

ZKU ONE

SHREYAS VIVEK

shreyasvivek01@gmail.com

+91 9741002164

Q A.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.

Ans:

Smart contracts is basically a bunch of code written to perform a specific task, it could be vaguely considered as a written contract, for example, say you want to trade chocolates with your friend, but you want some kind of proof for that, so you would take it in writing, smart contracts are similar to that, they are just written in code, and basically agrees to do something.

***for someone who is over 5 yrs**

Smart contracts could be deployed in a similar way as transacting between 2 accounts, but the catch is, you just send the contract, without specifying the recipient address. This is a very basic idea of deployment of smart contracts.

You would need ETH for gas fees for deployment.

Main things needed are the contracts bytecode which can be generated after compiling the contracts on IDEs

The next thing that you would need is a deployment plugin or script, there's no specific deployment plugin, but we could use the hardhat plugin as it quite simple to use and execute

Hardhat deploy plugin would be to have a scripts/deploy.js

Code below:

```
async function main() {  
  // We get the contract to deploy  
  const      = await      .getContractFactory("Greeter");  
  const      = await      .deploy("Hello, Hardhat!");  
  
  await      .deployed();  
}
```

```

        .log("Greeter deployed to:", address);
    }

    main()
    .then(() => deployContract())
    .catch((error) => {
        .error(error);
        .exit(1);
    });
});

```

A sample code to deploy using hardhat greeter

Deploying on a local host would be

1) starting the local node

npx hardhat node

2) on a new in the terminal deploy the contract on localhost

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

Q A.2

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

Say you need to eat a burger, you would go to shop and pay some money and get your order made for you, and then you would eat it.

The money that you pay for the food is for the ingredients, facilities, equipment used, miscellaneous, and then skills, time of the person preparing it. In a similar way, while transacting on the blockchain you must pay a certain amount of money to the person/miner who completes your transaction. This is called gas fees. And it has to be paid to the miners for their work.

***for someone who is over 5 yrs**

Gas is basically a small fraction of fee that has to be paid in order for transactions to be successful.

This gas fee is usually a small fraction in of the ETH it is used to allocate resources of the Ethereum virtual machine(EVM) and it is variable, for the same contract at different times, because it is basically a supply demand chain,

ie, if there are many requests, gas fees would be higher on average when compared to the fees when the request is not that much.

Gas optimization is quite essential when considering the overall smart contract because, higher gas fees would mean that fewer users would be utilizing the contract once deployed. But if we consider the overall cost for transaction.

One thing to note would be that if gas is optimized in one part of the contract, it would be increased in another part. So it would be quite crucial to decide when and where you have to optimize the gas.

Gas optimization would refer to minimizing the gas fees for smart contracts. And this could be optimized by an inbuilt solidity function

```
10 module.exports = {  
  9 ...  
  8 solc: {  
    7 optimizer: {  
      6 enabled: true,  
      5 runs: 200  
    4 }  
  3 }  
  2 }
```

Q A.3 Q B.1

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

Lets say you're in the middle of a very boring class. And you want to share a secret with your best friend who is sitting far away. You could write your message in a piece of paper and pass it on through your friends. But in doing so, everyone who passes your paper, would be able to know your secret. But you don't want anyone else to know your secret.

So what you could do is make a secret language with your best friend and then write the message In the secret language. Similarly in the world of computer science, the process of "making a secret language is called *ENCRYPTION* and the study of it is called *CRYPTOGRAPHY*. *HASHING* is one such type of

encryption and it is basically a mathematical function used to make this secret language and it no one could be able to decode or understand it. That's why many people tend to use hashing for protecting their secrets.

***for someone who is over 5 yrs**

Hash is basically a mathematical function that is used to convert any arbitrary input of variable length to an output of fixed length. This output is highly encrypted and is used mainly for its security/ encryption feature. Every hash value is unique, and only one hash value is generated for an input.

Many people use hashing to hide information is because, It can't be reverse engineered to get the input from the hashed value similar to a cyclic redundancy check or a checksum, it can be used to validate the authenticity or originality of the information. Because, for a given input there will be just one hash value generated, and if at all the data was tampered, it would lead to a change in the hash. When compared with the original data, the difference would indicate the data was tampered.

Q B.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.

HelloWorld_ZKU.sol Deployment

The screenshot displays the Remix IDE interface for the HelloWorld_ZKU.sol contract. The left sidebar shows the 'DEPLOY & RUN TRANSACTIONS' panel with a list of deployed contracts. The main editor shows the Solidity code for HelloWorld_ZKU.sol, which includes a constructor, a storage variable, and two public functions: storeNumber and retrieveNumber. The bottom panel shows the transaction history and the execution of the storeNumber function, which successfully stores the value 12345.

```
1 // SPDX-License-Identifier: MIT
2
3 //written by SHREYAS VIVEK shreyasvivek01@gmail.com
4
5 //solidity compiler version specification
6 pragma solidity >=0.7.0 <0.9.0;
7
8 //name and creation of contract named HelloWorld
9 contract HelloWorld {
10     //storeNum contract variable stores the number to be stored
11     uint storedNum;
12
13     /// Method for storing new number
14     /// storeNumber function to store the number
15     function storeNumber(uint number) public {
16
17         // number that is to be stored is stored in the storeNum variable
18         storedNum = number;
19     }
20
21     /// Method for retrieving the stored number
22     function retrieveNumber() public view returns (uint) {
23         return storedNum;
24     }
25 }
26
```

Deployed Contracts:

- HELLOWORLD AT 0XD91...39138 (MI)

storeNumber 12345

retrieveNumber

Low level interactions

CALLDATA

Transact

Transaction history:

- HELLOWORLD AT 0XD91...39138 (MI)
- HELLOWORLD AT 0XD2A...FD005 (MI)
- HELLOWORLD AT 0XB27...07C2C (MI)
- HELLOWORLD AT 0X9DB...A5B92 (MI)

Transaction details:

- [vm] from: 0x583...eddC4 to: Storage.(constructor) value: 0 wei data: 0x608...d0033 logs: 0 hash: 0x55c...22c02
- transact to HelloWorld.storeNumber pending ...
- [vm] from: 0x583...eddC4 to: Storage.(fallback) 0xd91...39138 value: 0 wei data: 0xb63...03039 logs: 0 hash: 0xa73...5743e
- call to HelloWorld.retrieveNumber
- CALL [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddc4 to: Storage.(fallback) data: 0xa90...9491b

AmendBallot_ZKU.sol Deployment

The screenshot displays the Remix IDE interface for the AmendBallot_ZKU.sol contract. The left sidebar shows the 'DEPLOY & RUN TRANSACTIONS' panel with a list of deployed contracts. The main editor shows the Solidity code for AmendBallot_ZKU.sol, which includes a constructor, a storage variable, and two public functions: delegate and giveRightToVote. The bottom panel shows the transaction history and the execution of the delegate function, which successfully delegates the vote to the specified address.

```
1 // SPDX-License-Identifier: GPL-3.0
2 //written by Shreyas Vivek shreyasvivek01@gmail.com
3 //solidity version specifier
4 pragma solidity >=0.7.0 <0.9.0;
5
6 //contract named ballot
7 contract Ballot {
8     //voter structure having weight, voted, delegate, vote
9     struct Voter {
10         uint weight;
11         bool voted;
12         address delegate;
13         uint vote;
14     }
15     //proposal structure having name and voteCount
16     struct Proposal {
17         bytes32 name;
18         uint voteCount;
19     }
20
21     //public address havinf chairperson
22     address public chairperson;
23
24     ContractDefinition Ballot {
25         1 reference(s) ^ ic voters;
26     }
27 }
28
```

Deployed Contracts:

- BALLOT AT 0X579...5CE76 (MEMORY)

delegate address to

giveRightToVote address voter

vota 0

chairperson

endTime

proposals uint256

startTime

voters address

winnerName

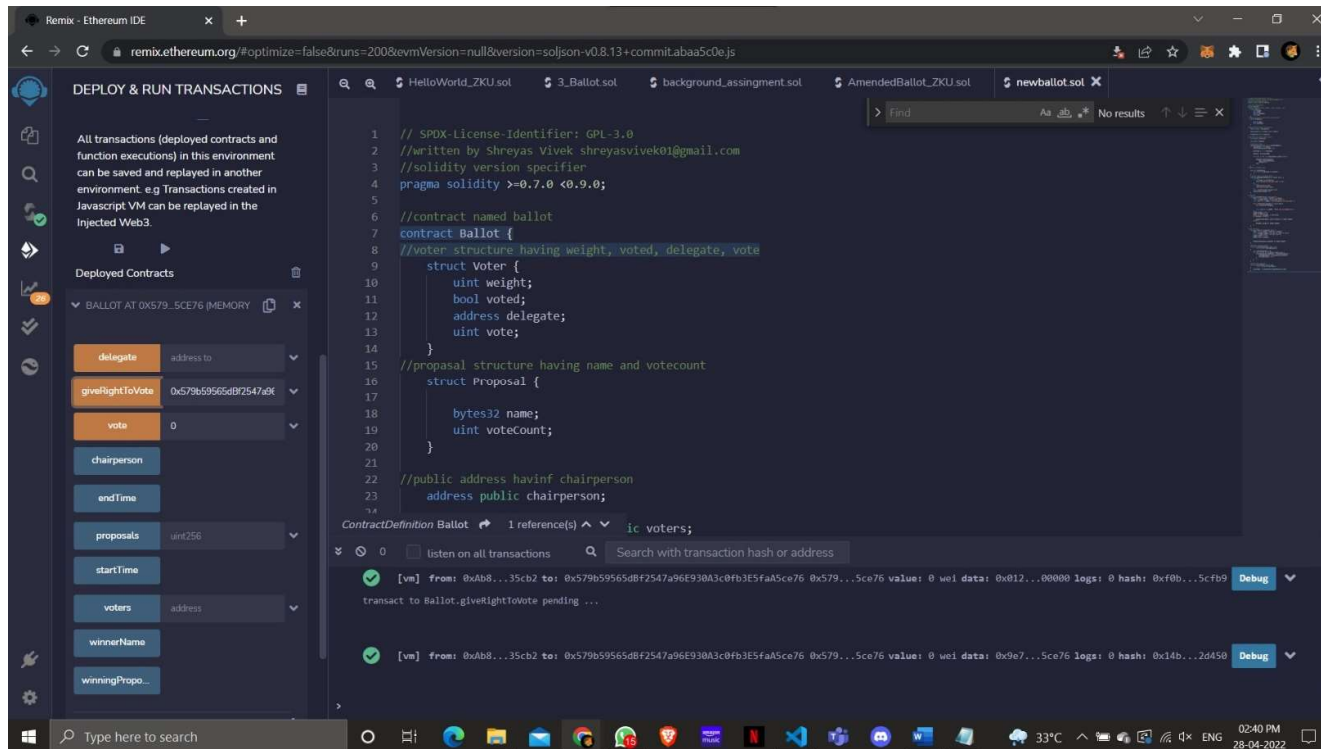
winningPropo...

Transaction history:

- BALLOT AT 0X579...5CE76 (MEMORY)

Transaction details:

- [vm] from: 0xAb8...35cb2 to: 0x579b59565dbf2547a96e930a3c0fb3e5fa5ce76 0x579...5ce76 value: 0 wei data: 0x012...00000 logs: 0 hash: 0x568...57432
- transact to Ballot.vote pending ...



GITHUB REPO LINK:

https://github.com/shreyas-vivek/ZKU_BACKGROUND_ASSIGNMENT