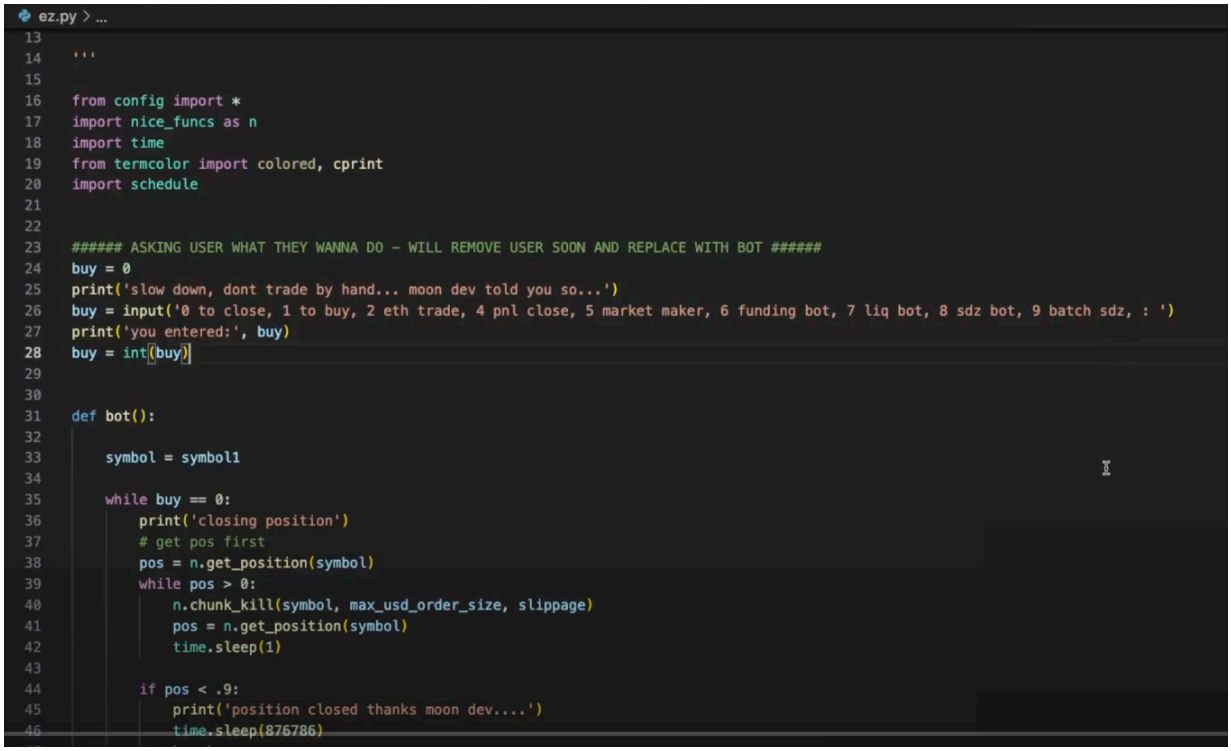
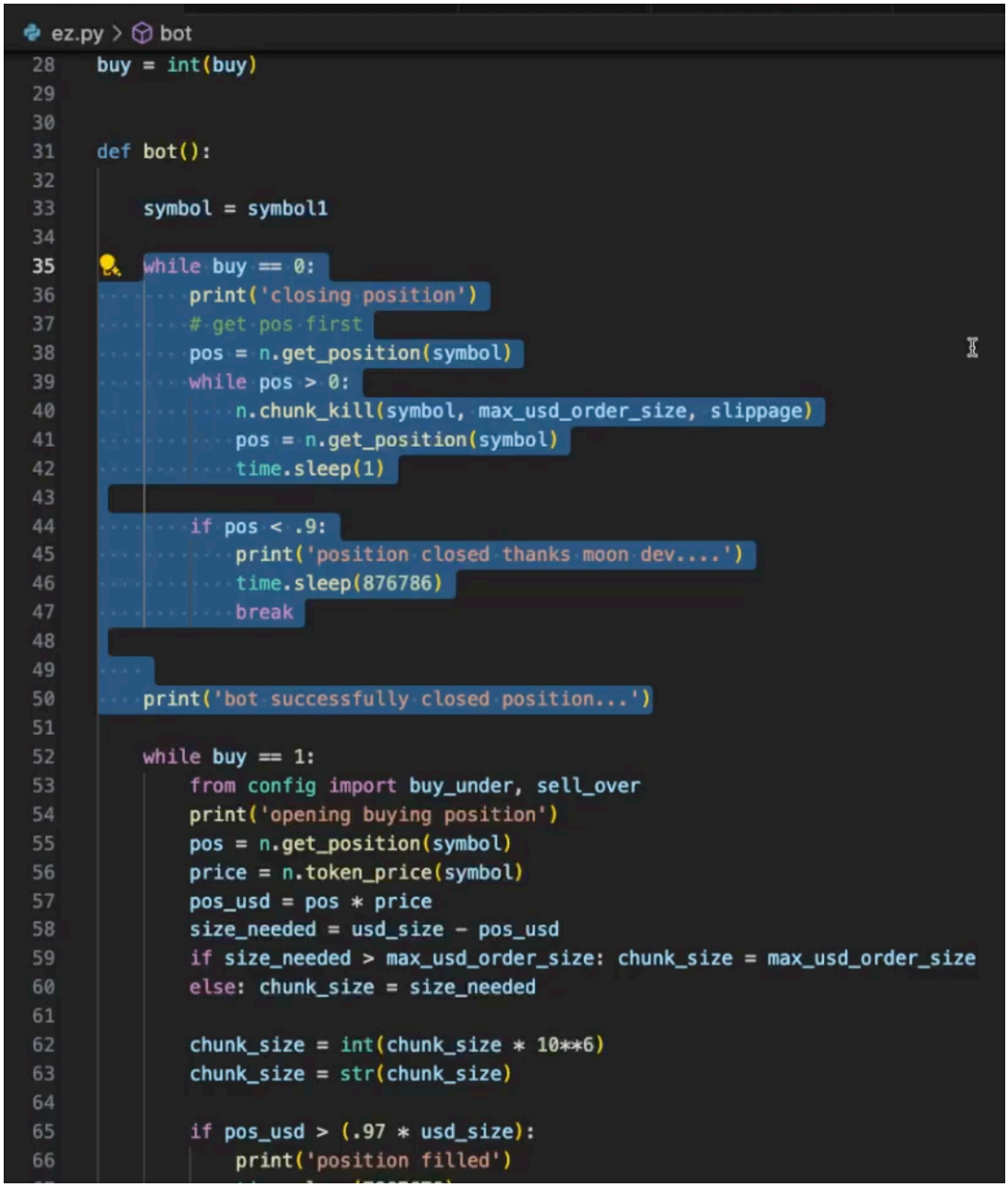
25_solana_buy_sell

Def #ez.py



```
ez.py > ...
13
14  '''
15
16  from config import *
17  import nice_funcs as n
18  import time
19  from termcolor import colored, cprint
20  import schedule
21
22
23  ###### ASKING USER WHAT THEY WANNA DO - WILL REMOVE USER SOON AND REPLACE WITH BOT ######
24  buy = 0
25  print('slow down, dont trade by hand... moon dev told you so...')
26  buy = input('0 to close, 1 to buy, 2 eth trade, 4 pnl close, 5 market maker, 6 funding bot, 7 liq bot, 8 sdz bot, 9 batch sdz, : ')
27  print('you entered:', buy)
28  buy = int(buy)
29
30
31  def bot():
32
33      symbol = symbol1
34
35      while buy == 0:
36          print('closing position')
37          # get pos first
38          pos = n.get_position(symbol)
39          while pos > 0:
40              n.chunk_kill(symbol, max_usd_order_size, slippage)
41              pos = n.get_position(symbol)
42              time.sleep(1)
43
44          if pos < .9:
45              print('position closed thanks moon dev....')
46              time.sleep(876786)
```


ez.py > bot
buy = int(buy)
def bot():
symbol = symbol1
while buy == 0:
print('closing position')
# get pos first
pos = n.get_position(symbol)
while pos > 0:
n.chunk_kill(symbol, max_usd_order_size, slippage)
pos = n.get_position(symbol)
time.sleep(1)
if pos < .9:
print('position closed thanks moon dev....')
time.sleep(876786)
break
print('bot successfully closed position...')
while buy == 1:
from config import buy_under, sell_over
print('opening buying position')
pos = n.get_position(symbol)
price = n.token_price(symbol)
pos_usd = pos * price
size_needed = usd_size - pos_usd
if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
else: chunk_size = size_needed
chunk_size = int(chunk_size * 10**6)
chunk_size = str(chunk_size)
if pos_usd > (.97 * usd_size):
print('position filled')

```python
def bot():
    print('bot successfully closed position...')

    while buy == 1:
        from config import buy_under, sell_over
        print('opening buying position')
        pos = n.get_position(symbol)
        price = n.token_price(symbol)
        pos_usd = pos * price
        size_needed = usd_size - pos_usd
        if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
        else: chunk_size = size_needed

        chunk_size = int(chunk_size * 10**6)
        chunk_size = str(chunk_size)

        if pos_usd > (.97 * usd_size):
            print('position filled')
            time.sleep(7867678)

        while pos_usd < (.97 * usd_size):

            print(f'position: {round(pos,2)} price: {round(price,8)} buy_under: {buy_under} pos_usd: ${round(pos_usd,2)}')

            try:

                for i in range(orders_per_open):
                    n.market_buy(symbol, chunk_size, slippage)
                    # cprint green background black text
                    cprint(f'chunk buy submitted of {symbol[-4:]} sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
                    time.sleep(1)

                time.sleep(tx_sleep)

                pos = n.get_position(symbol)
                price = n.token_price(symbol)
                pos_usd = pos * price
                size_needed = usd_size - pos_usd
                if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
                else: chunk_size = size_needed
                chunk_size = int(chunk_size * 10**6)
                chunk_size = str(chunk_size)

            except:

                try:
                    cprint(f'trying again to make the order in 30 seconds.....', 'light_blue', 'on_light_magenta')
                    time.sleep(30)
                    for i in range(orders_per_open):
                        n.market_buy(symbol, chunk_size, slippage)
                        # cprint green background black text
                        cprint(f'chunk buy submitted of {symbol[-4:]} sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
                        time.sleep(1)

                    time.sleep(tx_sleep)
                    pos = n.get_position(symbol)
                    price = n.token_price(symbol)
                    pos_usd = pos * price
                    size_needed = usd_size - pos_usd
                    if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
                    else: chunk_size = size_needed
                    chunk_size = int(chunk_size * 10**6)
```

```
92          except:
93
94              try:
95                  cprint(f'trying again to make the order in 30 seconds.....', 'light_blue', 'on_light_magenta')
96                  time.sleep(30)
97                  for i in range(orders_per_open):
98                      n.market_buy(symbol, chunk_size, slippage)
99                      # cprint green background black text
100                     cprint(f'chunk buy submitted of {symbol[-4:]} sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
101                     time.sleep(1)
102
103                 time.sleep(tx_sleep)
104                 pos = n.get_position(symbol)
105                 price = n.token_price(symbol)
106                 pos_usd = pos * price
107                 size_needed = usd_size - pos_usd
108                 if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
109                 else: chunk_size = size_needed
110                 chunk_size = int(chunk_size * 10**6)
111                 chunk_size = str(chunk_size)
112
113
114             except:
115                 cprint(f'Final Error in the buy, restart needed', 'white', 'on_red')
116                 time.sleep(10)
117                 break
118
119         pos = n.get_position(symbol)
120         price = n.token_price(symbol)
121         pos_usd = pos * price
122         size_needed = usd_size - pos_usd
123         if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
124         else: chunk_size = size_needed
125         chunk_size = int(chunk_size * 10**6)
126         chunk_size = str(chunk_size)
127
128
```

```
129         # cprint white on greeen
130         cprint(f'position filled of {symbol[-4:]} total: ${pos_usd}', 'white', 'on_green')
131         break
132
133
134     while buy == 2:
135
136         print('starting eth trade.. ')
137
138         eth_df = n.fetch_candle_data_with_smas('ETH', '1d', 200, sma_windows=[20, 41])
```

nicefuncs.py
#def_market_buy

```python
def market_buy(token, amount, slippage):

    import requests
    import sys
    import json
    import base64
    from solders.keypair import Keypair
    from solders.transaction import VersionedTransaction
    from solana.rpc.api import Client
    from solana.rpc.types import TxOpts
    import time

    # Import the secret key and Ankr endpoint from dontshare
    import dontshare.dontshare as d

    KEY = Keypair.from_base58_string(d.sol_key)
    SLIPPAGE = slippage  # 5000 is 50%, 500 is 5% and 50 is .5%
    PRIORITY_FEE = 5000  # Hardcoded prioritization fee (adjust as needed)
    QUOTE_TOKEN = "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"  # usdc

    # Use Ankr's RPC endpoint
    http_client = Client(d.ankr_key)
    #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")

    quote_url = f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}'
    swap_url = 'https://quote-api.jup.ag/v6/swap'

    while True:
        try:
            quote = requests.get(quote_url).json()
            # print(quote)

            txRes = requests.post(swap_url,
                                  headers={"Content-Type": "application/json"},
                                  data=json.dumps({
                                      "quoteResponse": quote,
                                      "userPublicKey": str(KEY.pubkey()),
                                      "prioritizationFeeLamports": PRIORITY_FEE  # Hardcoded fee
                                  })).json()
```

```python
    # Use Ankr's RPC endpoint
    #http_client = Client(d.ankr_key) # QUICKNODE
    http_client = Client("https://api.mainnet-beta.solana.com/")

    quote_url = f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}'
    swap_url = 'https://quote-api.jup.ag/v6/swap'
```

```
    while True:
        try:
            quote = requests.get(quote_url).json()
            # print(quote)

            txRes = requests.post(swap_url,
                                  headers={"Content-Type": "application/json"},
                                  data=json.dumps({
                                      "quoteResponse": quote,
                                      "userPublicKey": str(KEY.pubkey()),
                                      "prioritizationFeeLamports": PRIORITY_FEE  # Hardcoded fee
                                  })).json()
            # print(txRes)
            swapTx = base64.b64decode(txRes['swapTransaction'])
            # print(swapTx)
            tx1 = VersionedTransaction.from_bytes(swapTx)
            tx = VersionedTransaction(tx1.message, [KEY])
            txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
            print(f"https://solscan.io/tx/{str(txId)}")
            break
        except requests.exceptions.RequestException as e:
            print(f"Request failed: {e}")
            time.sleep(5)  # Wait for 5 seconds before retrying
        except Exception as e:
            print(f"An error occurred: {e}")
            time.sleep(5)  # Wait for 5 seconds before retrying


def market_buy_juprpc(token, amount, slippage):

    import requests
    import sys
    import json
    import base64
```

#Def market_sell

```
def market_sell(QUOTE_TOKEN, amount, slippage):

    import requests
    import sys
    import json
    import base64
    from solders.keypair import Keypair
    from solders.transaction import VersionedTransaction
    from solana.rpc.api import Client
    from solana.rpc.types import TxOpts
    import dontshare.dontshare as d

    print('inside market sell')

    KEY = Keypair.from_base58_string(d.sol_key)
    SLIPPAGE = slippage # 5000 is 50%, 500 is 5% and 50 is .5%
    print(f'slippage is {SLIPPAGE}')
    #PRIORITY_FEE = PRIORITY_FEE
    #QUOTE_TOKEN = "So11111111111111111111111111111111111111112"
    # QUOTE_TOKEN = "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v" # usdc

    # token would be usdc for sell orders cause we are selling
    token =  "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"

    http_client = Client(d.ankr_key)
    #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")


    quote = requests.get(f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}').json()
    #print(quote)
    txRes = requests.post('https://quote-api.jup.ag/v6/swap',
                          headers={"Content-Type": "application/json"},
                          data=json.dumps({
                              "quoteResponse": quote,
                              "userPublicKey": str(KEY.pubkey()),
```

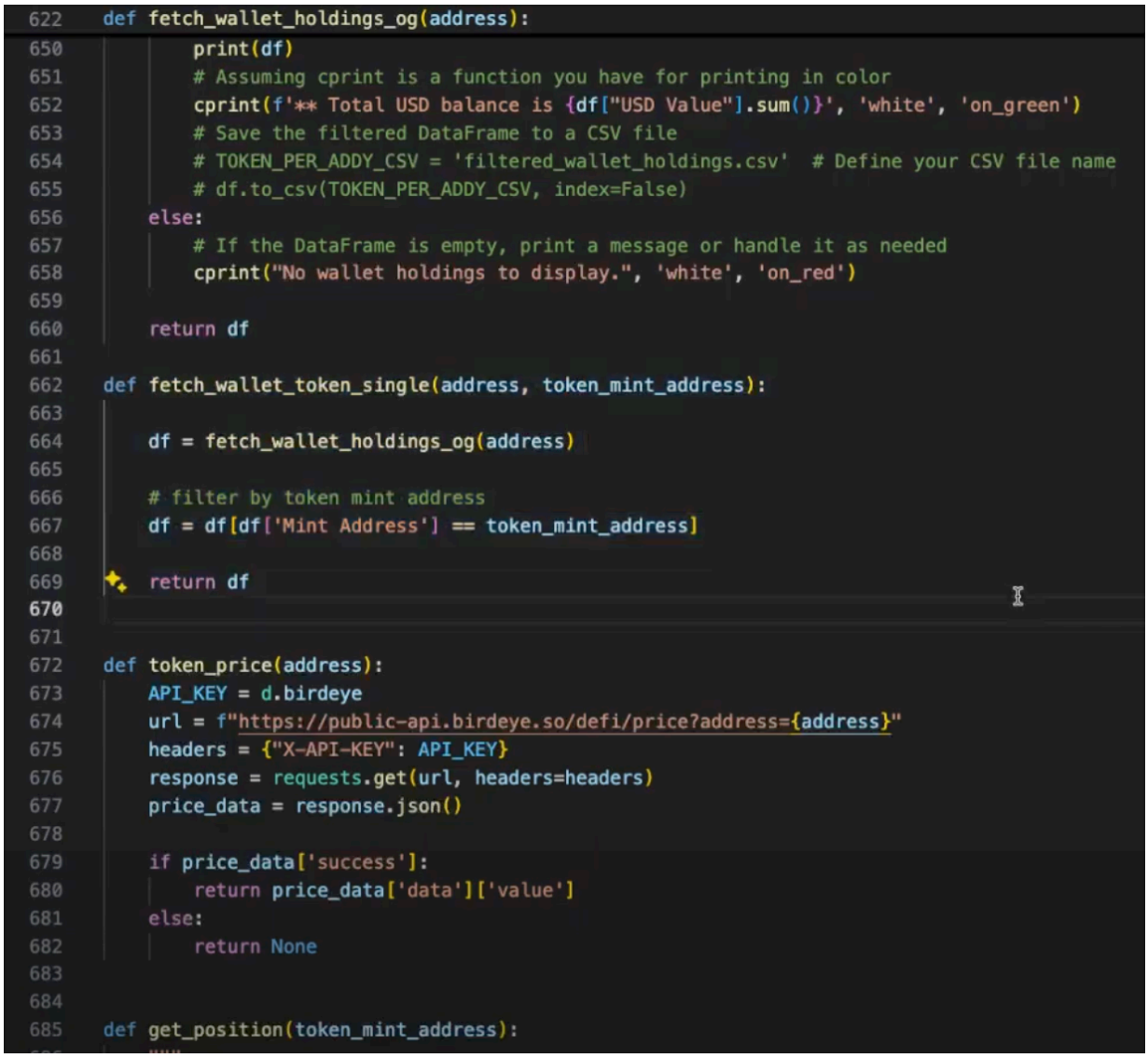
```
                                "userPublicKey": str(KEY.pubkey()),
                                "prioritizationFeeLamports": PRIORITY_FEE  # or replace 'auto' with your specific lamport value
                            })).json()
    #txRes = requests.post('https://quote-api.jup.ag/v6/swap',headers={"Content-Type": "application/json"}, data=json.dumps({"quoteResponse": quote, "userPublicKey
    #print(txRes)
    swapTx = base64.b64decode(txRes['swapTransaction'])
    #print(swapTx)
    tx1 = VersionedTransaction.from_bytes(swapTx)
    #print(tx1)
    tx = VersionedTransaction(tx1.message, [KEY])
    #print(tx)
    txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
    print(f"https://solscan.io/tx/{str(txId)}")
```

\#Def Get_poistions from nicefuncs.py

nice_funcs.py > get_position

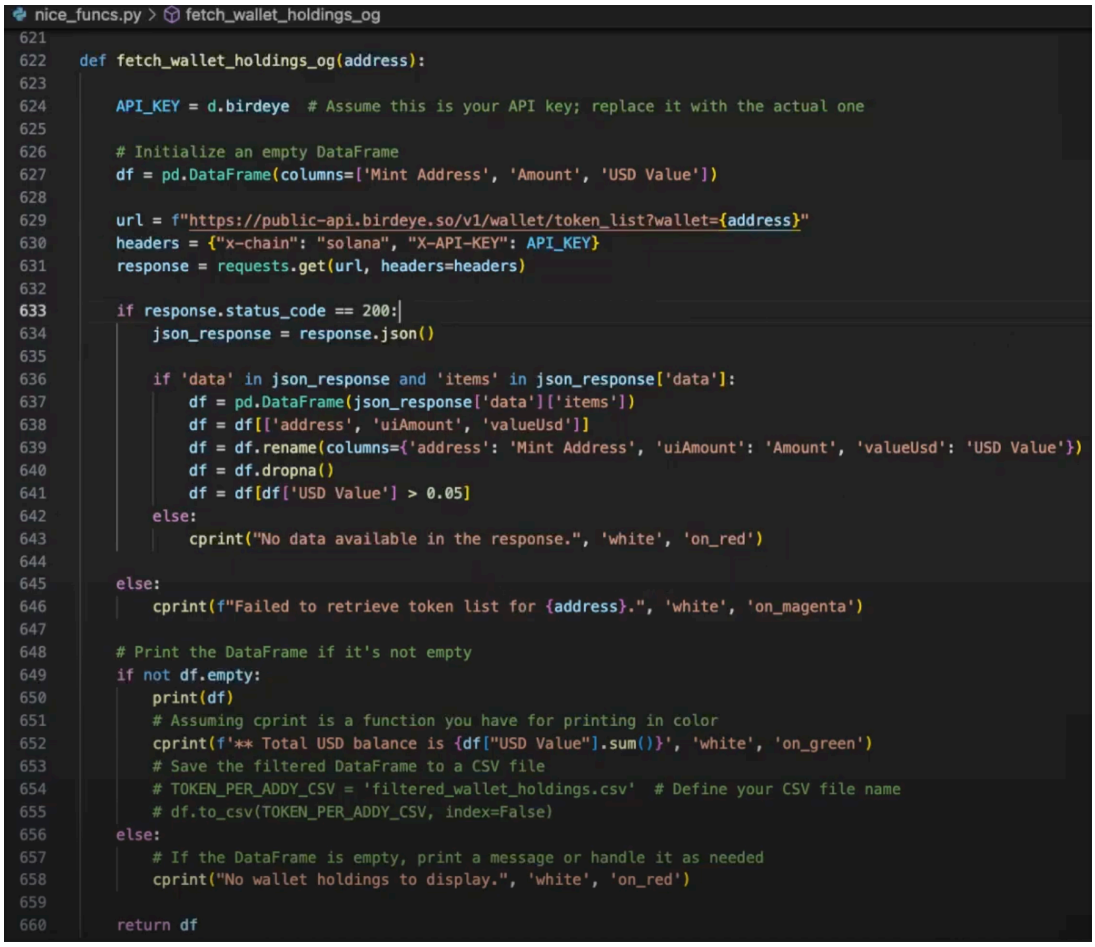
```
def get_position(token_mint_address):
    """
    Fetches the balance of a specific token given its mint address from a DataFrame.

    Parameters:
    - dataframe: A pandas DataFrame containing token balances with columns ['Mint Address', 'Amount'].
    - token_mint_address: The mint address of the token to find the balance for.

    Returns:
    - The balance of the specified token if found, otherwise a message indicating the token is not in the wallet.
    """

    dataframe = fetch_wallet_token_single(address, token_mint_address)

    #dataframe = pd.read_csv('data/token_per_addy.csv')

    print('-----------------')
    #print(dataframe)

    #print(dataframe)

    # Check if the DataFrame is empty
    if dataframe.empty:
        print("The DataFrame is empty. No positions to show.")
        return 0  # Indicating no balance found

    # Ensure 'Mint Address' column is treated as string for reliable comparison
    dataframe['Mint Address'] = dataframe['Mint Address'].astype(str)

    # Check if the token mint address exists in the DataFrame
    if dataframe['Mint Address'].isin([token_mint_address]).any():
        # Get the balance for the specified token
        balance = dataframe.loc[dataframe['Mint Address'] == token_mint_address, 'Amount'].iloc[0]
        #print(f"Balance for {token_mint_address[-4:]} token: {balance}")
        return balance
    else:
        # If the token mint address is not found in the DataFrame, return a message indicating so
        print("Token mint address not found in the wallet.")
        return 0  # Indicating no balance found
```

#Def fetch_wllet_token_single



```
622  def fetch_wallet_holdings_og(address):
650          print(df)
651          # Assuming cprint is a function you have for printing in color
652          cprint(f'** Total USD balance is {df["USD Value"].sum()}', 'white', 'on_green')
653          # Save the filtered DataFrame to a CSV file
654          # TOKEN_PER_ADDY_CSV = 'filtered_wallet_holdings.csv'  # Define your CSV file name
655          # df.to_csv(TOKEN_PER_ADDY_CSV, index=False)
656      else:
657          # If the DataFrame is empty, print a message or handle it as needed
658          cprint("No wallet holdings to display.", 'white', 'on_red')
659
660      return df
661
662  def fetch_wallet_token_single(address, token_mint_address):
663
664      df = fetch_wallet_holdings_og(address)
665
666      # filter by token mint address
667      df = df[df['Mint Address'] == token_mint_address]
668
669      return df
670
671
672  def token_price(address):
673      API_KEY = d.birdeye
674      url = f"https://public-api.birdeye.so/defi/price?address={address}"
675      headers = {"X-API-KEY": API_KEY}
676      response = requests.get(url, headers=headers)
677      price_data = response.json()
678
679      if price_data['success']:
680          return price_data['data']['value']
681      else:
682          return None
683
684
685  def get_position(token_mint_address):
```

#Def fetch_wallet_holdings



Nicefuncs.py
#def_chunk_hkill

nice_funcs.py > chunk_kill

```python
def chunk_kill(token_mint_address, max_usd_sell_size, slippage):

    '''

    this function closes the position in full  after CHUNKING ORDERS

    if the usd_size > 10k then it will chunk in 10k orders
    '''

    # if time is on the 5 minute do the balance check, if not grab from data/current_position.csv
    balance = get_position(token_mint_address)
    cprint(f'inside chunk... balance {balance}', 'white', 'on_black')

    # get current price of token
    price = token_price(token_mint_address)
    price = float(price)
    usd_value = balance * price

    if usd_value < max_usd_sell_size:
        sell_size = balance
    else:
        sell_size = max_usd_sell_size/price


    cprint(f'balance is {balance} and selling this amount {sell_size}', 'white', 'on_black')


    # round to 2 decimals
    sell_size = round_down(sell_size, 2)
    decimals = 0
    decimals = get_decimals(token_mint_address)
    sell_size = int(sell_size * 10 **decimals)

    #print(f'bal: {balance} price: {price} usdVal: {usd_value} TP: {tp} sell size: {sell_size} decimals: {decimals}')

    while usd_value > 0:
# 100 selling 70% ..... selling 30 left
        #print(f'for {token_mint_address[-4:]} closing position cause exit all positions is set to {EXIT_ALL_POSITIONS} and value is {usd_value}')
        try:
```

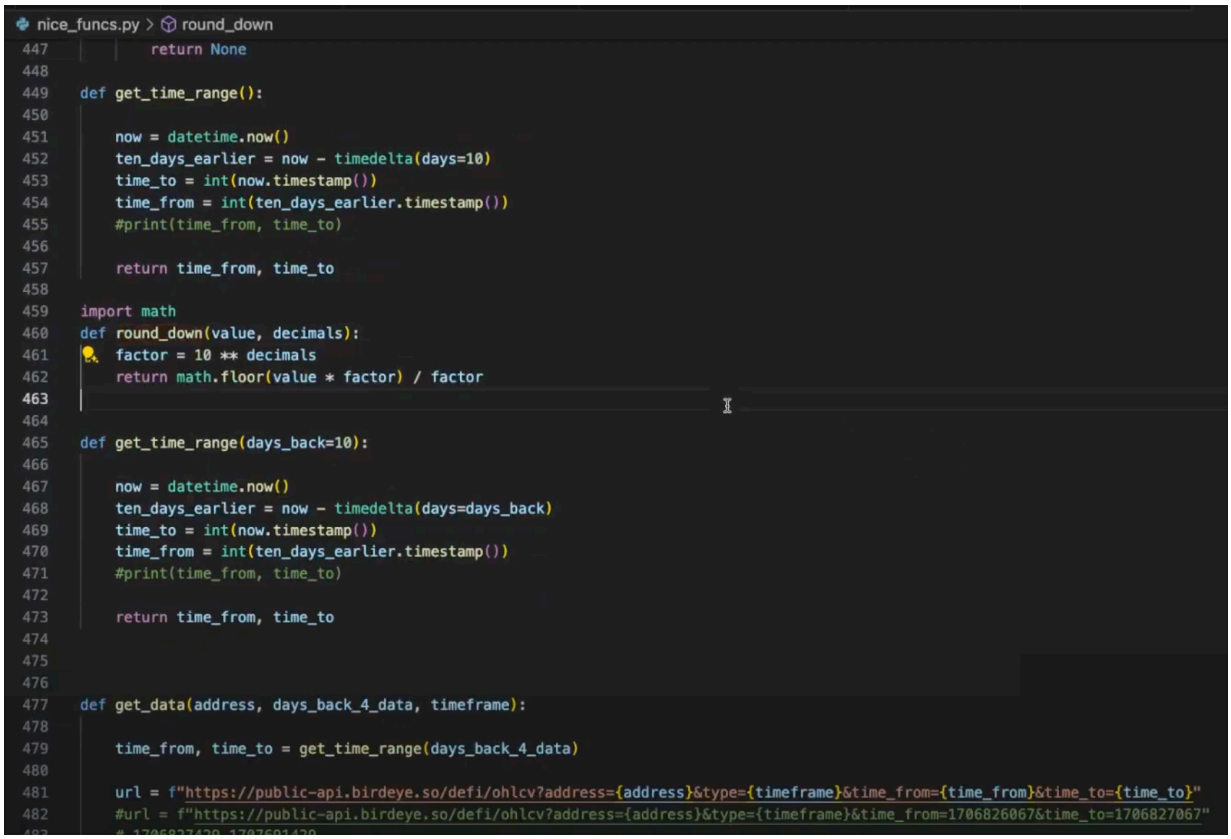
```python
def chunk_kill(token_mint_address, max_usd_sell_size, slippage):
        #print(f'for {token_mint_address[-4:]} closing position cause exit all positions is set to {EXIT_ALL_POSITIONS}
        try:

            market_sell(token_mint_address, sell_size, slippage)
            cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
            time.sleep(1)
            market_sell(token_mint_address, sell_size, slippage)
            cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
            time.sleep(1)
            market_sell(token_mint_address, sell_size, slippage)
            cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
            time.sleep(15)

        except:
            cprint('order error.. trying again', 'white', 'on_red')
            # time.sleep(7)

        balance = get_position(token_mint_address)
        price = token_price(token_mint_address)
        usd_value = balance * price
        tp = sell_at_multiple * USDC_SIZE


        if usd_value < max_usd_sell_size:
            sell_size = balance
        else:
            sell_size = max_usd_sell_size/price


        cprint(f'balance is {balance} and selling this amount {sell_size}', 'white', 'on_black')


        # down downwards to 2 decimals
        sell_size = round_down(sell_size, 2)

        decimals = 0
        decimals = get_decimals(token_mint_address)
        #print(f'xxxxxxxxx for {token_mint_address[-4:]} decimals is {decimals}')
        sell_size = int(sell_size * 10 **decimals)
```

```
1031            #print(f'xxxxxxxxx for {token_mint_address[-4:]} decimals is {decimals}')
1032            sell_size = int(sell_size * 10 **decimals)
1033
1034
1035        else:
1036            print(f'for {token_mint_address[-4:]} value is {usd_value} ')
1037            #time.sleep(10)
1038
1039        print('closing position in full...')
1040
1041    def kill_switch(token_mint_address, slippage):
1042
1043        ''' this function closes the position in full
```

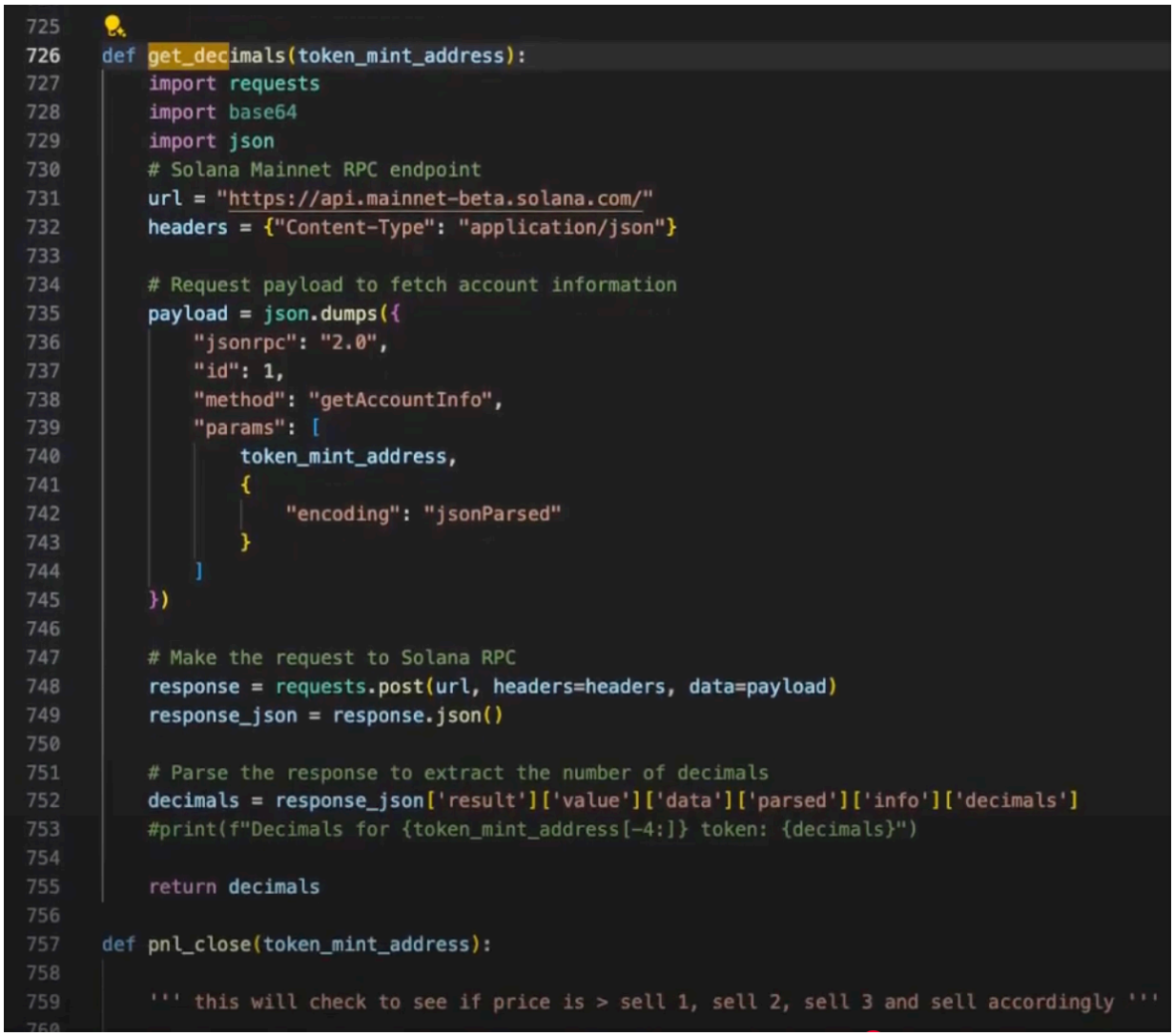
#Def get_token_price

```
nice_funcs.py > token_price
622  def fetch_wallet_holdings_og(address):
660      return df
661
662  def fetch_wallet_token_single(address, token_mint_address):
663
664      df = fetch_wallet_holdings_og(address)
665
666      # filter by token mint address
667      df = df[df['Mint Address'] == token_mint_address]
668
669      return df
670
671
672  def token_price(address):
673      API_KEY = d.birdeye
674      url = f"https://public-api.birdeye.so/defi/price?address={address}"
675      headers = {"X-API-KEY": API_KEY}
676      response = requests.get(url, headers=headers)
677      price_data = response.json()
678
679      if price_data['success']:
680          return price_data['data']['value']
681      else:
682          return None
683
684
685  def get_position(token_mint_address):
686      """
687      Fetches the balance of a specific token given its mint address from a DataFrame.
688
689      Parameters:
690      - dataframe: A pandas DataFrame containing token balances with columns ['Mint Address', 'Amount'].
691      - token_mint_address: The mint address of the token to find the balance for.
692
693      Returns:
694      - The balance of the specified token if found, otherwise a message indicating the token is not in the wallet.
695      """
```

#Def Round_down

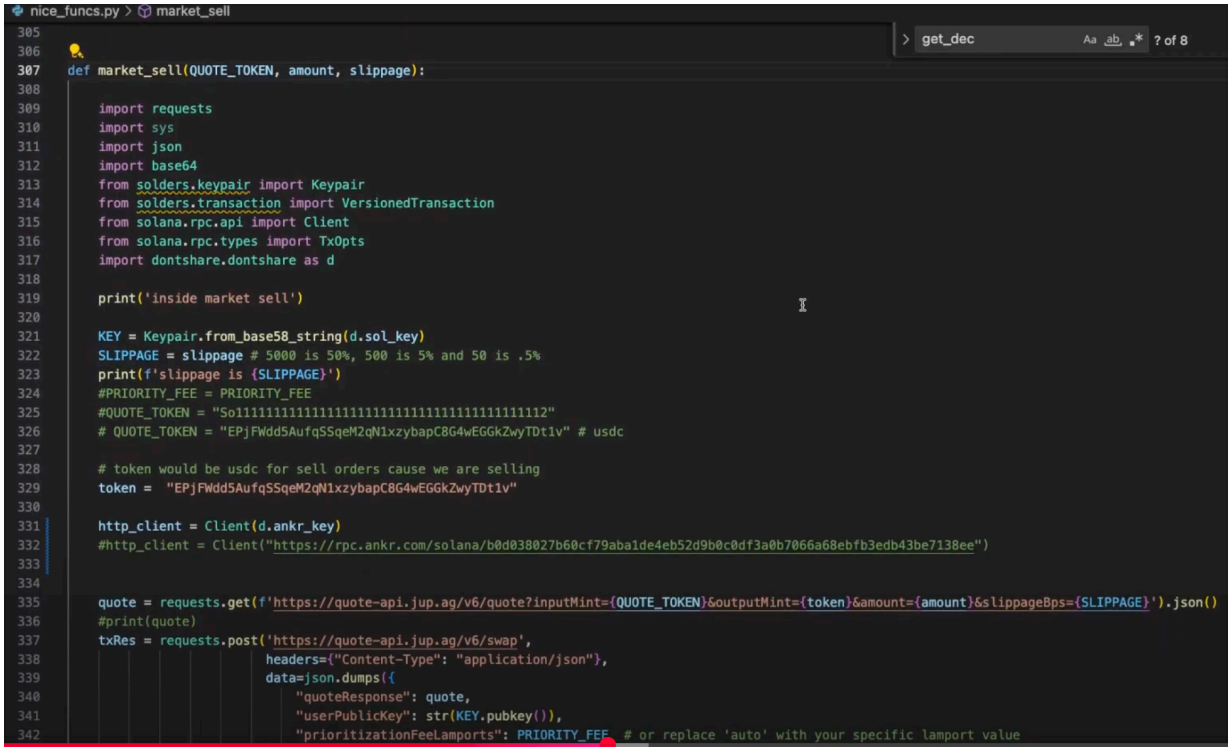


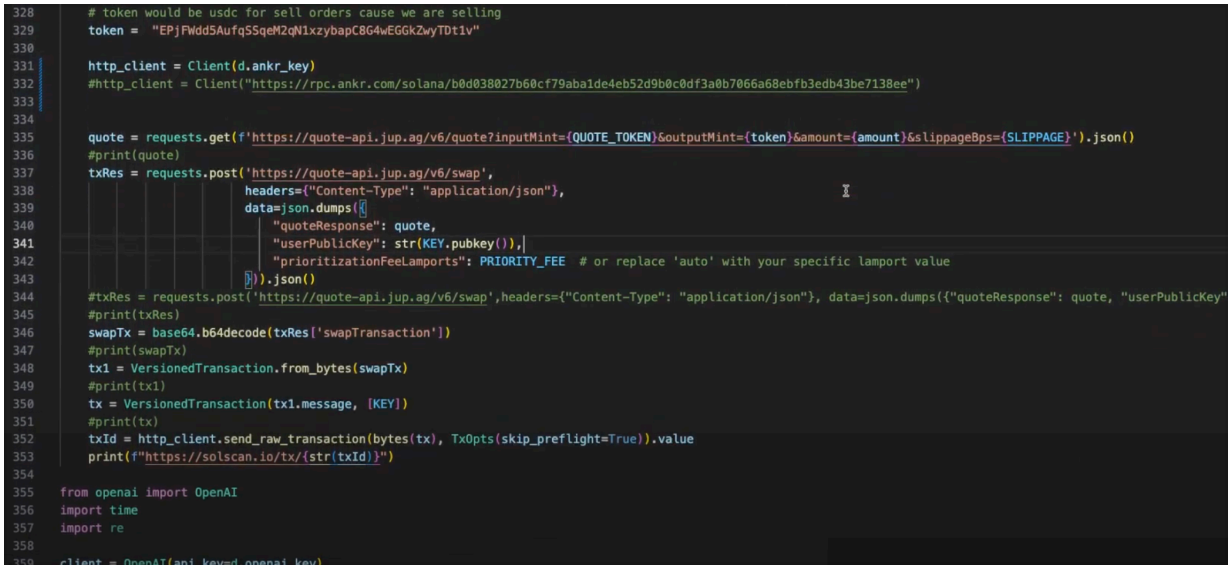
#Def get_decimals



```python
def get_decimals(token_mint_address):
    import requests
    import base64
    import json
    # Solana Mainnet RPC endpoint
    url = "https://api.mainnet-beta.solana.com/"
    headers = {"Content-Type": "application/json"}

    # Request payload to fetch account information
    payload = json.dumps({
        "jsonrpc": "2.0",
        "id": 1,
        "method": "getAccountInfo",
        "params": [
            token_mint_address,
            {
                "encoding": "jsonParsed"
            }
        ]
    })

    # Make the request to Solana RPC
    response = requests.post(url, headers=headers, data=payload)
    response_json = response.json()

    # Parse the response to extract the number of decimals
    decimals = response_json['result']['value']['data']['parsed']['info']['decimals']
    #print(f"Decimals for {token_mint_address[-4:]} token: {decimals}")

    return decimals

def pnl_close(token_mint_address):

    ''' this will check to see if price is > sell 1, sell 2, sell 3 and sell accordingly '''
```

#Def get_market_sell





25_Config file

```
5   ############### main.py configurations ###############
6
7   #symbol1 = 'EjmDTt8G3T725eFSV7oWmGD8J848guo3LZ1EB3RfwGSw' # harold
8   #symbol1 = '3psH1Mj1f7yUfaD5gh6Zj7epE8hhrMkMETgv5TshQA4o' # boden
9   symbol1 = 'EQGG5muKhviWmWJwy4Fi9TeeNpJUX7RpHAwkQMnTAyAj' # indian
10  #symbol1 = '6D7NaB2xsLd7cauWu1wKk6KBsJohJmP2qZH9GEfVi5Ui' # shark
11  #symbol1 = 'EKpQGSJtjMFqKZ9KQanSqYXRcF8fBopzLHYxdM65zcjm' # wif
12  #symbol1 = '5LafQUrVco6o7KMz42eqVEJ9LW31StPyGjeeu5sKoMtA' # mumu
13  #symbol1 = '89KX9c3rNoW5sVnwMQvPPFgYKxpABupCSqXUtB1CW5J2'
14  #symbol1 = '5mbK36SZ7J19An8jFochhQS4of8g6BwUjbeCSxBSoWdp' # michi
15  #symbol1 = '2ee2GwuKBEGWUxSwQeSxT3NjG3rak3ULx8kHjVUR26Jv' # glub
16  #symbol1 = '3hbKad9n27siYjhgSmQBP5r6C7n2aGYrzS2RKfpw4M8e' #cc
17  #symbol1 = '3hbKad9n27siYjhgSmQBP5r6C7n2aGYrzS2RKfpw4M8e'
18  #symbol1 = 'JCAbmex5ddCmQsJ2sKc1WJPbb1NUHg4eygvZbay2H2r9'
19  #symbol1 = '4o7piaDSo9bGFup7fbti3AsXthgSPnkcignGVU9sNxJV'
20  #symbol1 = 'EZUFNJMZTBpungQX2czEb9ZyCMjtdzsDGMK4UywDUa1F'
21  #symbol1 = 'AfLLKxJHebxoLG57iWABV71vRuSKPJLwMWbirMtkq2jD'
22  #symbol1 = '8H2t2FxnCt9VNmt3ReLP7eQhjzMVxb59Bj3BPsF41oqC'
23  #symbol1 = '3dD6qoZPjAT3fNDVd9CWvWEy3rigJdVP2WdG2Fe23SPd'
24  #symbol1 = 'GJgHsc1HU4ibmzW6oWQr8L2RRT95ATc1BoNuLkp94AwU' # WBS
25  #symbol1 = '5ibZoZneHrus9fThCHPWeH5tHJQ5HC1epicUEUi2pXnG'
26  #symbol1 = '6k33czoudK1Z5YNnWPUzw8uvBrsRjwHLJALERNch5Y4R'
```

```
28
29  # Define the token batch as a list of strings
30  token_batch = [
31      "dekNoN3D8mXa4JHLwTbVXz8aPAyJUkk443UjcSpJKi4",
32      "JD3S1oqnjG5trRTZXpLmKnS7LsKsppyFa51rPBBMWogv",
33      "5VgY2RzbB2adNpengMACREuRfePwwZ5L6RqBj3rwDUj3",
34      '6e6rViDzavLRv56nvZye5UofrKDg36mf6dTQrMCoPTW9'
35  ]
36
37  usd_size = 350 # .001 * size = exchange fee $15 -- .015
```

```
usd_size = 350 # .001 * size = exchange fee $15 -- .015
max_usd_order_size = 7 # max usd size to sell in one chunk
tx_sleep = 5 # for entering. closing we spam 3 in a row. hard coded.
slippage = 199 # 5000 = 50% slippage, so 500 = 5% and 50 = .5% slippage
PRIORITY_FEE = 100000 # 200000 is about .035 usd at 150 sol, after a bit of testing 100000 is sufficient and is .02 usd
orders_per_open = 5

# PNL CLOSE
lowest_balance = 1400
target_balance = 3000

# LIQ BOT
long_liq_threshold = 1000
short_liq_threshold = 1000
```