Hands-On: Solidity for beginners

M2 MFCA Lille 2020

santiago.bragagnolo@inria.fr

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

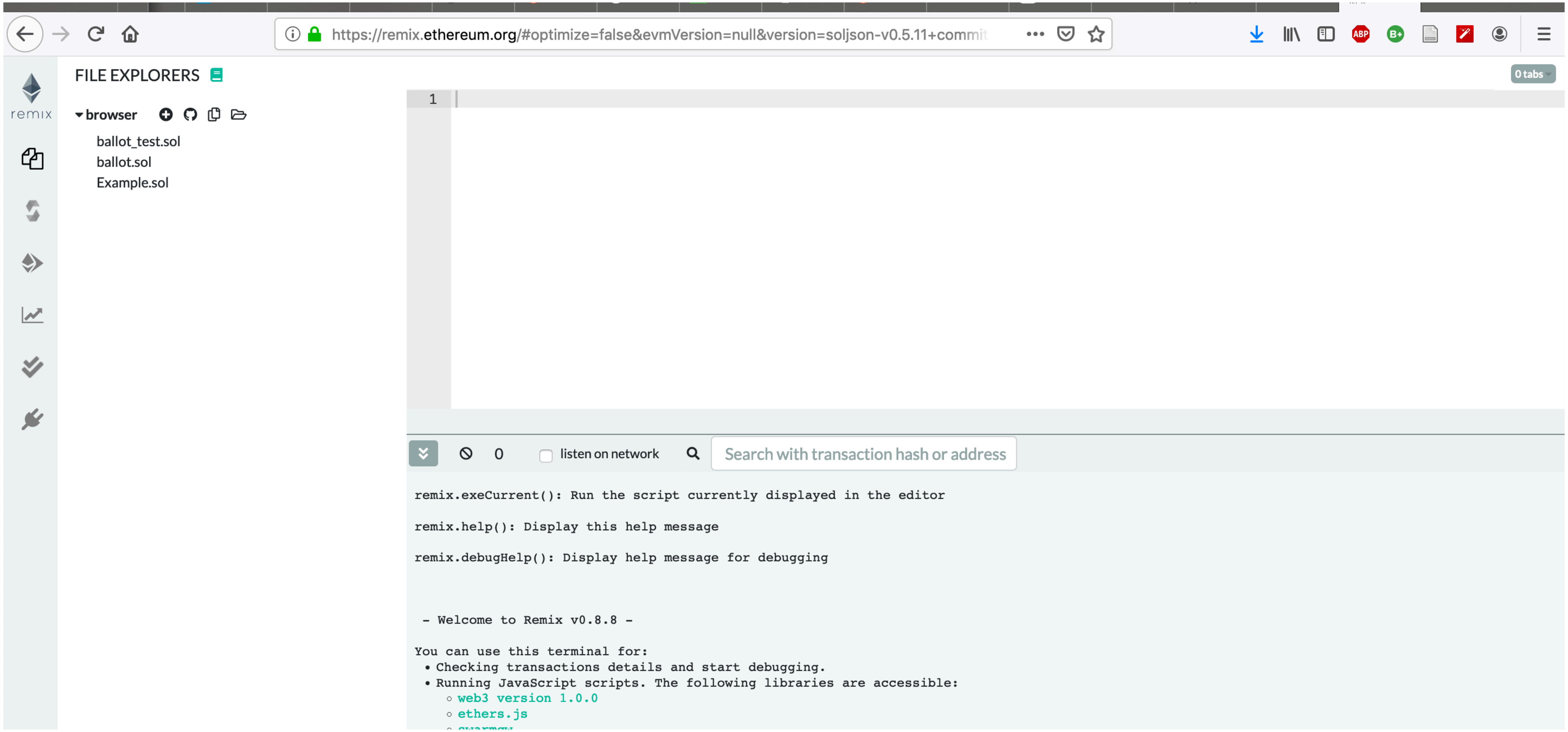
Solidity

- 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/

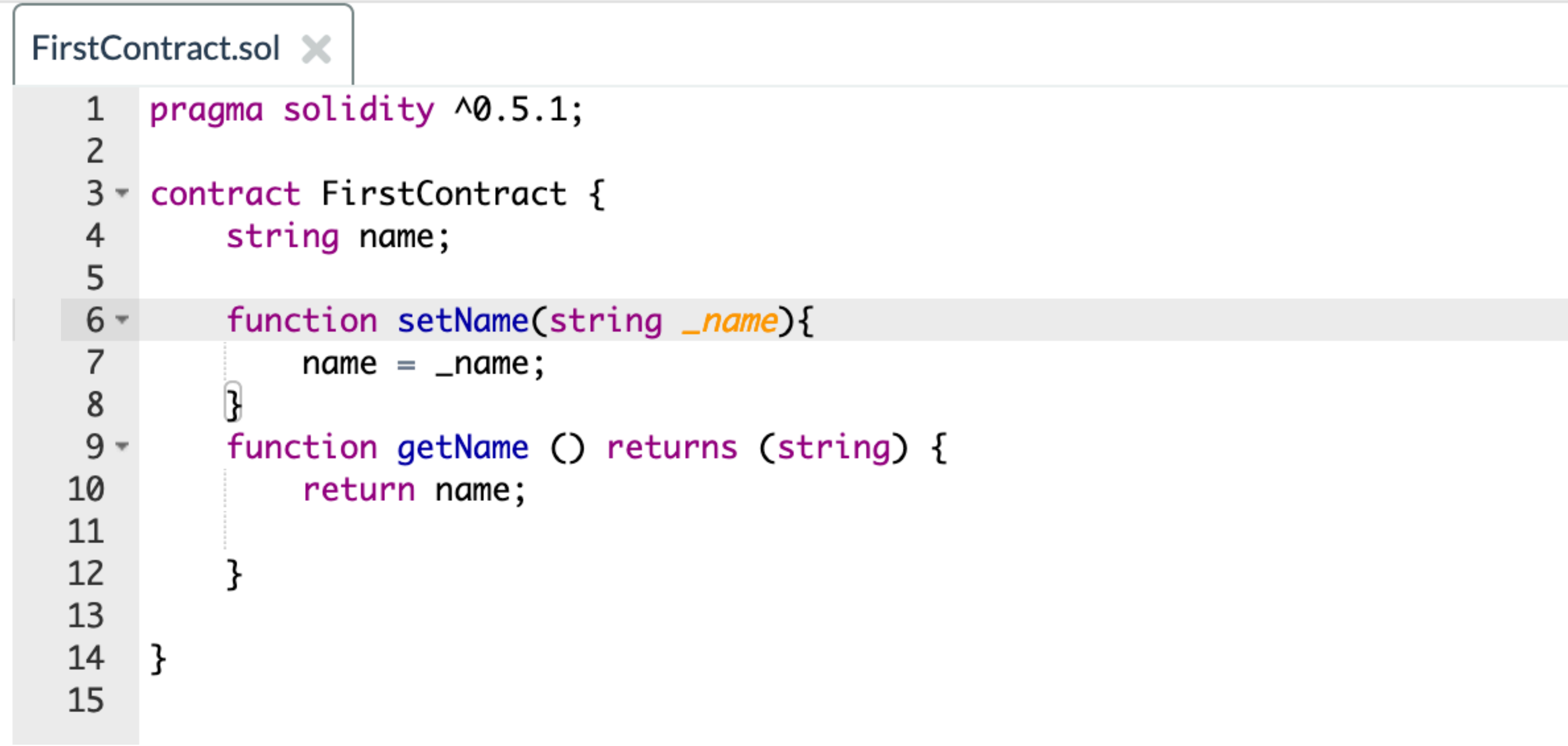
Solidity Web-IDE: Remix



https://remix.ethereum.org/#optimize=false&evmVersion=null&version=soljson-v0.5.11+commit
remix
FILE EXPLORERS
browser
ballot_test.sol
ballot.sol
Example.sol
0 tabs
1
0
listen on network
Search with transaction hash or address

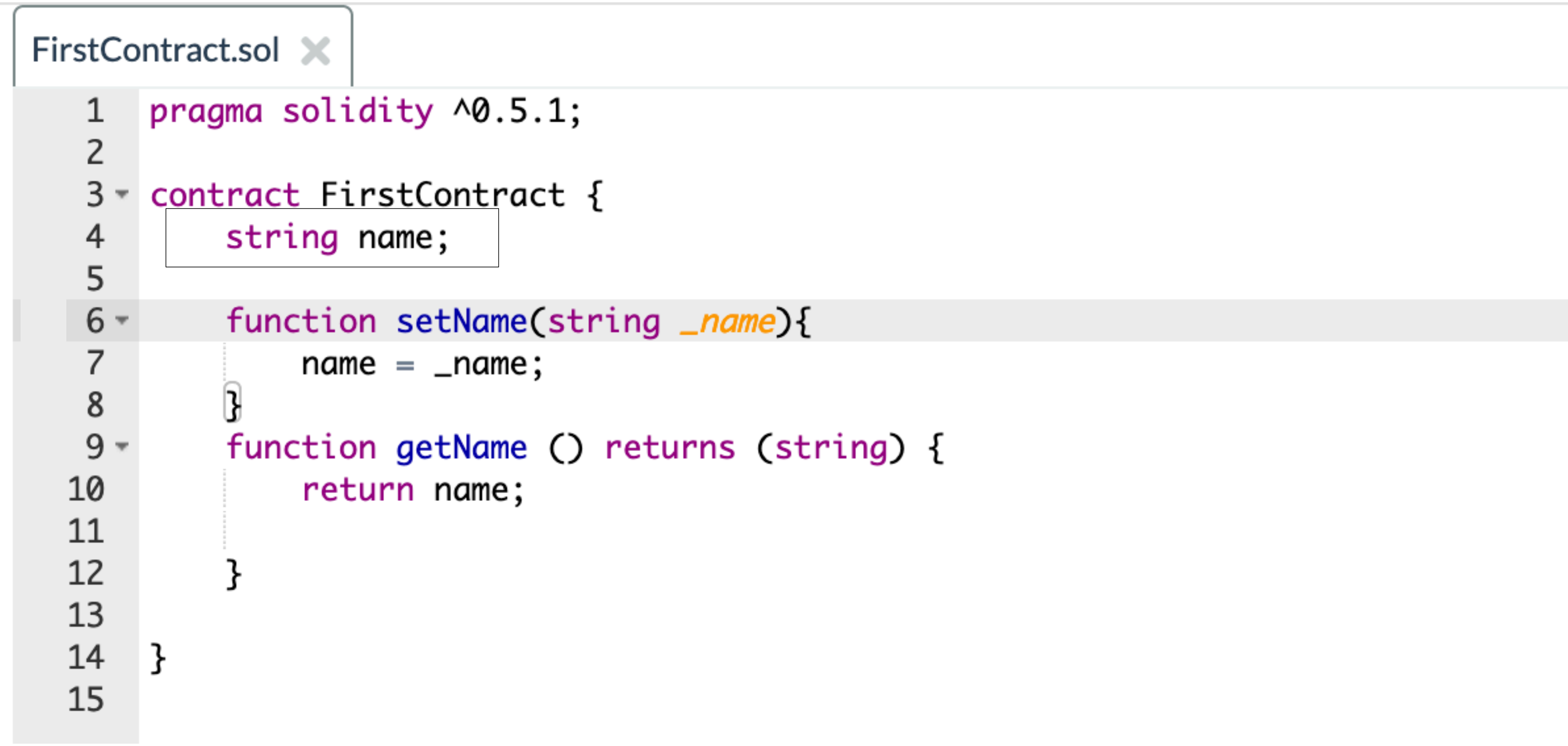
```
remix.exeCurrent(): Run the script currently displayed in the editor

remix.help(): Display this help message

remix.debugHelp(): Display help message for debugging



 - Welcome to Remix v0.8.8 -

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
```

First contract



```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName(string _name){
        name = _name;
    }
    function getName () returns (string) {
        return name;

    }

}

```

Primitive types

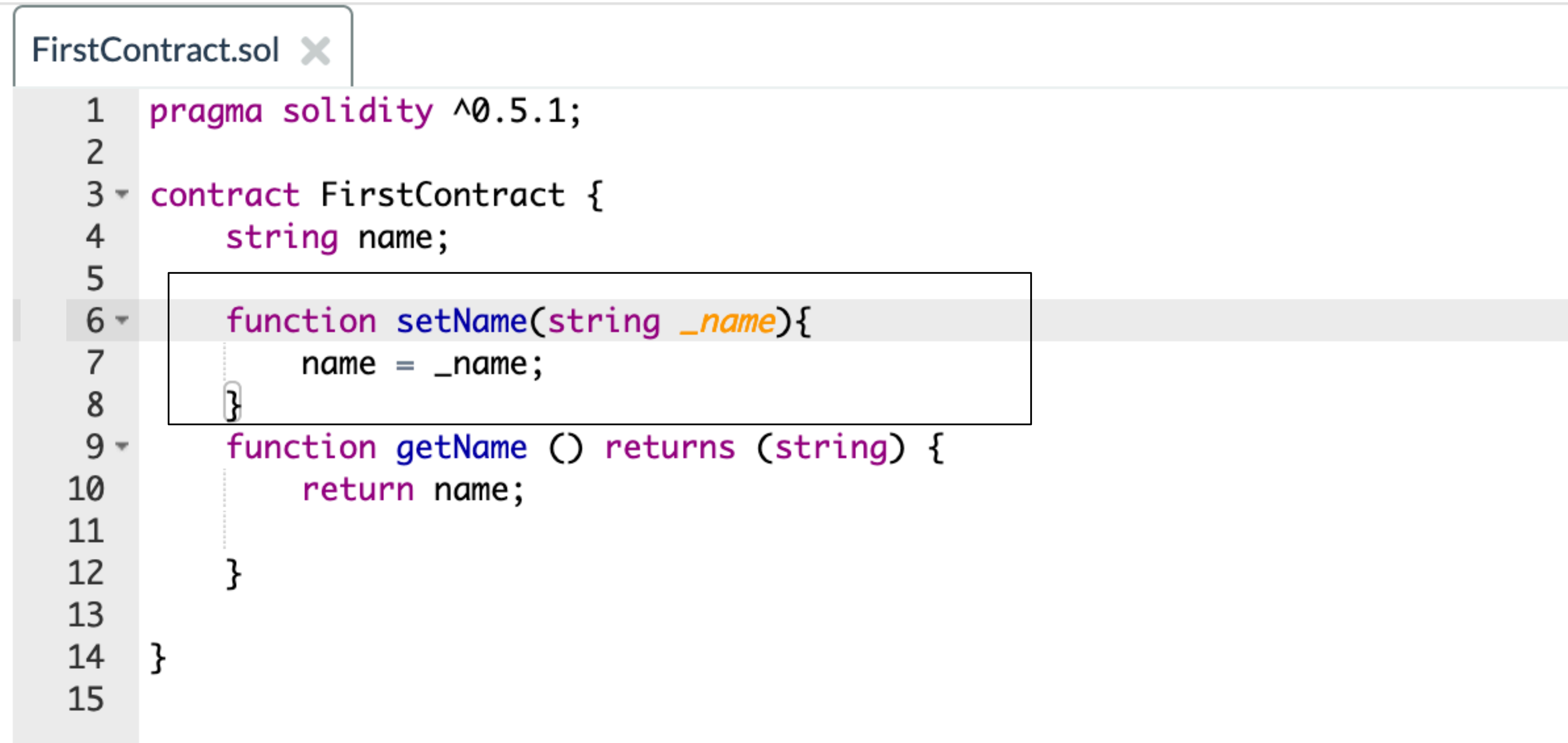


```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName(string _name){
        name = _name;
    }
    function getName () returns (string) {
        return name;

    }

}

```

Primitive types

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

Functions

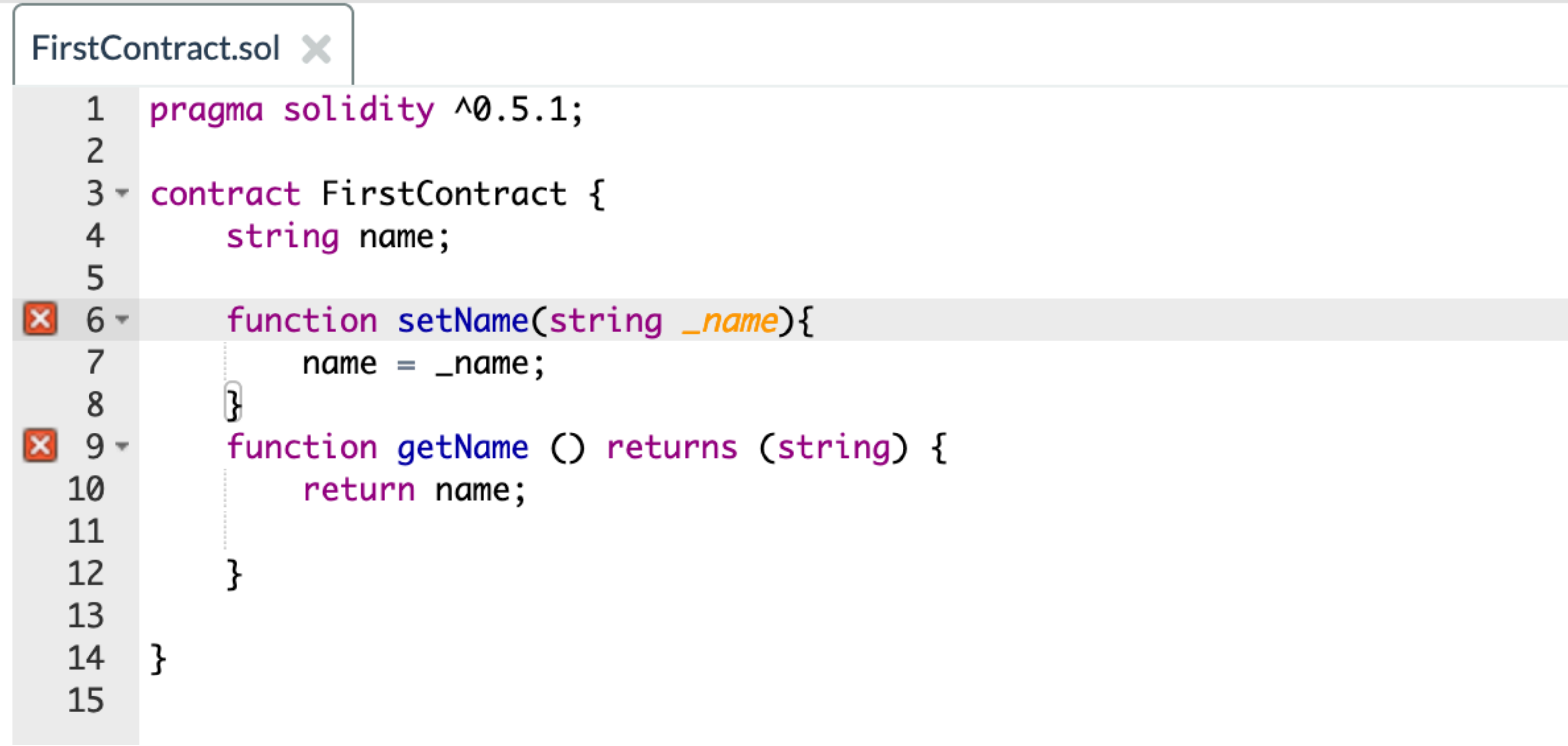


```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName(string _name){
        name = _name;
    }
    function getName () returns (string) {
        return name;

    }

}

```

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 :)

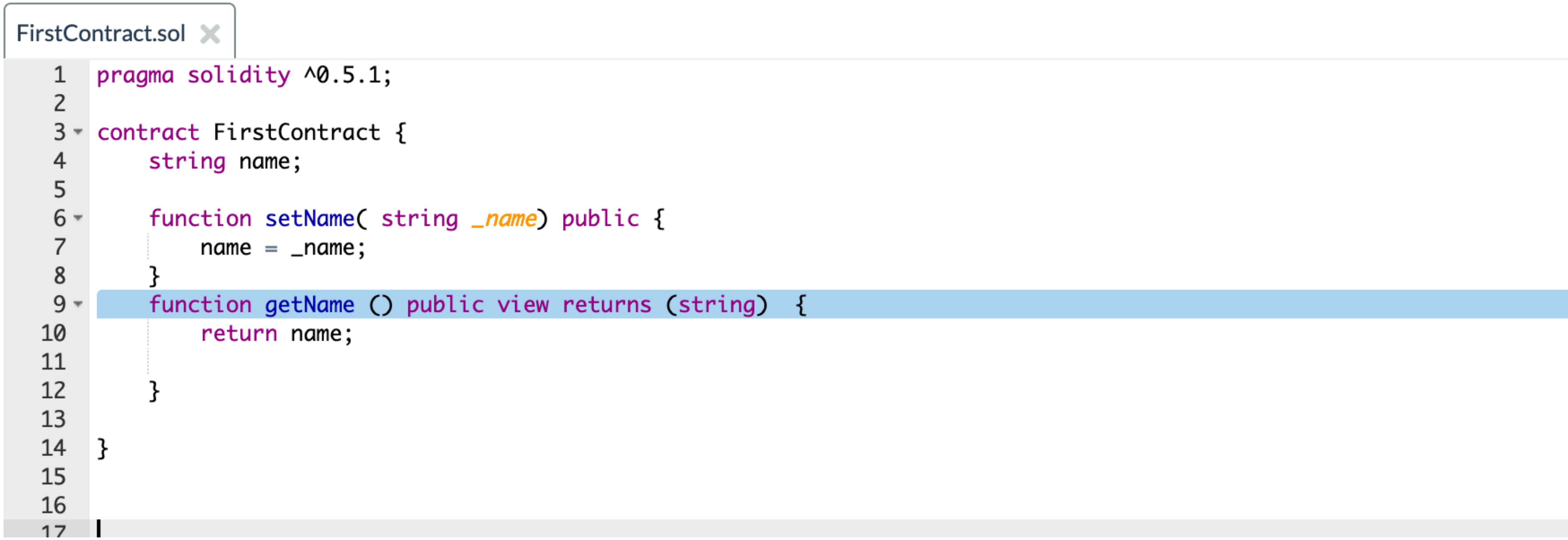


```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName(string _name){
        name = _name;
    }
    function getName () returns (string) {
        return name;

    }

}
```

Built-in method modifiers

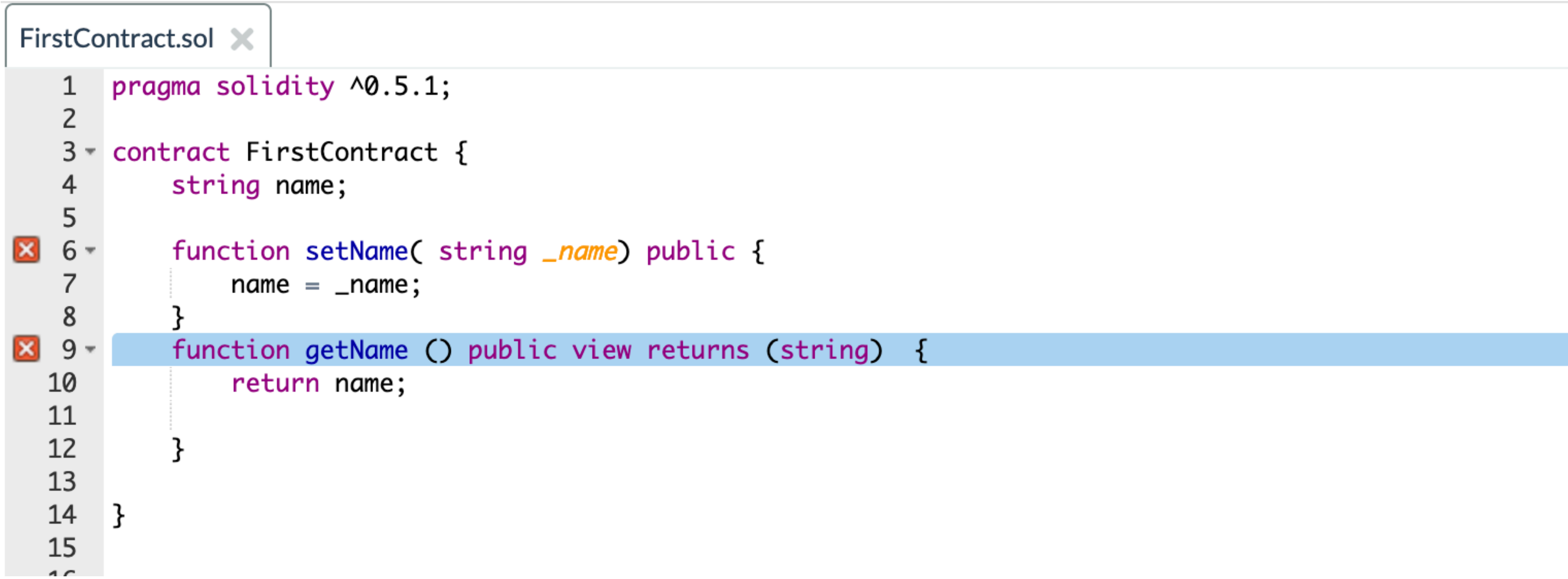
- Visibility Modifiers: private, public, internal, external
- Mutability Modifiers: view, pure
- Cryptocurrency Modifiers: payable

Built-in method modifiers



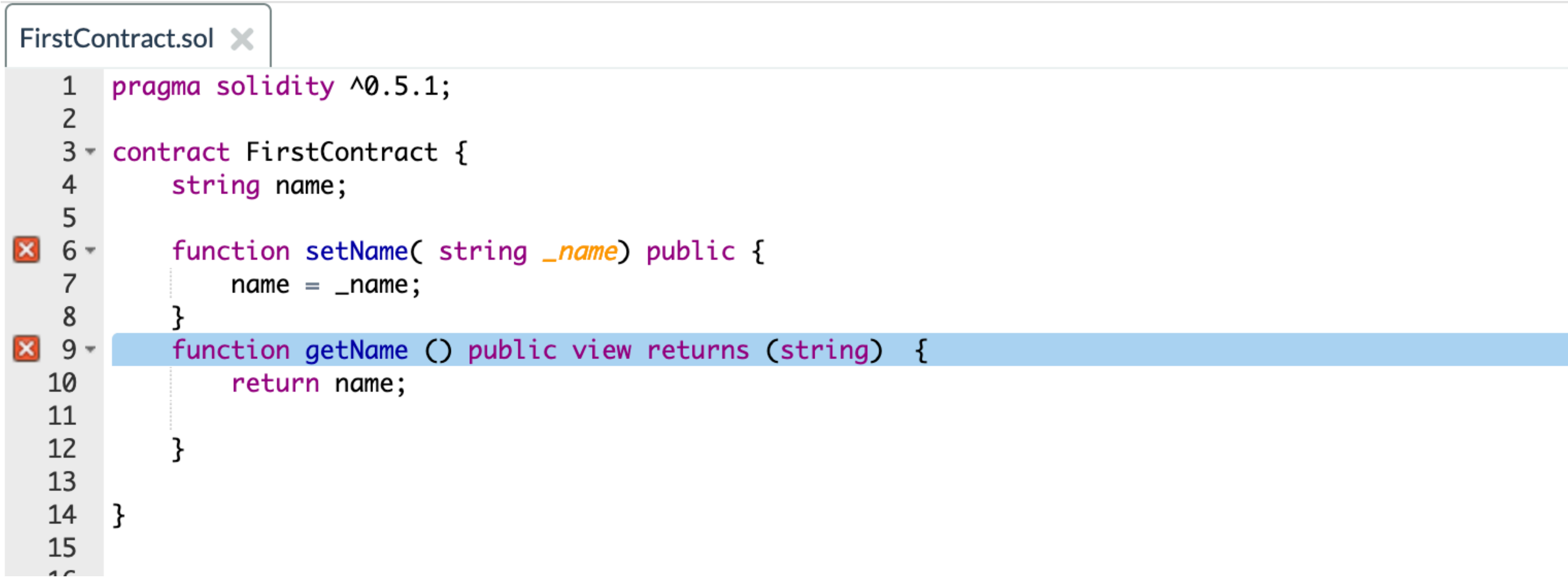
```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName( string _name) public {
        name = _name;
    }
    function getName () public view returns (string)  {
        return name;

    }

}


```

More compiling errors
:(



```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName( string _name) public {
        name = _name;
    }
    function getName () public view returns (string)  {
        return name;

    }

}

```

More compiling errors
:(

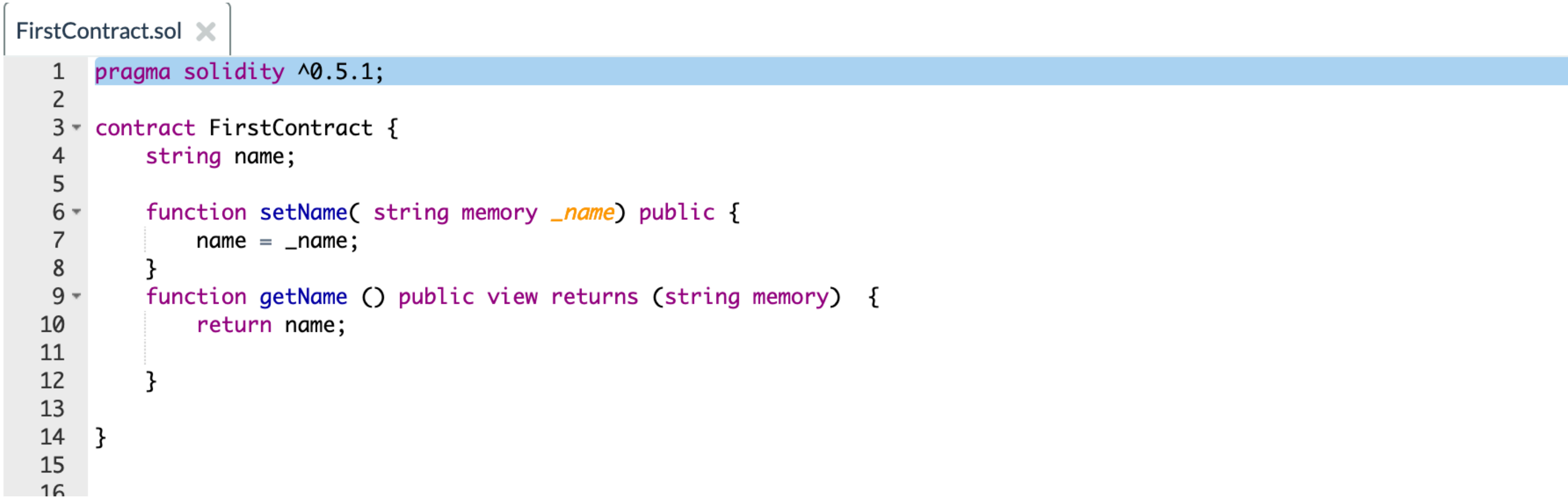


```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName( string _name) public {
        name = _name;
    }
    function getName () public view returns (string)  {
        return name;

    }

}
```

Built-in parameter modifiers

- Memory
- Storage

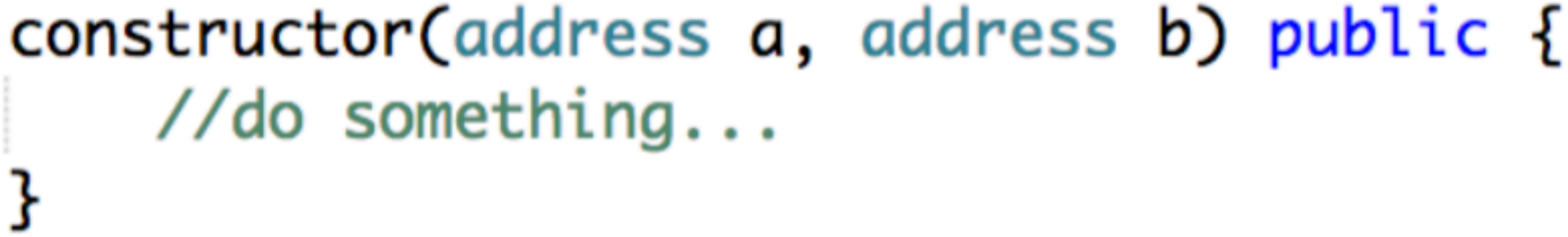
Built-in parameter modifiers



```
pragma solidity ^0.5.1;

contract FirstContract {
    string name;

    function setName( string memory _name) public {
        name = _name;
    }
    function getName () public view returns (string memory)  {
        return name;

    }

}

```

Constructor

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



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

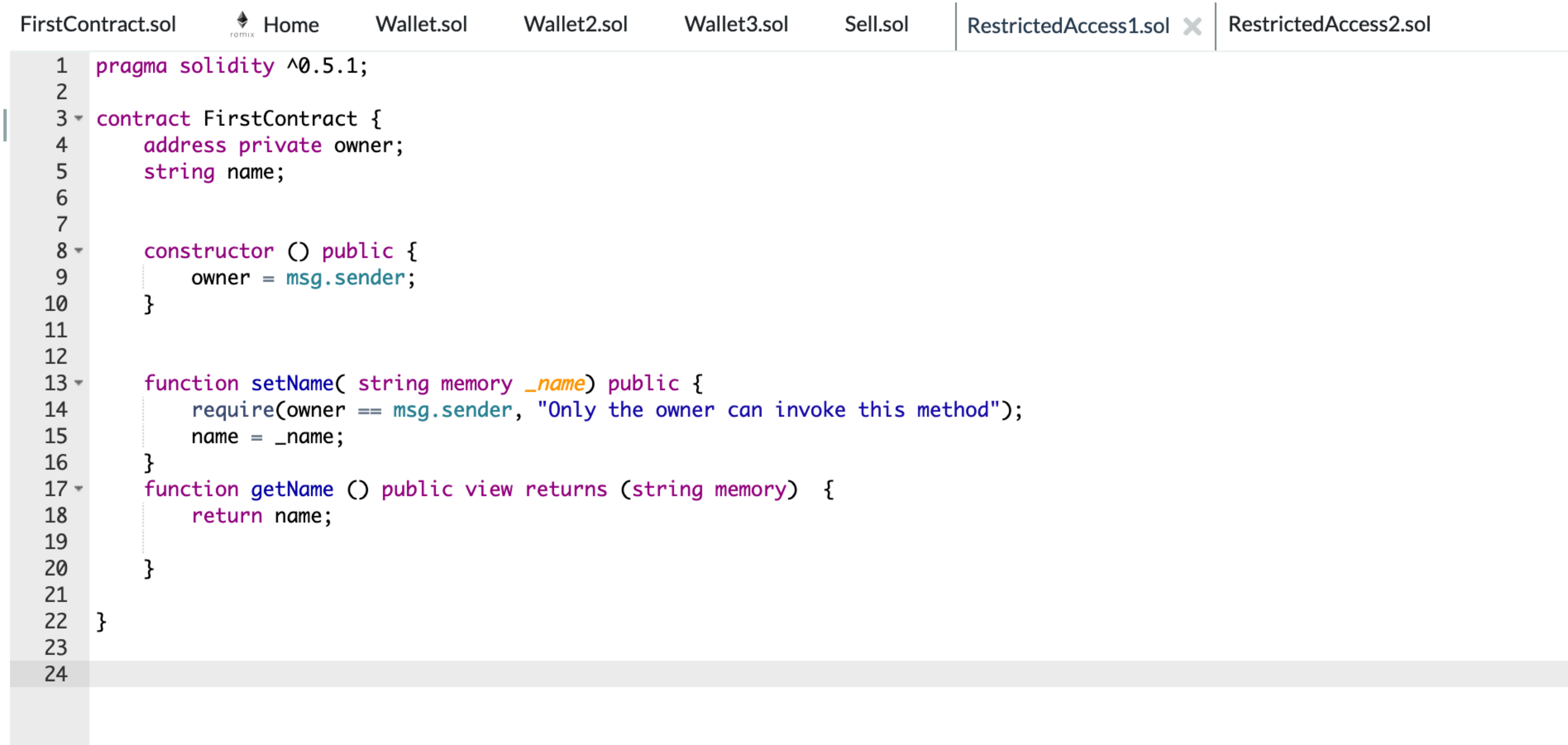
Pre-defined Variables

- 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

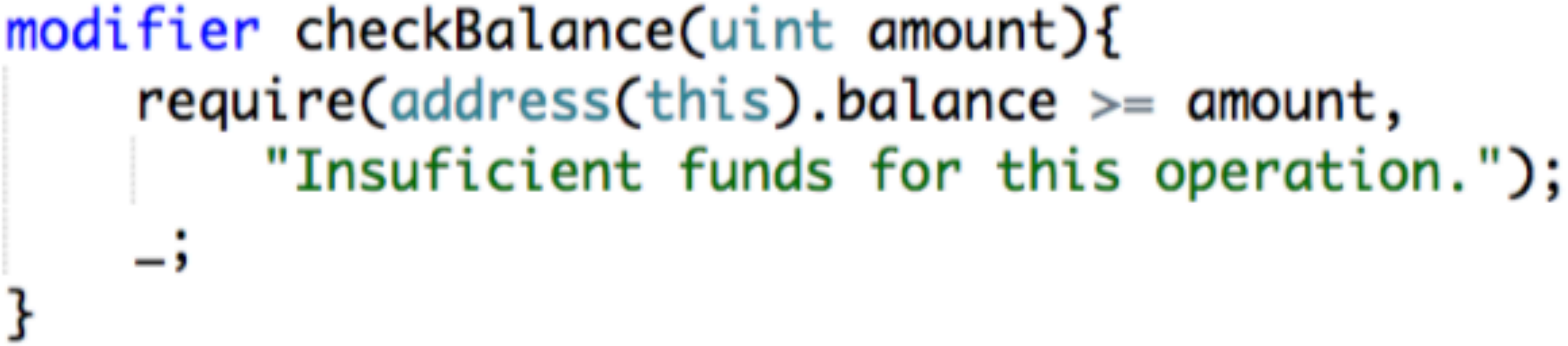
Restricted access V1



```
pragma solidity ^0.5.1;

contract FirstContract {
    address private owner;
    string name;


    constructor () public {
        owner = msg.sender;
    }


    function setName( string memory _name) public {
        require(owner == msg.sender, "Only the owner can invoke this method");
        name = _name;
    }
    function getName () public view returns (string memory)  {
        return name;

    }

}

```

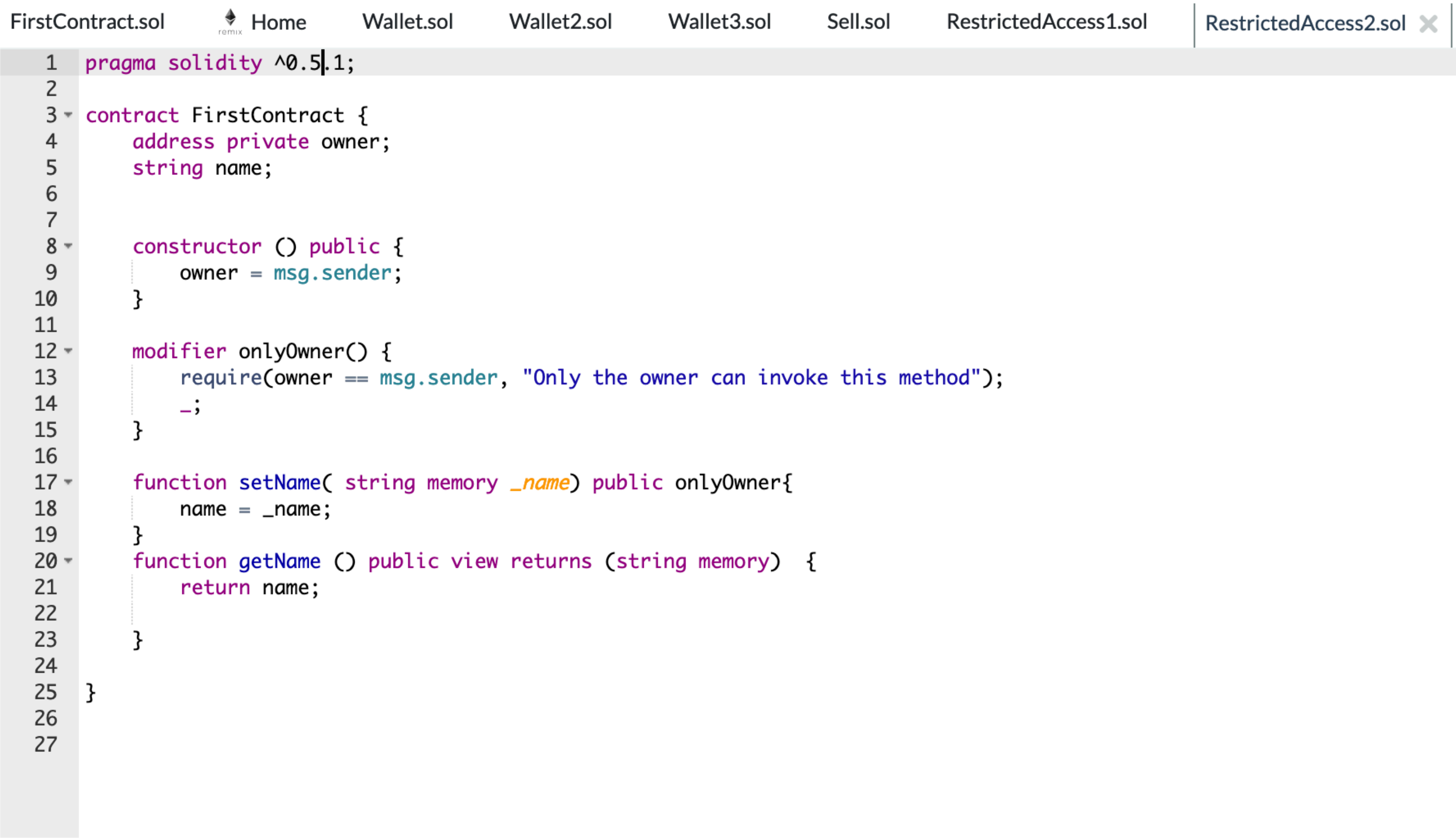
Custom Modifiers

- 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,
        "Insuficient funds for this operation.");
    _;
}
```

Restricted access V2

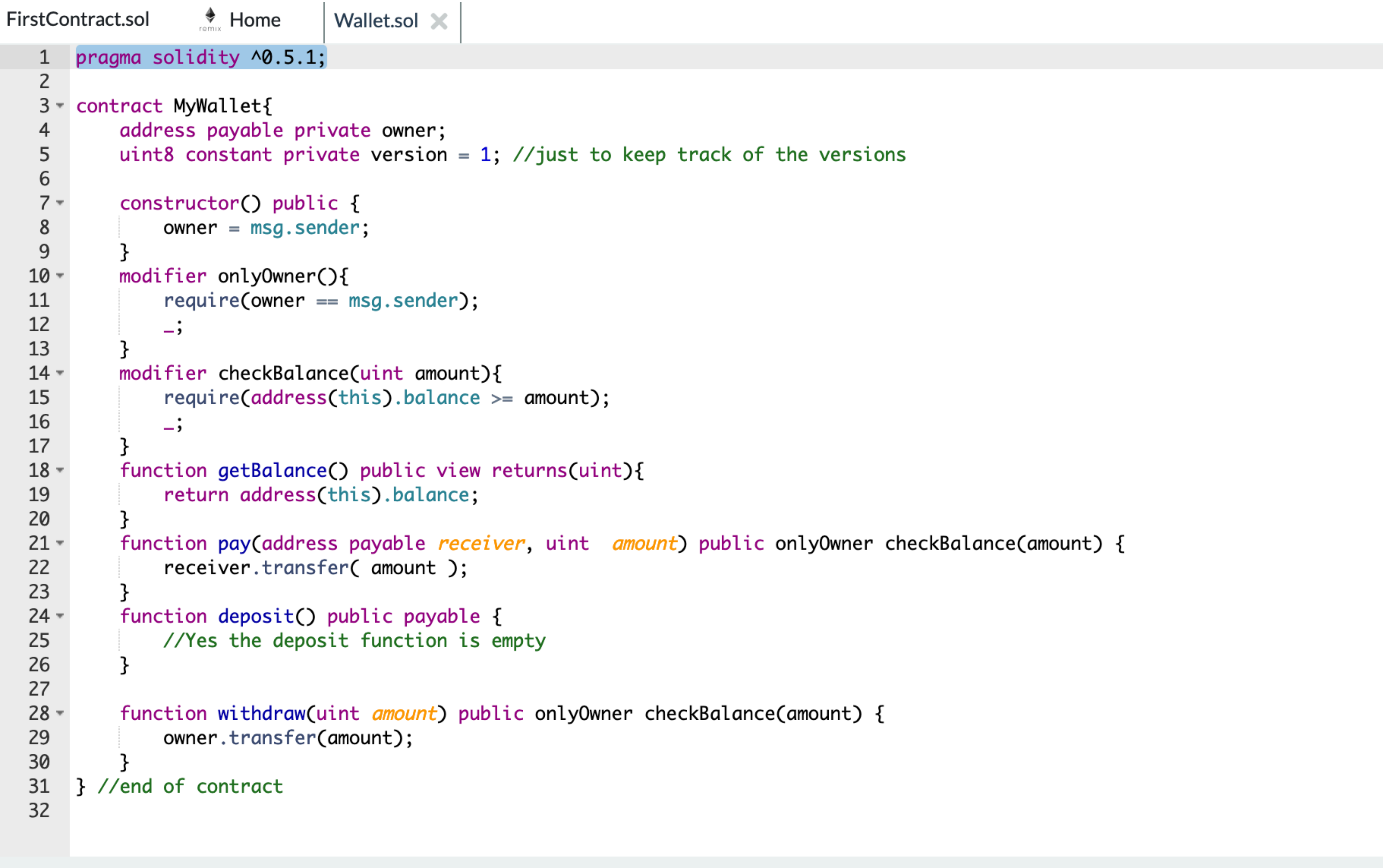


```
pragma solidity ^0.5.1;

contract FirstContract {
    address private owner;
    string name;


    constructor () public {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(owner == msg.sender, "Only the owner can invoke this method");
        _;
    }

    function setName( string memory _name) public onlyOwner{
        name = _name;
    }
    function getName () public view returns (string memory)  {
        return name;

    }

}


```

Dealing with money

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

Wallet v1

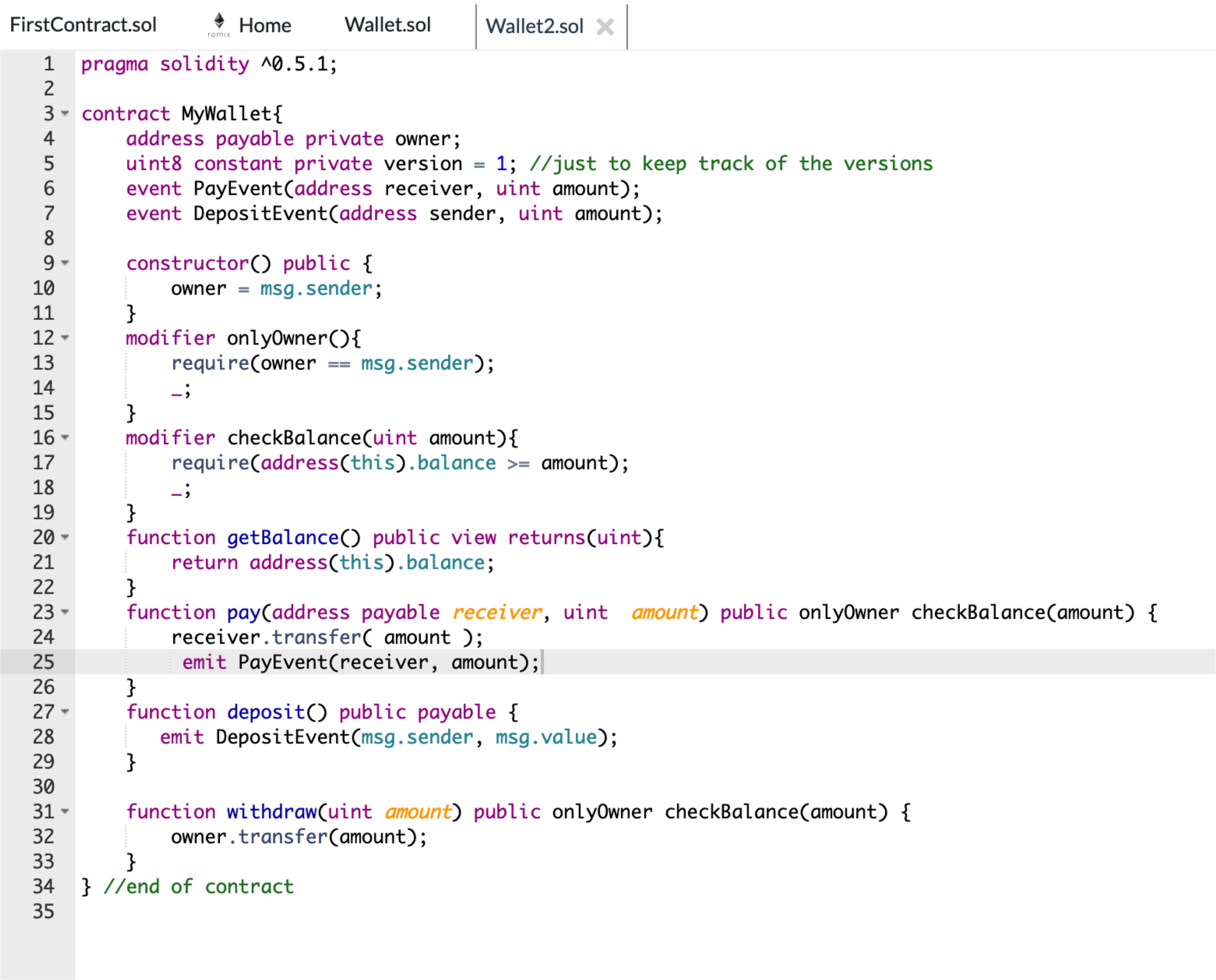


```
pragma solidity ^0.5.1;

contract MyWallet{
    address payable private owner;
    uint8 constant private version = 1; //just to keep track of the versions

    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner(){
        require(owner == msg.sender);
        _;
    }
    modifier checkBalance(uint amount){
        require(address(this).balance >= amount);
        _;
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
    function pay(address payable receiver, uint  amount) public onlyOwner checkBalance(amount) {
        receiver.transfer( amount );
    }
    function deposit() public payable {
        //Yes the deposit function is empty
    }

    function withdraw(uint amount) public onlyOwner checkBalance(amount) {
        owner.transfer(amount);
    }
} //end of contract

```

Events

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

Wallet v2

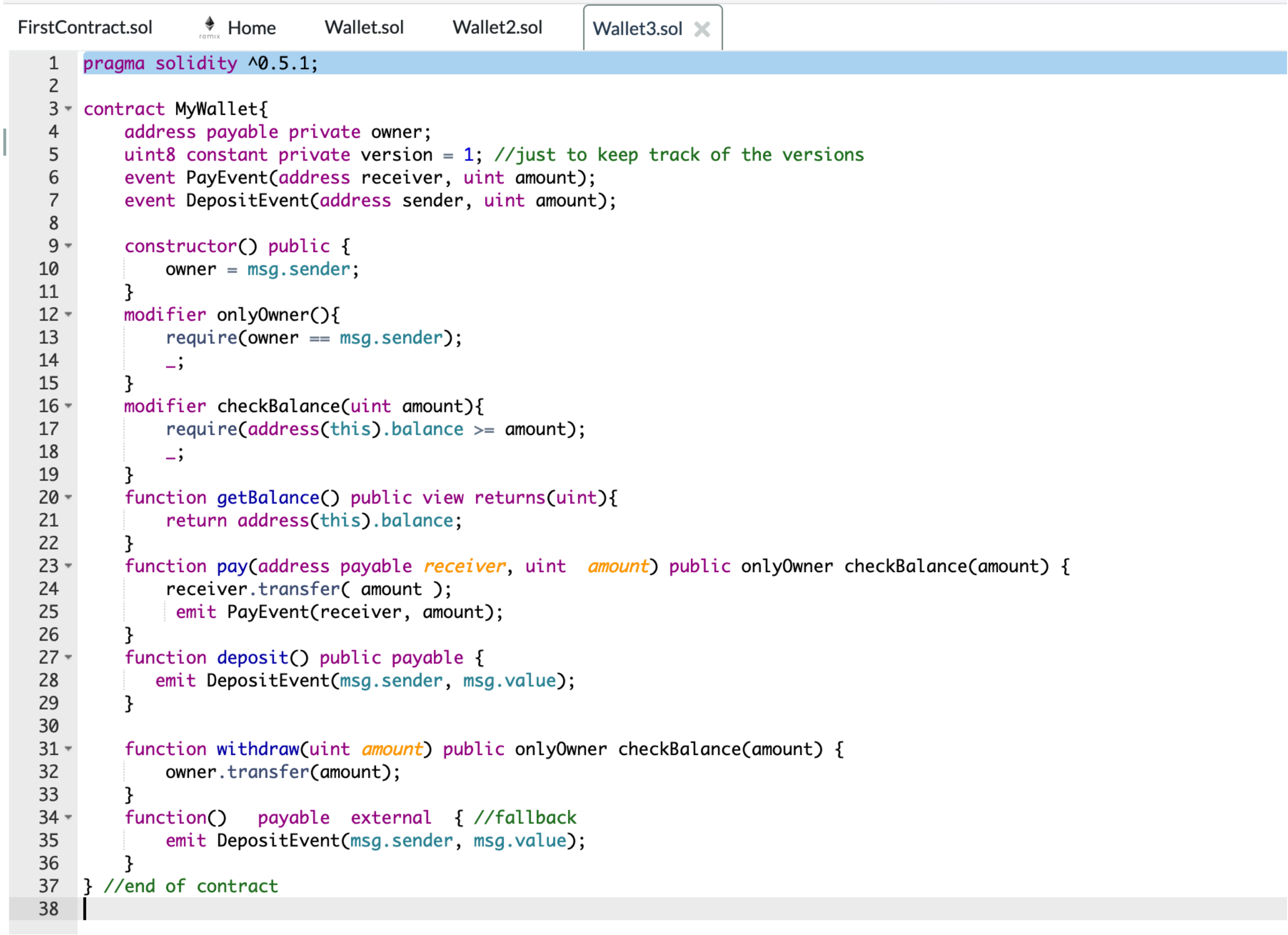


```solidity
pragma solidity ^0.5.1;

contract MyWallet{
    address payable private owner;
    uint8 constant private version = 1; //just to keep track of the versions
    event PayEvent(address receiver, uint amount);
    event DepositEvent(address sender, uint amount);

    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner(){
        require(owner == msg.sender);
        _;
    }
    modifier checkBalance(uint amount){
        require(address(this).balance >= amount);
        _;
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
    function pay(address payable receiver, uint  amount) public onlyOwner checkBalance(amount) {
        receiver.transfer( amount );
         emit PayEvent(receiver, amount);
    }
    function deposit() public payable {
       emit DepositEvent(msg.sender, msg.value);
    }

    function withdraw(uint amount) public onlyOwner checkBalance(amount) {
        owner.transfer(amount);
    }
} //end of contract

```

Fallback Function

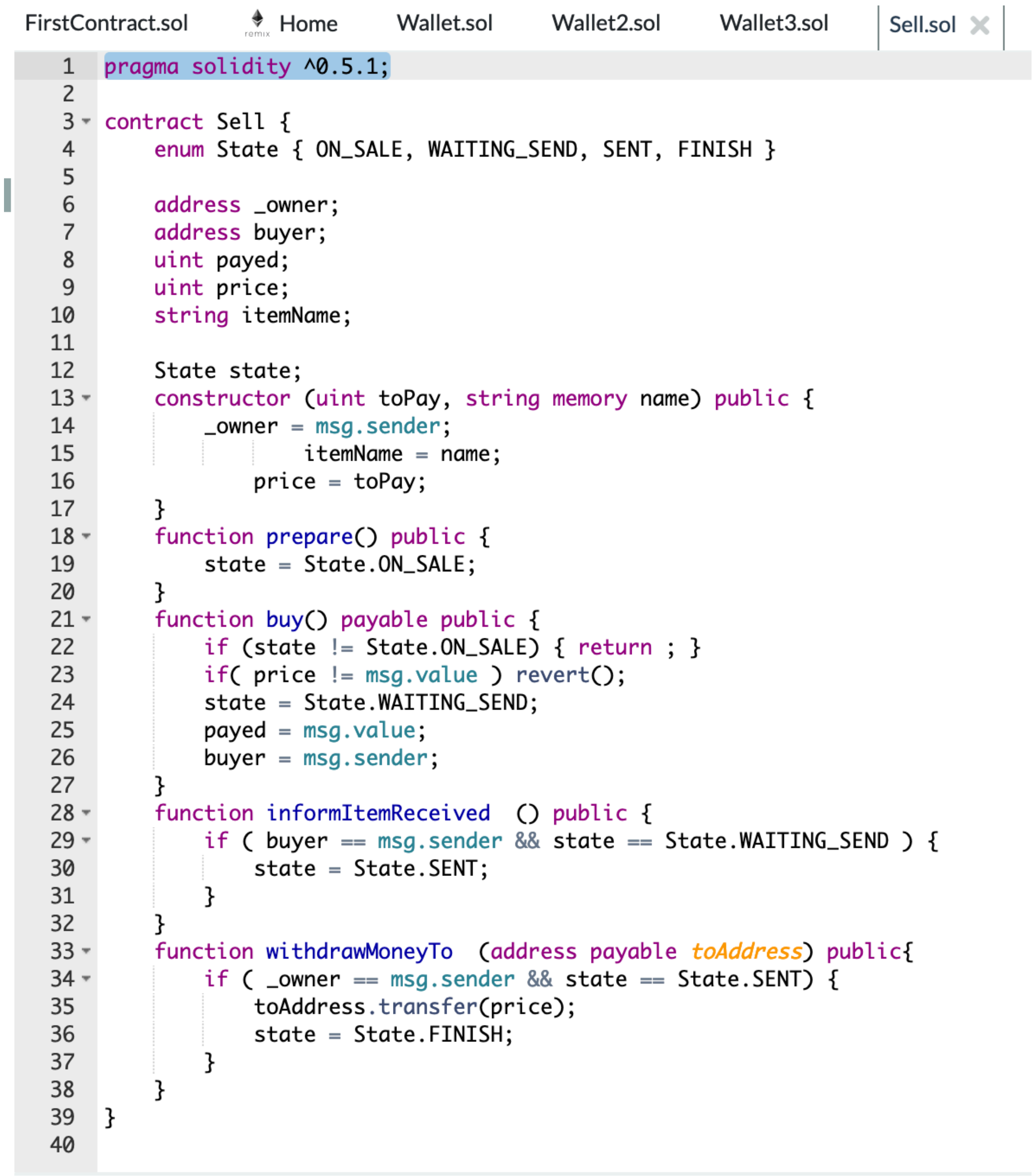
- Un-named function
- Cannot have any parameter, cannot return any value
- must be external
- could be payable
- Executed when a method name is not found
- Executed (if payable) when transferring money to this contract

Wallet v3



```
pragma solidity ^0.5.1;

contract MyWallet{
    address payable private owner;
    uint8 constant private version = 1; //just to keep track of the versions
    event PayEvent(address receiver, uint amount);
    event DepositEvent(address sender, uint amount);

    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner(){
        require(owner == msg.sender);
        _;
    }
    modifier checkBalance(uint amount){
        require(address(this).balance >= amount);
        _;
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
    function pay(address payable receiver, uint  amount) public onlyOwner checkBalance(amount) {
        receiver.transfer( amount );
         emit PayEvent(receiver, amount);
    }
    function deposit() public payable {
       emit DepositEvent(msg.sender, msg.value);
    }

    function withdraw(uint amount) public onlyOwner checkBalance(amount) {
        owner.transfer(amount);
    }
    function()   payable  external  { //fallback
        emit DepositEvent(msg.sender, msg.value);
    }
} //end of contract

```

State machine



```
pragma solidity ^0.5.1;

contract Sell {
    enum State { ON_SALE, WAITING_SEND, SENT, FINISH }

    address _owner;
    address buyer;
    uint payed;
    uint price;
    string itemName;

    State state;
    constructor (uint toPay, string memory name) public {
        _owner = msg.sender;
                itemName = name;
            price = toPay;
    }
    function prepare() public {
        state = State.ON_SALE;
    }
    function buy() payable public {
        if (state != State.ON_SALE) { return ; }
        if( price != msg.value ) revert();
        state = State.WAITING_SEND;
        payed = msg.value;
        buyer = msg.sender;
    }
    function informItemReceived  () public {
        if ( buyer == msg.sender && state == State.WAITING_SEND ) {
            state = State.SENT;
        }
    }
    function withdrawMoneyTo  (address payable toAddress) public{
        if ( _owner == msg.sender && state == State.SENT) {
            toAddress.transfer(price);
            state = State.FINISH;
        }
    }
}

```

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?