Workshop on
Blockchain Technologies and Applications

Smart Contracts with Ethereum (Hands-on)-I

Mohammed Sumair
M.Tech Scholar (CSE)
EECS Department
IIT Bhilai

Overview

- Smart Contracts
- Solidity
- Remix IDE
- Development of Smart Contracts

Smart Contracts

What are Smart Contracts?

- **Smart contracts** are computer programs that act as agreements where the terms of the agreement can be pre programmed with the ability to be executed and enforced.

Smart Contracts on Ethereum

A Next-Generation Smart Contract and Decentralized Application Platform

- ❑ Ethereum provides **Ethereum Virtual Machine (EVM)** on which smart contracts are executed.
- ❑ Provides a **Turing Complete Language**.
- ❑ **Infinite Loop problem and solution to it.**
- ❑ Every execution step has a cost associated in terms of **gas.**

Development of Smart Contracts

- ❑ **Solidity:** Programming Language for writing smart contracts.

- ❑ **Remix IDE** (Integrated Development Environment): A web application that can be used to write, debug, and deploy Ethereum Smart Contracts.

2?

Ethereum

Everything is reads

Smart contract

Framework

Introduction to Solidity

Solidity

- ➔ Programming language used for writing Smart Contracts.
- ➔ High-level
- ➔ Object-oriented
- ➔ Supports inheritance
- ➔ Libraries
- ➔ Complex user-defined types

Solidity: Types

- Statically typed language
- Value Types: Variable of these types are passed by value.
  - int/uint: 256-bit integer
  - bool: two-state value **true** or **false.**
- **address:** account identifier, similar to a 160-bit hash type.
- string/bytes
- **No floats**

Solidity: Variables

- **State variables:** Permanently stored in contract storage.
- **Local variables:** Till the function is executing.
- **Global variables:** Used to get information about the Blockchain
  - **now (uint):** Current block timestamp
  - **msg.value (uint):** Number of wei sent with the message
  - **msg.sender (address payable):** Sender of the message

Solidity: Scope

- ➔ **Public:** can be accessed internally or via messages.
- ➔ **Internal:** can be accessed internally or by deriving contracts.
- ➔ **Private:** can be accessed only internally.

Solidity: Operators

- Arithmetic Operators: +, -, *, /, %.
- Comparison Operators: ==, >, <, >=, <=.
- Logical Operators: &&, ||, !.
- Bitwise Operators: &, |, ^, <<, >>.
- Assignment Operators: =, +=, -=.
- Conditional Operator: ? : .

Solidity: Loops & Decision making

- For Loop
- While Loop
- Do..while Loop
- If statement
- If..else statement
- If..else if.. statement

Solidity: Array & Mapping

Array: Collection of variables of the same type.

```
        type[arraysize] arrayname;
    type[ ] arrayname;
```

- Length,   arrayname.length
- Push,   arrayname.push(......)

Mapping: mapping(keytype=>valuetype)

```
        mapping(address=>uint) public records;
```

Solidity: Structs

Structs: Used to represent a record.

```
struct structname{
type typename;
……………………
}
```

➢ Member access operator (.)

Solidity: Ether units

- 1 **wei**
- 1 gwei = 1×10^{9} wei
- 1 finney = 1×10^{15} wei
- 1 ether = 1×10^{18} wei

Solidity: Functions

- Constructor ✓
- Modifier ✓
- modifier onlyOwner {

```
        require(msg.sender == owner);
        _;
  }
```

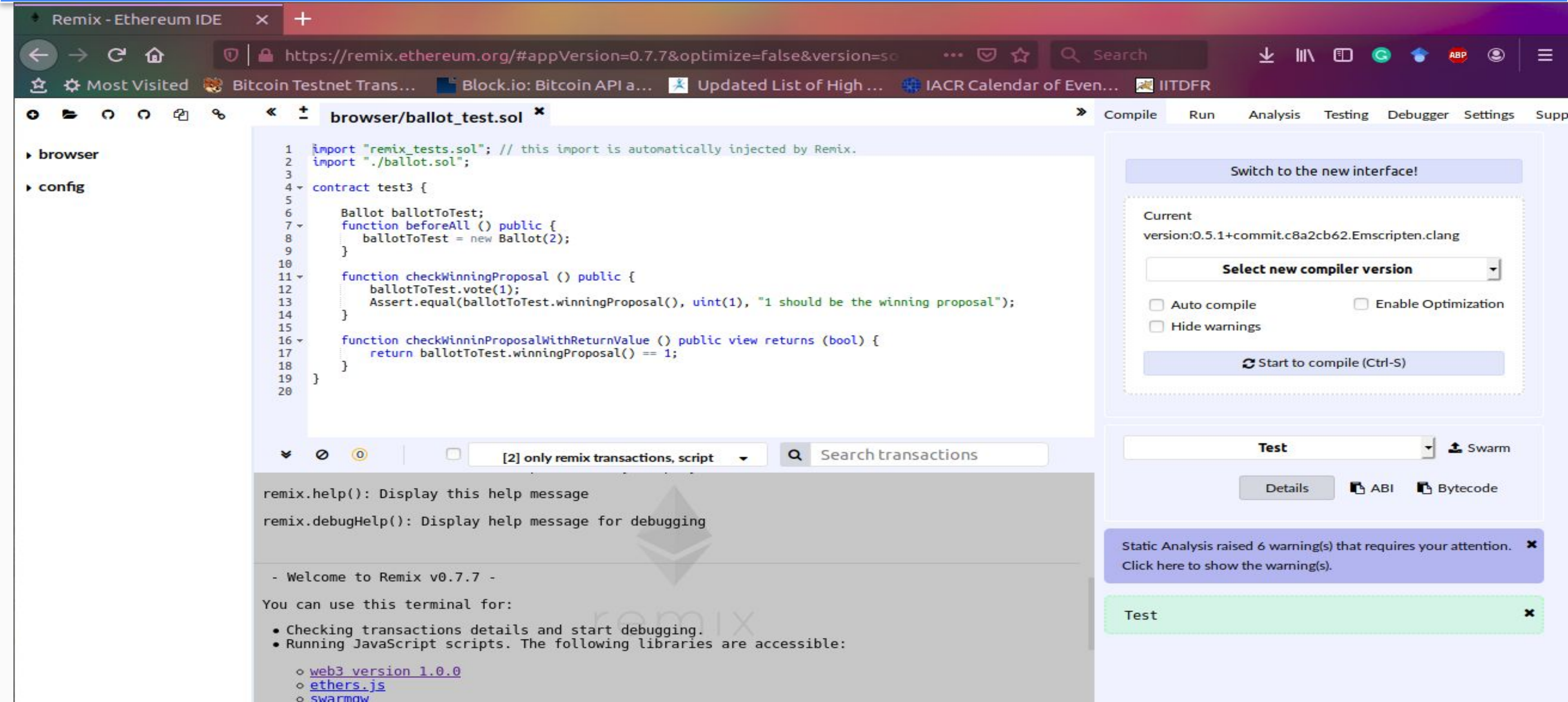
- **View** & **Pure** functions

Solidity

- ➔ Enums
- ➔ Inheritance
- ➔ Function Overloading
- ➔ Abstract Contracts
- ➔ Interfaces etc...

Introduction to Remix IDE

Remix IDE https://remix.ethereum.org



```
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "./ballot.sol";

contract test3 {

    Ballot ballotToTest;
    function beforeAll () public {
       ballotToTest = new Ballot(2);
    }

    function checkWinningProposal () public {
        ballotToTest.vote(1);
        Assert.equal(ballotToTest.winningProposal(), uint(1), "1 should be the winning proposal");
    }

    function checkWinninProposalWithReturnValue () public view returns (bool) {
        return ballotToTest.winningProposal() == 1;
    }
}
```

Current version:0.5.1+commit.c8a2cb62.Emscripten.clang

Select new compiler version

☐ Auto compile ☐ Enable Optimization

☐ Hide warnings

Start to compile (Ctrl-S)

Test | Swarm

Details | ABI | Bytecode

Static Analysis raised 6 warning(s) that requires your attention. Click here to show the warning(s).

Test

[2] only remix transactions, script

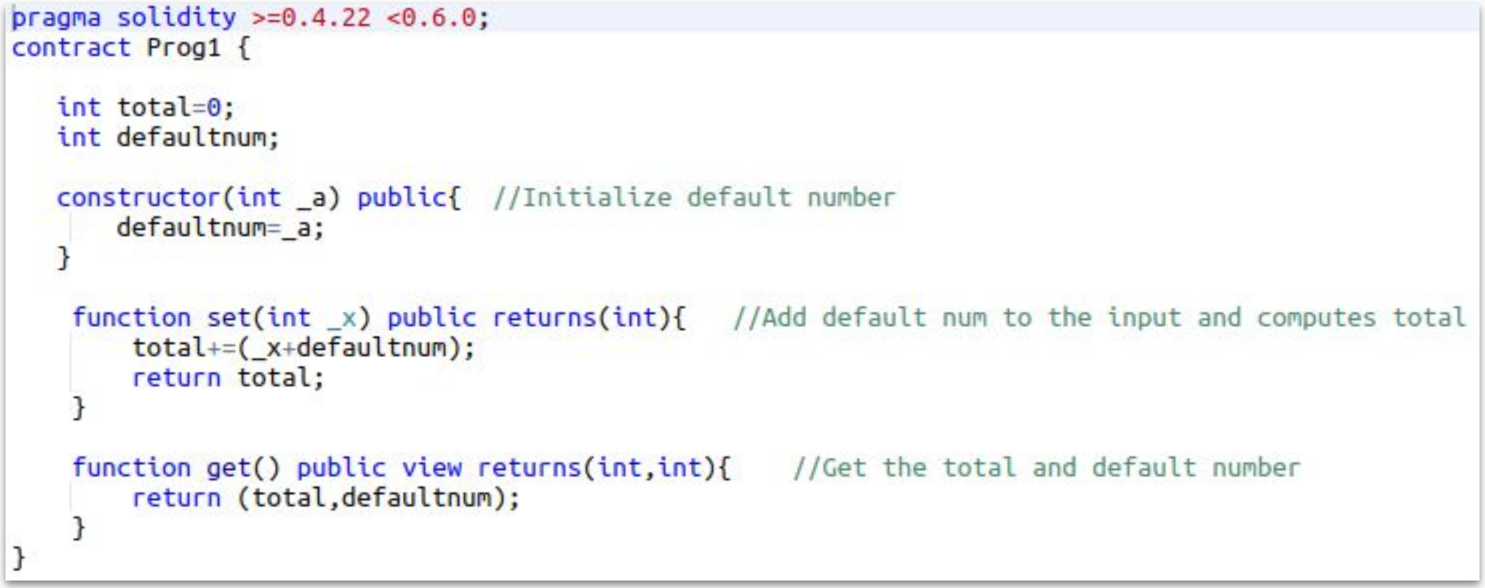
Search transactions

```
remix.help(): Display this help message

remix.debugHelp(): Display help message for debugging

 - Welcome to Remix v0.7.7 -

You can use this terminal for:

 • Checking transactions details and start debugging.
 • Running JavaScript scripts. The following libraries are accessible:

    ○ web3 version 1.0.0
    ○ ethers.js
    ○ swarmgw
```

Contract 1

Get & Set Contract

Solution

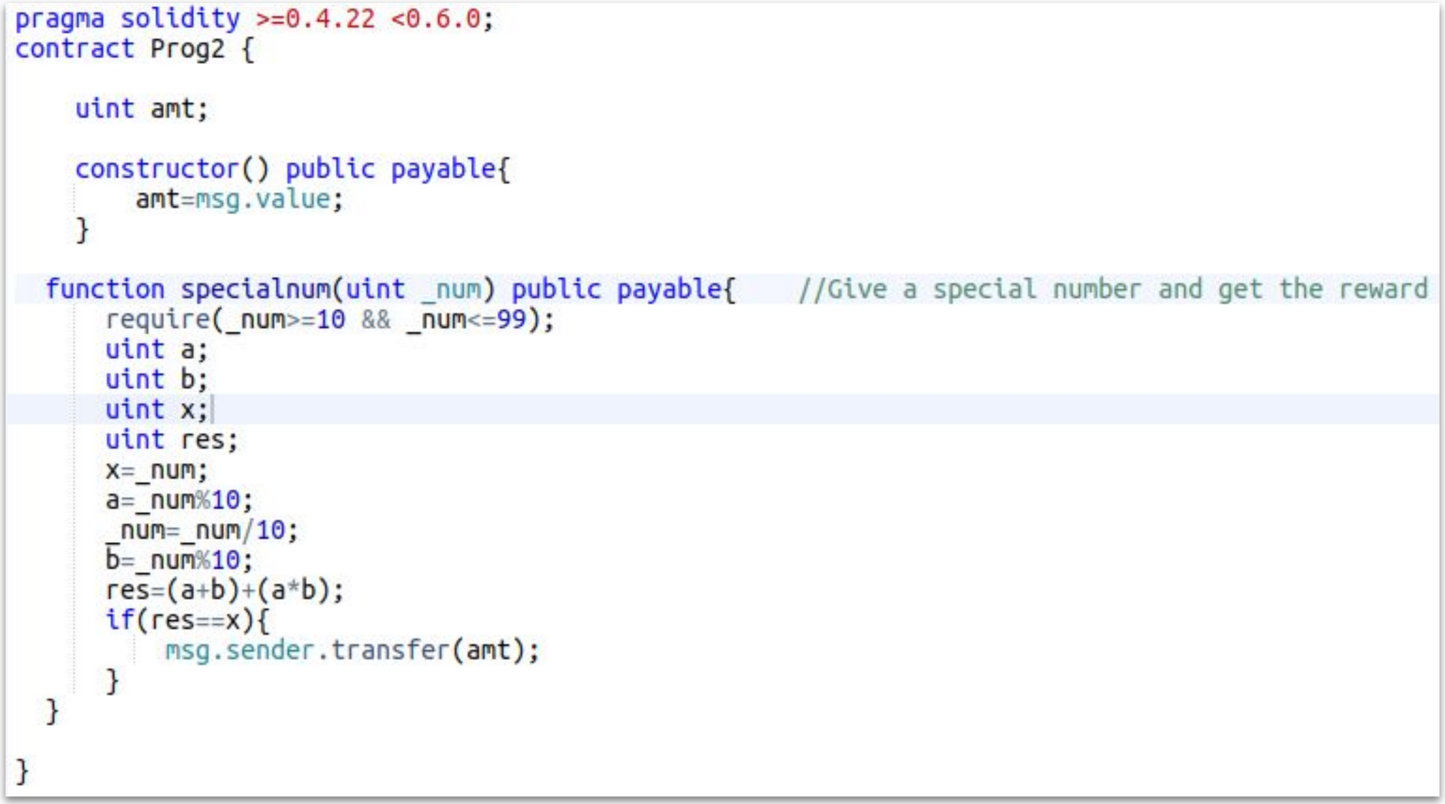


```
pragma solidity >=0.4.22 <0.6.0;
contract Prog1 {

   int total=0;
   int defaultnum;

   constructor(int _a) public{  //Initialize default number
       defaultnum=_a;
   }

    function set(int _x) public returns(int){   //Add default num to the input and computes total
        total+=(_x+defaultnum);
        return total;
    }

    function get() public view returns(int,int){    //Get the total and default number
        return (total,defaultnum);
    }
}
```

Contract 2

Unlock the reward if you can solve the puzzle.

Solution

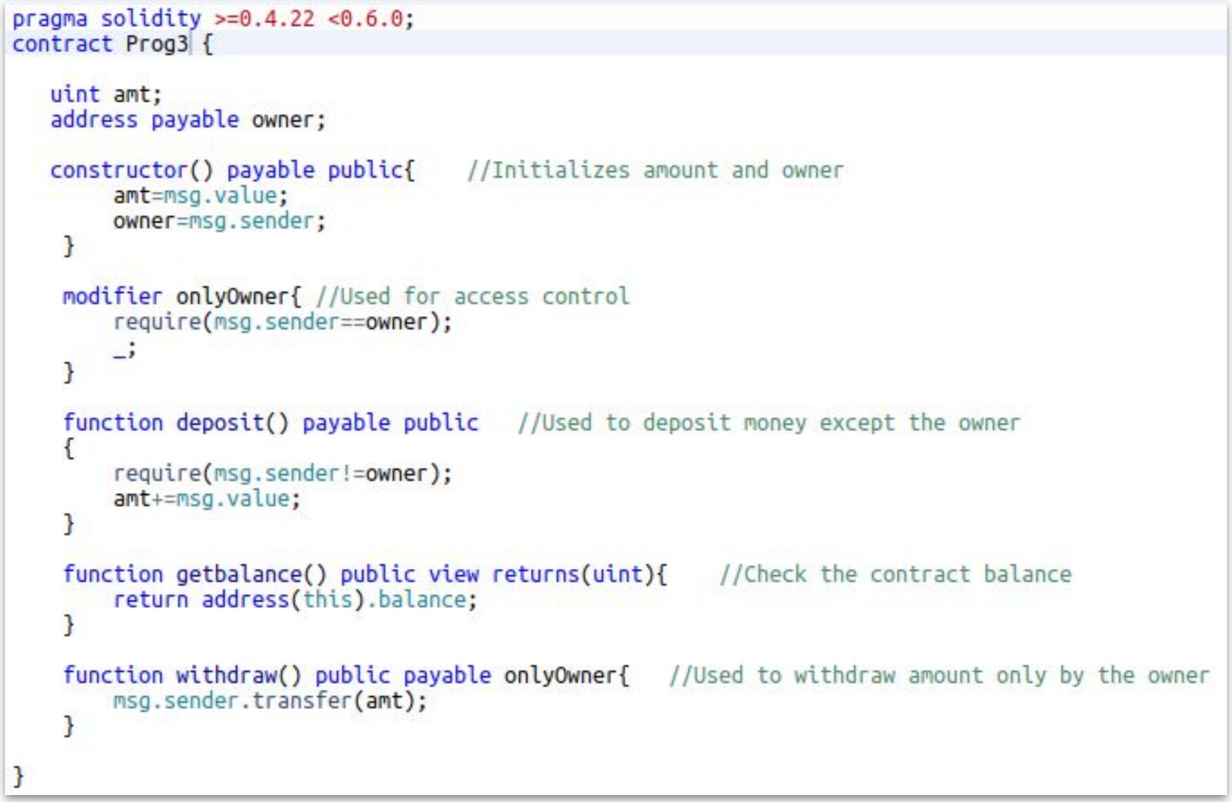


```
pragma solidity >=0.4.22 <0.6.0;
contract Prog2 {

    uint amt;

    constructor() public payable{
        amt=msg.value;
    }

  function specialnum(uint _num) public payable{    //Give a special number and get the reward
      require(_num>=10 && _num<=99);
      uint a;
      uint b;
      uint x;
      uint res;
      x=_num;
      a=_num%10;
      _num=_num/10;
      b=_num%10;
      res=(a+b)+(a*b);
      if(res==x){
          msg.sender.transfer(amt);
      }
  }

}
```

Contract 3

Crowdfunding Contract

Solution



```
pragma solidity >=0.4.22 <0.6.0;
contract Prog3 {

   uint amt;
   address payable owner;

   constructor() payable public{    //Initializes amount and owner
        amt=msg.value;
        owner=msg.sender;
    }

    modifier onlyOwner{ //Used for access control
        require(msg.sender==owner);
        _;
    }

    function deposit() payable public    //Used to deposit money except the owner
    {
        require(msg.sender!=owner);
        amt+=msg.value;
    }

    function getbalance() public view returns(uint){    //Check the contract balance
        return address(this).balance;
    }

    function withdraw() public payable onlyOwner{   //Used to withdraw amount only by the owner
        msg.sender.transfer(amt);
    }

}
```

(Hands-on)-II

- **Solidity - Some advanced features**
- **Secure and Fair MPC on Blockchain**
  - Coin Toss Smart Contract
- **Tools for Decentralized Applications(DApps)**

Any Questions?