

# A hands-on tutorial: Working with Smart Contracts in Ethereum

Was prepared with the assistance of Mohammad H. Tabatabaei from  
the University of Oslo



UNIVERSITY OF  
TORONTO



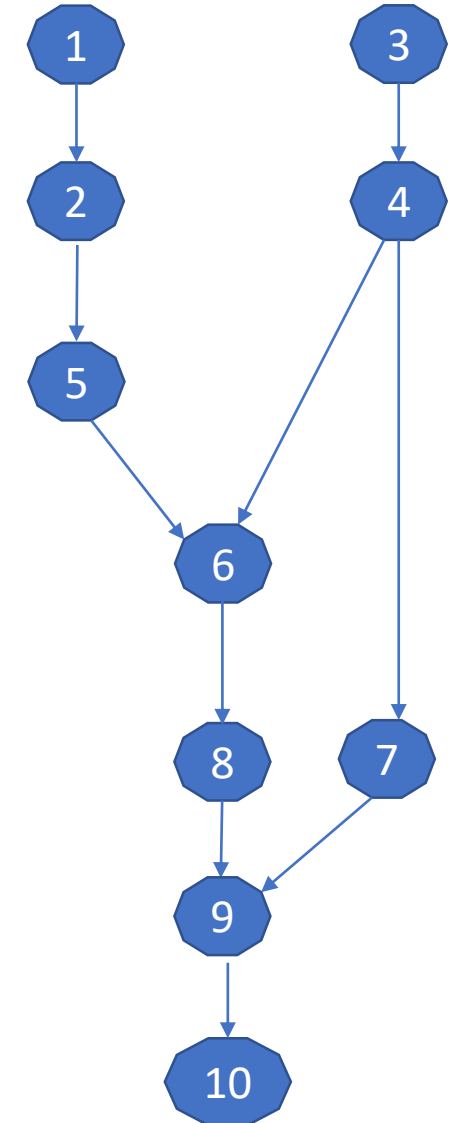
Le génie pour l'industrie



UiO • University of Oslo

# Different tools provide different functionality

Activities	Tools				
	Remix	Ganache	MyEtherWallet	Geth	
1	Configuring the Blockchain	-	-	-	+
2	Deploying the Blockchain	Not Persistent	+	-	+
3	Developing the contract	+	-	-	+
4	Compiling the contract	+	-	-	+
5	Creating user account	+	+	+	+
6	Deploying the contract	+	-	+	+
7	Creating the UI for interacting	+	-	+	+
8	Run the client	+	-	+	+
9	Interact with the contract & have fun	+	-	+	+
10	Monitoring the execution	-	+	-	+



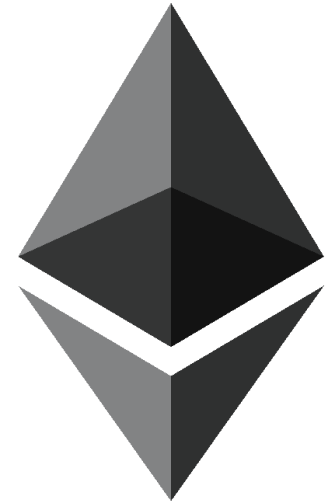
<https://remix.ethereum.org/>

<http://truffleframework.com/ganache/>

<https://github.com/kvhnuke/etherwallet/releases/tag/v3.21.06>

# Use which tool for what purpose? (1/2)

- Use Geth for everything?
  - Powerful but command-line only
- What should I use?
  - For developing contracts – mostly Remix
- What cannot Remix do?
  - Configure the blockchain
  - Create real (non-test) user accounts and transfer funds between user accounts
  - Monitor the execution
  - Other advanced operations



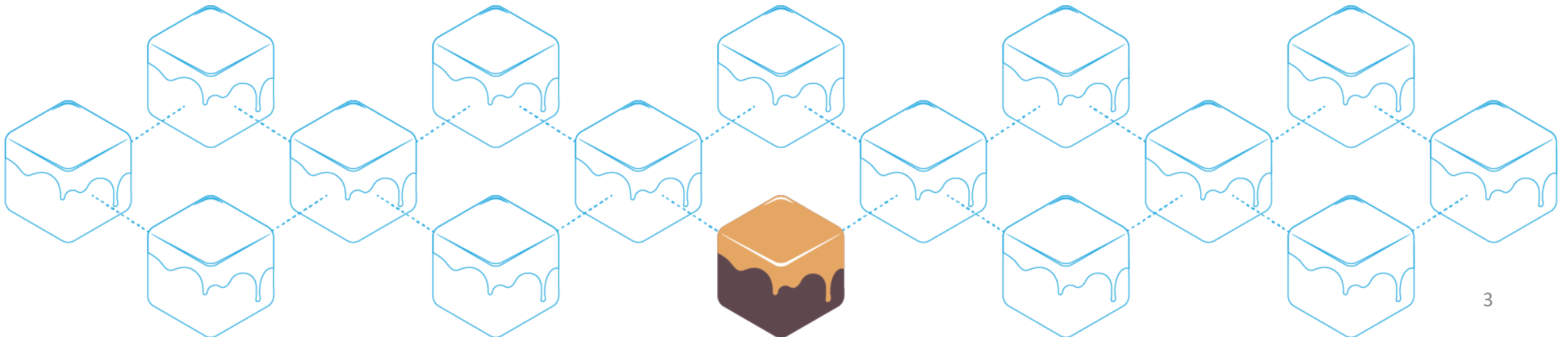
# Use which tool for what purpose? (2/2)

- Why use Ganache?

- To inspect and monitor the execution
- To visualize certain elements in a better way

- Why use MyEtherWallet?

- To create a personal wallet (real user account) and transfer funds between user accounts

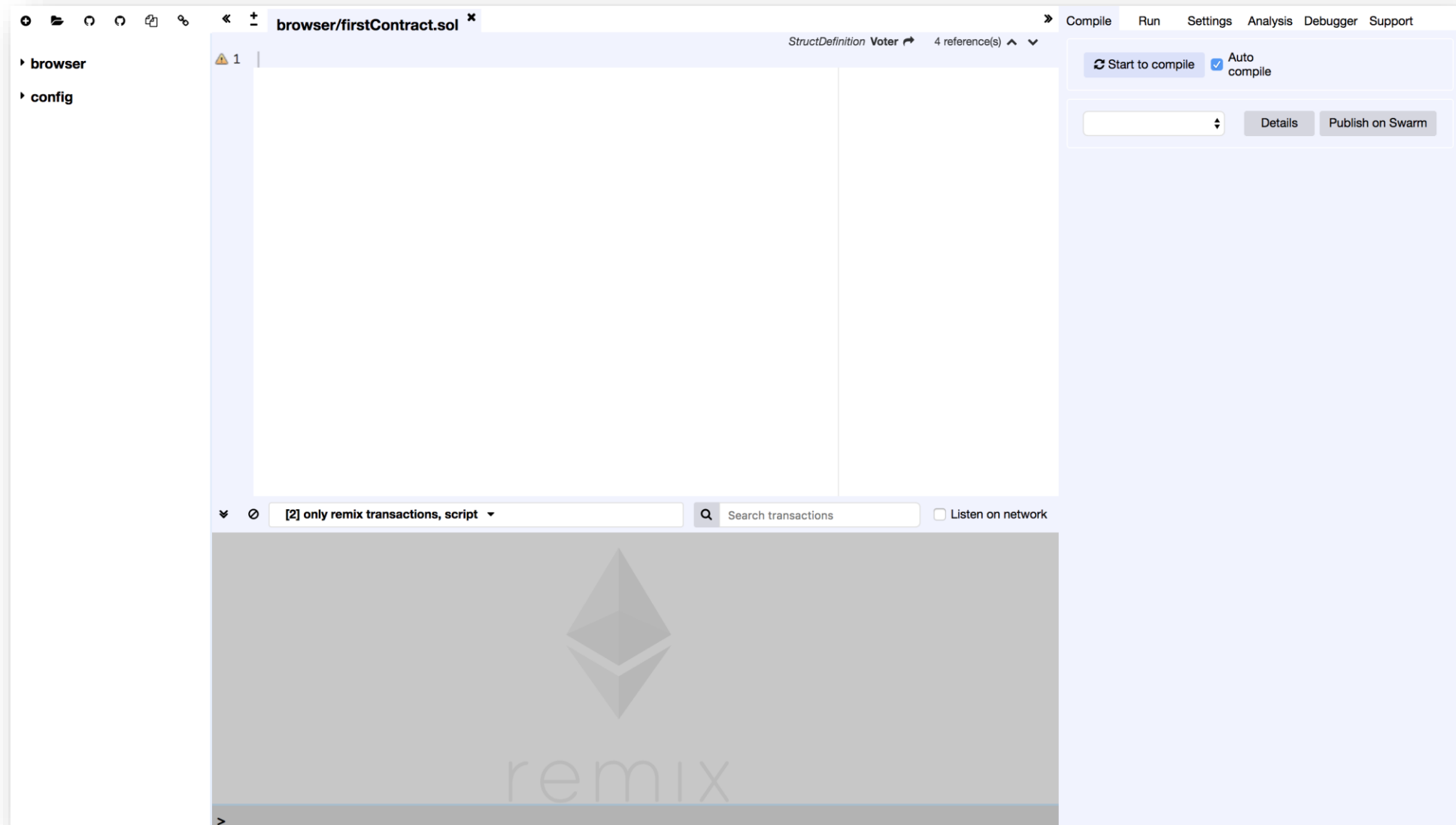


# Smart Contracts

1. Developing a simple contract
2. Compiling the contract
3. Deploying the contract
4. Interacting with the contract
5. Adding more functions to our code to make it more practical

# Open Remix : remix.ethereum.org

- An open source tool for writing, compiling and testing Solidity contracts



# Start Coding

- Setter and Getter: Set and get the information.

```
1  pragma solidity ^0.4.0;
2
3  contract financialContract{
4
5      uint amount = 13;
6
7      function getValue() constant returns(uint){
8          return amount;
9      }
10
11     function setValue(uint newValue) {
12         amount = newValue;
13     }
14
15 }
```

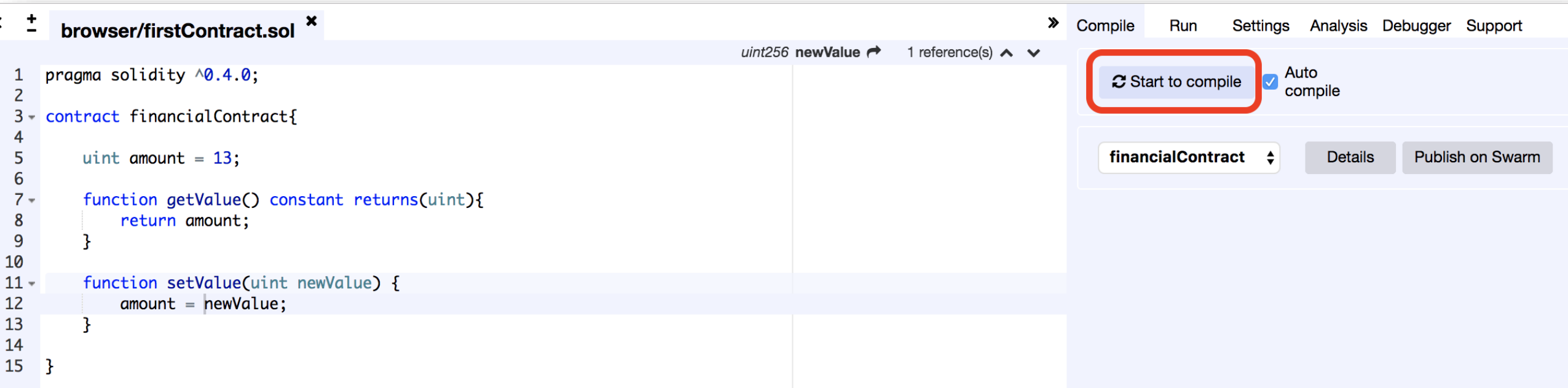
Variable

Getter function

Setter function

# Compile the Contract

- Compile tab: Start to compile button



The screenshot shows a web-based IDE interface for compiling a Solidity contract. The main editor displays the following code:

```
1 pragma solidity ^0.4.0;
2
3 contract financialContract{
4
5     uint amount = 13;
6
7     function getValue() constant returns(uint){
8         return amount;
9     }
10
11     function setValue(uint newValue) {
12         amount = newValue;
13     }
14
15 }
```

The right-hand side of the interface features a navigation menu with the following options: Compile, Run, Settings, Analysis, Debugger, and Support. The 'Compile' tab is active. In this tab, there is a 'Start to compile' button with a refresh icon, which is highlighted with a red rectangle. To its right is a checked checkbox labeled 'Auto compile'. Below these controls, there is a dropdown menu showing 'financialContract', a 'Details' button, and a 'Publish on Swarm' button.



# Set Environment (1/2)

- Run tab: Environment = JavaScript VM

The screenshot shows a Solidity IDE interface. On the left, a code editor displays the following Solidity code:

```
1 pragma solidity ^0.4.0;
2
3 contract financialContract{
4     uint amount = 13;
5
6     function getValue() constant returns(uint){
7         return amount;
8     }
9
10
11 function setValue(uint newValue) {
12     amount = newValue;
13 }
14
15 }
```

On the right, the 'Run' tab is active, showing deployment settings. The 'Environment' dropdown is set to 'JavaScript VM' and is highlighted with a red box. Below it, the 'Account' dropdown is set to '0x147...c160c (100 ether)'. The 'Gas limit' is set to '3000000' and the 'Value' is set to '0' with the unit 'wei'. A dropdown menu shows the selected contract 'financialContract'. Below this is a 'Deploy' button. At the bottom, there is a 'Load contract from Address' input field and an 'At Address' button. The status bar at the bottom indicates '0 pending transactions' and includes icons for saving, running, and deleting.

# Set Environment (2/2)

- JavaScript VM: All the transactions will be executed in a sandbox blockchain in the browser. Nothing will be persisted and a page reload will restart a new blockchain from scratch, the old one will not be saved.
- Injected Provider: Remix will connect to an injected web3 provider. Mist and Metamask are example of providers that inject web3, thus can be used with this option.
- Web3 Provider: Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.
- Gas Limit: The maximum amount of gas that can be set for all the transactions of a contract.
- Value: The amount of value for the next created transaction (wei =  $10^{-18}$  of ether).

# Types of Blockchain Deployment

- Private: e.g., Ganache sets a personal Ethereum blockchain for running tests, executing commands, and inspecting the state while controlling how the chain operates.
- Public Test: Like Ropsten, Kovan and Rinkeby which are existing public blockchains used for testing and which do not use real funds.
- Public Real: Like Bitcoin and Ethereum which are used for real and which available for everybody to join.

# Deploy the Contract on the Private Blockchain of Remix

- Run tab: Deploy button

The screenshot displays the Remix IDE interface. On the left, a code editor shows a Solidity contract named `financialContract` with the following code:

```
1 pragma solidity ^0.4.0;
2
3 contract financialContract{
4
5     uint amount = 13;
6
7     function getValue() constant returns(uint){
8         return amount;
9     }
10
11     function setValue(uint newValue) {
12         amount = newValue;
13     }
14 }
15 }
```

The right-hand side of the interface is the 'Run' tab, which contains deployment settings and a list of deployed contracts. The 'Deploy' button is highlighted with a red box. The settings are as follows:

- Environment: JavaScript VM
- Account: 0x147...c160c (99.9999999999998675)
- Gas limit: 3000000
- Value: 0 wei

Below the settings, the contract `financialContract` is selected, and the `Deploy` button is visible. Below the deployment section, there is a section for pending transactions, which currently shows `0 pending transactions`. At the bottom, there is a section for deployed contracts, showing `financialContract at 0x0fd...bcfb7 (memory)` with a dropdown menu containing `setValue` and `getValue`.

At the bottom of the interface, there is a search bar for transactions, a checkbox for 'Listen on network', and a status indicator showing `[2] only remix transactions, script`.

# Interact with the Contract

- Setter = Red Button: Creates transaction
- Getter= Blue Button: Just gives information

financialContract at 0x0fd...bcfb7 (memory)

setValue uint256 newValue

getValue

0: uint256: 13

Press getValue to see the initial amount

1

financialContract at 0x0fd...bcfb7 (memory)

setValue 50

getValue

0: uint256: 50

Input a value and press setValue button to create and confirm the transaction

2

Press getValue again to see the result

3

# Additional features

- Saving the address of the contract creator
- Limiting the users' access to functions
- Transferring funds from an account to the contract
- Withdrawing funds from the contract to an account

# Constructor

- A function with the name of the contract
- Will be called at the creation of the instance of the contract

```
1  pragma solidity ^0.4.0;
2
3  contract financialContract{
4
5      uint amount;
6      address issuer;
7
8  function financialContract(){
9      issuer = msg.sender;
10 }
11
12 function getValue() constant returns(uint){
13     return amount;
14 }
15
16 function setValue(uint newValue) {
17     amount = newValue;
18 }
19
20 }
```

We want to save  
the address of the  
contract creator

# Modifier

- Conditions you want to test in other functions
- First the modifier will execute, then the invoked function

```
1 pragma solidity ^0.4.0;
2
3 contract financialContract{
4
5     uint amount;
6     address issuer;
7
8     function financialContract(){
9         issuer = msg.sender;
10    }
11
12    modifier ifIssuer(){
13        if(issuer != msg.sender){
14            throw;
15        }else{
16            -;
17        }
18    }
19
20    function getValue() constant returns(uint){
21        return amount;
22    }
23
24    function setValue(uint newValue) ifIssuer {
25        amount = newValue;
26    }
27
28 }
```

Only the contract creator is permitted to set value



# Receive ether (1/2)

- Transfer money to the contract

Payable keyword  
allows receiving  
ether

We can get the  
balance of the  
contract

```
1  pragma solidity ^0.4.0;
2
3  contract financialContract{
4
5      address issuer;
6
7      function financialContract(){
8          issuer = msg.sender;
9      }
10
11     modifier ifIssuer(){
12         if(issuer != msg.sender){
13             throw;
14         }else{
15             -;
16         }
17     }
18
19     function receiveFunds() payable{
20
21     }
22
23     function getValue() constant returns(uint){
24         return this.balance;
25     }
26
27 }
```

# Receive ether (2/2)

1

Input the value as wei  
( $10^{-18}$  of ether)

Environment JavaScript VM VM (-) i

Account 0x147...c160c (99.9999999999998311€) 📄 ⊕

Gas limit 3000000

Value 100| wei ▾

financialContract ▾

Deploy

Load contract from Address

At Address

0 pending transactions



financialContract at 0x1df...bda71 (memory) 📄 ×

receiveFunds

setValue uint256 newValue ▾

getValue


2

Click the receiveFunds button to  
transfer the money to the  
contract

# Withdraw funds

- Transfer ether from the contract to the user account

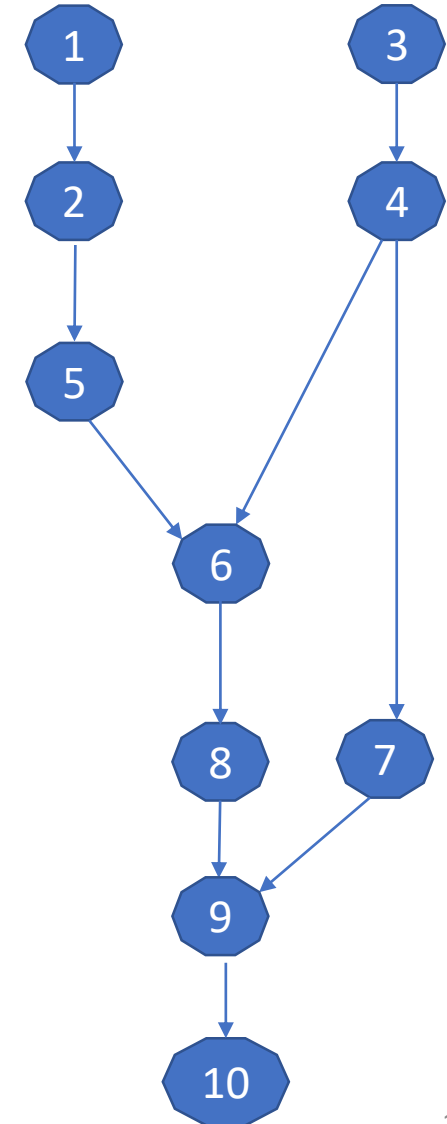
Transfer some money from the contract to the mentioned account



```
1  pragma solidity ^0.4.0;
2
3  contract financialContract{
4
5      address issuer;
6
7      function financialContract(){
8          issuer = msg.sender;
9      }
10
11     modifier ifIssuer(){
12         if(issuer != msg.sender){
13             throw;
14         }else{
15             -;
16         }
17     }
18
19     function receiveFunds() payable{
20
21     }
22
23     function getValue() constant returns(uint){
24         return this.balance;
25     }
26
27     function withdrawFunds(uint funds) ifIssuer{
28         issuer.send(funds);
29     }
30
31 }
```

# Now deploying a smart contract on an external blockchain

	Tools				
	Activities	Remix	Ganache	MyEtherWallet	Geth
1	Configuring the Blockchain	-	-	-	+
2	Deploying the Blockchain	Not Persistent	+	-	+
3	Developing the contract	+	-	-	+
4	Compiling the contract	+	-	-	+
5	Creating user account	+	+	+	+
6	Deploying the contract	+	-	+	+
7	Creating the UI for interacting	+	-	+	+
8	Run the client	+	-	+	+
9	Interact with the contract & have fun	+	-	+	+
10	Monitoring the execution	-	+	-	+



# Run Ganache

Ganache

ACCOUNTS BLOCKS TRANSACTIONS LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK 0	GAS PRICE 20000000000	GAS LIMIT 6721975	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING
--------------------	--------------------------	----------------------	--------------------	-------------------------------------	-----------------------------

MNEMONIC ? slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there

HD PATH m/44'/60'/0'/0/account\_index

ADDRESS	BALANCE	TX COUNT	INDEX	
0x231eAeEF9EA93F5370a1F633F32E45AF570980E8	100.00 ETH	0	0	🔑
0x970fc818790E900598C57E48b89B6D3D8896D416	100.00 ETH	0	1	🔑
0xb59BD5568d0be42C13fB521f845243F1CDaF2eF1	100.00 ETH	0	2	🔑

# MyEtherWallet

- add your custom network that you want to test your contracts on

The screenshot shows the MyEtherWallet interface. At the top, there is a navigation bar with the MyEtherWallet logo, version 3.21.05, language set to English, gas price of 41 Gwei, and the current network set to ETH (myetherapi.com). Below the navigation bar, there are links for 'New Wallet', 'Send Ether & Tokens', 'Swap', 'Send Offline', 'Contracts', 'ENS', 'DomainSale', 'Check TX Status', 'View Wallet Info', and 'Help'. The main content area is titled 'Create New Wallet' and contains a form with the heading 'Enter a password'. The form has a text input field with the placeholder text 'Do NOT forget to save this!' and a 'Create New Wallet' button. Below the form, there is a warning message: 'This password encrypts your private key. This does not act as a seed to generate your keys. You will need this password + your private key to unlock your wallet.' and links for 'How to Create a Wallet' and 'Getting Started'. On the right side, a dropdown menu is open, showing a list of networks. The 'Add Custom Network / Node' option at the bottom of the list is circled in red, and a red arrow points to it from the top right of the screen.

# Import your RPC server address and the port number from Ganache to MyEtherWallet

The image shows a screenshot of the Ganache application interface. The top navigation bar includes 'ACCOUNTS', 'BLOCKS', 'TRANSACTIONS', and 'LOGS'. Below this, a status bar displays 'CURRENT BLOCK 0', 'GAS PRICE 20000000000', 'GAS LIMIT 6721975', 'NETWORK ID 5777', 'RPC SERVER HTTP://127.0.0.1:7545', and 'MINING STATUS AUTOMINING'. The 'RPC SERVER' field is circled in red. A dialog box titled 'Set Up Your Custom Node' is overlaid on the right side. It contains a 'Node Name' field with 'Private ETH Node', a 'URL' field with 'http://127.0.0.1', and a 'Port' field with '7545'. Below these fields are checkboxes for 'HTTP Basic access authentication' and radio buttons for network selection: 'ETH' (selected), 'ETC', 'Ropsten', 'Kovan', 'Rinkeby', 'Custom', and 'Supports EIP-155'. At the bottom of the dialog are 'Cancel' and 'Save & Use Custom Node' buttons. Red arrows point from the circled RPC server address in Ganache to the 'URL' and 'Port' fields in the dialog box.

**RPC SERVER**  
HTTP://127.0.0.1:7545

### Set Up Your Custom Node

Instructions can be found here

**Node Name**  
Private ETH Node

**URL**  
http://127.0.0.1

**Port**  
7545

HTTP Basic access authentication

ETH  ETC  Ropsten  Kovan  Rinkeby  Custom  Supports EIP-155

Cancel Save & Use Custom Node

# MyEtherWallet

- Contracts tab: Deploy Contract

The screenshot shows the MyEtherWallet interface. At the top, there is a dark blue header with the MyEtherWallet logo on the left. To the right of the logo, the version number '3.21.05' is displayed, followed by a language dropdown menu set to 'English'. Further right, the gas price is shown as 'Gas Price: 41 Gwei', and the network is set to 'Network My Ether Node:eth (Custom)'. Below the header, a navigation bar contains several menu items: 'New Wallet', 'Send Ether & Tokens', 'Swap', 'Send Offline', 'Contracts', 'ENS', 'DomainSale', 'Check TX Status', 'View Wallet Info', and 'Help'. The 'Contracts' menu item is circled in red, and a red arrow points from it to the 'Deploy Contract' option in the main content area. The main content area has a light gray background and contains the text 'Interact with Contract or Deploy Contract', where 'Deploy Contract' is also circled in red. Below this text, there are two input fields: 'Byte Code' and 'Gas Limit'. The 'Byte Code' field is a large, empty text area. The 'Gas Limit' field contains the value '300000'.

MyEtherWallet 3.21.05 English Gas Price: 41 Gwei Network My Ether Node:eth (Custom)  
The network is really full right now. Check Eth Gas Station for gas price to use.

New Wallet Send Ether & Tokens Swap Send Offline **Contracts** ENS DomainSale Check TX Status View Wallet Info Help

Interact with Contract or **Deploy Contract**

Byte Code

Gas Limit  
300000



# Remix

- Type your contract and compile it

The screenshot displays the Remix IDE interface. On the left, a code editor shows a Solidity contract named `financialContract` with the following code:

```
1 pragma solidity ^0.4.0;
2
3 contract financialContract{
4
5     uint amount = 13;
6
7     function getValue() constant returns(uint){
8         return amount;
9     }
10
11     function setValue(uint newValue) {
12         amount = newValue;
13     }
14
15 }
```

On the right, the **Compile** tab is active. A red box highlights the **Start to compile** button, which is accompanied by a checked **Auto compile** checkbox. Below this, a dropdown menu shows `financialContract` with a downward arrow, and two buttons labeled **Details** and **Publish on Swarm** are visible.

# Remix

Click on Details Button: access ByteCode to import it to MyEtherWallet

The screenshot displays the Remix IDE interface. On the left, a code editor shows Solidity code for a contract named 'financialContract'. The main panel shows the compilation details for this contract. The 'NAME' section shows 'financialContract'. The 'METADATA' section lists compiler settings: compiler, language (Solidity), output, settings, sources, and version (1). The 'BYTECODE' section is highlighted with a red circle, and a red arrow points from the 'Details' button in the top right to this section. The bytecode is displayed as a JSON object with the following structure:

```
{
  "linkReferences": {},
  "object": "6080604052600d60005534801561001557600080fd5b5060df806100246000396000f3006080",
  "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0xD PUSH1 0x0 SSTORE CALLVALUE DUP1 ISZ",
  "sourceMap": "25:221:0:-;;;76:2;62:16;;25:221;8:9:-1;5:2;;;30:1;27;20:12;5:2;25:221:0;"
}
```

# Ganache

Access your private key for signing your contract in MyEtherWallet.

The screenshot shows the Ganache application interface. At the top, there are navigation tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, and LOGS. Below these are status indicators for CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). A search bar is located on the right side.

The main content area displays account information:

- MNEMONIC:** slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there
- HD PATH:** m/44'/60'/0'/0/account\_index
- ADDRESS:** 0x231eAeEF9EA93F5370a1F633F32E45AF570980E8
- BALANCE:** 100.00 ETH
- TX COUNT:** 0
- INDEX:** 0

A modal window is open, showing the same account details as above. A red circle highlights a key icon in the top right corner of the modal. A red arrow points from this icon to the private key field in the modal. The private key is displayed as:

```
a53cf8cb7b66d91ca388ef9ce4e45e39997f2773247c27bb2c7cae35a1b3d383
```

A "DONE" button is located at the bottom of the modal.





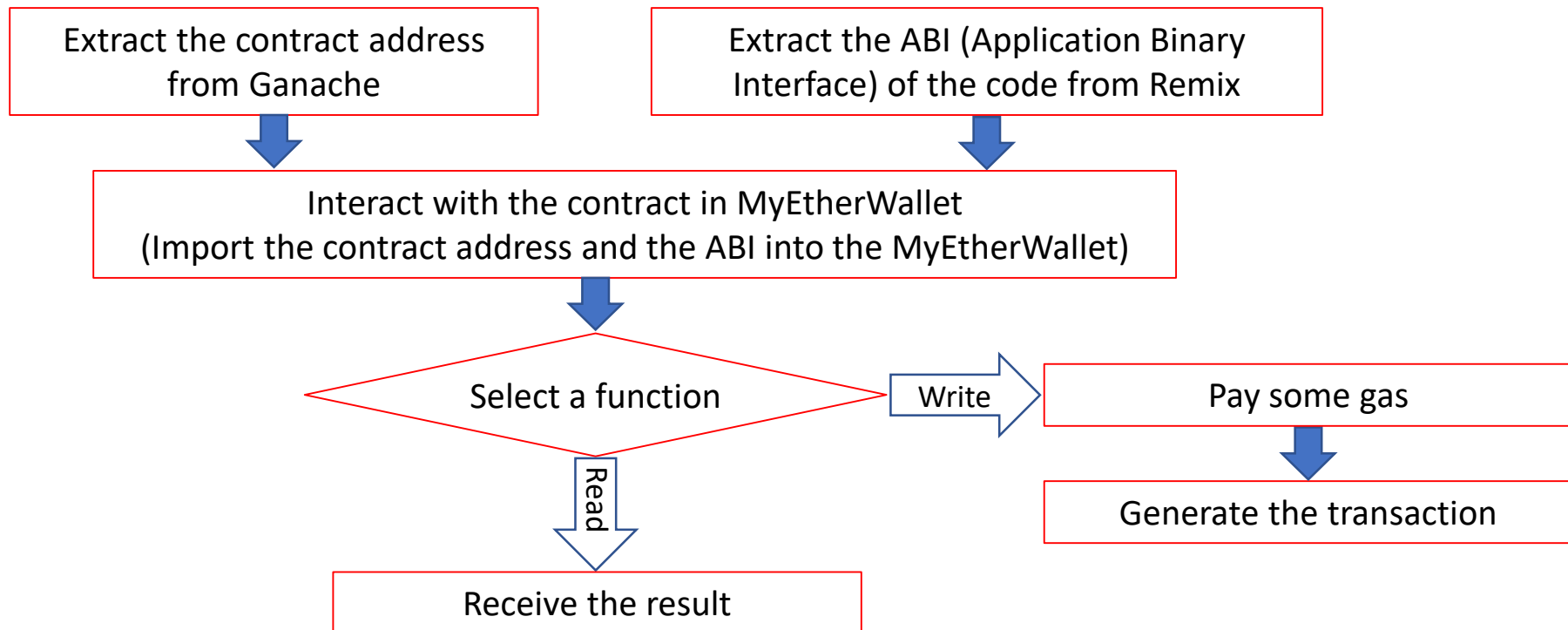
# Ganache

You can see now you have one transaction for your address and your balance has been changed because of the amount of gas you paid for creating the contract.

The screenshot shows the Ganache desktop application interface. At the top, there are navigation tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, and LOGS. A search bar is located on the right side of the top bar. Below the navigation tabs, there is a status bar with the following information: CURRENT BLOCK 1, GAS PRICE 20000000000, GAS LIMIT 6721975, NETWORK ID 5777, RPC SERVER HTTP://127.0.0.1:7545, and MINING STATUS AUTOMINING. The main area displays a list of accounts with their addresses, balances, and transaction counts. The first account's balance (99.99 ETH) and transaction count (1) are circled in red. The mnemonic for the first account is "slim rain lawn kiwi elegant behind vibrant dentist puppy reduce kidney there".

ADDRESS	BALANCE	TX COUNT	INDEX	
0x231eAeEF9EA93F5370a1F633F32E45AF570980E8	99.99 ETH	1	0	
0x970fc818790E900598C57E48b89B6D3D8896D416	100.00 ETH	0	1	
0xb59BD5568d0be42C13fB521f845243F1CDaF2eF1	100.00 ETH	0	2	
0x280AFA533B9fa1A97a6D2E4640412FD86FC5dd36	100.00 ETH	0	3	
0xD6D39E82AB17c30460F2CAc88425ECcaBf2757c5	100.00 ETH	0	4	

# Interacting with the smart contract



# Ganache

Transactions tab: Copy the created contract address

The screenshot shows the Ganache application interface. At the top, there are navigation tabs: ACCOUNTS, BLOCKS, TRANSACTIONS (highlighted with a red circle), and LOGS. Below the tabs is a search bar and a settings icon. The main content area displays transaction details for a 'CONTRACT CREATION' transaction. The details include:

TX HASH	FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0x1e40cc28802d152e810bd9f40bea83d83b1655fc9bace6e801ec6db5fcd84b1a	0x231eAeEF9EA93F5370a1F633F32E45AF570980E8	0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d	124604	0



# Remix

Click on Details button: Copy the ABI

(ABI is the interface that tells MyEtherWallet how to interact with the contract)

The screenshot displays the Remix IDE interface for a contract named 'Counter'. The main panel shows the contract's details, including the NAME, METADATA, BYTECODE, ABI, and WEB3DEPLOY sections. The ABI section is circled in red, and a red arrow points from the 'Details' button in the top right corner to the 'ABI' section. The ABI section contains the following information:

```
ABI
```

- 0:
- 1:
- 2:

The top right corner of the interface shows the 'Compile' tab with a 'Details' button circled in red. The 'Details' button is located next to the 'Counter' dropdown menu. The 'ABI' section is located at the bottom of the main panel.

# MyEtherWallet

Contracts tab:

Interact with Contract = Paste the contract address from Ganache and the ABI from Remix

The screenshot shows the MyEtherWallet interface with the 'Contracts' tab selected. The 'Interact with Contract' link is circled in red, with an arrow pointing to the 'Contract Address' input field, which also contains a red box around the address. The 'ABI / JSON Interface' field contains a red box around the JSON code. The 'Access' button is highlighted in blue.

New Wallet Send Ether & Tokens Swap Send Offline **Contracts** Ethers DomainSale Check TX Status View Wallet Info Help

**Interact with Contract** or Deploy Contract

**Contract Address**  
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

**Select Existing Contract**  
Select a contract...

**ABI / JSON Interface**

```
[{"outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function"}]
```

**Access**

# MyEtherWallet

You now can interact with the contract by selecting a function and invoking it

New Wallet Send Ether & Tokens Swap Send Offline **Contracts** ENS DomainSale Check TX Status View Wallet Info Help

## Interact with Contract or Deploy Contract

**Contract Address**  
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

**Select Existing Contract**  
Select a contract...

**ABI / JSON Interface**

```
[{"outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function"}]
```

**Access**

**Read / Write Contract**  
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

Select a function ▾

- getValue
- setValue

# MyEtherWallet

If you select the `getValue` function you will receive the value without paying any gas  
(There is no operation cost for getting information)

## Read / Write Contract

0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

getValue ▾

↳ uint256

13

# MyEtherWallet

If you choose a function that updates the state of the contract, you will need to pay gas for it in a transaction.

**Read / Write Contract**

0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

setValue ▾

**newValue** uint256

**WRITE**

**Warning!**

You are about to execute a function on contract.  
It will be deployed on the following network: ETH (Custom).

**Amount to Send** *In most cases you should leave this as 0.*

**Gas Limit**

**Generate Transaction**

**Raw Transaction**

```
{"nonce": "0x01", "gasPrice": "0x098bca5a00", "gasLimit": "0xa2a1", "to": "0xf22A8cA21D7eeF564FD5Ea743d"}
```

**Signed Transaction**

```
0xf8680185098bca5a0082a2a194f22a8ca21d7eef564fd5ea743dd9326197cfaa2d80845b34b96626a04285bd52ad31
```

No, get me out of here! **Yes, I am sure! Make transaction.**

197CFAA2d


**WRITE**

# MyEtherWallet

Now if you try getValue function again, you will see the change.

## Interact with Contract or [Deploy Contract](#)

**Contract Address**

 **Select Existing Contract**

**ABI / JSON Interface**

```
[{"outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function"}]
```

---

**Read / Write Contract**  
0xf22A8cA21D7eeF564FD5Ea743dd9326197CFAA2d

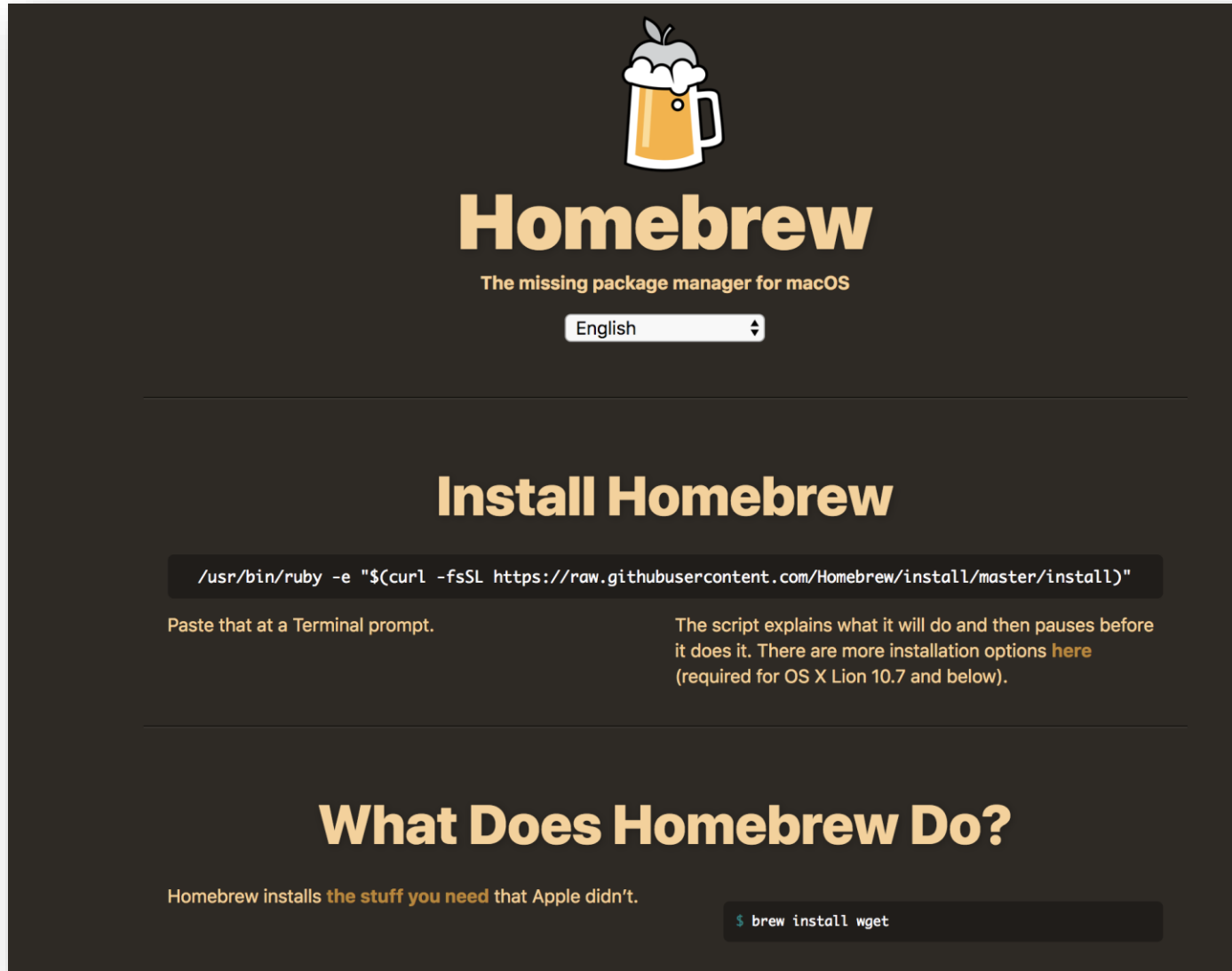
↳ uint256

# Create your own Ethereum Blockchain

- Instead of using Ganache with its default properties for private blockchain you can run your own blockchain
- Install Geth: One of the implementations of Ethereum written in Go
- Create the genesis block
- Create storage of the blockchain
- Deploy blockchain nodes
- Connect MyEtherWallet to your blockchain to interact with it

# Homebrew (package manager for mac)

- Install homebrew with the command from its website: <https://brew.sh/>



The screenshot shows the Homebrew website interface. At the top center is a logo of a beer mug with a green apple on top. Below the logo, the word "Homebrew" is written in a large, bold, yellow font. Underneath that, in a smaller white font, is the tagline "The missing package manager for macOS". Below the tagline is a language selection dropdown menu currently set to "English".

---

## Install Homebrew

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Paste that at a Terminal prompt.

The script explains what it will do and then pauses before it does it. There are more installation options [here](#) (required for OS X Lion 10.7 and below).

---

## What Does Homebrew Do?

Homebrew installs **the stuff you need** that Apple didn't.

```
$ brew install wget
```



# Geth

- An Ethereum program written in Go

1

```
mohammht — -bash — 80x24  
Last login: Wed May 30 10:38:04 on ttys001  
ds-install:~ mohammht$ brew tap ethereum/ethereum
```

2

```
ds-install:~ mohammht$ brew install ethereum
```

# Geth help

```
mohammht — -bash — 97x40
ds-install:~ mohammht$ geth help
NAME:
  geth - the go-ethereum command line interface

  Copyright 2013-2018 The go-ethereum Authors

USAGE:
  geth [options] command [command options] [arguments...]

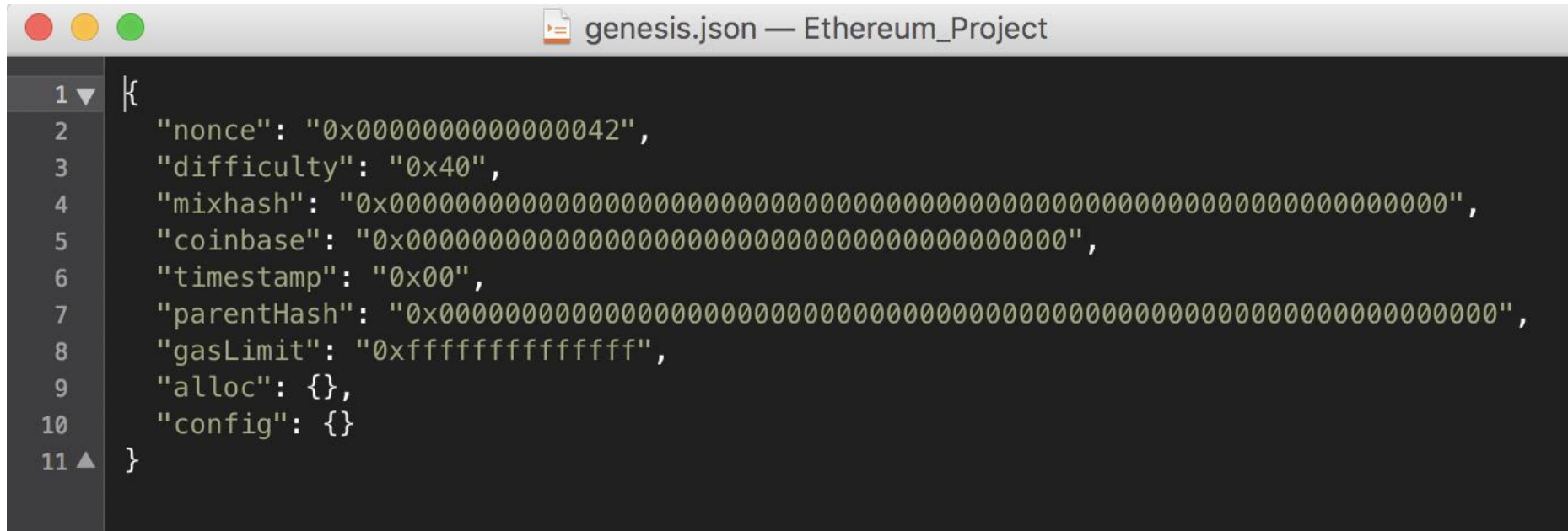
VERSION:
  1.8.9-stable

COMMANDS:
  account          Manage accounts
  attach           Start an interactive JavaScript environment (connect to node)
  bug              opens a window to report a bug on the geth repo
  console          Start an interactive JavaScript environment
  copydb           Create a local chain from a target chaindata folder
  dump             Dump a specific block from storage
  dumpconfig       Show configuration values
  export           Export blockchain into file
  export-preimages Export the preimage database into an RLP stream
  import           Import a blockchain file
  import-preimages Import the preimage database from an RLP stream
  init             Bootstrap and initialize a new genesis block
  js              Execute the specified JavaScript files
  license          Display license information
  makecache        Generate ethash verification cache (for testing)
  makedag          Generate ethash mining DAG (for testing)
  monitor          Monitor and visualize node metrics
  removedb         Remove blockchain and state databases
  version          Print version numbers
  wallet           Manage Ethereum presale wallets
  help, h          Shows a list of commands or help for one command

ETHEREUM OPTIONS:
  --config value   TOML configuration file
  --datadir "/Users/mohammht/Library/Ethereum" Data directory for the databases and keystore
  --keystore       Directory for the keystore (default = inside the
  datadir)
```

# Genesis block

- The first block in the chain and a json file that stores the configuration of the chain

A screenshot of a code editor window titled "genesis.json — Ethereum\_Project". The editor shows a JSON object with the following fields: "nonce", "difficulty", "mixhash", "coinbase", "timestamp", "parentHash", "gasLimit", "alloc", and "config". The "nonce" field has the value "0x00000000000000042". The "difficulty" field has the value "0x40". The "mixhash" field has a value of 64 zeros. The "coinbase" field has a value of 32 zeros. The "timestamp" field has the value "0x00". The "parentHash" field has a value of 64 zeros. The "gasLimit" field has the value "0xffffffff". The "alloc" and "config" fields are empty objects. The code is displayed on a dark background with a light-colored text color.

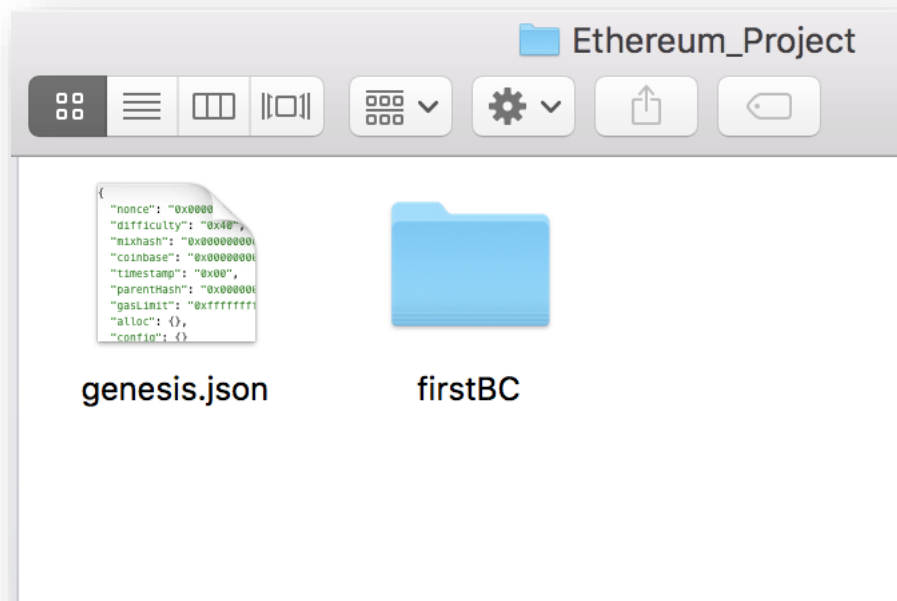
```
1 {  
2   "nonce": "0x00000000000000042",  
3   "difficulty": "0x40",  
4   "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
5   "coinbase": "0x0000000000000000000000000000000000000000",  
6   "timestamp": "0x00",  
7   "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
8   "gasLimit": "0xffffffff",  
9   "alloc": {},  
10  "config": {}  
11 }
```

- Create and store the file as genesis.json

# Create the storage of the blockchain

- Go to the directory of the genesis.json file
- Specify directory of your blockchain
- Create the storage from the genesis block

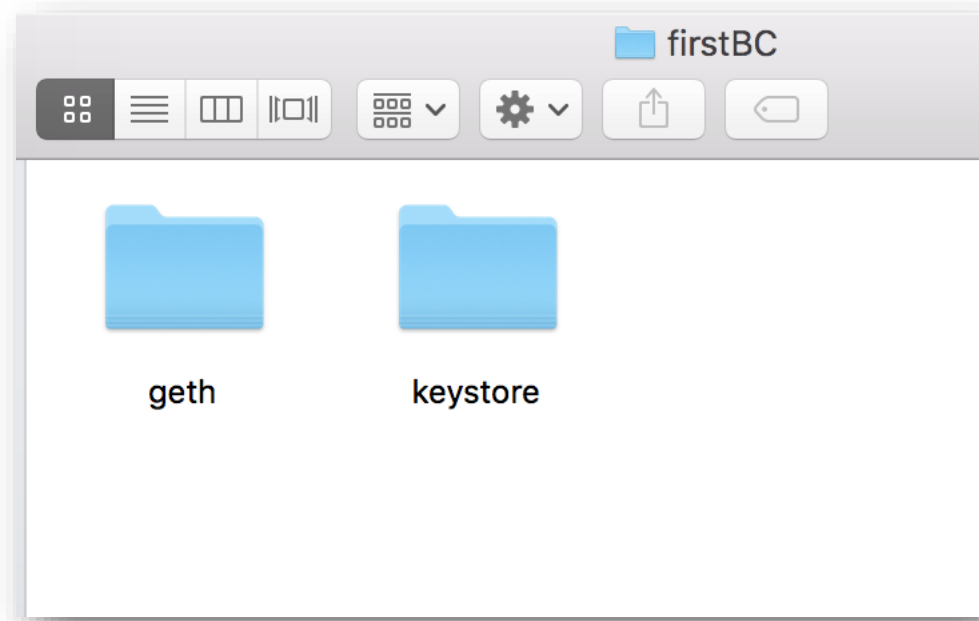
```
[ds-install:Documents mohammht$ cd Ethereum_Project/  
ds-install:Ethereum_Project mohammht$ geth --datadir firstBC init genesis.json
```



Folder name of your  
blockchain

# Inside the Blockchain Folder

- geth folder: Store your database
- keystore: Store your Ethereum accounts



# Start the Ethereum peer node

- Start the blockchain

```
geth --datadir fistBC --networkid 100 console
```

- Networkid provides privacy for your network.
- Other peers joining your network must use the same networkid.

# Blockchain started

- Type `admin.nodeInfo` to get the information about your current node

```
[> admin.nodeInfo
{
  enode: "enode://4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022b7ca7b9fba06bb0eda8b47210b198f3eeff7e67414d695ed6@[::]:30303",
  id: "4561ccdd7fdf3f0bdbc903b7bef7d472e136fe2b63012151a1dd3c27e52f49bda2ef66631e67022b7ca7b9fba06bb0eda8b47210b198f3eeff7e67414d695ed6",
  ip: "::",
  listenAddr: "[::]:30303",
  name: "Geth/v1.8.9-stable/darwin-amd64/go1.10.2",
  ports: {
    discovery: 30303,
    listener: 30303
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 4370000,
        chainId: 1,
        daoForkBlock: 1920000,
        daoForkSupport: true,
        eip150Block: 2463000,
        eip150Hash: "0x2086799aeebeae135c246c65021c82b4e15a2c451340993aacfd2751886514f0",
        eip155Block: 2675000,
        eip158Block: 2675000,
        ethash: {},
        homesteadBlock: 1150000
      },
      difficulty: 17179869184,
      genesis: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      head: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      network: 100
    }
  }
}
```

# Create an account

- Type *personal.newAccount* to create as many accounts as you need

```
> personal.newAccount('Type your password here')  
"0xa78eb41a10f096d4d8c4c9ca5196427aaa3fdb33"  
> █
```

- See the created account(s)

```
> eth.accounts  
["0xa78eb41a10f096d4d8c4c9ca5196427aaa3fdb33", "0x354d952e40fc35a47562d479c86e41f6623e5f8c"]  
>
```



# Mining

- Type *miner.start()* to start mining

```
[> miner.start()
INFO [05-30|12:07:54] Updated mining threads          threads=0
INFO [05-30|12:07:54] Transaction pool price threshold updated price=18000000000
null
> INFO [05-30|12:07:54] Starting mining operation
INFO [05-30|12:07:54] Commit new mining work          number=1 txs=0 uncles=0 elapsed=22
8.827µs
INFO [05-30|12:07:57] Generating DAG in progress      epoch=1 percentage=0 elapsed=2.013
s
INFO [05-30|12:07:59] Generating DAG in progress      epoch=1 percentage=1 elapsed=4.151
s
INFO [05-30|12:08:03] Generating DAG in progress      epoch=1 percentage=2 elapsed=7.322
s
INFO [05-30|12:08:06] Generating DAG in progress      epoch=1 percentage=3 elapsed=10.70
5s
INFO [05-30|12:08:09] Generating DAG in progress      epoch=1 percentage=4 elapsed=14.04
3s
INFO [05-30|12:08:13] Generating DAG in progress      epoch=1 percentage=5 elapsed=17.56
5s
INFO [05-30|12:08:16] Generating DAG in progress      epoch=1 percentage=6 elapsed=20.99
9s
INFO [05-30|12:08:20] Generating DAG in progress      epoch=1 percentage=7 elapsed=24.40
9s
```

Thank you