Lecture 35 - Alternative to Solidity

in the news

1. Axie Infinite, $625 million (150,000 ETH) theft was done by the Lazarus criminal group. This is the same folks that did the WanaCry Raonsomware - and it is a state sponsored North Korea entity.
2. Ethereum-based Stablecoin protocol Beanstalk looses $182 million - goes from stable to 0.

Vyper

There is “vyper” a python like contract language that is the most popular alternative to Solidity.

From the vyper docs:

Vyper is a contract-oriented, pythonic programming language that targets the Ethereum Virtual Machine (EVM).

Vyper was specifically developed to address the security issues which were there in Solidity. It is modeled after python, as you might have guessed from the name.

Vyper has dropped some Object-Oriented concepts such as Inheritance. This makes the contracts muck easier to read - you don’t have to chaise all over to find the methods. Also it eliminates problems with dependencies on foreign code.

Vyper goals 1. Make contracts auditable 2. Make contracts more secure 3. Make contracts less error-prone 4. Make contracts easier to understand 5. Make it easier to learn the contract programming language

Vyper is strongly typed and always uses math operations that are “safe”.

Vyper adds:

1. Bounds and overflow checking - always on.
2. Decidability: With vyper is is possible to compute a precise upper bound on the runtime (therefore the “gas”) for all functions.

Vyper skips: 1. modifiers 2. class inheritance 3. recursive calls 4. dynamic data types - no unbounded arrays 5. Overflow 6. No Infinite Loops 7. Assembly support -- this means no “proxy” contracts. But you can build a standard Solidity contract that is a proxy and call a vyper contract.

To Install it you need python - I am using Anaconda Python.
https://www.anaconda.com/products/distribution

Then:

```
$ pip install vyper
```

You should be able to get the version of `vyper` with.

```
$ vyper --version
```

Files in `vyper` are `.vy` files.

You still need a Solidity `solc` install to be able to use truffle and ganache. So will need to have:

Really Simple Contract

```
 1: # Minimual contract that stores a value
 2:
 3: stored_data: uint256
 4:
 5: @external
 6: def set(new_value : uint256):
 7:     self.stored_data = new_value
 8:
 9: @external
10: @view
11: def get() -> uint256:
12:     return self.stored_data
```

1. Comments start with a '#'
2. No special stuff for declaring the licenses for the contract
3. No special pragma to define the version of the language
4. No name for the contract - it is the file name.
5. Contract-global data is defined in the outer scope (lines 2..4)
6. External functions are declared "external" with a "decorator", `@external` . (Line 5)
7. Functions use `def name` (line 6)
8. Declarations are `name : type` (line 6)
9. Data is referred to with `self.` (line 7)
10. non-transactional functions (views) are declared with a decorator, `@view` . (Line 10)
11. The return type for a function is specified with `-> type` (line 11)
12. Way fewer reserved words - just "return" (line 12)

```
 1: const VyperStorage = artifacts.require("VyperStorage");
 2:
 3: contract("VyperStorage", () => {
 4:   it("...should store the value 89.", async () => {
 5:     const storage = await VyperStorage.deployed();
 6:
 7:     // Set value of 89
 8:     await storage.set(89);
 9:
10:     // Get stored value
11:     const storedData = await storage.get();
12:
13:     assert.equal(storedData, 89, "The value 89 was not stored.");
14:   });
15: });
```

testing... 1. Still using JS and Mocha. 2. Very little difference between the test. 3. Line 1 - load the contract 4. Line 7 - call to method - transaction 4. Line 11 - call to view 4. Line 13 - check data is correct

A Vote Example

```
 1: # Voting with delegation.
 2:
 3: # Information about voters
 4: struct Voter:
 5:     # weight is accumulated by delegation
 6:     weight: int128
 7:     # if true, that person already voted (which includes voting by delegating)
 8:     voted: bool
 9:     # person delegated to
10:     delegate: address
11:     # index of the voted proposal, which is not meaningful unless `voted` is True.
12:     vote: int128
13:
14: # Users can create proposals
15: struct Proposal:
16:     # short name (up to 32 bytes)
17:     name: bytes32
18:     # number of accumulated votes
19:     voteCount: int128
20:
21: voters: public(HashMap[address, Voter])
22: proposals: public(HashMap[int128, Proposal])
23: voterCount: public(int128)
24: chairperson: public(address)
25: int128Proposals: public(int128)
26:
27: @view
28: @internal
29: def _delegated(addr: address) -> bool:
30:     return self.voters[addr].delegate != ZERO_ADDRESS
31:
32: @view
33: @external
34: def delegated(addr: address) -> bool:
35:     return self._delegated(addr)
36:
37: @view
38: @internal
39: def _directlyVoted(addr: address) -> bool:
40:     return self.voters[addr].voted and (self.voters[addr].delegate == ZERO_ADDRESS)
41:
42: @view
43: @external
44: def directlyVoted(addr: address) -> bool:
45:     return self._directlyVoted(addr)
46:
47: # Setup global variables / Constructor - Assume that there are a max of 4
48: # things to vote for.
49: @external
50: def __init__(_proposalNames: bytes32[4]):
51:     self.chairperson = msg.sender
52:     self.voterCount = 0
53:     for i in range(4):
54:         self.proposals[i] = Proposal({
```

```
            name: _proposalNames[i],
            voteCount: 0
        })
        self.int128Proposals += 1

# Give a `voter` the right to vote on this ballot.  This may only be called by the `chairperson`.
@external
def giveRightToVote(voter: address):
    assert msg.sender == self.chairperson # Throws if the sender is not the chairperson.
    assert not self.voters[voter].voted   # Throws if the voter has already voted.
    assert self.voters[voter].weight == 0 # Throws if the voter's voting weight isn't 0.
    self.voters[voter].weight = 1
    self.voterCount += 1

# Used by `delegate` below, callable externally via `forwardWeight`
@internal
def _forwardWeight(delegate_with_weight_to_forward: address):
    assert self._delegated(delegate_with_weight_to_forward)
    # Throw if there is nothing to do:
    assert self.voters[delegate_with_weight_to_forward].weight > 0

    target: address = self.voters[delegate_with_weight_to_forward].delegate
    for i in range(4):
        if self._delegated(target):
            target = self.voters[target].delegate
            # The following effectively detects cycles of length <= 5,
            # in which the delegation is given back to the delegator.
            # This could be done for any int128ber of loops,
            # or even infinitely with a while loop.
            # However, cycles aren't actually problematic for correctness;
            # they just result in spoiled votes.
            # So, in the production version, this should instead be
            # the responsibility of the contract's client, and this
            # check should be removed.
            assert target != delegate_with_weight_to_forward
        else:
            # Weight will be moved to someone who directly voted or
            # hasn't voted.
            break

    weight_to_forward: int128 = self.voters[delegate_with_weight_to_forward].weight
    self.voters[delegate_with_weight_to_forward].weight = 0
    self.voters[target].weight += weight_to_forward

    if self._directlyVoted(target):
        self.proposals[self.voters[target].vote].voteCount += weight_to_forward
        self.voters[target].weight = 0

    # To reiterate: if target is also a delegate, this function will need
    # to be called again, similarly to as above.

# Public function to call _forwardWeight
@external
def forwardWeight(delegate_with_weight_to_forward: address):
    self._forwardWeight(delegate_with_weight_to_forward)

# Delegate your vote to the voter `to`.
@external
def delegate(to: address):
    # Throws if the sender has already voted
    assert not self.voters[msg.sender].voted
    # Throws if the sender tries to delegate their vote to themselves or to
    # the default address value of 0x0000000000000000000000000000000000000000
    # (the latter might not be problematic, but I don't want to think about it).
    assert to != msg.sender
    assert to != ZERO_ADDRESS
```

```
121:
122:     self.voters[msg.sender].voted = True    # Mark weight as having benn assigned.
123:     self.voters[msg.sender].delegate = to    # Send to sombody else
124:
125:     # This call will throw if and only if this delegation would cause a loop
126:     # of length <= 5 that ends up delegating back to the delegator.
127:     self._forwardWeight(msg.sender)
128:
129: # Give your vote (including votes delegated to you)
130: # to proposal `proposals[proposal].name`.
131: @external
132: def vote(proposal: int128):
133:     assert not self.voters[msg.sender].voted # can't vote twice
134:     assert proposal < self.int128Proposals   # can only vote on legitimate proposals
135:
136:     self.voters[msg.sender].voted = True     # Mark as having voted
137:     self.voters[msg.sender].vote = proposal     # record what was voted for.
138:
139:     # transfer msg.sender's weight to proposal
140:     self.proposals[proposal].voteCount += self.voters[msg.sender].weight
141:     self.voters[msg.sender].weight = 0        #  Discard weight - so can not use again.
142:
143: # Computes the winning proposal, the proposal with the max number of botes.
144: @view
145: @internal
146: def _winningProposal() -> int128:
147:     winning_vote_count: int128 = 0    # Assume 0 votes to start
148:     winning_proposal: int128 = 0    # Assume proposal at locaiton 0
149:     for i in range(4):
150:         if self.proposals[i].voteCount > winning_vote_count:
151:             winning_vote_count = self.proposals[i].voteCount
152:             winning_proposal = i
153:     return winning_proposal
154:
155: @view
156: @external
157: def winningProposal() -> int128:
158:     return self._winningProposal()
159:
160:
161: # Calls winningProposal() function to get the index
162: # of the winner contained in the proposals array and then
163: # returns the name of the winner
164: @view
165: @external
166: def winnerName() -> bytes32:
167:     return self.proposals[self._winningProposal()].name
```

1. Structured data - (line 4..12) (line 15..19)
2. Declare hash - line(21..22)
3. constructor is called `__ini__` - same as in a python class. (line 50)
4. fixed length toleration of array (line 50) - `bytes32[4]` .
5. Loop with `for i in range(4)` - (line 54)
6. Assignment to map is just an assignment - no special use of “push”.
7. General “style” is to make internal functions start with `_` (underscore). (Line 71)
8. both declare and assign in one line - (line 76)
9. has “assert” - much more clear than “require” in solidity - same functionality (line 115). No message required then not returned to client.
10. has a pre-defined constant for 0 address, ZERO_ADDRESS (line 120)