Hey, Its Alp from Turkey. First of all I would like to thanks for this education because I really interest of blockchain and web3 technology so It will be next step for improvement.

Question 1:

Smart contracts are contracts that work just like today's contracts, where the contract is connected and approved, and everything is constant integrated into the blockchain.The process of integrating a smart contract into the blockchain is called a deployment process. Ides like Remix.Ide, hardhat or commands like build are used.The smart contract is written with the .sol extension, this contract is compiled by various editors and if there is no problem, this contract is deployed to the blockchain.

Question 2:

Gas means that you want to complete the necessary transactions to successfully conclude a contract.The importance of gas optimization is important to prevent high gas fees and to control the codes uploaded to the blockchain.For example infinite loop

Question 3:

The hash function is a kind of encryption algorithm. This algorithm is a blockchain encryption algorithm and is a representation of the security of the blockchain as it is impossible to decrypt with current facilities.

Question 4:

all colors and can be translated into a mathematical language, this language is universal for all (including color blind).For example black like 0 white like 255.

Code Questions:
Here is my GitHub repo https://github.com/alptoksoz/ZeroKnowledgeUniversity

And also deploy and transaction ss

****

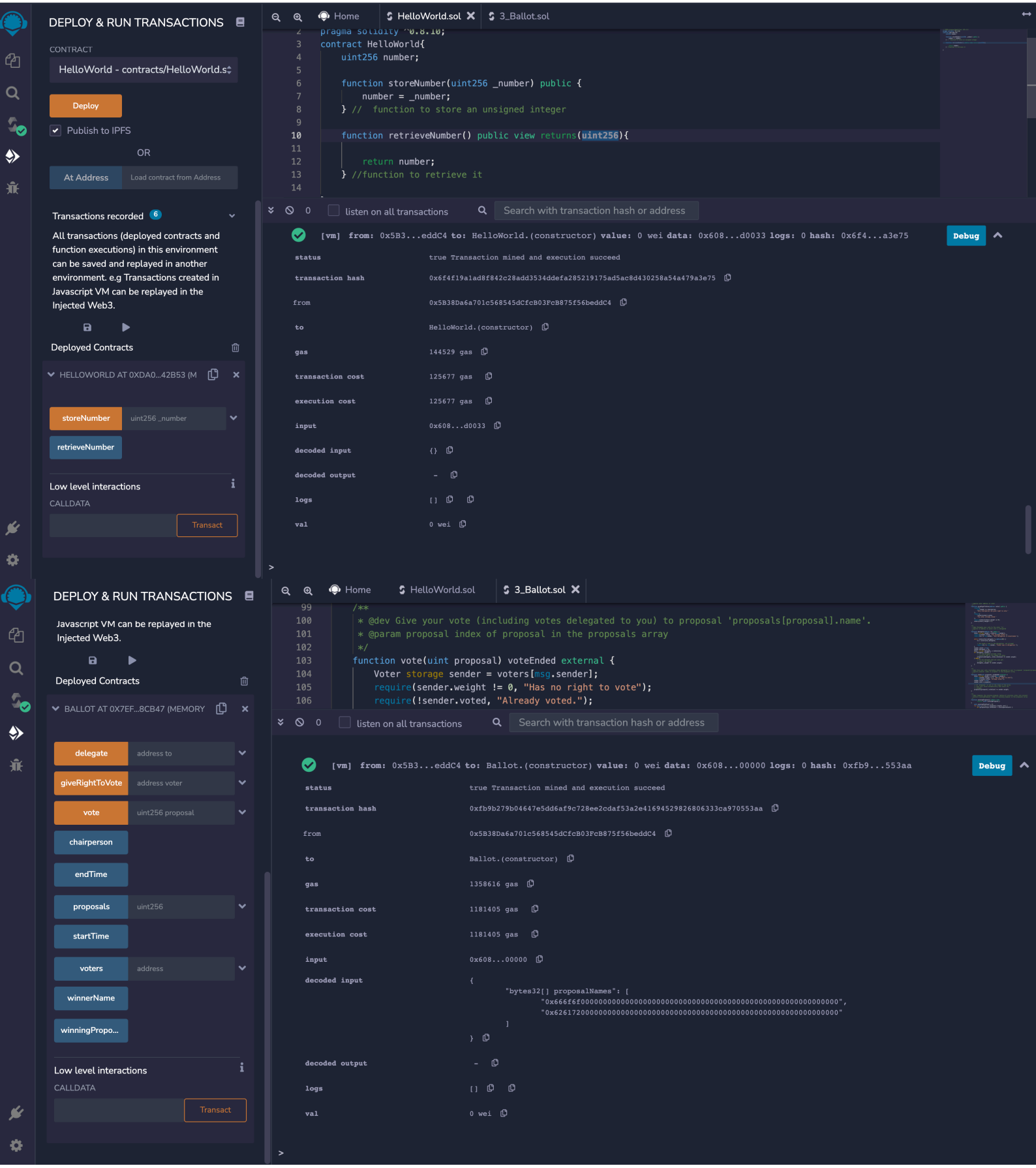
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.10;
contract HelloWorld{
    uint256 number;

    function storeNumber(uint256 _number) public {
        number = _number;
    } //  function to store an unsigned integer

    function retrieveNumber() public view returns(uint256){

        return number;
    } //function to retrieve it

}
```


DEPLOY & RUN TRANSACTIONS
CONTRACT
HelloWorld - contracts/HelloWorld.s
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 6
All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment. e.g Transactions created in Javascript VM can be replayed in the Injected Web3.
Deployed Contracts
HELLOWORLD AT 0XDA0...42B53 (M
storeNumber
uint256 _number
retrieveNumber
Low level interactions
CALLDATA
Transact
Home
HelloWorld.sol
3_Ballot.sol
pragma solidity ^0.8.10;
contract HelloWorld{
uint256 number;
function storeNumber(uint256 _number) public {
number = _number;
} // function to store an unsigned integer
function retrieveNumber() public view returns(uint256){
return number;
} //function to retrieve it
listen on all transactions
Search with transaction hash or address
[vm] from: 0x5B3...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...d0033 logs: 0 hash: 0x6f4...a3e75
Debug
status true Transaction mined and execution succeed
transaction hash 0x6f4f19a1ad8f842c28add3534ddefa285219175ad5ac8d430258a54a479a3e75
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to HelloWorld.(constructor)
gas 144529 gas
transaction cost 125677 gas
execution cost 125677 gas
input 0x608...d0033
decoded input {}
decoded output -
logs []
val 0 wei
DEPLOY & RUN TRANSACTIONS
Javascript VM can be replayed in the Injected Web3.
Deployed Contracts
BALLOT AT 0X7EF...8CB47 (MEMORY
delegate
address to
giveRightToVote
address voter
vote
uint256 proposal
chairperson
endTime
proposals
uint256
startTime
voters
address
winnerName
winningPropo...
Low level interactions
CALLDATA
Transact
/**
* @dev Give your vote (including votes delegated to you) to proposal 'proposals[proposal].name'.
* @param proposal index of proposal in the proposals array
*/
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.");
[vm] from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xfb9...553aa
Debug
status true Transaction mined and execution succeed
transaction hash 0xfb9b279b04647e5dd6af9c728ee2cdaf53a2e41694529826806333ca970553aa
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to Ballot.(constructor)
gas 1358616 gas
transaction cost 1181405 gas
execution cost 1181405 gas
input 0x608...00000
decoded input
"bytes32[] proposalNames": [
"0x666f6f0000000000000000000000000000000000000000000000000000000000",
"0x6261720000000000000000000000000000000000000000000000000000000000"
]
decoded output -
logs []
val 0 wei

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

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract Ballot {

    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;

    Proposal[] public proposals;
    uint256 public startTime = block.timestamp;
    uint256 public endTime = block.timestamp + 300 ;
    /**
     * @dev Create a new ballot to choose one of 'proposalNames'.
     * @param proposalNames names of proposals
     */
    constructor(bytes32[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        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
            }));
```

```
        }
    }

    /**
     * @dev Give 'voter' the right to vote on this ballot. May
only be called by 'chairperson'.
     * @param voter address of voter
     */
    function giveRightToVote(address voter) public {
        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;
    }

    /**
     * @dev Delegate your vote to the voter 'to'.
     * @param to address to which vote is delegated
     */
    function delegate(address to) public {
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You already voted.");
        require(to != msg.sender, "Self-delegation is
disallowed.");

        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.");
        }
        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;
        }
    }
```

```
    /**
     * @dev Give your vote (including votes delegated to you) to
proposal 'proposals[proposal].name'.
     * @param proposal index of proposal in the proposals array
     */
    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.
     * @return winningProposal_ index of winning proposal in the
proposals array
     */
    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;
            }
        }
    }

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

    modifier voteEnded{
        _;
        require((startTime=block.timestamp) <= endTime ,"Vote only
5 minutes,already finished");
    }
```

```
}
```