Homework 14

Interacting with Compound

Use the gitpod workspace https://gitpod.io/#https://github.com/ExtropyIO/Academy (https://gitpod.io/#https://github.com/ExtropyIO/Academy)

- Navigate to the DeFiHardhat directory , there is the beginnings of a project
- Run npm i to install the dependencies
- In a terminal start ganache that forks the mainnet
- npx ganache-cli -f <Your INFURA URL> --unlock 0x503828976D22510aad0201ac7EC88293211D23Da -p 8545

Contract

you can use the following interfaces as cut down versions of Open Zeppelin contracts

```
interface Erc20 {
    function approve(address, uint256) external returns (bool);

    function transfer(address, uint256) external returns (bool);
}



interface CErc20 {
    function mint(uint256) external returns (uint256);

    function exchangeRateCurrent() external returns (uint256);

    function supplyRatePerBlock() external returns (uint256);

    function redeem(uint) external returns (uint);

    function redeemUnderlying(uint) external returns (uint);
}
```

- Add these to your compilation file
- Add variables to hold the addresses of DAI and cDAI on the mainnet as the above types and set the correct addresses for these in the constructor.

DAI = '0x6b175474e89094c44da98b954eedeac495271d0f’
cDAI = ‘0x5d3a536E4D6DbD6114cc1Ead35777bAB948E3643’

- Then add a function to your Defi contract called ***addToCompound***
- This function should have 1 parameter of type uint256 for the amount of DAI to add.
- The function should then approve the cDAI contract for the input amount
- The addition to Compound occurs by calling

```
cToken.mint(amount);
```

  where cToken is instance of the cDAI contract

Unit Test

Add a unit test, which sends an amount of DAI to the Defi contract, then calls addToCompound to add that to Compound.

Afterwards test the balance of cDAI of your contract

Using Oracles

Modify your contract as follows

- Import the chainlink contacts

```
import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
```

- Add a constant

```
ETHPriceContract = 0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419;
```

- Declare a variable of type `AggregatorV3Interface`
- In the constructor create this using the ETHPriceContract address.
- Create a function to get the ETH / USD price, this function should call the function `latestRoundData` on the AggregatorV3Interface.

  You can find details of that function here (https://github.com/smartcontractkit/chainlink/blob/master/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol)

Unit Test

Now create a unit test to test your new function

Does the oracle price match what you see in Uniswap ?
Can you do a large enough trade to move the Uniswap price ?

If you want to try other token prices, the Ethereum mainnet data feeds are defined here data feeds (https://docs.chain.link/docs/ethereum-addresses/)