

Hands-On: Solidity for beginners

Poltyech Lille 2019

santiago.bragagnolo@inria.fr

<https://github.com/sbragagnolo/hands-on-solidity>

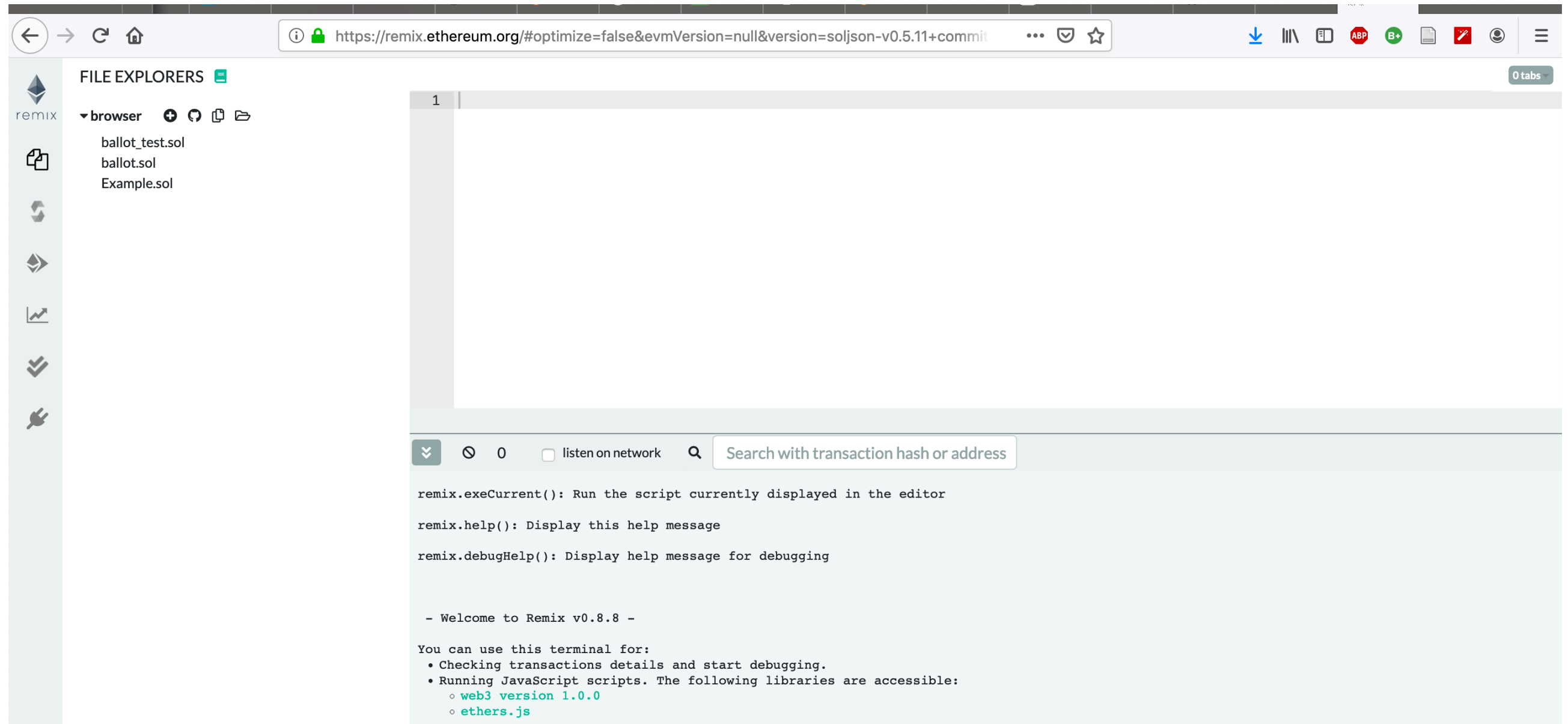
Solidity

- Ethereum's smart contract main language
- High level language inspired in Go, C, Javascript
- Statically typed (Java, C, C++, C#, Go, etc)

Solidity Web-IDE: Remix

<https://remix.ethereum.org/>

Solidity Web-IDE: Remix



First contract

FirstContract.sol ✕

```
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6      function setName(string _name){
7          name = _name;
8      }
9      function getName () returns (string) {
10         return name;
11     }
12 }
13
14 }
15
```

Primitive types

FirstContract.sol ✕

```
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6      function setName(string _name){
7          name = _name;
8      }
9      function getName () returns (string) {
10         return name;
11     }
12 }
13
14 }
15
```

Primitive types

- bool
- int / uint ... int8 / uint8 ... int256 / uint256
- string
- byte / bytes / bytes1... bytes32
- address

Functions

FirstContract.sol ✕

```
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6      function setName(string _name){
7          name = _name;
8      }
9      function getName () returns (string) {
10         return name;
11     }
12 }
13
14 }
15
```


Functions

- The “rules” enforced by a contract is defined by its functions
- It can return more than one value (or none at all)
- Functions can be overloaded

First compiling errors :)

```
FirstContract.sol x
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6  x function setName(string _name){
7      name = _name;
8  }
9  x function getName () returns (string) {
10     return name;
11
12 }
13
14 }
15
```

Built-in method modifiers

- Visibility Modifiers: private, public, internal, external
- Mutability Modifiers: view, pure
- Cryptocurrency Modifiers: payable

Built-in method modifiers

FirstContract.sol ✕

```
1 pragma solidity ^0.5.1;
2
3 contract FirstContract {
4     string name;
5
6     function setName( string _name) public {
7         name = _name;
8     }
9     function getName () public view returns (string) {
10         return name;
11     }
12 }
13
14 }
15
16
17 |
```

More compiling errors

:(

```
FirstContract.sol ✕
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6      function setName( string _name) public {
7          name = _name;
8      }
9      function getName () public view returns (string) {
10         return name;
11     }
12 }
13
14 }
15
16
```

More compiling errors

:(

```
FirstContract.sol x
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      string name;
5
6      function setName( string _name) public {
7          name = _name;
8      }
9      function getName () public view returns (string) {
10         return name;
11     }
12 }
13
14 }
15
16
```

Built-in parameter modifiers

- Memory
- Storage

Built-in parameter modifiers

```
FirstContract.sol ✕  
1  pragma solidity ^0.5.1;  
2  
3  contract FirstContract {  
4      string name;  
5  
6      function setName( string memory _name) public {  
7          name = _name;  
8      }  
9      function getName () public view returns (string memory) {  
10         return name;  
11     }  
12 }  
13  
14 }  
15  
16
```


Constructor

- Contracts can have only 1 constructor
- A constructor can have parameters
- Visibility: *public* or *internal*

```
constructor(address a, address b) public {  
    //do something...  
}
```

Pre-defined Variables

- `msg` : reference the current message call
 - `msg.sender` : address; the account that initiated the call
 - `msg.value` : uint; the amount of Wei sent
- `block` : reference the current block
 - `block.timestamp` or `now` : uint, timestamp

Exceptions

- An exception undo all changes and propagates through the call and sub-calls
- **assert(bool condition):** abort execution and revert state changes if condition is false (use for internal error)
- **require(bool condition, string memory message):** abort execution and revert state changes if condition is false (use for malformed input or error in external component). Also provide error message.
- **revert(string memory message):** abort execution and revert state changes providing an explanatory string

Restricted access V1

FirstContract.sol Home Wallet.sol Wallet2.sol Wallet3.sol Sell.sol RestrictedAccess1.sol X RestrictedAccess2.sol

```
1 pragma solidity ^0.5.1;
2
3 contract FirstContract {
4     address private owner;
5     string name;
6
7
8     constructor () public {
9         owner = msg.sender;
10    }
11
12
13    function setName( string memory _name) public {
14        require(owner == msg.sender, "Only the owner can invoke this method");
15        name = _name;
16    }
17    function getName () public view returns (string memory) {
18        return name;
19    }
20 }
21
22
23
24
```

Custom Modifiers

- Modifiers amend the semantics of a function.
- Usually used for checking conditions and raising exceptions.
- Similar to a limited aspect.

```
modifier checkBalance(uint amount){  
    require(address(this).balance >= amount,  
            "Insufficient funds for this operation.");  
    -;  
}
```

Restricted access V2

FirstContract.sol



Home

Wallet.sol

Wallet2.sol

Wallet3.sol

Sell.sol

RestrictedAccess1.sol

RestrictedAccess2.sol

```
1  pragma solidity ^0.5.1;
2
3  contract FirstContract {
4      address private owner;
5      string name;
6
7
8      constructor () public {
9          owner = msg.sender;
10     }
11
12     modifier onlyOwner() {
13         require(owner == msg.sender, "Only the owner can invoke this method");
14         _;
15     }
16
17     function setName( string memory _name) public onlyOwner{
18         name = _name;
19     }
20     function getName () public view returns (string memory) {
21         return name;
22     }
23 }
24
25 }
26
27
```

Dealing with money

- payable addresses implements money transfer methods
 - send
 - transfer
- only payable functions can handle money

Wallet v1

FirstContract.sol

Home

Wallet.sol

```
1  pragma solidity ^0.5.1;
2
3  contract MyWallet{
4      address payable private owner;
5      uint8 constant private version = 1; //just to keep track of the versions
6
7      constructor() public {
8          owner = msg.sender;
9      }
10     modifier onlyOwner(){
11         require(owner == msg.sender);
12         _;
13     }
14     modifier checkBalance(uint amount){
15         require(address(this).balance >= amount);
16         _;
17     }
18     function getBalance() public view returns(uint){
19         return address(this).balance;
20     }
21     function pay(address payable receiver, uint amount) public onlyOwner checkBalance(amount) {
22         receiver.transfer( amount );
23     }
24     function deposit() public payable {
25         //Yes the deposit function is empty
26     }
27
28     function withdraw(uint amount) public onlyOwner checkBalance(amount) {
29         owner.transfer(amount);
30     }
31 } //end of contract
32
```


Events

- Client notification
 - Allows clients to note a change
 - Allows clients to articulate their business
- Logging
 - Debugging
 - Easy auditory

Wallet v2

FirstContract.sol Home Wallet.sol Wallet2.sol x

```
1  pragma solidity ^0.5.1;
2
3  contract MyWallet{
4      address payable private owner;
5      uint8 constant private version = 1; //just to keep track of the versions
6      event PayEvent(address receiver, uint amount);
7      event DepositEvent(address sender, uint amount);
8
9      constructor() public {
10         owner = msg.sender;
11     }
12     modifier onlyOwner(){
13         require(owner == msg.sender);
14         _;
15     }
16     modifier checkBalance(uint amount){
17         require(address(this).balance >= amount);
18         _;
19     }
20     function getBalance() public view returns(uint){
21         return address(this).balance;
22     }
23     function pay(address payable receiver, uint amount) public onlyOwner checkBalance(amount) {
24         receiver.transfer( amount );
25         emit PayEvent(receiver, amount);
26     }
27     function deposit() public payable {
28         emit DepositEvent(msg.sender, msg.value);
29     }
30
31     function withdraw(uint amount) public onlyOwner checkBalance(amount) {
32         owner.transfer(amount);
33     }
34 } //end of contract
35
```

Fallback Function

- Un-named function
- Cannot have any parameter, cannot return any value
- must be external
- could be payable
- Executed when a method name is not found
- Executed (if payable) when transferring money to this contract

Wallet v3

FirstContract.sol



Home

Wallet.sol

Wallet2.sol

Wallet3.sol ✕

```
1 pragma solidity ^0.5.1;
2
3 contract MyWallet{
4     address payable private owner;
5     uint8 constant private version = 1; //just to keep track of the versions
6     event PayEvent(address receiver, uint amount);
7     event DepositEvent(address sender, uint amount);
8
9     constructor() public {
10         owner = msg.sender;
11     }
12     modifier onlyOwner(){
13         require(owner == msg.sender);
14         _;
15     }
16     modifier checkBalance(uint amount){
17         require(address(this).balance >= amount);
18         _;
19     }
20     function getBalance() public view returns(uint){
21         return address(this).balance;
22     }
23     function pay(address payable receiver, uint amount) public onlyOwner checkBalance(amount) {
24         receiver.transfer( amount );
25         emit PayEvent(receiver, amount);
26     }
27     function deposit() public payable {
28         emit DepositEvent(msg.sender, msg.value);
29     }
30
31     function withdraw(uint amount) public onlyOwner checkBalance(amount) {
32         owner.transfer(amount);
33     }
34     function() payable external { //fallback
35         emit DepositEvent(msg.sender, msg.value);
36     }
37 } //end of contract
38
```

State machine

```
FirstContract.sol  Home  Wallet.sol  Wallet2.sol  Wallet3.sol  Sell.sol x
1  pragma solidity ^0.5.1;
2
3  contract Sell {
4      enum State { ON_SALE, WAITING_SEND, SENT, FINISH }
5
6      address _owner;
7      address buyer;
8      uint payed;
9      uint price;
10     string itemName;
11
12     State state;
13     constructor (uint toPay, string memory name) public {
14         _owner = msg.sender;
15         itemName = name;
16         price = toPay;
17     }
18     function prepare() public {
19         state = State.ON_SALE;
20     }
21     function buy() payable public {
22         if (state != State.ON_SALE) { return ; }
23         if( price != msg.value ) revert();
24         state = State.WAITING_SEND;
25         payed = msg.value;
26         buyer = msg.sender;
27     }
28     function informItemReceived () public {
29         if ( buyer == msg.sender && state == State.WAITING_SEND ) {
30             state = State.SENT;
31         }
32     }
33     function withdrawMoneyTo (address payable toAddress) public{
34         if ( _owner == msg.sender && state == State.SENT) {
35             toAddress.transfer(price);
36             state = State.FINISH;
37         }
38     }
39 }
40
```

Tasks

- Explain what this State machine tries to guarantee
- Refactor the contract by using modifiers for the method requirements (such as ensuring the owner)
- How it would be an external application that uses such a contract?