Clarity vs. Solidity

A WORKSHOP IN SMART CONTRACT LANGUAGES

clarity /-/iro

I’m Max Efremov

Developer Evangelist for Hiro

Why should we care about { Clarity | Solidity }?

Why shou e about { Clarity |

rekt

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
4. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
5. **Nomad Bridge - REKT** *N/A*
   $190,000,000 | 08/01/2022
6. **Beanstalk - REKT** *Unaudited*
   $181,000,000 | 04/17/2022
7. **Wintermute - REKT 2** *N/A*
   $162,300,000 | 09/20/2022
8. **Compound - REKT** *Unaudited*
   $147,000,000 | 09/29/2021
9. **Vulcan Forged - REKT** *Unaudited*
   $140,000,000 | 12/13/2021
10. **Cream Finance - REKT 2** *Unaudited*
    $130,000,000 | 10/27/2021

clarity

is a programming language for writing smart contracts on the Stacks 2.0 blockchain.

clarity

is a programming language for writing smart contracts on the Stacks 2.0 blockchain.

SOLIDITY

is a programming language for writing smart contracts on the Ethereum Virtual Machine.

Solidity is

Compiled

Turing Complete

(UNDECIDABLE)

Rules and Limitations

Rules and Limitations

- Statically-typed, JavaScript-like language
  - Booleans, integers, fixed point numbers, addresses, fixed-size byte arrays, and enums

Rules and Limitations

- Statically-typed, JavaScript-like language
  - Booleans, integers, fixed point numbers, addresses, fixed-size byte arrays, and enums
- Inheritance support (“contract”-oriented language, similar to OOP)

Rules and Limitations

- Statically-typed, JavaScript-like language
  - Booleans, integers, fixed point numbers, addresses, fixed-size byte arrays, and enums
- Inheritance support (“contract”-oriented language, similar to OOP)
- Application Binary Interface (ABI) facilitates inter-application operability

Rules and Limitations

- Vulnerable to over- and underflows

Rules and Limitations

- Vulnerable to over- and underflows
- Reentrancy

Rules and Limitations

- Vulnerable to over- and underflows
- Reentrancy
- Costly loops

Rules and Limitations

- Vulnerable to over- and underflows
- Reentrancy
- Costly loops
- Unchecked external calls (empty error responses)

Rules and Limitations

- Vulnerable to over- and underflows
- Reentrancy
- Costly loops
- Unchecked external calls (empty error responses)
- Various suboptimal defaults ([tx.origin], [require], function visibility, opt-out vs. opt-in, etc.)

(((Clarity is

Decidable

(NOT TURING COMPLETE)

Interpreted

(NOT COMPILED)

(((Clarity is

Decidable

(NOT TURING COMPLETE)

RAMIFICATIONS

(((Clarity is

Decidable

(NOT TURING COMPLETE)

RAMIFICATIONS

1. It is easier to analyze and debug the behavior of your code

(((Clarity is

Decidable

(NOT TURING COMPLETE)

RAMIFICATIONS

1. It is easier to analyze and debug the behavior of your code
2. Certain classes of hacks, bugs, and exploits are fundamentally impossible, such as reentrancy attacks

(((Clarity is

Decidable

(NOT TURING COMPLETE)

RAMIFICATIONS

1. It is easier to analyze and debug the behavior of your code
2. Certain classes of hacks, bugs, and exploits are fundamentally impossible, such as reentrancy attacks
3. Predict contract termination and runtime costs, allowing for precise gas estimation

clarity

Rules and Limitations

clarity

Rules and Limitations

- Interpreted (source code on blockchain)

clarity

Rules and Limitations

- Interpreted (source code on blockchain)
- Statically-typed, LISP-like parenthetical language
  - Booleans, integers, buffers, strings, principals (i.e. addresses), fixed-length lists

clarity

Rules and Limitations

- Interpreted (source code on blockchain)
- Statically-typed, LISP-like parenthetical language
  - Booleans, integers, buffers, strings, principals (i.e. addresses), fixed-length lists
- “Composite” types like optionals, tuples, and responses

clarity

Rules and Limitations

- Interpreted (source code on blockchain)
- Statically-typed, LISP-like parenthetical language
  - Booleans, integers, buffers, strings, principals (i.e. addresses), fixed-length lists
- “Composite” types like optionals, tuples, and responses
- **Trait-based** instead of inheritance (avoids implicit inheritance vulnerabilities)

clarity

Rules and Limitations

- Recursion is illegal

clarity

Rules and Limitations

- Recursion is illegal
- Looping may only be performed via `map`, `filter`, or `fold`

clarity

Rules and Limitations

- Recursion is illegal
- Looping may only be performed via `map`, `filter`, or `fold`
- Fixed-length lists

clarity

Rules and Limitations

- Recursion is illegal
- Looping may only be performed via `map`, `filter`, or `fold`
- Fixed-length lists
- Variables are **immutable**

clarity

Post Conditions

clarity

Post Conditions

- Post-conditions are part of a transaction payload

clarity

Post Conditions

- Post-conditions are part of a transaction payload
- Users can specify conditions like:

clarity

Post Conditions

- Post-conditions are part of a transaction payload
- Users can specify conditions like:
  - “This transaction can send no more than 20 STX from my balance”

clarity

Post Conditions

- Post-conditions are part of a transaction payload
- Users can specify conditions like:
  - “This transaction can send no more than 20 STX from my balance”
  - “I should receive at least 100 $REED tokens after this transaction”

clarity

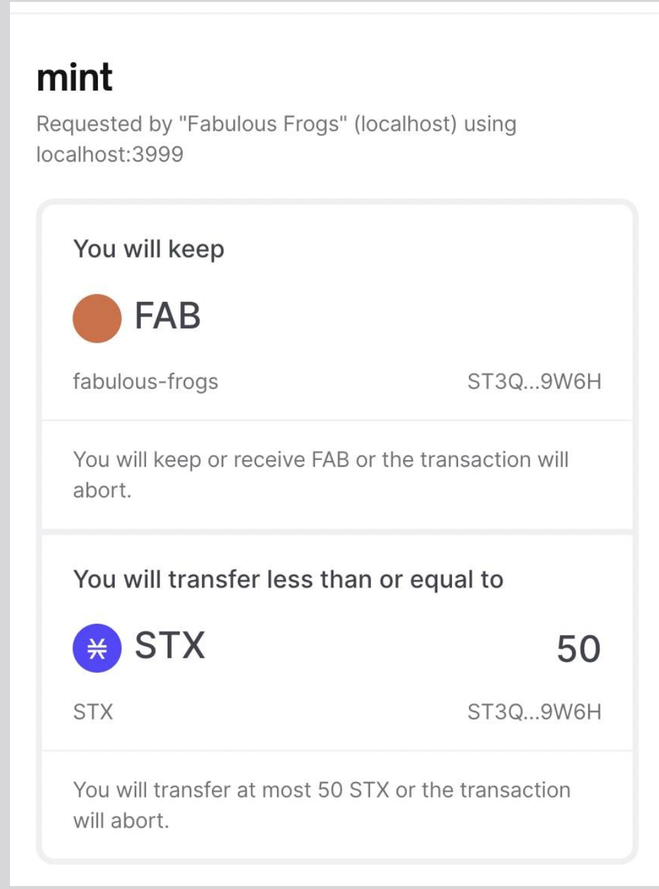
Post Conditions

- Post-conditions are part of a transaction payload
- Users can specify conditions like:
  - “This transaction can send no more than 20 STX from my balance”
  - “I should receive at least 100 $REED tokens after this transaction”
  - “I should own this NFT after this transaction”

clarity

Post Conditions

- Post-conditions are part of a transaction payload
- Users can specify conditions like:
  - “This transaction can send no more than 20 STX from my balance”
  - “I should receive at least 100 $REED tokens after this transaction”
  - “I should own this NFT after this transaction”



Contract = Data Space + Functions

hello-world.clar

```
;; hello.clar Defines a publicly-callable function `hello`
;; Takes a buffer of up to 40 bytes called `name`.
;; Always succeeds.
(define-public (hello (name (buff 40)))
   (begin
      (print “hello, “)
      (print name)
      (ok true)))   ;; returns (ok …) to indicate success
```

Simple contract example: `counter`

counter.clar

contracts > counter.clar

```
1
2   ;; counter
3   ;; contract that increments value of a count
4
5   (define-data-var count int 0)   ;; initalize count
6
7   (define-read-only (get-count)   ;; read value of count
8       (var-get count)
9   )
10
11  (define-public (increment)      ;; increment value of count
12      (ok (var-set count (+ (get-count) 1)))
13  )
14
15  (define-public (decrement)      ;; decrement value of count
16      (ok (var-set count (- (get-count) 1)))
17  )
18
```

Simple contract example: `counter`

counter.clar

contracts > counter.clar

```
1
2   ;; counter
3   ;; contract that increments value of a count
4
5   (define-data-var count int 0)   ;; initalize count
6
7   (define-read-only (get-count)   ;; read value of count
8       (var-get count)
9   )
10
11  (define-public (increment)      ;; increment value of count
12      (ok (var-set count (+ (get-count) 1)))
13  )
14
15  (define-public (decrement)      ;; decrement value of count
16      (ok (var-set count (- (get-count) 1)))
17  )
18
```

counter.sol

contracts > counter.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.4;
3
4   contract Counter {
5       int private count = 0;
6
7       function getCount() public view returns (int) {
8           return count;
9       }
10
11      function incrementCounter() public {
12          count += 1;
13      }
14
15      function decrementCounter() public {
16          count -= 1;
17      }
18  }
19
```

Reentrancy vulnerability

An external contract call that hands off control flow, leaving original contract vulnerable (to recursive `withdrawBalance` call for instance)

Reentrancy vulnerability

An external contract call that hands off control flow, leaving original contract vulnerable (to recursive `withdrawBalance` call for instance)

Let's say that contract A calls contract B.
Reentracy exploit allows B to call back into A
before A finishes execution.

Reentrancy vulnerability highlighted

```
(define-public (withdraw-balance)
  (let (
      (amountToWithdraw (default-to u0 (map-get? user-balances tx-sender)))
      (user tx-sender)
    )
    (map-set user-balances user u0)
    (try! (as-contract (stx-transfer? amountToWithdraw tx-sender user)))
    (ok amountToWithdraw)
  )
)
```

```
function withdrawBalance() {
  amountToWithdraw = userBalances[msg.sender];
  if (!(msg.sender.call.value(amountToWithdraw)()))
  {
    throw;
  }
  userBalances[msg.sender] = 0
}
```

Source: https://app.sigle.io/learnblock.id.blockstack/kjB7ymtoQg8qBiB6alPYa

Reentrancy vulnerability highlighted

```
(define-public (withdraw-balance)
  (let (
      (amountToWithdraw (default-to u0 (map-get? user-balances tx-sender)))
      (user tx-sender)
    )
    (map-set user-balances user u0)
    (try! (as-contract (stx-transfer? amountToWithdraw tx-sender user)))
    (ok amountToWithdraw)
  )
)
```

The Clarity code to perform a withdraw is much more explicit about **where** funds are being withdrawn from, and how errors can occur.

```
function withdrawBalance() {
  amountToWithdraw = userBalances[msg.sender];
  if (!(msg.sender.call.value(amountToWithdraw)()))
  {
    throw;
  }
  userBalances[msg.sender] = 0
}
```

While shorter, the Solidity *withdrawBalance* function relies on implicit behavior. This code is vulnerable to reentrancy—the same exploit that enabled The DAO attack

Source: https://app.sigle.io/learnblock.id.blockstack/kjB7ymtoQg8qBiB6alPYa


******
1 (define-data-var counter int 0)
2
3 (define-pu
4 (begin
5 (var-set
6 (ok (var-
BTC
Q+A

Start using Clarity today

Visit docs.hiro.so or clarity-lang.org and get started!

Further Reading:

- Clarity's "Hello World" quick start

Stacks Improvement Proposals (SIPs)

- SIP 002: Smart Contract Language
- SIP 005: Blocks, Transactions, and Accounts

Clarity Language Reference

clarity /-/iro