Brief Introduction to Smart Contracts and Solidity

CS 731: Blockchain Technology and Applications
Instructor: Chavhan Sujeet Yashavant
What is Ethereum?

It’s a Blockchain, With following additions

- A built-in programming Language
- Two types of accounts
  - User Accounts (Controlled by Private Keys)
  - Contract Accounts (Controlled by Code)
- Anyone can create an app by defining it as a Contract

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Smart Contracts
Smart contracts

- Tiny computer programs
- Stored inside a blockchain

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Smart Contract

- A code that resides on blockchain
- Executes when certain predetermined conditions are satisfied
Smart Contract

- agreement between mutually distrusting participants
- automatically enforced by the consensus mechanism of the blockchain
- without relying on a trusted authority.

What a Contract can Do?

- Send ETH to other contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage
- Call (i.e. start execution in) other Contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Smart Contract Execution

- Every (full) node on Ethereum network processes every transaction
pragma solidity ^0.4.17;

contract Inbox {

    string public message;

    function Inbox(string initialMessage) public {
        message = initialMessage;
    }

    function setMessage (string newMessage) public {
        message = newMessage;
    }
}

bytecode:
"6060604052345600f90000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Ethereum Virtual Machine (EVM)

- Global Singleton Computing Machine with a shared ledger of data
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Crowdfunding platform

Funded!

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Kickstarter for Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/
Kickstarter for Crowdfunding platform

Trusting a third-party is required

We can build a similar system with a Smart Contracts without the requirement of any third party
Kickstarter with Smart Contract

Supporters

Product team
Kickstarter with Smart Contract

Supporters

Product team
Kickstarter with Smart Contract

Supporters

Funded!

Product team

💲🔍 الطبقة هي الأصلية. 🍤💰🔍

２５
Kickstarter with Smart Contract

Supporters

Product team
Kickstarter with Smart Contract

Failed!

Supporters

Product team
pragma solidity ^0.4.17;

contract Inbox {
    string public message;

    function Inbox(string initialMessage) public {
        message = initialMessage;
    }

    function setMessage(string newMessage) public {
        message = newMessage;
    }

    function getMessage() public view returns (string) {
        return message;
    }
}
Introduction to Solidity: Version Pragma

```
pragma solidity ^0.4.17;
```

- Instructions to the compiler on how to treat the code.
- All solidity source code should start with a “version pragma” which is a declaration of the version of the solidity compiler this code should use.
- This helps the code from being incompatible with the future versions of the compiler which may bring changes.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
Introduction to Solidity: Contract keyword

```solidity
pragma solidity ^0.4.17;
contract Inbox {
    string public message;
    function Inbox(string initialMessage) public {
        message = initialMessage;
    }
    function setMessage(string newMessage) public {
        message = newMessage;
    }
    function getMessage() public view returns (string) {
        return message;
    }
}
```

It declares a contract under which is the code encapsulated.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
Introduction to Solidity: State Variables

```solidity
pragma solidity ^0.4.17;

contract Inbox {

    string public message;

    function Inbox(string initialMessage) public {
        message = initialMessage;
    }

    function setMessage(string newMessage) public {
        message = newMessage;
    }

    function getMessage() public view returns (string) {
        return message;
    }
}
```

Permanently stored in contract storage → written to Ethereum Blockchain.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
pragma solidity ^0.4.17;

contract Inbox {

    string public message;

    function Inbox(string initialMessage) public {
        message = initialMessage;
    }

    function setMessage(string newMessage) public {
        message = newMessage;
    }

    function getMessage() public view returns (string) {
        return message;
    }
}

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
pragma solidity ^0.4.17;

contract Inbox {
  string public message;
  function Inbox(string initialMessage) public {
    message = initialMessage;
  }
  function setMessage(string newMessage) public {
    message = newMessage;  // View
  }
  function getMessage() public view returns (string) {
    return message;
  }
}

Visibility

View

keyword
Introduction to Solidity: Function Visibility

- **Public** - any contract and account can call
- **Private** - only inside the contract that defines the function
- **External** - only other contracts and accounts can call
- **Internal** - only inside contract that inherits an internal function

Ref: https://solidity-by-example.org/visibility/
Introduction to Solidity: View and Pure functions

- View function declares that no state will be changed.
- Pure function declares that no state variable will be changed or read.

Ref: https://solidity-by-example.org/visibility/
Introduction to Solidity: Code Execution on Real Blockchain (Try this after success on Local Blockchain)

- **Testnet** (most of the course projects will do it):
  - Can use **Remix** and **Metamask**
  - Can use **hardhat** to deploy on **Goerli Testnet**

- **Mainnet**
  - Require real money
  - Do not try unless you become expert

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
Introduction to Solidity: Code Execution on Local Blockchain (Try this first)

- Offline (Blockchain inside local machine): I will post a video link on Discord about how to do it. It takes time.
  - Can use **Remix** and **Ganache**

- Online (Blockchain inside browser): Remix IDE
  - Simple one, first try this
  - Let’s see a Demo

Ref: https://www.geeksforgeeks.org/introduction-to-solidity
pragma solidity ^0.4.19;

contract Crowdfunding {
    address owner;
    uint256 deadline;
    uint256 goal;
    mapping(address => uint256) public pledgeOf;

    function Crowdfunding(uint256 numberOfDays, uint256 _goal) public {
        owner = msg.sender;
        deadline = now + (numberOfDays * 1 days);
        goal = _goal;
    }

    function pledge(uint256 amount) public payable {
        require(now < deadline); // in the fundraising period
        require(msg.value == amount);
        pledgeOf[msg.sender] += amount;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(now >= deadline); // in the withdrawal period
        require(msg.sender == owner);
        msg.sender.transfer(address(this).balance);
    }

    function getRefund() public {
        require(address(this).balance < goal); // funding goal not met
        require(now >= deadline); // in the withdrawal period
        uint256 amount = pledgeOf[msg.sender];
        pledgeOf[msg.sender] = 0;
        msg.sender.transfer(amount);
    }
}

pragma solidity ^0.8.4;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping(address => uint) public balances;
    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);
    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }
    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }
    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);
    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
Gas

- Halting problem
  - Can’t tell whether a program will halt or run infinitely

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Gas

- Halting problem
  - Can’t tell whether a program will halt or run infinitely

- Solution: Gas Limit

Ref: DEVC0N1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Gas Limit

- Each opcode has a fixed amount of gas assigned and is a measure of computational effort.
- Gas is the execution fee, paid by the sender of the transaction that triggered the computation.

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Gas Limit

- User sets max amount of Gas for a transaction
- Gas is lost if a user run out of Gas Limit, all changes are reversed
- If a transaction uses less gas than gas limit, then user gets remaining Gas

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Gas Limit

- Total fees = Total amount of Gas used \times gas Price
- The gas price is not fixed

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Why Optimization?

- Caller needs to pay Gas according to Computational Steps

Why Optimization?

- Caller needs to pay Gas according to Computational Steps
- Optimization $\Rightarrow$ Gas Saving $\Rightarrow$ Money Saving

Why Optimization?

- Caller needs to pay Gas according to Computational Steps
- Optimization $\Rightarrow$ Gas Saving $\Rightarrow$ Money Saving
- A smart contract gets invoked many times, a small optimization can result in huge saving

Gas Costly Pattern 1: Dead Code

Unoptimized Code

```solidity
function p1 ( uint x )
{
    if (x > 5)
    {
        if (x*x < 20)
        {
            Statement 1
            Statement 2
        }
        Statement 3
    }
}
```

Optimized Code

```solidity
function p1_opt ( uint x )
{
    if (x > 5)
    {
        Statement 2
    }
    Statement 3
}
```

Gas Costly Pattern 2: Opaque Predicate

Unoptimized Code

```solidity
function p2 ( uint x ) {
    if (x > 5)
        if (x > 1)
            ........
}
```

Optimized Code

```solidity
function p2_opt ( uint x ) {
    if (x > 5)
        ........
}
```

Gas Costly Pattern 3: Constant outcome of a Loop

```solidity
function p4 returns (uint) {
    uint sum = 0;
    for (uint i = 1; i <= 100; i++)
        sum += i;
    return sum;
}
```

```solidity
function p4_opt returns (uint) {
    return 5050;
}
```

Gas Costly Pattern 4: Comparison with Unilateral Outcome in a Loop


```solidity
1 function p7 (uint x, uint y) returns (uint)
2 {
3     for (int i = 0; i < 100; i++)
4         if (x > 0)
5             y += x;
6     return y;
7 }

1 function p7_opt (uint x, uint y) returns (uint)
2 {
3     if (x > 0)
4         for (int i = 0; i < 100; i++)
5             y += x;
6     return y;
7 }
```
Smart Contract Security

- Correctness is ensured by the consensus mechanism.
- Unfortunately, correctness is not sufficient to make Smart Contracts secure.

Smart Contract Vuln 1: Overflow and Underflow

```solidity
mapping (address => uint256) public balanceOf;

// INSECURE
function transfer(address _to, uint256 _value) {
    require(balanceOf[msg.sender] >= _value);
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}

// SECURE
function transfer(address _to, uint256 _value) {
    require(balanceOf[msg.sender] >= _value &&
            balanceOf[_to] + _value >= balanceOf[_to]);
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}
```

Smart Contract Vulnerability 2: Default Visibility

```solidity
contract Puzzle {
    uint256 amount = 0.5; // 0.5 ethers
    function submit_answer (string answer) {
        // Logic to check submitted answer
        if (/*answer is correct*/) _sendEther();
    }
    function _sendEther() {
        msg.sender.transfer(amount);
    }
}
```

Smart Contract Vuln 3: Timestamp Dependence

● A smart contract that utilizes a current timestamp to produce random numbers in order to determine lottery results
● Miners can put a timestamp within 30 seconds of block validation
● Miners can alter outcome of random number generator

Smart Contract Vuln 3: Timestamp Dependence

```solidity
pragma solidity ^0.5.0;

contract TimedCrowdsale {
  event Finished();
  event notFinished();
  // Sale should finish exactly at January 1, 2019
  function isSaleFinished() private returns (bool) {
    return block.timestamp >= 1546300800;
  }

  function run() public {
    if (isSaleFinished()) {
      emit Finished();
    } else {
      emit notFinished();
    }
  }
}
```

Ref: https://swcregistry.io/docs/SWC-116
THE END
Backup Slides
## Gas assigned per Opcode

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD/SUB</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MUL/DIV</td>
<td>5</td>
<td>Arithmetic Operation</td>
</tr>
<tr>
<td>ADDMOD/MULMOD</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>AND/OR/XOR</td>
<td>3</td>
<td>Comparison Operation</td>
</tr>
<tr>
<td>LT/GT/SLT/SGT/EQ</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>3</td>
<td>Stack Operation</td>
</tr>
</tbody>
</table>

## Gas assigned per Opcode

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>400</td>
<td>Get balance of an account</td>
</tr>
<tr>
<td>CREATE</td>
<td>32000</td>
<td>Create a new account using CREATE</td>
</tr>
<tr>
<td>CALL</td>
<td>25000</td>
<td>Message-call into an account</td>
</tr>
</tbody>
</table>

# Gas assigned per Opcode

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLOAD/MSTORE</td>
<td>3</td>
<td>Memory Operation</td>
</tr>
<tr>
<td>JUMP</td>
<td>8</td>
<td>Unconditional Jump</td>
</tr>
<tr>
<td>JUMPI</td>
<td>10</td>
<td>Conditional Jump</td>
</tr>
<tr>
<td>SLOAD</td>
<td>200</td>
<td>Storage Operation</td>
</tr>
<tr>
<td>SSTORE</td>
<td>5000/20000</td>
<td></td>
</tr>
</tbody>
</table>

*Ethereum project yellow paper* (2014)