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.

Smart Contracts are programmatic instructions that can be deployed in a blockchain to be interacted with later as a decentralized program capable of computation, storage and input and output of data.

- To deploy a smart contract, you will need:

- Your contract's bytecode
- ETH for gas
- A deployment script or plugin
- Access to an Ethereum node, either by running your own, connecting to a public node, or via an API key using a node service.

  If you are using Hardhat as a deployment plugin, here follows the instructions:

1. Start a local node

   npx hardhat node

2. Open a new terminal and deploy the smart contract in the localhost network

   npx hardhat run --network localhost scripts/deploy.js

As general rule, you can target any network configured in the hardhat.config.js

npx hardhat run --network <your-network> scripts/deploy.js

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

- Gas is the cost necessary to perform a transaction on the network. Miners set the price of gas based on supply and demand for the computational power of the network needed to process smart contracts and other transactions. Gas prices are denoted in small fractions of ether called gwei.

- Gas optimization is crucial for contract competitiveness because of the transactional costs involved.

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

   Hash is a none way cryptographic function that have some special properties such as: A tiny change on the message implies in a totally different has.

   Good hash functions should avoid collision.

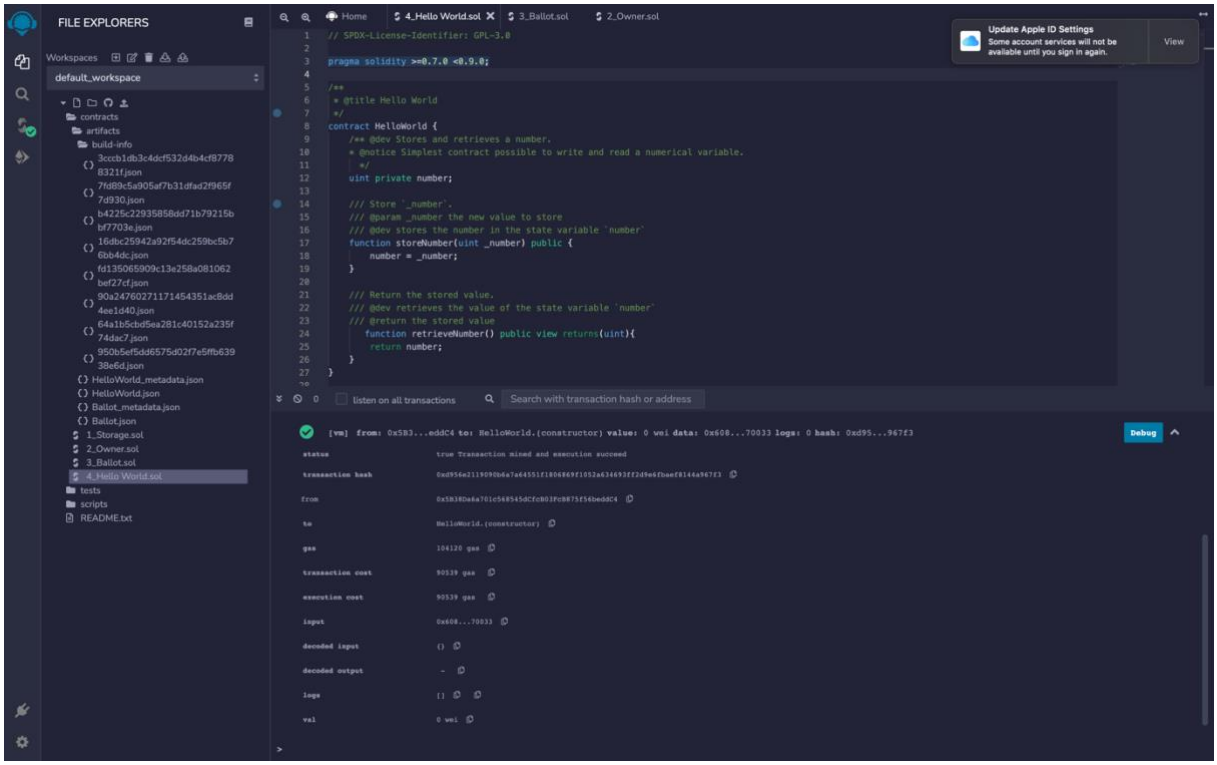
4. How would you prove to a colorblind person that two different colored objects are actually of different colors? You could check out Avi Wigderson talk about a similar problem here.

I would declare the colors of the balls in plain sight and ask for the person to mix (or not) the balls behind his/her back. I would consistently assert the different collors and as the person has the info if ball changed or not, after multiple interactions the degree of trust on the classification should raise.

Provide the answers to these questions in your submission.

. You sure you're solid with Solidity?

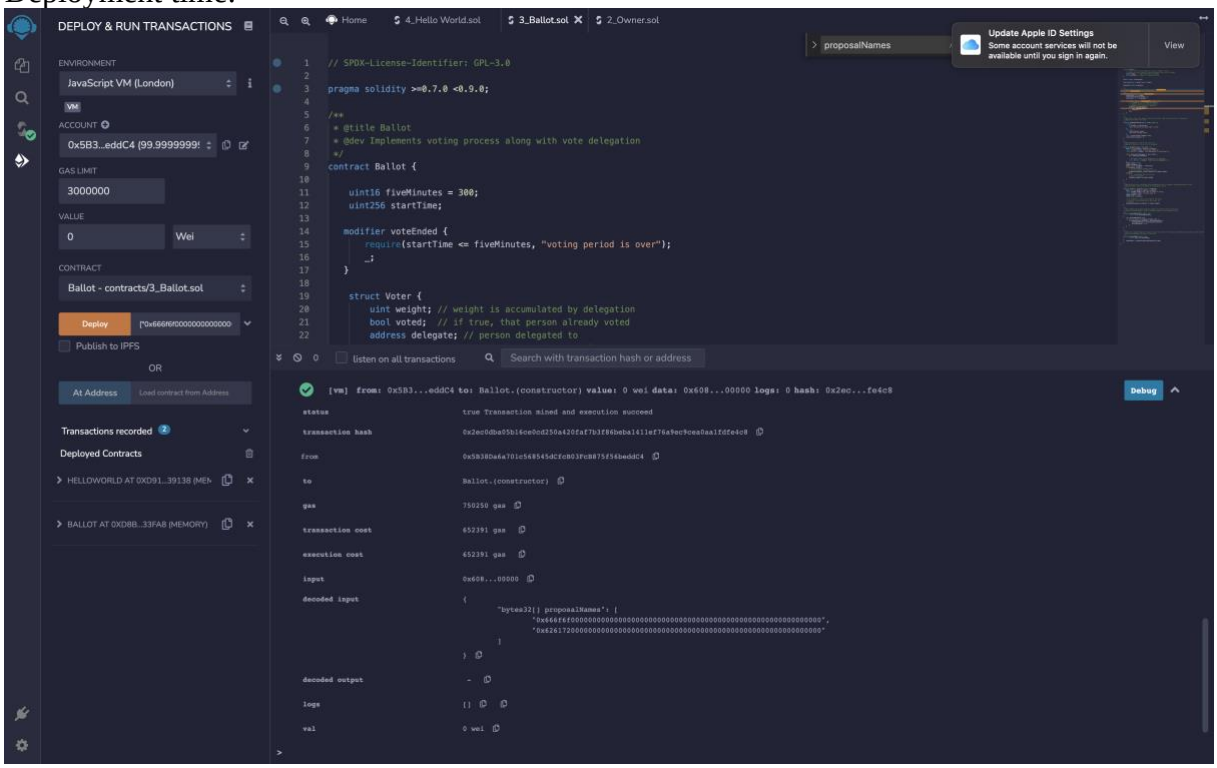
1. Program a super simple "Hello World" smart contract: write a `**storeNumber**` function to store an unsigned integer and then a `**retrieveNumber**` function to retrieve it. Clearly comment your code. Once completed, deploy the smart contract on remix. Push the .sol file to Github or Gist and include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

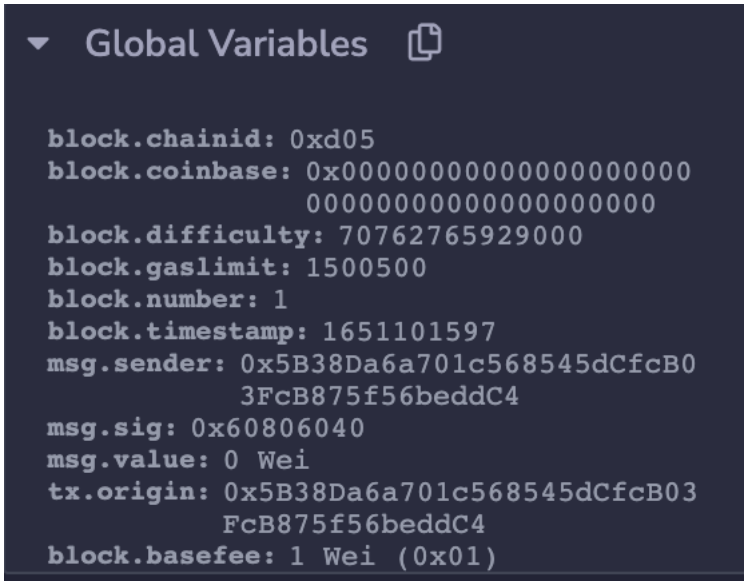


2. On the documentation page, the "Ballot" contract demonstrates a lot of features on Solidity. Read through the script and try to understand what each line of code is doing.
3. Suppose we want to limit the voting period of each Ballot contract to **5 minutes**. To do so, implement the following: Add a state variable **`startTime`** to record the voting start time. Create a modifier **`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.

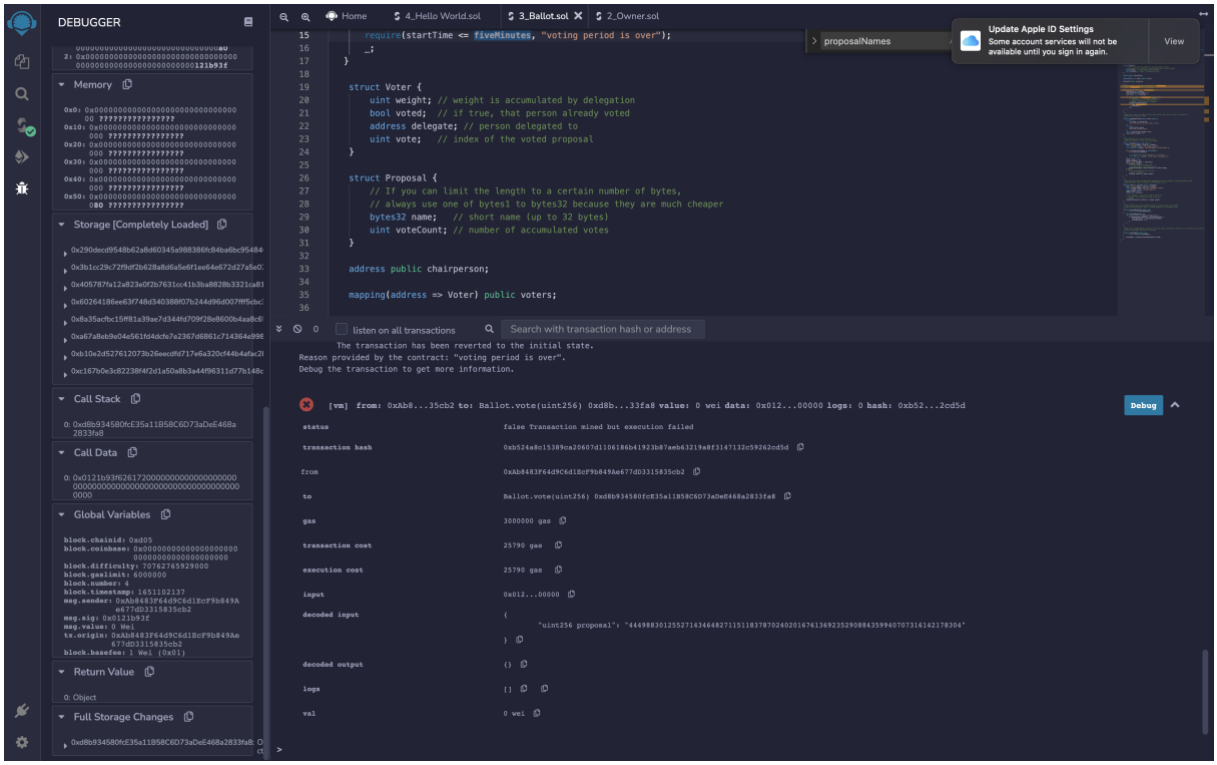
4. Deploy your amended script and test the newly implemented functionality in part 3. Submit (1) your amended version of the contract on Github or Gist and (2) screenshots showing the time of contract deployment as well as the transaction being reverted once past the voting period.

Deployment time:





Transaction Failing:


DEBUGGER
Home
4_Hello World.sol
3_Ballot.sol
2_Owner.sol
Update Apple ID Settings
Some account services will not be available until you sign in again.
View
Memory
Storage [Completely Loaded]
Call Stack
Call Data
Global Variables
Return Value
Full Storage Changes
proposalNames
require(startTime <= fiveMinutes, "voting period is over");
_;
}
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
}
struct Proposal {
// If you can limit the length to a certain number of bytes,
// always use one of bytes1 to bytes32 because they are much cheaper
bytes32 name; // short name (up to 32 bytes)
uint voteCount; // number of accumulated votes
}
address public chairperson;
mapping(address => Voter) public voters;
listen on all transactions
Search with transaction hash or address
The transaction has been reverted to the initial state.
Reason provided by the contract: "voting period is over".
Debug the transaction to get more information.
[vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xd8b...33fa8 value: 0 wei data: 0x012...00000 logs: 0 hash: 0xb52...2cd5d
Debug
status
false Transaction mined but execution failed
transaction hash
0xb524a8c15389ca20607d1106186b41923b87aeb63219a8f3147132e59262cd5d
from
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to
Ballot.vote(uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
gas
3000000 gas
transaction cost
25790 gas
execution cost
25790 gas
input
0x012...00000
decoded input
{
"uint256 proposal": "4449883012552714346482711511837870240201676136923529088435994070731614217830 4"
}
decoded output
{}
logs
[]
val
0 wei