

### ethereum vienna

Workshop Contract Development for Beginners



**Workshop #1: Contract Development for Beginners** 

Requirements: Basic Understanding of Ethereum

**Solidity Basics** 

Workshop #2: From Idea to Contract

Requirements: Basic Understanding of Solidity

Mapping the real world to ethereum concepts

**Advanced Solidity** 

Workshop #3: From Contract to DApp

Requirements: Basic Understanding of Solidity, HTML/JS, node.js

Interfacing with Ethereum using web3.js

Auxiliary Technologies: IPFS, Whisper and Swarm

### Agenda

- 1. EVM Fundamentals
- 2. Mix IDE
- 3. Intro to Solidity
- 4. First Exercise: Trusted Data Feed
- 5. More Solidity
- 6. Exercises: Advanced Feed, Subscription
- 7. Solidity Data Structures
- 8. Final Exercise: Implementing a marketplace



### Transaction

wraps a message

signed by a private key

only transactions appear in chain

sets gasprice for all contained messages



### Message

Sender (where the ether is sent from)

Recipient (e.g. the executing contract)

Value (can be 0)

Data (used to encode the function call)

Return Value (used to retrieve the result of a computation)

Gaslimit (the maximal gas usage local to this message)

**Executes either completely or not at all** 



#### Contract

160 bit address (same address space as external accounts)

Balance (in Wei)

EVM Bytecode

Runs at every received message

Has a persistent 256-to-256 bit storage

Private (to other contracts, public to external actors)

**but** Expensive

Can spawn new messages during execution



Stack machine

256 bit words

Has all the usual instructions plus

block data, tx data, msg data, contract data access

cryptographic functions

message sending



```
Storage
  expensive
  persistent
Memory
  cheaper
  byte-level access
Stack
  inaccessible in solidity (except assembly)
```



Out of gas exception

Logs

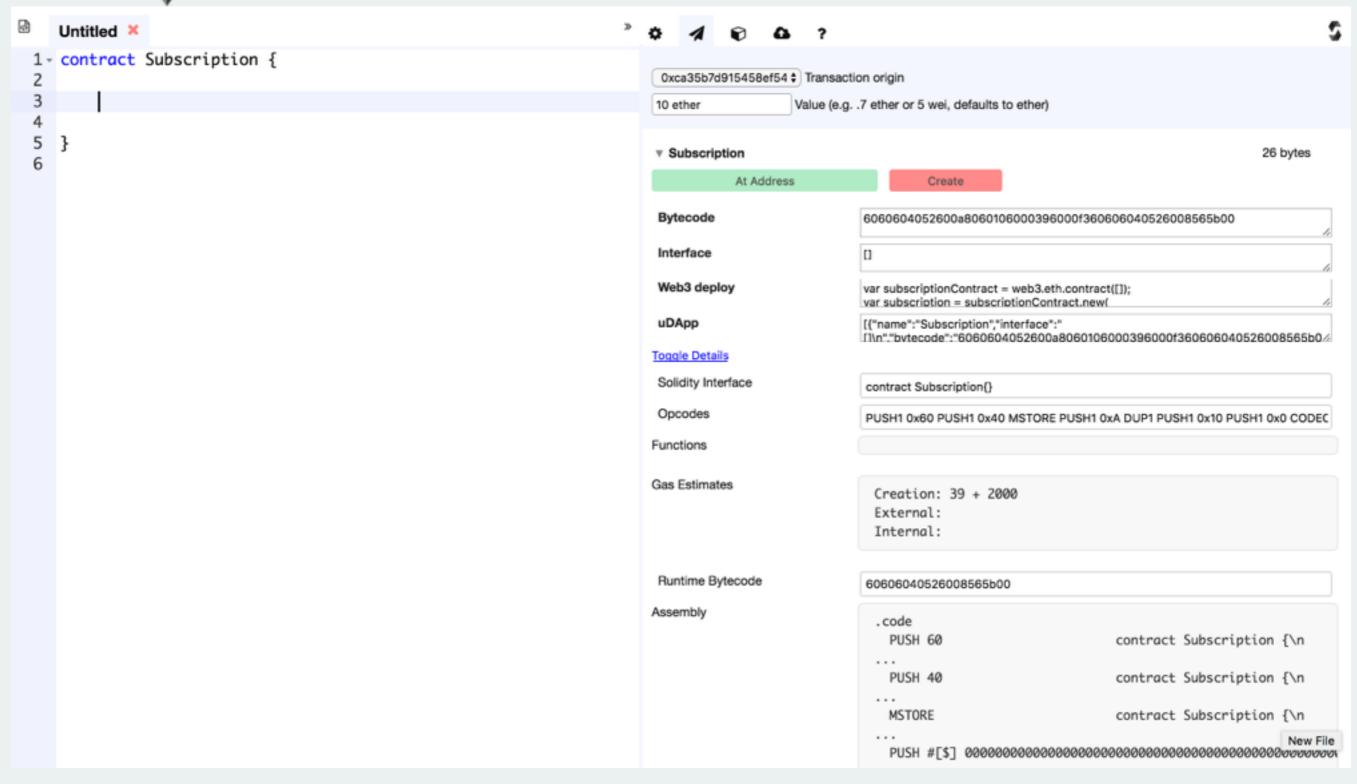
for UIs

**Light Clients** 

Logging

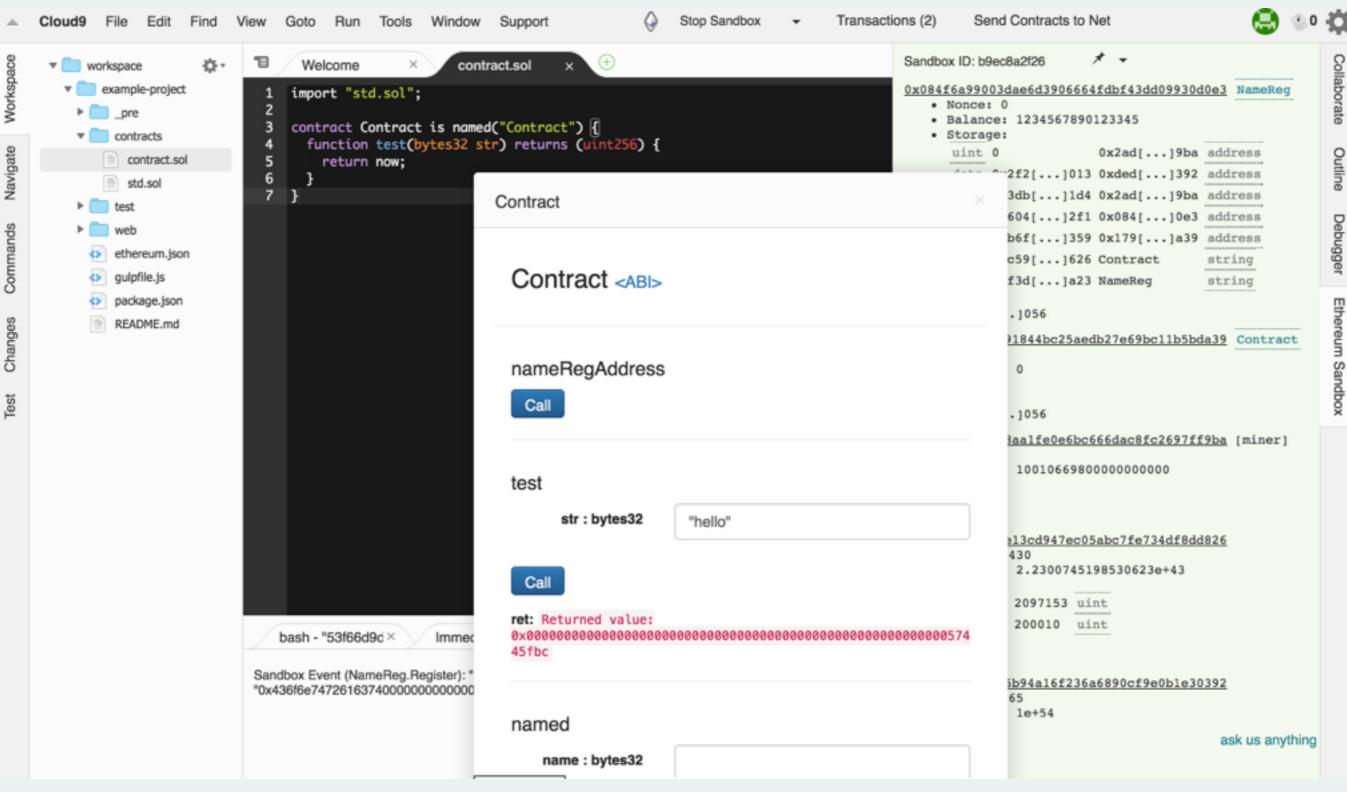
Self Destruct / Suicide

### ethereum Online Compiler





### ethereum Ethereum Studio





Solidity 0.4 released on Thursday

Ether Camp has not been updated yet (!)

But solidity online compiler has!

=> Incompatibilities

# ethereum Solidity

Developer writes contract with functions

```
Compiler generates
init code
dispatcher
At deployment the
contract constructor
is executed
```

```
pragma solc >= 0.4.1;
contract Sample {
    uint value;
    function Sample(uint v) {
        set(v);
    function set(uint v) {
        value = v;
    function get() returns (uint) {
        return value;
```

### Solidity

```
pragma solc  >= 0.4.1; indicates compiler version
contract Sample { starts a contract block
          contract name
 unsigned int 256 bit
 type
       uint value;
                          variable in contract storage
                           initialised to 0 by default
```

variable name

# ethereum Solidity

```
function name argument type name

function Sample(uint v) {
    set(v);
}
```

function call of set with argument v

Function with same name as contract = constructor Runs once at deployment

# ethereum Solidity

```
function set(uint v) {
     value = v;
    sets the storage of the variable value to v
                             return value type
function get() returns (uint) {
     return value;
   terminates function and returns value
   modifier code might still run (!) in solc >= 4.0
```



"Standard" types:

**Bool** 

Int: Signed 256 bit Integer (other sizes available)

**UInt**: Unsigned 256 bit Integer (other sizes available)

Array: Static and Dynamic

String (Unicode)

**Enum** 



#### Special types:

Address: 160 bit for ethereum address

Fields: balance

Functions: send, call, callcode, delegatecall

#### Mapping (hashtable-like):

maps from one solidity type to another

contains all keys at construction

#### **Contract Types**:

Inherits from address

Contract-specific functions

## ethereum Control Flow

If

```
function f (uint x) returns (uint) {
  if (x > 5) {
    return 3;
 } else {
    return 4;
```

### ethereum Control Flow

For

```
for (uint a = 0; a < 99; a++) {
}</pre>
```

While / Break

```
while(true) {
  break;
}
```



this.balance: gets the balance of the contract this.function(): calls a function by transaction super: inheritance

Automatic getter generation, declare a variable as public

Special variables for blockchain interaction



msg.

sender: immediate caller of the function

value: wei sent in the current message

gas: remaining gas available for the current message

tx.

origin: original creator of the transaction

gasprice: global gasprice (shared by all messages)



block.

coinbase: miner of the block

difficulty

timestamp: in unix time

blockhash

number: number of blocks since genesis

Special cryptographic functions (e.g. sha3)



Events for writing to the log

**Import** 

Standard contracts

Contract inheritance

```
pragma solc >= 0.4.1;
contract c {
    event GotWei(uint amount);
    function () payable {
        GotWei(msg.value);
    }
    like functions
}
```

Code from ancestor copied into child

Still only one contract



### **Trusted data feed**

Contains only one field

Can only be changed by the creator

Change Event

Field can be read by other contracts

relevant globals: msg.sender

## ethereum Modifier

#### Modifiers for code reuse

```
modifier afterDeadline() { if (now >= deadline) _; }

/* checks if the goal or time limit has been reached and ends the campaign */
function checkGoalReached() afterDeadline {
   if (amountRaised >= fundingGoal){
        // sends amountRaised wei to beneficiary account
        if (!beneficiary.send(amountRaised)) throw;
        FundTransfer(beneficiary, amountRaised, false);
   } else {
```

In solc < 4.0 it is only\_instead of \_;

# ethereum Exceptions

throw: creates and exception
execution aborts, state reverts
cannot be caught on contract functions
all gas is used

```
/* get the offer from the array */
var offer = offers[id];
/* throw if the sent value does not match the offer */
if(msg.value != offer.price) throw;
/* throw if the offer has already been taken */
if(offer.status != Status.OFFERED) throw;
```



Every address or contract object

has a send method, takes the amount in wei

```
function doSomething() {
  address recipient = 0x0;
  uint amount = 50 ether;
  var success = recipient.send(amount);
  if(!success) throw;
  if(!recipient.send(1 ether)) throw;
}
```

returns false if message does not succeed (does not throw!)

# ethereum Receiving Ether

```
function forward(address recipient) payable {
  if (!recipient.send(msg.value)) throw;
}
function () payable {
}
```

In solc >= 0.4, functions reject ether by default

If a function can be called with ether

explicit modifier **payable** necessary!



### **Trusted data feed**

Contains only one field

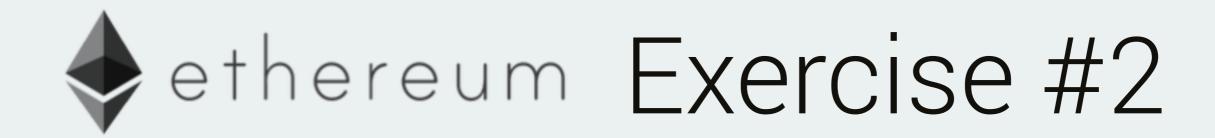
Can only be changed by the creator

Change Event

Field can be read by other contracts (for a fee)

Fee forwarded to creator

relevant globals: msg.value, throw



### **Subscription Contract**

Manages one subscription

Recipient: can withdraw PRICE wei per TIME

Creator: can cancel if there are not outstanding payments

#### relevant:

address.send(value): send value wei to address

block.timestamp: unix timestamp (in seconds)



Coerce address into contract type

```
Call the function on that

token public tokenReward;
Funder[] public funders;
mapping (address => bool) public

gas() to limit gas

// Coerce an address into a contract type
tokenReward = token(_reward);

// sends a sendCoin message to the tokenReward contract
```

tokenReward.sendCoin.value(10).gas(1000)(msg.sender, amount / price);
Warning: Recursion possible!

tokenReward.sendCoin(msg.sender, amount / price);

## ethereum Structs

```
/* data structure to hold information about campaign contributors */
struct Funder {
    address addr;
    uint amount;
}
```

```
// push an additional value onto the array
var funder = Funder({addr: msg.sender, amount: amount});
```

```
if (!funder.addr.send(funder.amount)) throw; /* P
FundTransfer(funder.addr, funder.amount, false);
```

# ethereum Arrays

```
Funder[] public funders;
```

dynamically sized array (starting with index 0)

push: adds a new element to the array

```
funders.push(Funder({addr: msg.sender, amount: amount}));
get element at index i
```

```
var funder = funders[i];
```

number of elements:

```
funders.length == index of the next pushed element
```

# ethereum Solidity

functions can have multiple return values retrieve values by deconstruction

```
function return2Values() returns (uint a, bool b) {
    a = 9;
    b = false;
}

function callThatFunction() {
    var (a,b) = return2Values();
}
```



#### External

Can only be called by a message

Public (default)

Can be called by anyone

Private

```
function f() private { }
function g() public { }
function h() external { }
function i() internal { }
```

Can only be called by the contract itself

Internal

Cannot be called by a message

## ethereum Enums

```
/* Status enum for the 3 possible states */
enum Status { OFFERED, TAKEN, CONFIRMED}
```

```
/* set status to confirmed */
offer.status = Status.CONFIRMED;
```

```
/* throw if offer is not taken */
if(offer.status != Status.TAKEN) throw;
```



#### **Market Contract**

Seller can add offers (with name and price)

Buyer can take offers (by sending the right amount)

Buyer can confirm the offer (and release funds)



**Workshop #1: Contract Development for Beginners** 

Requirements: Basic Understanding of Ethereum

**Solidity Basics** 

Workshop #2: From Idea to Contract

Requirements: Basic Understanding of Solidity

Mapping the real world to ethereum concepts

**Advanced Solidity** 

Workshop #3: From Contract to DApp

Requirements: Basic Understanding of Solidity, HTML/JS, node.js

Interfacing with Ethereum using web3.js

Auxiliary Technologies: IPFS, Whisper and Swarm





1vieCmqYB3DE8StinXYBGGvgJ9hoXP1ib

### The End

0x8DF95346D88aDBD11Cf799191F02c4e04C385f4E



