Fe vs Solidity

Comparing EVM smart contract languages

🙋‍♂️ pierrelf.eth

📅 2023-11-08

Contents

- Smart contract language design goals
- Solidity
- Fe
- State of the art
- Conclusion

We want complex contracts but…

… space on the blockchain is expensive 💰

```
600180600c6000396000f3fe00
```

Compiled programs
have to be super small

An analogy

Other considerations

- Portability (EVM)
- Resource efficiency
- Correctness

Solidity

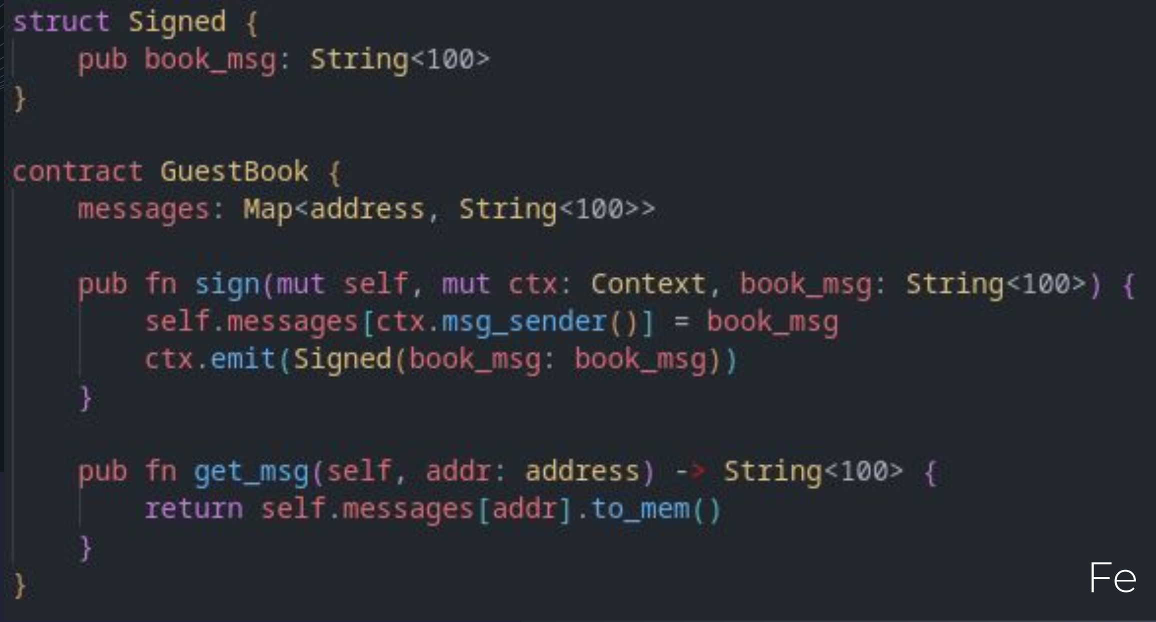
- De facto standard smart contract language for Ethereum Blockchain
- Feature rich
- Maturity
- Known security flaws
- Deprecated functions

- Compiles to EVM bytecode

Fe

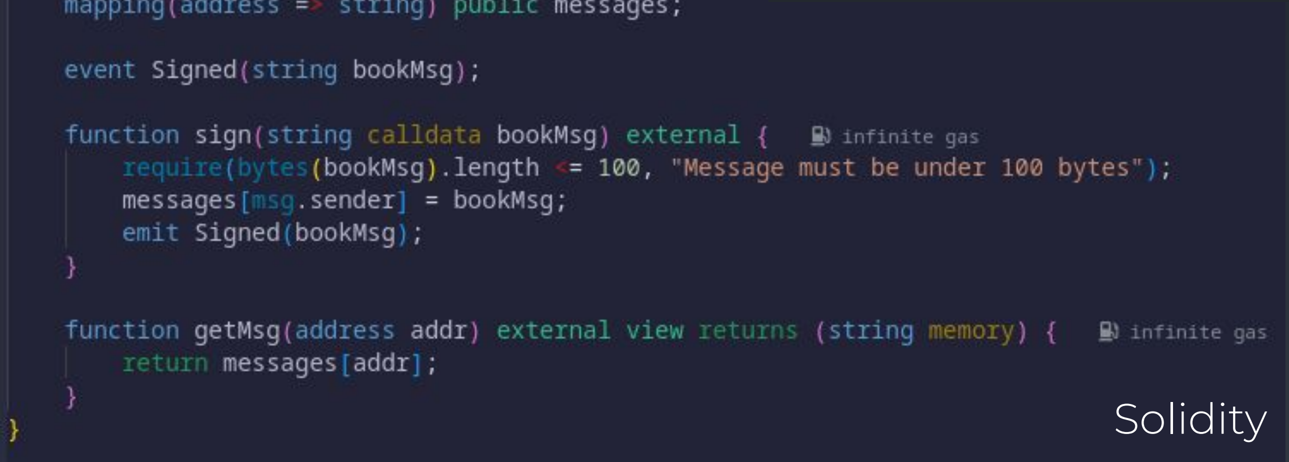
- Aims
  - Simpler semantics
  - Easier to verify
  - Less dynamic behavior - better gas cost prediction

- Python/Rust inspired syntax
- Compiles to EVM bytecode



```
struct Signed {
    pub book_msg: String<100>
}

contract GuestBook {
    messages: Map<address, String<100>>

    pub fn sign(mut self, mut ctx: Context, book_msg: String<100>) {
        self.messages[ctx.msg_sender()] = book_msg
        ctx.emit(Signed(book_msg: book_msg))
    }

    pub fn get_msg(self, addr: address) -> String<100> {
        return self.messages[addr].to_mem()
    }
}
```

Fe

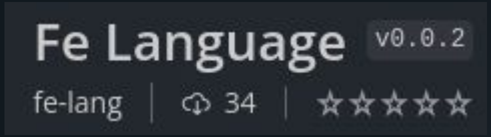


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract GuestBook {
    mapping(address => string) public messages;

    event Signed(string bookMsg);

    function sign(string calldata bookMsg) external {   infinite gas
        require(bytes(bookMsg).length <= 100, "Message must be under 100 bytes");
        messages[msg.sender] = bookMsg;
        emit Signed(bookMsg);
    }

    function getMsg(address addr) external view returns (string memory) {   infinite gas
        return messages[addr];
    }
}
```

Solidity

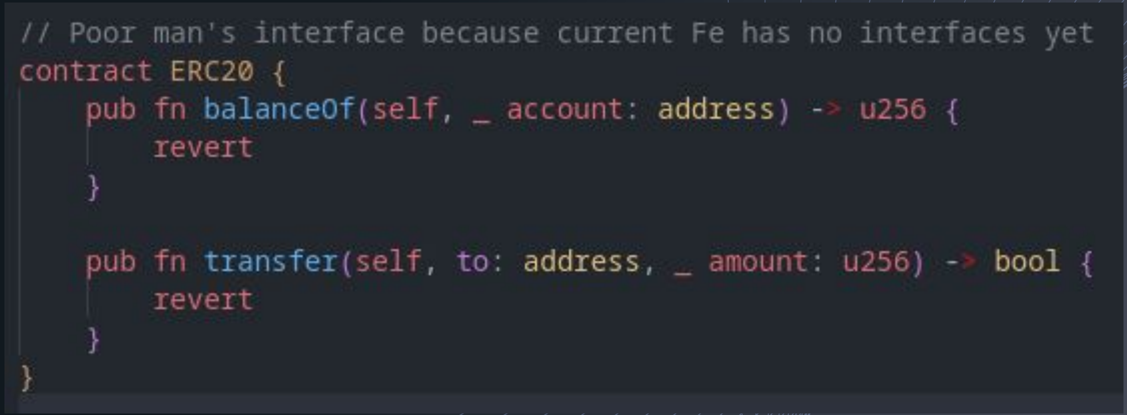
Fe

Downsides 😔

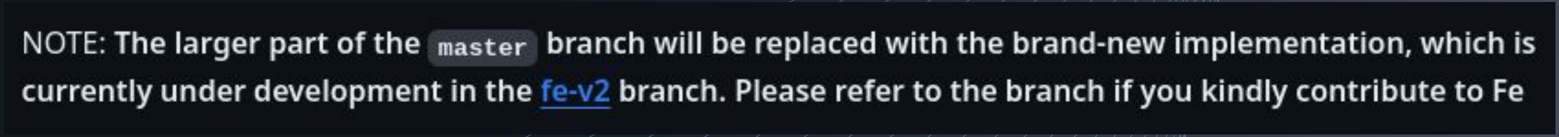


Fe Language v0.0.2

fe-lang | 34 | ☆☆☆☆☆



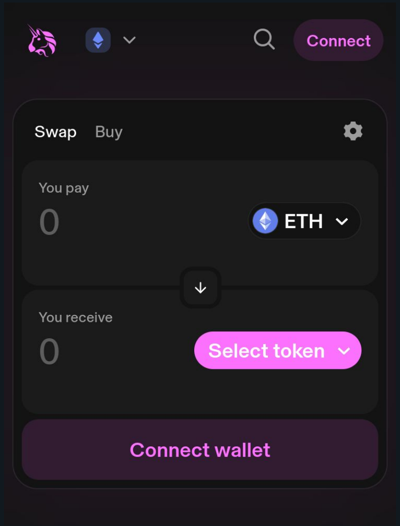
```
// Poor man's interface because current Fe has no interfaces yet
contract ERC20 {
    pub fn balanceOf(self, _ account: address) -> u256 {
        revert
    }

    pub fn transfer(self, to: address, _ amount: u256) -> bool {
        revert
    }
}
```

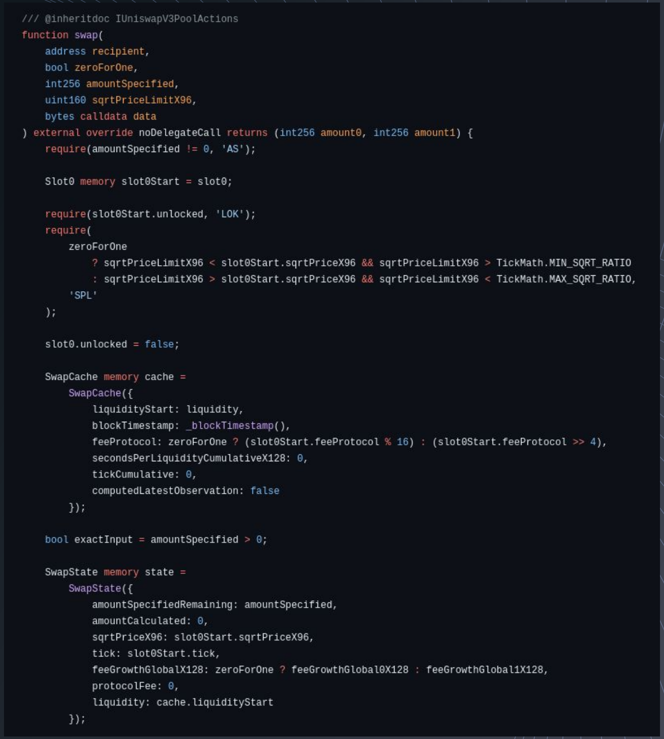


NOTE: The larger part of the `master` branch will be replaced with the brand-new implementation, which is currently under development in the fe-v2 branch. Please refer to the branch if you kindly contribute to Fe

State of the art

Solidity



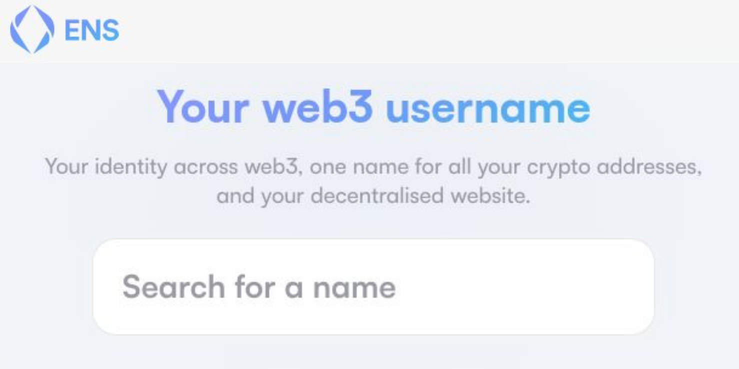


```
/// @inheritdoc IUniswapV3PoolActions
function swap(
    address recipient,
    bool zeroForOne,
    int256 amountSpecified,
    uint160 sqrtPriceLimitX96,
    bytes calldata data
) external override noDelegateCall returns (int256 amount0, int256 amount1) {
    require(amountSpecified != 0, 'AS');

    Slot0 memory slot0Start = slot0;

    require(slot0Start.unlocked, 'LOK');
    require(
        zeroForOne
            ? sqrtPriceLimitX96 < slot0Start.sqrtPriceX96 && sqrtPriceLimitX96 > TickMath.MIN_SQRT_RATIO
            : sqrtPriceLimitX96 > slot0Start.sqrtPriceX96 && sqrtPriceLimitX96 < TickMath.MAX_SQRT_RATIO,
        'SPL'
    );

    slot0.unlocked = false;

    SwapCache memory cache =
        SwapCache({
            liquidityStart: liquidity,
            blockTimestamp: _blockTimestamp(),
            feeProtocol: zeroForOne ? (slot0Start.feeProtocol % 16) : (slot0Start.feeProtocol >> 4),
            secondsPerLiquidityCumulativeX128: 0,
            tickCumulative: 0,
            computedLatestObservation: false
        });

    bool exactInput = amountSpecified > 0;

    SwapState memory state =
        SwapState({
            amountSpecifiedRemaining: amountSpecified,
            amountCalculated: 0,
            sqrtPriceX96: slot0Start.sqrtPriceX96,
            tick: slot0Start.tick,
            feeGrowthGlobalX128: zeroForOne ? feeGrowthGlobal0X128 : feeGrowthGlobal1X128,
            protocolFee: 0,
            liquidity: cache.liquidityStart
        });
```

Uniswap - Trillions of USD - 100% uptime
Use Smart Contracts to swap currencies

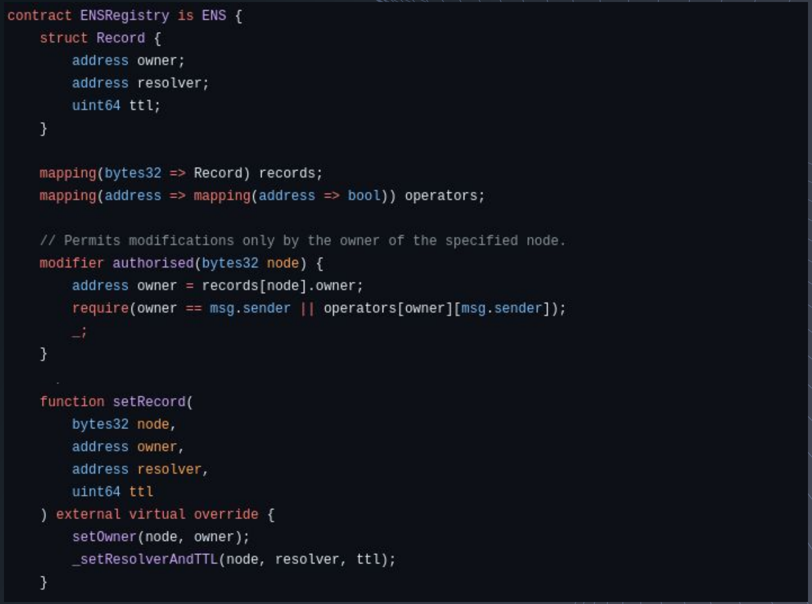
State of the art

Solidity



Your web3 username

Your identity across web3, one name for all your crypto addresses, and your decentralised website.



```
contract ENSRegistry is ENS {
    struct Record {
        address owner;
        address resolver;
        uint64 ttl;
    }

    mapping(bytes32 => Record) records;
    mapping(address => mapping(address => bool)) operators;

    // Permits modifications only by the owner of the specified node.
    modifier authorised(bytes32 node) {
        address owner = records[node].owner;
        require(owner == msg.sender || operators[owner][msg.sender]);
        _;
    }

    function setRecord(
        bytes32 node,
        address owner,
        address resolver,
        uint64 ttl
    ) external virtual override {
        setOwner(node, owner);
        _setResolverAndTTL(node, resolver, ttl);
    }
```

ENS uses smart contracts for DNS

State of the art

Fe

Conclusion

If you like the bleeding edge, Fe is for you!