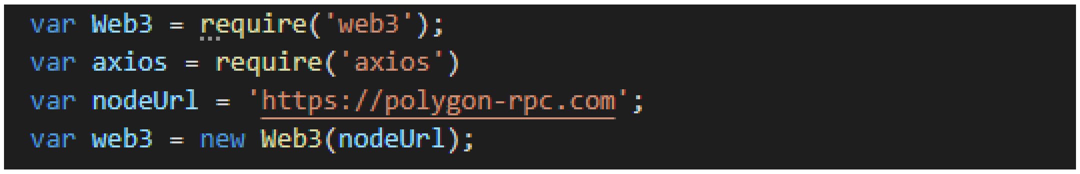
Diffuse Funds DeFi Task

Priyan Rai

Requirements: Node.js or any JavaScript runtime, web3 library, axios library (used for fetching urls). It can be run by simply putting the script file in a folder and calling the node command on it after the required libraries are installed.

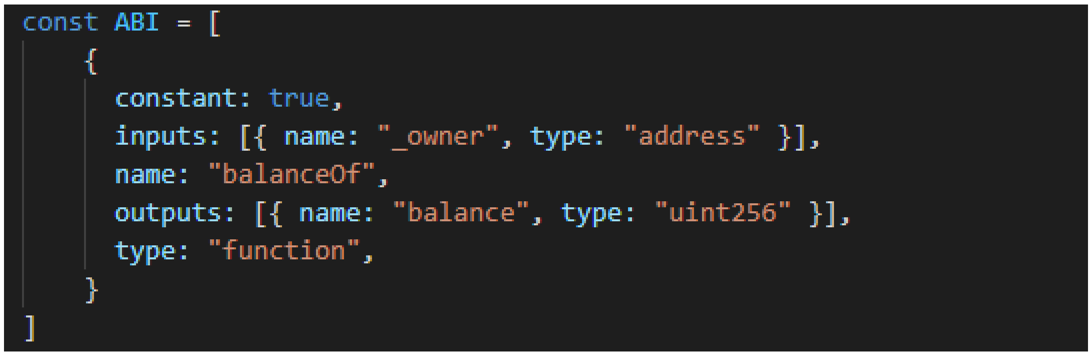
Part 1: Acquiring Token Balances in wallet.

- For this task, I used JavaScript, and initialized web3 to connect to the polygon main net as the provider.



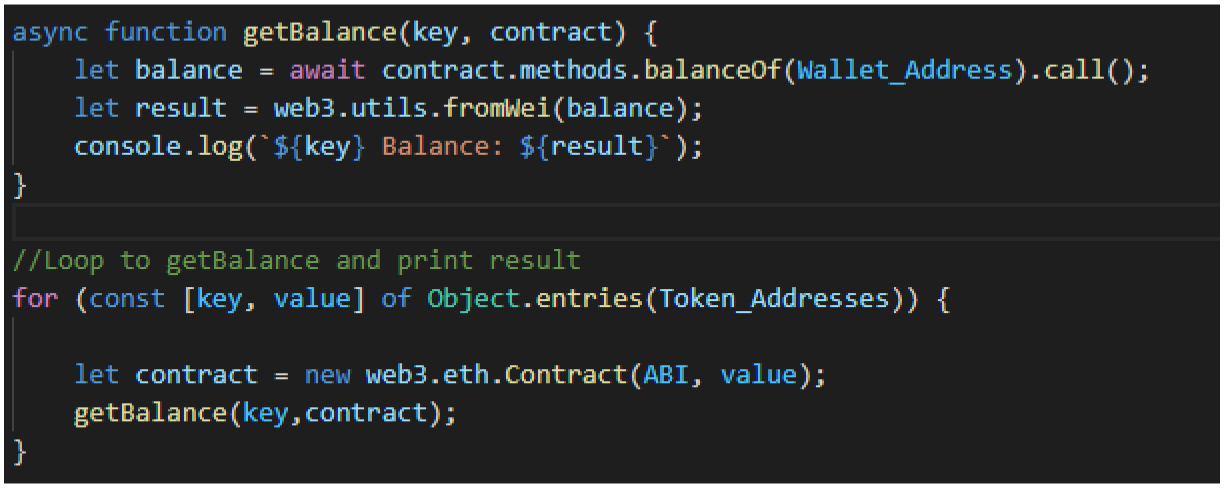
```
var Web3 = require('web3');
var axios = require('axios')
var nodeUrl = 'https://polygon-rpc.com';
var web3 = new Web3(nodeUrl);
```

- Then I made a basic Application Binary Interface to communicate with the wallet contracts.



```
const ABI = [
    {
      constant: true,
      inputs: [{ name: "_owner", type: "address" }],
      name: "balanceOf",
      outputs: [{ name: "balance", type: "uint256" }],
      type: "function",
    }
]
```

- Using the basic ABI, it was pretty straightforward to get the wallets token balances using web3.contract's methods.



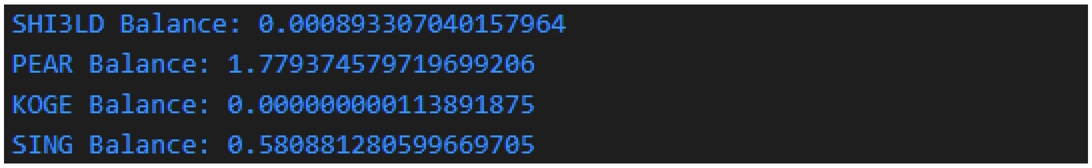
```
async function getBalance(key, contract) {
    let balance = await contract.methods.balanceOf(Wallet_Address).call();
    let result = web3.utils.fromWei(balance);
    console.log(`${key} Balance: ${result}`);
}

//Loop to getBalance and print result
for (const [key, value] of Object.entries(Token_Addresses)) {

    let contract = new web3.eth.Contract(ABI, value);
    getBalance(key,contract);
}
```

The balances at time of running the code were as follows:

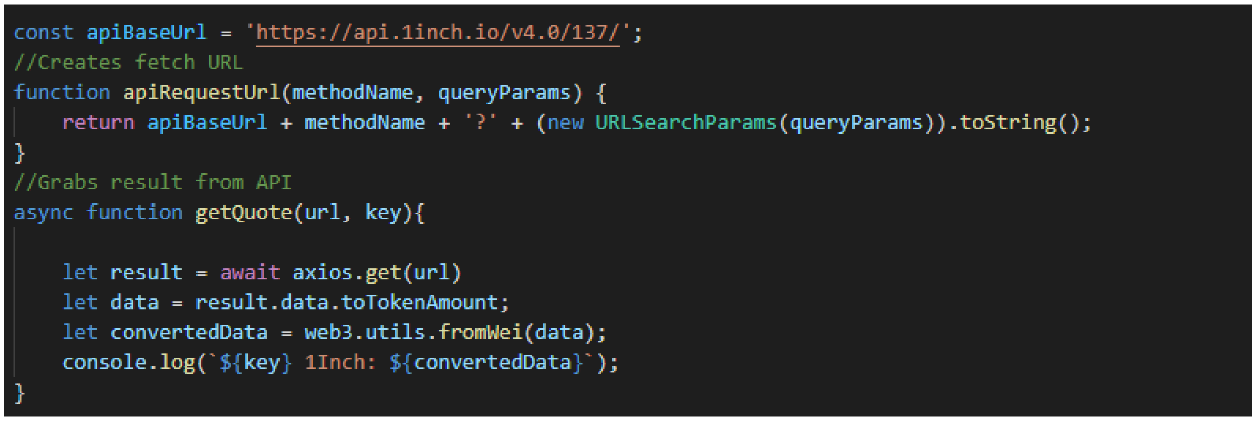
SH3ILD	0.000893307040157964
KOGE	0.000000000113891875
PEAR	1.779374579719699206
SING	0.580881280599669705



```
SHI3LD Balance: 0.000893307040157964
PEAR Balance: 1.779374579719699206
KOGE Balance: 0.000000000113891875
SING Balance: 0.580881280599669705
```

Part 2: Checking exchange rates at various DEXs.

- First was 1Inch. After diving through their API, I found a quote method that returns the amount received in the exchanged token as well as the route that could be taken to provide the best exchange rate.
- Calling their API was pretty simple as it needed a custom fetch URL for every request which was built by the following functions.



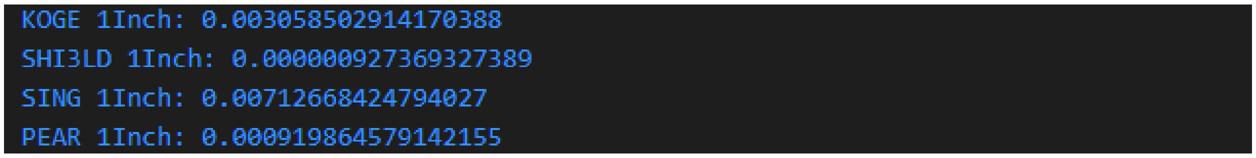
```
const apiBaseUrl = 'https://api.1inch.io/v4.0/137/';
//Creates fetch URL
function apiRequestUrl(methodName, queryParams) {
    return apiBaseUrl + methodName + '?' + (new URLSearchParams(queryParams)).toString();
}
//Grabs result from API
async function getQuote(url, key){

    let result = await axios.get(url)
    let data = result.data.toTokenAmount;
    let convertedData = web3.utils.fromWei(data);
    console.log(`${key} 1Inch: ${convertedData}`);
}
```

- Since I had the token addresses stored in an object, It was again looping through the addresses and getting the exchange rate through 1Inch. The table below shows the total amount returned in DAI.

The Exchange rate for 1Inch was as follows:

SH3ILD	0.000009273693273890
KOGE	0.003058502914170388
PEAR	0.000919864579142155
SING	0.00712668424794027
TOTAL	**0.011105979110580202**



```
KOGE 1Inch: 0.003058502914170388
SHI3LD 1Inch: 0.000000927369327389
SING 1Inch: 0.00712668424794027
PEAR 1Inch: 0.000919864579142155
```

Using Smart Contracts for ApeSwap and CafeSwap

- My approach was to use the factory contracts to find token pairs first.
- Then check liquidity for both pools
- Using the getAmountOut method from the Router contracts we can find the exchange value.

Unfortunately, I wasn’t able to progress further. I failed to find any token pairs using the Factory Contracts, I also tried to see the route of the exchange if it was possible to move the tokens through other pools but found no success.