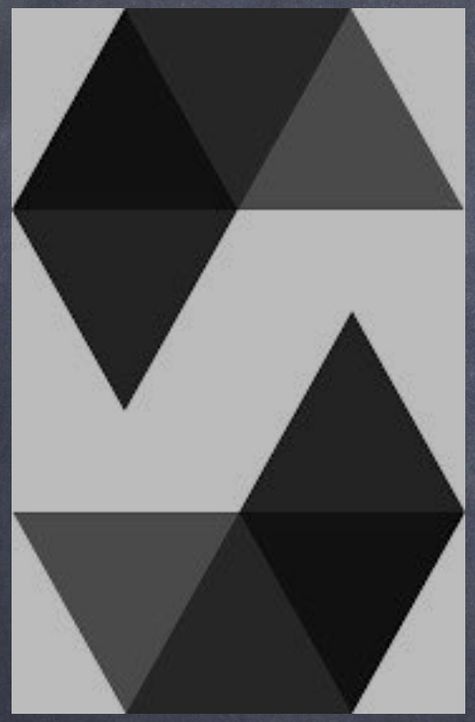
Solidity 101

Secureum Bootcamp

#1

Solidity



High-level Language
Ethereum

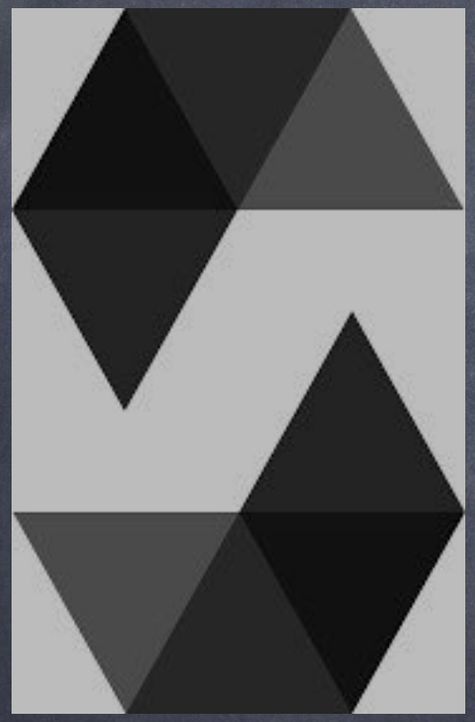
Smart Contracts
Targets EVM

2014: Gavin Wood
Christian Reitwiessner, Alex
Beregszaszi & Others

Fundamental Pillar
Few Alternatives

#2

Influence



Main: C++
Others: Python, JavaScript

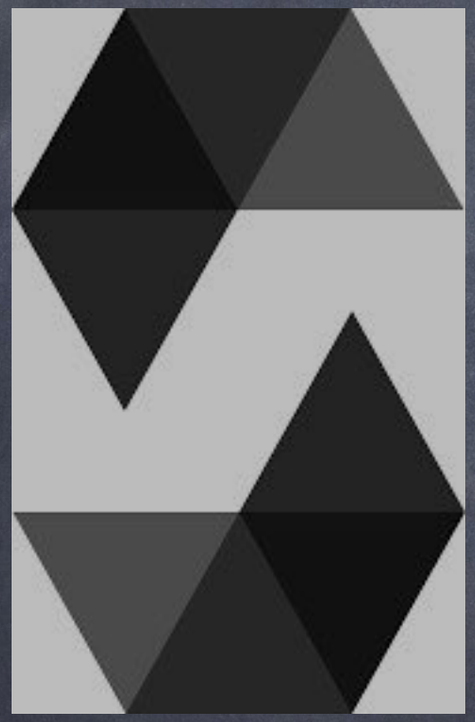
C++: Syntax, OOP etc.
Python: Modifers, super etc.

JavaScript: Scoping, var
Early, Not Anymore

Fundamental Pillar
Few Alternatives

#3

Features



Language Characteristics

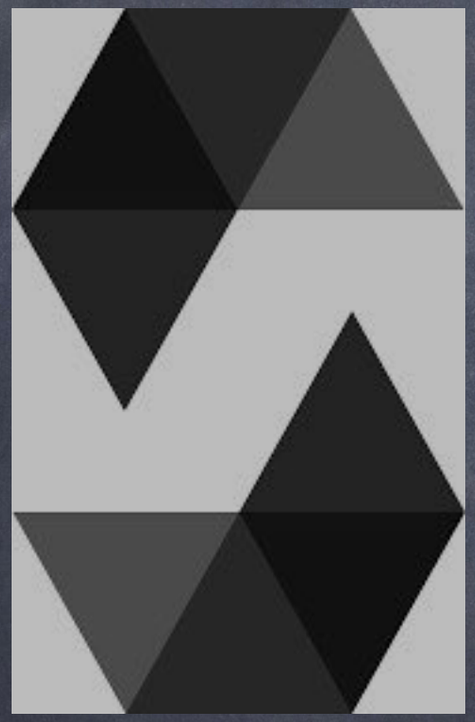
Curly-bracket Language
Object-oriented Language
Statically-typed

Inheritance (OOP), Libraries,
User-Defined Types

Fully-featured
High-level Language

#4

Layout



Solidity Source File Layout

Pragmas/Import Directives, Struct/Enum, Contract Definitions

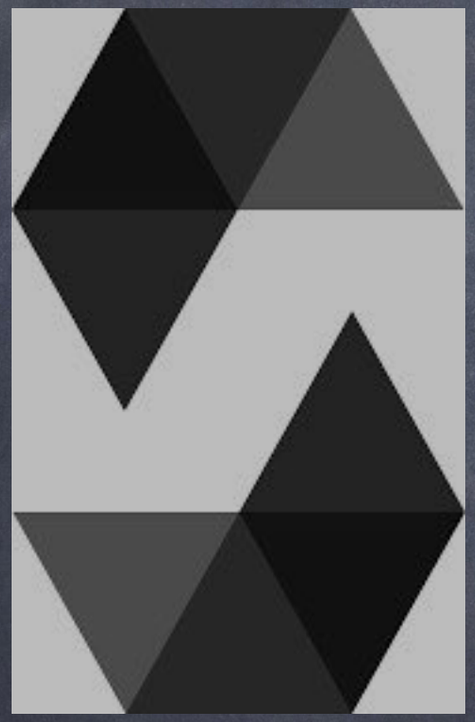
Contract: Struct/Enum, State Variables, Events, Errors, Modifiers, Constructor, Functions

Arbitrary Number Layout Best-practices

#5

SPDX

SPDX License Identifier
Software Package Data Exchange



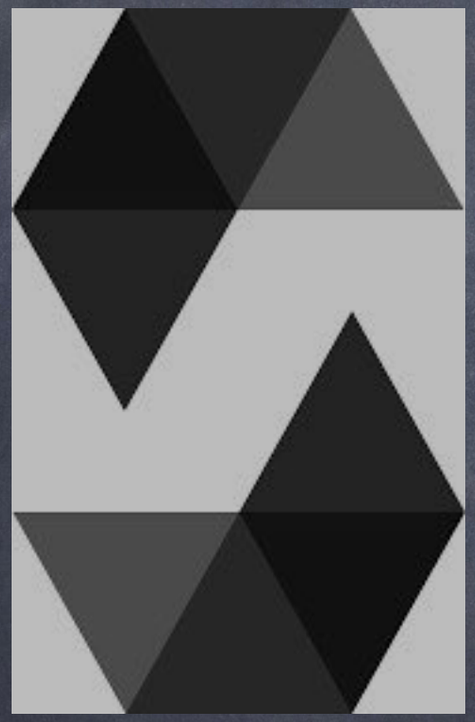
Comment -> License
File Beginning

// SPDX-License-Identifier:
AGPLv3

Compiler -> Bytecode
Metadata

#6

Pragma



Pragma Keyword

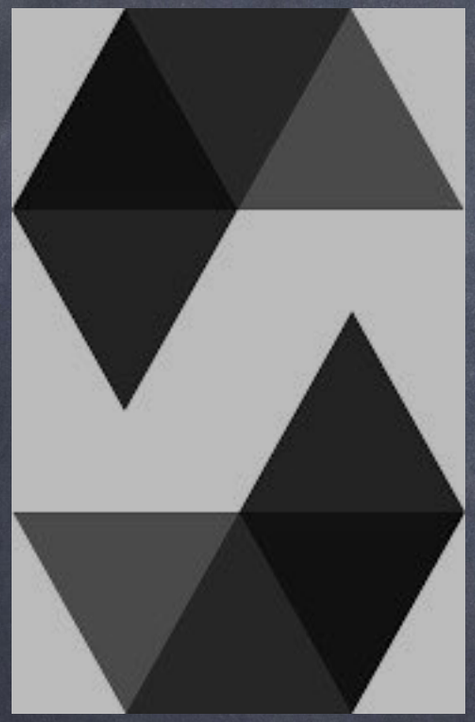
Compiler Features/Checks
pragma solidity ^0.8.0;

Version, ABI Coder v1/v2,
Experimental: SMTChecker

Every File, Not Imported

#7

Version



Version Pragma

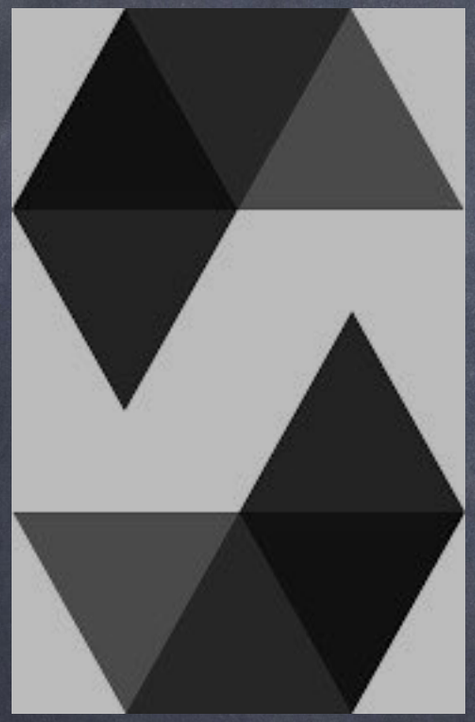
Solidity Compiler Version
pragma solidity x.y.z;

Used Vs Specified
Simple, Complex, Floating

Affects: Features,
Optimizations, Security

#8

ABI Coder



ABI Coder Pragma
Versions: v1 or v2

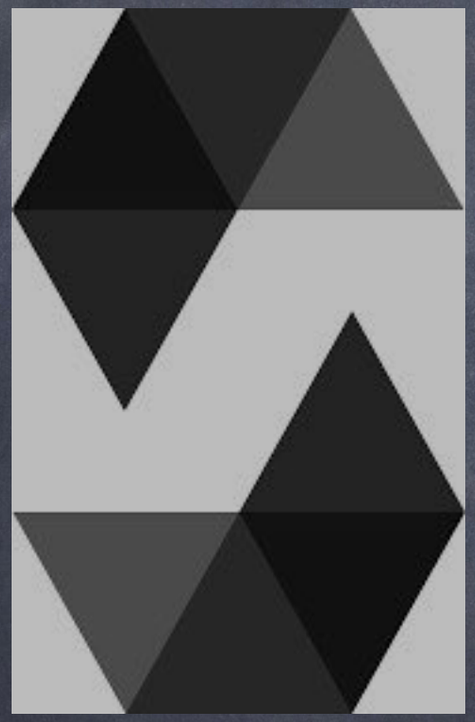
v2: Encode/Decode Nested Arrays/Structs

v2 Superset v1
File of Definition/Use

Affects: Encoding, Optimizations, Security

#9

Experimental



ABI Experimental
SMTChecker

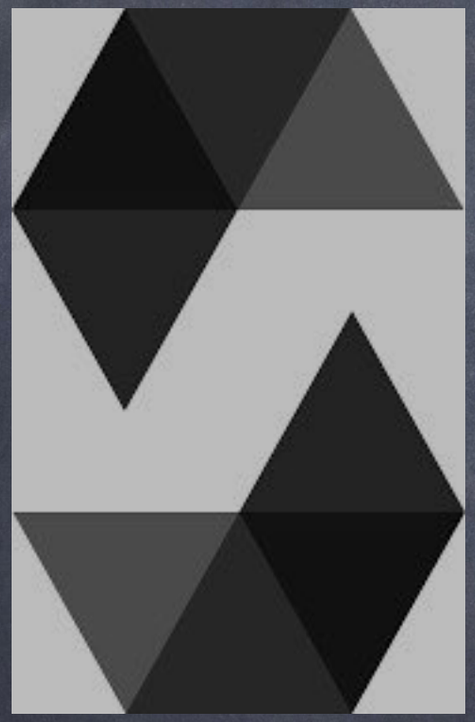
Not Enabled By Default
Safety Checks -> SMT Solver

Checks: require/assert,
Overflow/Underflow, Div/0,
Unreachable Code etc.

Affects: Security &
Optimizations

#10

Imports



Import Statements

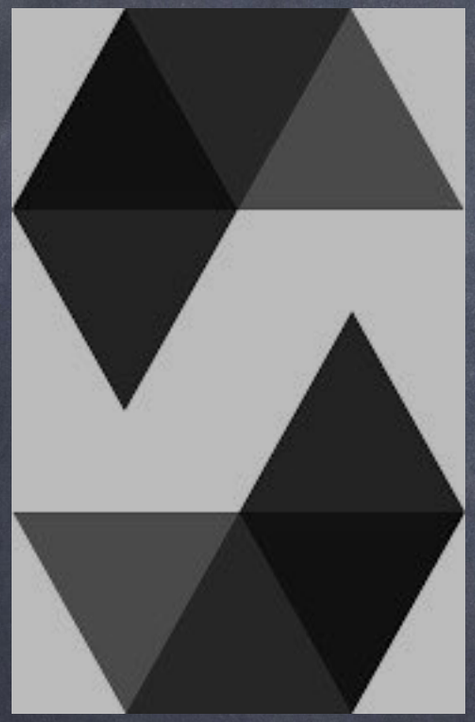
Similar to JavaScript
import filename;

Code Modularity
Code Reuse

Affects: Readability,
Security & Optimizations

#11

Comments



Inline Comments

Single Line: //
Multi-line: /* .. */

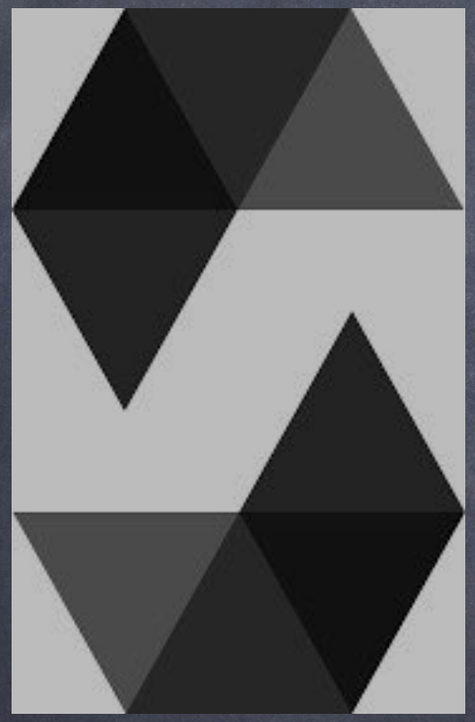
Implement -> Document
Assumptions/Invariants

Readability,
Maintainability

#12

NatSpec

Ethereum Natural Language Specification Format



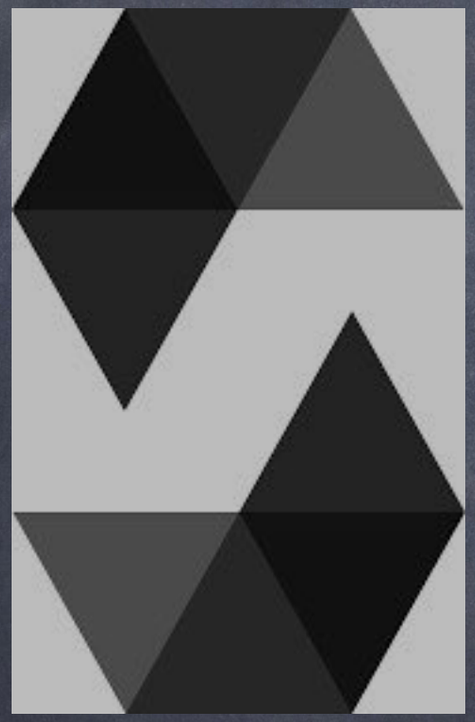
/// or /** .. */
Fn Declaration or Statements

@title, @author, @notice, @dev, @param, @return etc.

JSON Doc for Devs & Users

#13

Contracts



Contracts: Similar to Classes in OOP

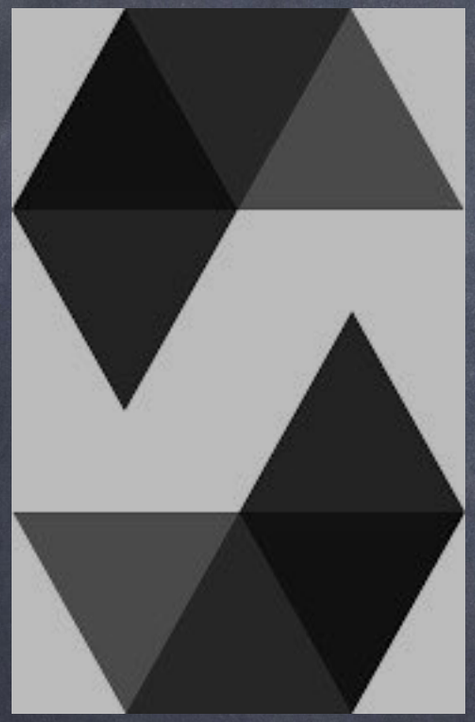
State -> Variables
Modification -> Functions

Other Contracts
Inherit & Interact

Smart Contracts
Fundamental Primitive

#14

Contracts



Contracts: Components

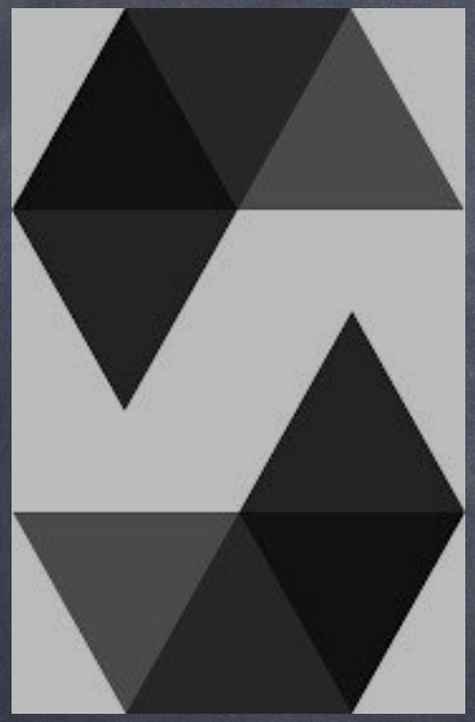
Struct/Enum, State Variables, Events, Errors, Modifiers, Constructor, Functions

Contracts, Libraries, Interfaces

Smart Contracts
Fundamental Primitive

#15

Variables



State Variables

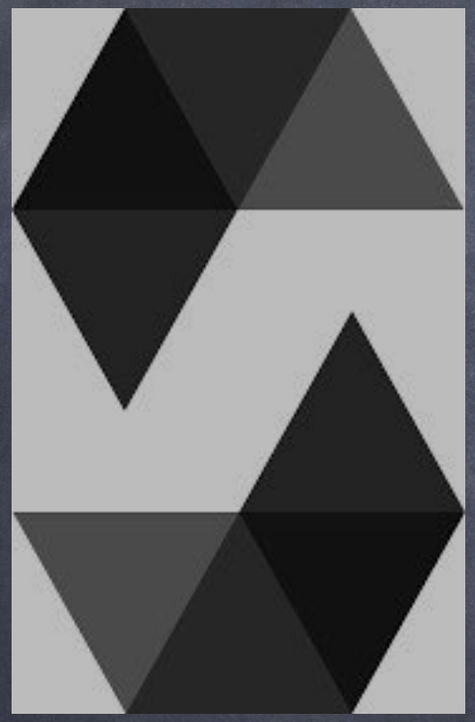
All Contract Functions

Data Location -> Contract Storage

State Variables
Contract State

#16

State Visibility



State Visibility Specifiers

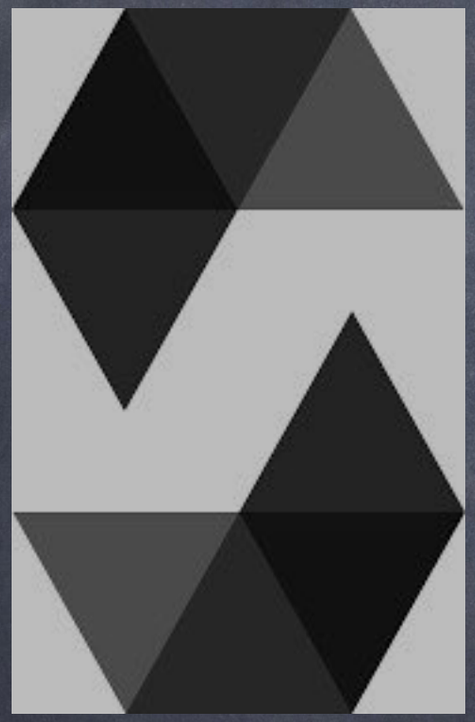
public: Interface, Internally/ Externally, Getter

internal: Current/Derived
private: Current

Contract Interface/Interactions

#17

State Mutability



State Mutability Specifiers

constant: Fixed Compile-time

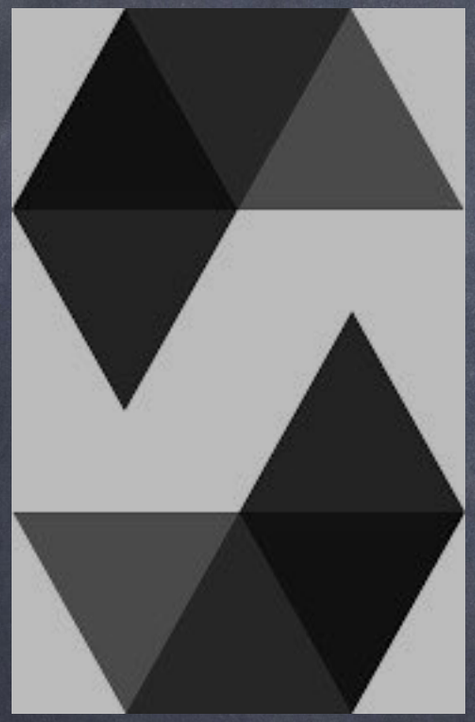
immutable: Fixed Construction-time

No Storage Slot
Storage/Gas Efficient

#18

Mutability & Gas

State Mutability
Gas Costs



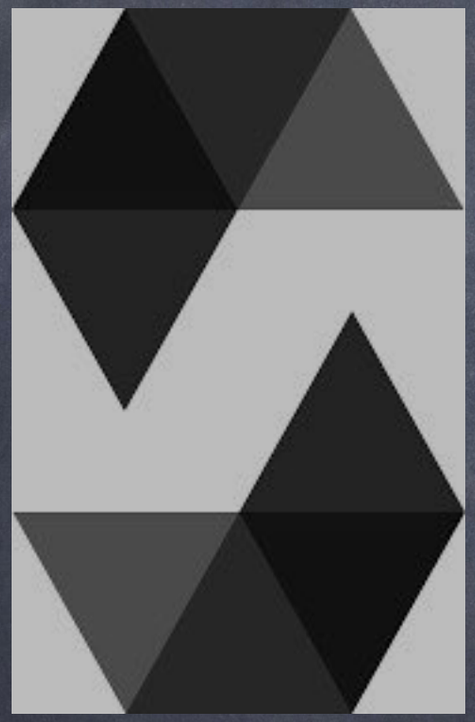
constant: Copied @Usage
Local Optimizations

immutable: Evaluated Once
& Copied

constant Vs immutable
Storage/Gas Efficient

#19

Functions



Contract Functions

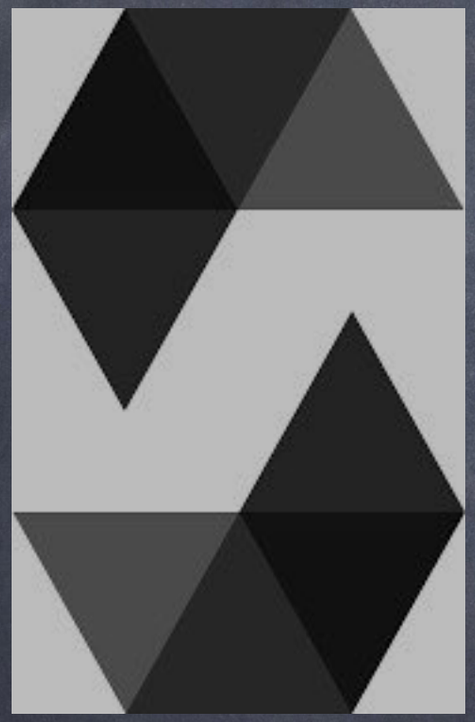
Executable Units of Code

Inside Contracts
File-level -> Free

Functions -> State
Modifications/Transitions

#20

Parameters



Function Parameters

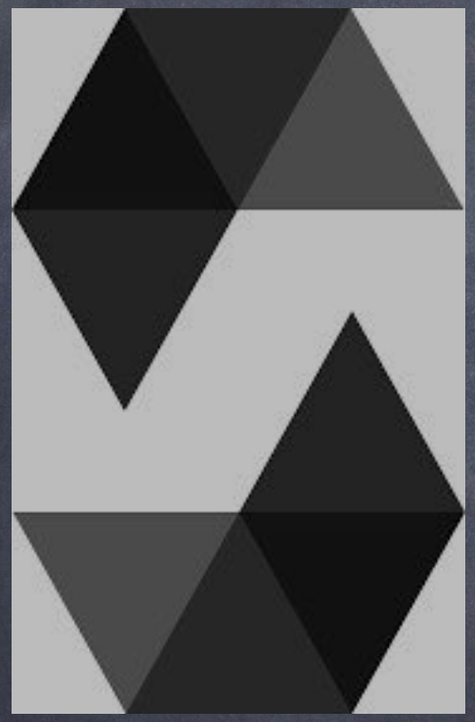
Declared: As Variables

Used/Assigned: Local Variables

Caller -> Arguments
Callee -> Parameters

#21

Return Variables



After returns Keyword

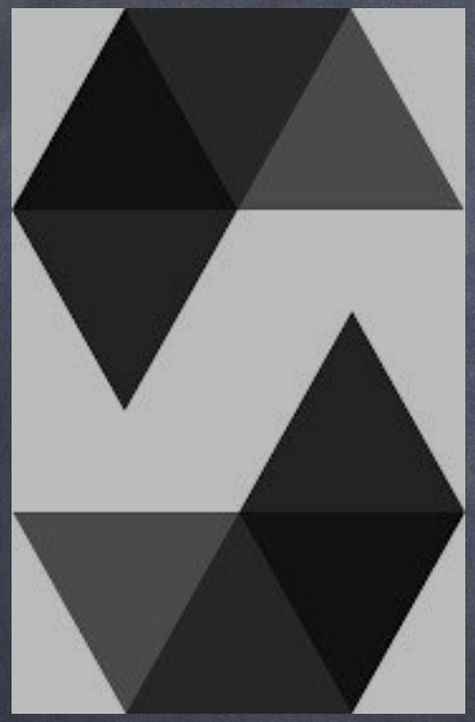
Single or Multiple
Named or Unnamed

Like Local Variables
return Statement

Caller -> Callee
Parameters -> Return Value

#22

Modifiers



Function Modifiers
modifier Keyword

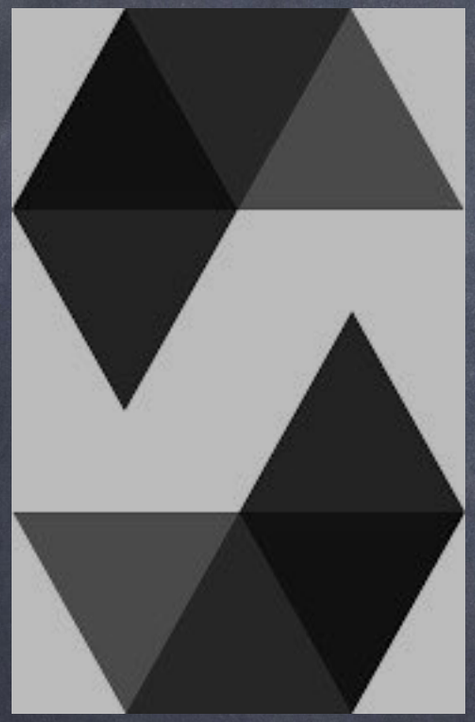
```
modifier mod() {Checks; _;}
```

```
function foo() mod {}
mod -> foo()
```

Pre-conditions: Access Control
Post-conditions: Accounting

#23

Function Visibility



Function Visibility
public/external/internal/private

public/external: Interface
external: Not Internally

internal: Base/Derived
private: Contract

Visibility: Access Control
Security Implications

#24

Function Mutability

View or Pure

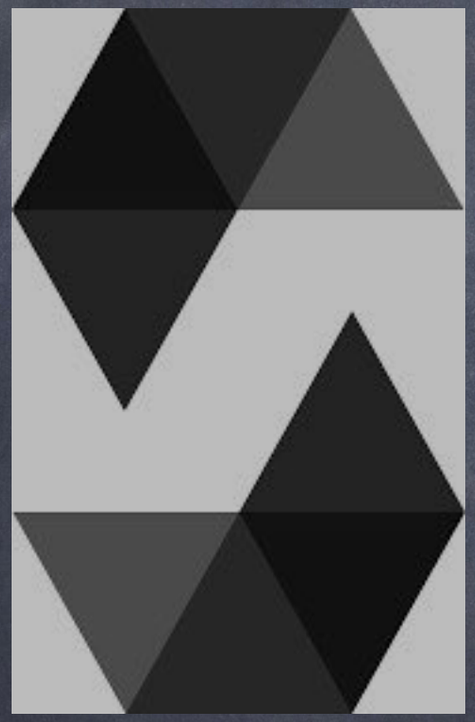
View: Read State
Not Modify -> STATICCALL

Pure: Not Modify
Not Read -> !EVM

Mutability: Write/Read
Security Implications

#25

Function Overloading



Multiple Functions
Same Name, Diff Parameters

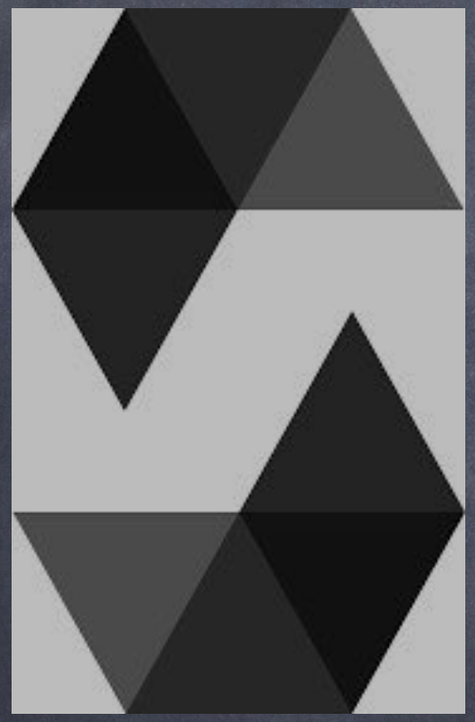
Function Arguments
Match Declarations in Scope

Return Variables
Not Considered

Overloading -> OOP

#26

Free Functions



Function Defined Outside Contracts

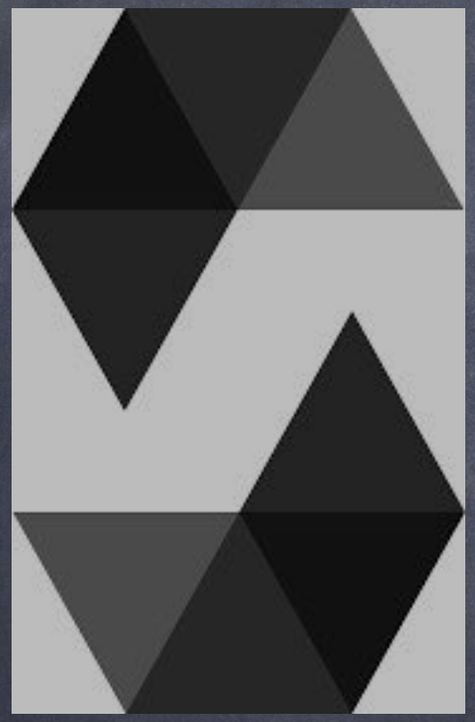
Contract Functions Vs Free Functions

Code Included in All Calling Contracts

Similar -> Internal Library functions -> Rarely Seen

#27

Events



EVM Logging Abstraction

Event -> Emit
Contract Log -> Blockchain

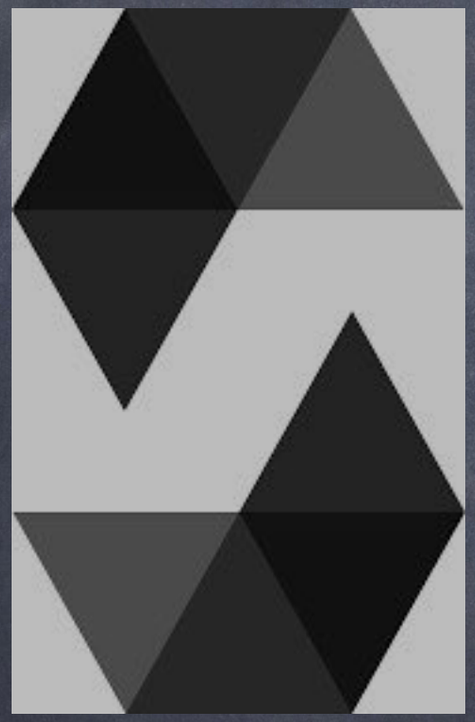
Contracts -> No Access
Off-chain -> RPC Access

Auditing & Logging
Security Monitoring

#28

Indexed Parameters

Event Parameters
Indexed or Not



Indexed -> Topics
Not Indexed -> Data

Topics -> Search/Filter
Up to 3 Indexed

Indexed -> ERC20 Spec
Gas -> Fast Search

#29

Emit

Events
Declaration -> Emit

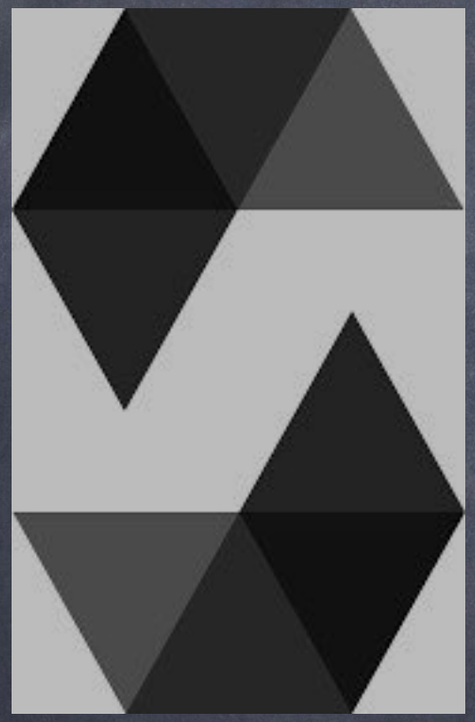
Emit -> Event Log

emit Deposit(msg.sender,
_id, msg.value);

Emit Correct Event
Emit Correct Parameters

#30

Structs



Custom Data Structure

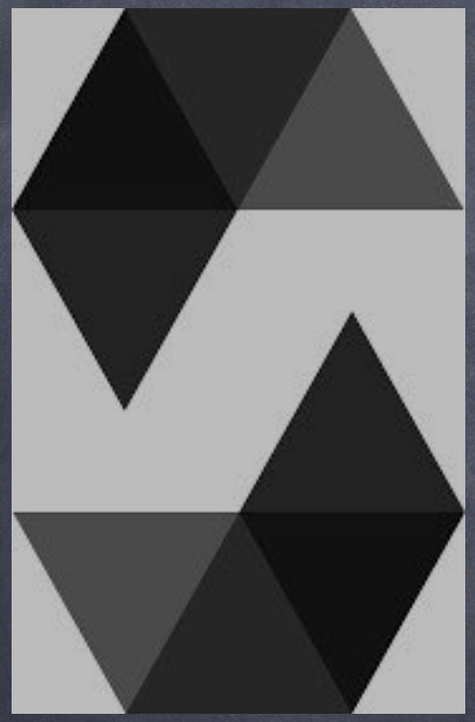
Several Variables
Different Types

Struct Members -> . Access
E.g.: s.user, s.amount

Aggregate Type
Commonly Used

#31

Enums



User-defined Type

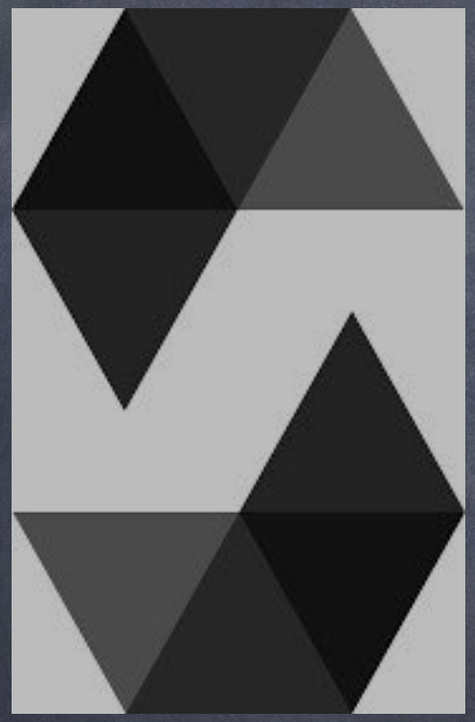
Finite Set -> Constant Values
Min: 1 Max: 256

```
enum ActionChoices { GoLeft, GoRight}
ActionChoices choice;
choice = ActionChoices.GoRight;
```

Improves Readability

32

Constructor



Contract Creation
Constructor Function

Optional & Only One
constructor Keyword

Initialize State
Contract Code -> Blockchain

Initializations Vs Defaults
Affects Security

#33

Receive Function

Ether Transfers: send/transfer
Empty Calldata

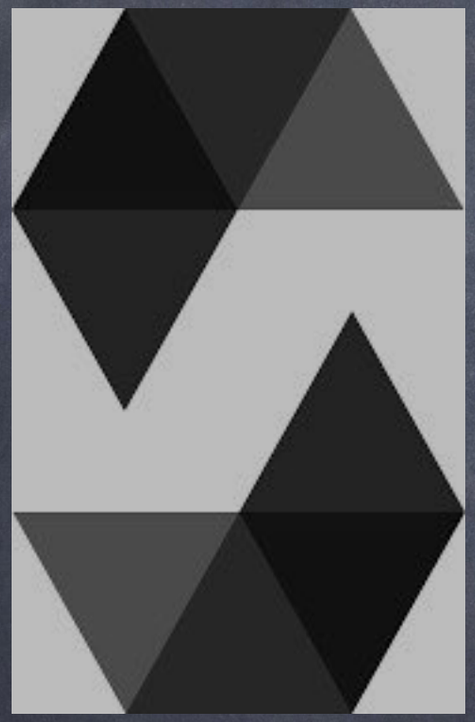
One, No Parameters/
Return, External, Payable

send/transfer -> 2300 Gas

Security: ETH Balance ->
Assumptions

#34

Fallback Function



No Match, No Data & No receive() -> fallback()

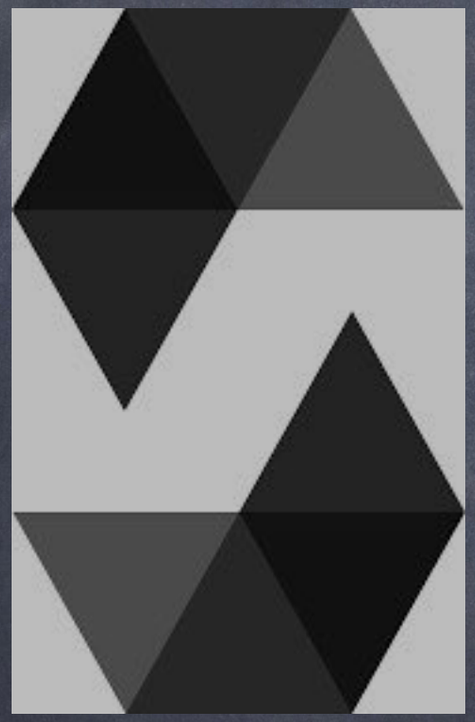
One, Calldata/Return, External, Payable for ETH

send/transfer -> 2300 Gas

Security: ETH Balance -> Assumptions

#35

Statically Typed



Variable Type -> Specified Compile-time

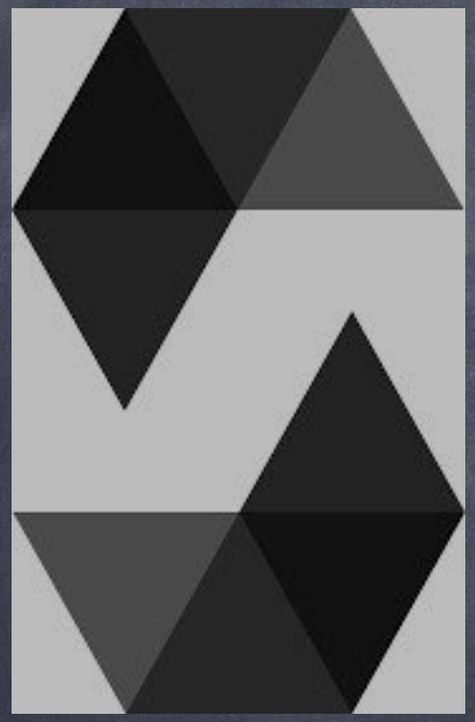
Vs Dynamically-typed Types -> Runtime Values

Compile-time -> Type Checking

Other Examples: C, C++, Java, Rust, Go, Scala

#36

Types



Value & Reference

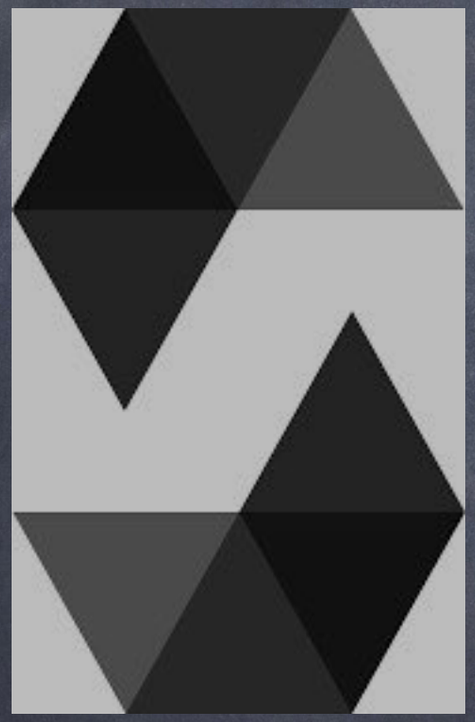
Value: Passed by Value
Copied: Arguments/Assignments

Reference: Passed by Reference
Multiple Names -> Same Variable

Security: State Updates/
Transitions

37

Value Type



Types: Value & Reference

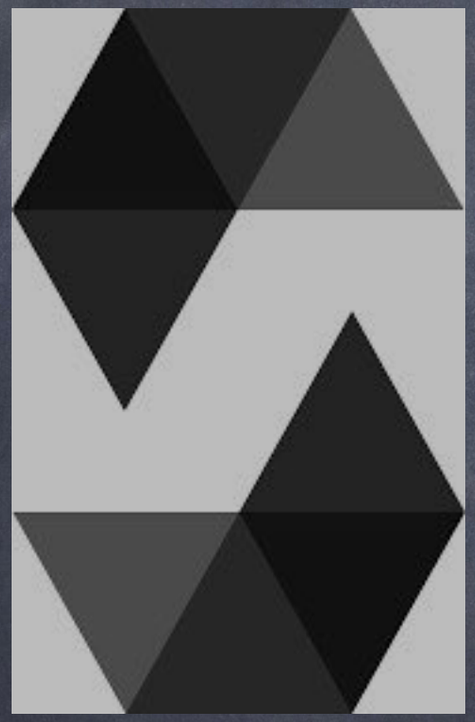
Value: Passed by Value
Copied: Arguments/Assignments

E.g.: Boolean, Integer,
Address, Enums etc.

Security: Copy -> Safer
Check -> Persistence

#38

Reference Type



Types: Value & Reference

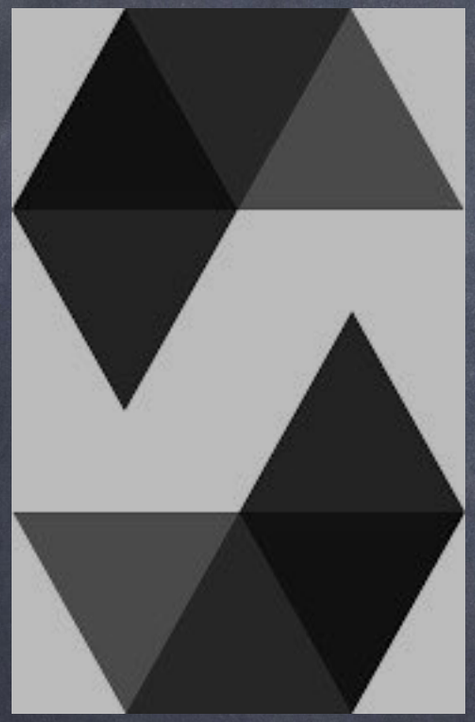
Passed by Reference
Multiple Names -> Same Variable

Arrays, Structs, Mappings

Security: Risky
Unintentional Modification

#39

Default Values



Variable: Declared, Not Initialized

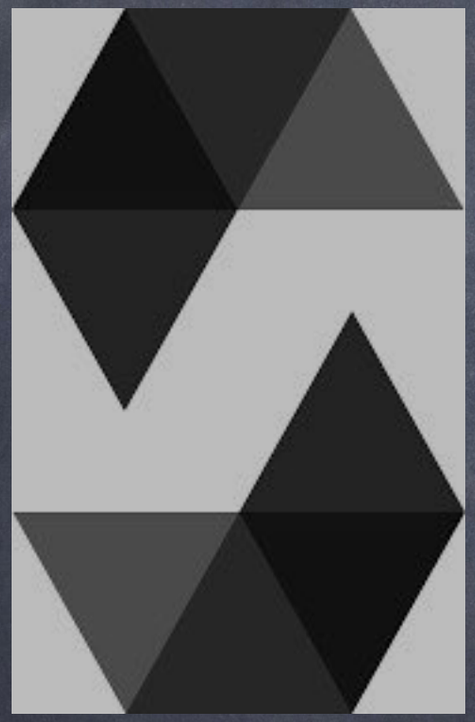
Default: Zero-state of Type

Boolean: False, Integer: 0, Address: 0, Enum: First Member

Security: Important
E.g. Address: Burn

#40

Scoping



Variable Visibility
C99 Scoping

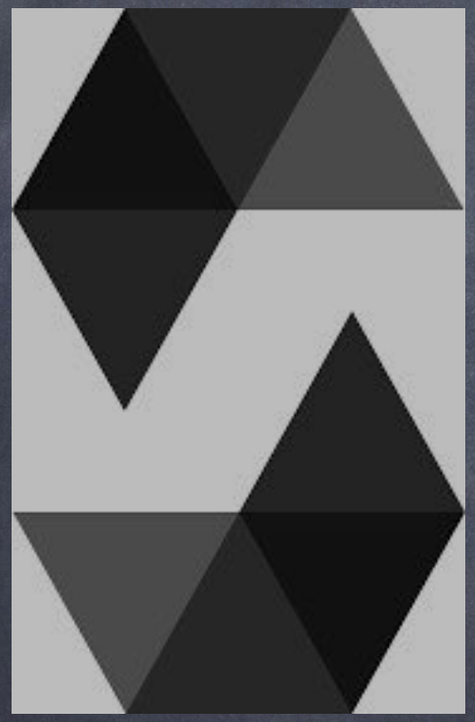
Declaration -> End of {}
Parameter -> Code Block

Functions/Contracts/State Variables: Visible Before Declaration

Security: Data Flow Analysis

#41

Boolean



bool Keyword

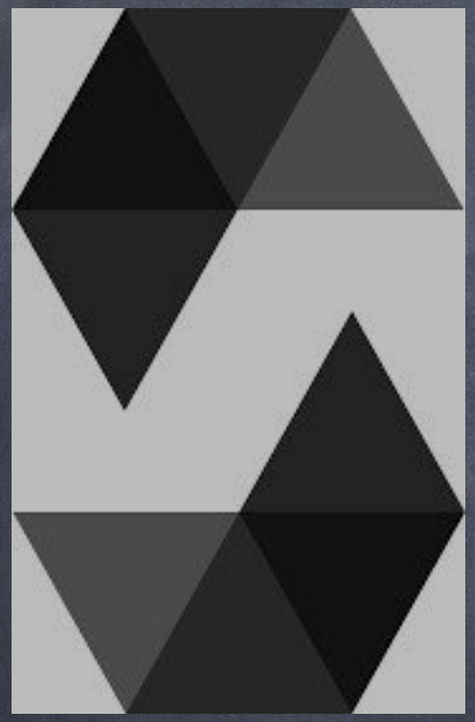
True or False
Operators: !, ==, !=, &&, ||

Short-circuit: ||, &&

Security: Control Flow, Checks, || Vs &&

#42

Integers



uint/int Keywords
Sizes: 8 -> 256 Bits

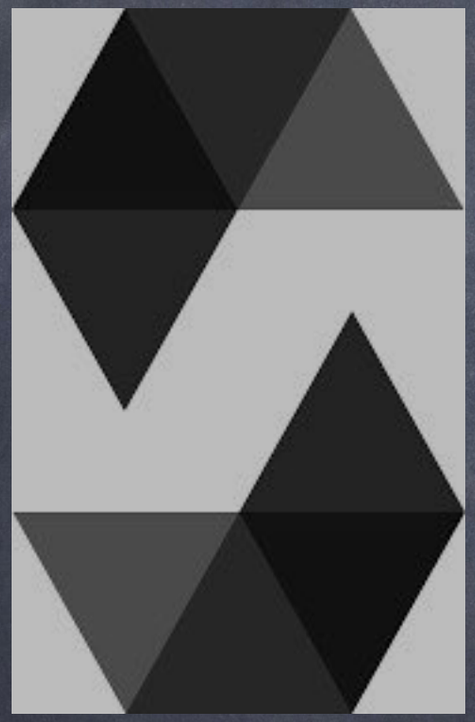
uint8, uint16...uint256
int8, int16...int256

Operators: Arithmetic/
Comparative/Bit/Shift

Security: Data Flow,
Underflow/Overflows

#43

Integer Arithmetic



E.g.: uint256
Range: 0 -> 2**256-1

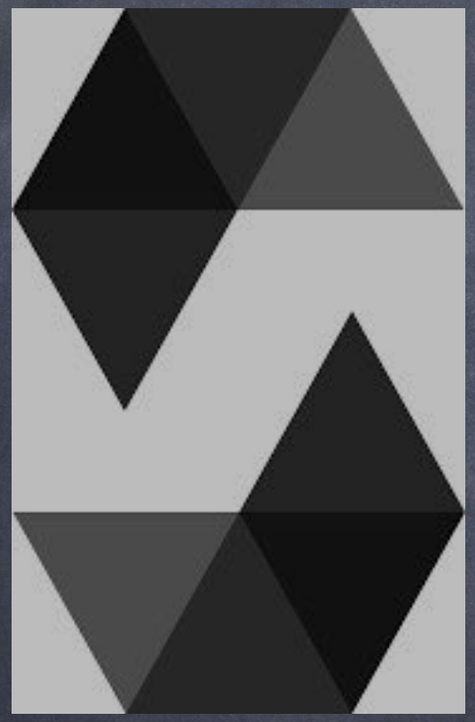
Underflow/Overflows -> Wrapping

Solidity < 0.8.0
SafeMath

Solidity >= 0.8.0
Checked Vs Unchecked

#44

Fixed Point



Fixed Vs Floating

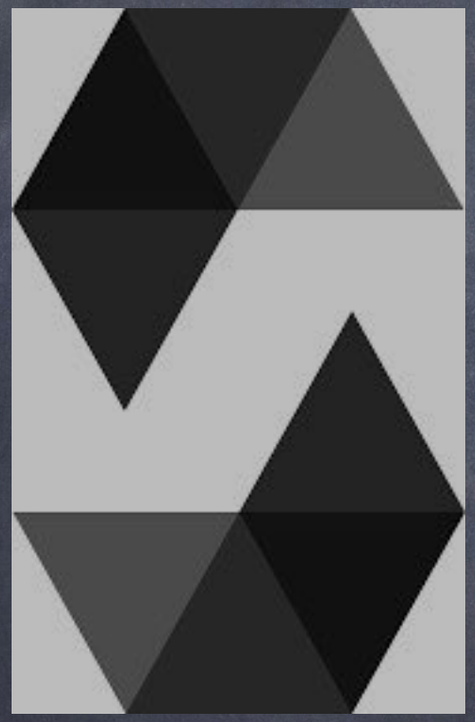
Keyword: fixed & ufixed

Solidity: No Real Support

Libraries: DSMath, PRBMath, ABDKMath64x64 etc.

#45

Address



Account Address: EOA/ Contract

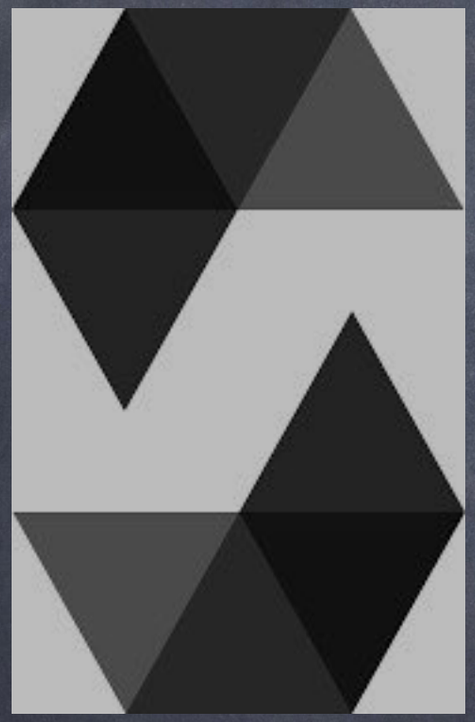
Keyword: account (payable)
20 Bytes

Operators: ==, !=, <, <=, >, >=
Implicit/Explicit Conversions

Security: Critical
Access Control, Balances etc.

#46

Address Members



balance, code, codehash

transfer & send
2300 Gas Stipend

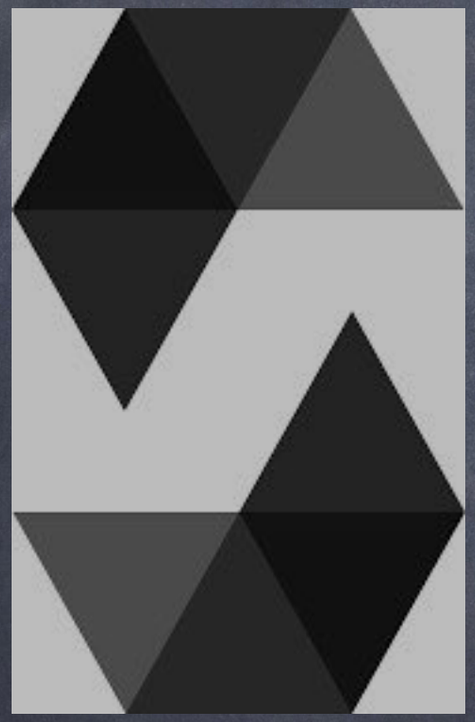
call, delegatecall, staticcall

Security: Balances, EOA/Contract, ETH Tx, External Calls

#47

transfer

Ether Transfer



transfer -> receive/
fallback

2300 Gas Subsidy
Failure -> Revert

Security: Reentrancy
Mitigation, Gas Assumption

#48

send

Ether Transfer

send -> receive/fallback

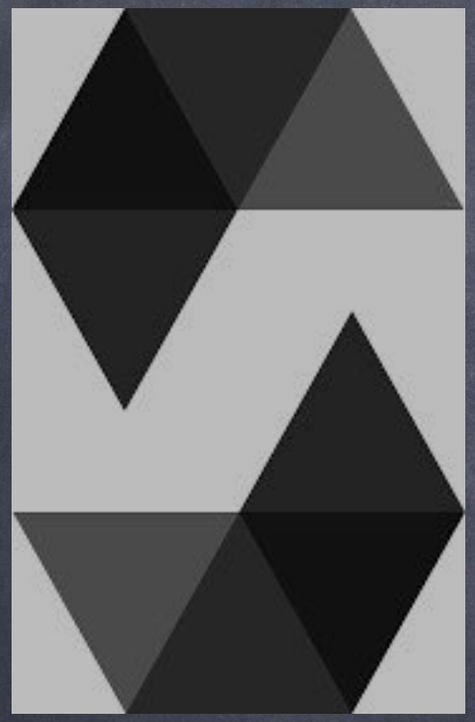
2300 Gas Subsidy
Failure -> No Revert

Security: Reentrancy Mitigation, Gas Assumption, Return Value Check

#49

External Calls

call, delegatecall, staticcall



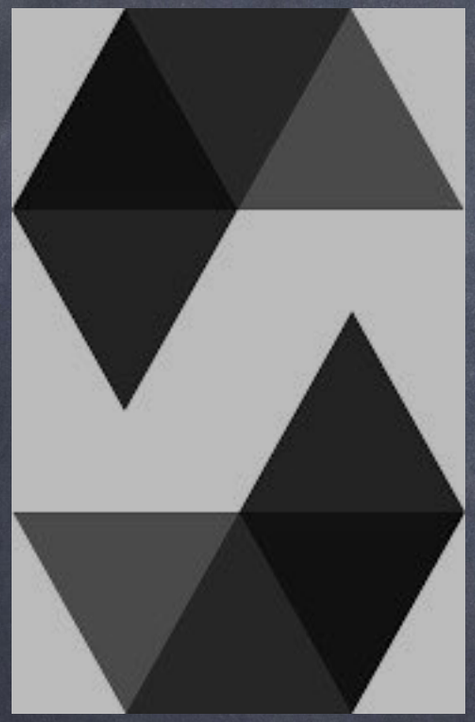
Parameter: bytes memory
Return: Success, bytes memory
abi.* Functions, gas/value

delegatecall -> State/Modify
staticcall -> State/Use

Security: Low-level Call, Untrusted Contract, Reentrancy, Return Value Check

#50

Contract Type



Every Contract -> Own Type

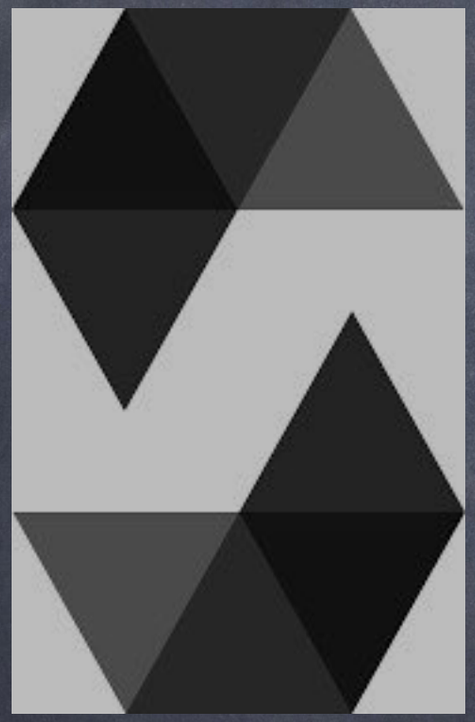
Contract <-> Address

No Operators

Members: External Functions & Public State Variables

#51

Bytes Arrays



Fixed-size Byte Arrays

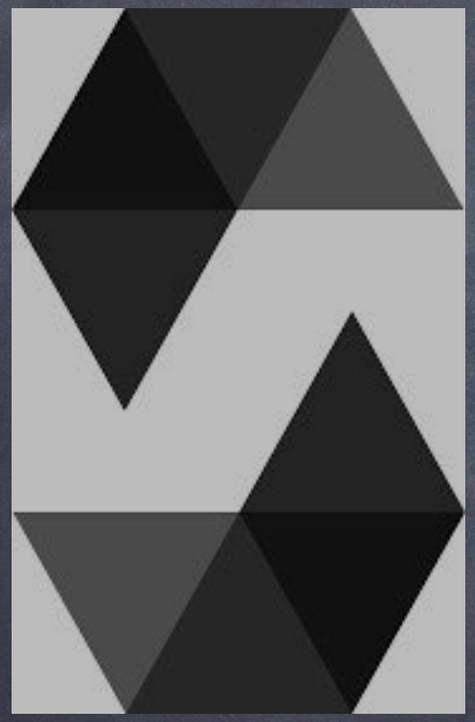
bytes1, bytes2,...bytes32

bytes Vs byte[] -> Wastage

bytes -> Byte Storage
E.g.: Hashes

#52

Literals



Five Types: Address, Rational/ Integer, String, Unicode, Hexadecimals

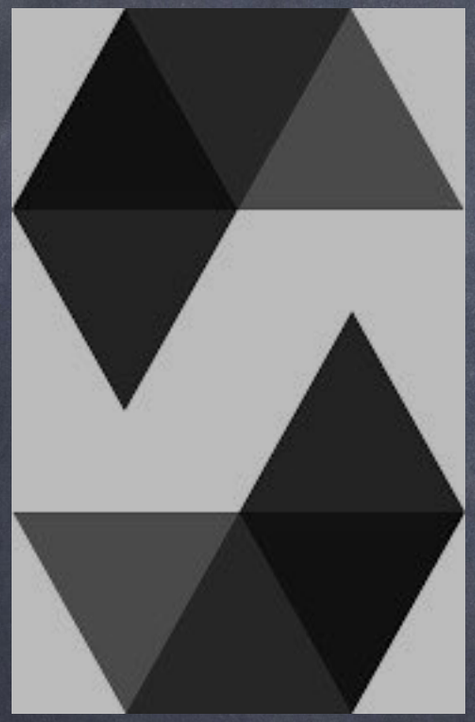
Address: 40 Hex & Checksum
Integer/Decimal: . & _

String: “..”,’..’
unicode & hex”..”/hex’..’

Usage: Constants

#53

Enums



User-defined Type

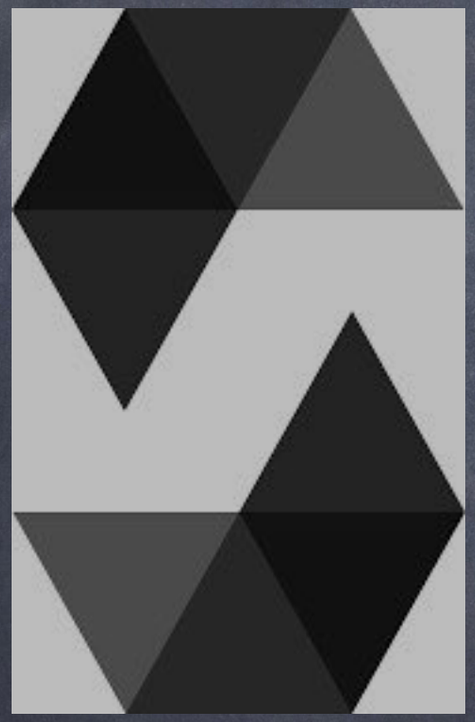
Members: At least 1
Max 256

Default: First Member

Usage: E.g.: State Names/
Transitions

#54

Function Types



Variables -> Function Type

Assigned <- Functions Parameters/Return Value

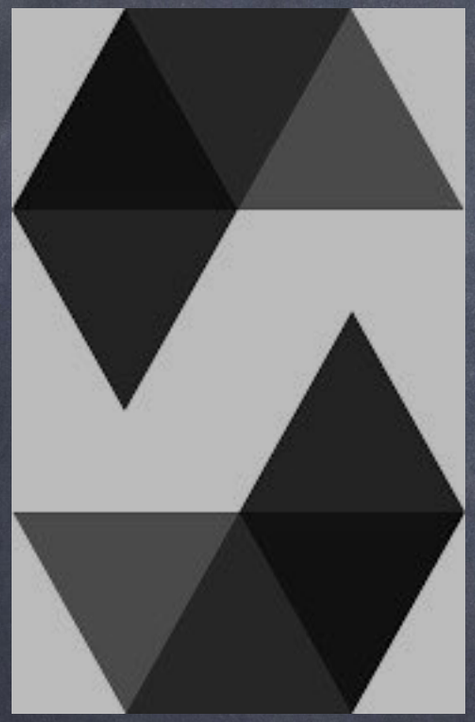
Internal Vs External

Usage: Minimal

#55

Data Location

Reference Types -> Data Location Annotation

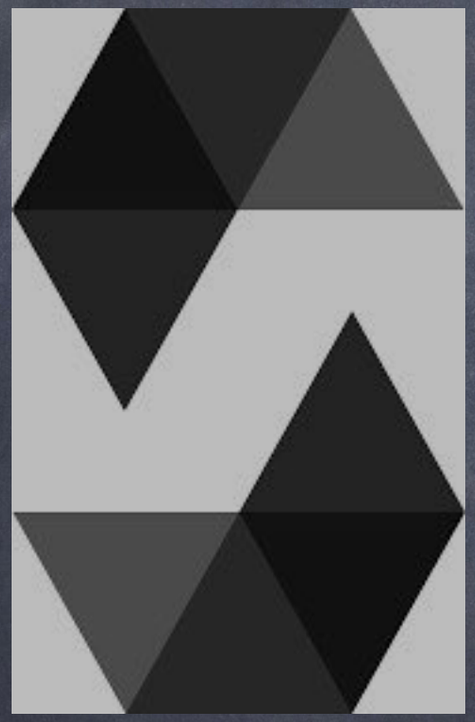


memory, storage, calldata

memory: function, storage: contract
calldata: non-modifiable, non-persistent, external fn parameters

Impact: Scope
Security: Persistence

Data Location & Assignments



Data Location -> Persistency & Assignment Semantics

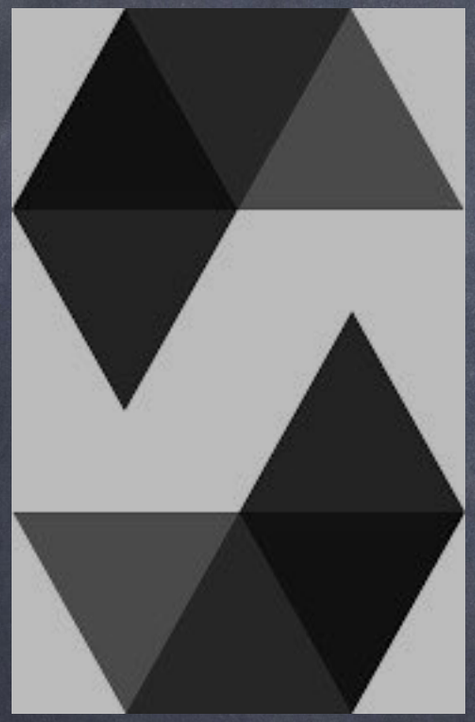
storage-memory -> copy
memory-memory -> reference

storage-storage -> reference
Others -> copy

Impact: Copy/Modification
Security: Reference Risk

#57

Arrays



Arrays: Static or Dynamic

T[k] or T[]
Elements -> Any Type

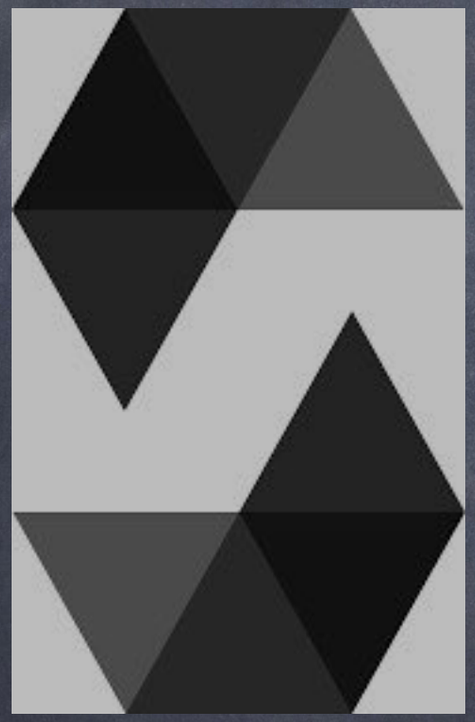
Indices -> Zero-based
Access Past -> Revert

Security: Correct Index, Off-by-one, Gas/DoS

#58

Array Members

length, push, push(x), pop



length -> Number of Elements
push -> Zero Element @ End

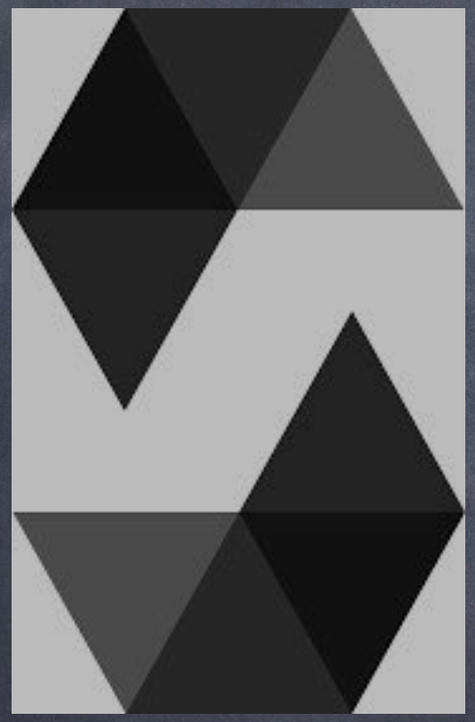
push(x) -> x @ End
pop -> Remove Element @ End

Security: Length & Off-by-one, push/pop Semantics

#59

bytes & string

bytes -> byte Data
Arbitrary Data/Length



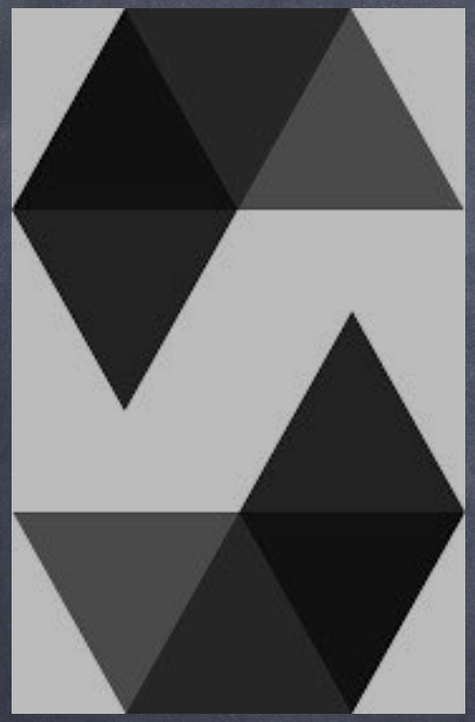
bytes1, bytes2..bytes32
bytes over byte[] -> Cheaper

string == bytes
No length/index Access

Solidity -> No string Fns
External Libraries

#60

Memory Arrays



Memory Arrays
Dynamic -> new Operator

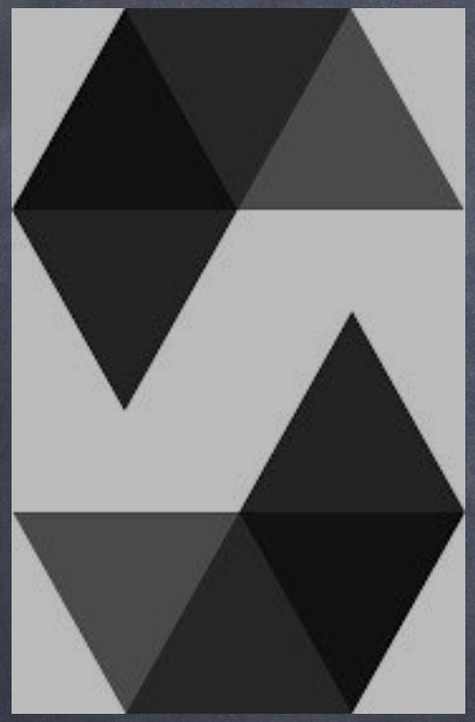
Memory Vs Storage
Cannot Resize -> No push

Calculate Size -> Advance
New Array -> Copy Over

E.g.: uint[] memory a =
new uint[](7);

#61

Array Literals



Comma-separated List
Within [..]

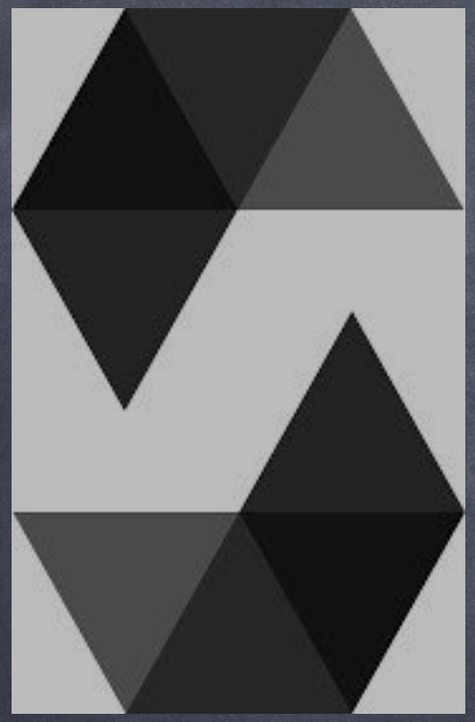
Statically-sized
Length = #Expressions

Type = First Element
Others Convertible

No Assignment to
Dynamically-sized Arrays

#62

Arrays
Gas Costs



push() & pop()

Push -> Constant Cost

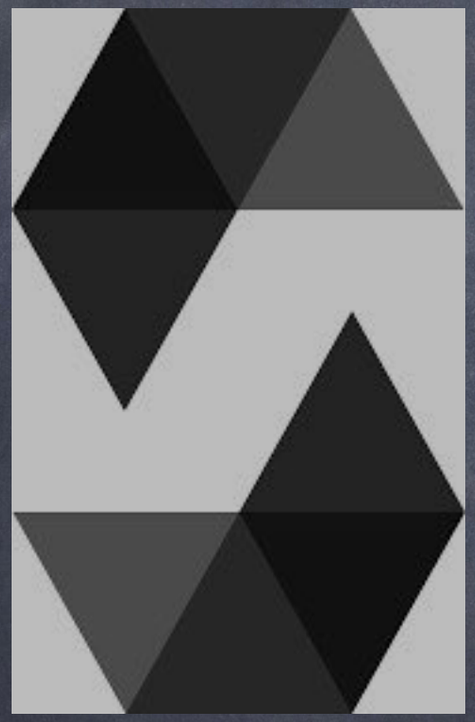
Pop -> Depends on Element Size

Pop Array Element -> Expensive -> Delete Element

#63

Arrays
Slices

View -> Contiguous Array Portion



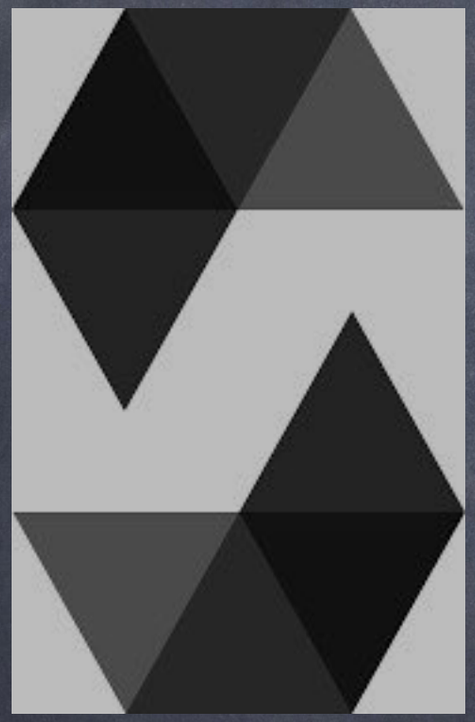
X[start:end] -> X[start] to X[end-1]

start & end -> Optional
start <= end & end < length

Only Calldata Arrays

#64

Struct Types



New Aggregate Type

Combine Value/Ref Types
One Unit

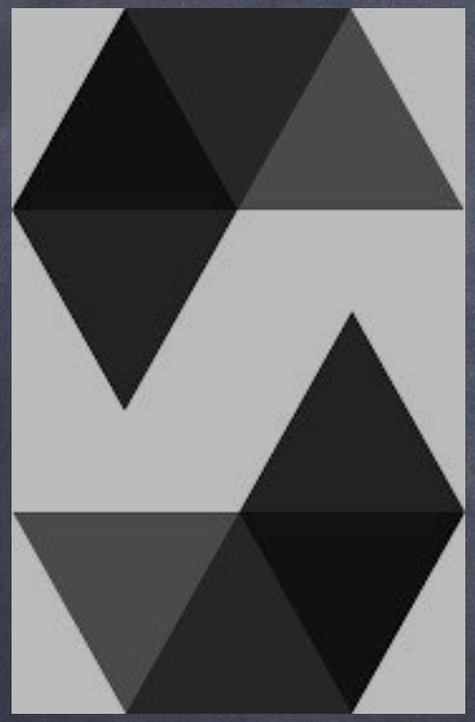
Inside or Contain
Mappings & Arrays

Cannot Contain
Member of Same Type

#65

Mapping Types

Key-Value Pairs
mapping(_Key => _Value) _Var



Key Type -> Restricted
Value Type -> Any

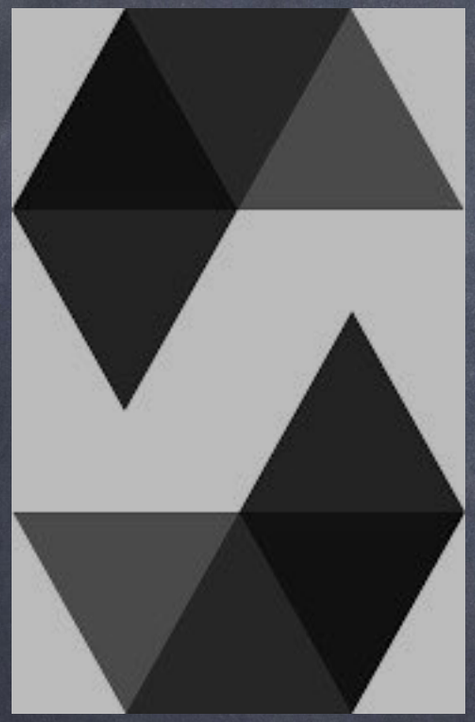
Key Value -> Not Stored
Keccak256 Lookup

No Length/Set, Storage
Only, No Iteration

#66

Shorthand Operators

LValues -> Assigned To



a += e is a = a+e

-=, *=, /=, %=, |=, &=, ^=

a++/a-- are a += 1 & a -= 1

Expr -> Prev Value

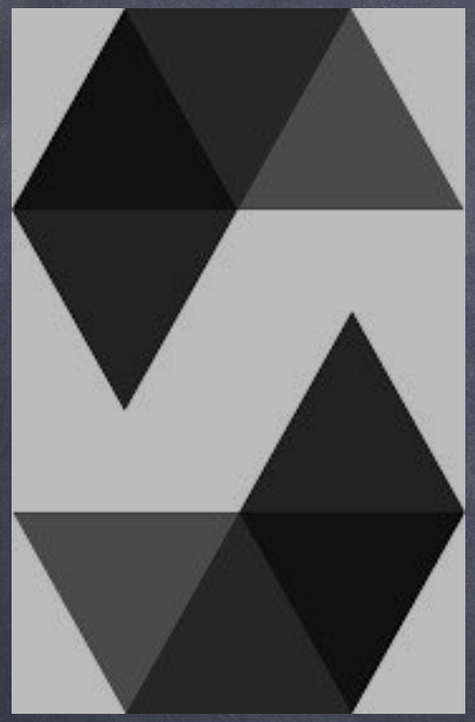
++a/--a are a += 1 & a -= 1

Expr -> Modified Value

#67

delete

delete a -> Assigns Initial Value to a



Ints -> 0, Dynamic Arrays -> Length 0, Static Arrays -> Same Length, Initial Value Elements

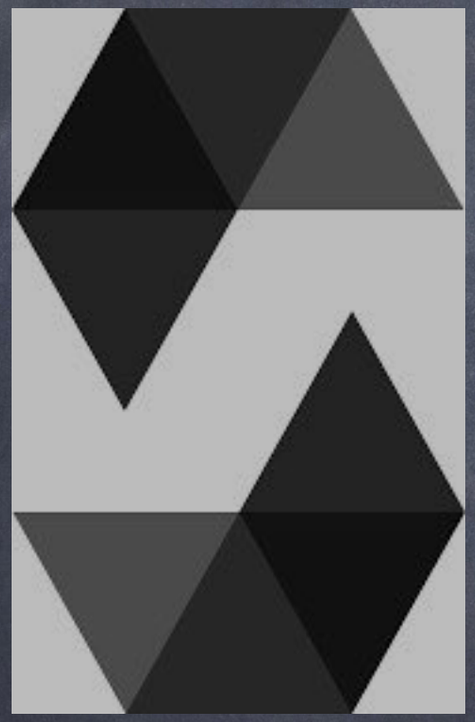
Arr[x] -> Deletes Item@x, Structs -> Members Reset Unless Mappings

Mappings -> No Effect
Specific Key -> Value@Key Deleted

#68

Implicit Conversions

Type Conversion
Compiler Applied



Assignments/Arguments/
Operators

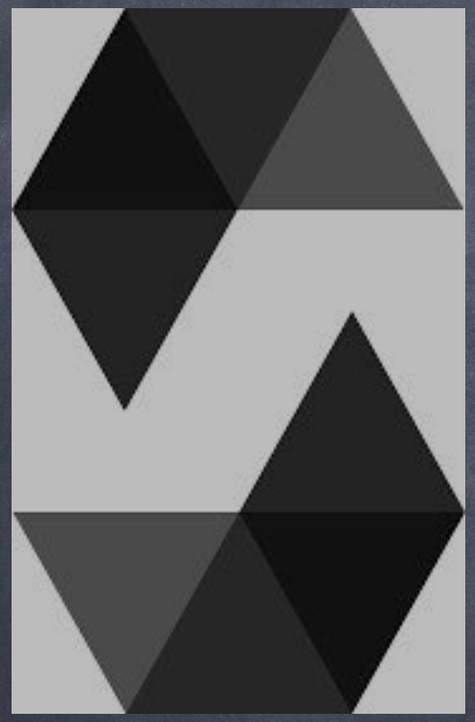
Semantic Sense
No Information Lost

E.g.: uint8 -> uint16, int128 -> int256, int128 X-> uint256, uint256 X-> uint128

#69

Explicit Conversions

Type Conversion
Developer Applied



Unexpected Behavior
Bypass Type Safety

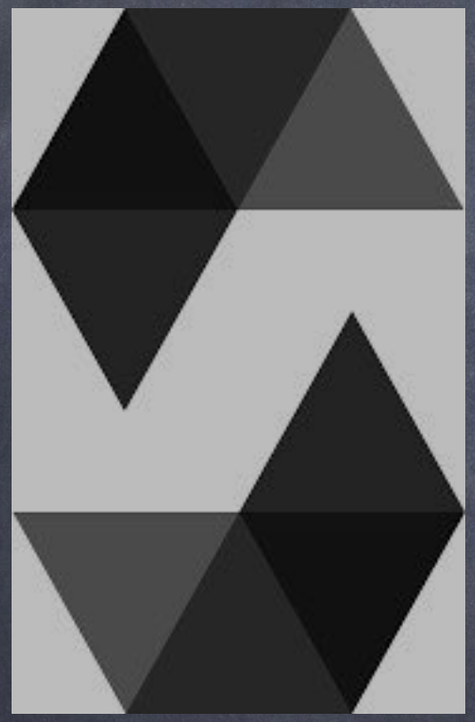
Smaller Type -> Higher Order Cut-off
Larger Type -> Higher Order Padded

Fixed-size: Bytes bytes1..bytes32
Smaller Type -> Lower Order Cut-off
Larger Type -> Lower Order Padded

#70

Conversions
Literals

Literals <> Elementary Types



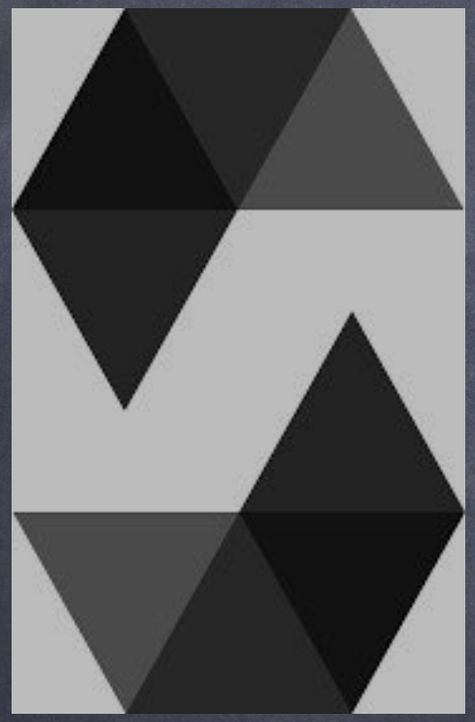
Decimals/Hex -> Integer
Decimals X-> Fixed-size byte Arrays

Hex -> Fixed-size byte Arrays
IFF #Hex Digits Exactly Fits

String/Hex String -> Fixed-size byte Arrays
IFF #Chars Exactly Fits

#71

Ether Units



Suffix -> Literal Number
wei, gwei, ether

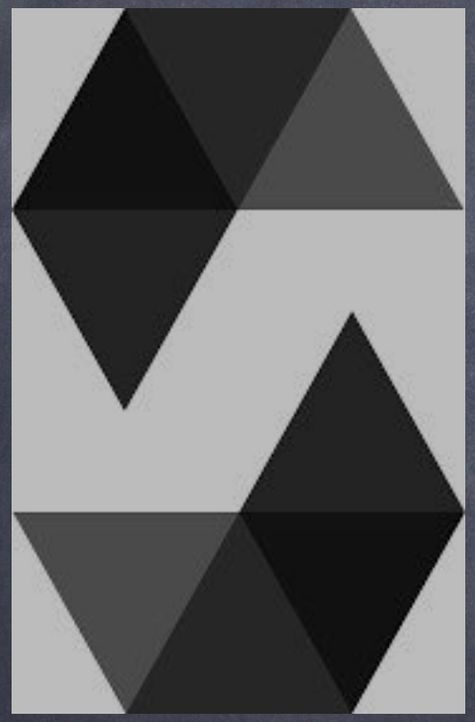
wei -> Base Unit
1 wei == 1

1 gwei == 1e9
1 ether == 1e18

Suffix
Multiplication by Power of 10

#72

Time Units



Suffix -> Literal Number
seconds/minutes/hours/days/
weeks

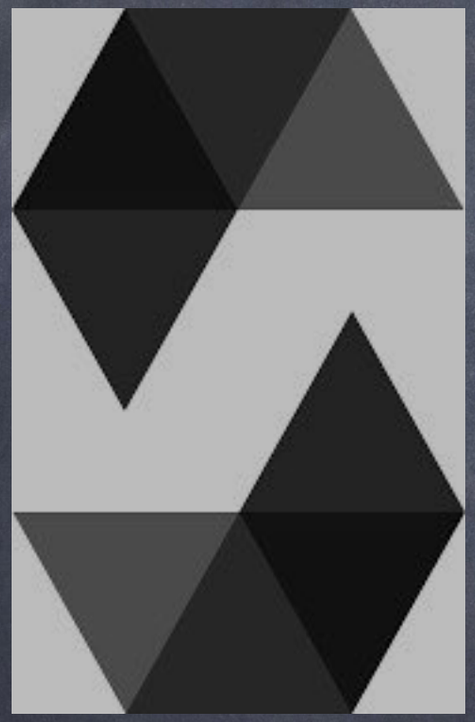
seconds -> Base Unit
1 == 1 seconds

1 minutes == 60 seconds, 1 hours
== 60 minutes, 1 days == 24
hours, 1 weeks == 7 days

Variables -> Multiply
E.g.: daysAfter * 1 days

#73

Block & Tx Properties



Block: Hash, ChainID, Number, Timestamp

Block: Coinbase, Difficulty, GasLimit

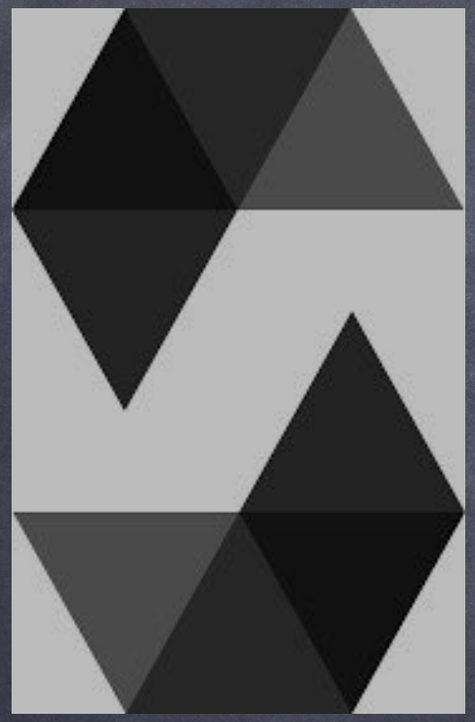
Msg: Value, Data, Sender, Sig

Tx: Gasprice, Gasleft, Origin

#74

Msg Values

Value -> wei Sent
Sender -> Sender Address



External Call -> Sender Changes

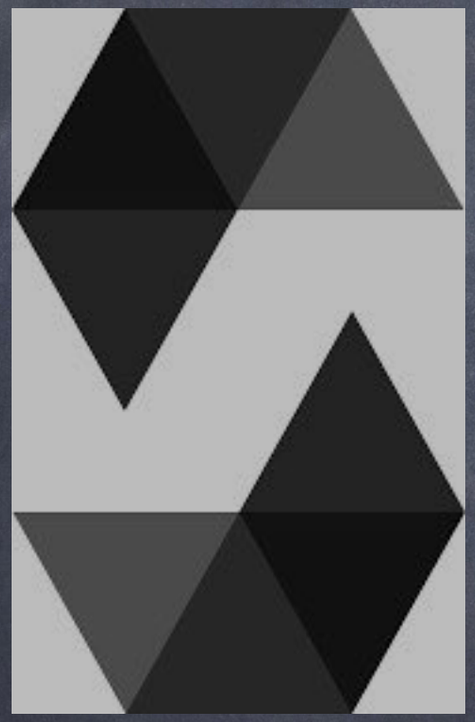
External Call -> Value Can Change

Security: Assumptions

#75

Randomness Source

block.timestamp
blockhash



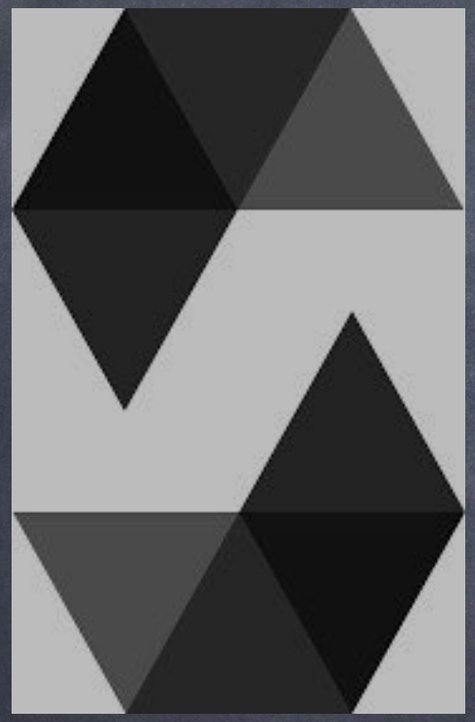
Miners -> Can Influence

Timestamp(block) > Timestamp(block-1)

Do Not Rely -> Source of Randomness

#76

Blockhash



blockhash(blockNumber)

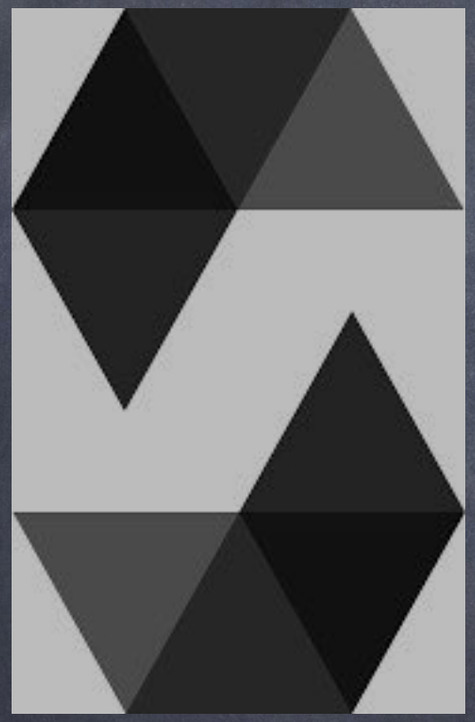
Hash of Given Block

Only 256 Recent Blocks
Excluding Current Block

All Other Values -> Zero

#77

ABI Encoding/
Decoding



abi.encode(...)
abi.decode(...)

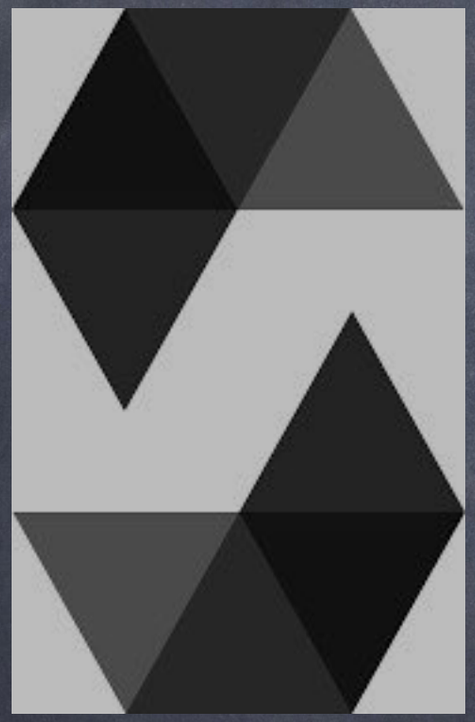
abi.encodeWithSelector(...)
abi.encodeWithSignature(...)

abi.encodePacked(...)

Packed Encoding ->
Ambiguous

#78

Error Handling



assert(bool condition) -> Panic/revert

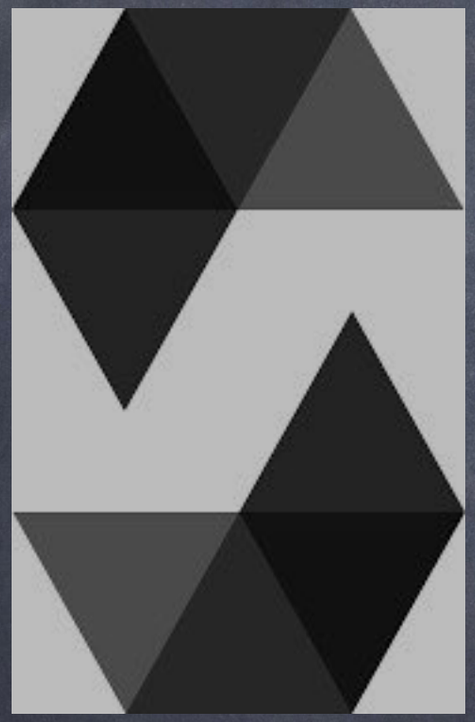
require(bool condition [, string memory message]) -> Error/revert

revert([string memory message]) -> Abort/revert

assert() vs require()

#79

Math/Crypto Functions



addmod() & mulmod()

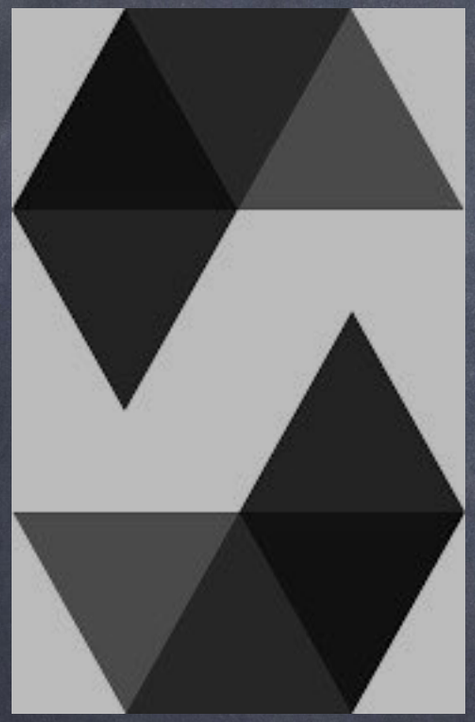
keccak256(bytes memory)

sha256(bytes memory)
ripemd160(bytes memory)

ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s)

#80

ecrecover
Malleability



Two Valid Signatures

Signature Malleability ->
Replay Attacks

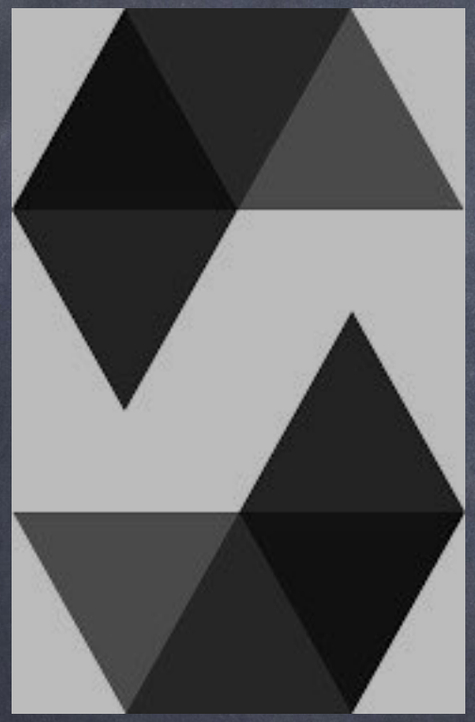
Sig -> (v, r, s)
s Values -> Lower Range

Unique Signatures ->
OpenZeppelin ECDSA

#81

Contract Related

this -> Current Contract
Convert -> Address



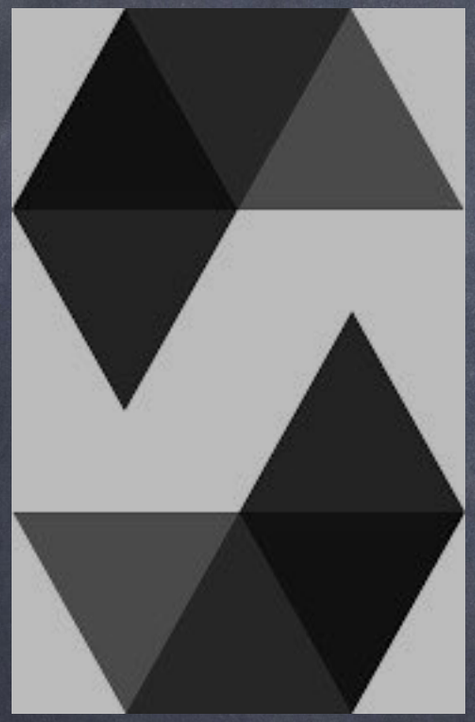
selfdestruct(address payable recipient)

selfdestruct(recipient) -> Destroy Current Contract

Send Funds -> Recipient
End Execution

#82

selfdestruct



Peculiarities

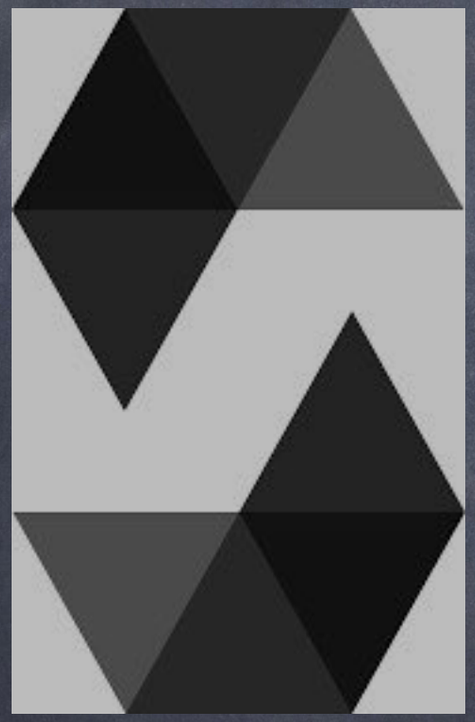
Recipient Contract receive() Not Executed

Contract Destroyed -> End of Tx

revert -> Undo Destruction

#83

Contract Type



type(X)
X -> Contract

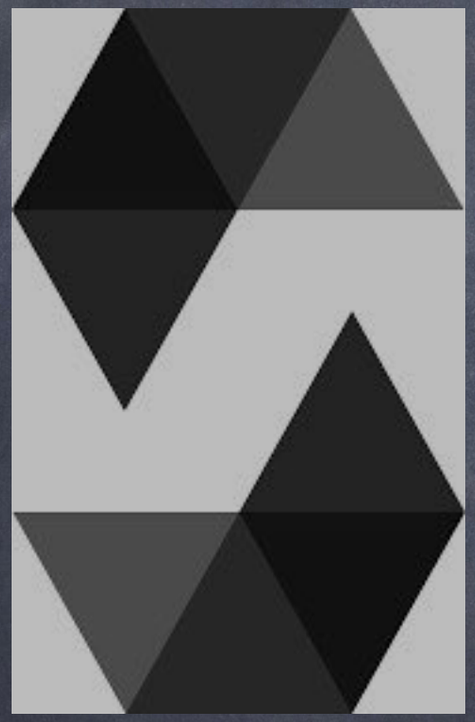
type(C).name: Contract Name

type(C).creationCode
type(C).runtimeCode

Interface I
type(I).interfaceId

#84

Integer Type



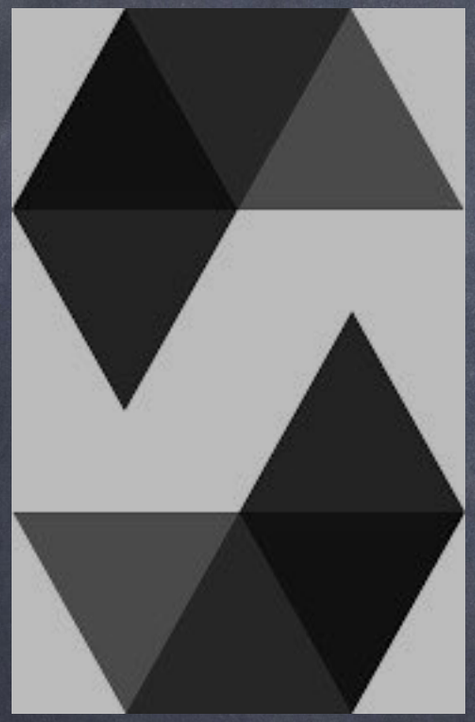
type(X)
X -> Integer

type(T).min
Smallest Value of T

type(T).max
Largest Value of T

E.g.: type(uint8).max == 255

Control Structures



Control Flow

if/else, while/do
for/break/continue/return

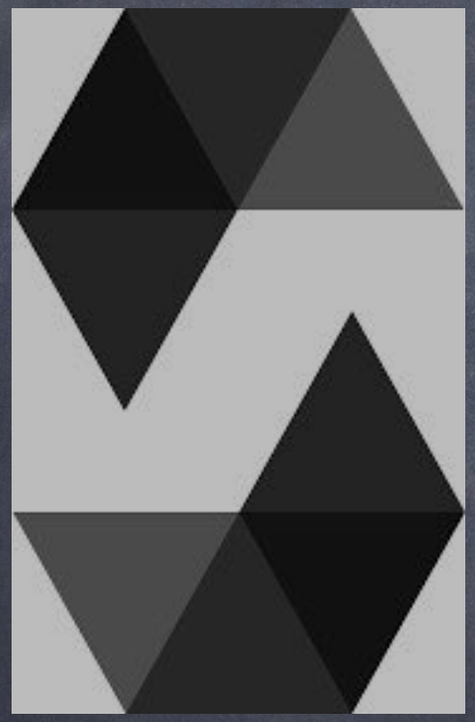
() X-> Omit
{} -> Omit Single Stmt Bodies

Non-boolean X-> Boolean
No Conversion E.g.: if (1)

#86

Exceptions

State-reverting Exceptions



Sub-call Exception -> Bubble-up
Except -> send/*call

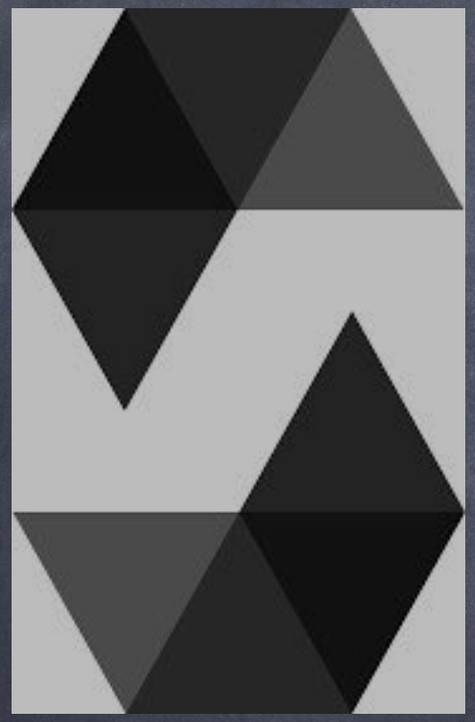
External Calls -> try/catch
Caller -> Selector+Data

Error(string) -> Regular
Panic(uint256) -> Assertions

#87

Low-level Calls

call/staticcall/
delegatecall



Non-existent Account
Return true

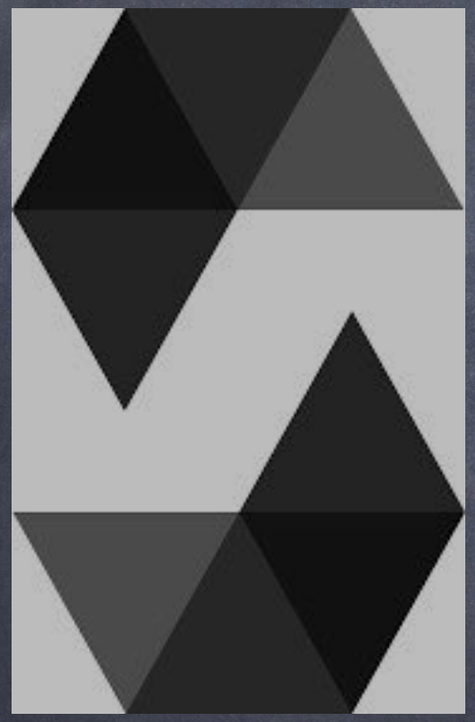
Check Account Existence
Before Call

Implication: No Contract ->
Assume Success

#88

Assert

assert() -> Panic(uint256)



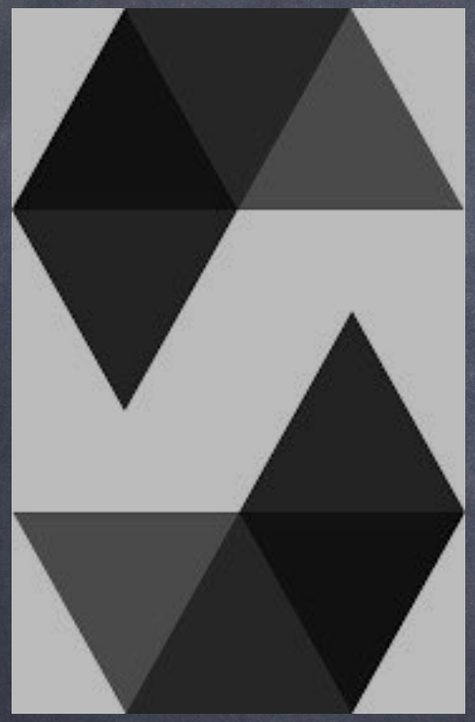
Internal Errors
Check Invariants

Normal Code -> No Panic

assert() Vs require()

#89

Panic



Panic Error Codes

0x01: false Argument
0x11: Overflow/Underflow

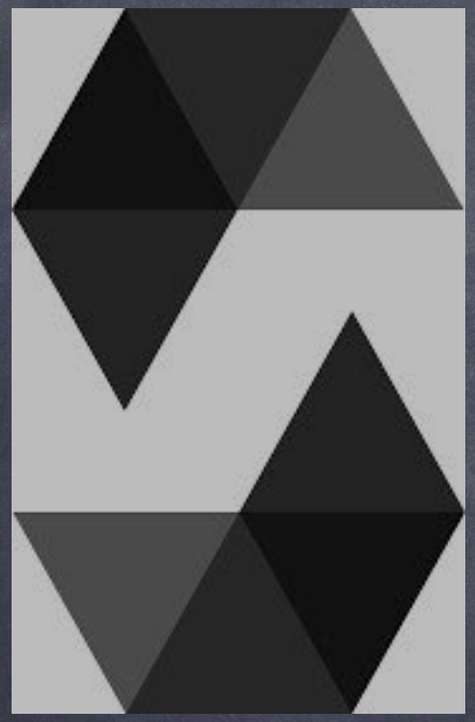
0x12: div/mod by 0
0x31: pop() Empty Array

0x32: Out-of-bounds Array Access..and Others

#90

Require

require() -> Error([string])



Detect Runtime Invalid Conditions

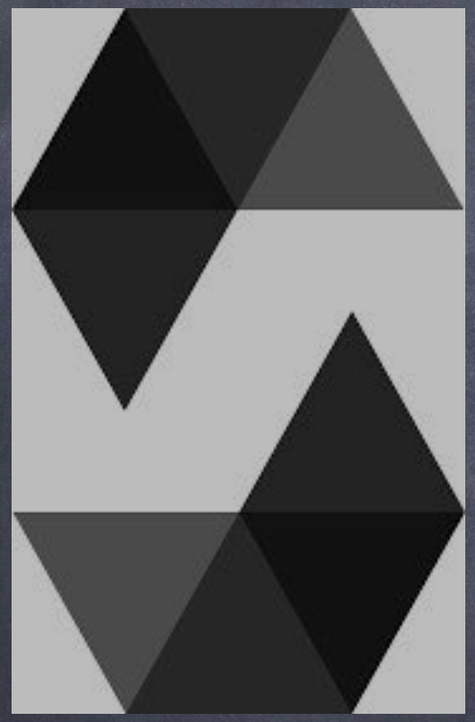
E.g.: Input Validation, Return Value Checks

Message String -> Optional

#91

Error

Error(string)



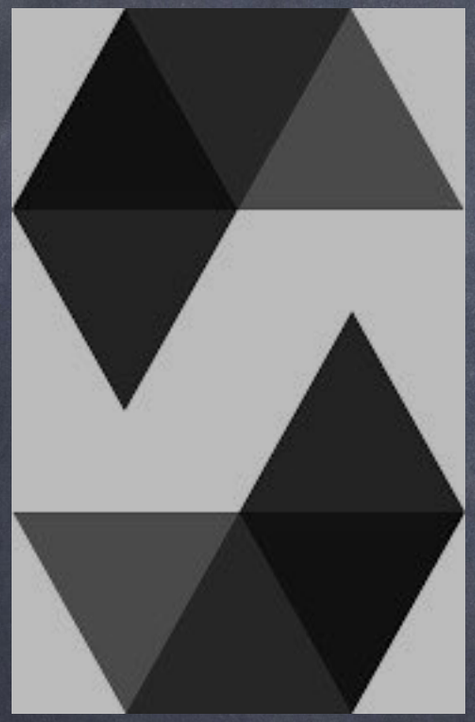
require(Arg == false)
revert([String])

External Call -> No Code

Receive ETH -> No payable
public Function/Getter

#92

Revert



revert CustomError(arg1,..)

revert([String])

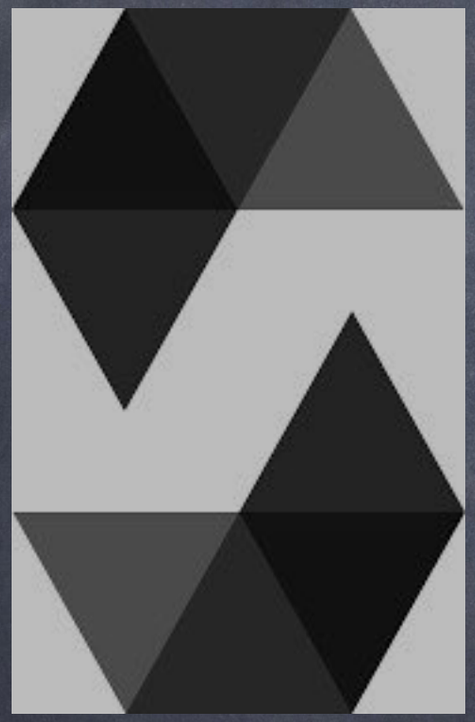
Abort Execution
Revert State Changes

Custom Error Vs String

#93

try/catch

try Expr [returns()] {..}
catch <Block> {..}



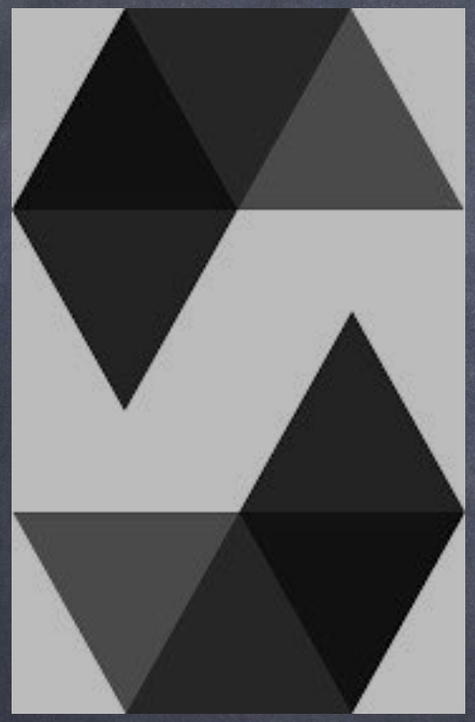
External Call Failure

No Error -> Success Block

Error -> catch Block
Block? -> Error Type

#94

catch Blocks



catch Error(string reason)

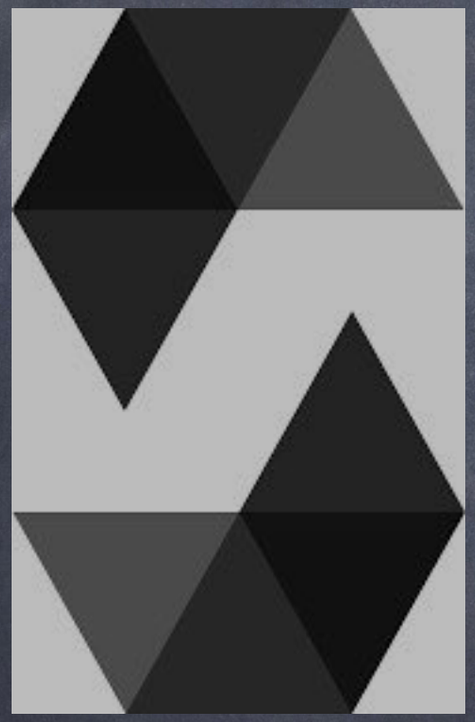
catch Panic(uint errorCode)

catch (bytes lowLevelData)

catch

#95

try/catch
State Change



Success Block Vs Error Block

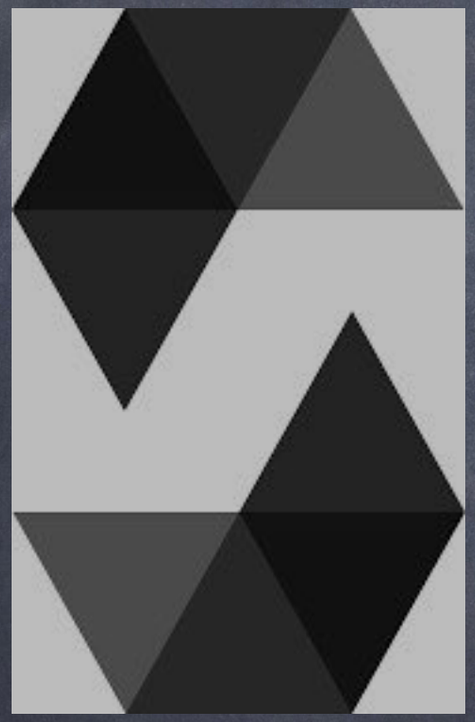
try/Success Block X-> Revert
External Call -> State Change

catch/Error Block -> Revert
External Call X-> State Change

try/catch Revert ->
Decoding/Low-level

#96

External Call Failure



Many Reasons

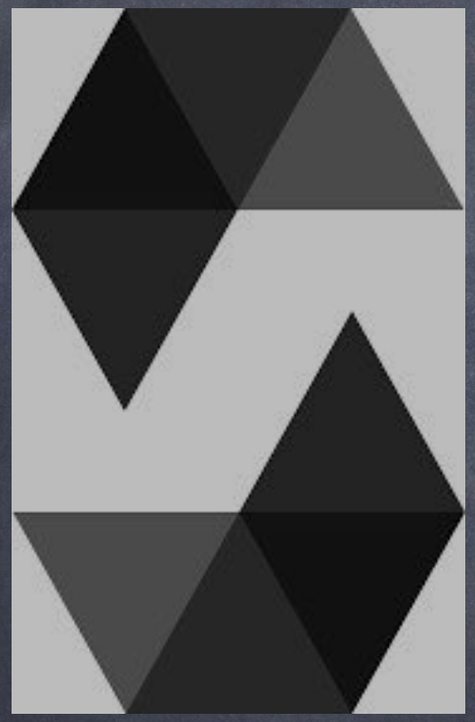
Error -> Called Contract Cannot Assume

Error -> Deeper Call Chain Callee Forwarder

OOG
Caller -> 63/64th Gas

#97

Programming Style



Coding Convention

Style -> Consistency

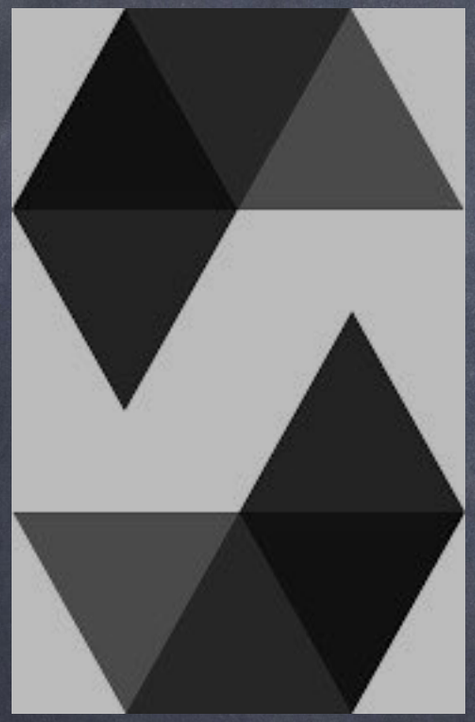
Consistency -> Function-Module-Project

Layout & Naming

#98

Code Layout

Indentation -> 4 Spaces
Spaces over Tabs X-> Mix



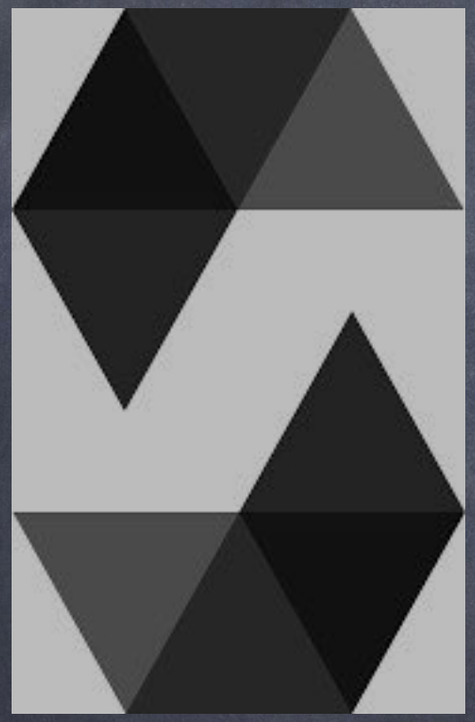
Blank Lines -> 2
Max Line Length -> 79/99

Wrapped Lines
Src File: UTF-8/ASCII

Imports -> Top
Fns -> constructor/external/
public/internal/private

#99

Code Layout
(More)



Whitespace in Expressions
Control Structures

Function Declaration
Mappings

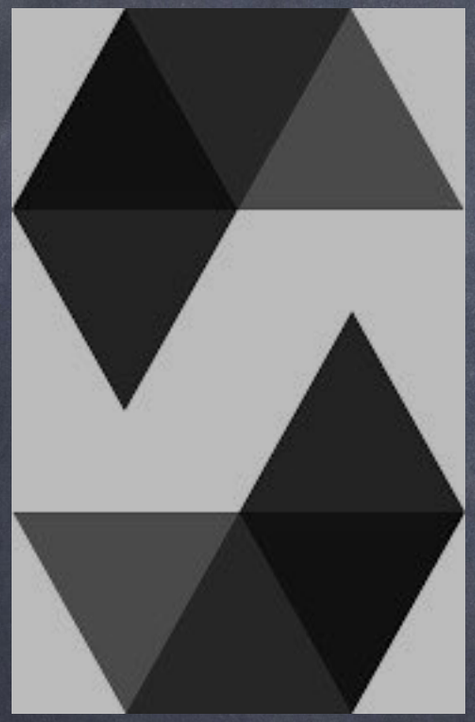
Variable Declaration
Strings -> "" Vs ''

Operators & Space
Order: pragma/import/contract/
types/state vars/events/functions

#100

Naming Convention

Types: lc/l_c/UC/U_C/CW/C_W/mC



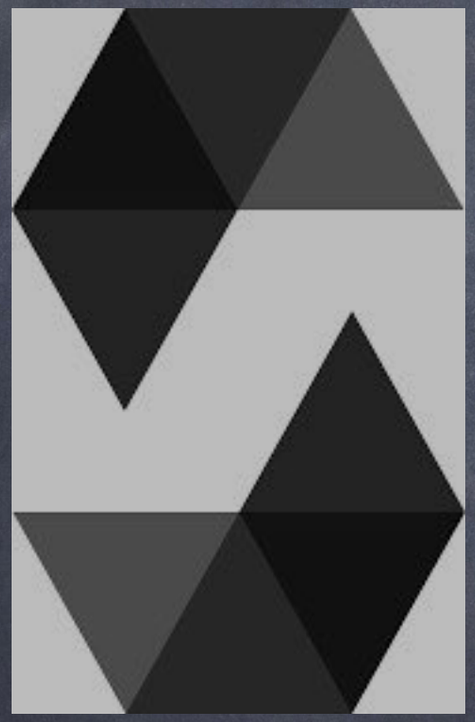
Avoid: l/I/0
Confused with 1 & 0

Contract -> CW
Struct -> CW

Event -> CW
Function -> mC

#101

Naming (More)



Function Arg -> mC
Local/State Var -> mC

Constant -> U_C
Modifier -> mC

Enum -> CW
Collision -> built-ins

Security: Readability & Maintainability