OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems Advanced & Blockchain

Blockchain, Ethereum

Thomas Bocek

11 October 2020

Solidity - http://solidity.readthedocs.io

- Version Pragma - reject compiled with specific compiler versions

```
pragma solidity ^0.4.23; //not before 0.4.23, not after 0.5.0
```

- Comments

```
// This is a single-line comment.

/*
This is a
multi-line comment.
*/
```

https://learnxinyminutes.com/docs/solidity/

https://solidity.readthedocs.io/en/v0.7.3/

- Contract with State Variables (state change expensive, see gas prices)

```
pragma solidity ^0.5.7;
//minimal contract
contract Example1 {
    uint256 counter;
}
```

OST

Solidity IDE - https://remix.ethereum.org

- Types
  - Value types: bool: true and false, int / **uint** (int8, int16, …, int256), **address**: 20 bytes, fixed size arrays: bytes1, bytes2, bytes3, ..., bytes32, variable sized: bytes (push), strings
  - Reference types: memory / storage
- Structs

```
struct Account {
  string name;
  uint256 amount;
}
```

- Mapping (you cannot iterate over mappings, as a key array)

```
mapping (address => uint256) accounts; //mapping with basic types
mapping (uint256 => Account) accounts; //mapping with structs
```

OST

Solidity – Basics

- Arrays

```
string[] names

uint256 newLength = names.push("John");
```

- Another contract

```
pragma solidity ^0.7.3;

contract Example2 {

    struct Account {

       string addr;

       uint256 amount;

    }
    uint256 public counter;

    mapping (uint256 => Account) accounts;

}
```

- Create getter/setter

```
function get(uint256 nr) public view returns (string memory) {

    return accounts[nr].addr;

}

function set(uint256 nr, string memory addr) public {

    require(owner == msg.sender);

    accounts[counter++] = Account(addr, nr);

}
```

- Read state variables
  - Constant function does not modify state
  - Reads “for free”

OST

Solidity – Functions

- «Standard» Functions: Can read and modify the state

```
uint256 counter;
function setCounter(uint256 _newValue)
public {
    counter = _newValue;
}
```

- View Functions: Do not modify the state (it's free!)

```
uint256 counter;
function getCounter() public view returns
(uint256) {
    return counter;
}
```

- Pure Functions: Do not read from or modify the state.

```
function multiply(uint256 a,
uint256 b) pure returns (uint256)
{
    return a * b;
}
```

OST

Solidity – Functions

- Internal Functions: Only callable internally

```
uint256 counter;
function setCounter(uint256 _newValue) internal
{
    counter = _newValue;
}
```

- External Functions: Only callable externally
  Note: public and external differs in terms of gas usage

```
function setValues(uint256[20] _values)
external {
    // do something
}
```

- Payable Functions: Receives plain Ether

```
function deposit() public
payable {
  // Access msg.value to
  //get amount of Ether
}
```

OST

Solidity – Basics

Creating a smart contract that can get or set the balance of an address the *wrong* way.

```
contract BankExample {
  mapping(address => uint256) private balances;

  function getBalance(address _address) view public returns (uint256) {
    return balances[_address];
  }
  function setBalance(uint256 _newBalance, address _address) public {
    balances[_address] = _newBalance;
  }
}
```

OST

Solidity – Basics

- Special Variables and Global Functions
  - `block.number` (type of uint256): current block number
  - `block.difficulty` (type of `uint256`): current block difficulty
  - `block.gaslimit` (type of `uint256`): current block gaslimit
  - `gasleft() returns (uint 256)`: remaining gas
  - `msg.sender` (type of `address`): sender of the message
  - `msg.value` (type of `uint256`): number of wei sent with the message
  - `block.time` (type of `uint256`): current block timestamp

OST

Smart Contract Ownership and Events

OST

Solidity – ownership

- Preconditions

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

- Set owner

```
pragma solidity ^0.7.3;
contract Example3 {
    address owner;
    constructor() {
        owner = msg.sender;
    }
}
```

- Events – notifications

```
pragma solidity ^0.7.3;
contract Example4 {
    event Message(string msg);
```

- E.g. in the Geth console

```
at =
browser_example1_sol_example2Contract.at(
"0x…")

var event = at.Message(function(error,
result){console.log(result.args.msg)});
```

OST

Solidity – Ownership

- Who is the owner of a smart contract?
- How to transfer ownership?
- The *wrong* way:

```
contract OwnedContract {
    address owner;
    constructor() public {
        owner = msg.sender;
    }

    function transfer(address _newOwner) public
{
        owner = _newOwner;
    }
}
```

OST

Solidity – Basics

- Everybody can transfer the ownership
- Use `require` to test user inputs

```
if (msg.sender != owner) {
throw; }
// Do something only the owner
is allowed to
```

behaves exactly the same as

```
require(msg.sender == owner,
"Unauthorized");
// Do something only the owner
is allowed to
```

- The use of revert(), assert(), and require() in Solidity

```
contract OwnedContract {
  address owner;
  constructor() public {
    owner = msg.sender;
  }

  function transfer(address _newOwner) public
{
    require(owner == msg.sender,
        "Sender not authorized");
    owner = _newOwner;
  }
}
```

OST

Solidity – Events/Notifications

- Events are a way for smart contracts written in Solidity to log that something has occurred
- Interested observers, notably JavaScript front ends for decentralized apps, can watch for events and react accordingly.

Solidity – Events/Notifications

```
contract EventsExample {

  event OwnerChanged(
    address _oldOwner,
    address _newOwner
  );

  function transfer(address _newOwner) public {
    require(owner == msg.sender, "Sender not authorized");
    emit OwnerChanged(owner, _newOwner);
    owner = _newOwner;
  }

}
```

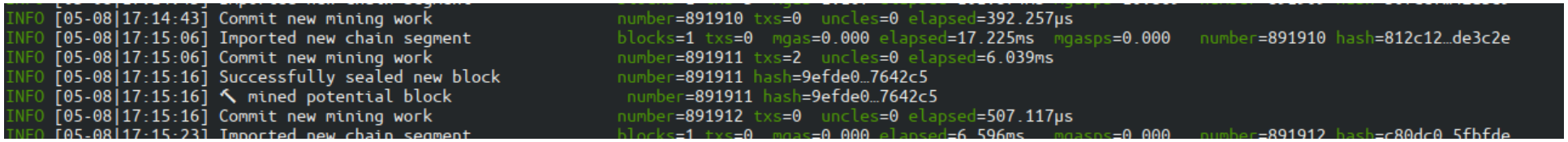
OST

Smart Contract Deployment

OST

Solidity Deployment (with a light client)

- Create address (geth) – or parity, or MEW, or…
  - Ethereum address = create random private key → derive public key → hash it, take last 20bytes
    - geth --rinkeby account new
- Important sites
  - https://rinkeby.etherscan.io/ (blockchain explorer)



  - No mining (use twitter with https://www.rinkeby.io)
- Start geth
  - geth --rinkeby --syncmode light --rpc --rpccorsdomain '*' --rpcapi "eth,net,web3,personal" --unlock 0 --password <(echo 123456) --allow-insecure-unlock
  - geth attach http://127.0.0.1:8545 or
  - Connect Remix Solidity IDE to geth / or use JavaScript Ethereum blockchain in Remix

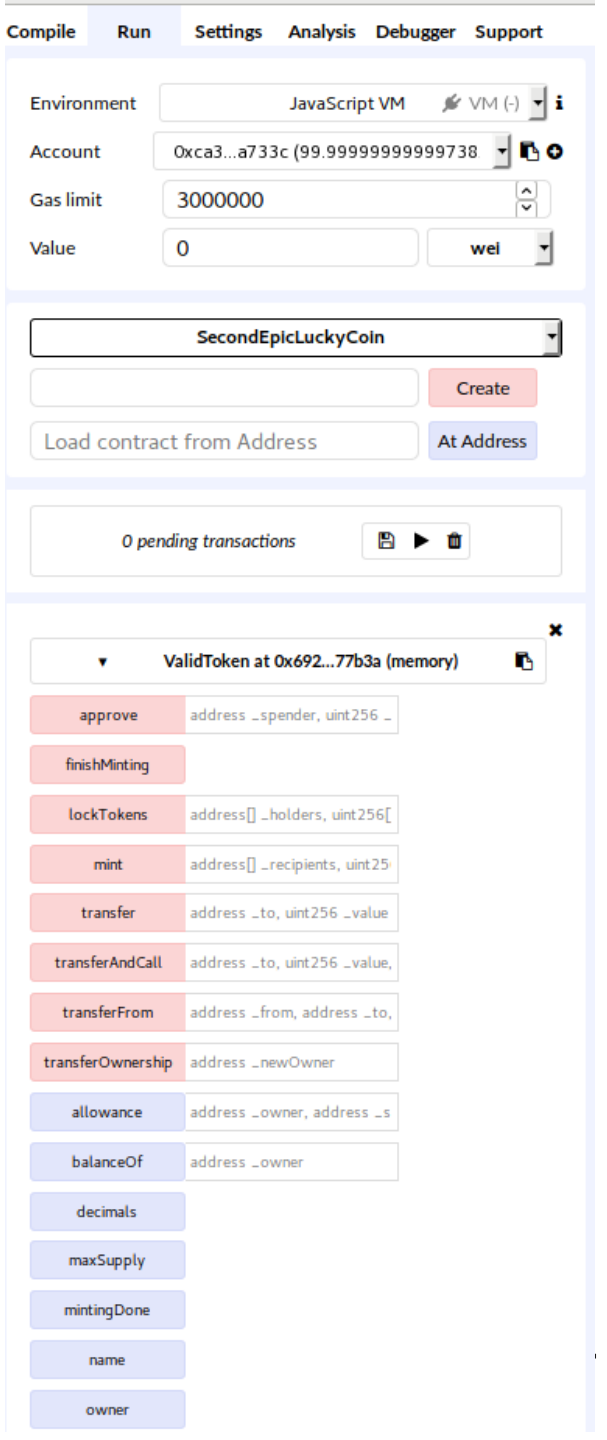
OST

Solidity IDE

- Create your first contract

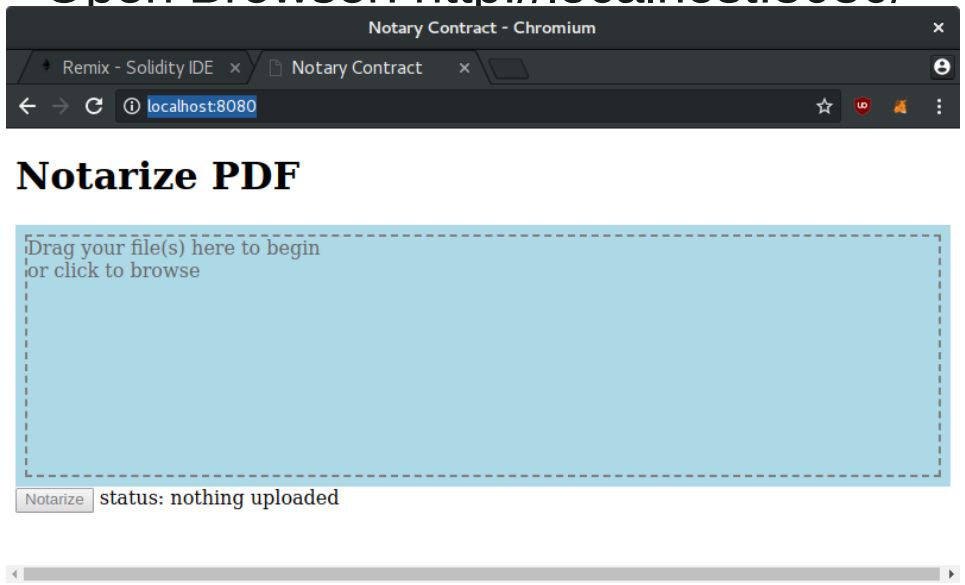
```
pragma solidity ^0.7.3;

//minimal contract

contract Example1 {

    uint counter;

}
```

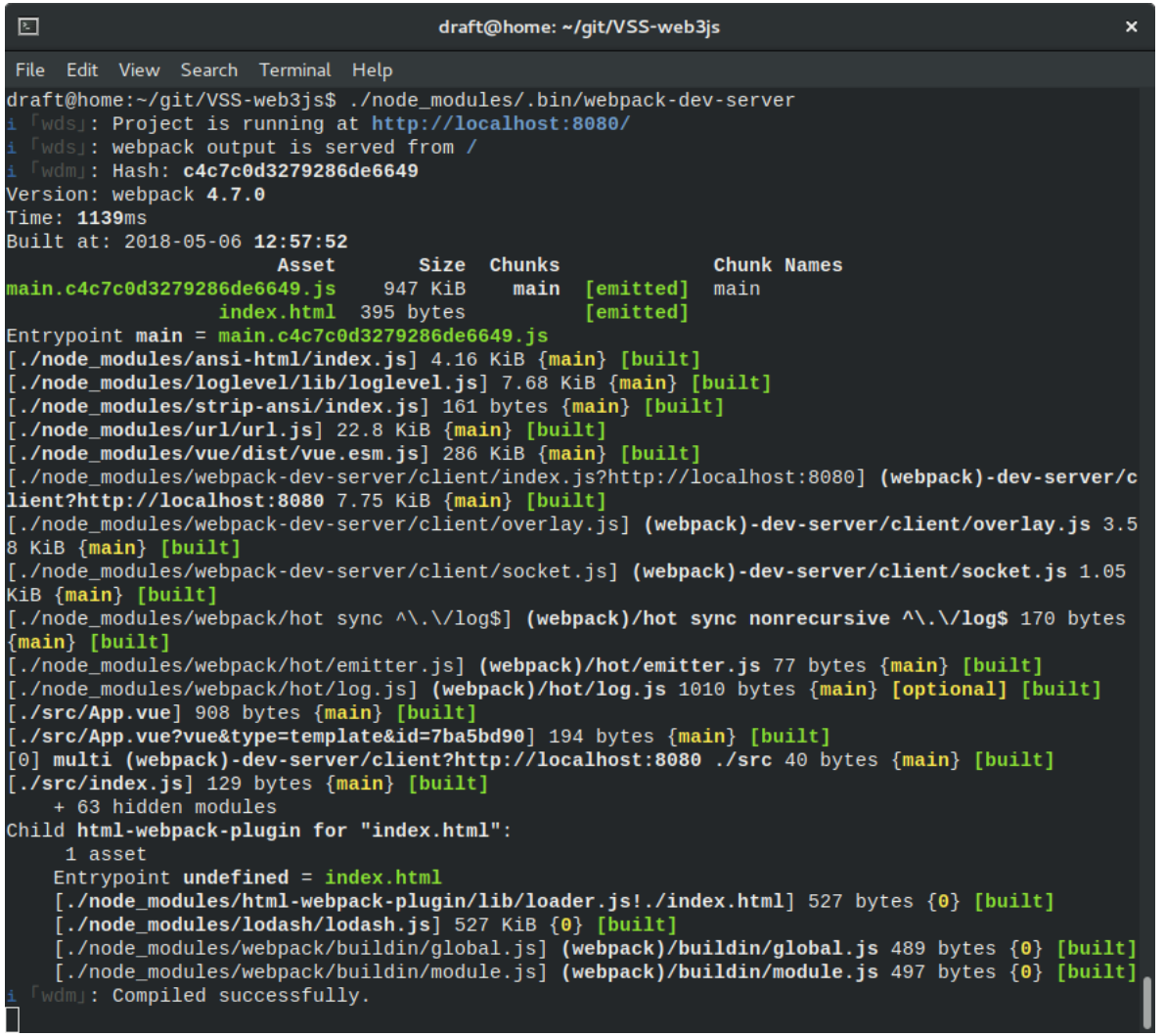
- Remix – Solidity IDE – Environment: Web3 Provider
  - http://localhost:8545
- «Deploy» contract
- Mixed content - workaround: use http
  - **https://remix.ethereum.org**
- Select Web3 Provider (geth is your Web3 Provider)
  - Alternatively: use scripts, MetaMask, or JavaScript EVM



Example

- Installation
  - yarn install
  - ./node_modules/.bin/webpack-dev-server
- Open Browser: http://localhost:8080/

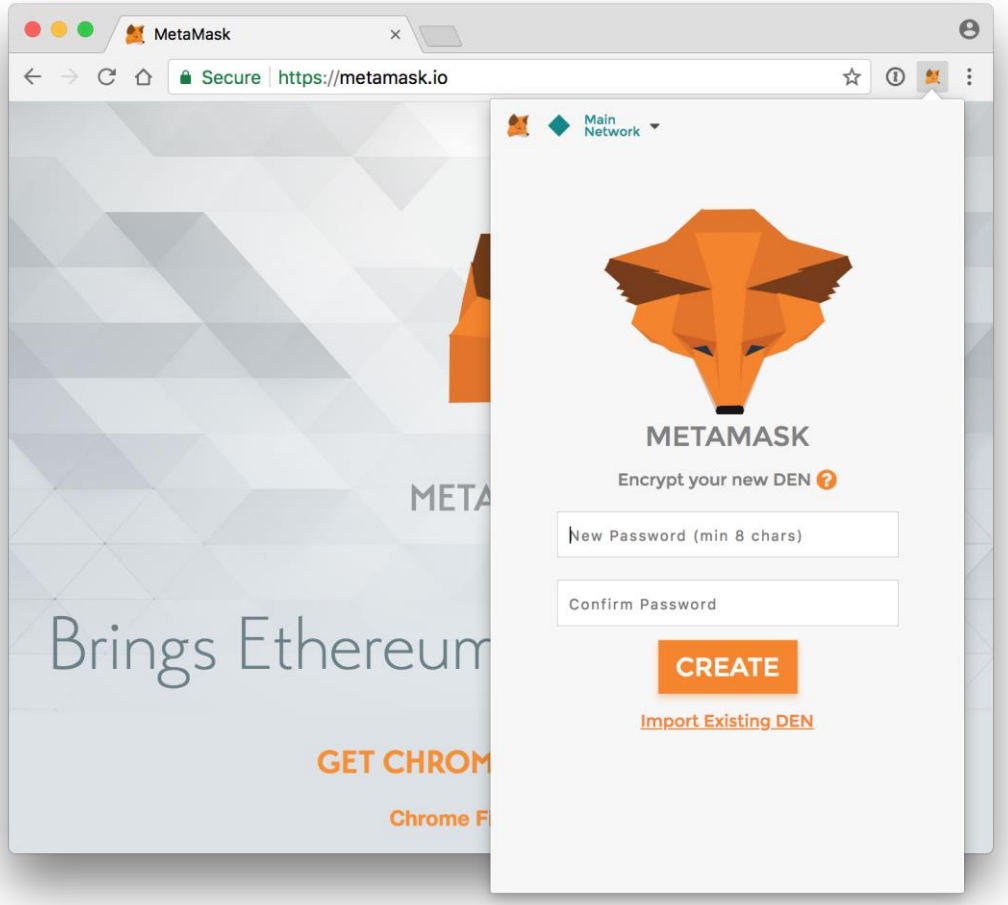




MetaMask

- MetaMask
  - Browser plugin to make Ethereum transactions in browsers
  - Manage your key pairs and sign blockchain transactions
  - MetaMask injects javascript library - web3.js
  - Uses infura

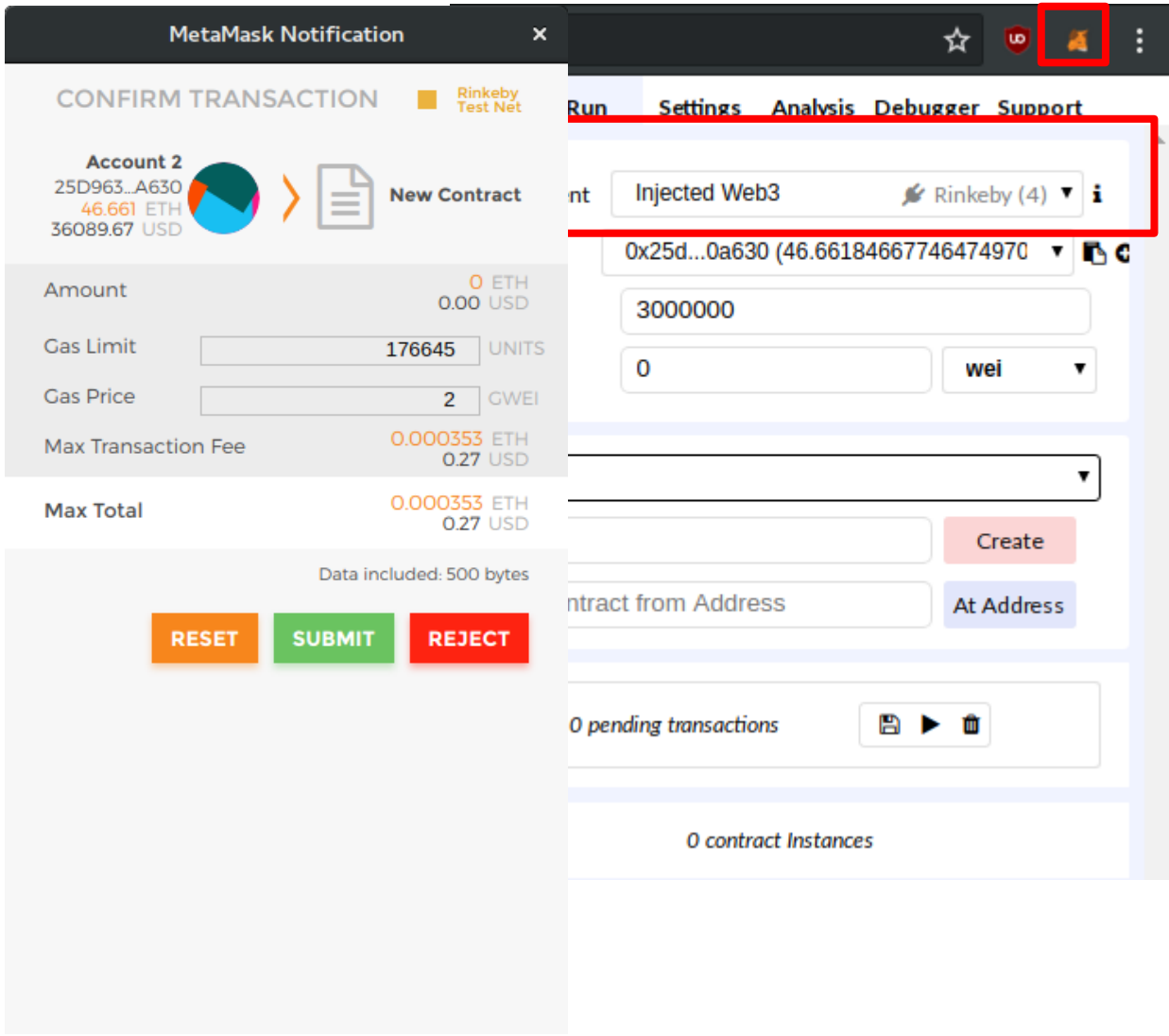
- Remix IDE: https://remix.ethereum.org
  - Use Notary.sol from https://github.com/tbocek/VSS-WEB3/blob/master/Notary.sol
  - Alternatively, use IntelliJ solidity plugin and deploy via geth, parity, or a local test blockchain



OST

Create Contract

- Connect to MetaMask
  - Injected Web3
  - If you see your accounts, good!
- Create contract!
  - Add address to the code (manually)
- Store value
  - 0x654cf88c**4cff3e62696f7cbb55fbba922b2a75897a010347e371e9398b658c4affa611aa**
  - Hash is: 0x4cff3e62696f7cbb55fbba922b2a75897a010347e371e9398b658c4affa611aa
  - What is 0x654cf88c?
- First four bytes of the Keccak (SHA-3) hash of the signature of the function.
  - Keccak(store(bytes32))



OST

Smart Contract Tokens

OST

ERC20

- ERC20 is a technical standard for smart contracts on the Ethereum blockchain for implementing tokens
  - proposed on November 19, 2015
  - Oct 2020: more than 310’000 ERC20 token contracts: USDT, USDC, WBTC, MOD Token
- ERC20 Token (Simplified)
  - No allowance / approval / transferFrom, also no Approval event
  - Creator gets 1000 coins
  - Using SafeMath (always use it)

```
// SPDX-License-Identifier: BSD

pragma solidity ^0.7.3;

abstract contract ERC20 {
    function totalSupply() virtual public view returns (uint256);
    function balanceOf(address _owner) virtual public view returns (uint256 balance);
    function transfer(address _to, uint _value) virtual public returns (bool success);
    event Transfer(address indexed _from, address indexed _to, uint _value);
}

contract YourTestToken is ERC20 {
    string public constant name = "ADSB 2020 Token";
    string public constant symbol = "A20";

    mapping (address => uint) balances;

    constructor() {
        balances[msg.sender] = 1000;
        emit Transfer(address(0), msg.sender, 1000);
    }

    function totalSupply() override public pure returns (uint256) {
        return 1000;
    }
    function balanceOf(address _owner) override public view returns (uint256) {
        return balances[_owner];
    }
    function transfer(address _to, uint256 _value) override public returns (bool) {
            require(balances[msg.sender] >= _value);
            balances[msg.sender] -= _value;
            balances[_to] += _value;
            emit Transfer(msg.sender, _to, _value);
    }
}
```


OST

ERC20 Token Standard

- Describes the functions and events that an Ethereum token contract has to implement

```
contract ERC20 {
    function totalSupply() public view returns (uint256 totalSupply);
    function balanceOf(address _owner) public view returns (uint256 balance);
    function transfer(address _to, uint _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
        public returns (bool success);
    function approve(address _spender, uint _value)
      public returns (bool success);
    function allowance(address _owner, address _spender) public view
      returns (uint remaining);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
    event Transfer(address indexed _from, address indexed _to, uint _value);
}
```

- Understanding ERC20 Token Contracts

OST

ERC20 Token Standard

- Our token: only partial ERC20 support

```
contract ERC20 {
    function totalSupply() public view returns (uint256 totalSupply);
    function balanceOf(address _owner) public view returns (uint256 balance);
    function transfer(address _to, uint _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
      public returns (bool success);
    function approve(address _spender, uint _value)
      public returns (bool success);
    function allowance(address _owner, address _spender) public view
      returns (uint remaining);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
    event Transfer(address indexed _from, address indexed _to, uint _value);
}
```

OST