ETHEREUM
SOLIDITY IN-DEPTH

Collin Chin
Ali Mousa

BLOCKCHAIN
AT BERKELEY

PRESENTATION OUTLINE

DEV INTRO JUNE 2017

WHO We Are

DEV INTRO JUNE 2017

We're a student-run organization at UC Berkeley dedicated to serving the crypto and blockchain communities. Our members include Berkeley students, alumni, community members, and blockchain enthusiasts from all educational and industrial backgrounds.

BLOCKCHAIN
AT BERKELEY

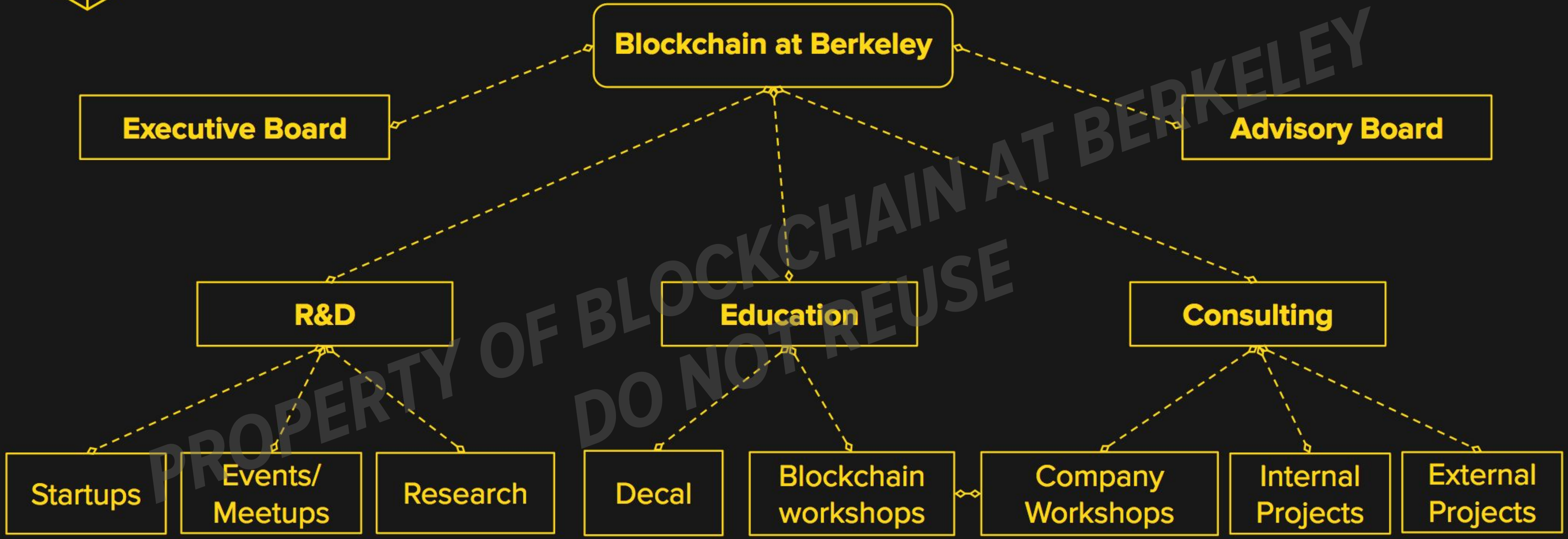
AUTHOR: APARNA KRISHNAN

WHAT WE HAVE DONE

DEV INTRO JUNE 2017

- Supply Chain Consulting Project with Airbus
- Game Theory and Network attacks Deep Dives at Hong Kong Bitcoin Meetups
- Taught a semester long class at UC-Berkeley
- Read more about us here

STRUCTURE


Blockchain at Berkeley
Executive Board
Advisory Board
R&D
Education
Consulting
Startups
Events/ Meetups
Research
Decal
Blockchain workshops
Company Workshops
Internal Projects
External Projects

ABOUT ALI

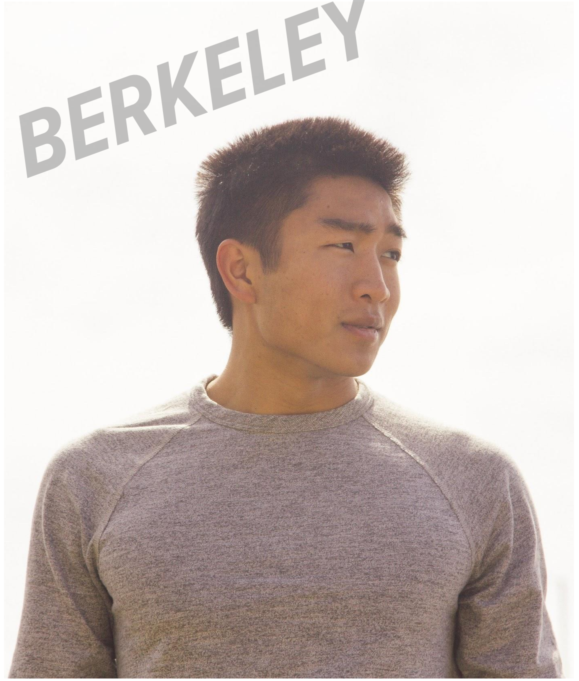
DEV INTRO JUNE 2017

- UC Berkeley Student: Computer Science
- Intern at **BlockApps**, the first company incubated out of ConsenSys
  - Developed Blockchain Management Dashboard which moved blockchain interactions (creating users, uploading contracts, calling functions, etc) from the command line to an intuitive user interface
- Lead developer on B@B's external consulting project with Airbus
  - Developed Supply Chain Management dApp which integrated a decentralized identity service
- Developed Decentralized Data Marketplace on IOTA
- Designed Ethereum Smart Contract Development Workshop material for new B@B developers

ABOUT COLLIN

DEV INTRO JUNE 2017

- **UC Berkeley** Student: EECS (Electrical Engineering and Computer Science)
- Software Engineering Intern for **Gnosis** at **Consensys**.
  - Contributed to truffle deployment and testing of GnosisJS platform library.
- External Consulting Developer for **Airbus** Project (2017):
  - Wrote solidity contracts and deployed POC to **Kovan** testnet.
- Internal Developer for **IoT Oracles** Project (2016):
  - Engineered Arduino Unos as IoT devices to resolve **Augur** prediction market events.
- Technical Writer for B@B Medium Blog
- Designed Ethereum Smart Contract Development Workshop material for new B@B developers



ethereum
HOMESTEAD RELEASE
BLOCKCHAIN APP PLATFORM

1

More on Solidity

1.1

ETHEREUM AND SOLIDITY RECAP

THE BANK CONTRACT

```
/* 'contract' has similarities to 'class' in other languages (class variables,
inheritance, etc.) */
contract SimpleBank { // CapWords
    // Declare state variables outside function, persist through life of contract

    // dictionary that maps addresses to balances
    // always be careful about overflow attacks with numbers
    mapping (address => uint) private balances;

    // "private" means that other contracts can't directly query balances
    // but data is still viewable to other parties on blockchain
    address public owner;
    // 'public' makes externally readable (not writeable) by users or contracts


    // Events - publicize actions to external listeners
    event LogDepositMade(address accountAddress, uint amount);

    // Constructor, can receive one or many variables here; only one allowed
    function SimpleBank() {
        // msg provides details about the message that's sent to the contract
        // msg.sender is contract caller (address of contract creator)
        owner = msg.sender;
    }
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

THE BANK CONTRACT

```
/// @notice Deposit ether into bank
/// @return The balance of the user after the deposit is made
function deposit() public returns (uint) {
    balances[msg.sender] += msg.value;
    // no "this." or "self." required with state variable
    // all values set to data type's initial value by default

    LogDepositMade(msg.sender, msg.value); // fire event

    return balances[msg.sender];
}
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

THE BANK CONTRACT

```
    /// @notice Withdraw ether from bank
    /// @dev This does not return any excess ether sent to it
    /// @param withdrawAmount amount you want to withdraw
    /// @return The balance remaining for the user
    function withdraw(uint withdrawAmount) public returns (uint remainingBal) {
        if(balances[msg.sender] >= withdrawAmount) {
            // Note the way we deduct the balance right away, before sending - due to
            // the risk of a recursive call that allows the caller to request an amount
greater
            // than their balance
            balances[msg.sender] -= withdrawAmount;

            if (!msg.sender.send(withdrawAmount)) {
                // increment back only on fail, as may be sending to contract that
                // has overridden 'send' on the receipt end
                balances[msg.sender] += withdrawAmount;
            }
        }

        return balances[msg.sender];
}
```

THE BANK CONTRACT

```
/// @notice Get balance
    /// @return The balance of the user
    // 'constant' prevents function from editing state variables;
    // allows function to run locally/off blockchain
    function balance() constant returns (uint) {
        return balances[msg.sender];
    }

    // Fallback function - Called if other functions don't match call or
    // sent ether without data
    // Typically, called when invalid data is sent
    // Added so ether sent to this contract is reverted if the contract fails
    // otherwise, the sender's money is transferred to contract
    function () {
        throw; // throw reverts state to before call
    }
}
```

1.2

DATA TYPES AND ASSOCIATED METHODS

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
// with 'constant', compiler replaces each occurrence with actual value


// 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
```

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
ACCOUNTS

Types of accounts:

- Contract account: address set on create (func of creator address, num transactions sent)
- External Account: (person/external entity): address created from public key

The `public` keyword allows an address (any variable, really) externally accessible

The address is given a "getter"

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) {} // 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

```
address public owner;

// All addresses can be sent ether
owner.send(SOME_BALANCE); // returns false on failure


if (owner.send) {} // 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
```

YOU CAN OVERRIDE THE SEND FUNCTION BY WRITING YOUR OWN (DANGER!)

DATA TYPES

ADDRESS, SENDING ETHER

`<address>.balance (uint256)`:

balance of the Address in Wei

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

send given amount of Wei to Address, throws on failure

`<address>.send(uint256 amount) returns (bool)`:

send given amount of Wei to Address, returns `false` on failure

DATA TYPES

ADDRESS, SENDING ETHER

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

send given amount of Wei to Address, throws on failure

~~`<address>.send(uint256 amount) returns (bool)`~~:

~~send given amount of Wei to Address, returns `false` on failure~~

The choice is obvious

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

BLOCKCHAIN AT BERKELEY

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 byte1-byte32, so use those when possible
```

DATA TYPES

STRING

```
// same as bytes, but does not allow length or index access (for now)
string n = "hello";

// stored in UTF8, note double quotes, not single


// string utility functions to be added in future


// prefer bytes32/bytes, as UTF8 uses more storage
```

DATA TYPES

TYPE INFERENCE, FUNCTION ASSIGNMENT

```
// 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

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

DATA TYPES

DEFAULT VALUES, DELETE, UNWRAP TUPLES

```
// 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;


// Destructuring/Tuples
(x, y) = (2, 7); // assign/swap multiple value
```

1.3

DATA STRUCTURES

DATA STRUCTURES

ARRAYS

```
// 2. DATA STRUCTURES
// 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];
}

// 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
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

DATA STRUCTURES

STRUCTS

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

STRUCTS, DATA STORAGE

```
// 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 createdState = uint(State.Created); //  0

// 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 func
params
// stack holds small local variables

// for most types, can explicitly set which data location to use
```

OPERATORS

```
// Comparisons, bit operators and arithmetic operators are provided
// exponentiation: **
// exclusive or: ^
// bitwise negation: ~
```

1.4

GLOBAL VARIABLES

GLOBAL VARIABLES

THIS, MSG, TX

```
this; // address of contract
// often used at end of contract life to send remaining balance to party

this.balance;
this.someFunction(); // calls func externally via call, not via internal jump



// ** msg - Current message received by the contract ** **
msg.sender; // address of sender
msg.value; // amount of ether provided to this contract in wei
msg.data; // bytes, complete call data
msg.gas; // remaining gas


// ** tx - This transaction **
tx.origin; // address of sender of the transaction
tx.gasprice; // gas price of the transaction
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

GLOBAL VARIABLES

BLOCK, STORAGE

```
// ** block - Information about current block **
now; // current time (approximately), alias for block.timestamp (uses Unix
time)
block.number; // current block number
block.difficulty; // current block difficulty
block.blockhash(1); // returns bytes32, only works for most recent 256
blocks
block.gasLimit();



// ** storage - Persistent storage hash **
storage['abc'] = 'def'; // maps 256 bit words to 256 bit words
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

BLOCKCHAIN AT BERKELEY

1.5

FUNCTIONS

FUNCTIONS

```
function increment(uint x) returns (uint) {
    x += 1;
    return x;
}

// Functions can return many arguments, and by specifying returned
arguments
// name don't need to explicitly return
function increment(uint x, uint y) returns (uint x, uint y) {
    x += 1;
    y += 1;
}
// Call previous function
uint (a,b) = increment(1,1);
```

FUNCTIONS

```
// 'constant' indicates that function does not/cannot change persistent
vars
// Constant function execute locally, not on blockchain
uint y;

function increment(uint x) constant returns (uint x) {
    x += 1;
    y += 1; // this line would fail
    // y is a state variable, and can't be changed in a constant function
}
```

FUNCTIONS

VISIBILITY

```
// These can be placed where 'constant' is, including:
// public - visible externally and internally (default)
// external
// private - only visible in the current contract
// internal - only visible in current contract, and those deriving from it

// Functions hoisted - and can assign a function to a variable
function a() {
    var z = b;
    b();
}

function b() {

}

// Prefer loops to recursion (max call stack depth is 1024)
```

1.5

EVENTS

BLOCKCHAIN
AT BERKELEY

EVENTS

EVENTS ARE AMAZING

```
// Declare
event LogSent(address indexed from, address indexed to, uint amount);

// note capital first letter

// Call
Sent(from, to, amount);
// For an external party (a contract or external entity), to watch:
Coin.Sent().watch({}, '', function(error, result) {
    if (!error) {
        console.log("Coin transfer: " + result.args.amount +
            " coins were sent from " + result.args.from +
            " to " + result.args.to + ".");
        console.log("Balances now:\n" +
            "Sender: " + Coin.balances.call(result.args.from) +
            "Receiver: " + Coin.balances.call(result.args.to));
    }
}
// Common paradigm for one contract to depend on another (e.g., a
// contract that depends on current exchange rate provided by another)
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

MODIFIERS

MODIFIERS ARE ALSO AMAZING

```
// C. Modifiers
// Modifiers validate inputs to functions such as minimal balance or
user auth;
// similar to guard clause in other languages

// '_' (underscore) often included as last line in body, and indicates
// function being called should be placed there
modifier onlyAfter(uint _time) { if (now <= _time) throw; _ }


modifier onlyOwner { if (msg.sender == owner) _ }
// commonly used with state machines


modifier onlyIfState (State currState) { if (currState != State.A) _ }
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

MODIFIERS

MODIFIERS ARE ALSO AMAZING

```
// Append right after function declaration
function changeOwner(newOwner)
onlyAfter(someTime)
onlyOwner()
onlyIfState(State.A)
{
    owner = newOwner;
}

// underscore can be included before end of body,
// but explicitly returning will skip, so use carefully
modifier checkValue(uint amount) {
    _
    if (msg.value > amount) {
        uint amountToRefund = amount - msg.value;
        if (!msg.sender.send(amountToRefund)) {
            throw;
        }
    }
}
```

AUTHOR: ALI MOUSA
Example: https://learnxinyminutes.com/docs/solidity/

1.6

LOOPS

LOOPS

LOOPS ARE DANGEROUS

```
// All basic logic blocks work - including if/else, for, while, break, continue
// return - but no switch

// Syntax same as javascript, but no type conversion from non-boolean
// to boolean (comparison operators must be used to get the boolean val)

// For loops that are determined by user behavior, be careful - as contracts
have a maximal amount of gas for a block of code - and will fail if that is
exceeded
// For example:
for(uint x = 0; x < refundAddressList.length; x++) {
    if (!refundAddressList[x].send(SOME_AMOUNT)) {
        throw;
    }
}
```

1.7

EXTERNAL CONTRACTS

EXTERNAL CONTRACTS

USAGE

```
contract InfoFeed {function info() returns (uint ret) { return 42; }}

contract Consumer {
    InfoFeed feed; // points to contract on blockchain


    function setFeed(address addr) { // Set feed to existing contract instance
        feed = InfoFeed(addr);// automatically cast, be careful; constructor is not called
    }

    function createNewFeed() { // Set feed to new instance of contract
        feed = new InfoFeed(); // new instance created; constructor called
    }

    function callFeed() {
        // final parentheses call contract, can optionally add
        // custom ether value or gas
        feed.info.value(10).gas(800)();
    }
}
```

INHERITANCE

USAGE

```
// Order matters, last inherited contract (i.e., 'def') can override parts of
// previously inherited contracts
contract MyContract is abc, def("a custom argument to def") {

// Override function
    function z() {
        if (msg.sender == owner) {
            def.z(); // call overridden function from def
            super.z(); // call immediate parent overriden function
        }
    }
}

// abstract function
function someAbstractFunction(uint x);
// cannot be compiled, so used in base/abstract contracts
// that are then implemented
```

IMPORTS

USAGE

```
// C. Import

import "filename";
import "github.com/ethereum/dapp-bin/library/iterable_mapping.sol";

// Importing under active development
// Cannot currently be done at command line
```

1.8

KEYWORDS

REVERT, REQUIRE, ASSERT

USAGE

The revert function can be used to flag an error and revert the current call

The require function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

assert acts like it might in any other programming language. Use of revert require should guarantee that your assert does not fail

REVERT, REQUIRE, ASSERT

USAGE

```
pragma solidity ^0.4.0;

contract Sharer {
    function sendHalf(address addr) payable returns (uint balance) {
        require(msg.value % 2 == 0); // Only allow even numbers
        uint balanceBeforeTransfer = this.balance;
        addr.transfer(msg.value / 2);
        // Since transfer throws an exception on failure and
        // cannot call back here, there should be no way for us to
        // still have half of the money.
        assert(this.balance == balanceBeforeTransfer - msg.value / 2);
        return this.balance;
    }
}
```

REVERT, REQUIRE, ASSERT

USAGE

assert(bool condition):

throws if the condition is not met - to be used for internal errors.

require(bool condition):

throws if the condition is not met - to be used for errors in inputs or external components.

revert():

abort execution and revert state changes

SELFDESTRUCT

USAGE

```
// selfdestruct current contract, sending funds to address (often creator)
selfdestruct(SOME_ADDRESS);

// removes storage/code from current/future blocks
// helps thin clients, but previous data persists in blockchain

// Common pattern, lets owner end the contract and receive remaining funds
function remove() {
    if(msg.sender == creator) { // Only let the contract creator do this
        selfdestruct(creator); // Makes contract inactive, returns funds
    }
}

// May want to deactivate contract manually, rather than selfdestruct
// (ether sent to selfdestructed contract is lost)
```

1.9

CONTRACT DESIGN AND MORE

CRYPTOGRAPHIC FUNCTIONS

USAGE

```
// Cryptography
// All strings passed are concatenated before hash action
sha3("ab", "cd");
ripemd160("abc");
sha256("def");
```

COMMITMENT SCHEME

USAGE

```
// All variables are publicly viewable on blockchain, so anything
// that is private needs to be obfuscated (e.g., hashed w/secret)

// Steps: 1. Commit to something, 2. Reveal commitment
sha3("some_bid_amount", "some secret"); // commit

// call contract's reveal function in the future
// showing bid plus secret that hashes to SHA3
reveal(100, "mySecret");
```

STORAGE

TIPS

- Storage is expensive because it is permanent
- Things like multidimensional arrays are expensive
- Store things OFF THE BLOCKCHAIN **unless absolutely necessary**

STORAGE

TIPS

THERE ARE NO SECRETS ON THE BLOCKCHAIN

All data on the blockchain can be easily viewed by anyone in the world

UNITS OF TIME

USAGE

```
// Currency is defined using wei, smallest unit of Ether
uint minAmount = 1 wei;
uint a = 1 finney; // 1 ether == 1000 finney
// Time units
1 == 1 second
1 minutes == 60 seconds

// Can multiply a variable times unit, as units are not stored in a variable
uint x = 5;
(x * 1 days); // 5 days

// Careful about leap seconds/years with equality statements for time
// (instead, prefer greater than/less than)
```

DOCUMENTING CONTRACTS

USAGE

Contract natspec - always above contract definition

```
/// @title Contract title
/// @author Author name
```

Function natspec

```
/// @notice information about what function does; shown when
function to execute
/// @dev Function documentation for developer
```

Function parameter/return value natspec

```
/// @param someParam Some description of what the param does
/// @return Description of the return value
```

STYLE
GUIDELINES

Quick summary:

- 4 spaces for indentation
- Two lines separate contract declarations (and other top level declarations)
- Avoid extraneous spaces in parentheses
- Can omit curly braces for one line statement (if, for, etc)
- else should be placed on own line

2

HANDS ON DEVELOPMENT

BLOCKCHAIN
AT BERKELEY

CONCLUSION

VFC JUNE 2017

Blockchain technology (and distributed tech) is going to lead to a new world of decentralization.

Remember:

- Why is blockchain better than using a centralized database?
- Almost any app that exists today can (theoretically) be built in a completely decentralized way
- Layered Approach
- The only limitation is what we can secure and scale

QUESTIONS

UNIQUE VALUE PROPOSITION

VFC JUNE 2017

- Our mission is legitimacy, education, and impact
  - Not incentivized by money
  - Not bound to a platform
- Interface with academia and business
- Focus on the tangible