

HelloWorld.sol

The screenshot shows the Truffle IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays a deployed contract named 'HELLOWORLD AT 0xD88...33F'. It shows a balance of 0 ETH and a 'greet' function call result of '0: string: Hello World!'. On the right, the code editor shows the Solidity source code for the HelloWorld contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract HelloWorld {
    string public greet = "Hello World!";
}
```

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

The screenshot shows the Truffle IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays a deployed contract named 'VALUETYPES AT 0xF8E...9FBE8'. It shows a balance of 0 ETH and several function calls: 'addr' returns '0x5B38Da6a701c568545dCfcB03FcB875f56beddC4'; 'b' returns '0: bool: true'; 'i' returns '0: int256: 123'; 'maxInt' returns '0: int256: 578960446186580977117 8549250434395392663499233282 0282019728792003956564819967'; and 'minInt' returns '0: int256: -578960446186580977117 8549250434395392663499233282 0282019728792003956564819968'. On the right, the code editor shows the Solidity source code for the ValueTypes contract:

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

contract ValueTypes{
    bool public b = true;
    uint public u = 123;

    int public i =123;

    int public minInt = type(int).min;
    int public maxInt = type(int).max;
    address public addr = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
}
```

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



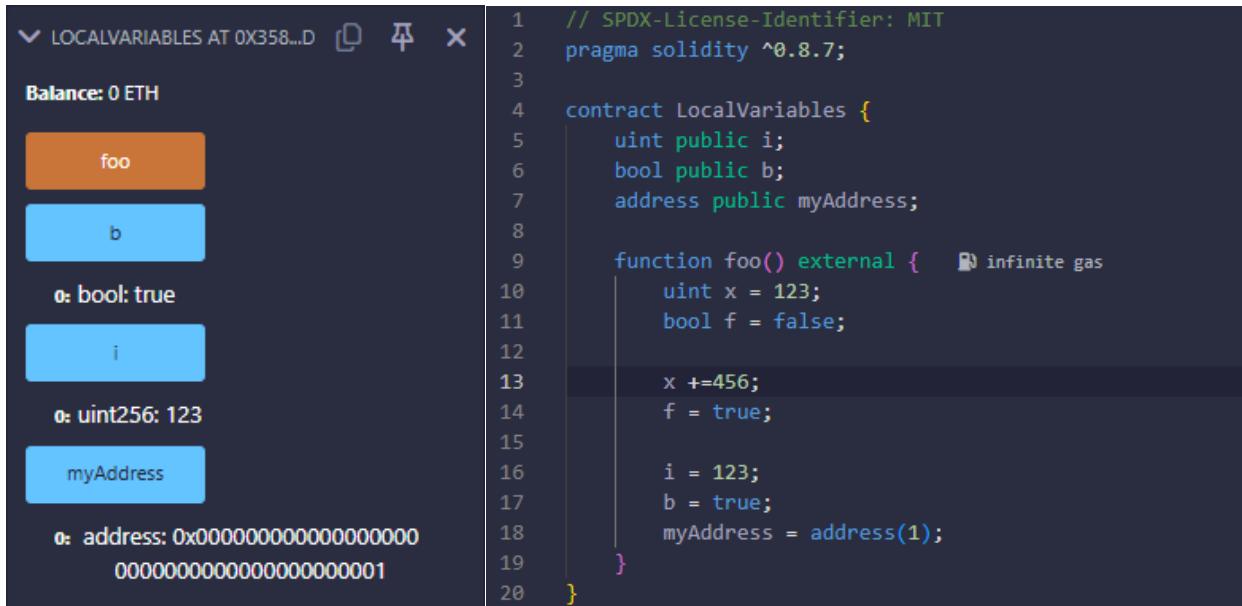
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract FunctionIntro{
    function add(uint x, uint y) external pure returns (uint) {
        return (x + y);
    }

    function sub (uint x, uint y) external pure returns (uint) {
        return (x - y);
    }
}
```

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



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

contract LocalVariables {
    uint public i;
    bool public b;
    address public myAddress;

    function foo() external { infinite gas
        uint x = 123;
        bool f = false;

        x +=456;
        f = true;

        i = 123;
        b = true;
        myAddress = address(1);
    }
}
```

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

```
GLOBALVARIABLES AT 0x0FC... 1 // SPDX-License-Identifier: MIT
Balance: 0 ETH
globalVars
0: address: 0x5B38Da6a701c568545
    dCfcB03FcB875f56beddC4
1: uint256: 1763443986
2: uint256: 12
pragma solidity ^0.8.7;

contract GlobalVariables {
    function globalVars() external view returns (address, uint, uint){
        address sender = msg.sender;
        uint timestamp = block.timestamp;
        uint blockNum = block.number;
        return (sender, timestamp, blockNum);
    }
}
```

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 pure returns (uint) {
        return a + b;
    }
}
```

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

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

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

contract ViewAndPure {
    function multiply(uint a,uint b) internal pure 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.