Week 0 Assignment

Author: Paolo Antonio Rossi (luduvigo)

A. Conceptual Knowledge

1. **What is a smart contract? How are they deployed? You should be able to describe how a smart contract is deployed and the necessary steps.**

A smart contract is a program that is deployed on the blockchain. A smart contract runs when predetermined conditions are met, and generates a certain outcome, without the need of an intermediary. Deploying a smart contract involves sending a transaction, containing the bytecode of the smart contract, without specifying a recipient.
In order to deploy a smart contract we need to compile the smart contract code and generate the bytecode, create a deployment script, access to a blockchain node and run the deployment script to deploy the smart contract bytecode.

2. **What is gas? Why is gas optimization such a big focus when building smart contracts?**

Deploying and interacting with smart contracts costs a fee, that is commonly called gas. Gas is required to successfully conduct a transaction or execute a contract on blockchain. Gas price calculation depends on two factors: supply/demand and complexity of the operation that needs to be executed. Due to this gas optimization can make smart contract interactions cheaper.

3. **What is a hash? Why do people use hashing to hide information?**

Hashing is a process in which we generate new values using a hash function. A very important characteristic of an hashing algorithm is that is quite easy to generate a hash from an input, but it's practically impossible to retrieve the input from the hash result. One of the main usages of an hash is to generate, given a specific input, a fixed length string, that is calculated using that input. So, without sharing the input, we can prove that that specific input generated a specific hash.

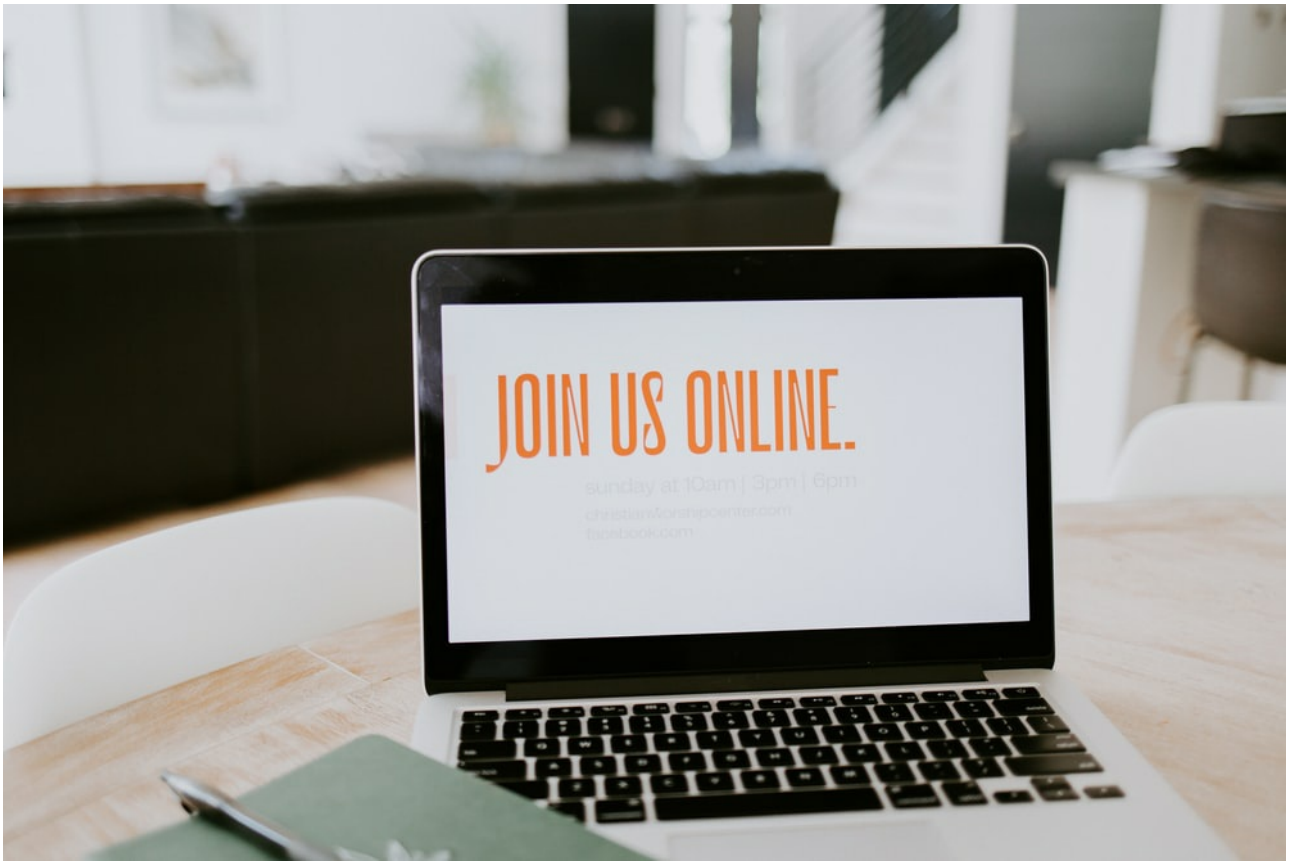
4. **How would you prove to a colorblind person that two different colored objects are actually of different colors?**

You could check out Avi Wigderson talk about a similar problem here (https://www.youtube.com/watch?v=5ovdoxnfFVc&t=4s).

One way could be to pick the two different coloured objects and tell him that one of them has a specific color. I will ask him to switch the two objects as many times as he wants behind his back, remembering which one was the special object. To prove that the objects have different colors I should be able to consistently tell him which is which, and prove that I can distinguish between the two colors.

B. You sure you're solid with Solidity?

1. **Program a super simple "Hello World" smart contract: write a storeNumber function to store an unsigned integer and then a retrieveNumber function to retrieve it. Clearly comment your code. Once completed, deploy the smart contract on remix (http://remix.ethereum.org/). Push the .sol file to Github or Gist and include a screenshot of the Remix UI once deployed in your final submission pdf.**



.

2. **On the documentation page, [the "Ballot" contract (https://docs.soliditylang.org/en/v0.8.11/solidity-by-example.html#voting)](https://docs.soliditylang.org/en/v0.8.11/solidity-by-example.html#voting) demonstrates a lot of features on Solidity. Read through the script and try to understand what each line of code is doing.**

3. **Suppose we want to limit the voting period of each Ballot contract to 5 minutes. To do so, implement the following: Add a state variable startTime to record the voting start time. Create a [modifier (https://www.youtube.com/watch?v=b6FBWsz7Val)](https://www.youtube.com/watch?v=b6FBWsz7Val) voteEnded that will check if the voting period is over. Use that modifier in the vote function to forbid voting and revert the transaction after the deadline.**

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/// @title Voting with delegation.
contract Ballot {
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }
```

```
// This is a type for a single proposal.
struct Proposal {
    bytes32 name;   // short name (up to 32 bytes)
    uint voteCount; // number of accumulated votes
}

modifier voteEnded {
  require(block.timestamp <= startTime + 5 minutes, "Voting period has ended");
  _;
}

address public chairperson;

// This declares a state variable that
// stores a `Voter` struct for each possible address.
mapping(address => Voter) public voters;

// A dynamically-sized array of `Proposal` structs.
Proposal[] public proposals;

// Voting starting time.
uint startTime;

/// Create a new ballot to choose one of `proposalNames`.
constructor(bytes32[] memory proposalNames) {
    chairperson = msg.sender;
    voters[chairperson].weight = 1;
    startTime = block.timestamp;

    // For each of the provided proposal names,
    // create a new proposal object and add it
    // to the end of the array.
    for (uint i = 0; i < proposalNames.length; i++) {
        // `Proposal({...})` creates a temporary
        // Proposal object and `proposals.push(...)`
        // appends it to the end of `proposals`.
        proposals.push(Proposal({
            name: proposalNames[i],
            voteCount: 0
        }));
    }
}

// Give `voter` the right to vote on this ballot.
// May only be called by `chairperson`.
function giveRightToVote(address voter) external {
    // If the first argument of `require` evaluates
    // to `false`, execution terminates and all
    // changes to the state and to Ether balances
    // are reverted.
    // This used to consume all gas in old EVM versions, but
```

```
        // not anymore.
        // It is often a good idea to use `require` to check if
        // functions are called correctly.
        // As a second argument, you can also provide an
        // explanation about what went wrong.
        require(
            msg.sender == chairperson,
            "Only chairperson can give right to vote."
        );
        require(
            !voters[voter].voted,
            "The voter already voted."
        );
        require(voters[voter].weight == 0);
        voters[voter].weight = 1;
    }

    /// Delegate your vote to the voter `to`.
    function delegate(address to) external {
        // assigns reference
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You already voted.");

        require(to != msg.sender, "Self-delegation is disallowed.");

        // Forward the delegation as long as
        // `to` also delegated.
        // In general, such loops are very dangerous,
        // because if they run too long, they might
        // need more gas than is available in a block.
        // In this case, the delegation will not be executed,
        // but in other situations, such loops might
        // cause a contract to get "stuck" completely.
        while (voters[to].delegate != address(0)) {
            to = voters[to].delegate;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
        }

        // Since `sender` is a reference, this
        // modifies `voters[msg.sender].voted`
        sender.voted = true;
        sender.delegate = to;
        Voter storage delegate_ = voters[to];
        if (delegate_.voted) {
            // If the delegate already voted,
            // directly add to the number of votes
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
```

```
            delegate_.weight += sender.weight;
        }
    }

    /// Give your vote (including votes delegated to you)
    /// to proposal `proposals[proposal].name`.
    function vote(uint proposal) voteEnded external {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Has no right to vote");
        require(!sender.voted, "Already voted.");
        sender.voted = true;
        sender.vote = proposal;

        // If `proposal` is out of the range of the array,
        // this will throw automatically and revert all
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /// @dev Computes the winning proposal taking all
    /// previous votes into account.
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    // Calls winningProposal() function to get the index
    // of the winner contained in the proposals array and then
    // returns the name of the winner
    function winnerName() external view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

4. **Deploy your amended script and test the newly implemented functionality in part 3. Submit (1) your amended version of the contract on Github or Gist and (2) screenshots showing the time of contract deployment as well as the transaction being reverted once past the voting period.**