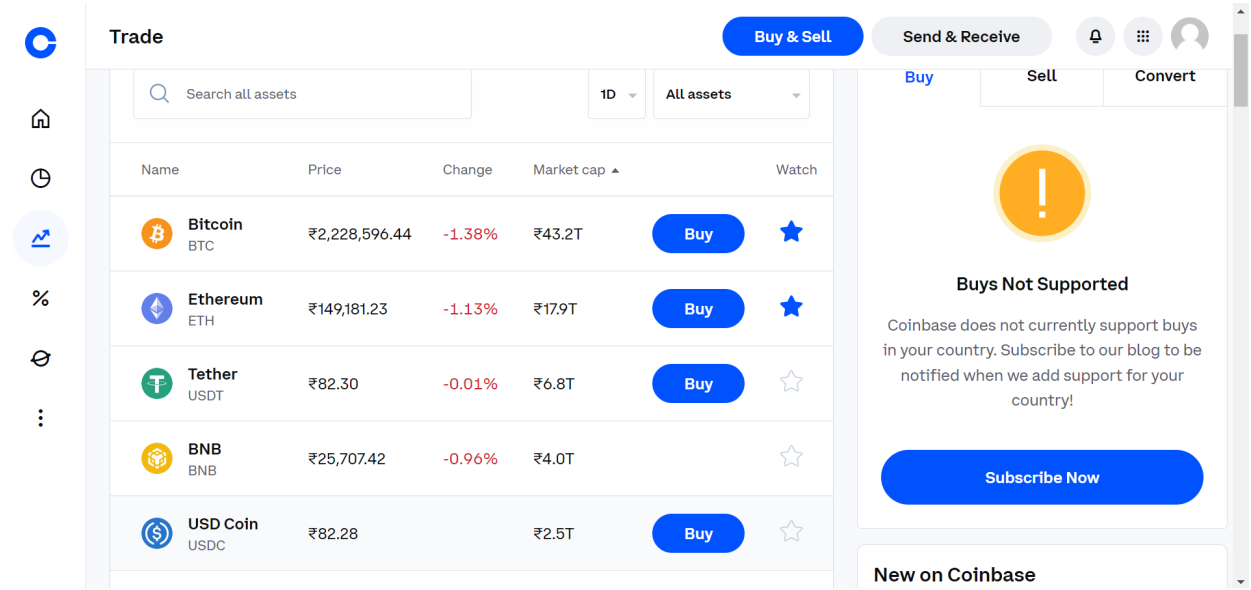
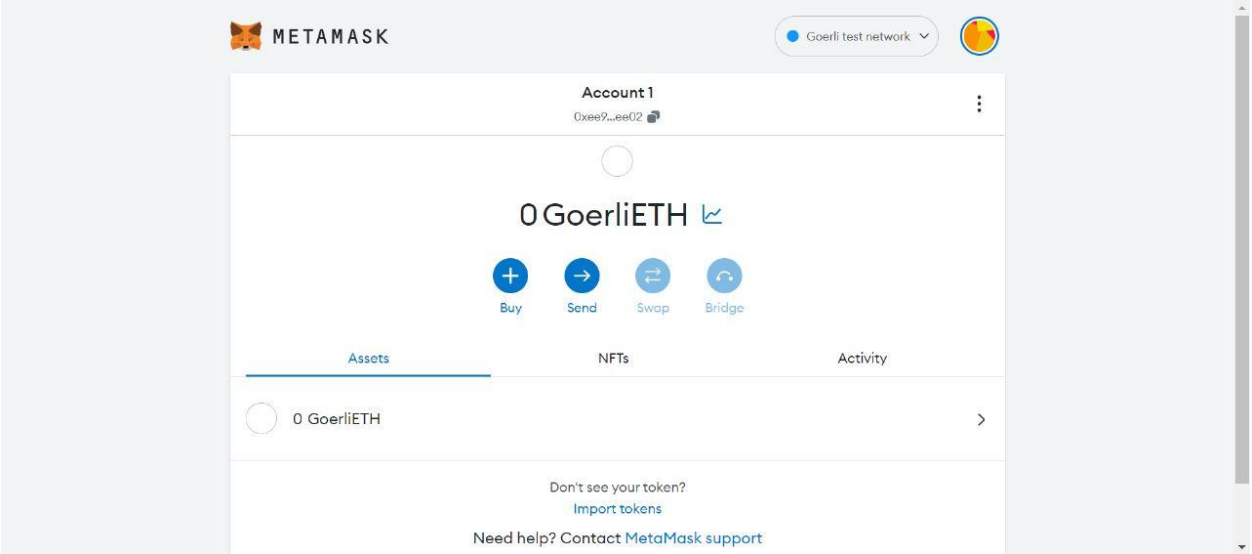
Assignment 1





[This is a Goerli **Testnet** transaction only]

Transaction Hash:	0x160ed03c98a7efdf3f73e09b644e492ca9fcb8c52e0e29b192c04316e7f47545
Status:	Success
Block:	8710179 1 Block Confirmation
Timestamp:	16 secs ago (Mar-24-2023 01:20:48 PM +UTC)
From:	0x168f6Dec26CBBB3749654e0e3Cc4Fc29314fdf6C
To:	0x191f60454C44E997D18F1e5C1eC0858b366e05A2
Value:	0.1 ETH ($0.00)
Transaction Fee:	0.010925872484247 ETH ($0.00)
Gas Price:	520.279642107 Gwei (0.000000520279642107 ETH)
More Details:	+ Click to show more

A transaction is a cryptographically signed instruction that changes the blockchain state. Block explorers track the details of all transactions in the network.

Assignment 2

1) Create a local Ethereum network using Hardhat or any other tool, build a smart contract that lets you send a 👋(wave) to your contract and keep track of the total # of waves. Compile it to run locally.
2) Connect to any Ethereum walleteg. Metamask. Deploy the contract with testnet. Connect wallet with your webapp. Call the deployed contract through your web app. Then store the wave messages from users in arrays using structs

Wave.sol (Smart Contract):-

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract WavePortal {
uint256 totalWaves;
function wave() public {
totalWaves += 1;
}
function getTotalWaves() public view returns (uint256) {
return totalWaves;
}
}
```

```
$ npx hardhat compile
Compiled 1 Solidity file successfully
```

WaveTest.js (Testing The Contract):-

```
const { expect } = require('chai');
const Wave=artifacts.require('Wave');
contract('Wave', async accounts => {
let wave;
before(async () => {
wave = await Wave.deployed();
});
it('should increase the wave count when a user sends a wave', async () => {
const waveCountBefore = await wave.getTotalWaves();
await wave.wave('Hi, this is user 1');
const waveCountAfter = await wave.getTotalWaves();
expect(waveCountAfter).to.equal(waveCountBefore + 1);
});
it('should store the sender and message for each wave', async () => {
const message1 = 'Hi, this is user 1';
await wave.wave(message1);
const wave1 = await wave.getWave(0);
expect(wave1.sender).to.equal(accounts[0]);
```

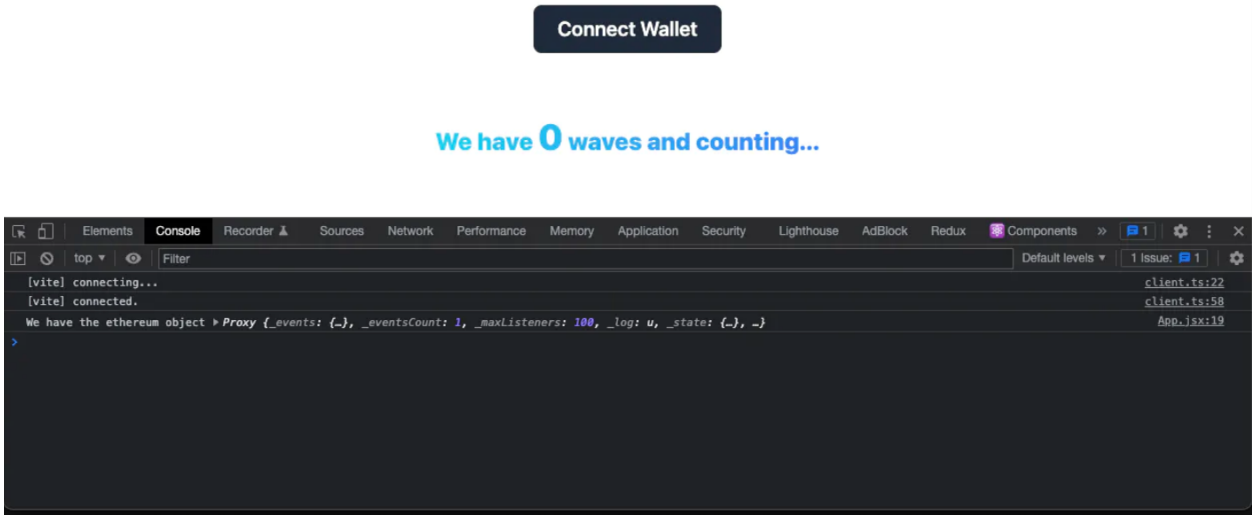
```
expect(wave1.message).to.equal(message1);});
});
```

Deploy.js (Deploying Code to Ethereum):-

```
const hre=require("hardhat");
async function main() {
const Wave = await hre.ethers.getContractFactory("Wave");
const wave = await Wave.deploy();
console.log("Wave deployed to:", wave.address);
}
main()
.then(() => process.exit(0))
.catch(error => {
console.error(error);
process.exit(1);
});
```

Output (Count Of The Waves On Testing Network):

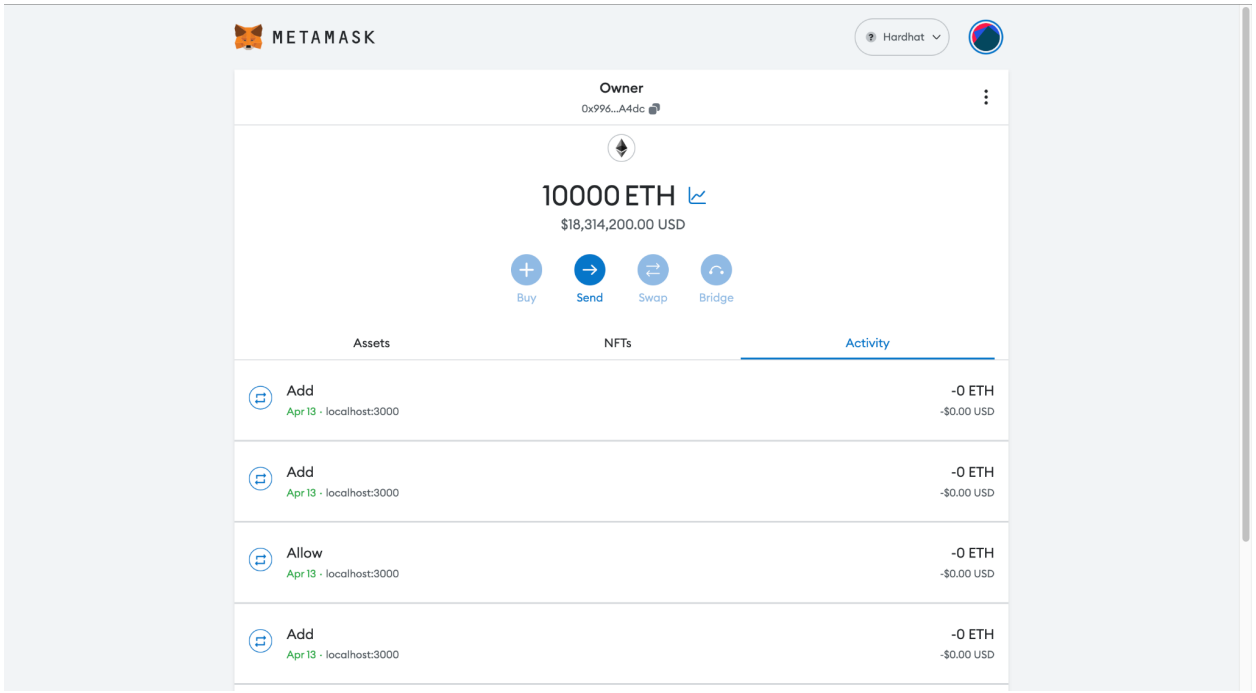
```
> await wave.getTotalWaves()
0

> await wave.wave()

> await wave.getTotalWaves()
1

> await wave.wave()

> await wave.getTotalWaves()
2

> await wave.wave()

> await wave.getTotalWaves()
3
```



Web Output:
BLOCK ADDRESS(Accounts Connected With the Private Keys) FOR STORING THE WAVES:
$ npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/
Accounts
========
WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.
Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
Private Key:
0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80
Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key:
0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d
Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key:
0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a
Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)
Private Key:
0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6
Account #4: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 (10000 ETH)
Private Key:
0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a
Account #5: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc (10000 ETH)
Private Key:
0x8b3a350cf5c34c9194ca85829a2df0ec3153be0318b5e2d3348e872092edffba
Account #6: 0x976EA74026E726554dB657fA54763abd0C3a0aa9 (10000 ETH)
Private Key:
0x92db14e403b83dfe3df233f83dfa3a0d7096f21ca9b0d6d6b8d88b2b4ec1564e

Account #7: 0x14dC79964da2C08b23698B3D3cc7Ca32193d9955 (10000 ETH)
Private Key:
0x4bbbf85ce3377467afe5d46f804f221813b2bb87f24d81f60f1fcdbf7cbf4356
Account #8: 0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f (10000 ETH)
Private Key:
0xdbda1821b80551c9d65939329250298aa3472ba22feea921c0cf5d620ea67b97
Account #9: 0xa0Ee7A142d267C1f36714E4a8F75612F20a79720 (10000 ETH)
Private Key:
0x2a871d0798f97d79848a013d4936a73bf4cc922c825d33c1cf7073dff6d409c6
Account #10: 0xBcd4042DE499D14e55001CcbB24a551F3b954096 (10000 ETH)
Private Key:
0xf214f2b2cd398c806f84e317254e0f0b801d0643303237d97a22a48e01628897
Account #11: 0x71bE63f3384f5fb98995898A86B02Fb2426c5788 (10000 ETH)
Private Key:
0x701b615bbdfb9de65240bc28bd21bbc0d996645a3dd57e7b12bc2bdf6f192c82
Account #12: 0xFABB0ac9d68B0B445fB7357272Ff202C5651694a (10000 ETH)
Private Key:
0xa267530f49f8280200edf313ee7af6b827f2a8bce2897751d06a843f644967b1
Account #13: 0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec (10000 ETH)
Private Key:
0x47c99abed3324a2707c28affff1267e45918ec8c3f20b8aa892e8b065d2942dd
WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.
eth_blockNumber
eth_getBlockByNumber
eth_blockNumber (2)
net_version
eth_getBlockByNumber
eth_blockNumber
eth_getBalance (8)
eth_blockNumber
eth_gasPrice
eth_blockNumber (30)

METAMASK STORING THE ADDRESSES AND WAVES



Assignment 3

Prepare your build system and Building Bitcoin Core.
a. Write a Hello World smart contract in a higher programming language (Solidity).
b. Solidity example using arrays and functions

HelloWorld.sol

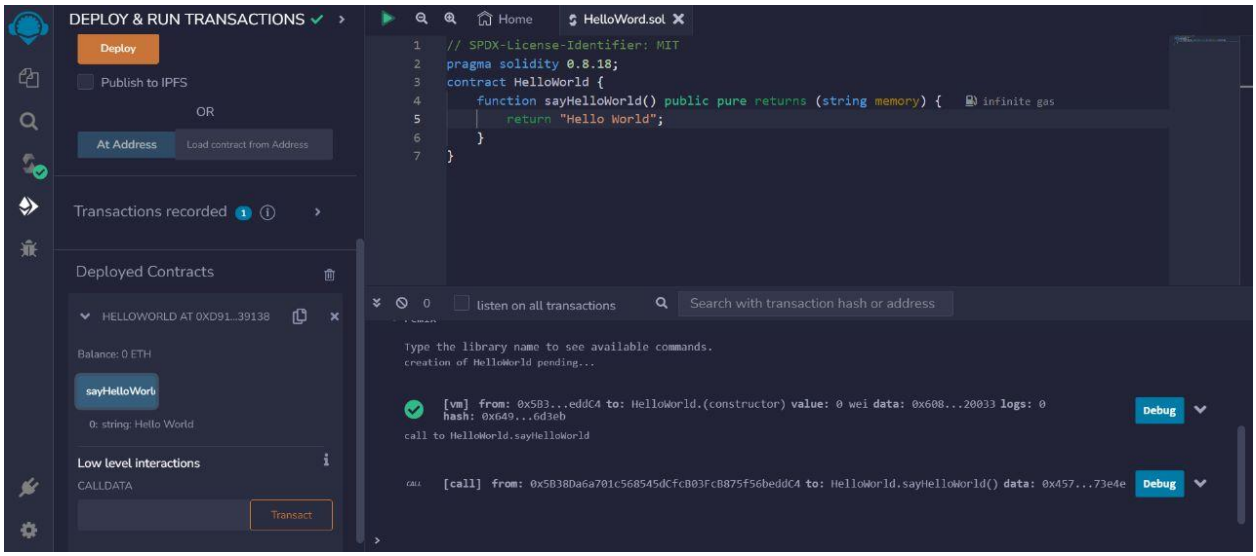
```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18;
contract HelloWorld {
    function sayHelloWorld() public pure returns (string memory) {
        return "Hello World";
    }
}
```

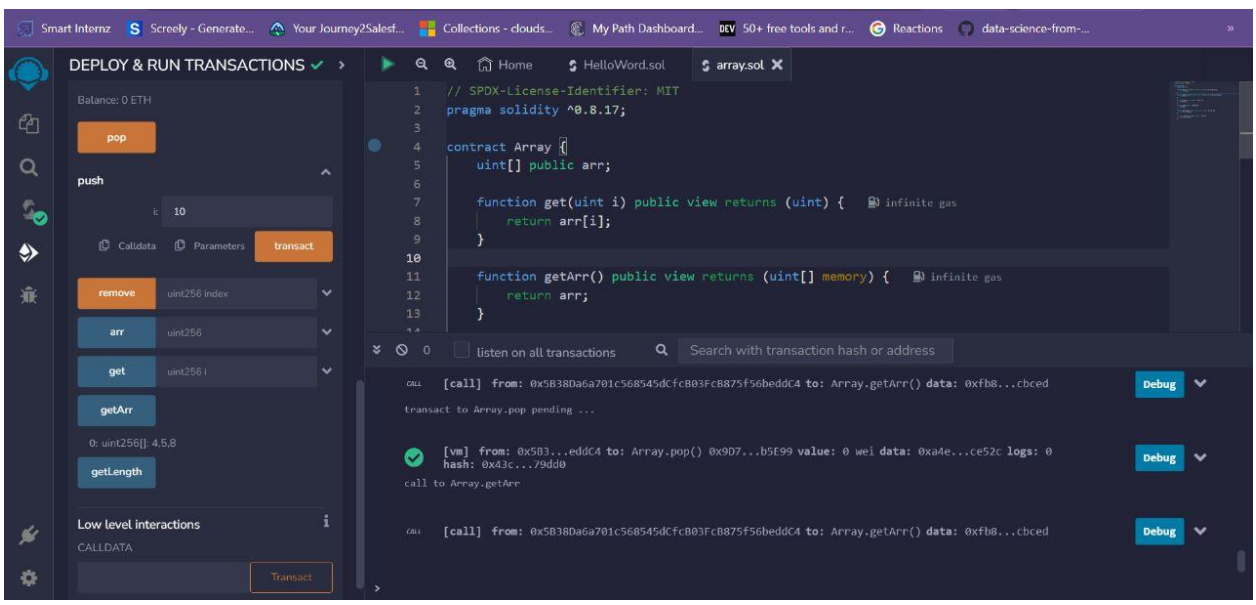
Array.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Array {
    uint[] public arr;

    function get(uint i) public view returns (uint) {
        return arr[i];
    }

    function getArr() public view returns (uint[] memory) {
        return arr;
    }

    function push(uint i) public {
        arr.push(i);
    }

    function pop() public {
        arr.pop();
    }

    function getLength() public view returns (uint) {
        return arr.length;
    }

    function remove(uint index) public {
        delete arr[index];
    }
```

```
}
```

Outputs:


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 1
Deployed Contracts
HELLOWORLD AT 0XD91...39138
Balance: 0 ETH
sayHelloWorld
0: string: Hello World
Low level interactions
CALLDATA
Transact
Home
HelloWord.sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18;
contract HelloWorld {
function sayHelloWorld() public pure returns (string memory) {
return "Hello World";
}
}
infinite gas
listen on all transactions
Search with transaction hash or address
Type the library name to see available commands.
creation of HelloWorld pending...
[vm] from: 0x5B3...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...20033 logs: 0
hash: 0x649...6d3eb
call to HelloWorld.sayHelloWorld
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: HelloWorld.sayHelloWorld() data: 0x457...73e4e
Debug

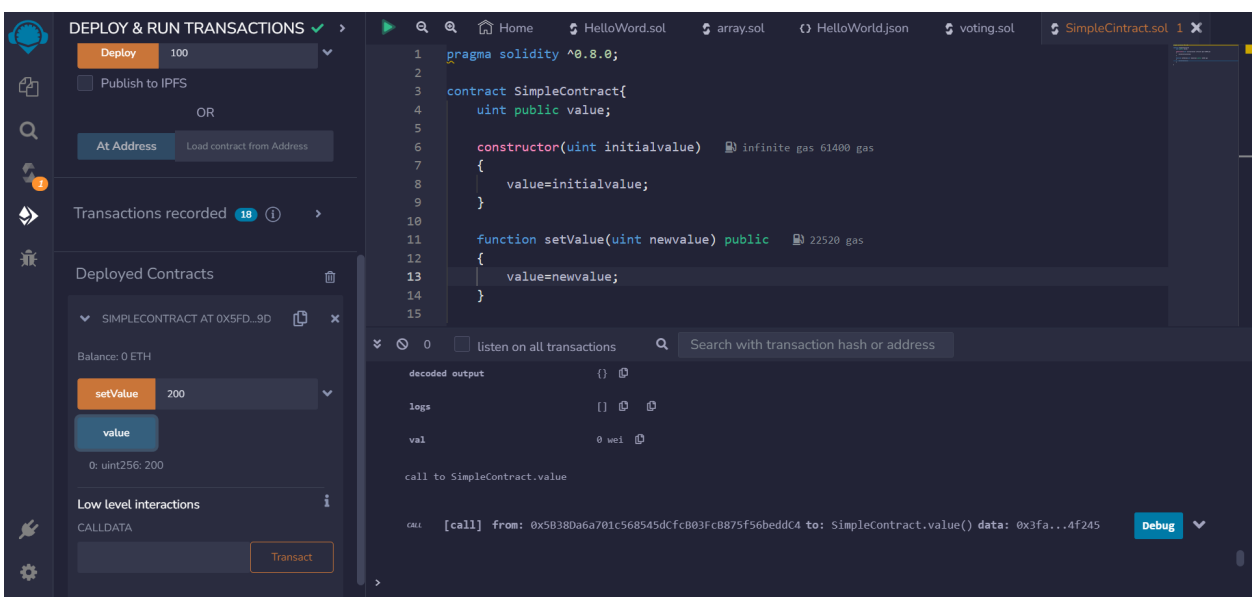

Smart Internz
Screely - Generate...
Your Journey2Salesf...
Collections - clouds...
My Path Dashboard...
50+ free tools and r...
Reactions
data-science-from-...
DEPLOY & RUN TRANSACTIONS
Balance: 0 ETH
pop
push
i: 10
Calldata
Parameters
transact
remove
uint256 index
arr
uint256
get
uint256 i
getArr
0: uint256[]: 4,5,8
getLength
Low level interactions
CALLDATA
Transact
Home
HelloWord.sol
array.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract Array {
uint[] public arr;
function get(uint i) public view returns (uint) {
return arr[i];
}
function getArr() public view returns (uint[] memory) {
return arr;
}
infinite gas
listen on all transactions
Search with transaction hash or address
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Array.getArr() data: 0xfb8...cbced
transact to Array.pop pending ...
[vm] from: 0x5B3...eddC4 to: Array.pop() 0x9D7...b5E99 value: 0 wei data: 0xa4e...ce52c logs: 0
hash: 0x43c...79dd0
call to Array.getArr
Debug

Assignment 4

Code-

```
pragma solidity ^0.8.0;

contract SimpleContract{
    uint public value;

    constructor(uint initialvalue)
    {
        value=initialvalue;
    }

    function setValue(uint newvalue) public
    {
        value=newvalue;
    }

}
```

Output -



Assignment 5

Code-

```
pragma solidity 0.8.18;
contract CollegePollingSystem {

    // Define a struct to represent a poll
    struct Poll {
        string question;
        string[] options;
        mapping (string => uint) votes;
        bool active;
    }

    // Define a hashmap to store the polls
    mapping (uint => Poll) public polls;
    uint public numPolls;

    // Define a struct to represent a voter
    struct Voter {
        bool voted;
        uint pollIndex;
    }

    // Define a hashmap to store the voters
    mapping (address => Voter) public voters;

    // Define an event to log poll creation
    event PollCreated(uint pollIndex, string question, string[] options);

    // Define an event to log vote submission
    event VoteSubmitted(address voter, uint pollIndex, string option);

    // Create a new poll
    function createPoll(string memory question, string[] memory options)
public {
        numPolls++;
        polls[numPolls].question = question;
        polls[numPolls].options = options;
```

```
        polls[numPolls].active = true;

        emit PollCreated(numPolls, question, options);
    }

    // Vote in a poll
    function vote(uint pollIndex, string memory option) public {
        // Check if poll exists and is active
        require(pollIndex <= numPolls && polls[pollIndex].active == true,
"Poll does not exist or is inactive");

        // Check if voter has already voted
        require(voters[msg.sender].voted == false, "Voter has already
voted");

        // Check if option is valid
        require(isOptionValid(pollIndex, option) == true, "Invalid
option");

        // Record vote
        polls[pollIndex].votes[option]++;
        voters[msg.sender].voted = true;
        voters[msg.sender].pollIndex = pollIndex;

        emit VoteSubmitted(msg.sender, pollIndex, option);
    }

    // Check if an option is valid
    function isOptionValid(uint pollIndex, string memory option) public
view returns (bool) {
        for (uint i = 0; i < polls[pollIndex].options.length; i++) {
            if (keccak256(bytes(polls[pollIndex].options[i])) ==
keccak256(bytes(option))) {
                return true;
            }
        }
        return false;
    }

    // Get the number of votes for an option
```

```
    function getVotes(uint pollIndex, string memory option) public view
returns (uint) {
        require(pollIndex <= numPolls && polls[pollIndex].active == false,
"Poll does not exist or is active");
        require(isOptionValid(pollIndex, option) == true, "Invalid
option");

        return polls[pollIndex].votes[option];
    }
}
```

Output -