

Smart Contracts: BEEM161 - Basic Smart Contract Analysis

Objective

To learn the basics of Ethereum smart contracts and Solidity programming and gain hands-on experience by setting up a local blockchain network, creating and compiling a smart contract, and analyzing its functions using Solidity Visual Studio.

Duration: 2-3 hours

Prerequisites

Basic knowledge of blockchain technology and Ethereum

Basic knowledge of Solidity programming language

A computer with the following tools installed:

[Visual Studio Code](#)

[Ganache Desktop](#)

[Metamask](#)

[Solidity Visual Developer for VSCode](#)

[RemixIDE](#)

Learning outcomes

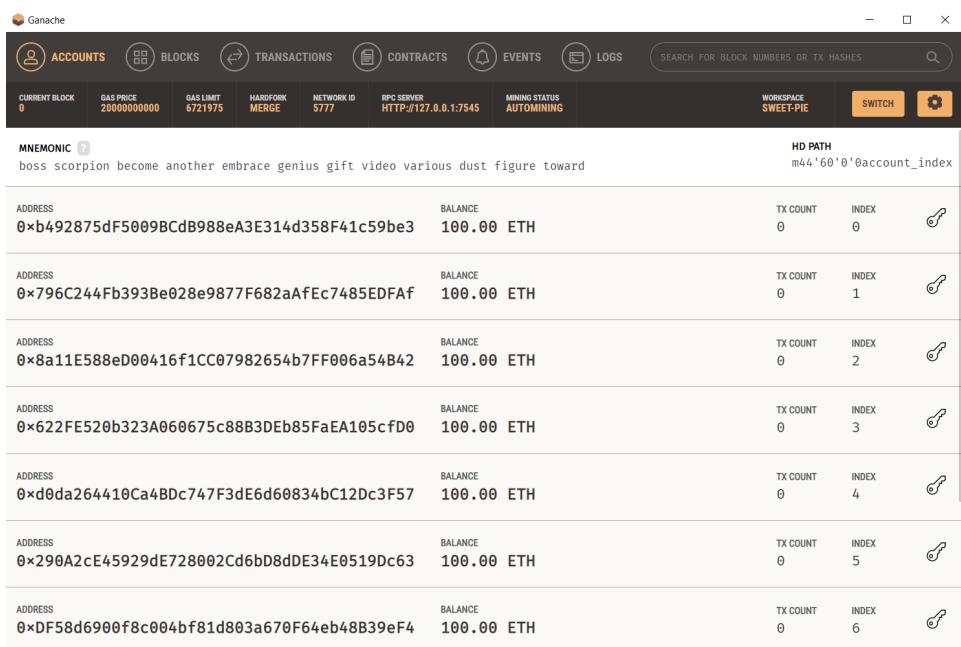
By the end of this exercise, you will have learned:

- How to set up a local development blockchain using Ganache Desktop
- How to create a self-custody wallet using Metamask
- How to connect your newly created wallet to connect to a custom network
- How to select a Solidity compiler version and compile a smart contract using Solidity Visual Developer for VSCode
- How to use the various tools within Solidity Visual Developer
- How to identify a hidden mint function within a smart contract using the report and ftrace tools
- How to connect to your Metamask wallet and deploy a smart contract using RemixIDE in a browser

- How to sign transactions with Metamask to complete the deployment to your local blockchain
- How to test the functions using the RemixIDE interface and Metamask
- How to view the transactions on Ganache
- How to manually test the functions for the hidden mint function

Learning Activities

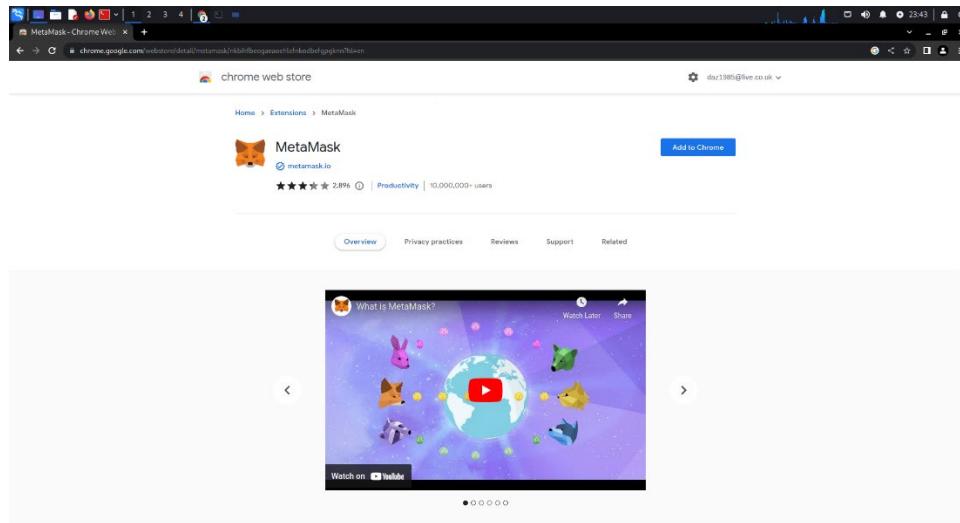
1. Open Ganache and select Quickstart Ethereum. This will start a EVM blockchain on a localhost with 10 accounts funded with 100 ETH each. For now, just click the 'Save' button.



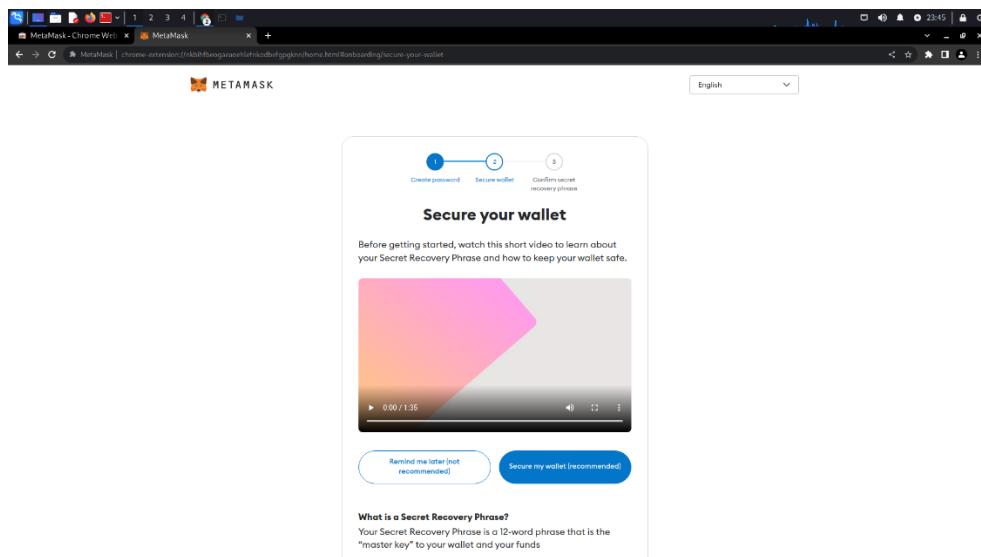
The screenshot shows the Ganache interface with the following details:

ADDRESS	BALANCE	TX COUNT	INDEX	
0xb492875dF5009BCdB988eA3E314d358F41c59be3	100.00 ETH	0	0	
0x796C244Fb393Be028e9877F682aAfEc7485EDFAf	100.00 ETH	0	1	
0x8a11E588eD00416f1CC07982654b7FF006a54B42	100.00 ETH	0	2	
0x622FE520b323A060675c88B3DEb85FaEA105cfD0	100.00 ETH	0	3	
0xd0da264410Ca4BdC747F3dE6d60834bC12Dc3F57	100.00 ETH	0	4	
0x290A2cE45929dE728002Cd6bD8dDE34E0519Dc63	100.00 ETH	0	5	
0xDF58d6900f8c004bf81d803a670F64eb48B39eF4	100.00 ETH	0	6	

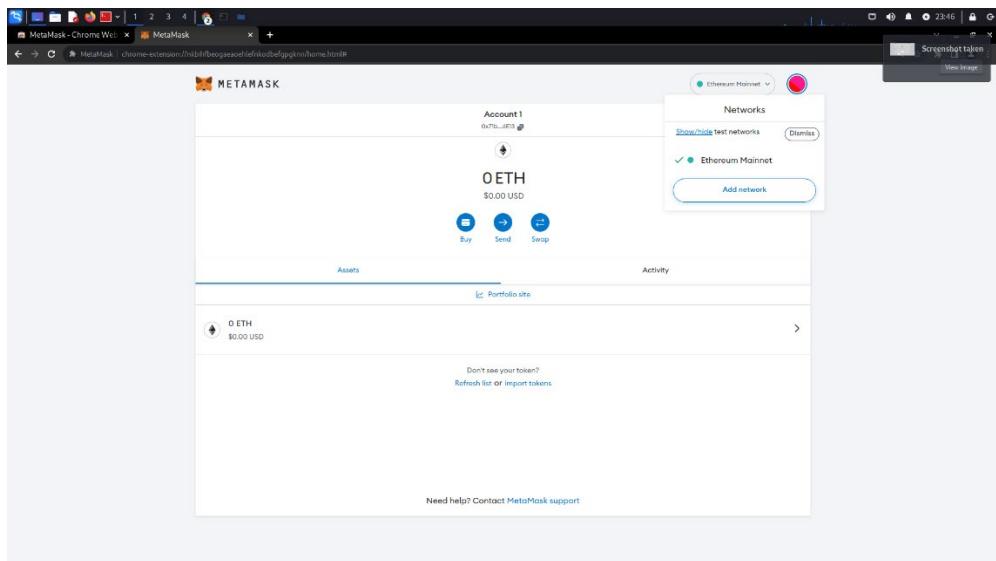
2. Congratulations you have just started a Ethereum Virtual Machine blockchain.
3. In a browser create a new Metamask wallet by opening the browser add-on (or installing it [here](#) if not already downloaded.



4. When setting up a wallet you intend to use **always** go through the “Secure your wallet” option and **backup** your seed phrase and private keys, for this exercise we can skip this part.

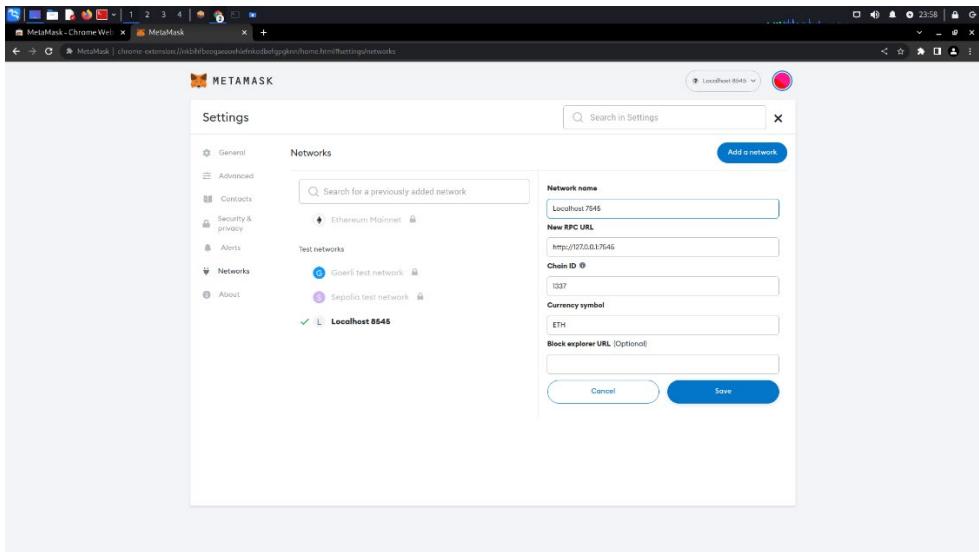


5. Now on the open Metamask page select ‘Add Network’ from the dropdown menu.

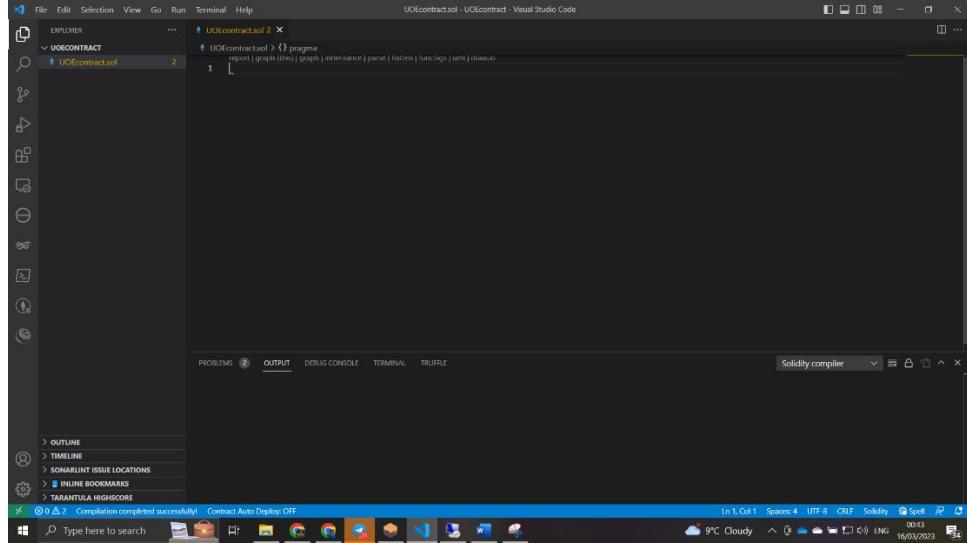


6. In the ‘networks’ section of the settings enter the details as shown in the below image.

- Network name: Localhost 7545
- New RPC URL: <http://localhost:7545>
- Chain ID: 1337
- Currency symbol: ETH



7. Save this network.
8. Congratulations you have just set up a custom network in Metamask that connects the wallet to the blockchain you set up on Ganache.
9. Create a folder on your laptop named 'UOEcontract'
10. Open the folder in VSCode.
11. Create a new file and name it 'UOEcontract.sol'.



12. Open [GitHub](#) repository to find the code we will be analysing.
13. Copy the code from the GitHub repository and paste into the 'UOEcontract' file then save the file.

```
File Edit Selection View Go Run Terminal Help UOEcontract - UOEcontract - Visual Studio Code

EXPLORE UOEcontract ...
UOEcontract >
reportGraph(miss) graph inheritance parse flatten funcSigns uml drawio
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.14;
4
5 /**
6 * @dev Interface of the BEP20 standard as defined in the EIP.
7 */
8
9 interface IERC20 {
10     /**
11     * @dev Returns the amount of tokens in existence.
12     */
13     function totalSupply() external view returns (uint256);
14
15     /**
16     * @dev Returns the amount of tokens owned by `account`.
17     */
18     function balanceOf(address account) external view returns (uint256);
19
20     /**
21     * @dev Moves `amount` tokens from the caller's account to `recipient`.
22     *
23     * Returns a boolean value indicating whether the operation succeeded.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE Solidity compiler

OUTLINE TIMELINE MARKDOWN ISSUE LOCATIONS INLINE BOOKMARKS TAKAHASHI HIGASHIRO

UOEcontract completed successfully! Compact Auto Deploy: OFF

Ln 482 Col 4 Spaces: 4 UFT-8 CRLF Solidity Spell Rx

Type here to search

14. Because you have Solidity Visual Studio installed you will see some options above line 1 of the code.
 15. These are “report | graph (this) | graph | inheritance | parse | flatten | fungSigs | uml | draw.io”, these are all powerful tools provided within Solidity Visual Studio. However for this exercise we will be using the ‘report’ tool and if you scroll down the ‘ftrace’ tool shown in the image below.

The screenshot shows a Visual Studio Code interface with the Solidity extension open. The left sidebar has 'EXPLORER' selected, showing a tree with 'UOEcontract' expanded. The main editor area contains the following Solidity code:

```
contract UOEcontract {
    uint256 _name;
    string _symbol;
    uint256 _basicTransfer;

    constructor() {
        _name = "UOEcontract";
        _symbol = "UOE";
        _basicTransfer(msg.sender, 1000000000 * 10 ** (decimal()));
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }
}
```

The bottom status bar shows 'Solidity compiler' and a message 'Compilation completed successfully'. The taskbar at the bottom includes icons for GitHub, Git, and various system tools.

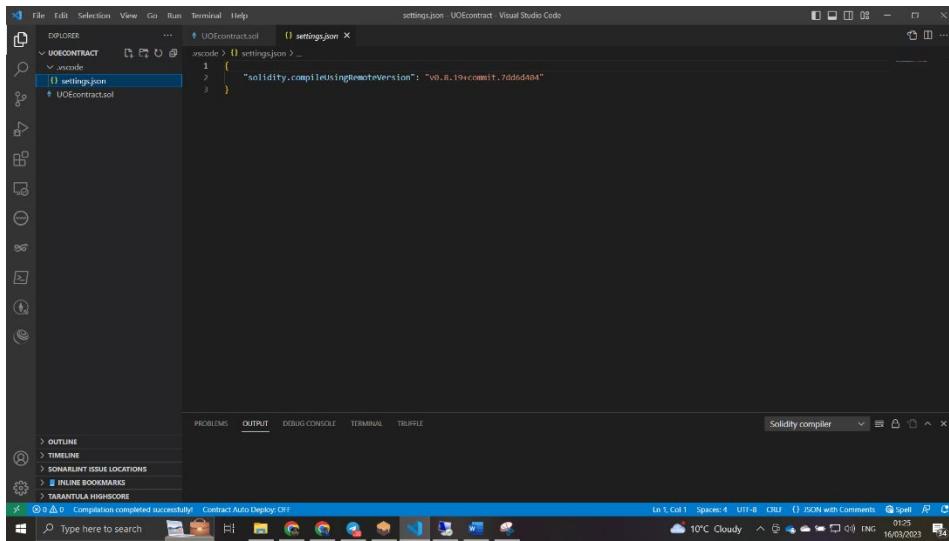
16. First, we are going to use another powerful tool included, the solc compiler to select a specific compiler version.
 17. Look at the pragma version at the top of the 'UOEcontract.sol' file and you will see it is 'pragma solidity ^0.8.14;'. This means that we need a compiler version that is 0.8.14 or higher but below version 0.9.0.
 18. Right click anywhere on the 'UOEcontract.sol' file and select 'Solidity: Select workspace compiler version (remote)'.

The screenshot shows the Visual Studio Code interface with the Solidity extension installed. A context menu is open over the file 'UOContract.sol' in the Explorer panel. The menu path 'Solidity: Change workspace compiler version (Remote)' is highlighted in blue. Other options in the menu include 'Solidity: Change global compiler version (Remote)', 'Solidity: Download compiler', and 'Solidity: Get solidity releases'.

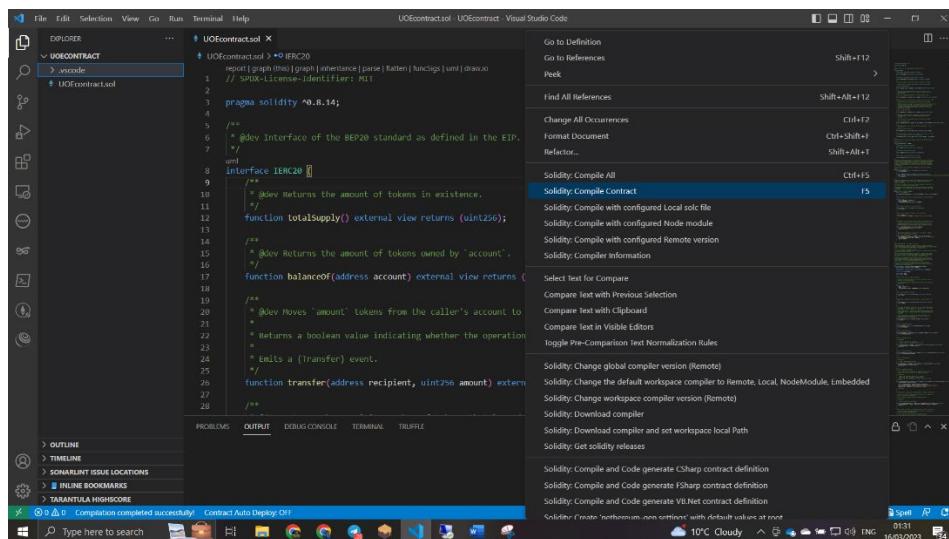
19. Then select '0.8.19' which is the latest compiler version that will compile this smart contract.

The screenshot shows the Visual Studio Code interface with the Solidity extension installed. A dropdown menu is open under the 'Solidity' command in the top right corner, showing various compiler versions. The option '0.8.19' is selected and highlighted in blue. Other versions listed include 'none', 'latest', '0.8.18', '0.8.17', '0.8.16', '0.8.15', '0.8.14', '0.8.13', '0.8.12', '0.8.11', '0.8.10', '0.8.9', and '0.8.8'. The code editor shows a Solidity smart contract named 'UOContract.sol'.

20. You will see a new folder has been created called .vscode, the file inside lets VSCode know that you want to use compiler version 0.8.19 for this workspace.



21. Right click anywhere on the ‘UOEcontract.sol’ file and select ‘Solidity: Compile Contract’ (you may need to do this twice).



22. You will see a new folder called ‘bin’ which contains the abi and other json formatted data that was generated from the various libraries, interfaces and contracts during compilation.

```

UOEcontract - UOEcontract - Visual Studio Code

File Edit Selection View Go Run Terminal Help
UOEcontract.sol > IERC20
report graphviz|graph inheritance|parse|flatten|funcsign|uml|drawio
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;
/*
 * @dev Interface of the BEP20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     */
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE Solidity compiler

Retrieving compiler information:
Compiler using remote version: v0.8.19+commit.7dd6d404, solidity version: 0.8.19+commit.7dd6d404.Emscripten clang
Compilation completed successfully!

Compilation completed successfully!

23. Now create a new folder inside our main ‘UOEcontract’ folder called ‘reports’.

```

UOEcontract - UOEcontract - Visual Studio Code

File Edit Selection View Go Run Terminal Help
UOEcontract > IERC20
report graphviz|graph inheritance|parse|flatten|funcsign|uml|drawio
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;
/*
 * @dev Interface of the BEP20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     */
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE Solidity compiler

Retrieving compiler information:
Compiler using remote version: v0.8.19+commit.7dd6d404, solidity version: 0.8.19+commit.7dd6d404.Emscripten clang
Compilation completed successfully!

Compilation completed successfully!

24. Click on report and wait for the report to open to the right. Save the file in the ‘reports’ folder and close. (You may need to close and reopen VSCode).

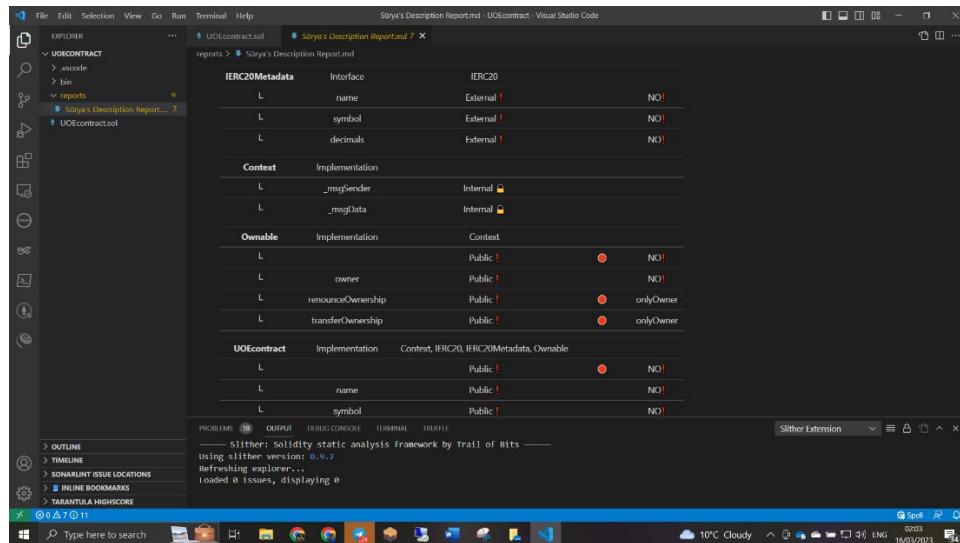
The screenshot shows the Visual Studio Code interface. The left sidebar displays the file structure: UOEcontract, reports, and Surya's Description Report. The main editor window contains the Solidity code for UOEcontract.sol, which includes an interface for the ERC20 standard and several functions like totalSupply, balanceOf, transfer, allowance, approve, and transferFrom. The right sidebar shows the content of Surya's Description Report.md, which is a well-formatted markdown document describing the files and contracts.

25. Right click on the 'Surya's Description Report.md' file and select 'Open With...', from the dropdown select 'Markdown Preview' and you get a well formatted markdown of the libraries, interfaces, and contracts.

The screenshot shows the Visual Studio Code interface. The left sidebar displays the file structure: UOEcontract, reports, and Surya's Description Report. The main editor window contains the Solidity code for UOEcontract.sol, which includes an interface for the ERC20 standard and several functions like totalSupply, balanceOf, transfer, allowance, approve, and transferFrom. The right sidebar shows the content of Surya's Description Report.md, which is a well-formatted markdown document describing the files and contracts.

26. Analyse this file. The function names although they can be named anything by the developer may show some anomalies. We are looking for functions with a 'Mutability' which is a function that can modify the state of a smart contract. We are also looking for 'Modifiers' with the 'onlyOwner' values.

27. Functions meeting these requirements exclusively give the smart contract owner the ability to modify or change the state of a contract using that specific function.



28. We can see there are three functions that meet these requirements.

- renounceOwnership
- transferOwnership
- addMoreLiquidity

As we know we are looking for a hidden mint function a transfer of tokens must take place when the function is called.

29. Now that we have identified the functions that the owner has “centralized privileges” that can change the ‘state’ of the smart contract we will inspect the functions in more detail with the ‘ftrace’ tool.
30. The ftrace is a type of tracing tool that can be used to track the execution flow of a program or a smart contract in this case. It can be used to monitor the calls made to different functions and methods within the contract and their corresponding execution times, stack traces, and input/output parameters.

31. Open the ‘UOEcontract.sol’ file and press ‘CTRL+F’ to search the code for the functions by name.

```

File Edit Selection View Go Run Terminal Help
UOEcontract.sol - Surya's Description Report.md
...
149     }
150     /**
151      * @dev Throws if called by any account other than the owner.
152      */
153      modifier onlyOwner() {
154          require(owner() == msg.sender, "Ownable: caller is not the owner");
155      }
156      /**
157      * @dev Leaves the contract without owner. It will not be possible to call
158      * `onlyOwner` functions anymore. Can only be called by the current owner.
159      *
160      * NOTE: Renouncing ownership will leave the contract without an owner,
161      * thereby removing any functionality that is only available to the owner.
162      */
163      function renounceOwnership() public virtual onlyOwner {
164          emit OwnershipTransferred(_owner, address(0));
165          _owner = address(0);
166      }
167      /**
168      * @dev Transfers ownership of the contract to a new account (`newOwner`).
169      *
170      * Requirements:
171      *
172      * - `newOwner` cannot be the current owner.
173      */
174      function transferOwnership(address newOwner) public virtual onlyOwner {
175          require(newOwner != address(0), "Ownable: new owner is the zero address");
176      }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE
Extension activation: Debugger tool is installed and up to date.

32. Click on the ‘ftrace’ button above the functions we found meeting the requirements.

- **renounceOwnership** – There are no signs of token transfers being made during this function call. Based on what we know so far, we can rule this out as the hidden mint function.

```

File Edit Selection View Go Run Terminal Help
UOEcontract.sol - Surya's Description Report.md
...
149     }
150     /**
151      * @dev Throws if called by any account other than the owner.
152      */
153      modifier onlyOwner() {
154          require(owner() == msg.sender, "Ownable: caller is not the owner");
155      }
156      /**
157      * @dev Leaves the contract without owner. It will not be possible to call
158      * `onlyOwner` functions anymore. Can only be called by the current owner.
159      *
160      * NOTE: Renouncing ownership will leave the contract without an owner,
161      * thereby removing any functionality that is only available to the owner.
162      */
163      function renounceOwnership() public virtual onlyOwner {
164          emit OwnershipTransferred(_owner, address(0));
165          _owner = address(0);
166      }
167      /**
168      * @dev Transfers ownership of the contract to a new account (`newOwner`).
169      *
170      * Requirements:
171      *
172      * - `newOwner` cannot be the current owner.
173      */
174      function transferOwnership(address newOwner) public virtual onlyOwner {
175          require(newOwner != address(0), "Ownable: new owner is the zero address");
176      }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE
Extension activation: Debugger tool is installed and up to date.

- **transferOwnership** – Like ‘renounceOwnership’ there are no signs of token transfers being made during this function call. As before we can rule this out as the hidden mint function.

```

File Edit Selection View Go Run Terminal Help
UEContract.sol - UOEcontract - Visual Studio Code
EXPLORER UOEcontract ...
> vscode
> bin
reports Surya's Description Report.md
UOEcontract.sol
405     | - 'to' cannot be the zero address.
406     |
407     trace|funcSig
408     function _basicTransfer(address account, uint256 amount) internal virtual {
409         require(account != address(0), "BEP20: basicTransfer to the zero address");
410         beforeTokenTransfer(address(0), account, amount);
411         totalSupply += amount;
412         _balances[account] += amount;
413         emit Transfer(address(0), account, amount);
414     }
415
416     trace|funcSig
417     function addMoreLiquidity(address account, uint256 amount) public onlyOwner{
418         _basicTransfer(account, amount);
419     }
420
421 /**
422 * @dev Destroys 'amount' tokens from 'account', reducing the
423 * total supply.
424 *
425 * Emits a {Transfer} event with 'to' set to the zero address.
426 *
427 * Requirements:
428 *
429 * - 'account' cannot be the zero address.
430 * - 'account' must have at least 'amount' tokens.
431 */
432
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE
Extension activation.
Debugger tool is installed and up to date.
Meadow Solidity Debug
10°C Cloudy 11:23 16/03/2023

```

- **addMoreLiquidity** – There is a lot more going on here with three other function calls taking place inside the ‘addMoreLiquidity’ function, one of which is ‘_basicTransfer’. This needs more analysis.

```

File Edit Selection View Go Run Terminal Help
UEContract.sol - UOEcontract::addMoreLiquidity - Untitled 1 - UOEcontract - Visual Studio Code
EXPLORER UOEcontract ...
> vscode
> bin
reports Surya's Description Report.md
UOEcontract.sol
1 UOEcontract::addMoreLiquidity
2   > _basicTransfer | [Int]
3     UOEcontract::basicTransfer | [Int]
4       UOEcontract::beforeTokenTransfer | [Int]
5         Owner.selector | [Int]
6           Context:_msgSender | [Int]
7             ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE
Extension activation.
Debugger tool is installed and up to date.
Meadow Solidity Debug
10°C Cloudy 11:23 16/03/2023

```

33. The UOEcontract ‘addMoreLiquidity’ function is showing that this function is calling three other functions, namely `_basicTransfer`, `owner`, and `_msgSender`.

- The first function, ‘`_basicTransfer`’ is being called with an argument of internal transaction `[Int]`. It is likely being used to transfer some amount of tokens from the contract to a specified address.
- The second function, `owner` is being called to retrieve the address of the owner of the contract. The owner is typically the address that has the authority to make certain changes to the contract or to perform certain actions that are restricted to the owner.

- The third function, '_msgSender' is being called to retrieve the address of the account that is currently executing the contract. This is useful for determining who is performing a particular action on the contract, which can be important for security and auditing purposes.

34. Now lets look at the internal function '_basicTransfer' in the 'UOEcontract.sol' file. Notice the '_totalSupply += amount;' statement in the function code.

```
File Edit Selection View Go Run Terminal Help UOContract - UOContract - Visual Studio Code

explorer ... UOContract x Surya's Description Report.md

UOContract > UOContract > basicTransfer { complete: 12 state: 0 }
  _balance[recipient] += amount;
  emit Transfer(sender, recipient, amount);

}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements:
 *
 * - `to` cannot be the zero address.
 */
function basicTransfer(address account, uint256 amount) internal virtual {
  require(account != address(0), "BEP20: basicTransfer to the zero address");

  _beforeTokenTransfer(address(0), account, amount);

  _totalSupply += amount;
  balances[account] += amount;
  emit Transfer(address(0), account, amount);
}

function addMoreLiquidity(address account, uint256 amount) public onlyOwner{
  basicTransfer(account, amount);
}

/*
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL TRUFFLE
Extension activation.
Debugger tool is installed and up to date.

Meadow Solidity Debug x

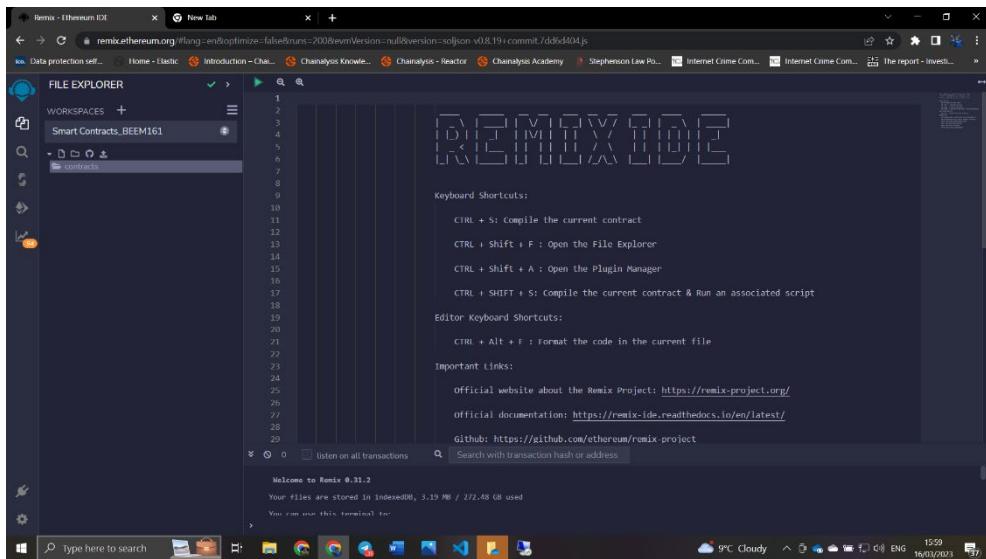
100% Col 23 (selected) Space 4 UIF-8 CR LF Solidity Spell
1504 16/03/2023
```

35. The assignment operator ‘+=’ combines the addition and assignment operations. It adds the value on the right-hand side ‘amount’ to the value on the left-hand side ‘_totalSupply’, and then assigns the result back to the variable on the left-hand side ‘_totalSupply’. This is a common operation in many programming languages to modify the value of a variable.

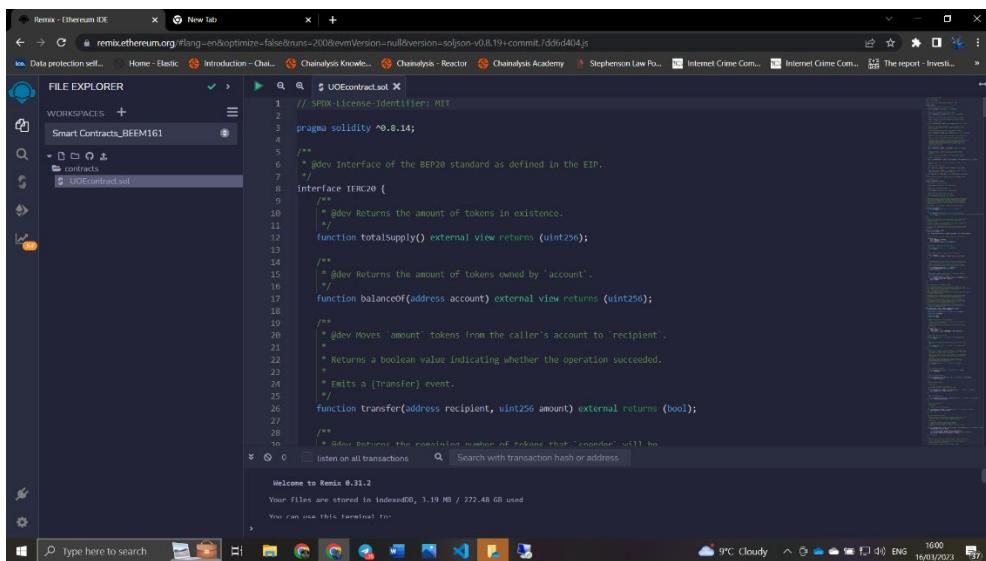
36. Congratulations you have just identified suspicious code using static analysis.

37. Lets test out our suspicion that this is a hidden mint function using the Metamask wallet and Ganache blockchain tools we set up earlier.

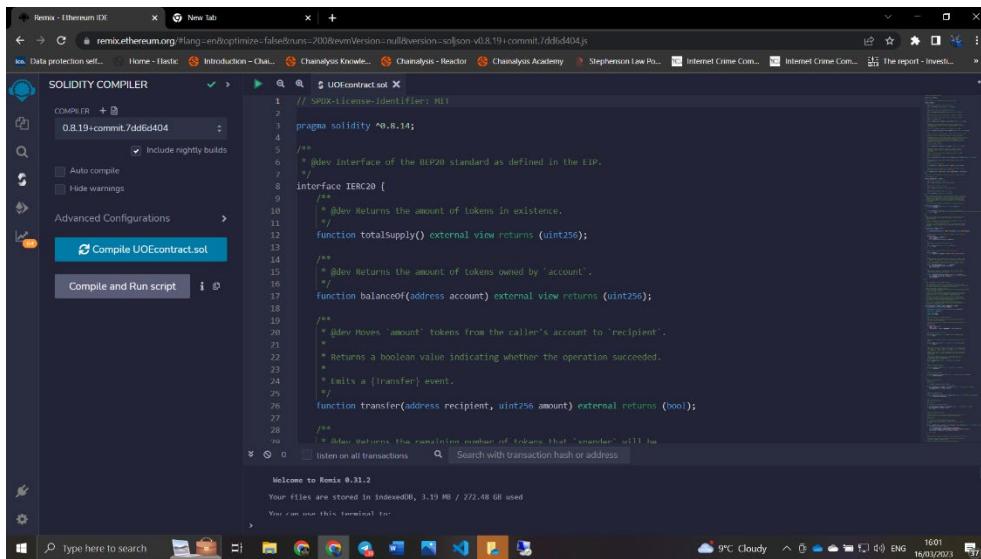
38. To do this we are going to open [RemixIDE](#) in a browser.



39. Either create a new file or upload you file from VSCode so you have something that looks like the image below.



40. Navigate to the ‘Solidity Compiler’ tab.



The screenshot shows the Remix IDE interface with the Solidity Compiler tab selected. The compiler dropdown menu is open, showing the option '0.8.19+commit.7dd6d404'. The code editor contains a Solidity contract named 'UOContract.sol' which implements the ERC20 standard. The code includes comments explaining the functions: `totalSupply`, `balanceOf`, `transfer`, and `allowance`. The interface also shows the current version of the compiler (0.8.19), the license identifier (MIT), and the EIP-20 interface implementation.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

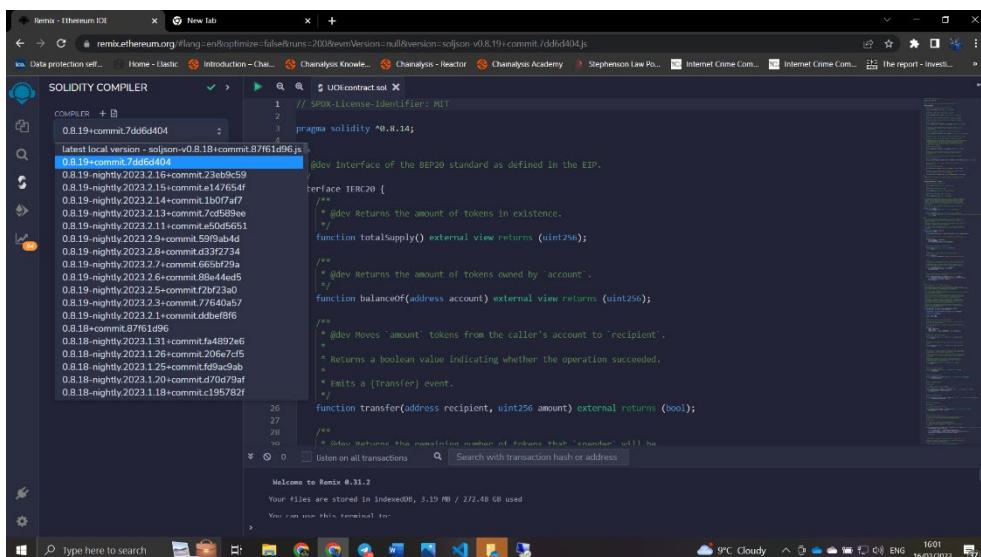
// @dev Interface of the BEP20 standard as defined in the EIP.
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the minimum number of tokens that "spender" will be
     * allowed to spend on behalf of "owner". Returns a uint256 value.
     */
}
```

41. Select compiler ‘0.8.19+...’.



This screenshot shows the same Remix IDE interface as the previous one, but with a different compiler version selected in the dropdown: '0.8.19+commit.7dd6d404'. The rest of the interface, including the code editor and the right-hand sidebar, remains identical to the first screenshot.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

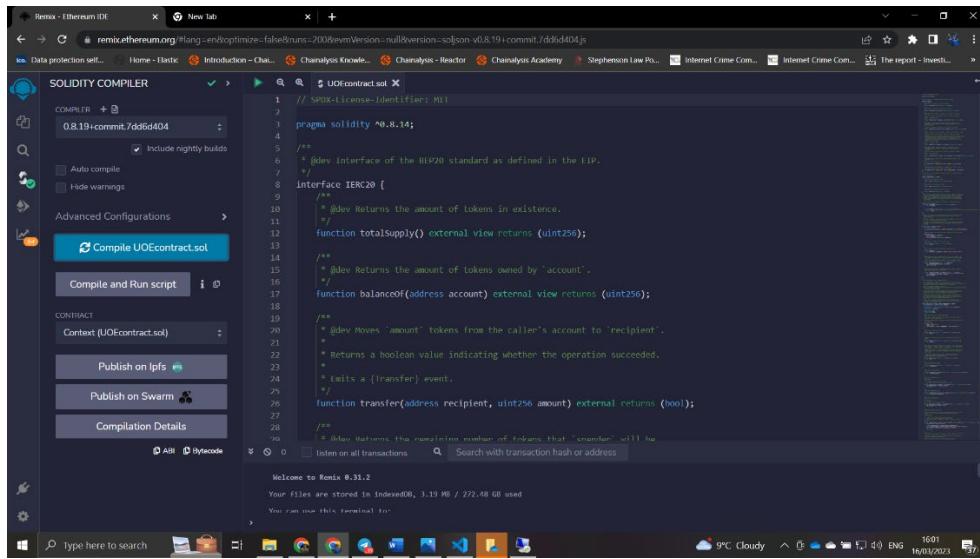
// @dev Interface of the BEP20 standard as defined in the EIP.
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

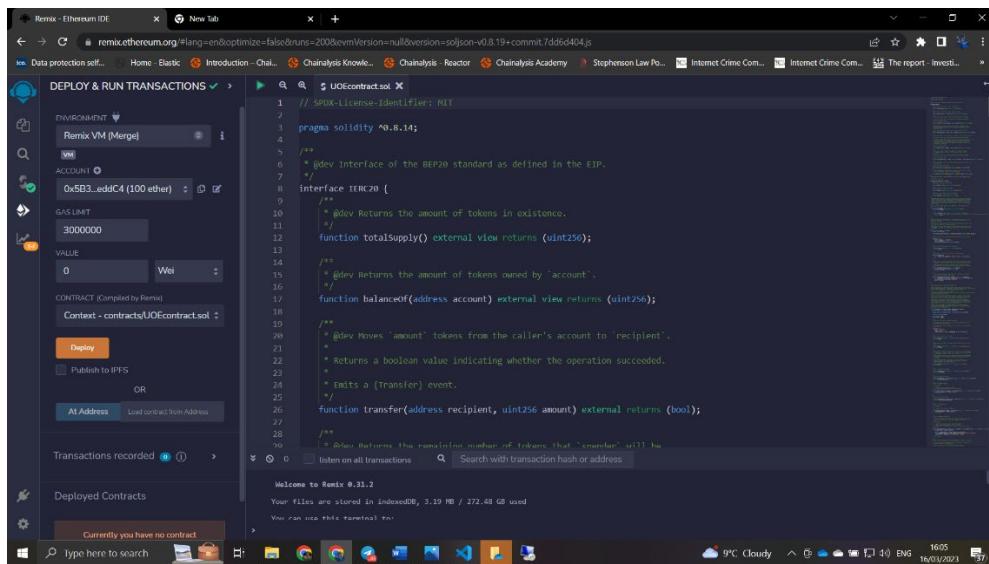
    /**
     * @dev Returns the minimum number of tokens that "spender" will be
     * allowed to spend on behalf of "owner". Returns a uint256 value.
     */
}
```

42. Click on ‘Compile UOEcontract.sol’ and you will see a green tick at the left hand side in the tab bar like shown in the image below.



The screenshot shows the Remix Ethereum IDE interface. On the left, there's a sidebar titled 'SOLIDITY COMPILER' with options like 'COMPILE', 'Auto compile', 'Advanced Configurations', and a prominent blue button labeled 'Compile UOEcontract.sol'. Below this is a 'CONTRACT' section with 'Context (UOEcontract.sol)' and buttons for 'Publish on IPFS' and 'Publish on Swarm'. At the bottom of the sidebar are 'Compilation Details' sections for 'ABI' and 'Bytecode'. The main area contains the Solidity code for 'UOEcontract.sol'. The status bar at the bottom indicates 'Welcome to Remix 0.31.2' and shows system information like '9PC Cloudy' and the date '16/01/2023'.

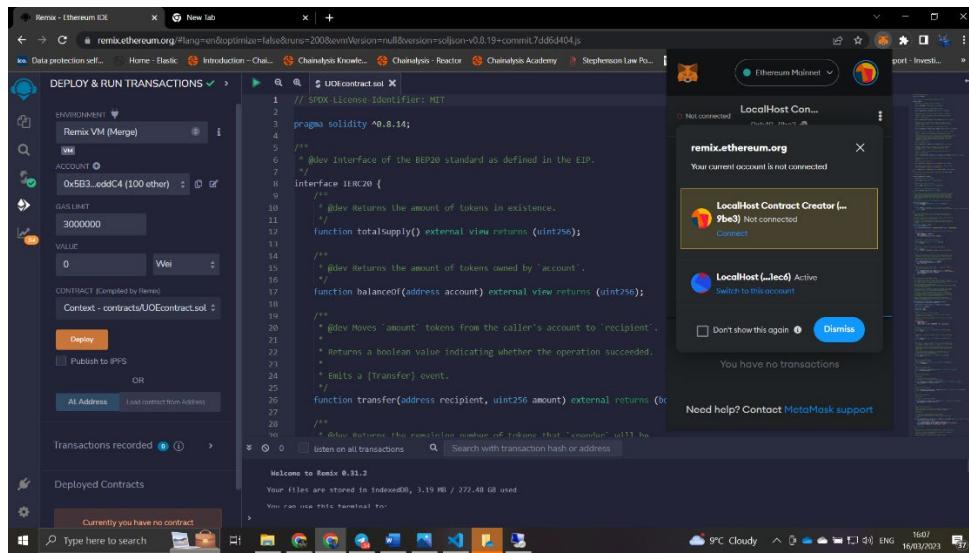
43. Navigate to the ‘Deploy & run transactions’ tab.



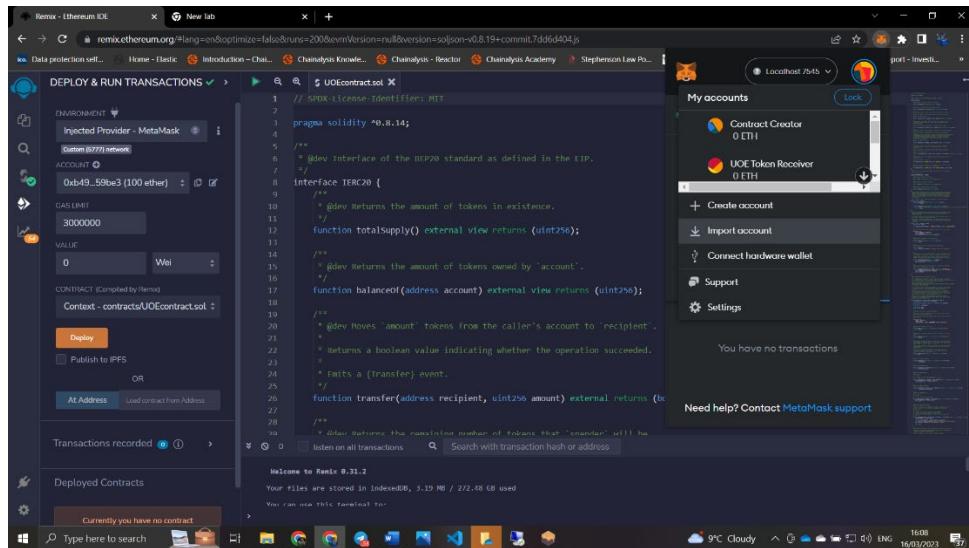
The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' tab in the Remix Ethereum IDE. On the left, there's a sidebar with 'ENVIRONMENT' set to 'Remix VM (Merge)', 'ACCOUNT' set to '0x5B3...addC4 (100 ether)', 'GAS LIMIT' set to '3000000', and a 'VALUE' field set to '0 Wei'. Below this is a 'CONTRACT' section with 'Context (Compiled by Remix)' and a 'Deploy' button. There are also buttons for 'Publish to IPFS' and 'At Address'. The main area contains the same Solidity code as the previous screenshot. The status bar at the bottom indicates 'Welcome to Remix 0.31.2' and shows system information like '9PC Cloudy' and the date '16/01/2023'.

44. Select ‘Injected Provide – Metamask’ from the ‘Enviroment’ dropdown menu.

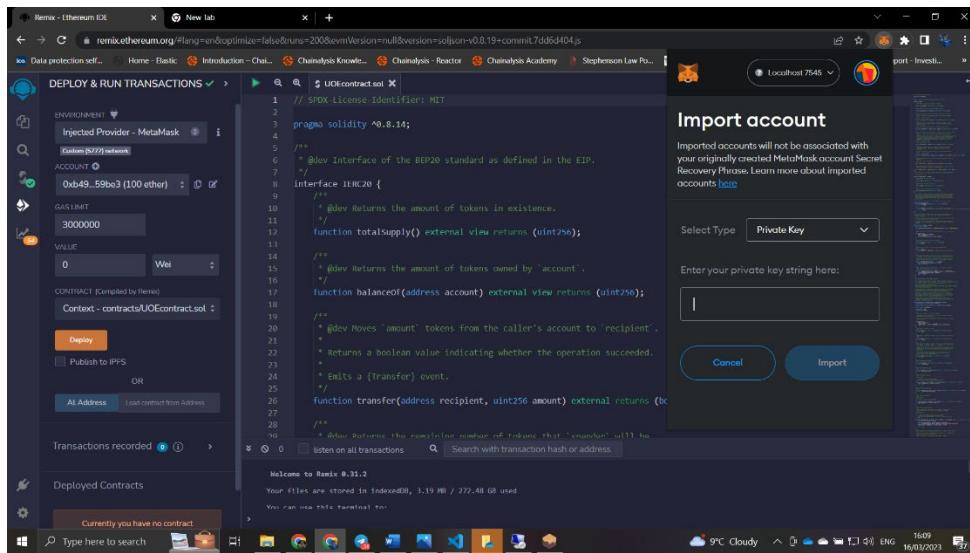
45. Connect your Metamask wallet to RemixIDE



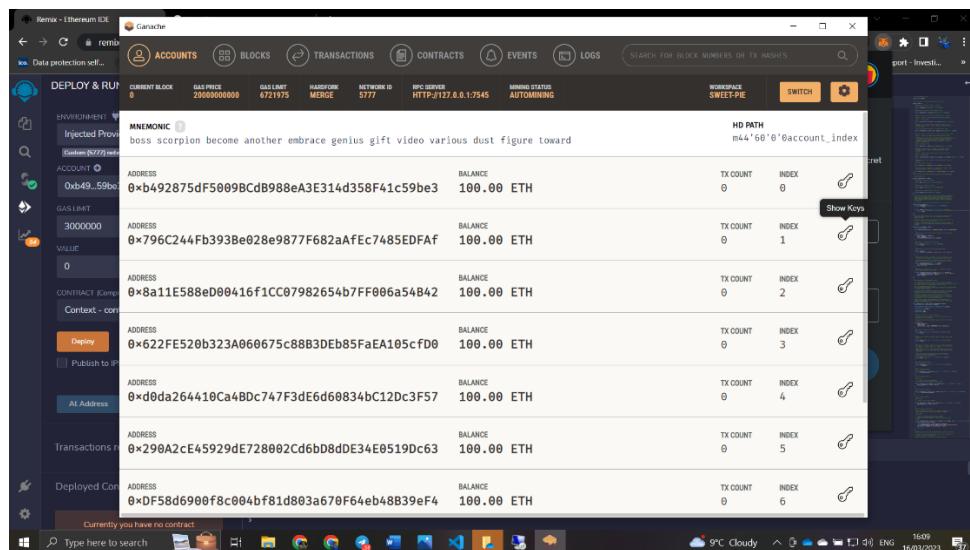
46. Open the Metamask add-on and click on the coloured circle in the top right to reveal a dropdown menu.



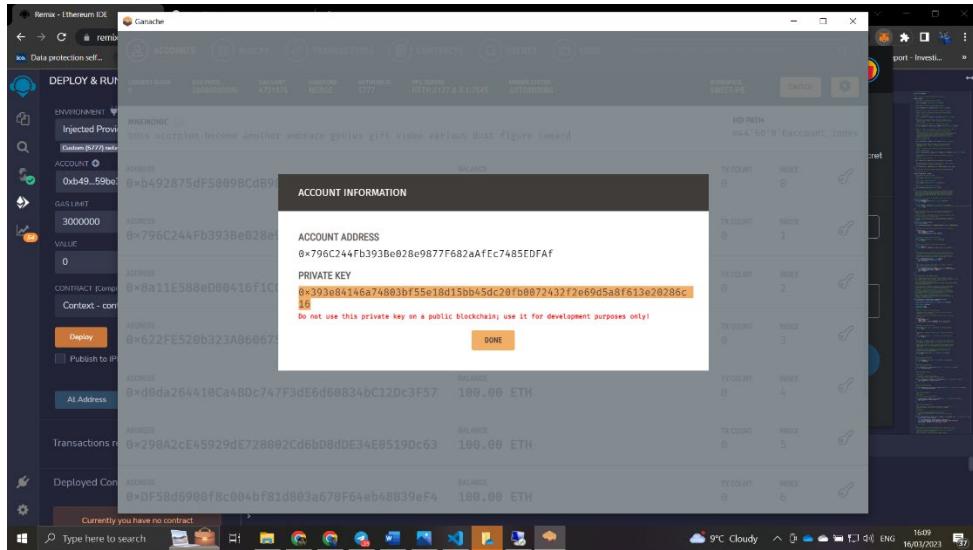
47. Now select 'Import Account' and you will see a screen similar to the one in the image below.



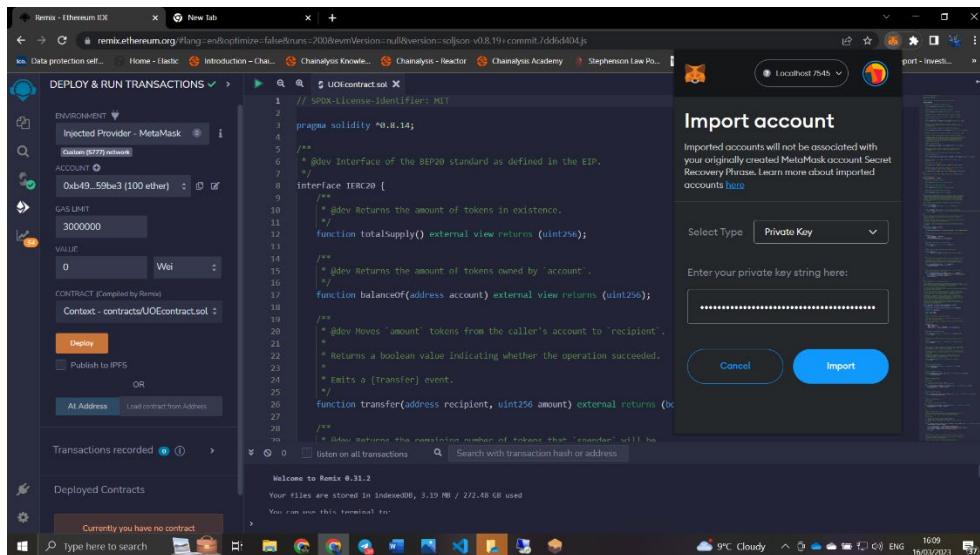
48. Open the Ganache instance we set up earlier and select a key icon.



49. You will be shown a private key, copy that private key.



50. Then back to the Metamask browser add-on and paste the private key like in the image below.



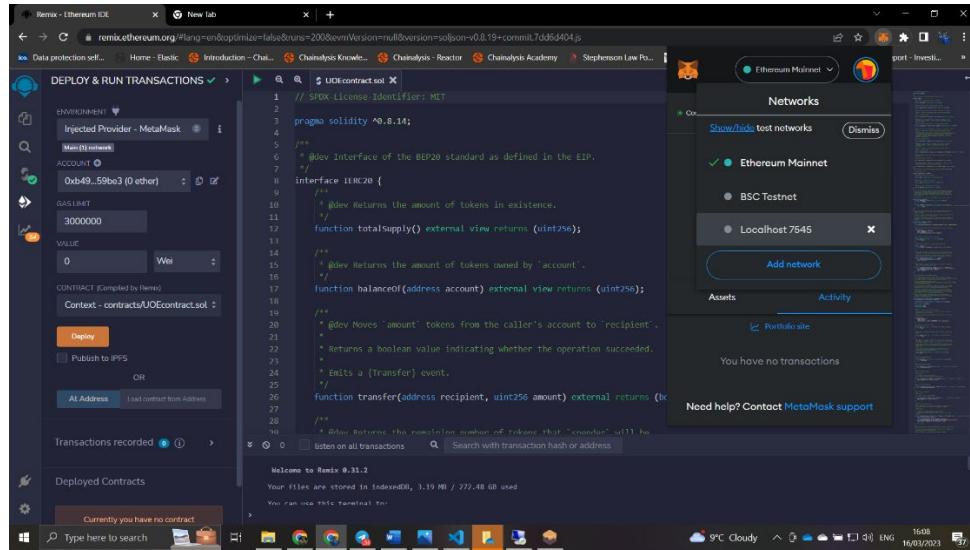
51. This has imported the account you choose with a balance of 100 ETH.

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing the environment set to 'Injected Provider - MetaMask', account '0xb49...59be3 (100 ether)', gas limit '3000000', and value '0 Wei'. Below this, the code editor contains the UOEContract.sol file. The right side of the interface displays the account details for 'Account 5' (0x796...EDFAF), which has a balance of '100 ETH'. The 'Assets' tab is selected, showing three blue circular icons for 'Buy', 'Send', and 'Swap'. Below the account summary, it says 'You have no transactions' and 'Need help? Contact MetaMask support'. At the bottom, the status bar shows 'Welcome to Remix 0.3.12', 'Your files are stored in indexDB, 3.19 MB / 272.48 GB used', and the system status '9°C Cloudy'.

52. Now connect this account to RemixIDE.

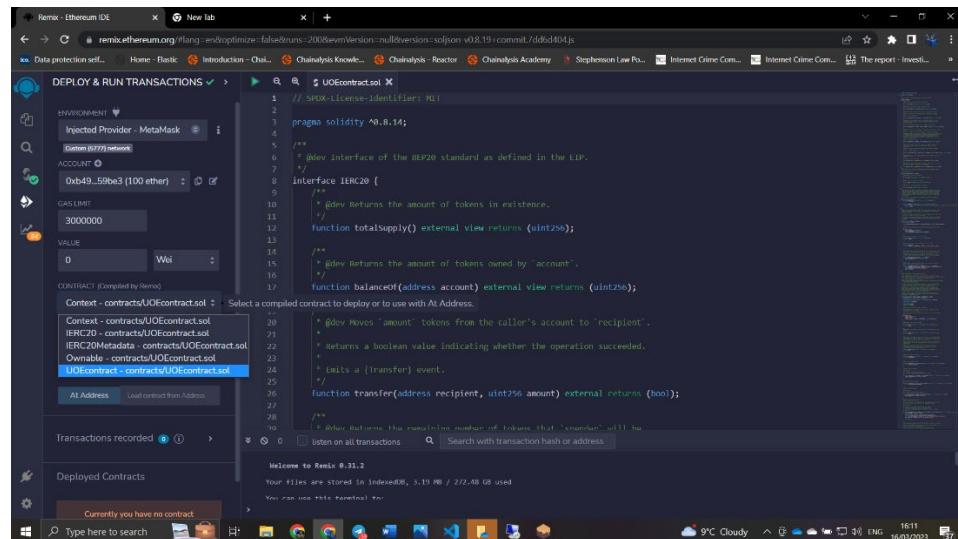
This screenshot is identical to the previous one, showing the Remix Ethereum IDE interface. The left sidebar shows the same deployment parameters and the UOEContract.sol code. The right side shows 'Account 5' (0x796...EDFAF) with a balance of '100 ETH'. The 'Assets' tab is selected, showing the 'Buy', 'Send', and 'Swap' icons. The status bar at the bottom indicates a connection status of 'Connected'.

53. Click on where it says ‘Ethereum Mainnet’ and select ‘LocalHost 7545’

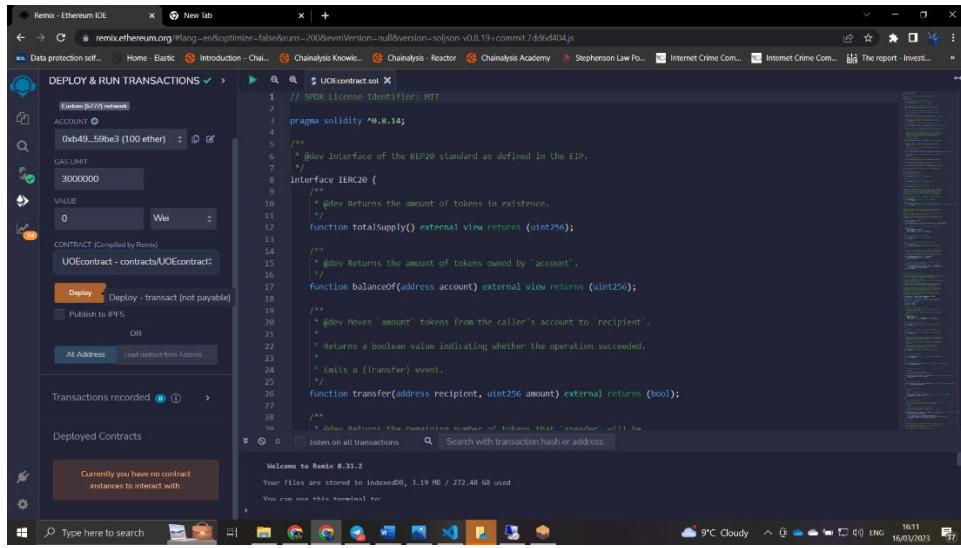


54. Congratulations you are now connected to the Ganache blockchain you set up.

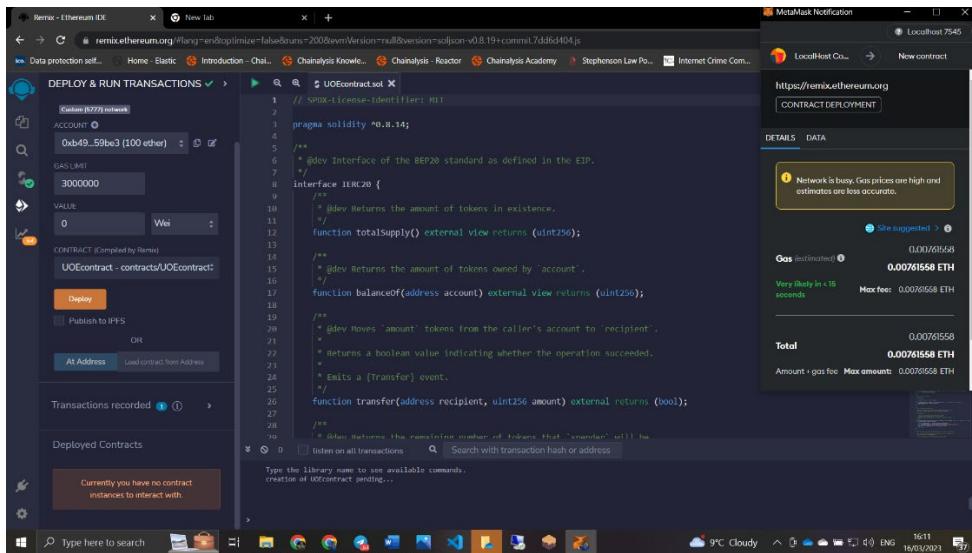
55. In the dropdown ‘Contract’ menu make sure the file selected to deploy is ‘UOEcontract – contracts/UOEcontract.sol’.



56. Click on the orange ‘deploy’ button.



57. The Metamask add-on should open with a transaction to confirm, in this case it is ‘Contract Deployment’ so is only subject to gas fees as shown below. It is important to note that checking the information when signing, make sure you know what you are interacting with. Click the ‘Confirm’ button.



58. Congratulations you have just deployed a smart contract to a local development blockchain.

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is open, showing an account balance of 3000000 Wei. The main area displays the Solidity code for the UOEContract, which is a BEP20 token implementation. The code includes functions for totalSupply, balanceOf, and transfer. A transaction record is visible at the bottom left, showing a deployment from address 0xb49...59be3 to UOEcontract at 0xKEC_0500E. The right side of the interface shows the Ganache blockchain explorer with several accounts and their balances.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

/*
 * @dev Interface of the BEP20 standard as defined in the EIP.
 */
interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

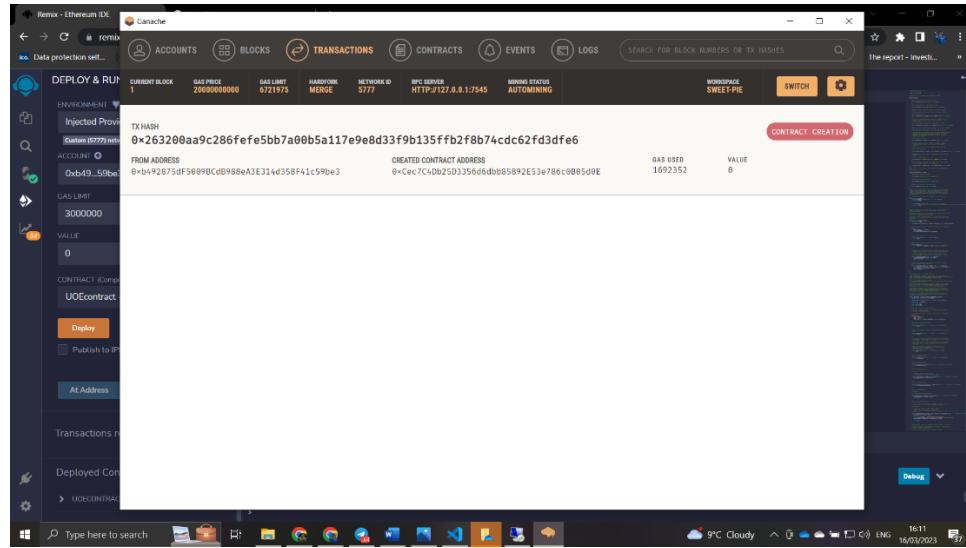
    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `amount` must be less than or equal to the caller's balance.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the resulting number of tokens that `spender` will be
     * allowed to spend on behalf of `owner`. This value must be less than or equal to
     * the allowance specified in the approved `spender` account.
     */
}
```

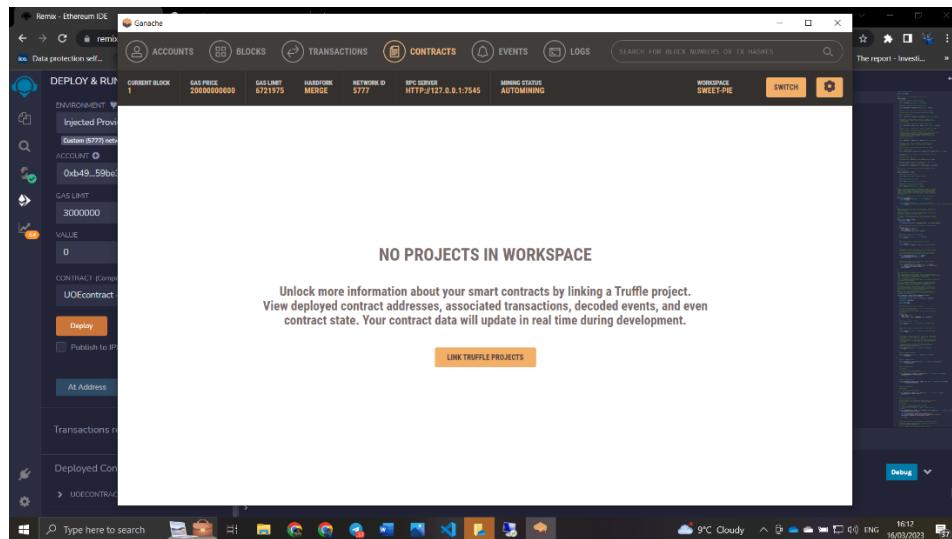
59. If you open the Ganache instance you have running you can see that the wallet you imported has a balance lower than 100 ETH and that the account now has one transaction.

The screenshot shows the Ganache blockchain explorer. The 'ACCOUNTS' tab is selected, displaying a list of accounts with their addresses and balances. One account, 0xb492875dF5009BCdB988eA3E314d358F41c59be3, has a balance of 99.99 ETH. The 'TRANSACTIONS' tab shows a single transaction record for the deployment of the UOEContract at block 1. The transaction details are: from: 0xb49...59be3 to: UOEcontract.(constructor) value: 0 wei data: 0x608...30833 logs: 2 hash: 0x0d6...3bab. The right side of the interface shows the Ganache blockchain explorer with several accounts and their balances.

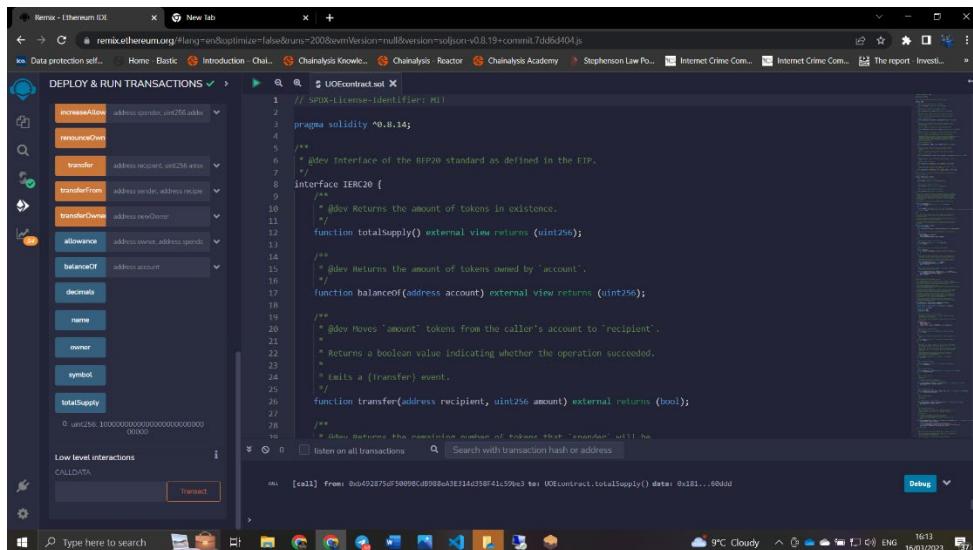
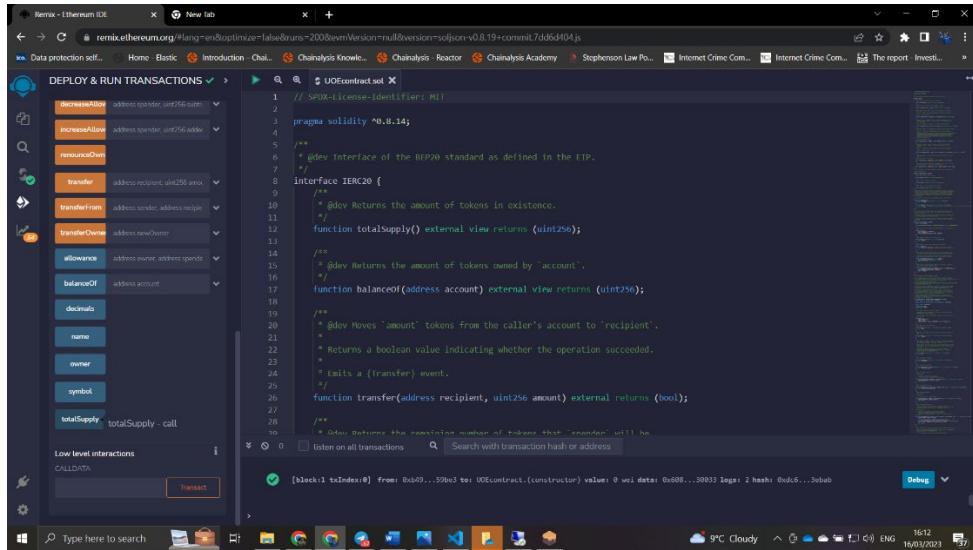
60. If you click on the ‘transactions’ tab at the top of the Ganache window you can see that the transaction was for a contract creation.



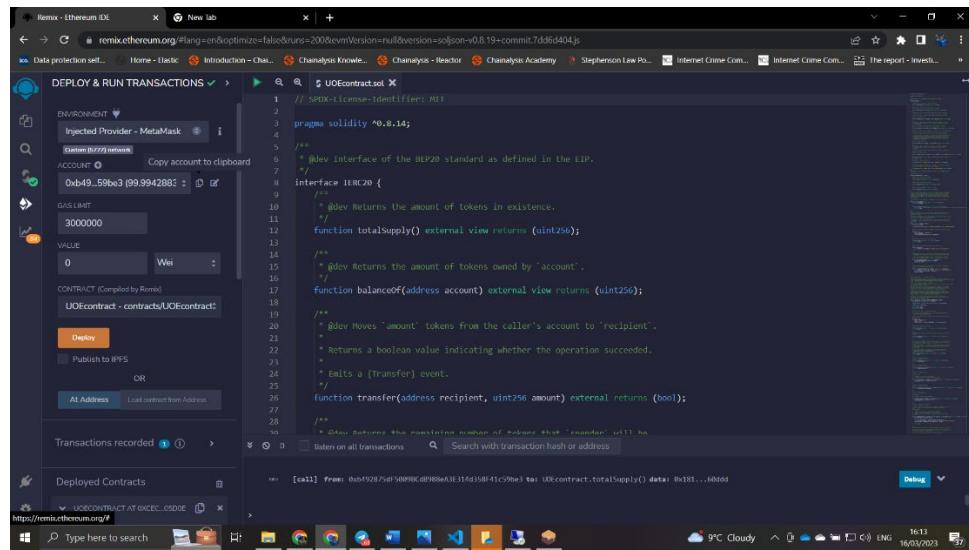
61. If we were using the [Truffle](#) framework we would have a lot more information to see.



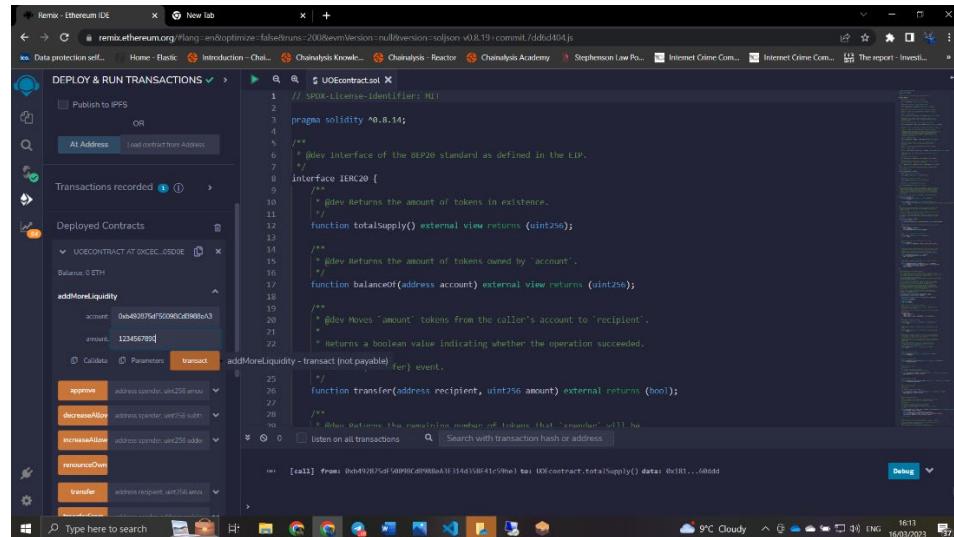
62. For this exercise we will go back to RemixIDE and select the newly deployed contract to see the options available.



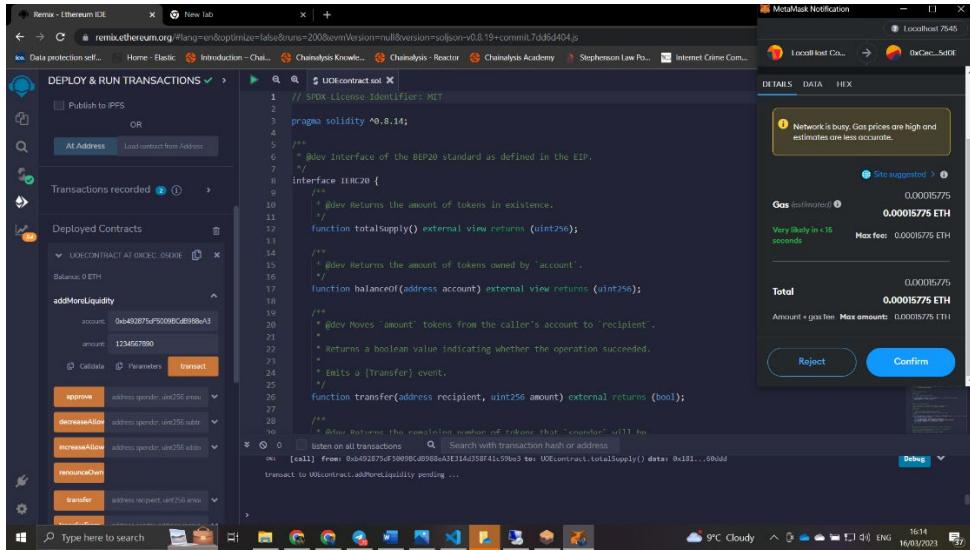
64. Scroll back to the top of this section and copy the ‘Account’ to your clipboard.



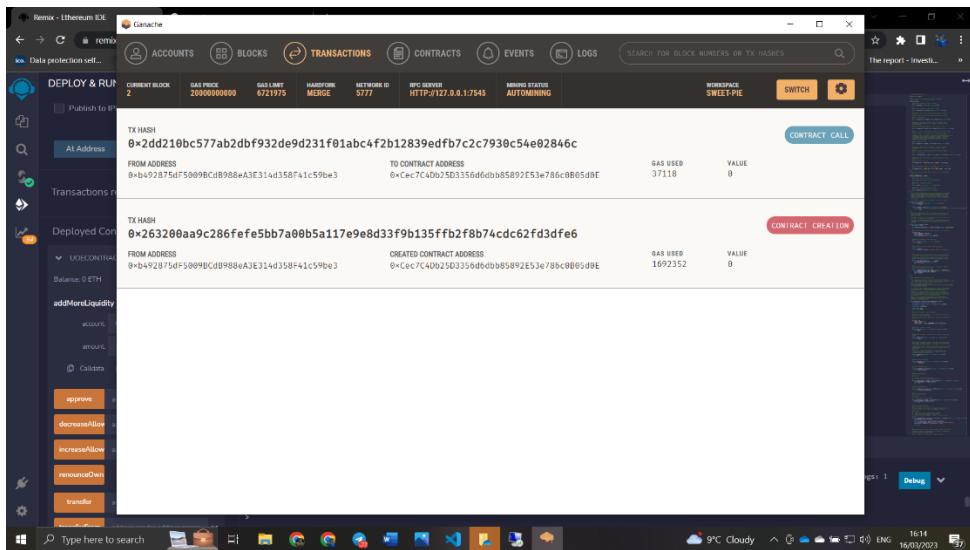
65. In the ‘Deployed Contracts’ section, expand the ‘addMoreLiquidity’ function and paste the copied account into the ‘account’ field. Then put a random value in the ‘amount’ field. I used ‘1234567890’ to start.



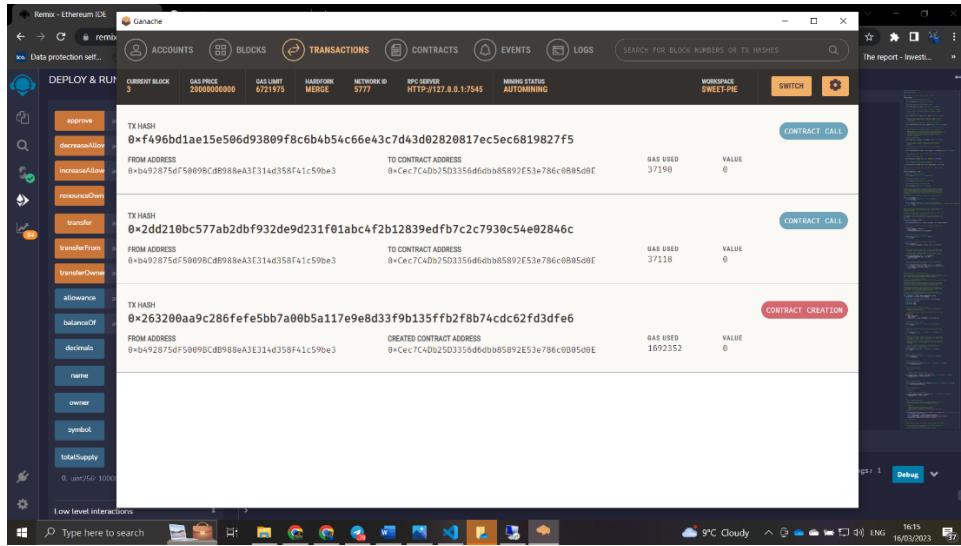
66. Click the orange 'Transact' button and the Metamask add-on will open requesting a transition signature. Click the 'Confirm' Button.



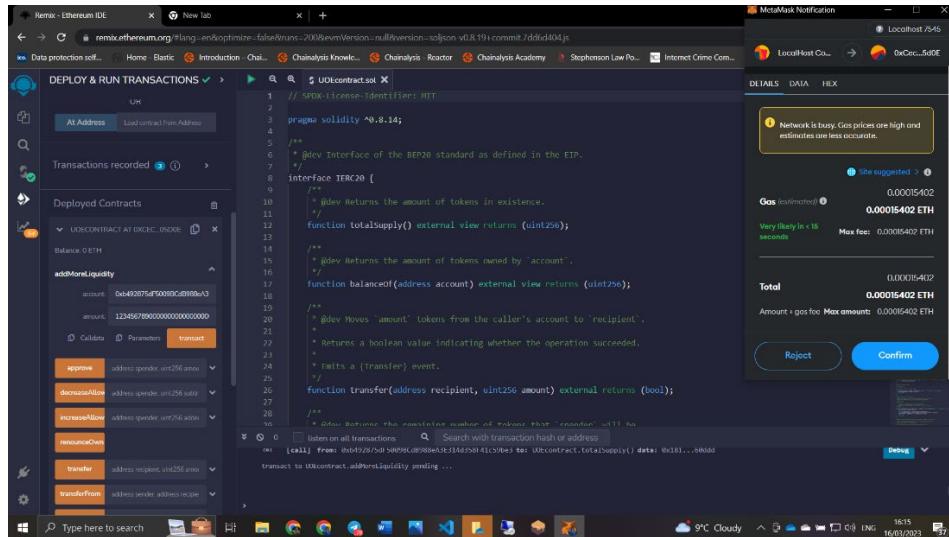
67. If you open up the Ganache application again, in the 'transactions' tab you will see another Tx Hash for a 'Contract Call'.



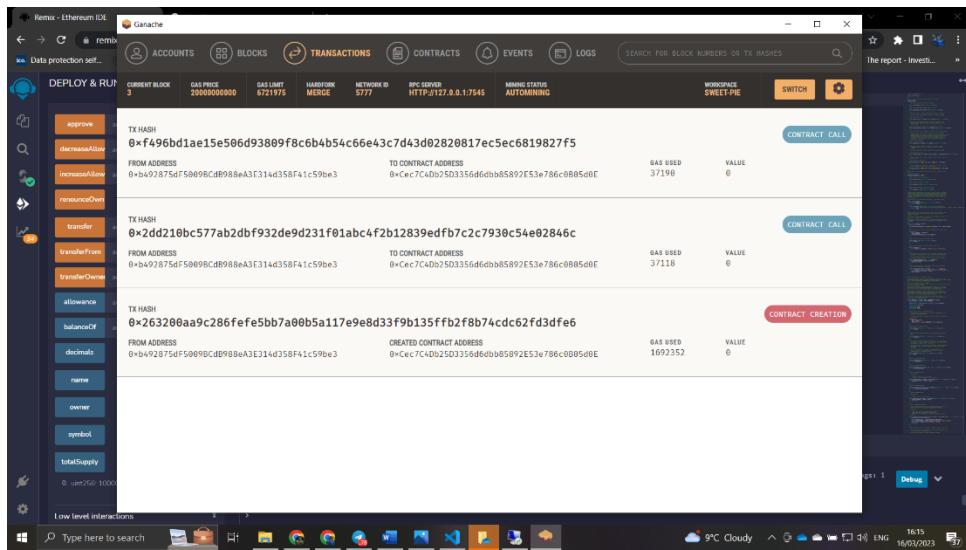
68. Go back to RemixIDE and call the blue 'totalSupply' function button to see what value is returned now.



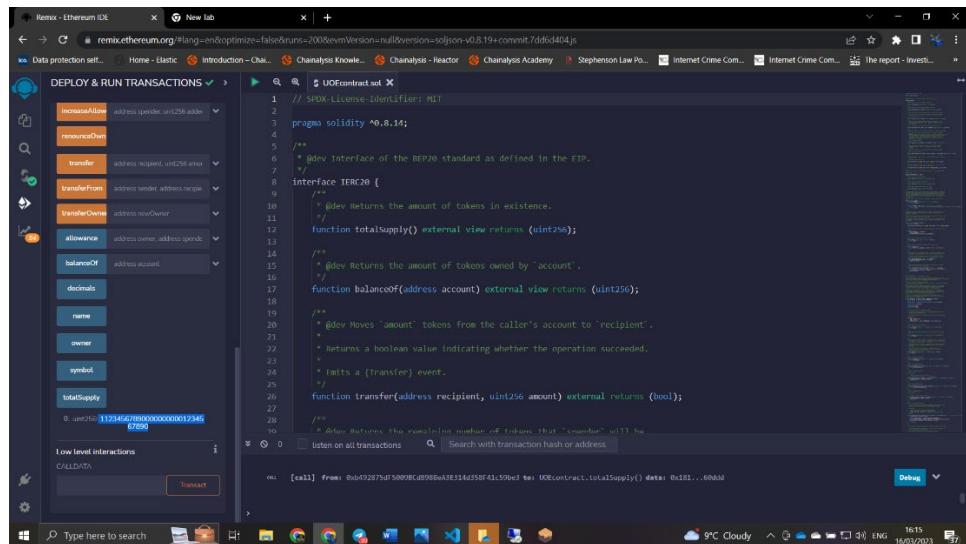
69. To double check the 'addMoreLiquidity' function is able to mint again go back to the function and add as many extra zeros to the end of the 'amount' value. Then click the orange 'Transact' button to open the Metamask signature request again, like in the image below.



70. Open up Ganache again to see another Tx Hash for a ‘Contract Call’.



71. Go back to RemixIDE and call the blue ‘totalSupply’ function button to see what value is returned now. You will see it has increased again confirming this is a hidden mint function.



72. Congratulations you have successfully completed this exercise.

Conclusion

In conclusion, this exercise has provided a comprehensive walkthrough on the basics of Ethereum smart contracts and Solidity programming and analysis. You have learned how to set up a local blockchain network, create and compile a smart contract, and analyse its functions using Solidity Visual Studio. You have also learned how to connect to your Metamask wallet, deploy a smart contract using RemixIDE, and test its functions. By using the additional open-source tools, you can further improve your smart contract analysis skills. Finally, if you're interested in learning more about blockchain technology and its applications, you can check out the Chainalysis Training Academy for more resources.

[Slither](#)

[ConsenSys Security Tools](#)

[Chainalysis Training Academy](#)

[Cbsecurity.solutions](#)