

Lab 01:

Solidity, a Smart Contract Language

Nick Zoghb





LAB OUTLINE

2

1



ETHEREUM VIRTUAL MACHINE

2



SOLIDITY: OVERVIEW

3



QUICK DOCS

4



EXERCISE: SOLIDITY + REMIX

1

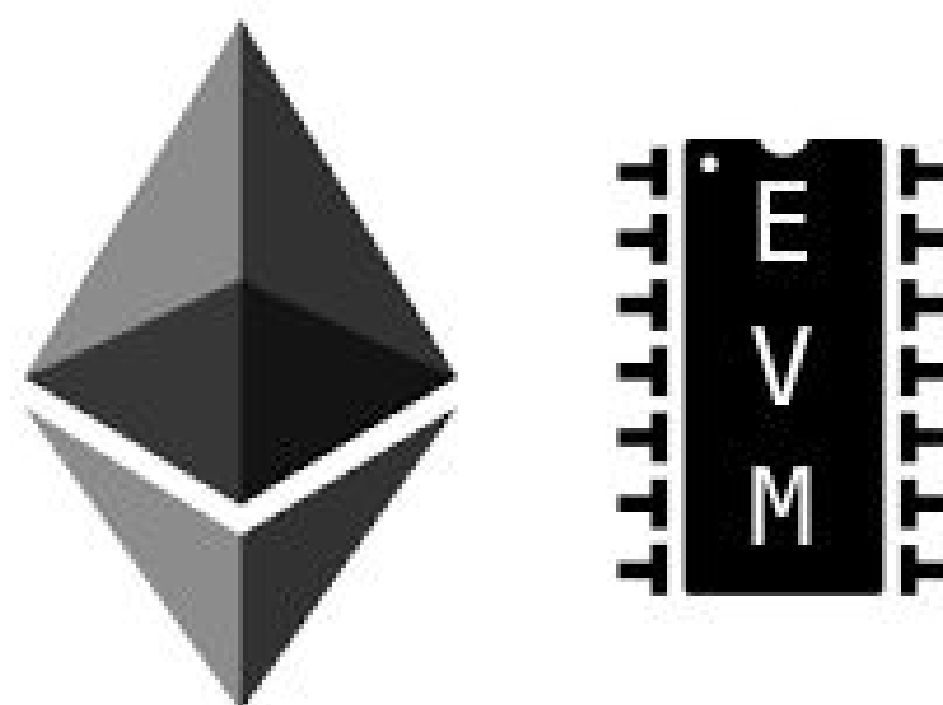
ETHEREUM VIRTUAL MACHINE

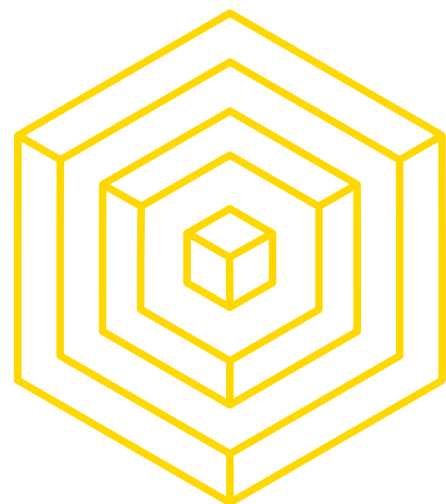


THE ETHEREUM VIRTUAL MACHINE

THE WORLD COMPUTER

- Ethereum implements an execution environment known as the *Ethereum Virtual Machine*
 - Every node participating in the network runs the EVM
- Nodes will go through the transactions listed in the block they are verifying and run the code as triggered by the transaction within the EVM
- Every full node does the same calculations and stores the same values





OPCODES

WHAT MAKES THE COMPUTER TICK

5

See [here](#) for more

1	Value	Mnemonic	Gas Used	Subset	Removed from stack	Added to stack	Notes
2	0x00	STOP	0	zero	0	0	Halts execution.
3	0x01	ADD	3	verylow	2	1	Addition operation
4	0x02	MUL	5	low	2	1	Multiplication operation.
5	0x03	SUB	3	verylow	2	1	Subtraction operation.
6	0x04	DIV	5	low	2	1	Integer division operation.
7	0x05	SDIV	5	low	2	1	Signed integer division operation.
8	0x06	MOD	5	low	2	1	Modulo remainder operation.
9	0x07	SMOD	5	low	2	1	Signed modulo remainder operation.
10	0x08	ADDMOD	8	mid	3	1	Modulo addition operation.
11	0x09	MULMOD	8	mid	3	1	Modulo multiplication operation.
12	0x0a	EXP	$(exp == 0) ? 10 : (10 + 10 * (1 + \log_{256}(exp)))$		2	1	Exponential operation.
13	0x0b	SIGNEXTEND	5	low	2	1	Extend length of two's complement.
14	0x10	LT	3	verylow	2	1	Less-than comparison.
15	0x11	GT	3	verylow	2	1	Greater-than comparison.



OPCODES

WHAT MAKES THE COMPUTER TICK

Here given are the various exceptions to the state transition rules given in section 9 specified for each instruction, together with the additional instruction-specific definitions of J and C . For each instruction, also specified is α , the additional items placed on the stack and δ , the items removed from stack, as defined in section 9.

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted.

Value	Mnemonic	δ	α	Description
-------	----------	----------	----------	-------------

0x00	STOP	0	0	Halts execution.
------	------	---	---	------------------

0x01	ADD	2	1	Addition operation. $\mu'_s[0] \equiv \mu_s[0] + \mu_s[1]$
------	-----	---	---	---

0x02	MUL	2	1	Multiplication operation. $\mu'_s[0] \equiv \mu_s[0] \times \mu_s[1]$
------	-----	---	---	--

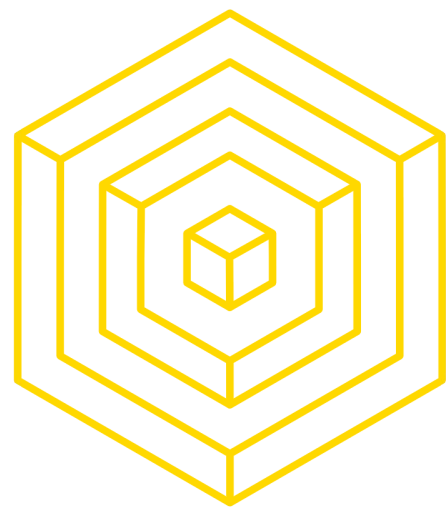
0x03	SUB	2	1	Subtraction operation. $\mu'_s[0] \equiv \mu_s[0] - \mu_s[1]$
------	-----	---	---	--

0x04	DIV	2	1	Integer division operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$
------	-----	---	---	--

0x05	SDIV	2	1	Signed integer division operation (truncated). $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ -2^{255} & \text{if } \mu_s[0] = -2^{255} \wedge \mu_s[1] = -1 \\ \text{sgn}(\mu_s[0] \div \mu_s[1]) \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.
------	------	---	---	---

0x06	MOD	2	1	Modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \mu_s[0] \bmod \mu_s[1] & \text{otherwise} \end{cases}$
------	-----	---	---	---

See [here](#) for more

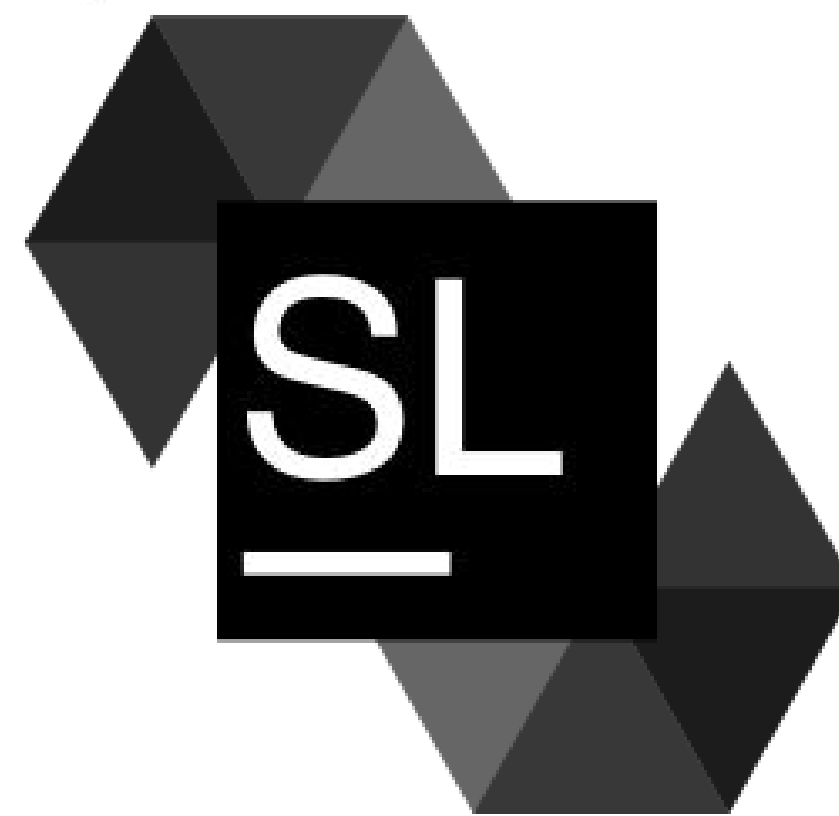


THE ETHEREUM VIRTUAL MACHINE

THE WORLD COMPUTER

7

- Solidity lets you program on Ethereum, a blockchain-based virtual machine that allows the creation and execution of smart contracts, without requiring centralized or trusted parties
- Statically typed, contract programming language that has similarities to Javascript and C
 - Like objects in OOP, each contract contains state variables, functions, and common data types
 - Contract-specific features include modifier (guard) clauses, event notifiers for listeners, and custom global variables



2 SOLIDITY OVERVIEW



THE BANK CONTRACT

LEARNING FAST

WHAT DOES A BANK NEED TO DO?

1. Allow Deposits
2. Allow Withdrawals
3. Balance Checks

[Learn X in Y minutes](#), a whirlwind
tour of your favorite language

```
1 contract SimpleBank {
2     mapping (address => uint) private balances;
3     address public owner;
4     event LogDepositMade(address accountAddress, uint amount);
5
6     function SimpleBank() {
7         owner = msg.sender;
8     }
9
10    function deposit() public returns (uint) {
11        balances[msg.sender] += msg.value;
12        LogDepositMade(msg.sender, msg.value);
13        return balances[msg.sender];
14    }
15
16    function withdraw(uint withdrawAmount) public returns (uint remainingBal) {
17        if(balances[msg.sender] >= withdrawAmount) {
18            balances[msg.sender] -= withdrawAmount;
19            if (!msg.sender.send(withdrawAmount)) {
20                balances[msg.sender] += withdrawAmount;
21            }
22        }
23        return balances[msg.sender];
24    }
25
26    function balance() constant returns (uint) {
27        return balances[msg.sender];
28    }
29
30    function () {
31        throw;
32    }
33 }
```



THE BANK CONTRACT

LEARNING FAST

- **contract** has similarities to **class** in other languages (class variables, inheritance, etc.)
 - Declare state variables outside function, persist through life of contract
- **mapping** is a dictionary that maps addresses to balances
 - always be careful about overflow attacks with numbers
 - **private** means that other contracts can't directly query balances
 - but data is still viewable to other parties on blockchain
- **public** makes externally readable (not writeable) by users or contracts

```
1 contract SimpleBank {  
2  
3  
4  
5     mapping (address => uint) private balances;  
6  
7  
8  
9  
10    address public owner;  
11  
12
```



THE BANK CONTRACT

LEARNING FAST

- **event** - publicize actions to external listeners
- **Constructor** - can receive one or many variables here; only one allowed
- **msg** provides details about the message that's sent to the contract
 - **msg.sender** is contract caller (address of contract creator)

```
4  event LogDepositMade(address accountAddress, uint amount);  
5  
6  function SimpleBank() {  
7      owner = msg.sender;  
8  }
```




THE BANK CONTRACT

LEARNING FAST

- **deposit()**

- Takes no parameters, but we are still sending Ether!
- **public** makes externally readable (not writeable) by users or contracts
- Returns user's balance as an unsigned integer (**uint**)

```
14 function deposit() public returns (uint) {  
15  
16     balances[msg.sender] += msg.value;  
17  
18     LogDepositMade(msg.sender, msg.value);  
19  
20  
21  
22  
23     return balances[msg.sender];  
24 }  
25
```

- **balances[msg.sender]**, no **this** or **self** required with state variable
- **LogDepositMade** event fired



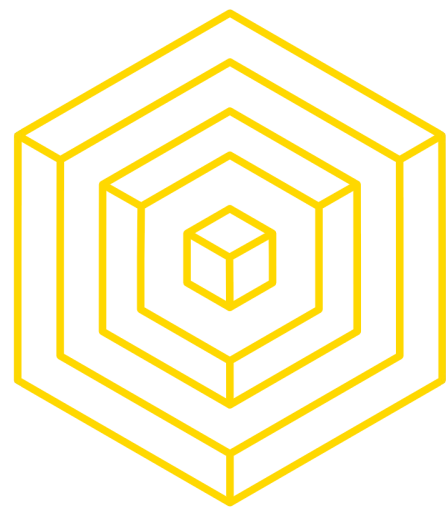
THE BANK CONTRACT

LEARNING FAST

13

- **Withdraw()**
 - **withdrawAmount** parameter
 - Returns user's balance
- Note the way we deduct the balance right away, before sending
 - We do this because of the risk of a recursive call that allows the caller to request an amount greater than their balance
- Increment back only on fail, as may be sending to contract that has overridden 'send' on the receipt end

```
function withdraw(uint withdrawAmount) public returns (uint remainingBal) {  
  
    if(balances[msg.sender] >= withdrawAmount) {  
  
        balances[msg.sender] -= withdrawAmount;  
  
        if (!msg.sender.send(withdrawAmount)) {  
  
            balances[msg.sender] += withdrawAmount;  
        }  
    }  
    return balances[msg.sender];  
}
```

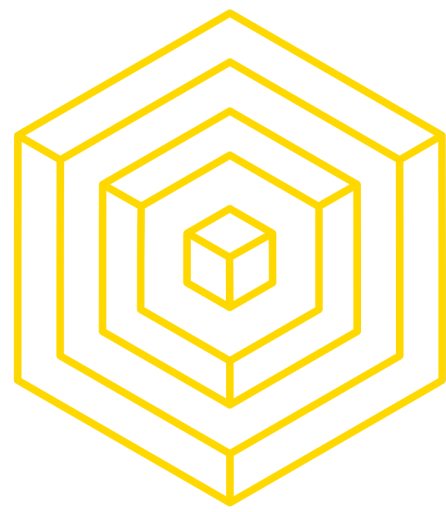


THE BANK CONTRACT

LEARNING FAST

- **balance():**
 - constant prevents function from editing state variables
 - Returns user's balance
 - allows function to run locally/off blockchain

```
function balance() constant returns (uint) {  
  
    return balances[msg.sender];  
}
```

THE BANK CONTRACT

LEARNING FAST

15

() : Fallback function - Called if other functions don't match call or sent ether without data

- Typically, called when invalid data is sent
- Ether sent to this contract is reverted if the contract fails otherwise

- **throw** : throw reverts state to before call

```
function () {  
    throw;  
}
```

3

QUICK DOCS

3.1 DATA TYPES



DATA TYPES

INTEGERS

```
// uint used for currency amount (there are no doubles or floats) and for dates (in unix time)
```

```
uint x;
```

```
// int of 256 bits, cannot be changed after instantiation
```

```
int constant a = 8;
```

```
int256 constant a = 8; // same effect as line above, here the 256 is explicit
```

```
uint constant VERSION_ID = 0x123A1; // A hex constant
```



DATA TYPES

INTEGERS

[Read the docs](#)

```
// For int and uint, can explicitly set space in steps of 8 up to 256; e.g. int8, int16, int24
```

```
uint8 b;
```

```
int64 c;
```

```
uint248 e;
```

```
// Be careful that you don't overflow, and protect against attacks that do
```

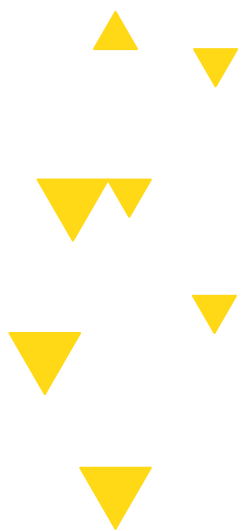
```
// No random functions built in, use other contracts for randomness
```

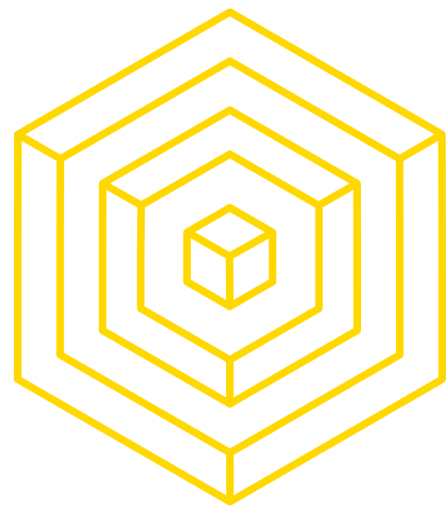
DEMO



TIME FOR A DEMO

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





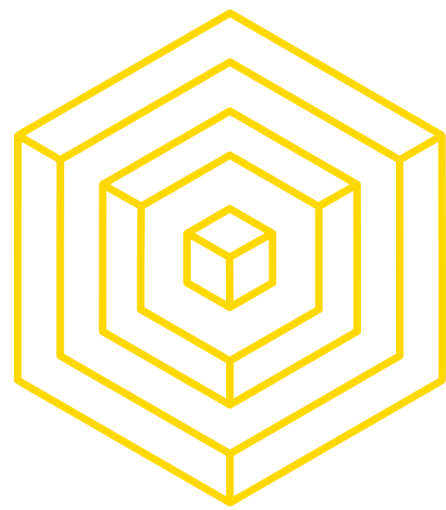
DATA TYPES

TYPE CASTING, BOOLEAN, ADDRESS

```
// Type casting
int x = int(b);

bool b = true; // or do 'var b = true;' for inferred typing

// Addresses - holds 20 byte/160 bit Ethereum addresses
// No arithmetic allowed
address public owner;
```



DATA TYPES

TYPE CASTING, BOOLEAN, ADDRESS

```
// Type casting
int x = int(b);

bool b = true; // or do 'var b = true;' for inferred typing

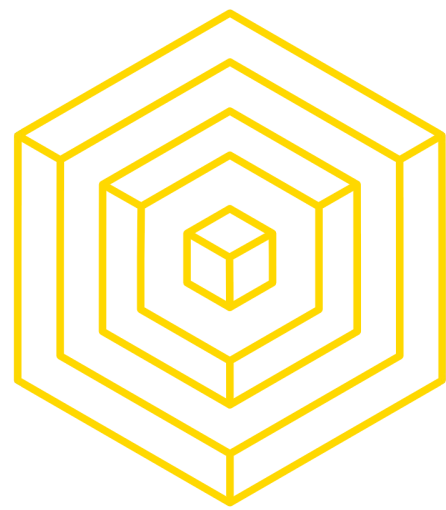
// Addresses - holds 20 byte/160 bit Ethereum addresses
// No arithmetic allowed
address public owner;
```



DATA TYPES

ADDRESS, SENDING ETHER

```
address public owner;  
  
// All addresses can be sent ether  
owner.send(SOME_BALANCE); // returns false on failure  
  
if (owner.send(*20 ether*)) {}  
  
// REMEMBER: wrap in 'if', as contract addresses have  
// functions executed on send and these can fail  
// Also, make sure to deduct balances BEFORE attempting a send, as there is a risk of a  
recursive call that can drain the contract
```

DATA TYPES

ADDRESS, SENDING ETHER

- Balance of the address in *Wei*

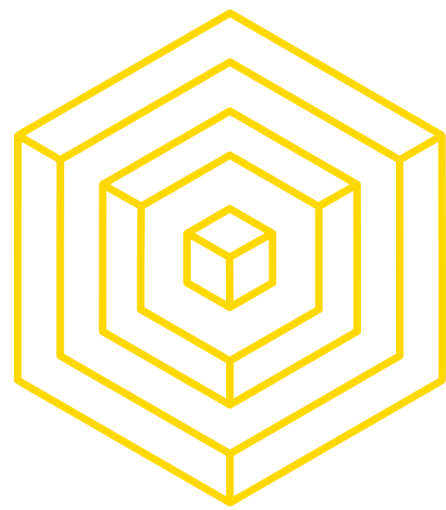
```
<address>.balance
```

- Send given amount of Wei to address, throws on failure

```
<address>.transfer(uint256 amount)
```

- Send given amount of Wei to address, returns false on failure

```
<address>.transfer(uint256 amount)
```



DATA TYPES

ADDRESS, SENDING ETHER

- Balance of the address in *Wei*

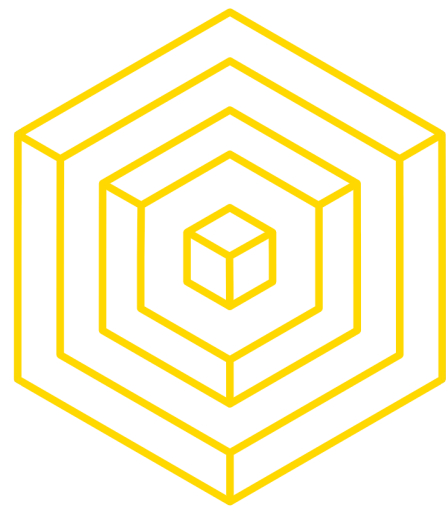
```
<address>.balance
```

- Send given amount of Wei to address, throws on failure

```
<address>.transfer(uint256 amount)
```

- ~~Send given amount of Wei to address, returns false on failure~~

```
<address>.transfer(uint256 amount)
```



DATA TYPES

BYTES

```
// Bytes available from 1 to 32  
byte a; // byte is same as bytes1  
bytes2 b;  
bytes32 c;  
  
// Dynamically sized bytes  
bytes m; // A special array, same as byte[] array (but packed tightly)  
// More expensive than bytes1 - bytes32, so use those when possible
```




DATA TYPES

BYTES

```
// same as bytes, but does not allow length or index access (for now)  
string n = "hello";  
  
// stored in UTF-8, note double quotes, not single  
  
// string utility functions to be added in future  
  
// prefer bytes32/bytes, as UTF-8 uses more storage
```



DATA TYPES

BYTES

```
// Type inference
// var does inferred typing based on first assignment,
// can't be used in functions parameters
var a = true;
// use carefully, inference may provide wrong type
// e.g., an int8, when a counter needs to be int16
```



DATA TYPES

FUNCTION ASSIGNMENT, DEFAULT VALUES, DELETE, UNWRAP TUPLES

```
// Variables can be used to assign function to variable
function a(uint x) returns (uint) {
    return x * 2;
}
var f = a;
f(22); // call

// By default, all values are set to 0 on instantiation
// Delete can be called on most types
// (does NOT destroy value, but sets value to 0, the initial value)
uint x = 5;
```


3.2 DATA STRUCTURES



DATA STRUCTURES

ARRAYS

```
// Arrays
bytes32[5] nicknames; // static array
bytes32[] names; // dynamic array
uint newLength = names.push("John"); // adding returns new length of the array
// Length
names.length; // get length
names.length = 1; // lengths can be set (for dynamic arrays in storage only)
// multidimensional array
uint x[][5]; // arr with 5 dynamic array elements (opposite order of most languages)
```



DATA STRUCTURES

MAPPINGS

```
// Dictionaries (any type to any other type)
mapping (string => uint) public balances;
balances["charles"] = 1;
console.log(balances["ada"]); // is 0, all non-set key values return zeroes
// 'public' allows following from another contract
contractName.balances("charles"); // returns 1
// 'public' created a getter (but not setter) like the following:
function balances(string _account) returns (uint balance) {
    return balances[_account];
}
```



DATA STRUCTURES

MAPPINGS

```
// Nested mappings
mapping (address => mapping (address => uint)) public custodians;

// To delete
delete balances["John"];
delete balances; // sets all elements to 0
// Unlike other languages, CANNOT iterate through all elements in
// mapping, without knowing source keys - can build data structure
// on top to do this
```




DATA STRUCTURES

STRUCTS

34

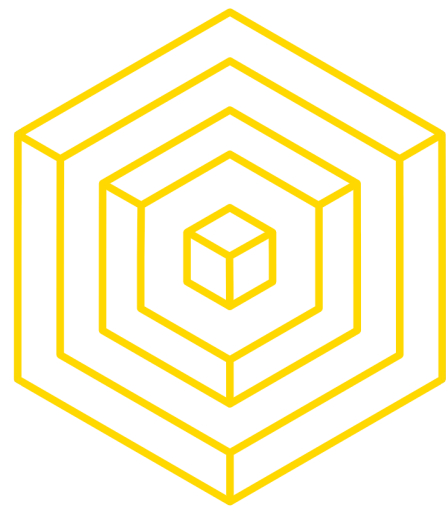
```
// Structs and enums
struct Bank {
    address owner;
    uint balance;
}
Bank b = Bank({
    owner: msg.sender,
    balance: 5
});
```



DATA STRUCTURES

STRUCTS

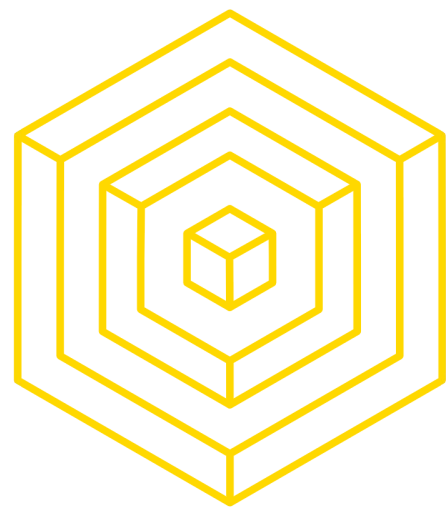
```
// or
struct Bank {
    address owner;
    uint balance;
}
Bank c = Bank(msg.sender, 5);
c.amount = 5; // set to new value
delete b;
// sets to initial value, set all variables in struct to 0, except mappings
```



DATA STRUCTURES

ENUMS

```
// Enums  
  
enum State { Created, Locked, Inactive } // often used for state machine  
  
State public state; // Declare variable from enum  
state = State.Created;  
  
// enums can be explicitly converted to ints  
uint createState = uint(State.Created); // 0
```



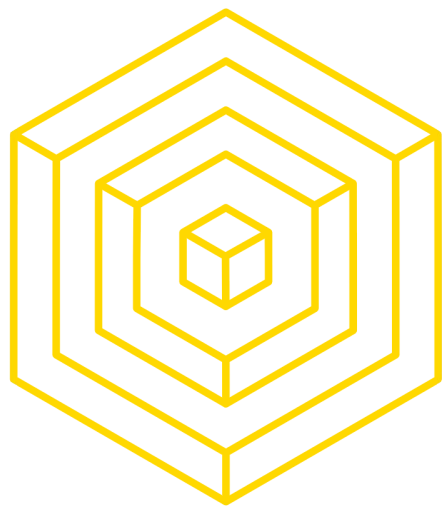
WHERE DATA GOES

NOTE ON MEMORY AND STORAGE

- Data locations: `memory` vs. `storage` vs. `stack` - all complex types (arrays, structs) have a data location
 - `memory` does not persist, `storage` does
- Default is `storage` for local and state variables; `memory` for function arguments
 - For most types, the data location to use can be explicitly set
- The stack holds small local variables
 - Used for values in intermediate calculations
 - General consensus is not to interact with it as a developer
 - As in this case

4

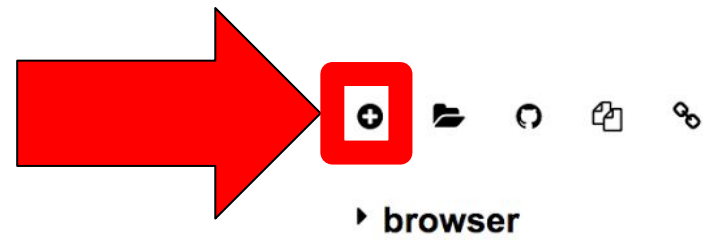
EXERCISE



REMIX IDE

THE WORLD ON A SINGLE WEB APP

39




« + browser/ballot.sol x »

```
1 pragma solidity ^0.4.0;
2 contract Ballot {
3
4     struct Voter {
5         uint weight;
6         bool voted;
7         uint8 vote;
8         address delegate;
9     }
10    struct Proposal {
11        uint voteCount;
12    }
13
14    address chairperson;
15    mapping(address => Voter) voters;
16    Proposal[] proposals;
17
18    /// Create a new ballot with $(_numProposals) different proposals.
19    function Ballot(uint8 _numProposals) public {
20        chairperson = msg.sender;
21        voters[chairperson].weight = 1;
22        proposals.length = _numProposals;
23    }
24
25    /// Give $(toVoter) the right to vote on this ballot.
26    /// May only be called by $(chairperson).
27    function giveRightToVote(address toVoter) public {
```

[2] only remix transactions, script

Q Search transactions

☐ Listen on network


remix

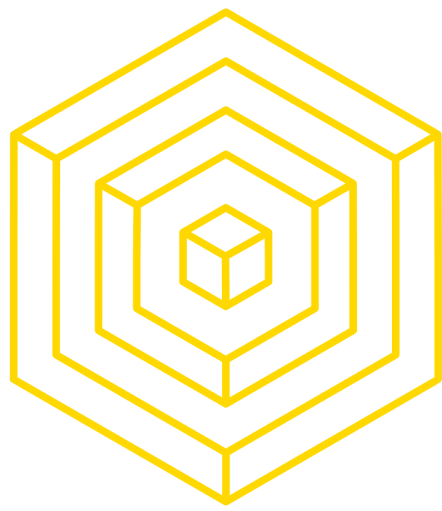
Compile Run Settings Analysis Debugger Support

Start to compile Auto compile

Ballot Details Publish on Swarm

Static Analysis raised 2 warning(s) that requires

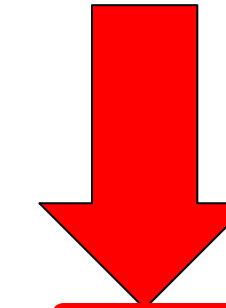
Ballot



REMIX IDE

THE WORLD ON A SINGLE WEB APP

40



« + browser/ballot.sol browser/Example.sol x »


```
1 pragma solidity ^0.4.0;
2 contract Example {
3
4     function do_something() constant returns (string) {
5         return "hello";
6     }
7
8 }
```

⌵

⊙ [2] only remix transactions, script ▾

🔍 Search transactions

🔊 Listen on network


remix

» Compile Run Settings Analysis Debugger Support

🔄 Start to compile

☒ Auto compile

Example ▾

Details

Publish on Swarm

Static Analysis raised 1 warning(s) that requires✖

browser/Example.sol:4:5: Warning: No visibility s✖

function do_something() constant returns (str

^

Spanning multiple lines.

browser/Example.sol:4:5: Warning: Function state ✖

function do_something() constant returns (str

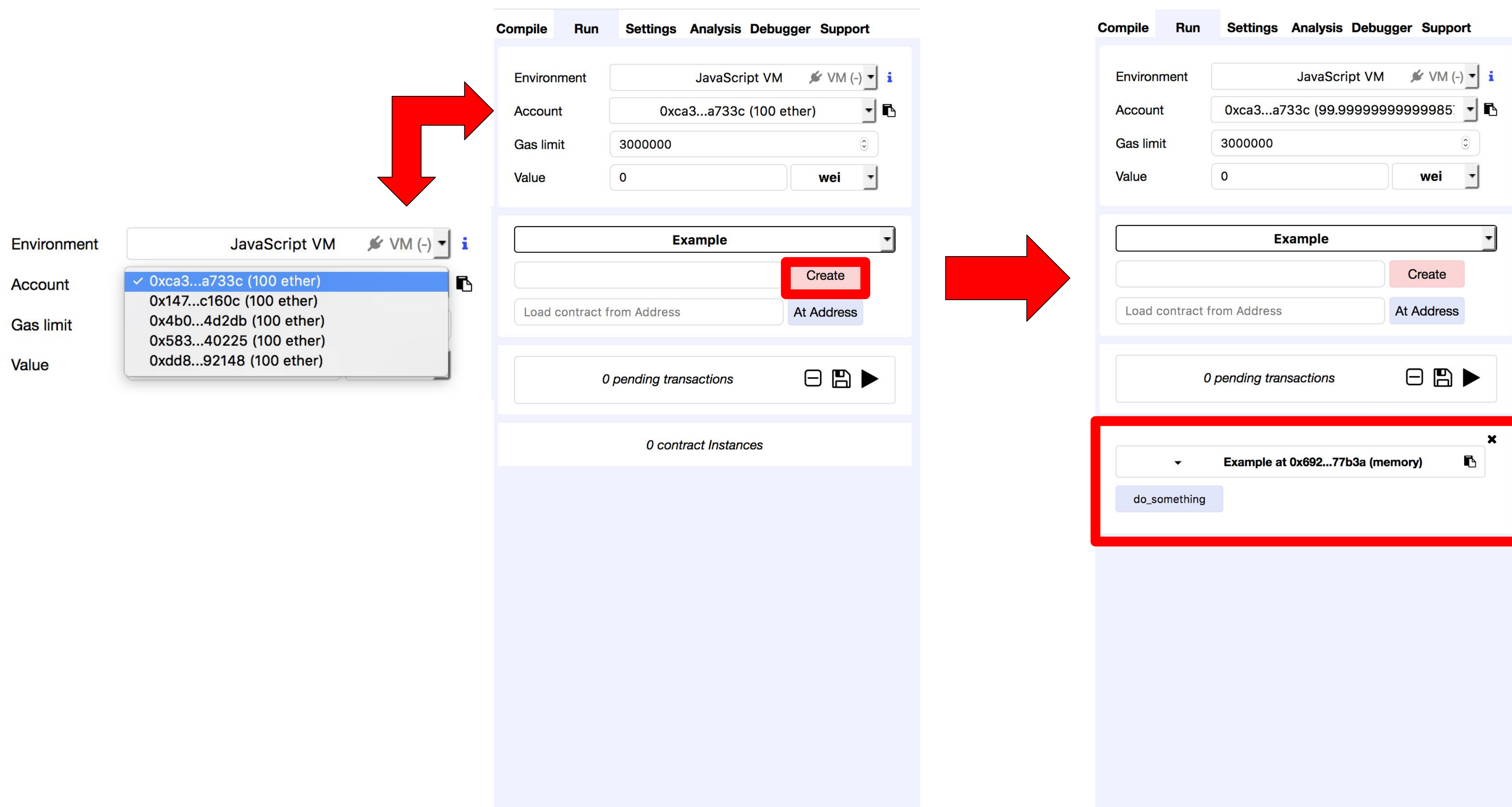
^

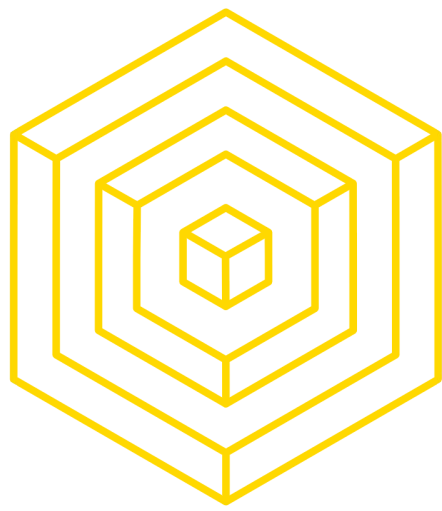
Spanning multiple lines.



REMIX IDE

THE WORLD ON A SINGLE WEB APP





REMIX IDE

THE WORLD ON A SINGLE WEB APP

Compile Run Settings Analysis Debugger Support

Environment JavaScript VM VM (-) i

Account 0xca3...a733c (99.99999999999985)

Gas limit 3000000

Value 0 wei

Example

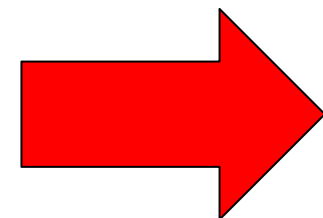
Create

Load contract from Address At Address

0 pending transactions

Example at 0x692...77b3a (memory)

do_something



Compile Run Settings Analysis Debugger Support

Environment JavaScript VM VM (-) i

Account 0xca3...a733c (99.99999999999985)

Gas limit 3000000

Value 0 wei

Example

Create

Load contract from Address At Address

0 pending transactions

Example at 0x692...77b3a (memory)

do_something 0: string: hello

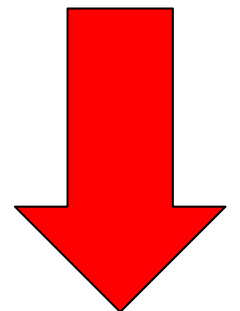
creation of Example pending...

[vm] from:0xca3...a733c, to:Example.(constructor), value:0 wei, data:0x606...10029, 0 logs, hash:0xd0f...0fd8d Details Debug

call to Example.do_something

[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to:Example.do_something(), data:c3d15...15da7, return: Details Debug

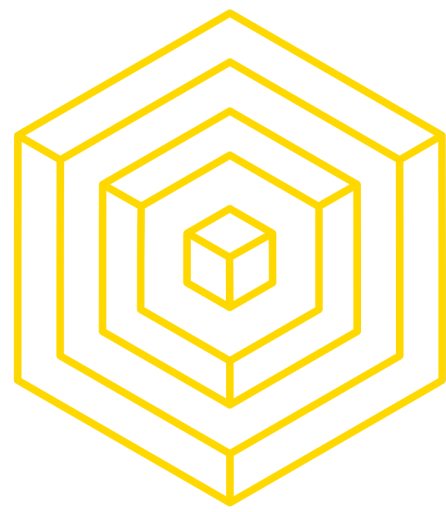
```
{
  "0": "string: hello"
}
```



[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to:Example.do_something(), data:c3d15...15da7, return: Details Debug

```
{
  "0": "string: hello"
}
```

from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	Example.do_something() 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
transaction cost	21934 gas (Cost only applies when called by a contract)
execution cost	662 gas (Cost only applies when called by a contract)
input	c3d15da7
decoded input	{}
decoded output	{ "0": "string: hello" }
logs	[]



THINGS TO DO

UH-OH, TIME TO WORK

We want you to flex your creative problem-solving skills in this new environment. With a mind for clean code, use the Remix IDE to implement the following:

- A 'greeter' contract with a 'greet' method that returns the string "hello, World!"
 - BONUS: The user should be able to change the greeting without redeploying the contract
- The Fibonacci function
 - Iteratively



THINGS TO DO (CONTINUED)

UH-OH, TIME TO WORK

- An XOR function
 - Input '1' or '0'
 - This does not require any bitwise operations!
 - BONUS: Input a string of 1's and 0's, e.g. "10001110101101"
- A method to concatenate two strings
 - Importing a module is fine
 - BONUS: Do not use a module

SEE YOU NEXT TIME

Ethereum Mechanics

Ethereum

Dapps

Smart Contracts

Types of Transactions

Types of Accounts

Gas