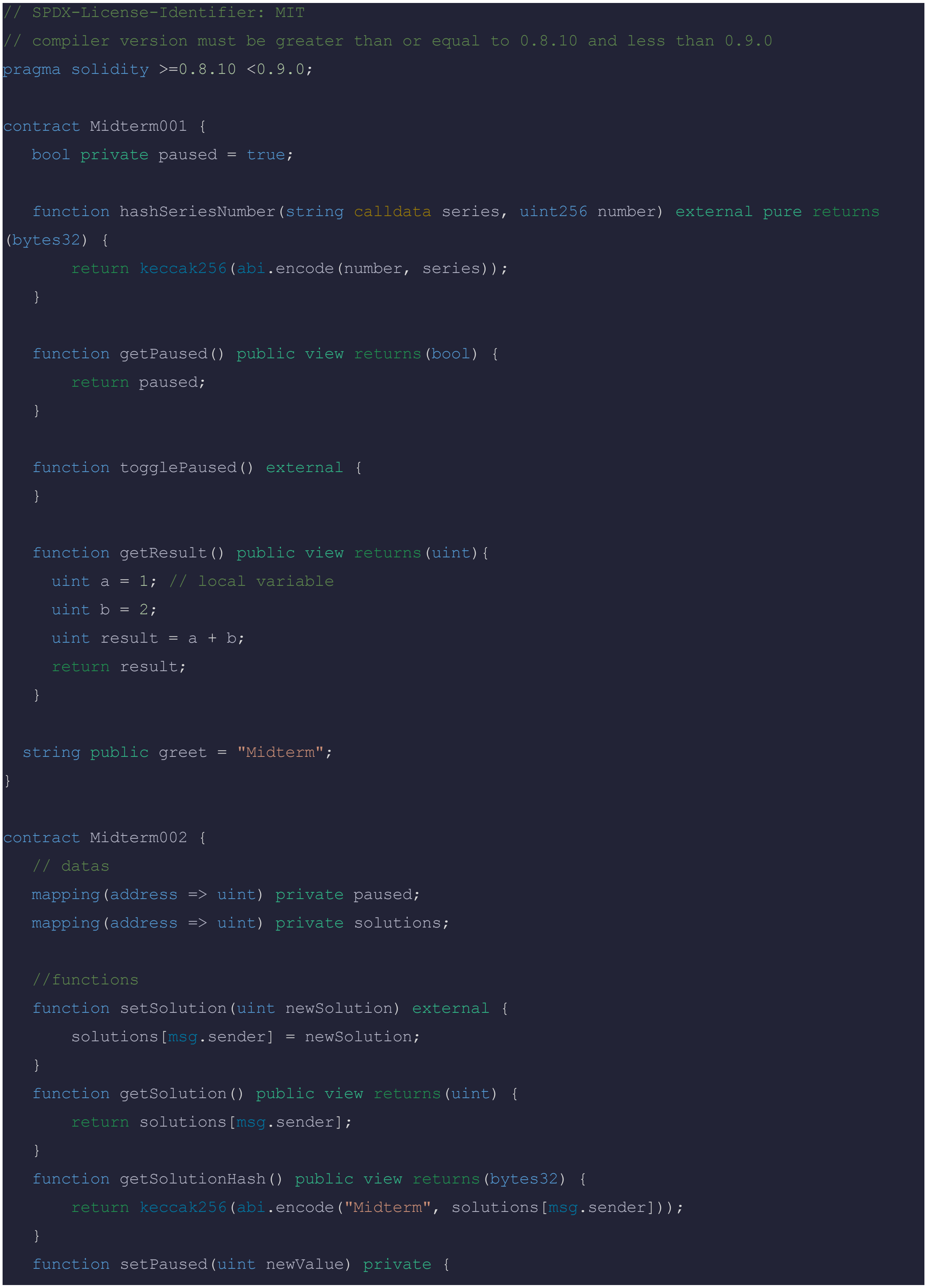
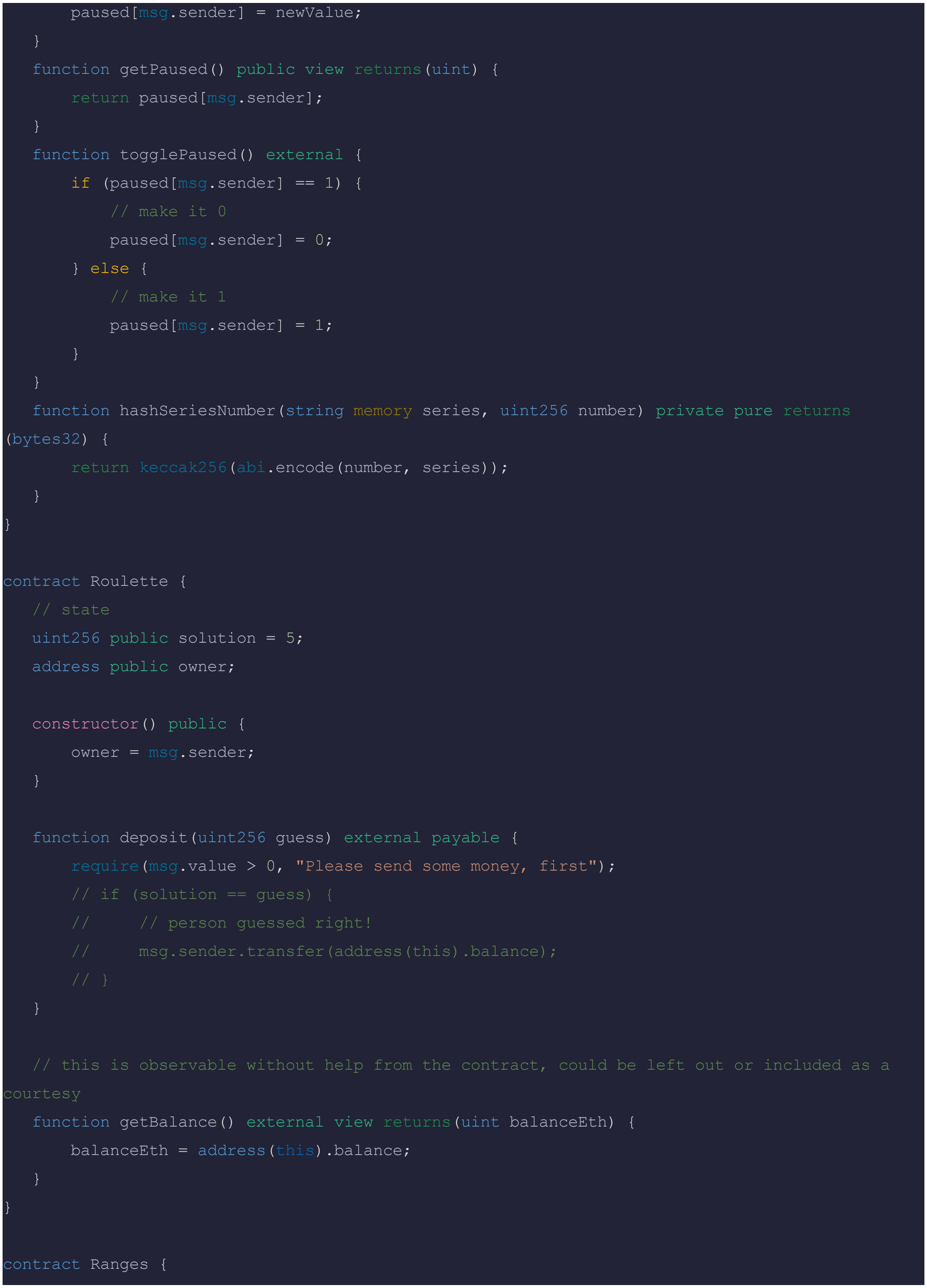
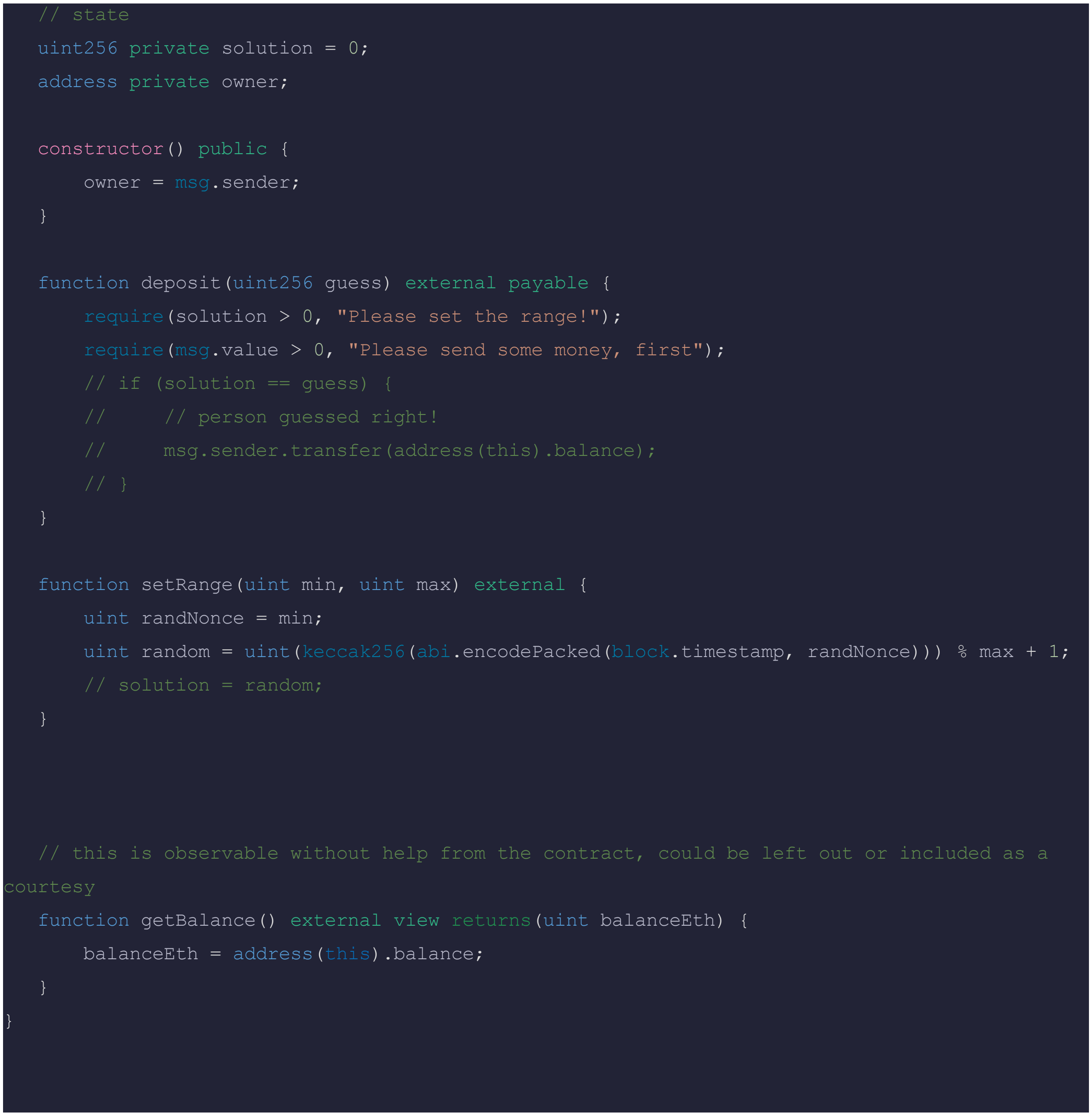


```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.10 and less than 0.9.0
pragma solidity >=0.8.10 <0.9.0;

contract Midterm001 {
   bool private paused = true;

   function hashSeriesNumber(string calldata series, uint256 number) external pure returns (bytes32) {
       return keccak256(abi.encode(number, series));
   }

   function getPaused() public view returns(bool) {
       return paused;
   }

   function togglePaused() external {
   }

   function getResult() public view returns(uint){
     uint a = 1; // local variable
     uint b = 2;
     uint result = a + b;
     return result;
   }

  string public greet = "Midterm";
}

contract Midterm002 {
   // datas
   mapping(address => uint) private paused;
   mapping(address => uint) private solutions;

   //functions
   function setSolution(uint newSolution) external {
       solutions[msg.sender] = newSolution;
   }
   function getSolution() public view returns(uint) {
       return solutions[msg.sender];
   }
   function getSolutionHash() public view returns(bytes32) {
       return keccak256(abi.encode("Midterm", solutions[msg.sender]));
   }
   function setPaused(uint newValue) private {
```



```
        paused[msg.sender] = newValue;
    }
    function getPaused() public view returns(uint) {
        return paused[msg.sender];
    }
    function togglePaused() external {
        if (paused[msg.sender] == 1) {
            // make it 0
            paused[msg.sender] = 0;
        } else {
            // make it 1
            paused[msg.sender] = 1;
        }
    }
    function hashSeriesNumber(string memory series, uint256 number) private pure returns (bytes32) {
        return keccak256(abi.encode(number, series));
    }
}

contract Roulette {
    // state
    uint256 public solution = 5;
    address public owner;

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

    function deposit(uint256 guess) external payable {
        require(msg.value > 0, "Please send some money, first");
        // if (solution == guess) {
        //     // person guessed right!
        //     msg.sender.transfer(address(this).balance);
        // }
    }

    // this is observable without help from the contract, could be left out or included as a courtesy
    function getBalance() external view returns(uint balanceEth) {
        balanceEth = address(this).balance;
    }
}

contract Ranges {
```



```
    // state
    uint256 private solution = 0;
    address private owner;

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

    function deposit(uint256 guess) external payable {
        require(solution > 0, "Please set the range!");
        require(msg.value > 0, "Please send some money, first");
        // if (solution == guess) {
        //     // person guessed right!
        //     msg.sender.transfer(address(this).balance);
        // }
    }

    function setRange(uint min, uint max) external {
        uint randNonce = min;
        uint random = uint(keccak256(abi.encodePacked(block.timestamp, randNonce))) % max + 1;
        // solution = random;
    }



    // this is observable without help from the contract, could be left out or included as a courtesy
    function getBalance() external view returns(uint balanceEth) {
        balanceEth = address(this).balance;
    }
}
```