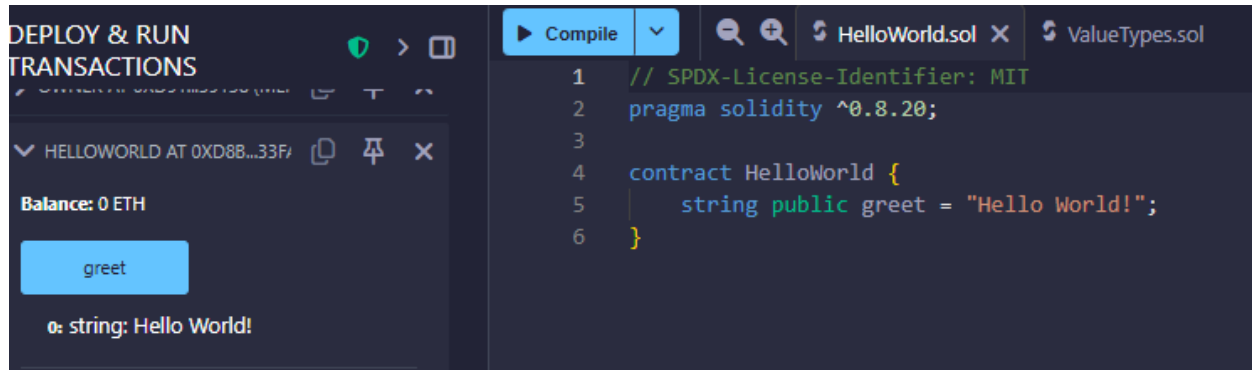
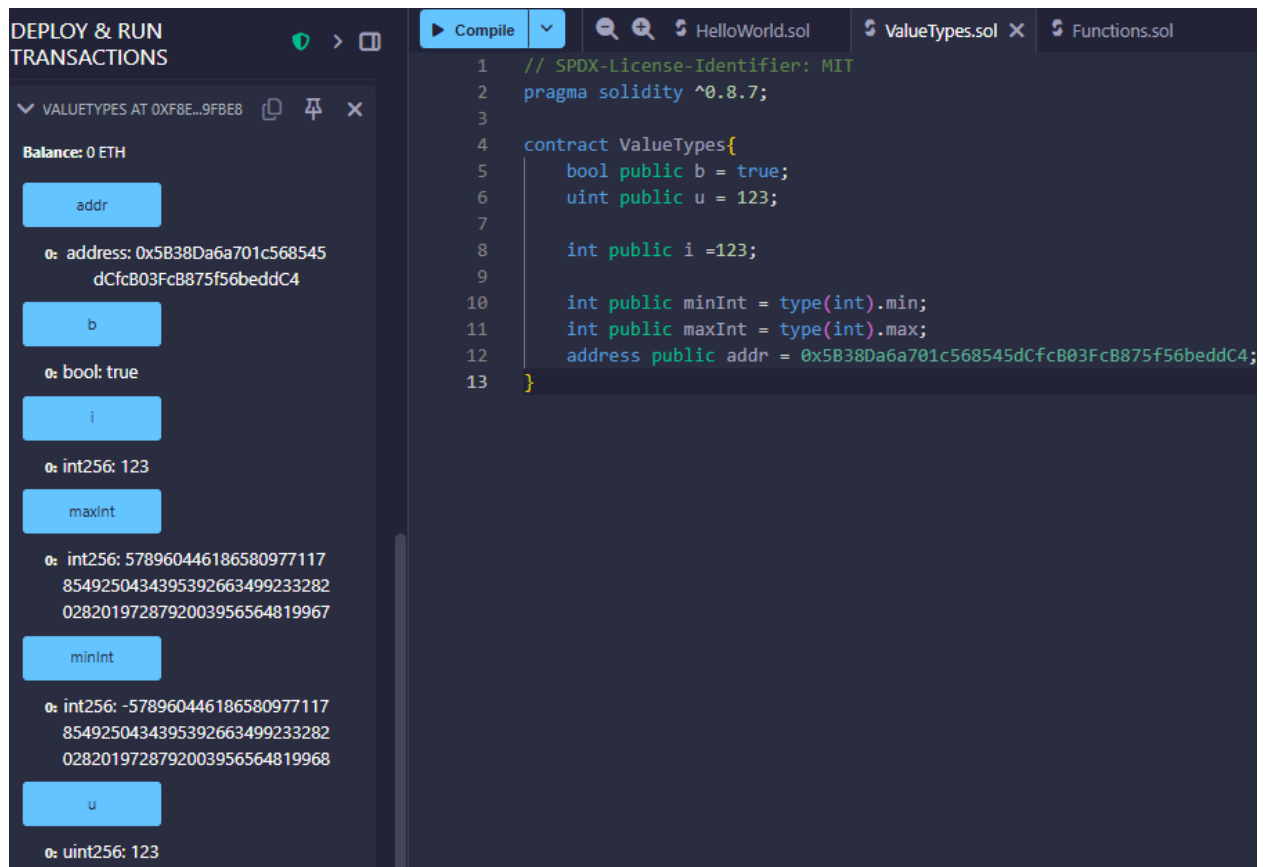


## HelloWorld.sol



It defines a public string variable `greet` that is set to "Hello World!". The contract is deployed and the `greet` function can be called to return the string "Hello World!".

## ValueTypes.sol



This solidity contract defines different data types. It has a boolean b, an unsigned integer u, and an integer i, as well as the minimum and maximum values of an integer type (minInt and maxInt). Additionally, an Ethereum address addr is also defined in the contract.

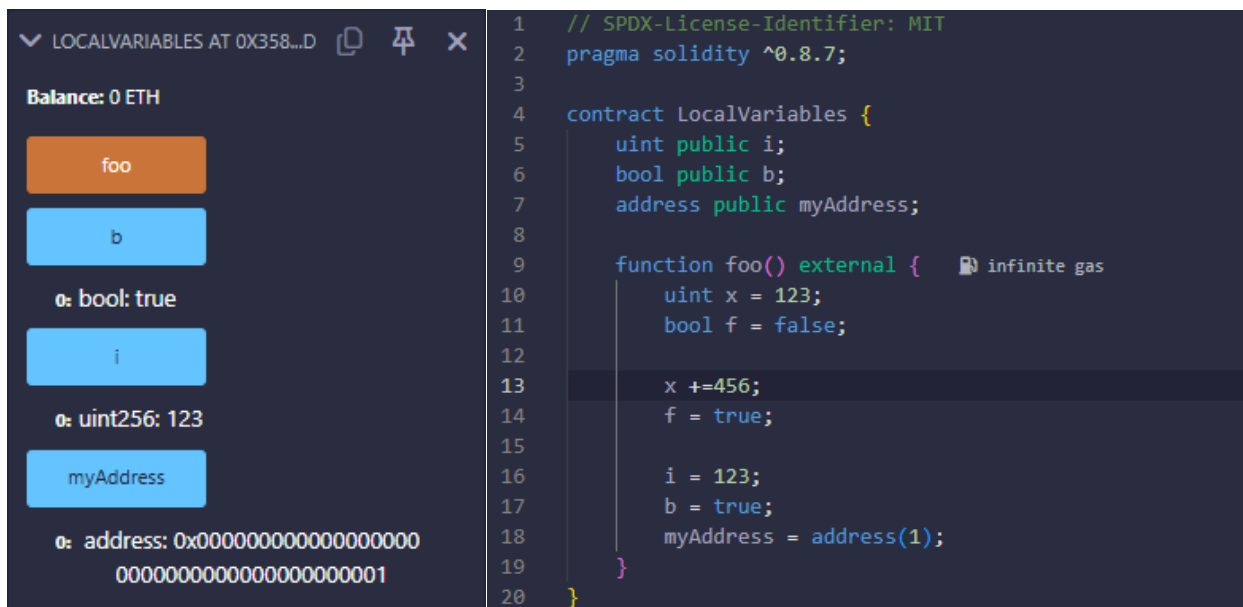
## Functions.sol



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4 contract FunctionIntro{
5     function add(uint x, uint y) external pure returns (uint) {
6         return (x + y);
7     }
8
9     function sub (uint x, uint y) external pure returns (uint) {
10        return (x - y);
11    }
12 }
```

This contract uses functions to perform addition and subtraction. The values of x and y are based on the input of the sender. The result of the addition (add) is 10, and the result of the subtraction (sub) is -2.

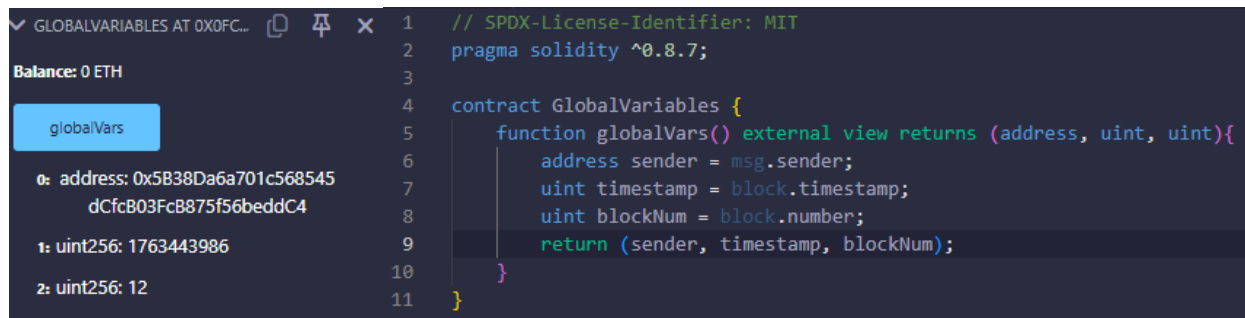
## LocalVariables.sol



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 contract LocalVariables {
5     uint public i;
6     bool public b;
7     address public myAddress;
8
9     function foo() external {
10         uint x = 123;
11         bool f = false;
12
13         x +=456;
14         f = true;
15
16         i = 123;
17         b = true;
18         myAddress = address(1);
19     }
20 }
```

This solidity contract defines variables of different data types (uint, Boolean, and an address). It has a function foo that assigns value to the variables when called.

## GlobalVariables.sol



The screenshot shows a Solidity IDE with a contract named `GlobalVariables`. The contract code is as follows:

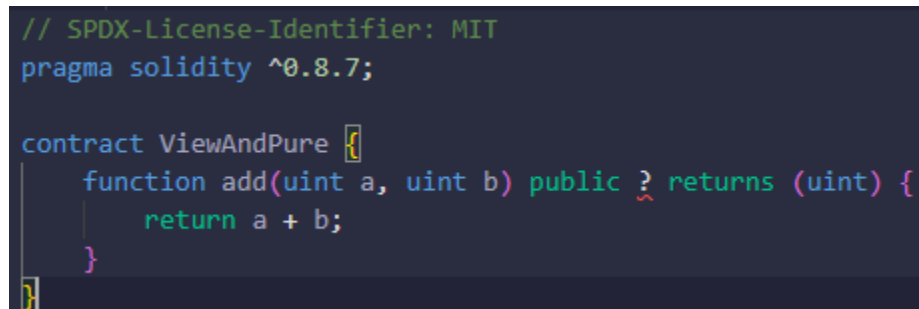
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 contract GlobalVariables {
5     function globalVars() external view returns (address, uint, uint){
6         address sender = msg.sender;
7         uint timestamp = block.timestamp;
8         uint blockNum = block.number;
9         return (sender, timestamp, blockNum);
10    }
11 }
```

On the left, the state of the contract is displayed:

- Balance: 0 ETH
- globalVars (selected)
- 0: address: 0x5B38Da6a701c568545dCfcB03Fc8875f56beddC4
- 1: uint256: 1763443986
- 2: uint256: 12

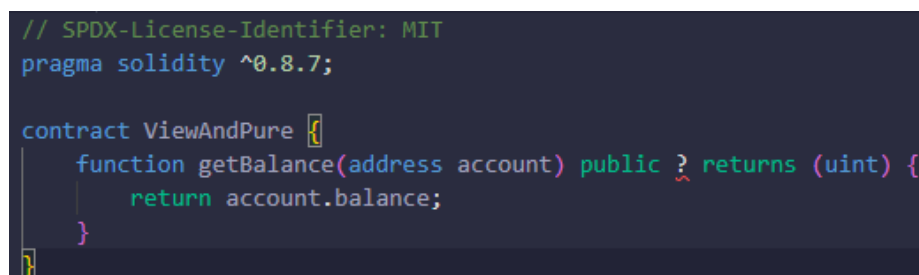
This contract reads three global values from the blockchain. It gathers the caller address, current block timestamp, and block number. It returns these values because the function is marked `view` and does not change state.

## Review Questions



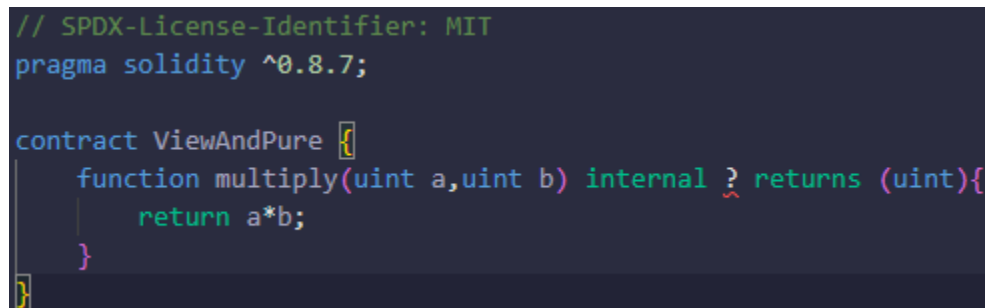
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract ViewAndPure {
    function add(uint a, uint b) public returns (uint) {
        return a + b;
    }
}
```



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract ViewAndPure {
    function getBalance(address account) public returns (uint) {
        return account.balance;
    }
}
```



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract ViewAndPure {
    function multiply(uint a, uint b) internal returns (uint) {
        return a * b;
    }
}
```

1. The function `add` is pure because it only uses the inputted values. It does not read contract storage.
2. The second function `getBalance` is view because it reads the `balance` field from the blockchain and it does not change any state variables.
3. The third function `multiple` works only using the provided parameters and does not interfere with contract storage.