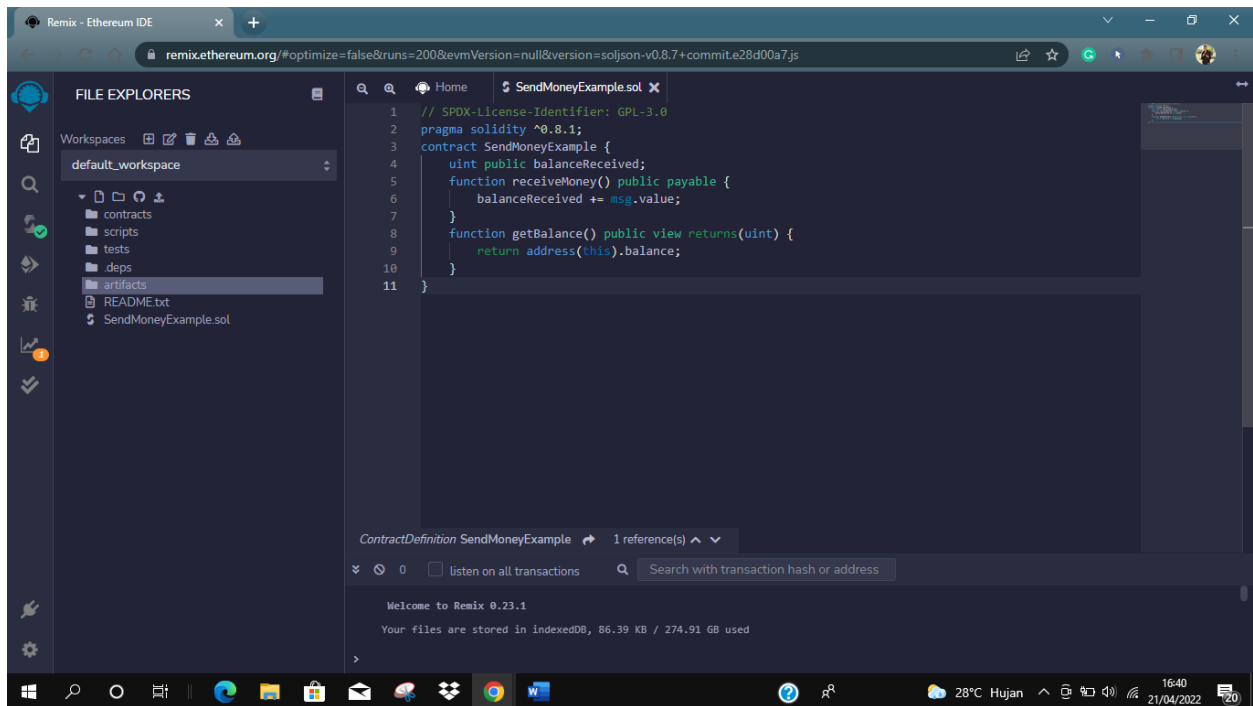
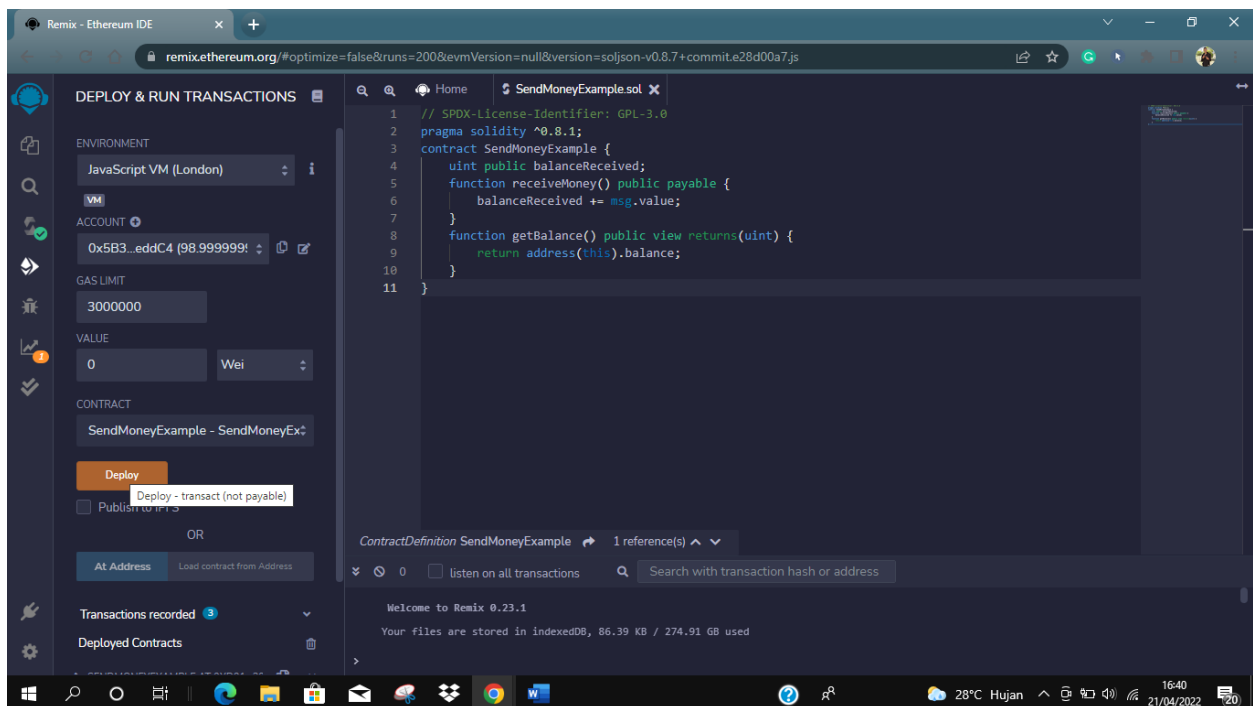
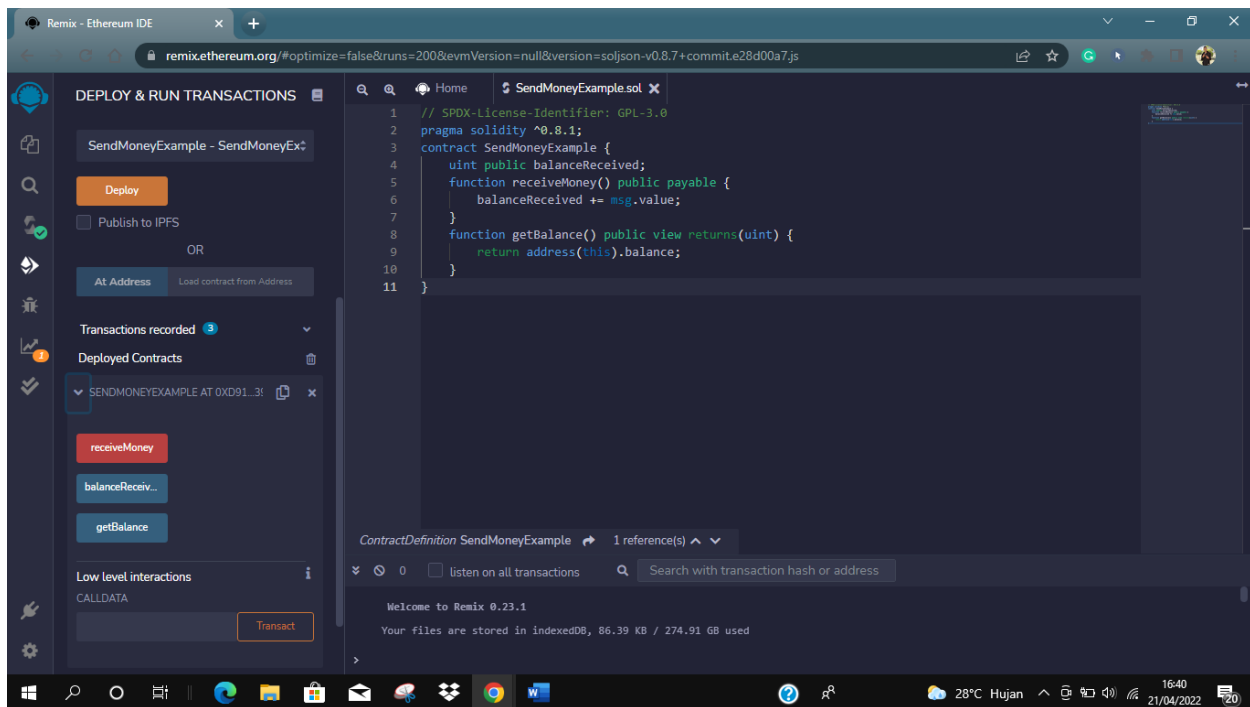


Let's start with a simple Smart Contract. Create a new file in Remix

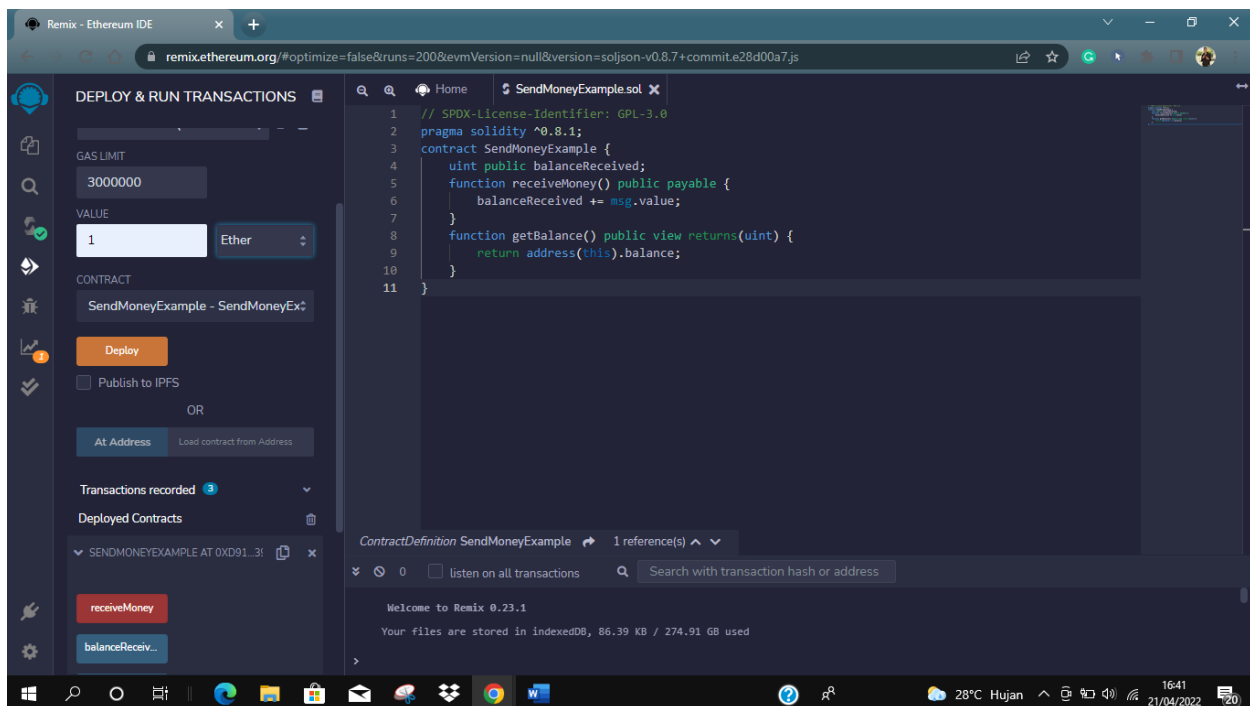


Head over to the Deploy and Run Transactions Plugin and deploy the Smart Contract into the JavaScript VM

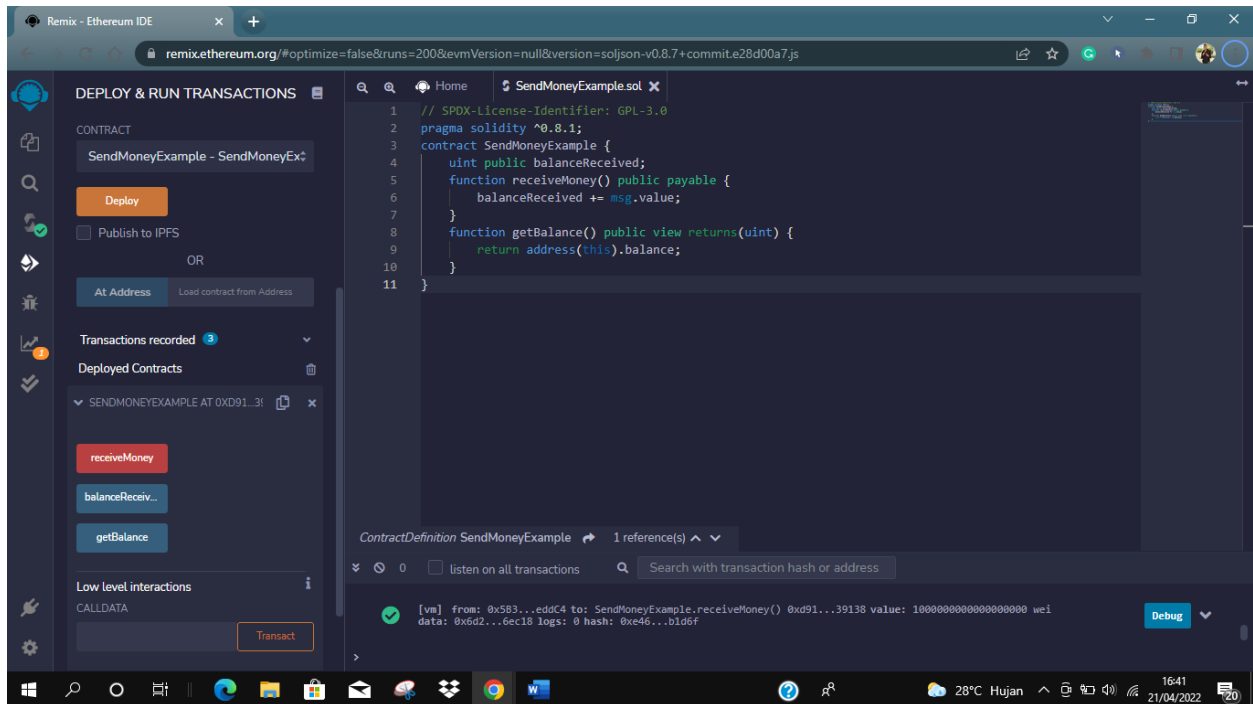




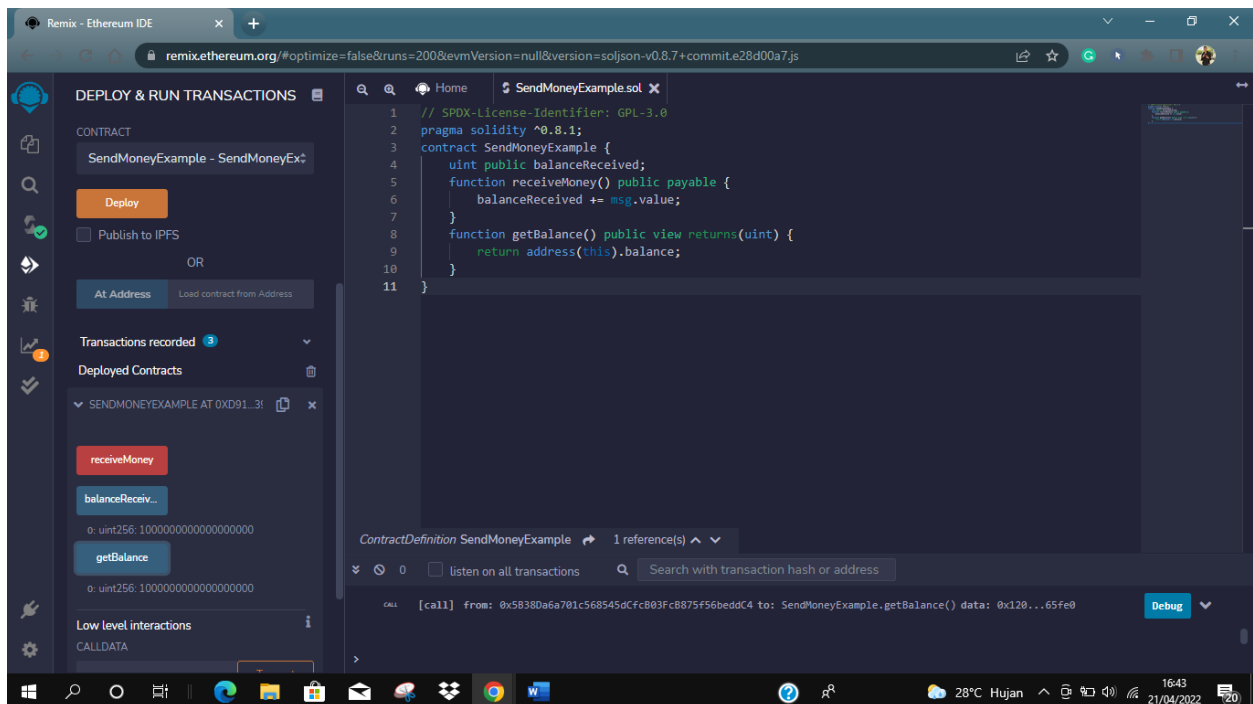
Scroll up to the "value" field and put "1" into the value input field and select "ether" from the dropdown. Then scroll down to the Smart Contract and hit the red "receiveMoney" button:



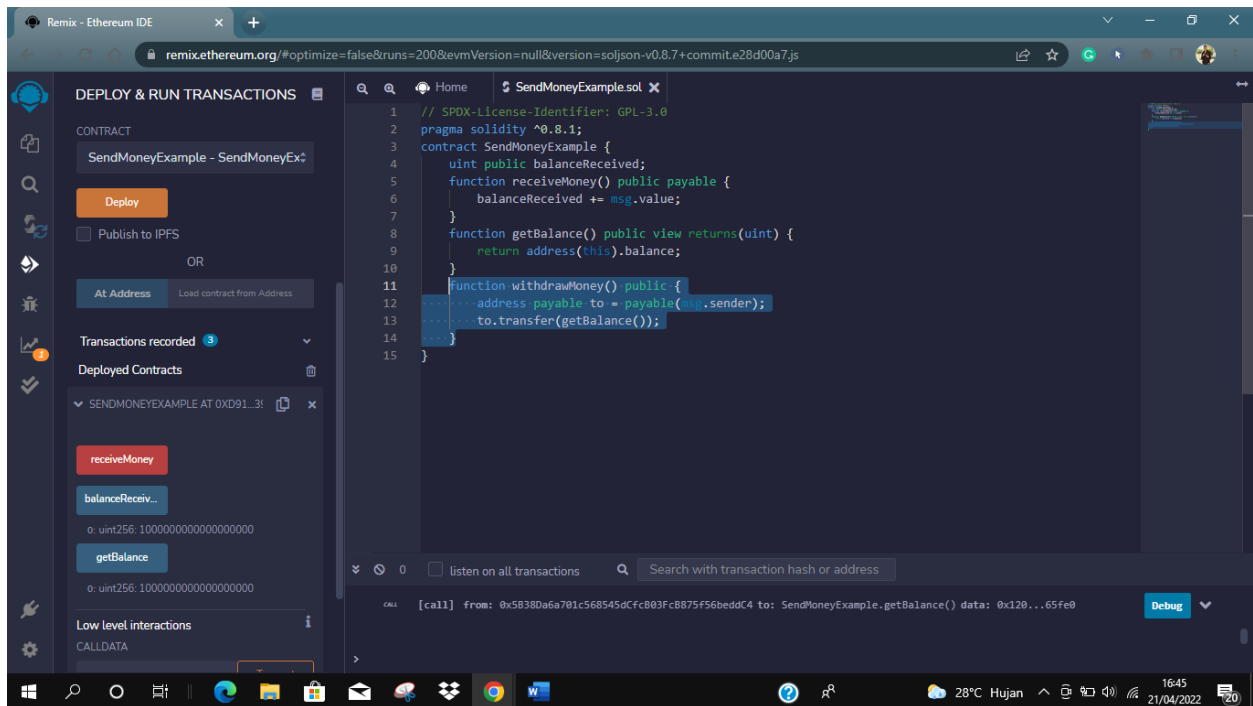
Also observe the terminal, see that there was a new transaction sent to "the network" (although just a simulation in the



Now we sent 1 Ether, or 10^{18} Wei, to the Smart Contract. According to our code the variable balanceReceived and the function getBalance() should have the same value.



Let's add the following function to the Smart Contract:

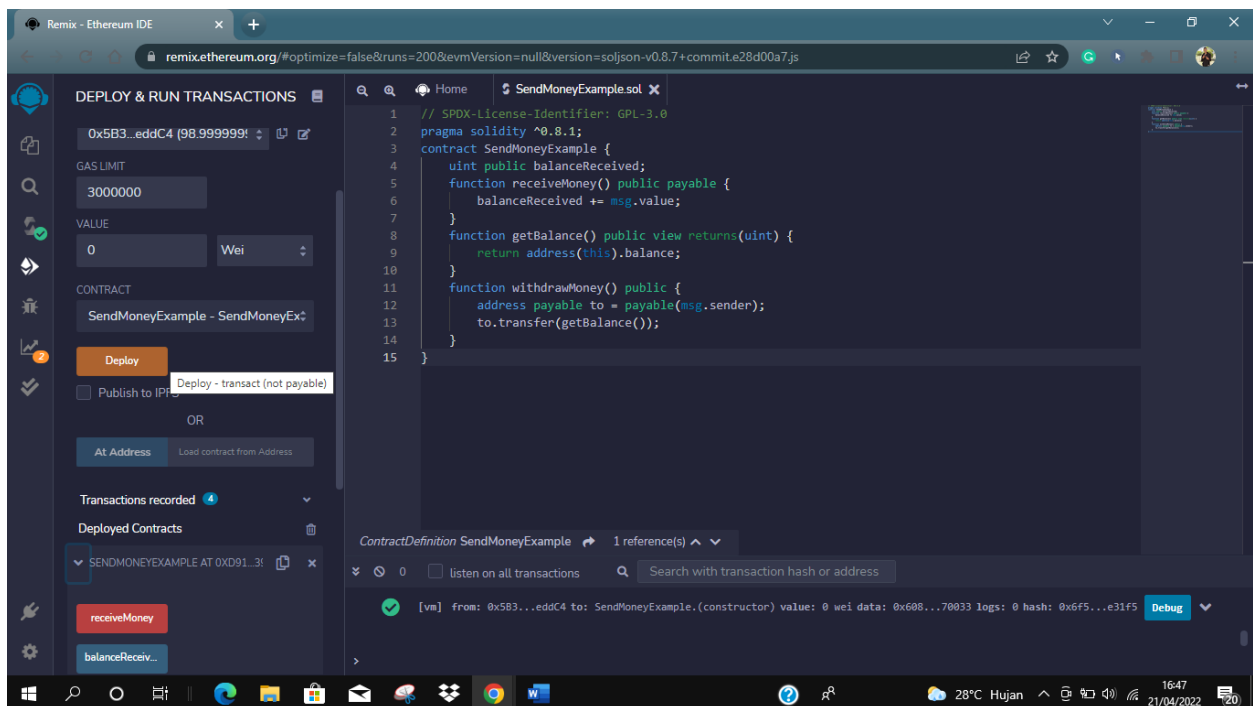


The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays the 'SendMoneyExample - SendMoneyEx' contract. The 'Deploy' button is visible, along with options to 'Publish to IPFS' or 'At Address'. Below, the 'Transactions recorded' section shows three transactions, and the 'Deployed Contracts' section lists 'SENDMONEYEXAMPLE AT 0XD91...3'. The main editor displays the Solidity code for 'SendMoneyExample.sol':

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.1;
3 contract SendMoneyExample {
4     uint public balanceReceived;
5     function receiveMoney() public payable {
6         balanceReceived += msg.value;
7     }
8     function getBalance() public view returns(uint) {
9         return address(this).balance;
10    }
11    function withdrawMoney() public {
12        address payable to = payable(msg.sender);
13        to.transfer(getBalance());
14    }
15 }
```

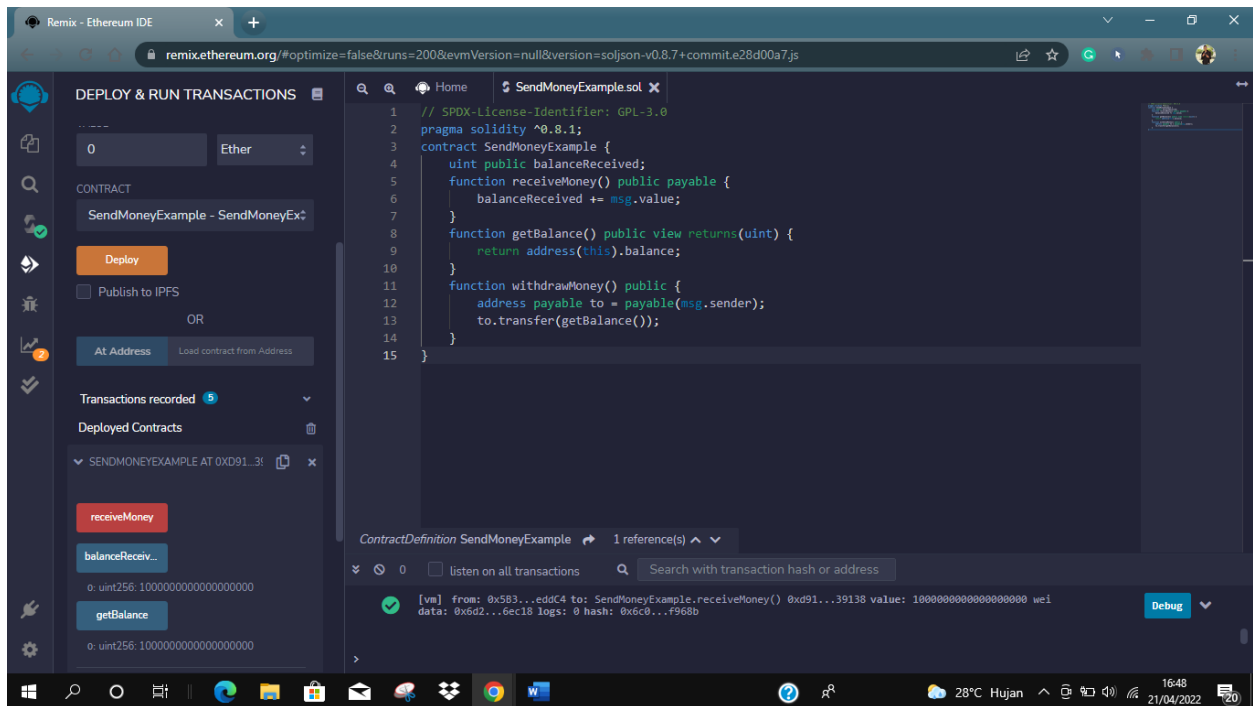
The bottom panel shows a transaction log with a call from '0x5830a6a791c568545dcfc803fc8b75f56beddC4' to 'SendMoneyExample.getBalance()' with data '0x120...65fe0'.

Deploy the new version and send again 1 Ether to the Smart Contract. To avoid confusion, I recommend you close the previous Instance, we won't need it anymore.

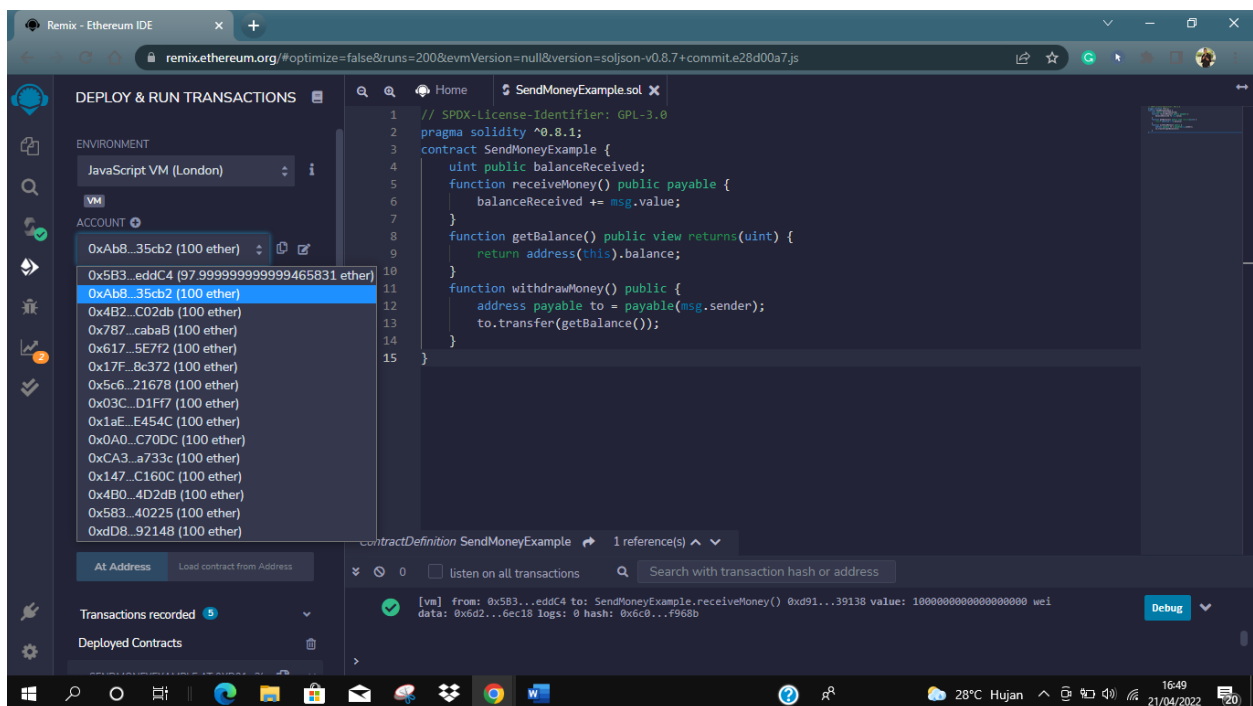


The screenshot shows the Remix IDE interface after deploying the smart contract. The 'DEPLOY & RUN TRANSACTIONS' panel shows the 'SendMoneyExample - SendMoneyEx' contract with a 'Deploy' button. The 'GAS LIMIT' is set to '3000000' and the 'VALUE' is '0 Wei'. The 'Deploy' button is highlighted, and a tooltip 'Deploy - transaction (not payable)' is visible. The main editor displays the same Solidity code as the previous screenshot. The bottom panel shows a transaction log with a successful deployment from '0x583...eddC4' to 'SendMoneyExample.(constructor)' with a value of '0 wei' and data '0x608...70033 logs: 0 hash: 0x6f5...e31f5'.

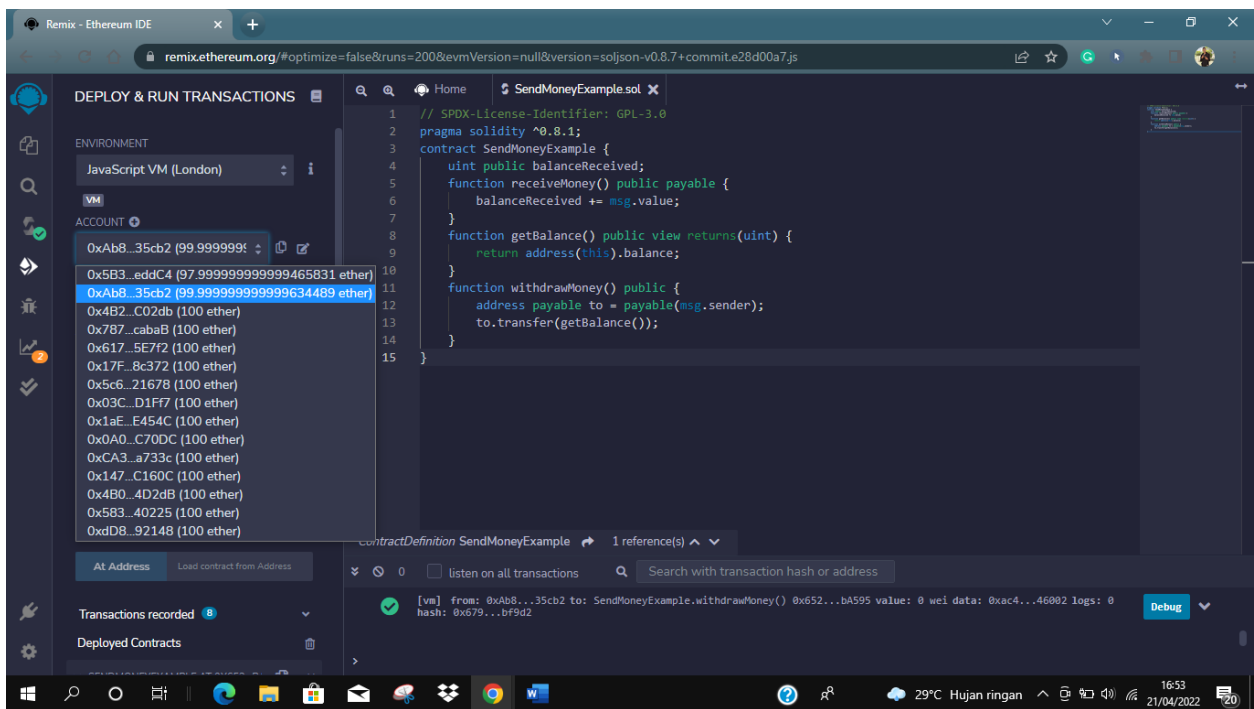
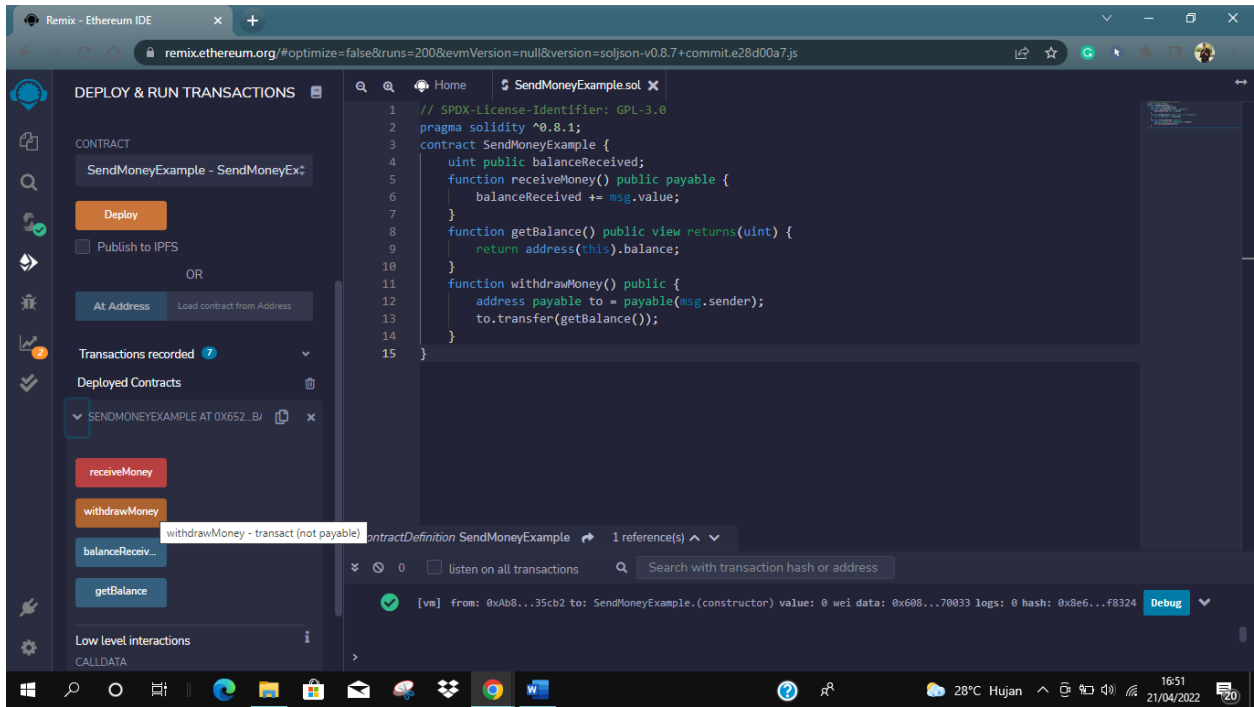
Put in "1 Ether" into the value input box. hit "receiveMoney" in your new contract Instance.

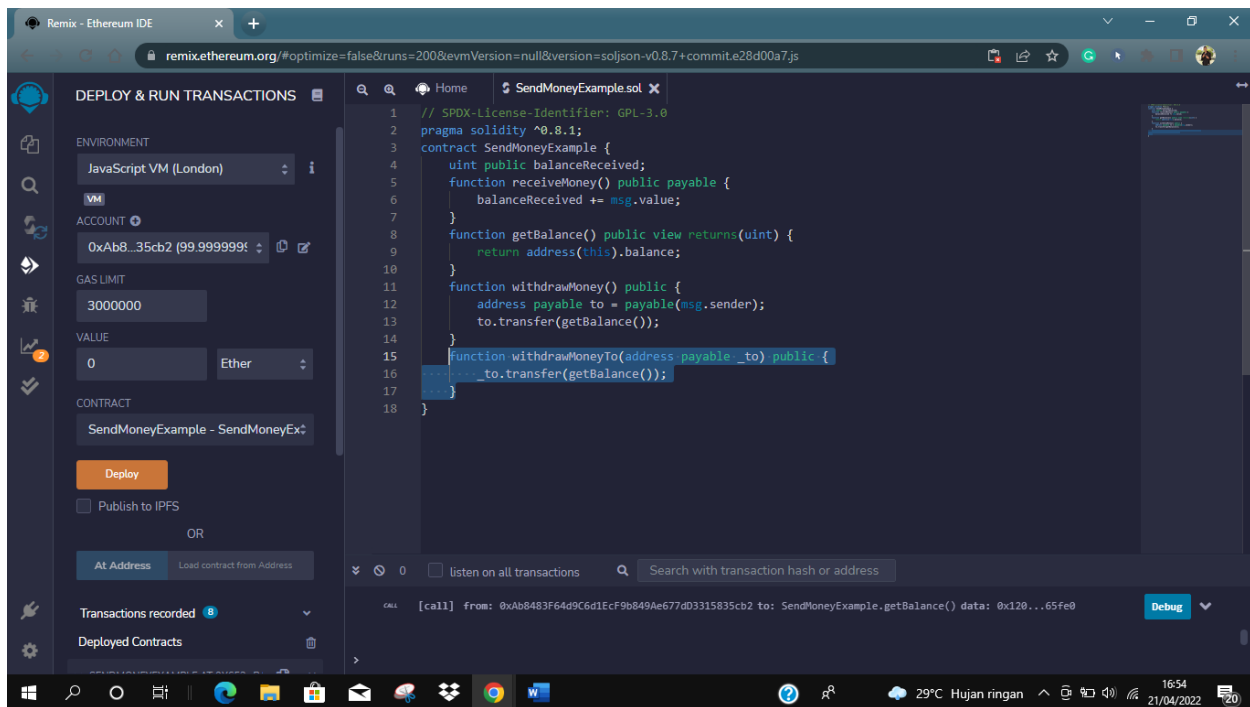


Select the second Account from the Accounts dropdown:

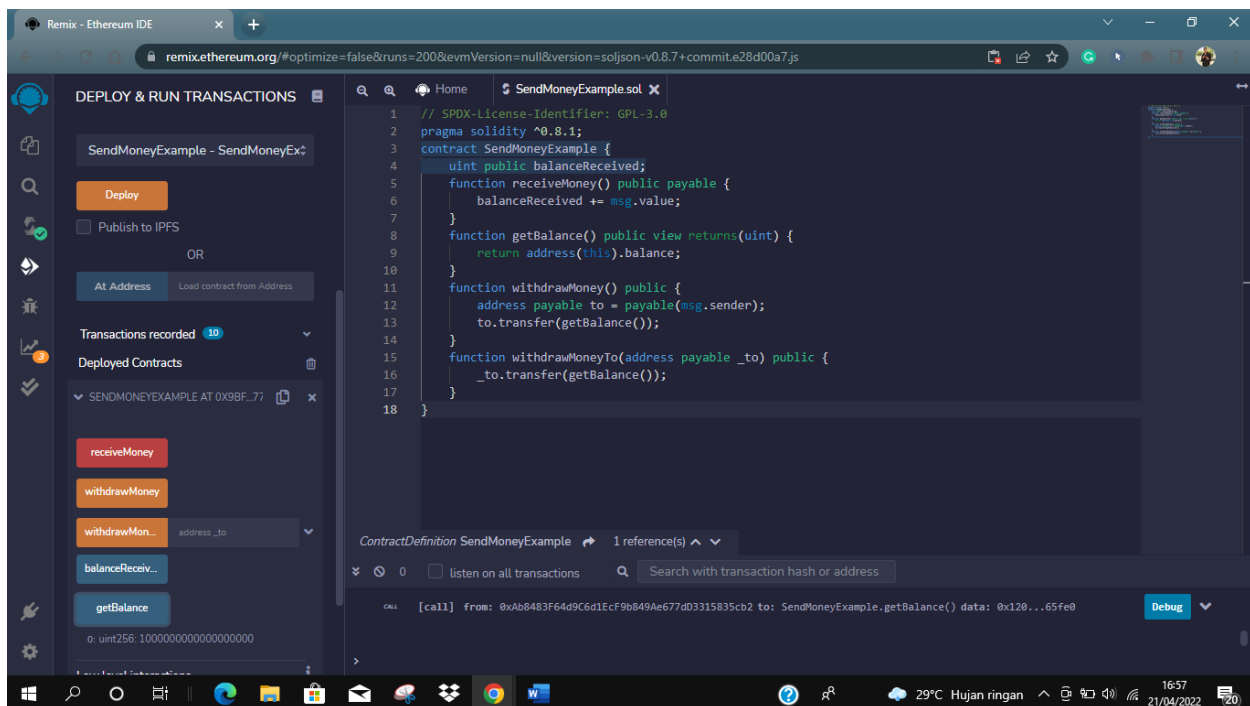


Then hit the "withdrawMoney" button:

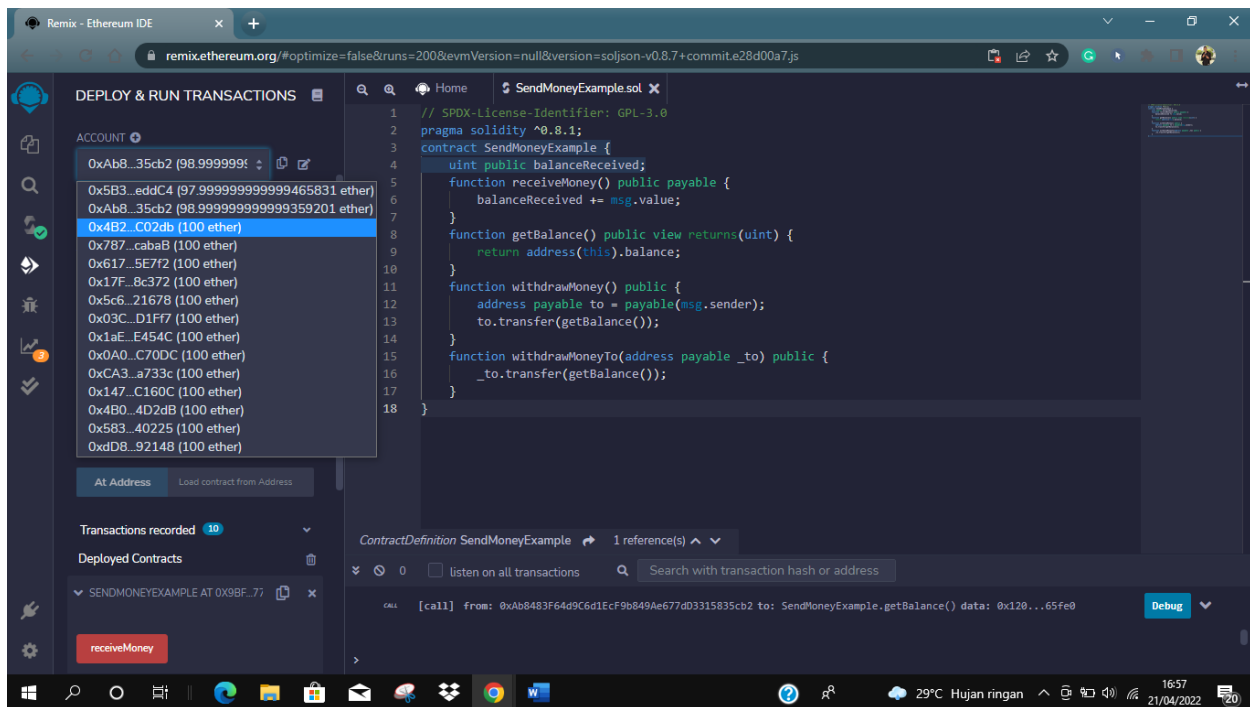




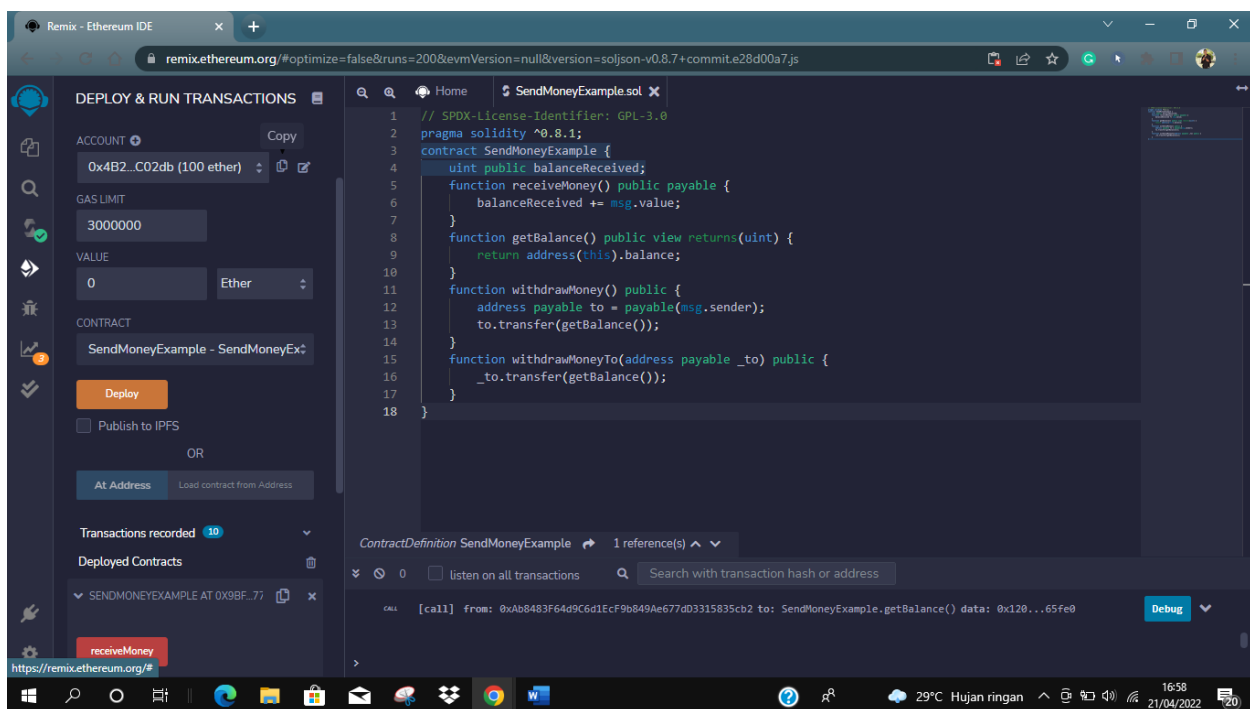
Deploy the Smart Contract. Close the old Instance. Send 1 Ether to the Smart Contract (don't forget the value input field!). Make sure the Balance shows up correctly.



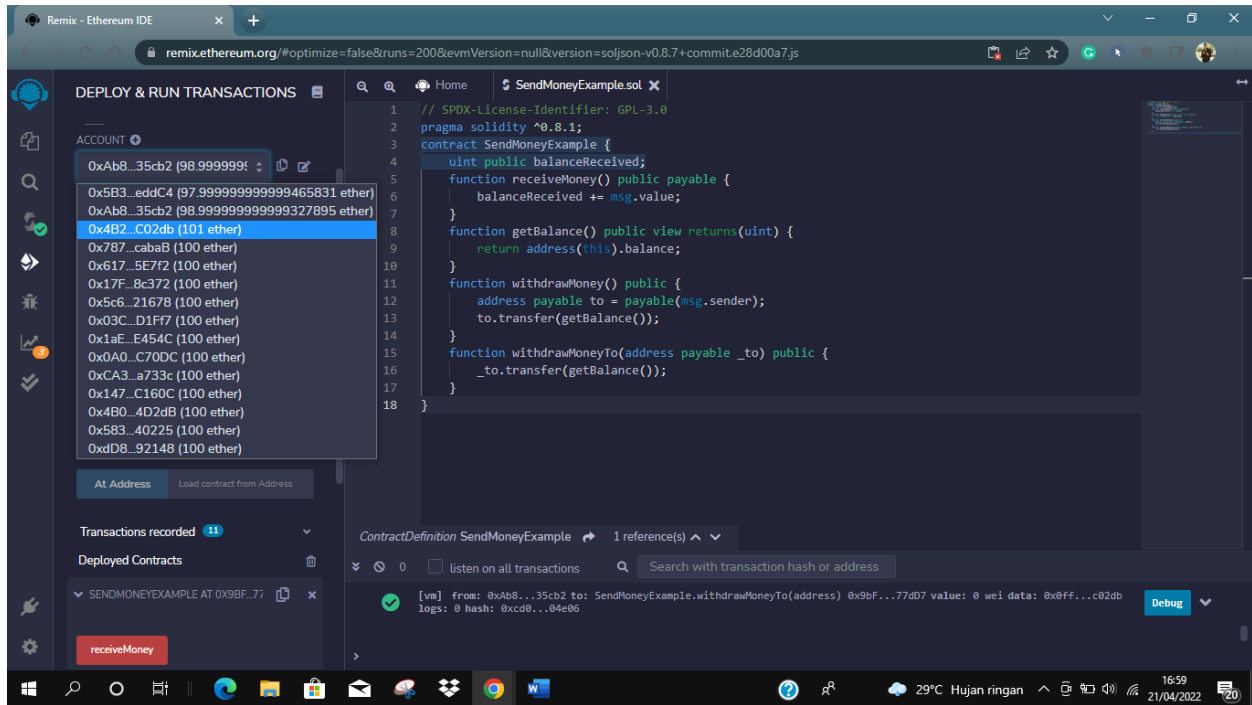
Select the third account from the dropdown



Hit the little "copy" icon:



Switch back to the first Account. Paste the Account you copied into the input field next to "withdrawMoneyTo": Now open the Accounts dropdown. See the balance of your third Account? 101 Ether!!!



Click "withdrawMoney" - and nothing happens. The Balance stays the same until 1 Minute passed since you hit "receiveMoney".

The screenshot displays the Remix Ethereum IDE interface. The top bar shows the browser address: `remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.7+commit.e28d00a7.js`. The left sidebar contains the 'DEPLOY & RUN TRANSACTIONS' panel, which includes a 'Deployed Contracts' section. Under this section, a contract named 'SENDMONEYEXAMPLE AT 0xAc4...31' is listed. Below the contract name, there are several buttons: 'receiveMoney' (orange), 'withdrawMoney' (orange), 'withdrawMon...' (orange), 'balanceReceiv...' (blue), 'getBalance' (blue), and 'lockedUntil' (blue). A tooltip is visible over the 'withdrawMoney' button, stating 'withdrawMoney - transact (not payable)'. The main editor area shows the Solidity code for the 'SendMoneyExample.sol' contract. The code includes functions for 'receiveMoney', 'getBalance', 'withdrawMoney', and 'withdrawMoneyTo'. The 'withdrawMoney' function has a condition that checks if the current time is less than the 'lockedUntil' time, which is set to 'block.timestamp + 1 minutes'. The bottom status bar shows a call log with a transaction from '0xab8483f64d9c6d1ecf9b849ae677d03315835cb2' to 'SendMoneyExample.getBalance()' with data '0x120...65fe0'. The system tray at the bottom indicates the date and time as '21/04/2022 17:07'.

```
contract payable {
    uint256 public balanceReceived;
    uint256 public lockedUntil;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
        lockedUntil = block.timestamp + 1 minutes;
    }

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }

    function withdrawMoney() public {
        if(lockedUntil < block.timestamp) {
            address payable to = payable(msg.sender);
            to.transfer(getBalance());
        }
    }

    function withdrawMoneyTo(address payable _to) public {
        if(lockedUntil < block.timestamp) {
            _to.transfer(getBalance());
        }
    }
}
```