Join Our Telegram

https://t.me/ntublockchain

BLOCKCHAIN
AT NTU

Lecture 4:
Smart Contracts & Solidity

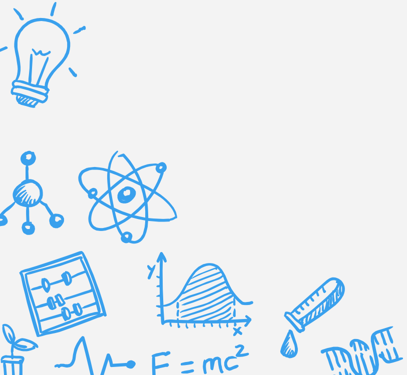
Phang Jun Yu

BLOCKCHAIN
AT NTU

Revision

Recap

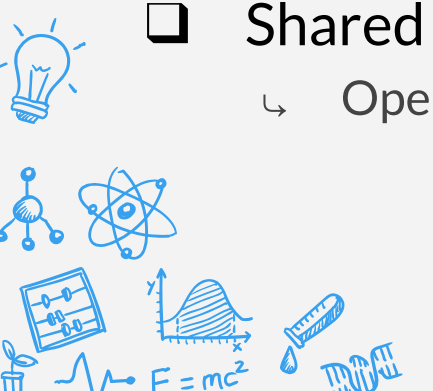
- ❑ Transaction & Messages
- ❑ Gas



BLOCKCHAIN
AT NTU

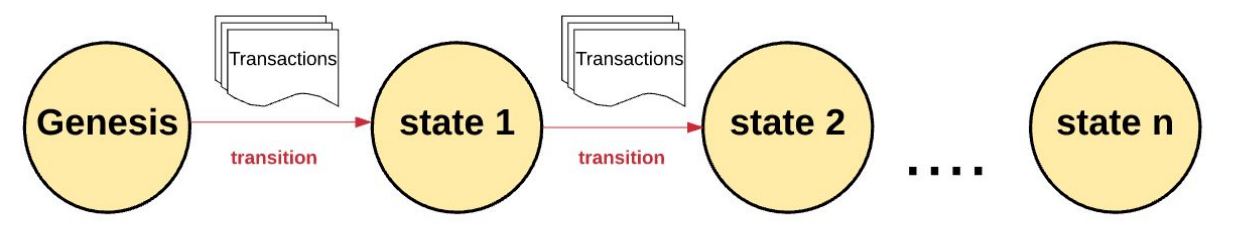
Ethereum Blockchain

- A blockchain is a “**cryptographically secure transactional singleton machine** with **shared state**”
- Cryptographically secure
  - Digital signatures, Hashes, etc
- Transactional Singleton Machine
  - Single instance of the machine for all transactions (global truth)
- Shared State
  - Open and shared by everybody



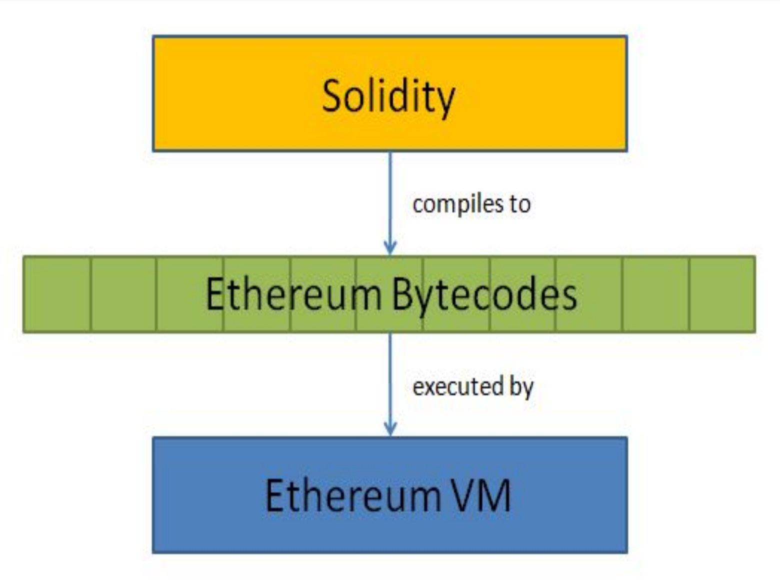
Ethereum Virtual Machine

- ❑ Transaction-based state machine.
- ❑ A state machine refers to something that will read a series of inputs and, based on those inputs, will transition to a new state.



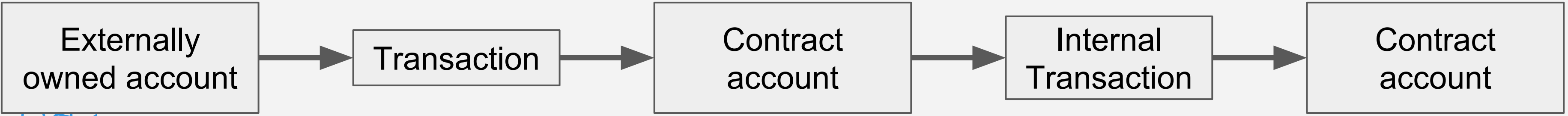
Ethereum Virtual Machine

- EVM is a system designed to operate as a **<u>runtime environment for smart contracts</u>**
- High level programming (smart contract) languages:
    - ↳ Solidity
    - ↳ Vyper
    - ↳ Serpent

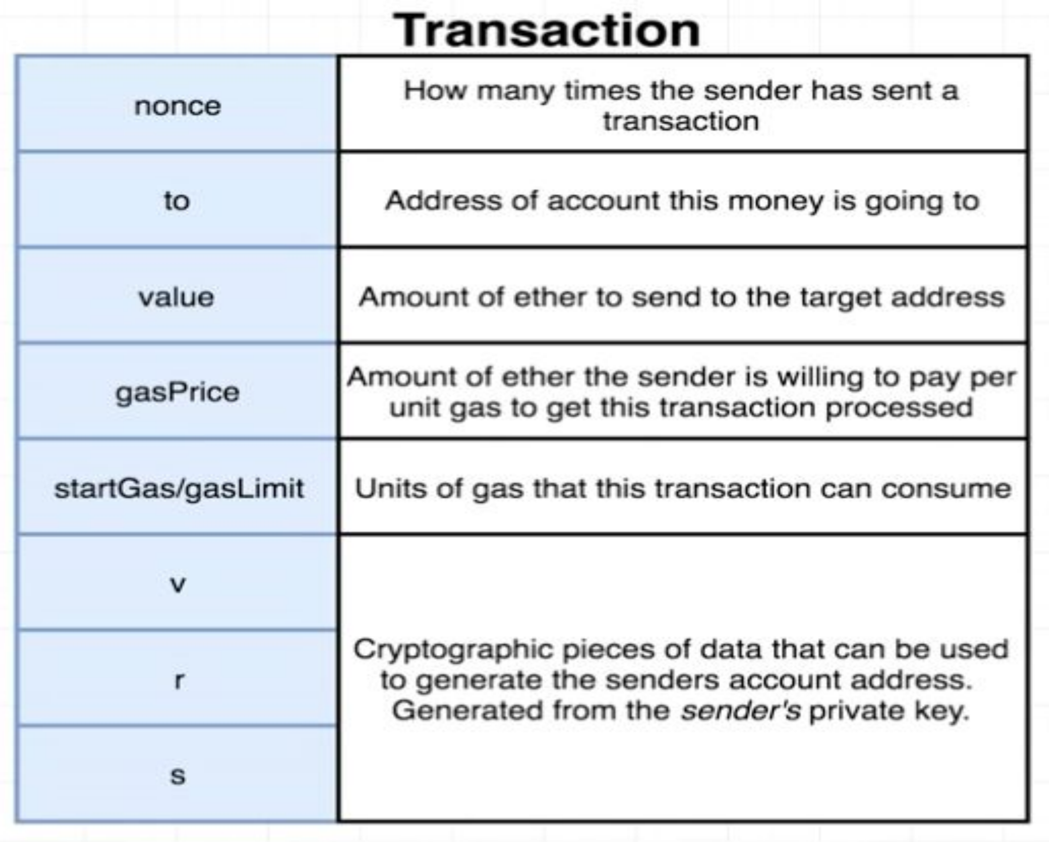

Solidity
compiles to
Ethereum Bytecodes
executed by
Ethereum VM

Transactions

- Transactions can only be sent from an externally owned account
  - Call (read)
  - Send (write)
- Messages (internal transactions)
  - Invoked by transactions
  - Not explicitly published on the blockchain



Transactions

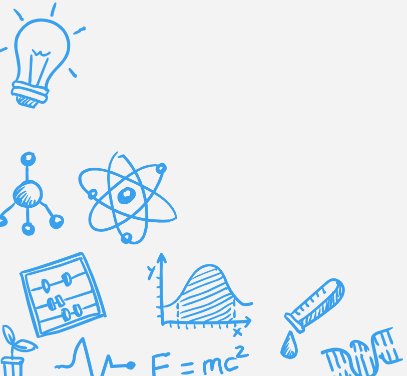


Transaction

nonce	How many times the sender has sent a transaction
to	Address of account this money is going to
value	Amount of ether to send to the target address
gasPrice	Amount of ether the sender is willing to pay per unit gas to get this transaction processed
startGas/gasLimit	Units of gas that this transaction can consume
v	Cryptographic pieces of data that can be used to generate the senders account address. Generated from the *sender's* private key.
r	
s	

Gas

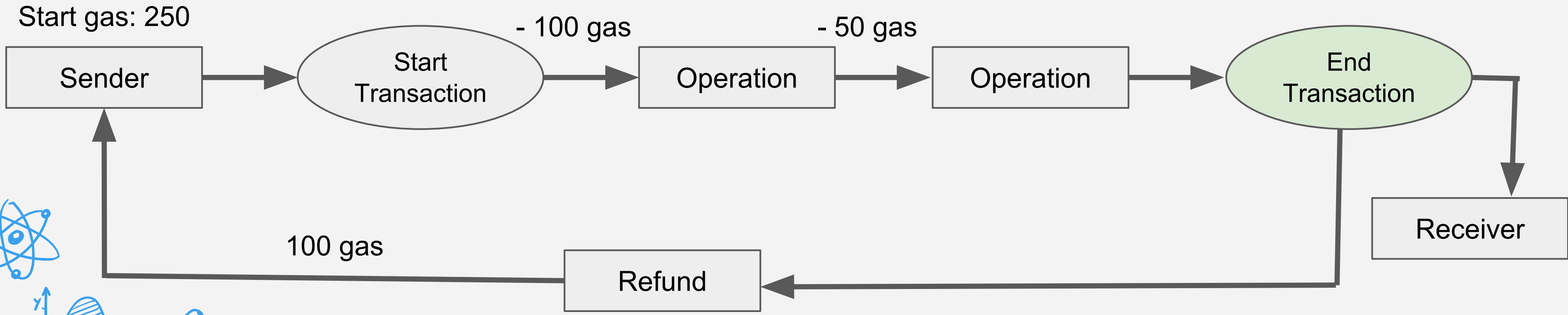
- Every **<u>computation</u>** and **<u>transaction</u>** incurs a *gas fee*
- Fee prevents:
  - Infinite Loops (remember that EVM is Turing Complete)
  - Denial of Service (disincentivises network spamming)
- Gas is the **<u>unit</u>** used to measure fees
  - Measured in *gwei* - 1 Eth = 1,000,000,000 *gwei*



BLOCKCHAIN
AT NTU

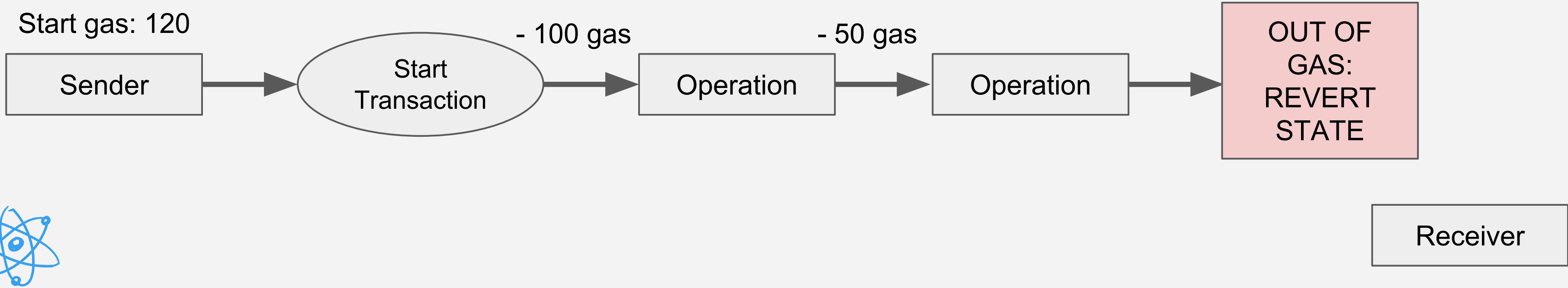
Gas

- User sets **gas limit** and **gas price** in each transaction
  - gas fee = gas limit * gas price
  - Higher prices incentivises miners to include transaction in next block
  - Unused gas is refunded
  - During computation if gas runs out, only tx value is refunded, not gas fee



Gas

- ❑ User sets **gas limit** and **gas price** in each transaction
  - ↳ gas fee = gas limit * gas price
  - ↳ Higher prices incentivises miners to include transaction in next block
  - ↳ Unused gas is refunded
  - ↳ During computation if gas runs out, only tx value is refunded, not gas fee



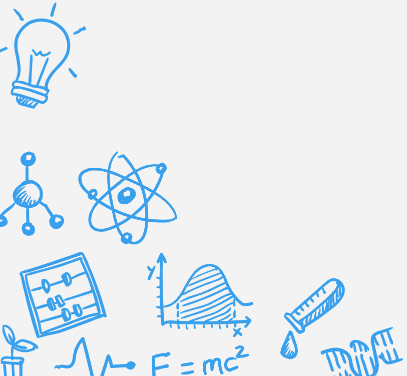
Start gas: 120

Sender → Start Transaction → - 100 gas → Operation → - 50 gas → Operation → OUT OF GAS: REVERT STATE

Receiver

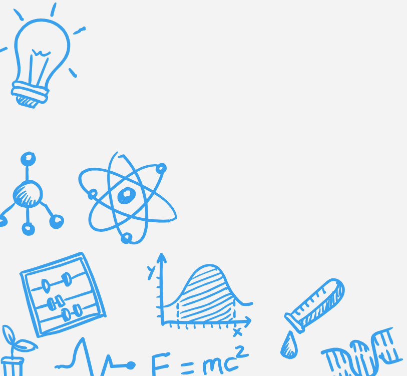
Gas

- Gas is also used to pay for storage (stored on all nodes)
  - Proportional to storage size
  - Incentivises users to keep data small
  - If tx frees data space, gas will be refunded



Agenda

- ☐ **Motivation**: Smart Contracts
- ☐ **Solidity**: Syntax & Data Structures
- ☐ **Practice**: Simple Contracts



Smart Contracts

Smart Contracts - What are they?

Let's say you want to sell a house

- ❑ Trust with payment issues
- ❑ Complex Paperwork
- ❑ Hire real estate agent
- ❑ Commission fees

Smart Contract:

- ❑ If-Then
- ❑ Escrow Service

Smart Contracts

Autonomy

Savings

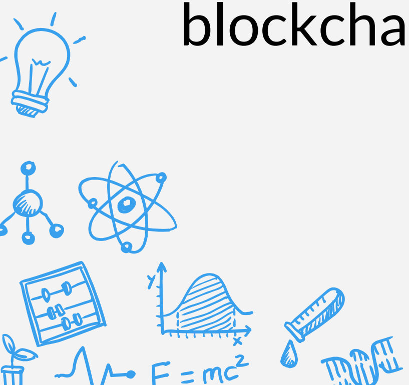
Efficiency

Trust

Safety

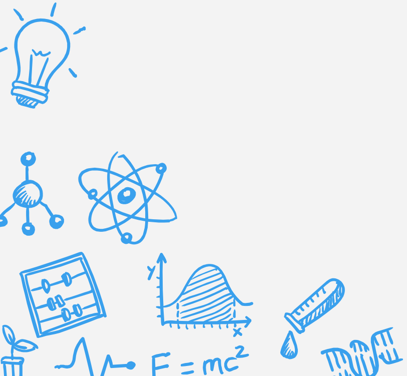
Smart Contracts - Trust and Safety

- ☐ A Program on the Ethereum Blockchain - executed by the EVM
- ☐ Violation of contract requires subverting the entire network
- ☐ Allows for secure Peer-to-Peer agreements that can stay on the blockchain forever



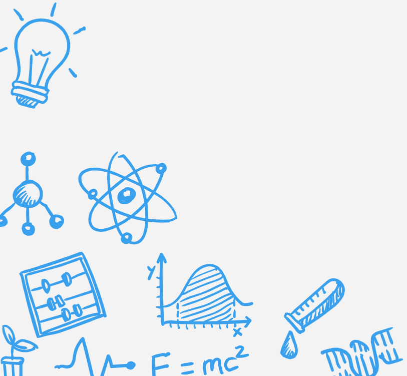
Smart Contracts - Why?

- ☐ Honest Computing (Honest)
- ☐ Tamper-Proof
- ☐ No single point of failure
  - ↳ Decentralized
- ☐ Autonomous code
  - ↳ AWS?



Smart Contracts

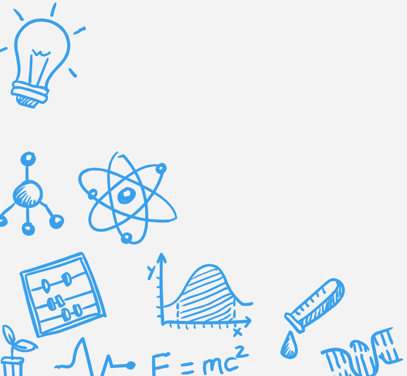
- React to external world when “triggered” by **transactions** that call functions
- Have direct control over:
  - Internal ether balance
  - Internal contract state
  - Permanent storage



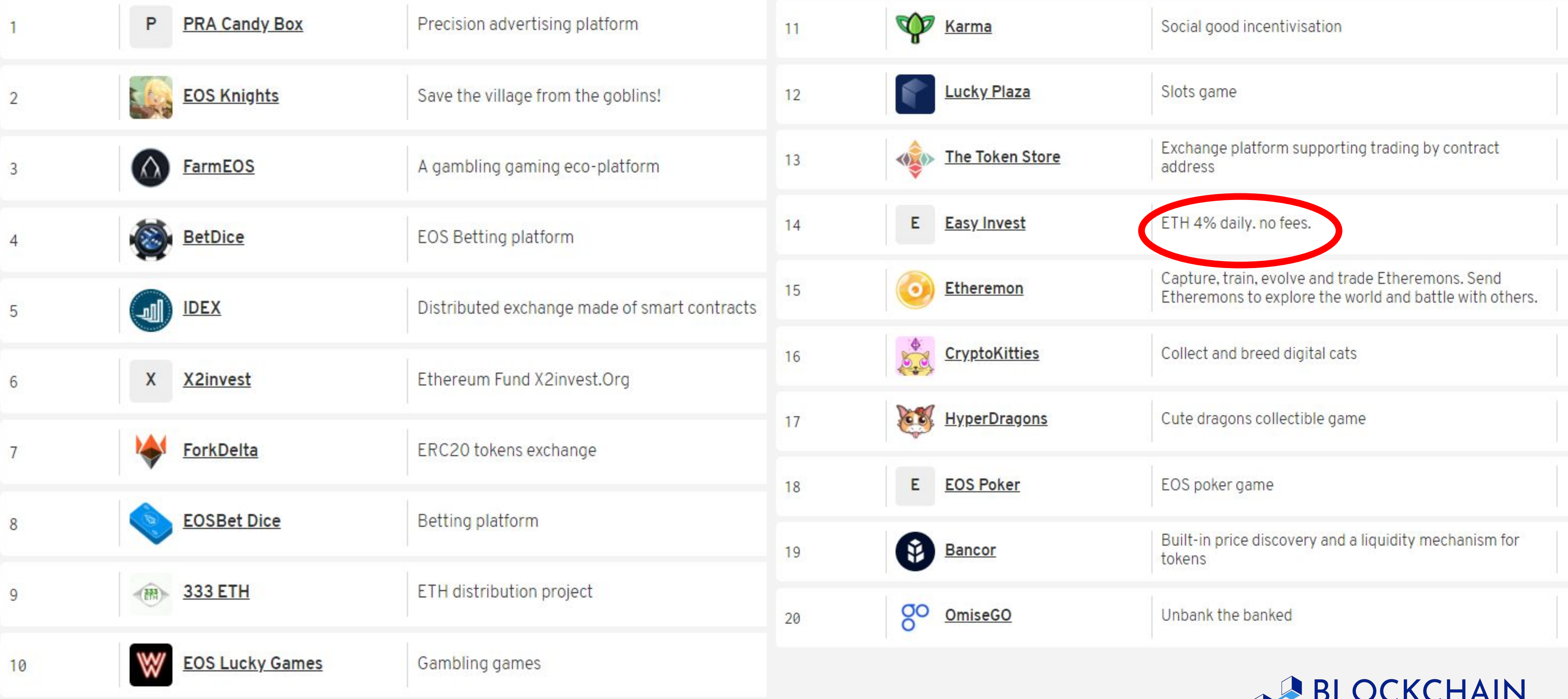
Smart Contracts

What can they do?

❑ Store and maintain data
❑ Manage contract or relationship between untrusting parties
❑ Provide functions to other contracts
❑ Complex authentication



Applications of Smart Contracts?



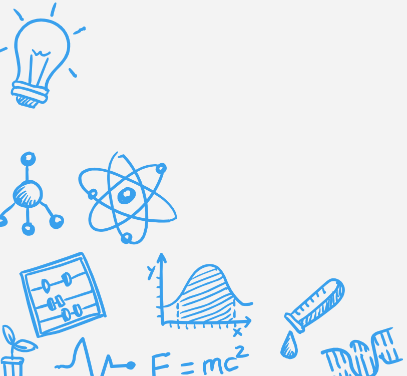
1	PRA Candy Box	Precision advertising platform
2	EOS Knights	Save the village from the goblins!
3	FarmEOS	A gambling gaming eco-platform
4	BetDice	EOS Betting platform
5	IDEX	Distributed exchange made of smart contracts
6	X2invest	Ethereum Fund X2invest.Org
7	ForkDelta	ERC20 tokens exchange
8	EOSBet Dice	Betting platform
9	333 ETH	ETH distribution project
10	EOS Lucky Games	Gambling games
11	Karma	Social good incentivisation
12	Lucky Plaza	Slots game
13	The Token Store	Exchange platform supporting trading by contract address
14	Easy Invest	ETH 4% daily. no fees.
15	Etheremon	Capture, train, evolve and trade Etheremons. Send Etheremons to explore the world and battle with others.
16	CryptoKitties	Collect and breed digital cats
17	HyperDragons	Cute dragons collectible game
18	EOS Poker	EOS poker game
19	Bancor	Built-in price discovery and a liquidity mechanism for tokens
20	OmiseGO	Unbank the banked

Solidity

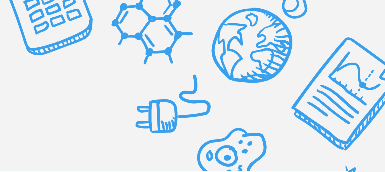
BLOCKCHAIN
AT NTU

What is Solidity?

- Solidity is a **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.



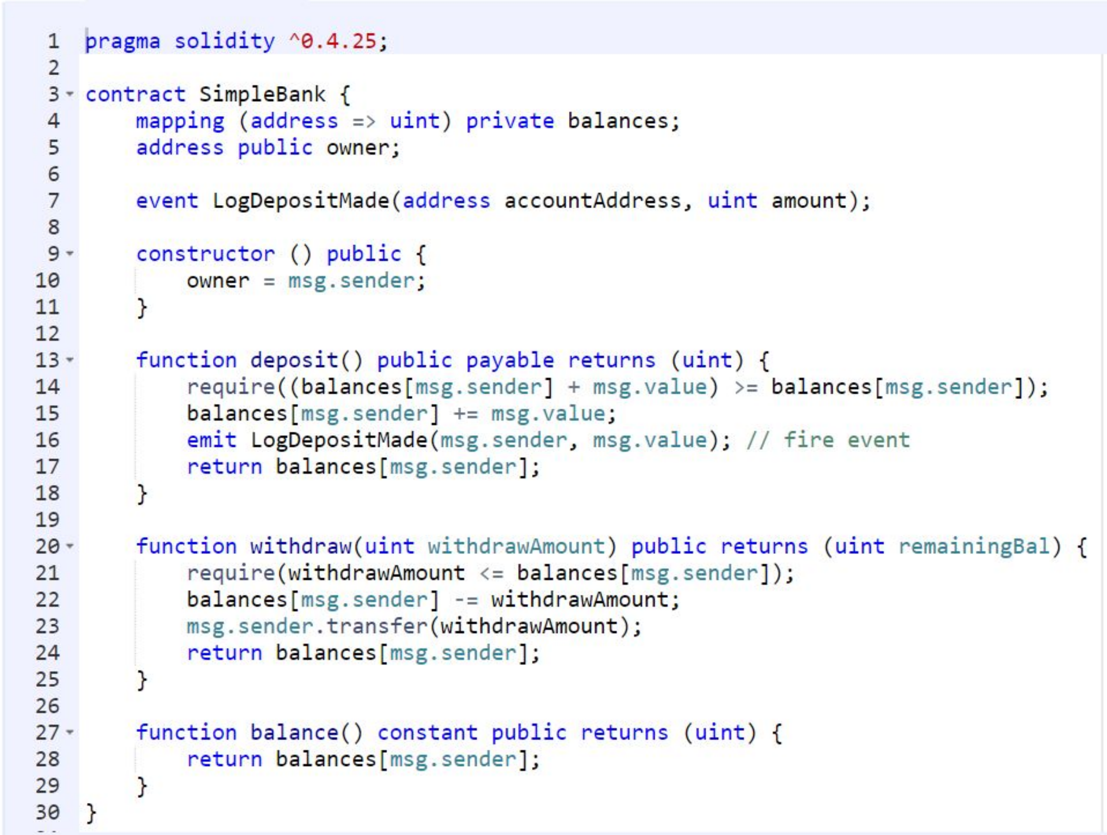
Let’s look at a simple Bank contract?



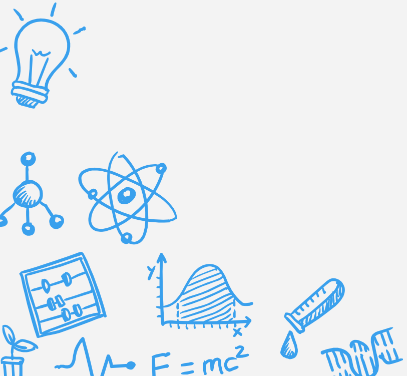
What does a bank need to do?

- ☐ Deposit funds
- ☐ Withdraw funds
- ☐ Check balances

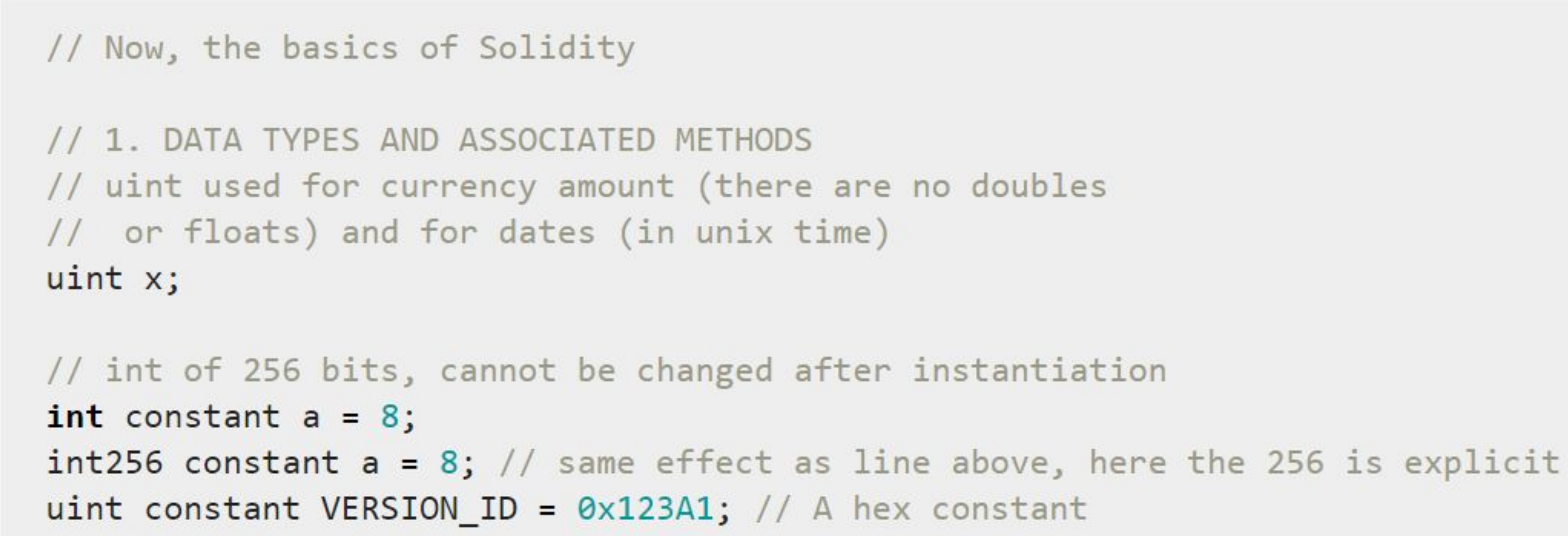
https://learnxinyminutes.com/docs/solidity/



```
pragma solidity ^0.4.25;

contract SimpleBank {
    mapping (address => uint) private balances;
    address public owner;

    event LogDepositMade(address accountAddress, uint amount);

    constructor () public {
        owner = msg.sender;
    }

    function deposit() public payable returns (uint) {
        require((balances[msg.sender] + msg.value) >= balances[msg.sender]);
        balances[msg.sender] += msg.value;
        emit LogDepositMade(msg.sender, msg.value); // fire event
        return balances[msg.sender];
    }

    function withdraw(uint withdrawAmount) public returns (uint remainingBal) {
        require(withdrawAmount <= balances[msg.sender]);
        balances[msg.sender] -= withdrawAmount;
        msg.sender.transfer(withdrawAmount);
        return balances[msg.sender];
    }

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

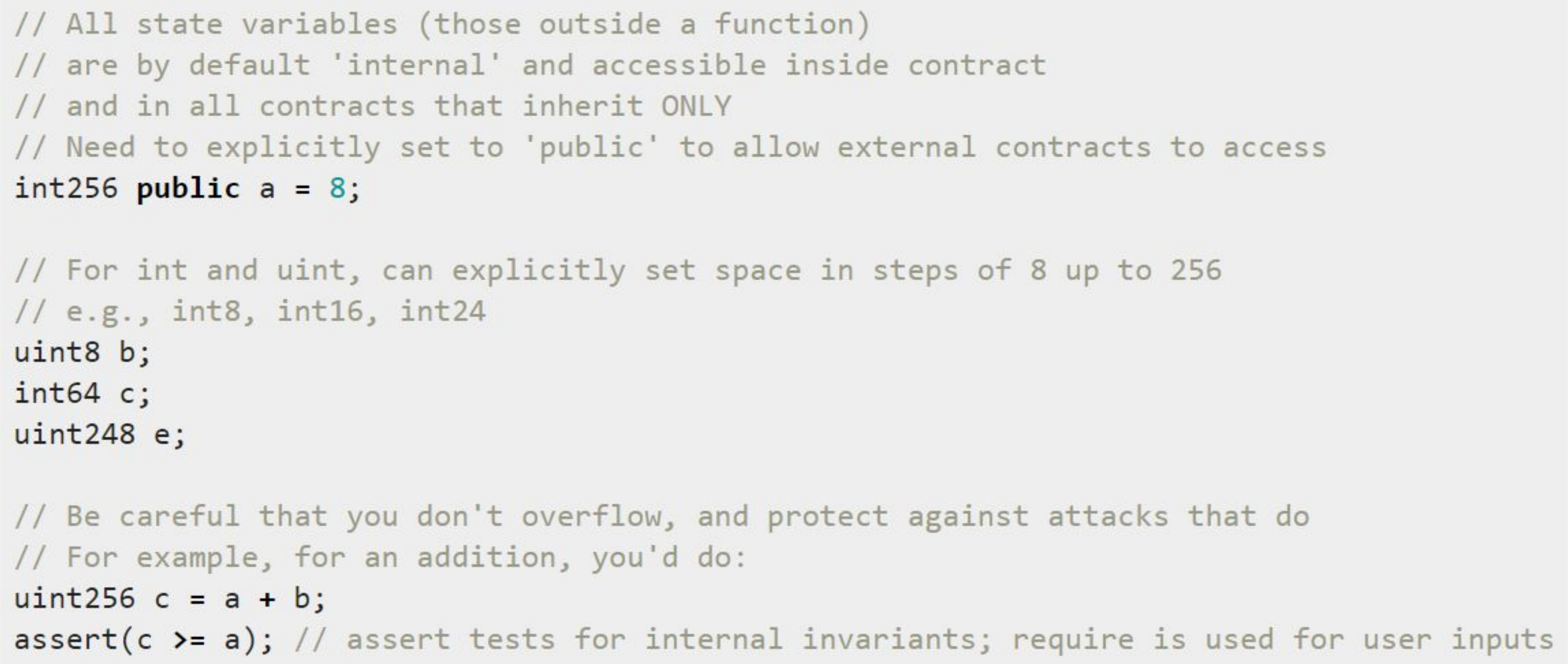


Integers



```
// Now, the basics of Solidity

// 1. DATA TYPES AND ASSOCIATED METHODS
// 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
```

Integers

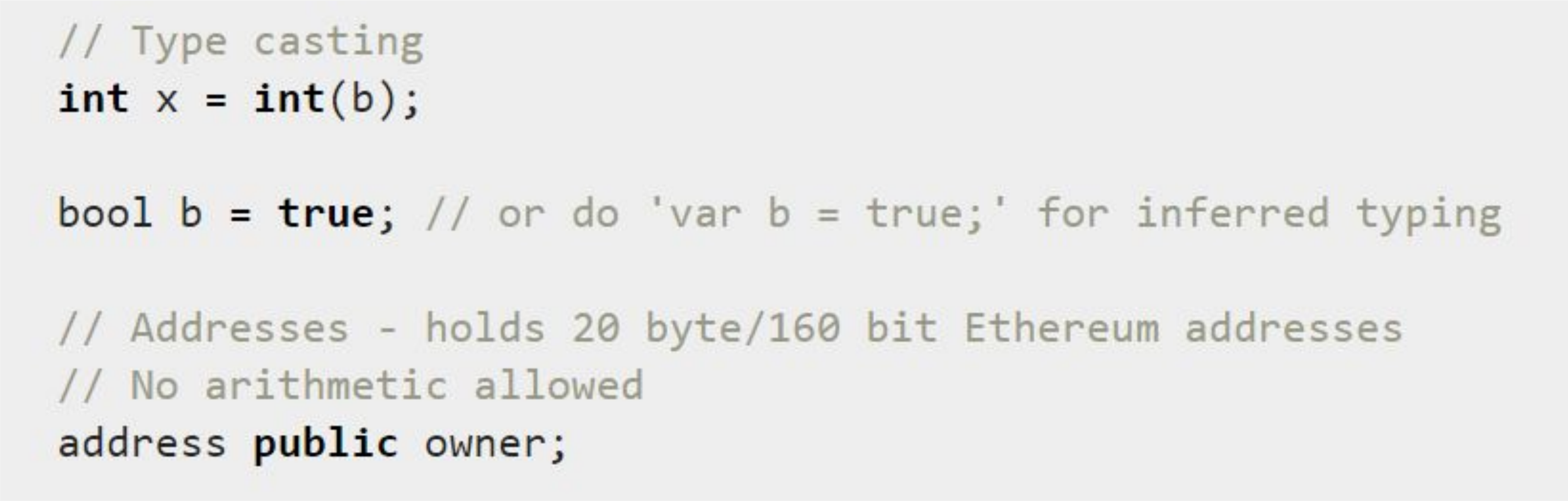


```
// All state variables (those outside a function)
// are by default 'internal' and accessible inside contract
// and in all contracts that inherit ONLY
// Need to explicitly set to 'public' to allow external contracts to access
int256 public a = 8;

// 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
// For example, for an addition, you'd do:
uint256 c = a + b;
assert(c >= a); // assert tests for internal invariants; require is used for user inputs
```

Quick Demo

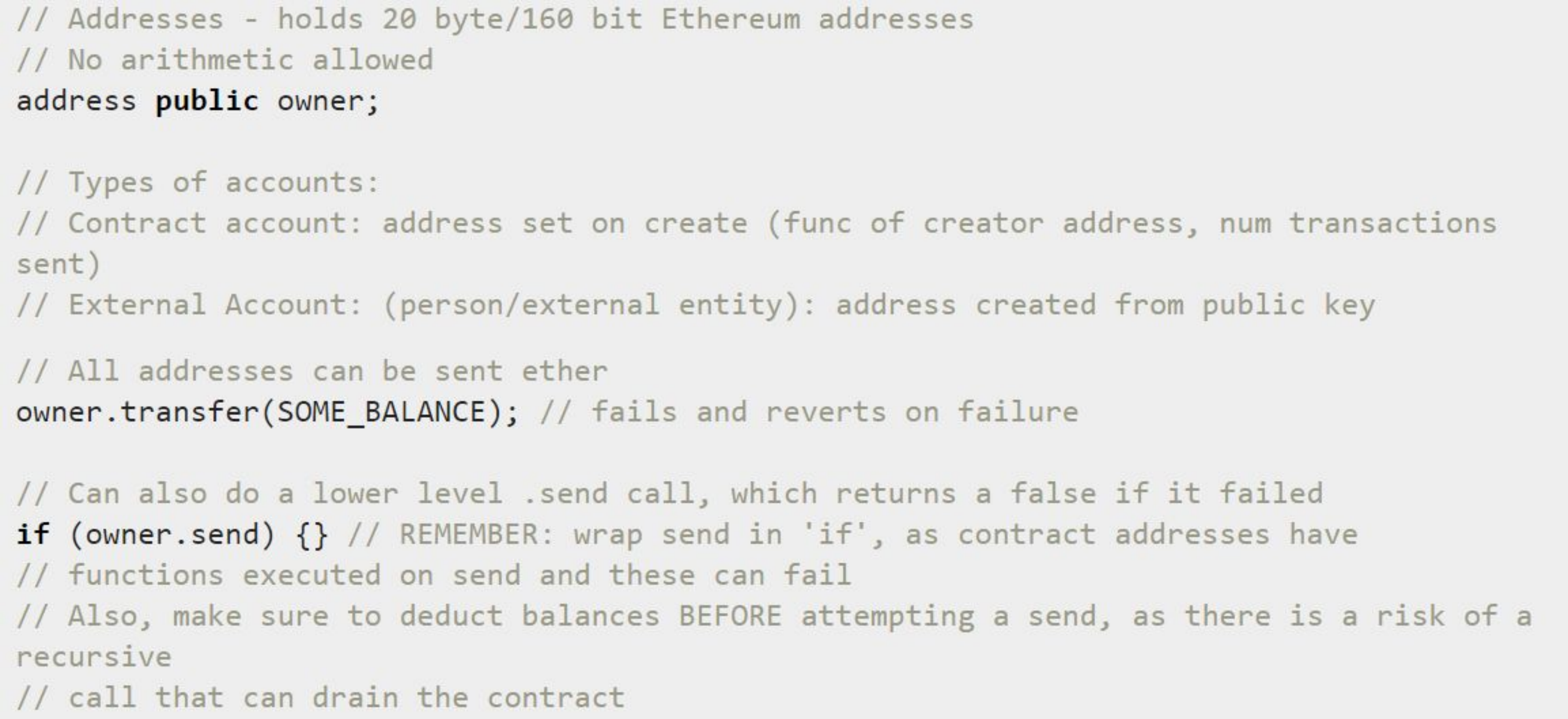
http://remix.ethereum.org

Type Casting and Booleans



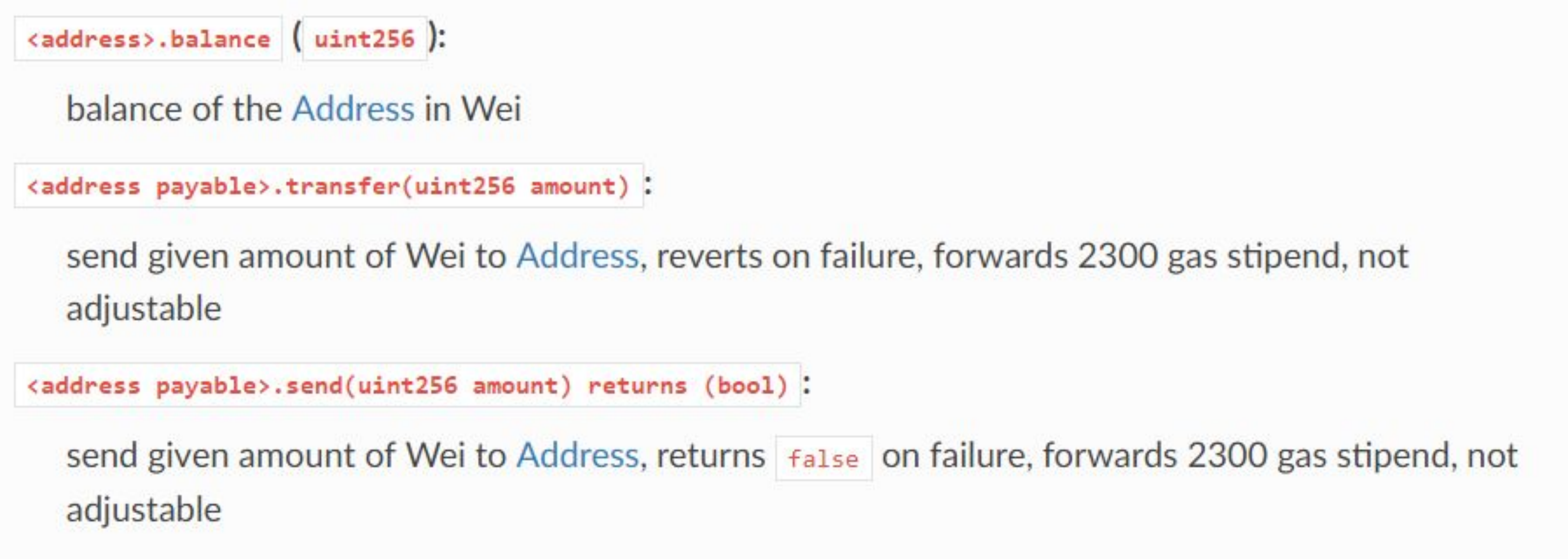
```
// Type casting
int x = int(b);

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;
```

Addresses



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

// 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

// All addresses can be sent ether
owner.transfer(SOME_BALANCE); // fails and reverts on failure

// Can also do a lower level .send call, which returns a false if it failed
if (owner.send) {} // REMEMBER: wrap send 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
```

Addresses



`<address>.balance` (`uint256`):

balance of the Address in Wei

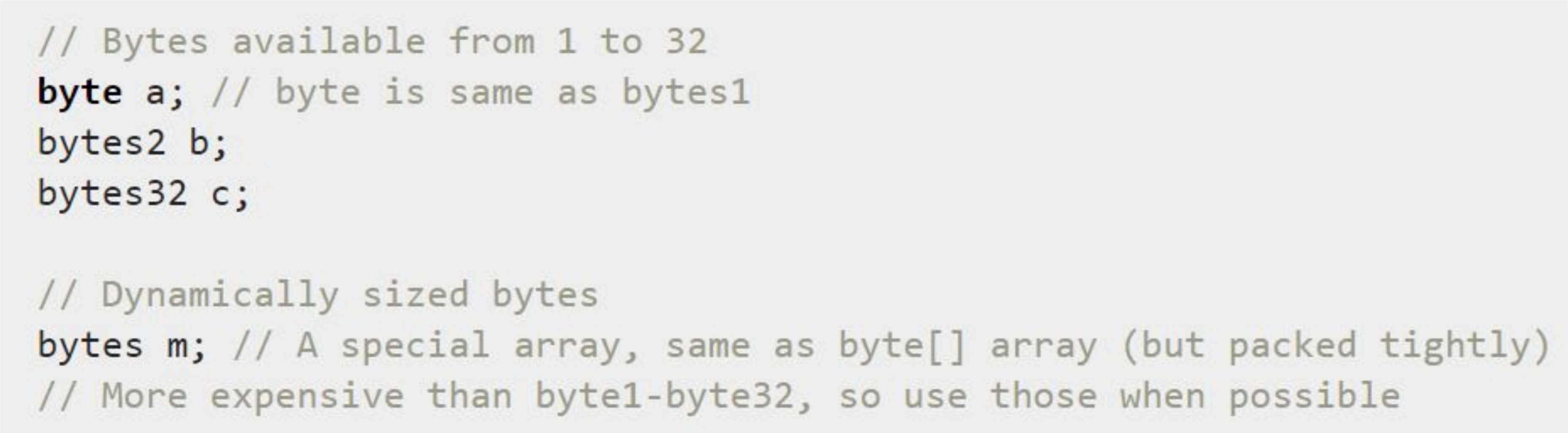
`<address payable>.transfer(uint256 amount)`:

send given amount of Wei to Address, reverts on failure, forwards 2300 gas stipend, not adjustable

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

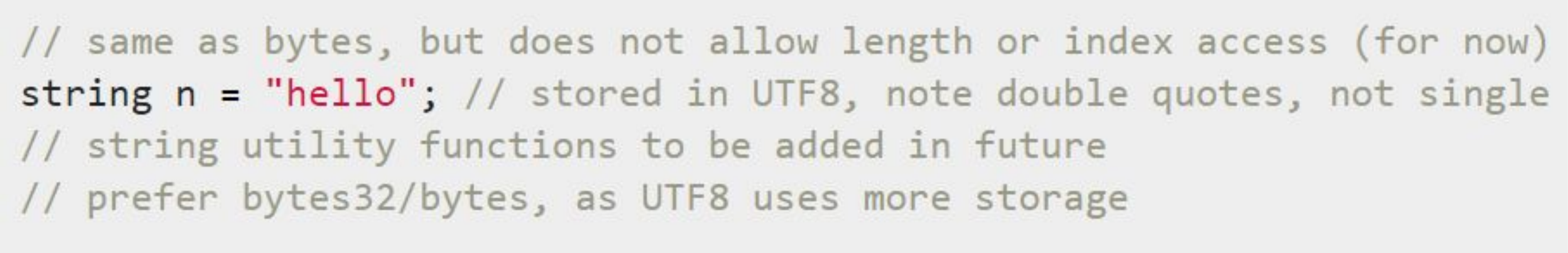
send given amount of Wei to Address, returns `false` on failure, forwards 2300 gas stipend, not adjustable

Bytes

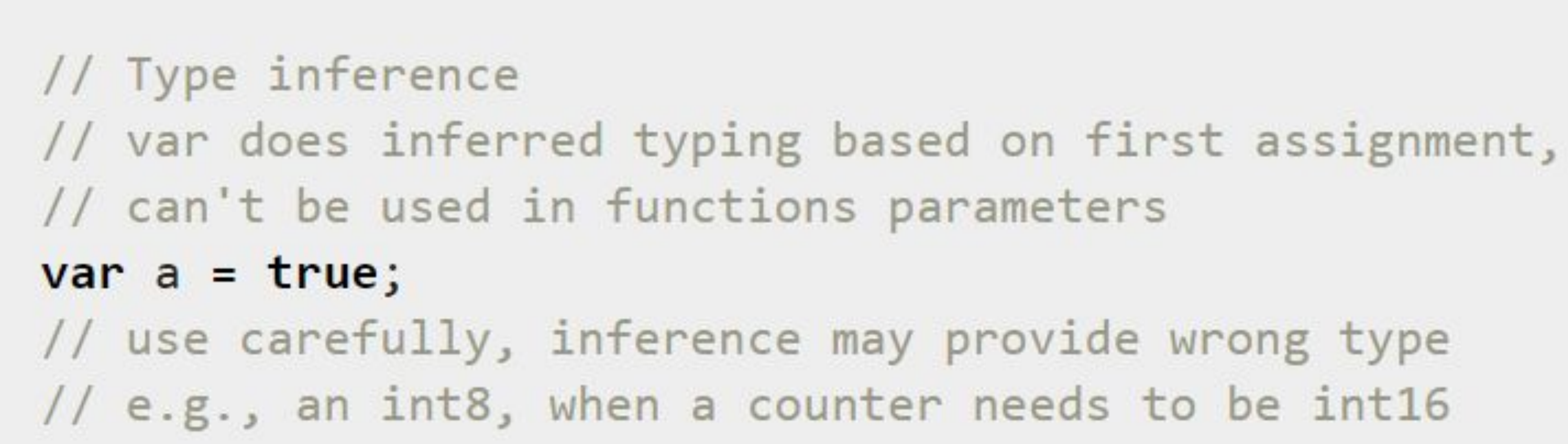


```
// Bytes available from 1 to 32
byte a; // byte is same as bytes1
bytes2 b;
bytes32 c;

// 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
```

Bytes & Type Casting

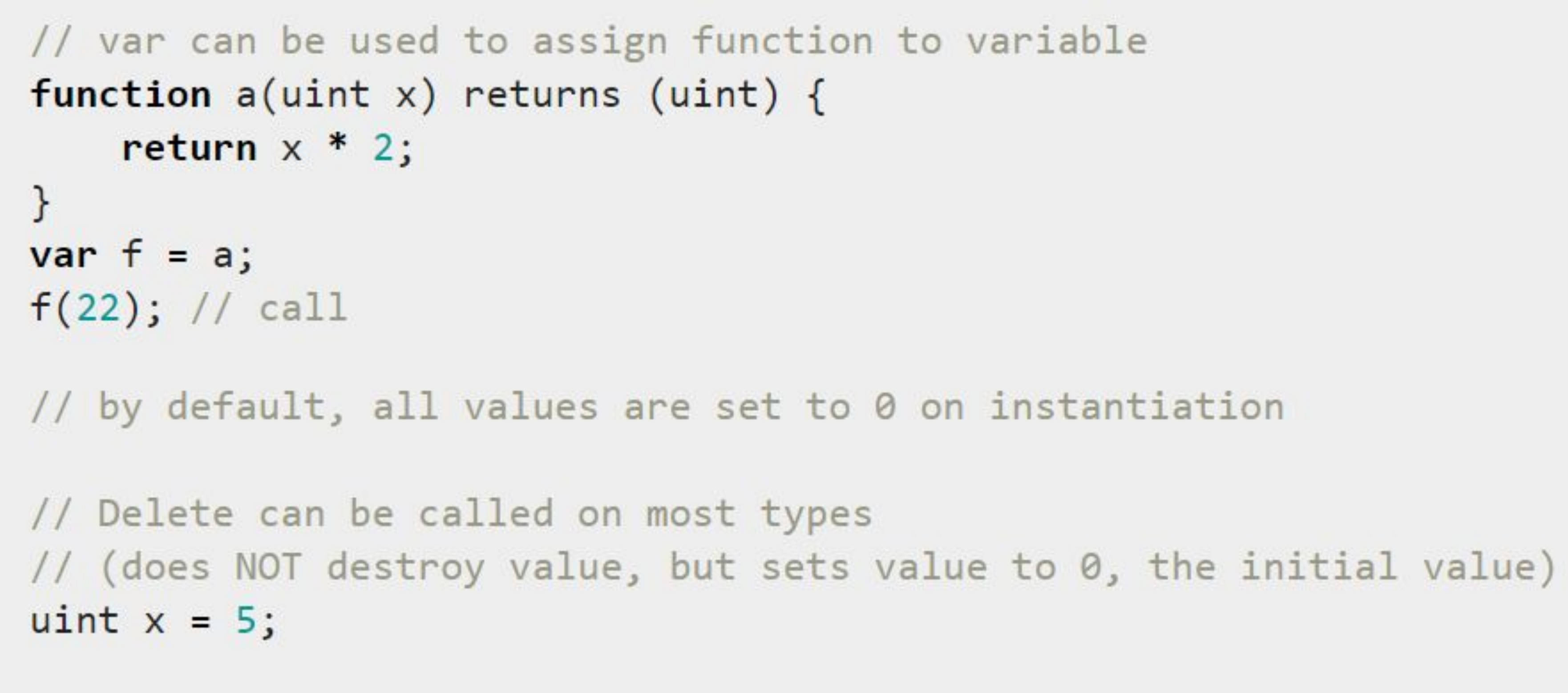


```
// same as bytes, but does not allow length or index access (for now)
string n = "hello"; // stored in UTF8, note double quotes, not single
// string utility functions to be added in future
// prefer bytes32/bytes, as UTF8 uses more storage
```



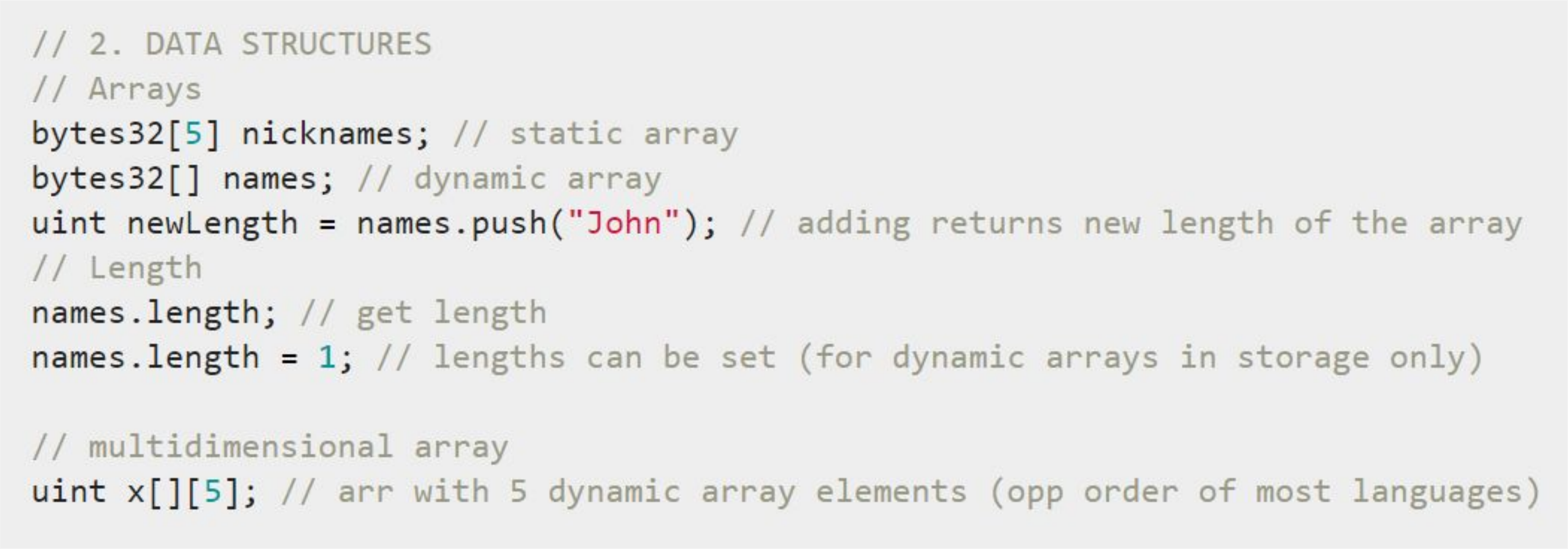
```
// Type inference
// var does inferred typing based on first assignment,
// can't be used in functions parameters
var a = true;
// use carefully, inference may provide wrong type
// e.g., an int8, when a counter needs to be int16
```

Function Assignments & Delete



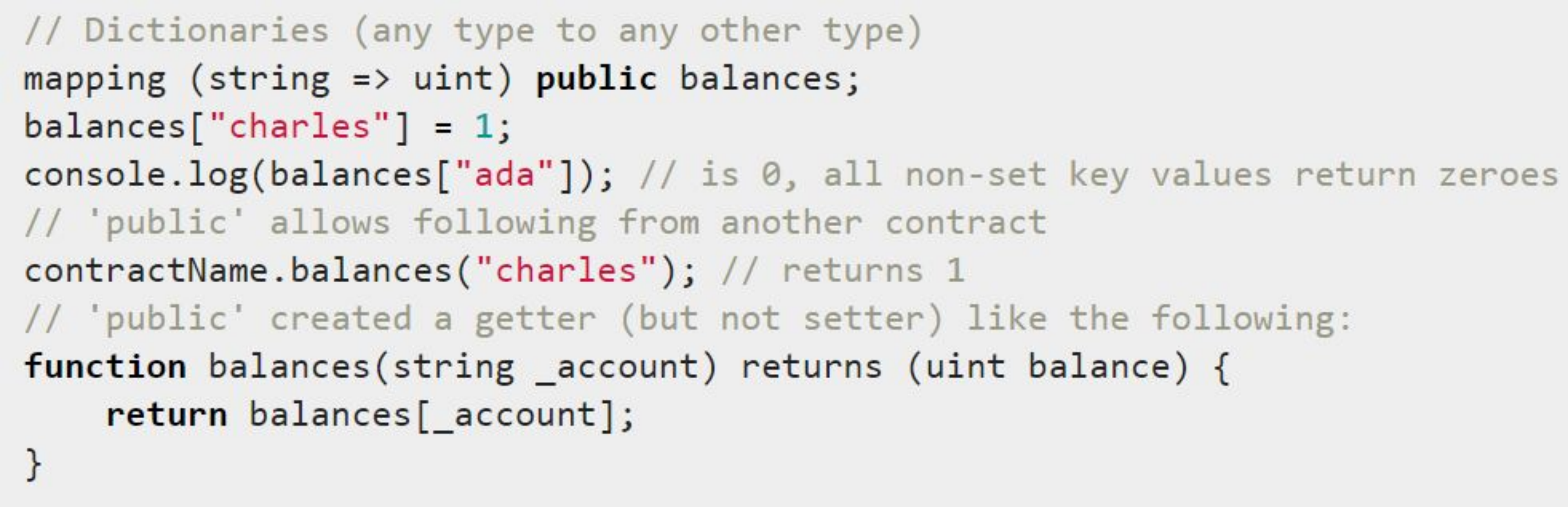
```
// var can be used to assign function to variable
function a(uint x) returns (uint) {
    return x * 2;
}
var f = a;
f(22); // call

// 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;
```

Data Structures - Arrays



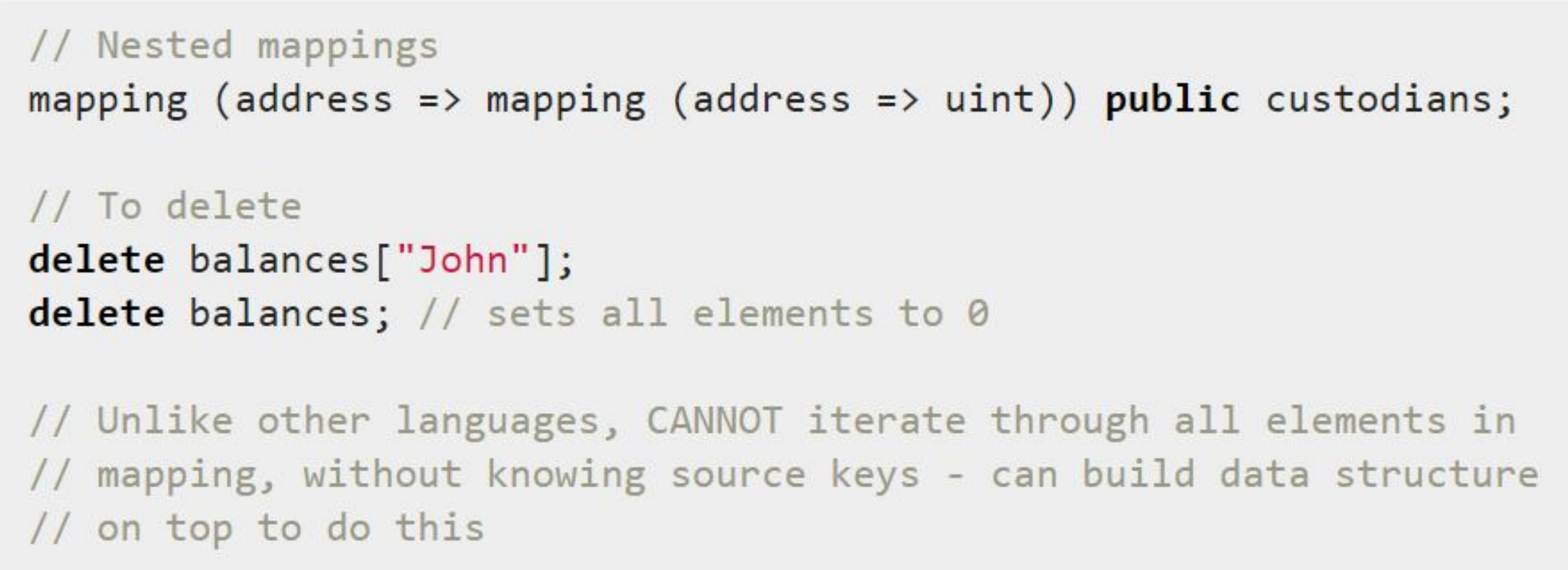
```
// 2. DATA STRUCTURES
// Arrays
bytes32[5] nicknames; // static array
bytes32[] names; // dynamic array
uint newLength = names.push("John"); // adding returns new length of the array
// Length
names.length; // get length
names.length = 1; // lengths can be set (for dynamic arrays in storage only)

// multidimensional array
uint x[][5]; // arr with 5 dynamic array elements (opp order of most languages)
```

Data Structures - Mapping



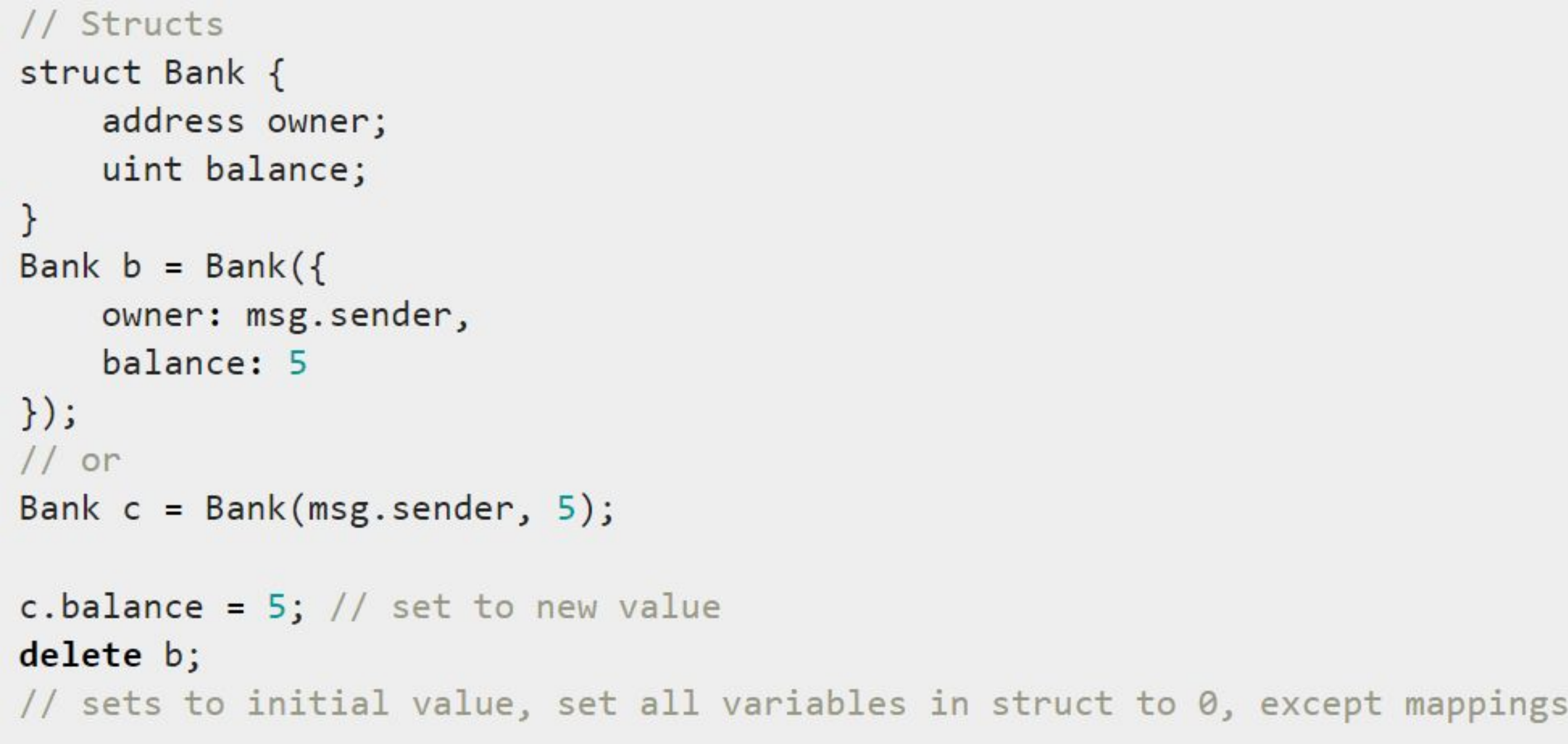
```
// Dictionaries (any type to any other type)
mapping (string => uint) public balances;
balances["charles"] = 1;
console.log(balances["ada"]); // is 0, all non-set key values return zeroes
// 'public' allows following from another contract
contractName.balances("charles"); // returns 1
// 'public'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

Data Structures - Nested Mappings



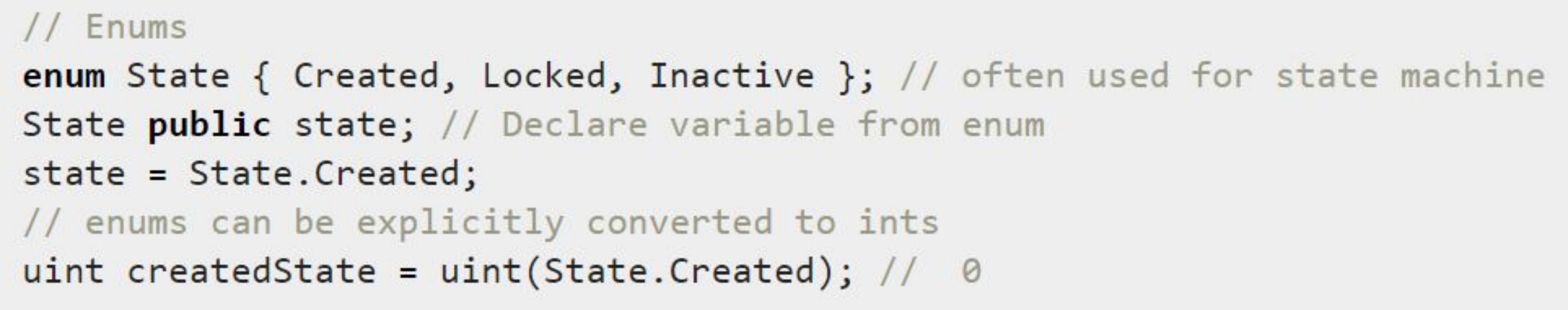
```
// Nested mappings
mapping (address => mapping (address => uint)) public custodians;

// 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



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

c.balance = 5; // set to new value
delete b;
// sets to initial value, set all variables in struct to 0, except mappings
```

Data Structures - Enums



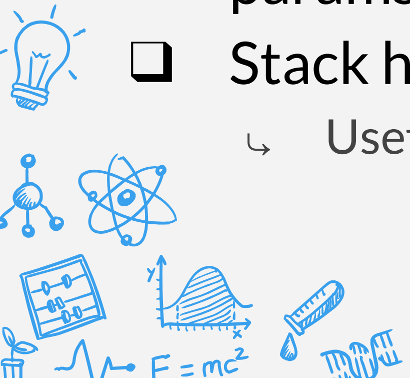
```
// Enums
enum State { Created, Locked, Inactive }; // often used for state machine
State public state; // Declare variable from enum
state = State.Created;
// enums can be explicitly converted to ints
uint createdState = uint(State.Created); //  0
```

Data Storage

Data locations: *Memory* vs. *storage* vs. *stack*- all complex types (arrays, structs) have a data location

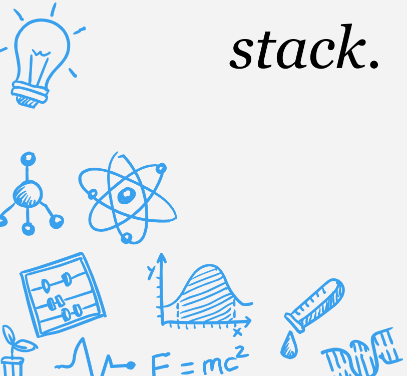
- ❑ *Memory* does not persist, *storage* does
  - ↳ RAM vs Hard Disk
- ❑ Default is *storage* for local and state variables; *memory* for func params
- ❑ Stack holds small local variables
  - ↳ Useful for values in intermediate calculations

Find out more here



Data Storage

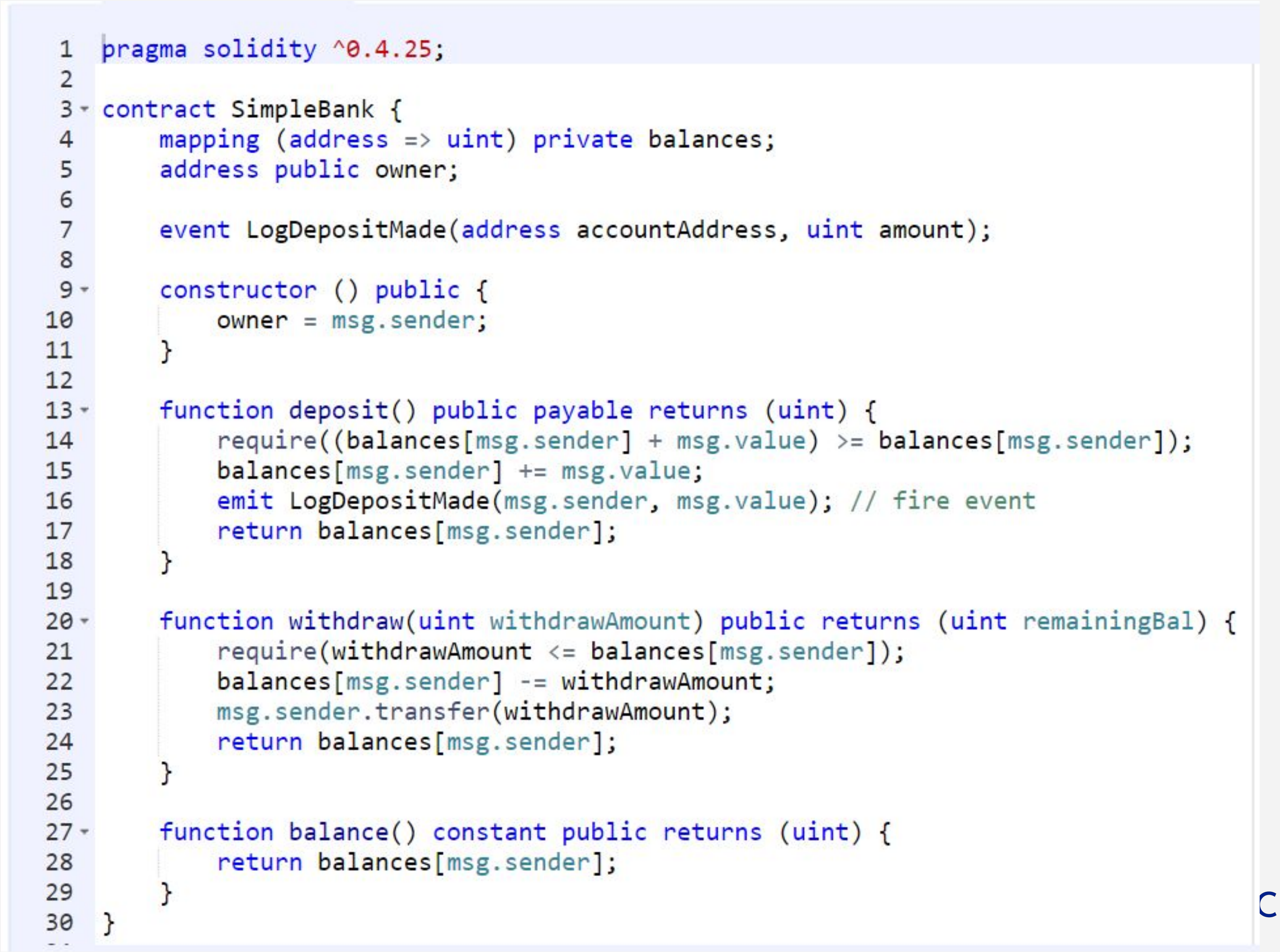
- ❑ State variables are always in *storage*
- ❑ Function arguments are in *memory*
- ❑ Local variables of **struct**, **array** or **mapping** type reference are stored in *storage* by default.
- ❑ Local variables of a value type (uint, int etc) are stored in the *stack*.



Data Storage

- Private: Only called by functions within the contract itself
- Internal: Can be called by functions within the contract and descendants of this contract
- External: Can only be called by functions from outside of this contract
- Public: Can be called by functions within or outside of the contract
- Link: https://solidity.readthedocs.io/en/v0.4.25/contracts.html?highlight=protected#visibility-and-getters

Oh!



```
pragma solidity ^0.4.25;

contract SimpleBank {
    mapping (address => uint) private balances;
    address public owner;

    event LogDepositMade(address accountAddress, uint amount);

    constructor () public {
        owner = msg.sender;
    }

    function deposit() public payable returns (uint) {
        require((balances[msg.sender] + msg.value) >= balances[msg.sender]);
        balances[msg.sender] += msg.value;
        emit LogDepositMade(msg.sender, msg.value); // fire event
        return balances[msg.sender];
    }

    function withdraw(uint withdrawAmount) public returns (uint remainingBal) {
        require(withdrawAmount <= balances[msg.sender]);
        balances[msg.sender] -= withdrawAmount;
        msg.sender.transfer(withdrawAmount);
        return balances[msg.sender];
    }

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

Practice Time!

http://remix.ethereum.org/

Let’s Practice!

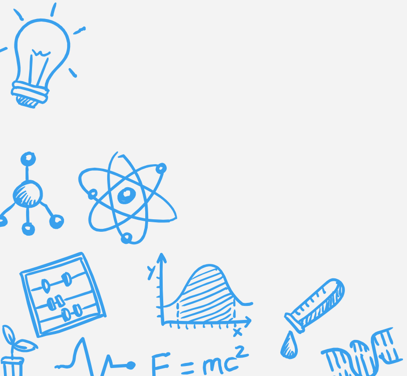
Using the Remix IDE:

- ☐ Write a “Greeter” Contract with a “greet” method that returns the string “Hello NTU!”
- ☐ Bonus: Allow the user to change the greeting string without re-deploying the contract

Let’s Practice!

Using the Remix IDE:

- ☐ Output the Fibonacci sequence - iteratively
  - ↳ The function should have a method that takes the value’s position in the Fibonacci sequence as an input
  - ↳ For example: 1, 1, 2, 3, 5, 8, 13, ….
  - ↳ function fib_seq (6) should return 8

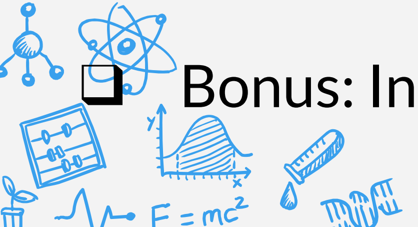


BLOCKCHAIN
AT NTU

Let’s Practice!

Using the Remix IDE:

- ☐ Write an “XOR” function
  - ↳ Inputs should be either 1 or 0
  - ↳ Should not require bitwise operations
  - ↳ Example: given an input of 1 and 0, the function should return 1
  - ↳ Given an input of 0 and 0, return 0
  - ↳ Given an input of 0 and 1, return 1
  - ↳ Given an input of 1 and 1, return 0



- ☐ Bonus: Input a string of 1’s and 0’s e.g. “100001010”

BLOCKCHAIN AT NTU

Let’s Practice!

Using the Remix IDE:

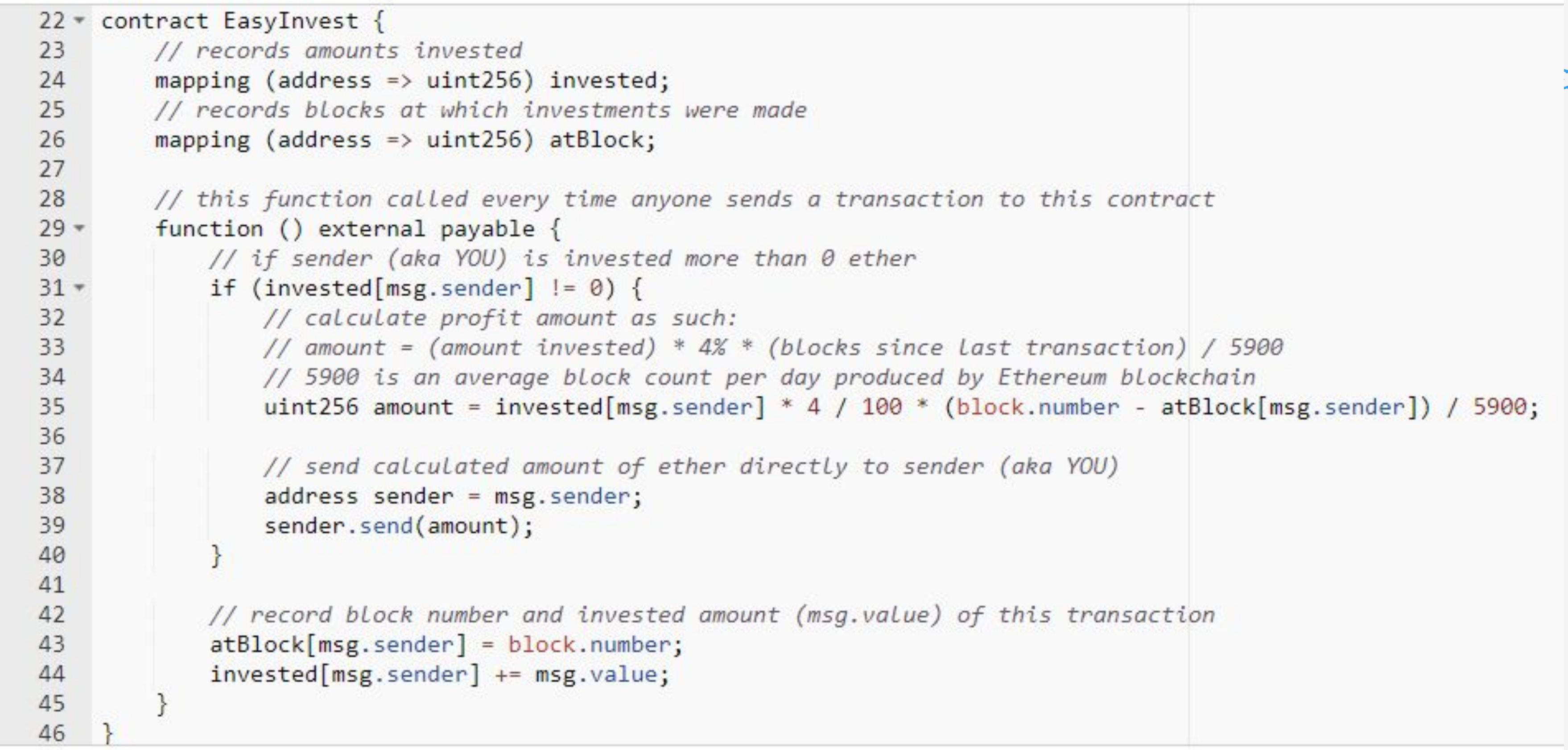
- ☐ Write a function to concatenate two strings
  - ↳ You can import a module
  - ↳ Example: Given “abc” and “def”, the function should return “abcdef”

- ☐ Bonus: Try it without importing a module!

Conclusion

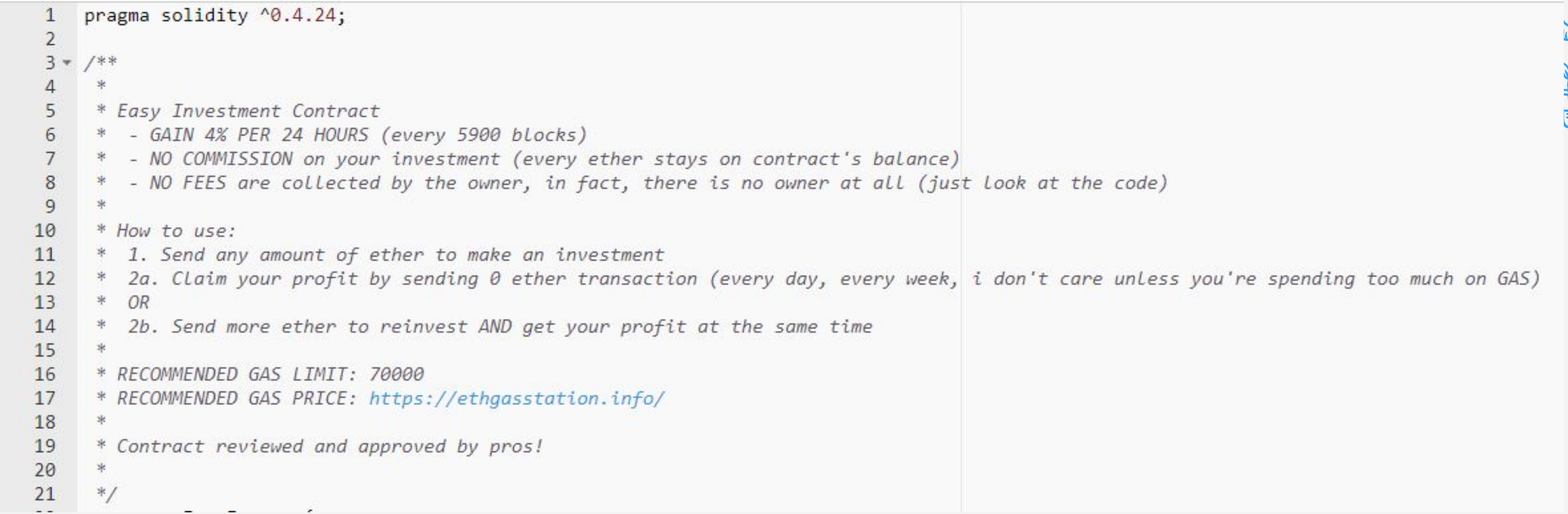
- ☐ Smart Contracts
- ☐ Solidity
  - ↳ Integers
  - ↳ Bytes
  - ↳ Addresses
  - ↳ Functions
  - ↳ Arrays
  - ↳ Mappings
- ☐ Exercises
  - ↳ Greeting, Fibonacci, XOR, Concatenate

Bonus

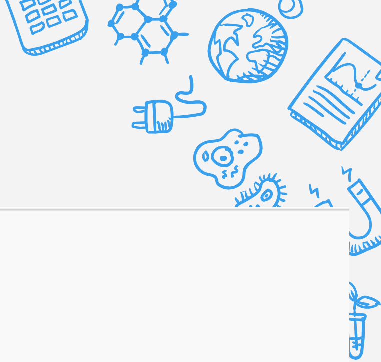


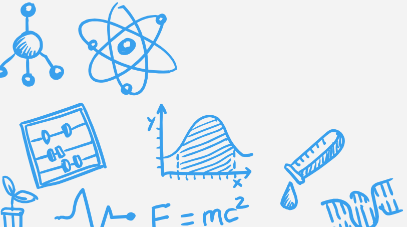
```
contract EasyInvest {
    // records amounts invested
    mapping (address => uint256) invested;
    // records blocks at which investments were made
    mapping (address => uint256) atBlock;

    // this function called every time anyone sends a transaction to this contract
    function () external payable {
        // if sender (aka YOU) is invested more than 0 ether
        if (invested[msg.sender] != 0) {
            // calculate profit amount as such:
            // amount = (amount invested) * 4% * (blocks since last transaction) / 5900
            // 5900 is an average block count per day produced by Ethereum blockchain
            uint256 amount = invested[msg.sender] * 4 / 100 * (block.number - atBlock[msg.sender]) / 5900;

            // send calculated amount of ether directly to sender (aka YOU)
            address sender = msg.sender;
            sender.send(amount);
        }

        // record block number and invested amount (msg.value) of this transaction
        atBlock[msg.sender] = block.number;
        invested[msg.sender] += msg.value;
    }
}
```

Bonus



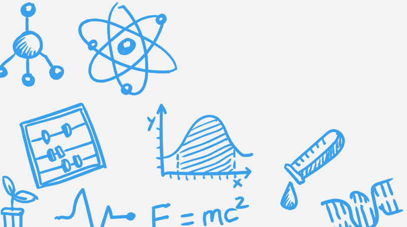
```
pragma solidity ^0.4.24;

/**
 *
 * Easy Investment Contract
 *  - GAIN 4% PER 24 HOURS (every 5900 blocks)
 *  - NO COMMISSION on your investment (every ether stays on contract's balance)
 *  - NO FEES are collected by the owner, in fact, there is no owner at all (just look at the code)
 *
 * How to use:
 *  1. Send any amount of ether to make an investment
 *  2a. Claim your profit by sending 0 ether transaction (every day, every week, i don't care unless you're spending too much on GAS)
 *  OR
 *  2b. Send more ether to reinvest AND get your profit at the same time
 *
 * RECOMMENDED GAS LIMIT: 70000
 * RECOMMENDED GAS PRICE: https://ethgasstation.info/
 *
 * Contract reviewed and approved by pros!
 *
 */
```





Assignment

- ☐ Reviewing:
  - ↳ Understand basic Solidity syntax
- ☐ Explore:
  - ↳ ReadTheDocs - Solidity
  - ↳ LearnXInYMinutes - Solidity
  - ↳ Diagrammatic Interpretation of Ethereum
- ☐ Homework:
  - ↳ Upload your Solidity code for the Practice Tasks in the feedback form!



BLOCKCHAIN
AT NTU

Thank you!

🔨 with 💙 by

👷 Phang Jun Yu

💬 https://t.me/ntublockchain

Quiz

Feedback