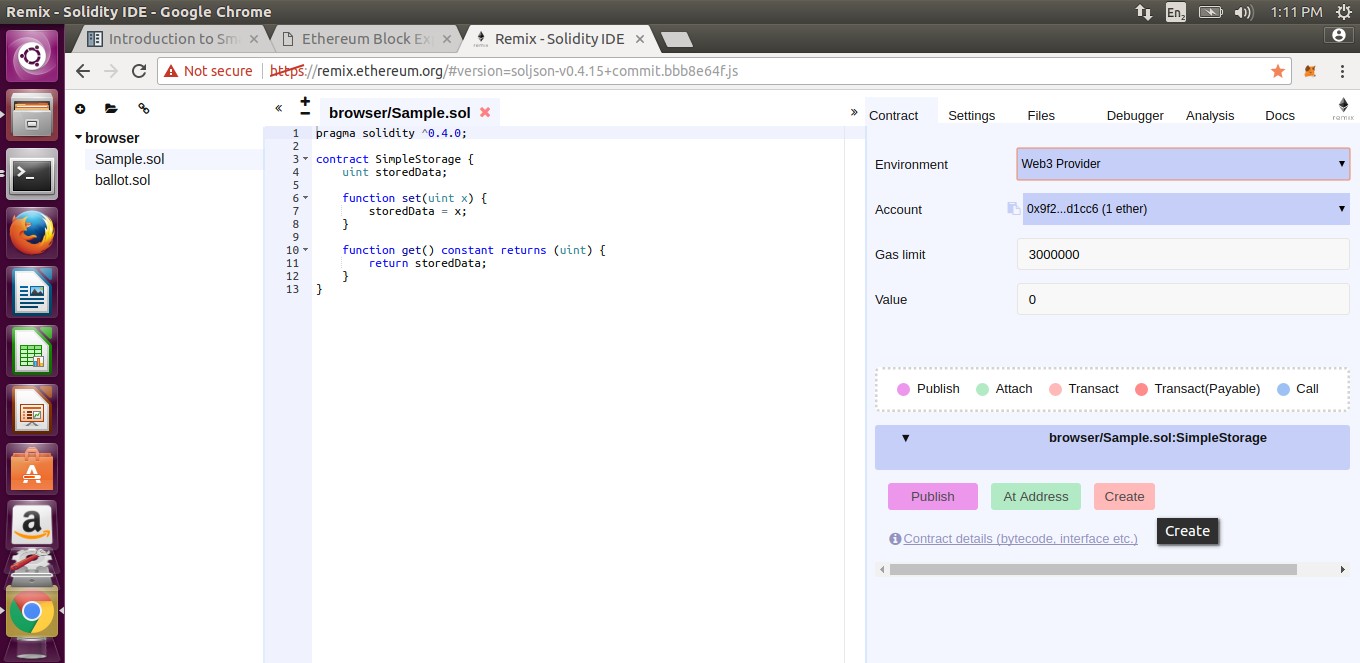
The browser indicates “Not secure” connection, however it connects to the local host port



Step 14: Now that we are connected to the port, we can create our very first smart contract

## To create contract:

* + - Select the account that you want to use to push your contract
    - Write a contract and click on create button



Click on the dropbox to check the

account balances

Click here to create contract

14

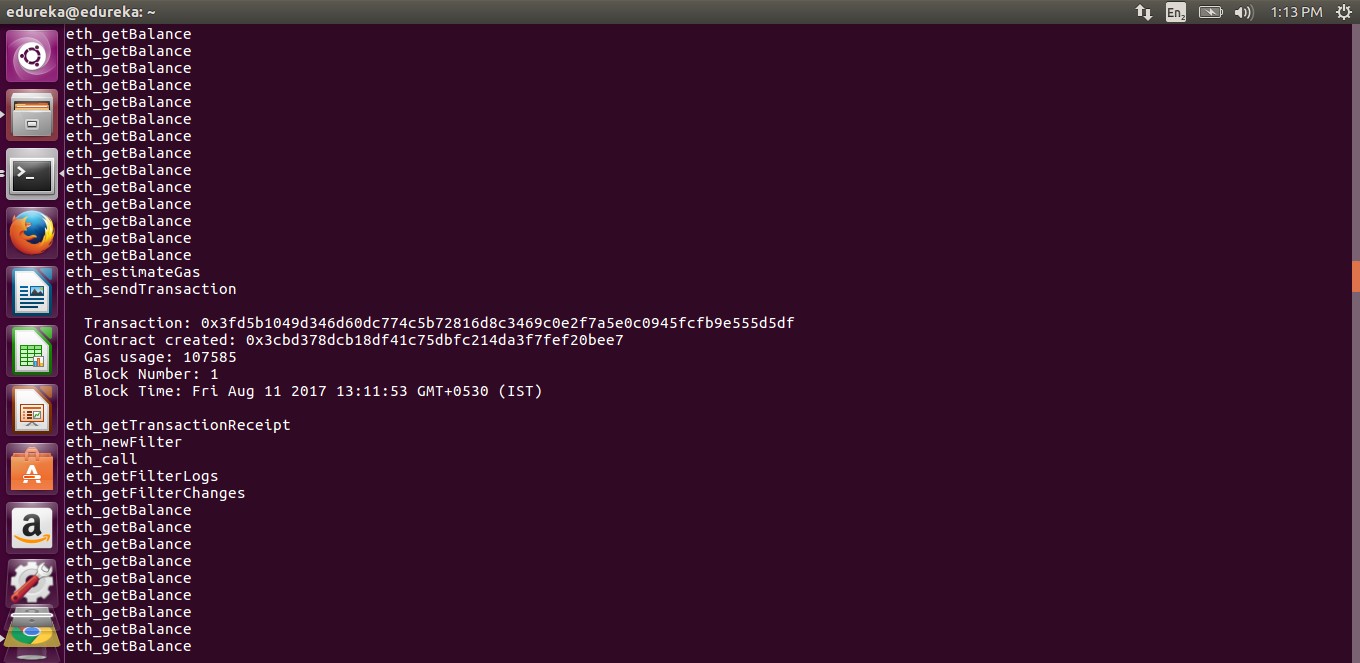
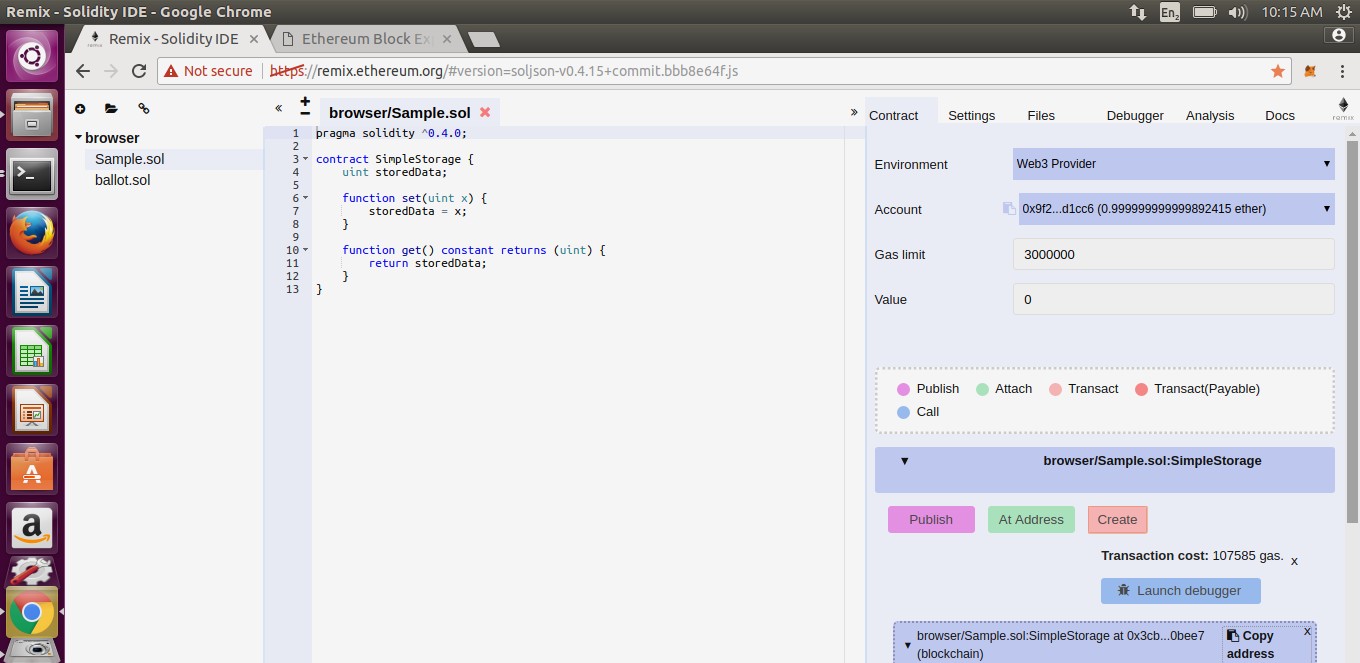
Solid Hands-on in Solidity



Step 15: The contract gets created.

## The contract gets created and the balance of the account used to create the contract goes down by some value.

**It is the value of ether which get exhausted as a gas for pushing contract.**



Step 16: Check the Block that Includes your Transaction.

Transaction cost as

“gas used”

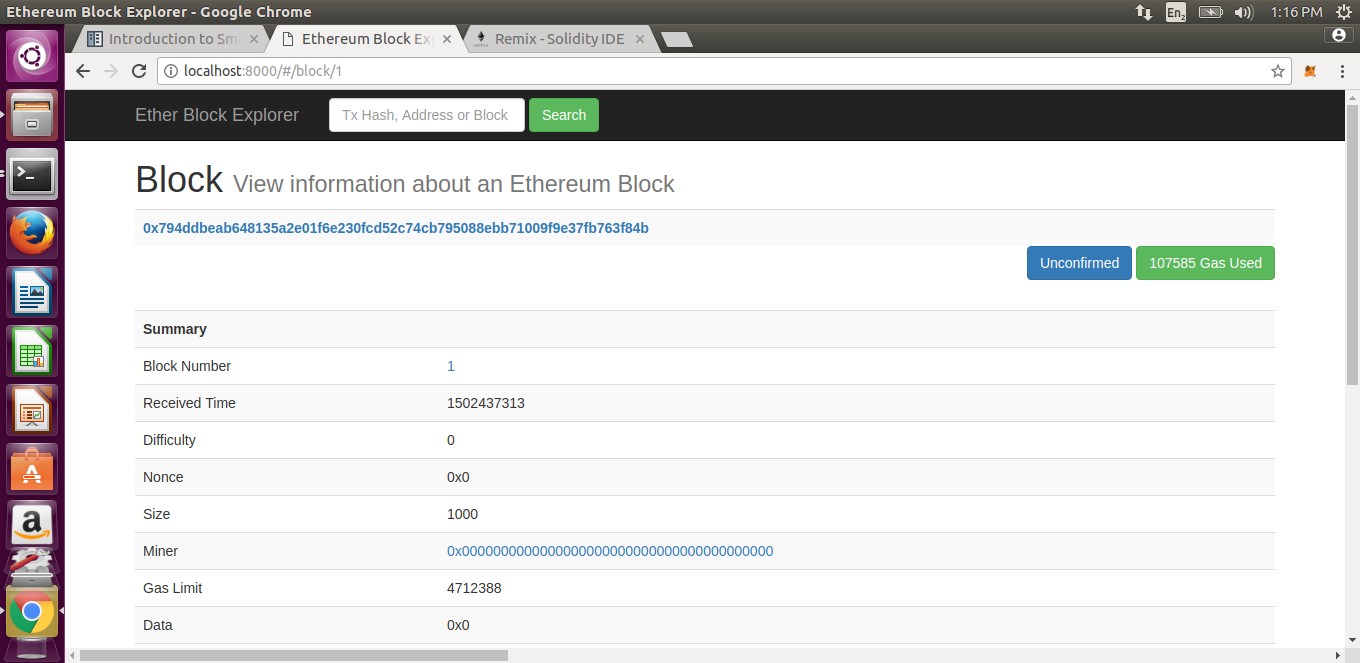
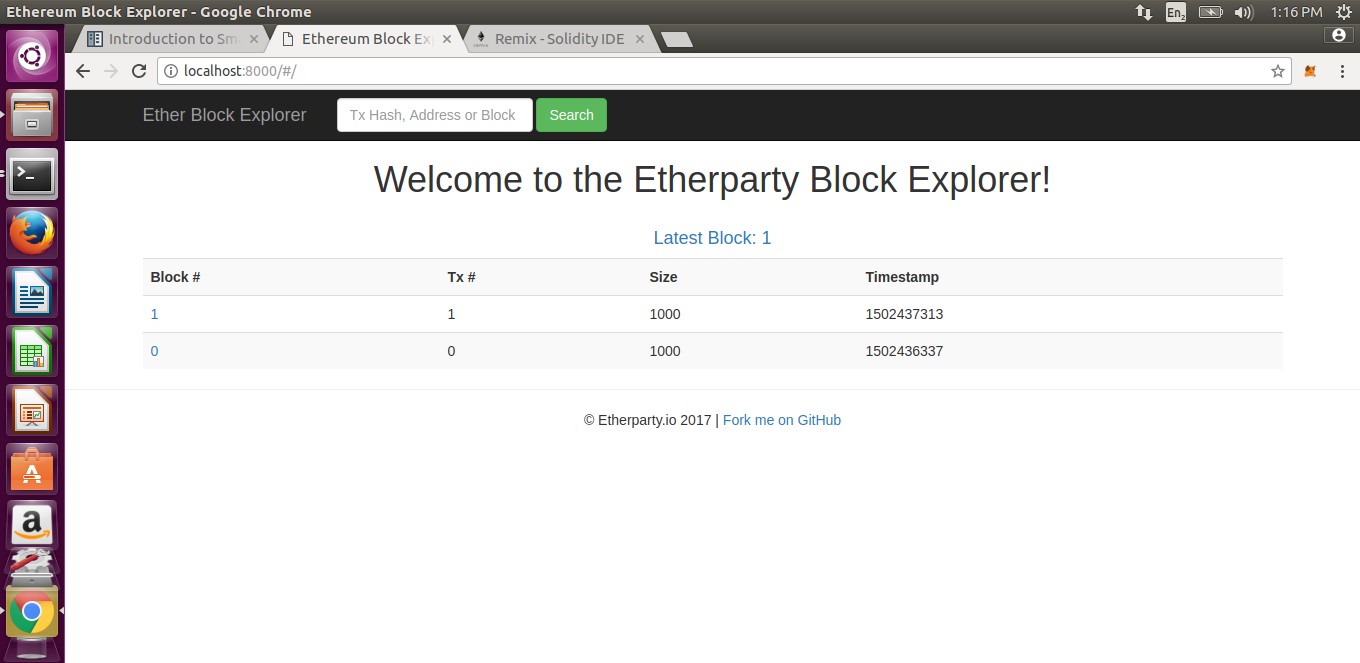
15

Solid Hands-on in Solidity



Step 17: Transaction Reflecting in Block Explorer.

The transaction gets included in Block Explorer, and can be seen after you refresh the page.



Step 18: Block Details in Block Explorer.

The block contains all the details of the transaction.

Account address

16

Solid Hands-on in Solidity



# Following are the smart contract you can try on remix browser:

**1.Blind Auction:**

pragma solidity ^0.4.11; contract BlindAuction {

struct Bid {

bytes32 blindedBid; uint deposit;

}

address public beneficiary; uint public biddingEnd; uint public revealEnd;

bool public ended;

mapping(address => Bid[]) public bids; address public highestBidder;

uint public highestBid;

// Allowed withdrawals of previous bids mapping(address => uint) pendingReturns;

event AuctionEnded(address winner, uint highestBid);

/// Modifiers are a convenient way to validate inputs to

/// functions. `onlyBefore` is applied to `bid` below:

/// The new function body is the modifier's body where

/// `\_` is replaced by the old function body.

modifier onlyBefore(uint \_time) { require(now < \_time); \_; } modifier onlyAfter(uint \_time) { require(now > \_time); \_; }

function BlindAuction( uint \_biddingTime, uint \_revealTime, address \_beneficiary

) {

beneficiary = \_beneficiary; biddingEnd = now + \_biddingTime; revealEnd = biddingEnd + \_revealTime;

}

/// Place a blinded bid with `\_blindedBid` = keccak256(value,

/// fake, secret).

/// The sent ether is only refunded if the bid is correctly

/// revealed in the revealing phase. The bid is valid if the

/// ether sent together with the bid is at least "value" and

/// "fake" is not true. Setting "fake" to true and sending

/// not the exact amount are ways to hide the real bid but

/// still make the required deposit. The same address can

/// place multiple bids.

function bid(bytes32 \_blindedBid)

17

Solid Hands-on in Solidity



payable onlyBefore(biddingEnd)

{

bids[msg.sender].push(Bid({ blindedBid: \_blindedBid, deposit: msg.value

}));

}

/// Reveal your blinded bids. You will get a refund for all

/// correctly blinded invalid bids and for all bids except for

/// the totally highest.

function reveal( uint[] \_values, bool[] \_fake, bytes32[] \_secret

)

onlyAfter(biddingEnd) onlyBefore(revealEnd)

{

uint length = bids[msg.sender].length; require(\_values.length == length); require(\_fake.length == length); require(\_secret.length == length);

uint refund;

for (uint i = 0; i < length; i++) { var bid = bids[msg.sender][i]; var (value, fake, secret) =

(\_values[i], \_fake[i], \_secret[i]);

if (bid.blindedBid != keccak256(value, fake, secret)) {

// Bid was not actually revealed.

// Do not refund deposit. continue;

}

refund += bid.deposit;

if (!fake && bid.deposit >= value) { if (placeBid(msg.sender, value))

refund -= value;

}

// Make it impossible for the sender to re-claim

// the same deposit. bid.blindedBid = bytes32(0);

}

msg.sender.transfer(refund);

}

// This is an "internal" function which means that it

// can only be called from the contract itself (or from

// derived contracts).

function placeBid(address bidder, uint value) internal

returns (bool success)

{

if (value <= highestBid) { return false;

18

Solid Hands-on in Solidity

}

if (highestBidder != 0) {

// Refund the previously highest bidder. pendingReturns[highestBidder] += highestBid;

}

highestBid = value; highestBidder = bidder; return true;

}

// Withdraw a bid that was overbid. function withdraw() {

uint amount = pendingReturns[msg.sender];

if (amount > 0) {

// It is important to set this to zero because the

recipient call

}

// can call this function again as part of the receiving

// before `send` returns (see the remark above about

// conditions -> effects -> interaction). pendingReturns[msg.sender] = 0;

msg.sender.transfer(amount);

}



19

/// End the auction and send the highest bid

/// to the beneficiary. function auctionEnd()

onlyAfter(revealEnd)

{

require(!ended); AuctionEnded(highestBidder, highestBid); ended = true;

// We send all the money we have, because some

// of the refunds might have failed. beneficiary.transfer(this.balance);

}

}

Solid Hands-on in Solidity



**2.Safe Remote Purchase**

pragma solidity ^0.4.11; contract Purchase {

uint public value;

address public seller; address public buyer;

enum State { Created, Locked, Inactive } State public state;

// Ensure that `msg.value` is an even number.

// Division will truncate if it is an odd number.

// Check via multiplication that it wasn't an odd number. function Purchase() payable {

seller = msg.sender; value = msg.value / 2;

require((2 \* value) == msg.value);

}

modifier condition(bool \_condition) { require(\_condition);

\_;

}

modifier onlyBuyer() { require(msg.sender == buyer);

\_;

}

modifier onlySeller() { require(msg.sender == seller);

\_;

}

modifier inState(State \_state) { require(state == \_state);

\_;

}

event Aborted();

event PurchaseConfirmed(); event ItemReceived();

/// Abort the purchase and reclaim the ether.

/// Can only be called by the seller before

/// the contract is locked. function abort()

onlySeller inState(State.Created)

{

Aborted();

20

Solid Hands-on in Solidity



state = State.Inactive; seller.transfer(this.balance);

}

/// Confirm the purchase as buyer.

/// Transaction has to include `2 \* value` ether.

/// The ether will be locked until confirmReceived

/// is called.

function confirmPurchase() inState(State.Created) condition(msg.value == (2 \* value)) payable

{

PurchaseConfirmed(); buyer = msg.sender; state = State.Locked;

}

/// Confirm that you (the buyer) received the item.

/// This will release the locked ether. function confirmReceived()

onlyBuyer inState(State.Locked)

{

ItemReceived();

// It is important to change the state first because

// otherwise, the contracts called using `send` below

// can call in again here. state = State.Inactive;

// NOTE: This actually allows both the buyer and the seller to

// block the refund - the withdraw pattern should be used.

buyer.transfer(value); seller.transfer(this.balance);

}

}

21