T1A3 - Create A Terminal Application

Repository

2w00fz Github - My Repo, T1A3

Coding style conventions: In line with PEP8. Common styling includes:

- Limiting redundant white spaces
- 4 spaces per indent
- Limiting redundant operators

  ```
  #wrong
  if x == True
  #right
  if x
  ```

- Using specific Exception Handlers (Key Error, Type Error etc)
- Avoid using single letter variables that look similar to numerals
- Class Names using CamelCase
- Functions and variables using lower_case_with_underscores

Three Features

Pull Transaction Ledger From KuCoin

Use of Variables and Variable Scope

REQUIRES RECIEVING DATA THAT IS STORED IN A DICTIONARY.

- All the values are either recieved as strings and some need to be converted to integers and floats. One value contains another dictionary that needs to be deconstructed in order to parse it's values.

LOOPS AND CONDITIONAL CONTROL STRUCTURES

- Loops are required to fetch data day by day (as limited by the API)
- Loops also required to increase the page number if more than 500 entries are returned
- Loops required to retry the API call if an error is returned
- Conditions are required for changing behaviour based on error codes, or the code reaching the end of the days it has been fetching data for
- Conditons also required for checking that the Ledger Data does not already exist (a user can input data to be processed that has previously been fetched)

ERROR HANDLING

- Making sure that user supplied data is not empty
- Pausing the program if a Rate Limit has been reached with the API
- User supplied file name does not contain illegal characters

Calculate Combined Profit and Loss for each Asset

Use of Variables and Variable Scope, Control Structures

MAKING DECISIONS ABOUT DIFFERENT VARIABLES

- For each item processed only one asset is given
- It could either be the Asset that was traded for, or it could be the Base Asset used as the Currency
- To determine this the value will be passed to a different function to and compared with other data recieved to determine what it is.

- If it is a Base Asset it needs to be sent to another module to query the API for it's value in USDT (1 USDT = 1 USD, theoretically...)
- These details then need to be passed to an Order object and an Asset object to track the overall value and to create a list of orders that can be exported.

ERROR HANDLING

- The Rate Limit for fetching prices is very low so prices will be saved for a day ahead of what is required, and an exception will be printed with a countdown timer to retry when it happens
- Some assets may have changed symbol (or delisted) and will give a different error that will return a place holder value, and a log entry made so that it can be manually reviewed afterwards.
- Type errors are avoided by Type Converting numbers to strings and vice versa

Export Results as a CSV

Use of Variables and Variable Scope, Control Structures

The values to be exported are Objects that are contained in within a dictionary. They write_csvs() function gets called from the ProccessOrders class by the executing function in the main program. A header is added first containing a string of the keys, which act as the first row in a spreadsheet (column names), then the returned values are destructured and restructured as a string, with formatting inline with CSV standards.

Error Handling

All data being written ends up in an fstring so the input data is largely agnostic about the data type, however the name provided by the user to save the file will be checked for illegal characters using regex, alerting them to any characters provided that are not allowed.

INSTALLATION

System Requirements

Hardware

Minimum

Raspberry Pi B+ 512MB RAM 1GB Free Space

Preferential

2GB RAM 2GB Free Space

Software

Python 3.xx

Installation Procedure

Dependencies

- Git
- KuCoin Python SDK

Other Requirements

API Keys are required to fetch account data These can be generated from the API Managment section in your KuCoin profile. Your API Key, API Secret and API Passphrase are required.

Installation

Option 1

Create a file named run.sh and paste following code in to it

```
#/bin/bash

git clone "https://github.com/2w00fs/T1A3.git"
cd T1A3
pip install kucoin-python

touch credentials.py
echo Please input your API Key
read api_key
echo api_key= \"$api_key\" >> credentials.py
echo Please input your API Secret
read api_secret
echo api_secret= \"$api_secret\" >> credentials.py
echo Please input your API Passphrase
read api_passphrase
echo api_passphrase= \"$api_passphrase\" >> credentials.py

python3 main.py
```

Open a terminal in the directory that contains the script, and that you would like to install it in and run "bash run.sh"

Option 2

- From the Command Line, run "git clone "https://github.com/2w00fs/T1A3.git" to download the repository.
- Run "pip install kucoin-python" to install the KuCoin SDK
- Create a file named "credentials.py" and store your API details in the following format:

  ```
  api_key=
  api_secret=
  api_passphrase=
  ```

- Run the program using "python3 main.py"

Usage

When running using "python3 main.py" the only argument you will be prompted for is a File Name. Naming Conventions are standard characters, however Spaces are not allowed.

Two Command Flags are optional:

```
-ledger or --ledgername
```

eg. python3 main.py -ledger myfirstledger or

```
-prices --pricelist
```

eg. python3 main.py --pricelist savedprices

They can be used to specify data files that may have previously been downloaded by the program. They can be found in the root directory with a file extension of .json They can be used as an input and they will be updated with newer values if any are available.

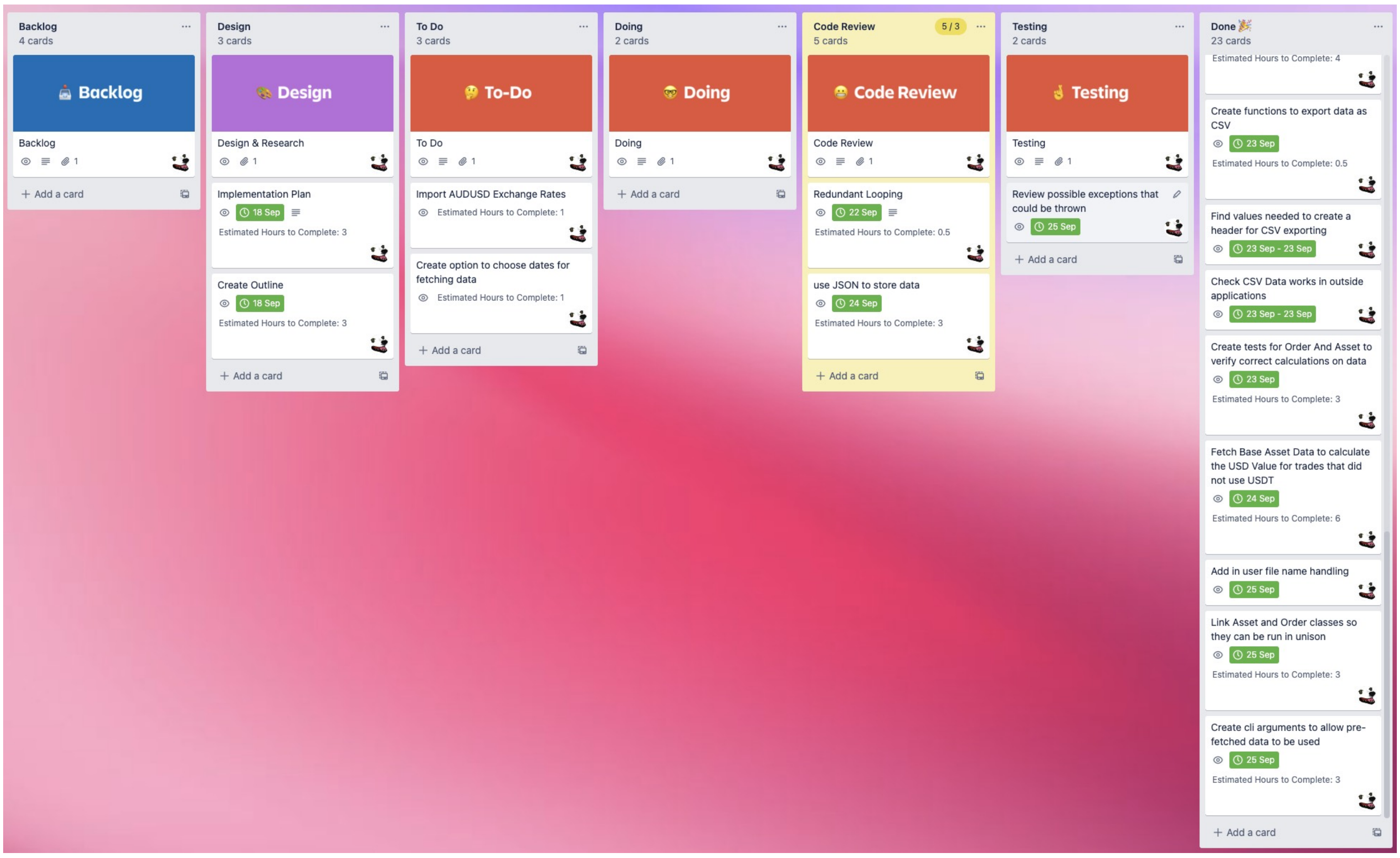
Exported CSV files will be located in the root directory with the name provided, suffixed with _Assets or _Orders for the respective data. A Log file will also be stored for any errors that may occur.

Implementation Plan

Checklist of Features

Checklist	ETA	Feature 1	Checklist	ETA	Feature 2	Checklist	ETA	Feature 3
		Collate trade ledger data from KuCoin API			**Calculates combined PNL for each asset**			**Export Data to CSV**
✓	**FOREVER**	**USE A FLOW CHART TO MAP OUT IDEAS AND KEEP TRACK OF CONTROL FLOW / WHAT YOU ARE THINKING**	✓	**FOREVER**	**USE A FLOW CHART TO MAP OUT IDEAS AND KEEP TRACK OF CONTROL FLOW / WHAT YOU ARE THINKING**	✓	**FOREVER**	**USE A FLOW CHART TO MAP OUT IDEAS AND KEEP TRACK OF CONTROL FLOW / WHAT YOU ARE THINKING**
✓	**MONDAY**	Pip install their Python SDK	✓	**TUESDAY**	Work out which asset is the Main Asset traded and which was used as the currency (Base Asset)	✓	**WEDNESDAY**	Research the format of a CSV
✓	**MONDAY**	Create API key	✓	**TUESDAY / WEDNESDAY**	Combine individual fill orders in to one order per Order Id	✓	**THURSDAY**	Take input from user to name the file
✓	**MONDAY**	Review documentation to find functions to call in their modules	✓	**TUESDAY**	Adjust the size of each asset so if it was sold it would have a negative size, and if it was bought it would have a positive value	✓	**THURSDAY**	Check they haven't entered any illegal characters that the file system can't use
✓	**MONDAY**	Test that a proper connection can be established	✓	**WEDNESDAY**	Add and subtract the asset size from a list of assets (seperate to the order list)	✓	**FRIDAY**	Add Header Values before the body of data is looped through it, end each write to file with "\n" to generate a new line
✓	**MONDAY**	Look at what data type is returned, and the names of the variables to facilitate processing and using when it is returned	✓	**WEDNESDAY**	Create data with known values to test the functions and check the output matches what was expected	✓	**FRIDAY**	Loop through Order Data, seperating each value with a comma and writing it to file
✓	**TUESDAY**	Fetch data and deconstruct it to print out each item	✓	**THURSDAY**	Get the USD value of the Base Asset to calculate the value of the trade when it was made	✓	**FRIDAY**	!! Close the file
✓	**TUESDAY**	Print out data so it can be manually copied as a dictionary and pasted in to a new file to be imported as a variable	✓	**FRIDAY**	Add the USD Value to the Asset list and as an extra value for each Order	✓	**FRIDAY**	Repeat with Asset Data
✓	**WEDNESDAY / THURSDAY**	Use json module to properly store and export ledger data				✓	**FRIDAY**	Check that it opens up properly in Numbers/Excel/Google Sheets

Trello Board to keep track of task and add new ones different needs or problems arise



Backlog — 4 cards
- Backlog — 1
- + Add a card

Design — 3 cards
- Design & Research — 1
- Implementation Plan — 18 Sep — Estimated Hours to Complete: 3
- Create Outline — 18 Sep — Estimated Hours to Complete: 3
- + Add a card

To Do — 3 cards
- To Do — 1
- Import AUDUSD Exchange Rates — Estimated Hours to Complete: 1
- Create option to choose dates for fetching data — Estimated Hours to Complete: 1
- + Add a card

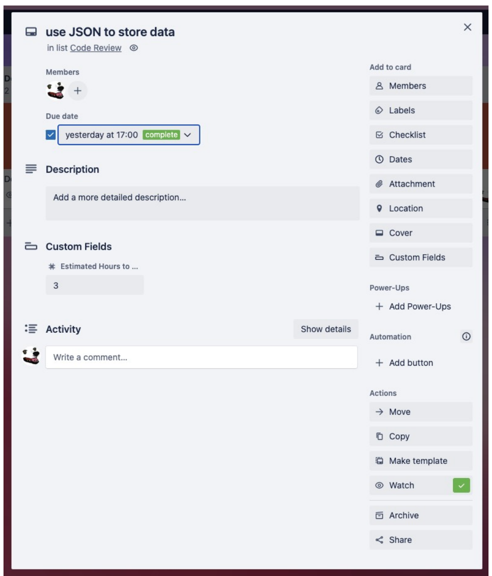
Doing — 2 cards
- Doing — 1
- + Add a card

Code Review — 5 cards — 5 / 3
- Code Review — 1
- Redundant Looping — 22 Sep — Estimated Hours to Complete: 0.5
- use JSON to store data — 24 Sep — Estimated Hours to Complete: 3
- + Add a card

Testing — 2 cards
- Testing — 1
- Review possible exceptions that could be thrown — 25 Sep
- + Add a card

Done — 23 cards
- Estimated Hours to Complete: 4
- Create functions to export data as CSV — 23 Sep — Estimated Hours to Complete: 0.5
- Find values needed to create a header for CSV exporting — 23 Sep - 23 Sep
- Check CSV Data works in outside applications — 23 Sep - 23 Sep
- Create tests for Order And Asset to verify correct calculations on data — 23 Sep — Estimated Hours to Complete: 3
- Fetch Base Asset Data to calculate the USD Value for trades that did not use USDT — 24 Sep — Estimated Hours to Complete: 6
- Add in user file name handling — 25 Sep
- Link Asset and Order classes so they can be run in unison — 25 Sep — Estimated Hours to Complete: 3
- Create cli arguments to allow pre-fetched data to be used — 25 Sep — Estimated Hours to Complete: 3
- + Add a card

Easy to add in times with



use JSON to store data
in list Code Review

Members

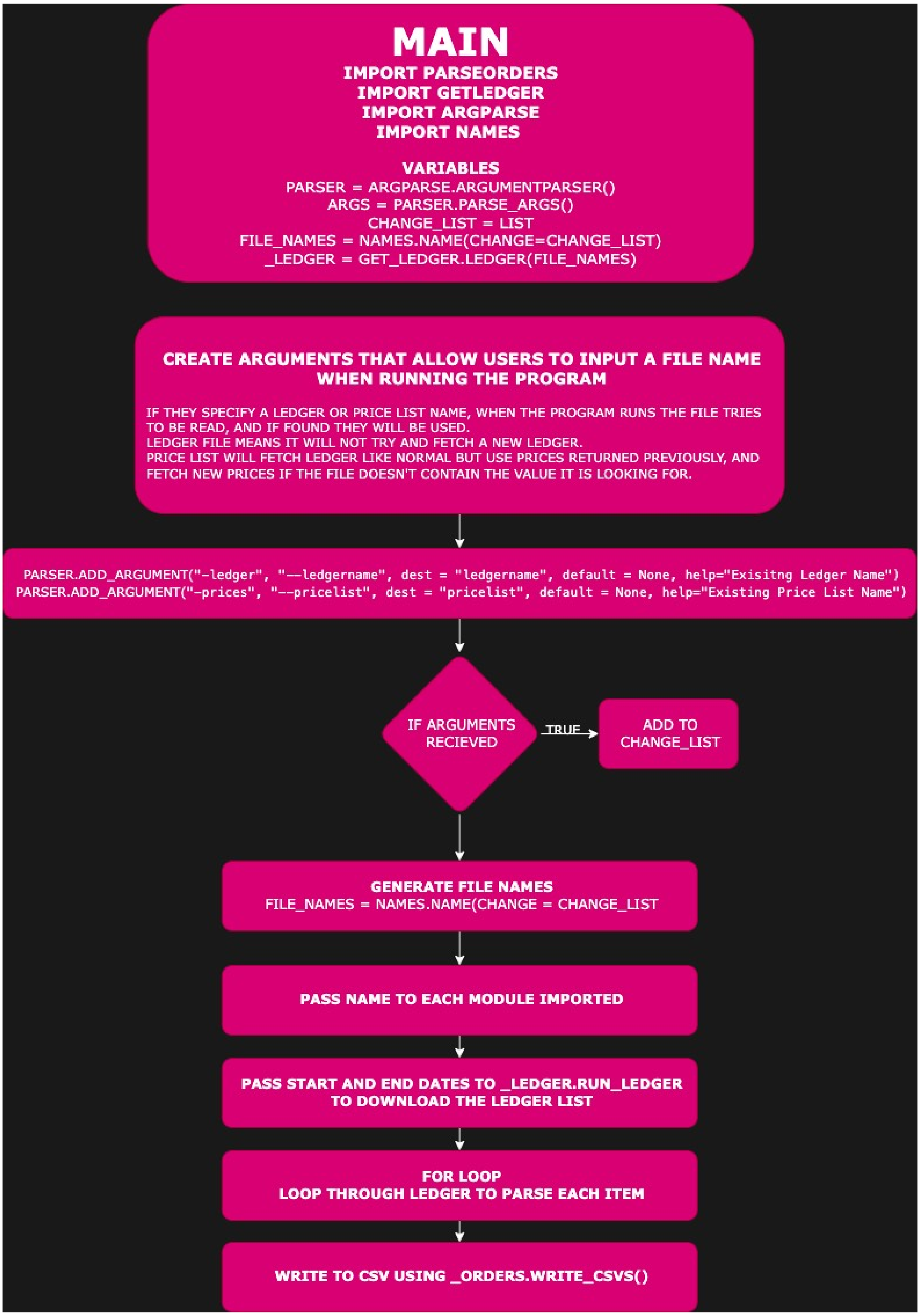
Due date: yesterday at 17:00 — complete

Description
Add a more detailed description...

Custom Fields
Estimated Hours to ...: 3

Activity — Show details
Write a comment...

Lots of Flow Charts to help visualise what I am trying to achieve



MAIN

IMPORT PARSEORDERS
IMPORT GETLEDGER
IMPORT ARGPARSE
IMPORT NAMES

VARIABLES

```
PARSER = ARGPARSE.ARGUMENTPARSER()
ARGS = PARSER.PARSE_ARGS()
CHANGE_LIST = LIST
FILE_NAMES = NAMES.NAME(CHANGE=CHANGE_LIST)
_LEDGER = GET_LEDGER.LEDGER(FILE_NAMES)
```

CREATE ARGUMENTS THAT ALLOW USERS TO INPUT A FILE NAME WHEN RUNNING THE PROGRAM

IF THEY SPECIFY A LEDGER OR PRICE LIST NAME, WHEN THE PROGRAM RUNS THE FILE TRIES TO BE READ, AND IF FOUND THEY WILL BE USED.
LEDGER FILE MEANS IT WILL NOT TRY AND FETCH A NEW LEDGER.
PRICE LIST WILL FETCH LEDGER LIKE NORMAL BUT USE PRICES RETURNED PREVIOUSLY, AND FETCH NEW PRICES IF THE FILE DOESN'T CONTAIN THE VALUE IT IS LOOKING FOR.

```
PARSER.ADD_ARGUMENT("-ledger", "——ledgername", dest = "ledgername", default = None, help="Exisitng Ledger Name")
PARSER.ADD_ARGUMENT("-prices", "——pricelist", dest = "pricelist", default = None, help="Existing Price List Name")
```

IF ARGUMENTS RECIEVED

ADD TO CHANGE_LIST

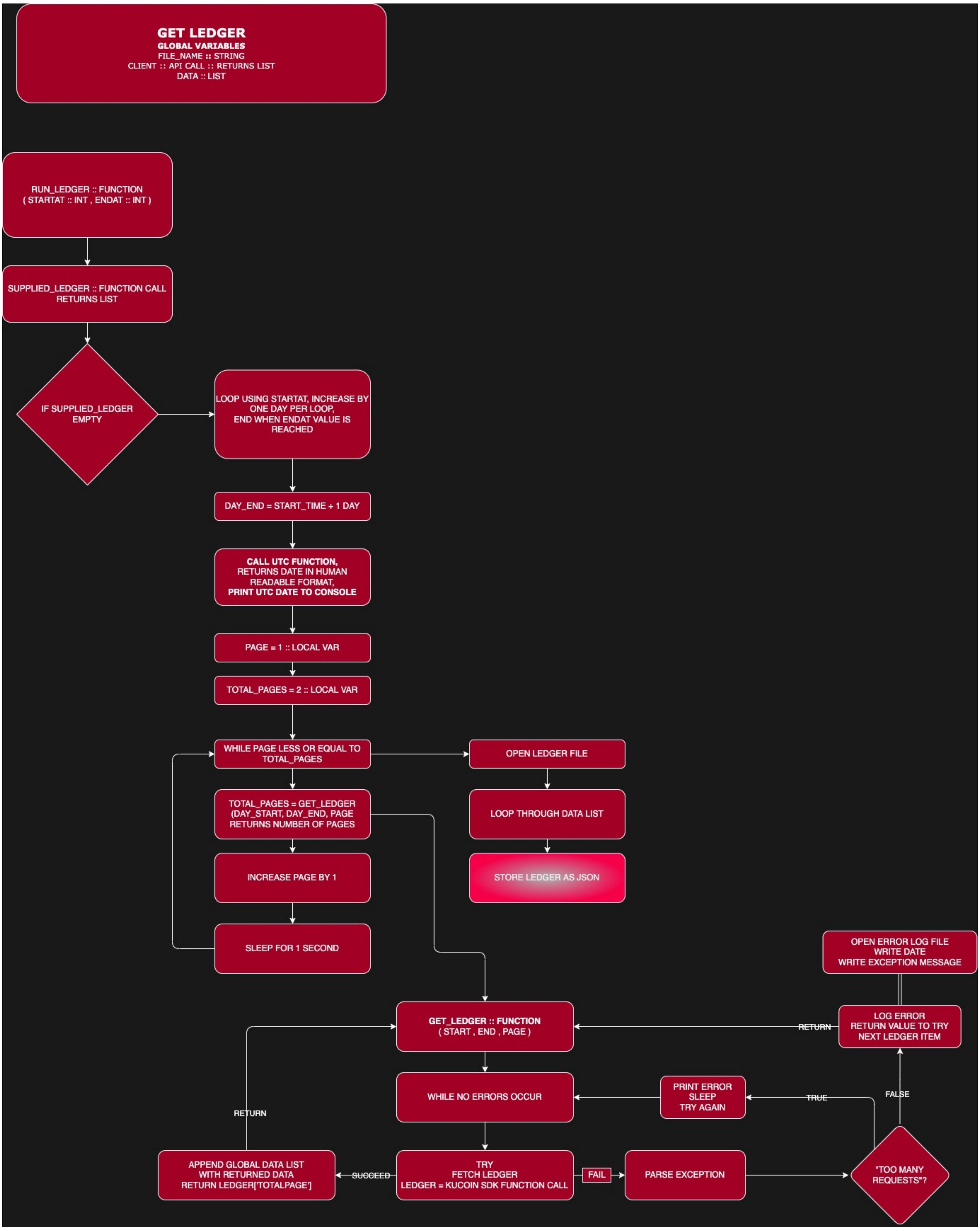
GENERATE FILE NAMES
FILE_NAMES = NAMES.NAME(CHANGE = CHANGE_LIST

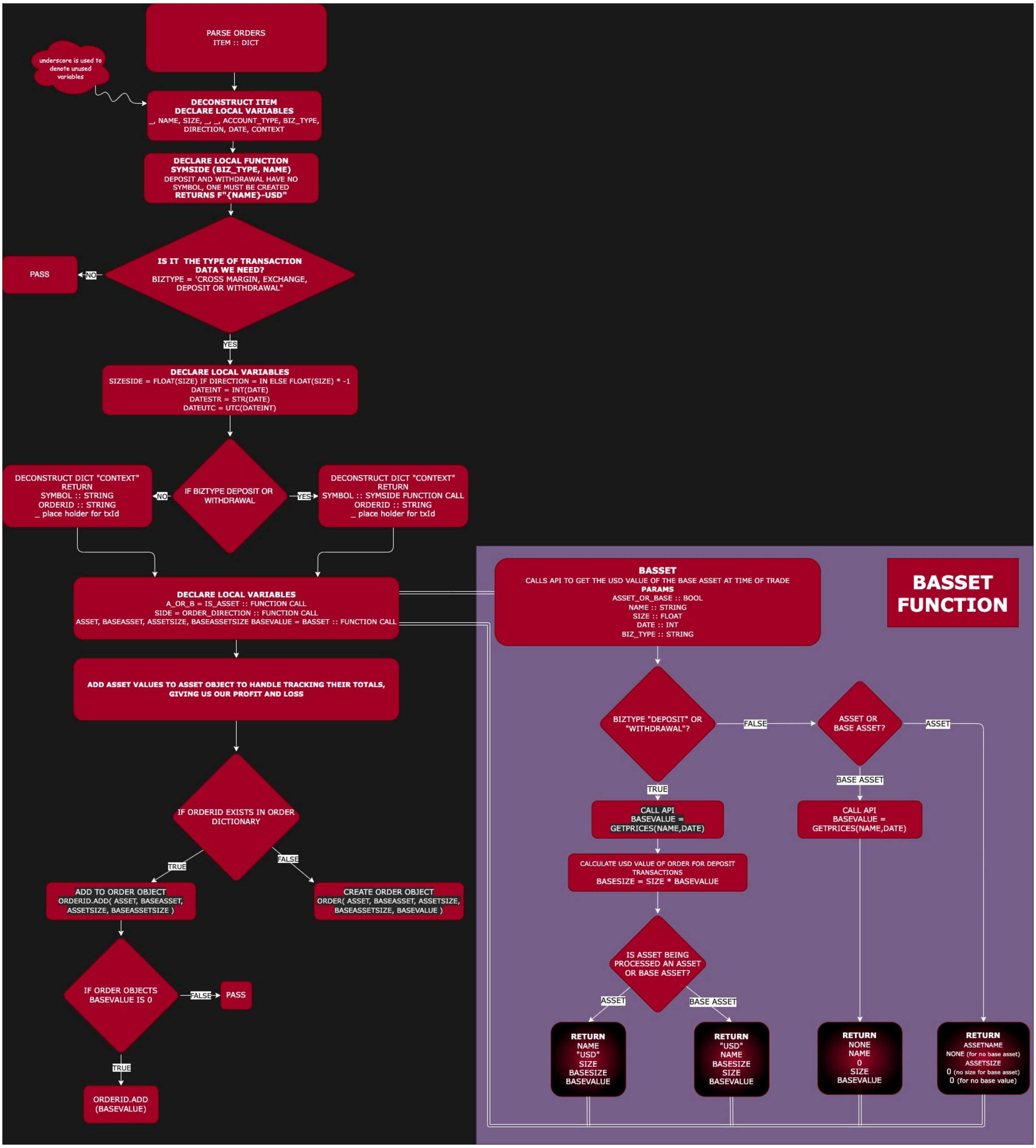
PASS NAME TO EACH MODULE IMPORTED

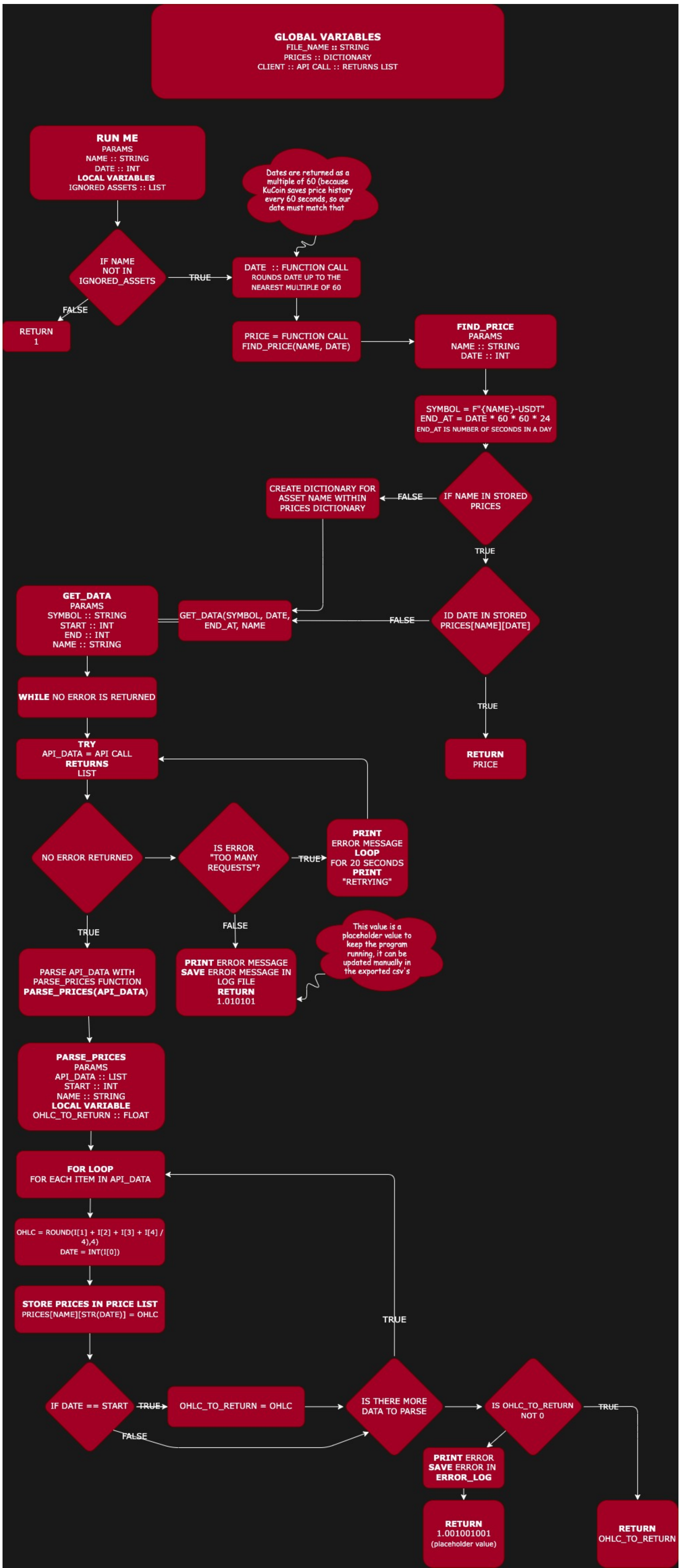
PASS START AND END DATES TO _LEDGER.RUN_LEDGER TO DOWNLOAD THE LEDGER LIST

FOR LOOP
LOOP THROUGH LEDGER TO PARSE EACH ITEM

WRITE TO CSV USING _ORDERS.WRITE_CSVS()


GET LEDGER
GLOBAL VARIABLES
FILE_NAME :: STRING
CLIENT :: API CALL :: RETURNS LIST
DATA :: LIST
RUN_LEDGER :: FUNCTION (STARTAT :: INT , ENDAT :: INT)
SUPPLIED_LEDGER :: FUNCTION CALL RETURNS LIST
IF SUPPLIED_LEDGER EMPTY
LOOP USING STARTAT, INCREASE BY ONE DAY PER LOOP, END WHEN ENDAT VALUE IS REACHED
DAY_END = START_TIME + 1 DAY
CALL UTC FUNCTION, RETURNS DATE IN HUMAN READABLE FORMAT, PRINT UTC DATE TO CONSOLE
PAGE = 1 :: LOCAL VAR
TOTAL_PAGES = 2 :: LOCAL VAR
WHILE PAGE LESS OR EQUAL TO TOTAL_PAGES
TOTAL_PAGES = GET_LEDGER (DAY_START, DAY_END, PAGE RETURNS NUMBER OF PAGES
INCREASE PAGE BY 1
SLEEP FOR 1 SECOND
OPEN LEDGER FILE
LOOP THROUGH DATA LIST
STORE LEDGER AS JSON
OPEN ERROR LOG FILE WRITE DATE WRITE EXCEPTION MESSAGE
GET_LEDGER :: FUNCTION (START , END , PAGE)
RETURN
LOG ERROR RETURN VALUE TO TRY NEXT LEDGER ITEM
WHILE NO ERRORS OCCUR
PRINT ERROR SLEEP TRY AGAIN
TRUE
FALSE
RETURN
APPEND GLOBAL DATA LIST WITH RETURNED DATA RETURN LEDGER['TOTALPAGE']
SUCCEED
TRY FETCH LEDGER LEDGER = KUCOIN SDK FUNCTION CALL
FAIL
PARSE EXCEPTION
"TOO MANY REQUESTS"?


PARSE ORDERS
ITEM :: DICT
underscore is used to denote unused variables
DECONSTRUCT ITEM
DECLARE LOCAL VARIABLES
_, NAME, SIZE, _, _, ACCOUNT_TYPE, BIZ_TYPE, DIRECTION, DATE, CONTEXT
DECLARE LOCAL FUNCTION
SYMSIDE (BIZ_TYPE, NAME)
DEPOSIT AND WITHDRAWAL HAVE NO SYMBOL, ONE MUST BE CREATED
RETURNS F"{NAME}-USD"
IS IT THE TYPE OF TRANSACTION DATA WE NEED?
BIZTYPE = 'CROSS MARGIN, EXCHANGE, DEPOSIT OR WITHDRAWAL"
NO
PASS
YES
DECLARE LOCAL VARIABLES
SIZESIDE = FLOAT(SIZE) IF DIRECTION = IN ELSE FLOAT(SIZE) * -1
DATEINT = INT(DATE)
DATESTR = STR(DATE)
DATEUTC = UTC(DATEINT)
IF BIZTYPE DEPOSIT OR WITHDRAWAL
NO
DECONSTRUCT DICT "CONTEXT"
RETURN
SYMBOL :: STRING
ORDERID :: STRING
_ place holder for txId
YES
DECONSTRUCT DICT "CONTEXT"
RETURN
SYMBOL :: SYMSIDE FUNCTION CALL
ORDERID :: STRING
_ place holder for txId
DECLARE LOCAL VARIABLES
A_OR_B = IS_ASSET :: FUNCTION CALL
SIDE = ORDER_DIRECTION :: FUNCTION CALL
ASSET, BASEASSET, ASSETSIZE, BASEASSETSIZE BASEVALUE = BASSET :: FUNCTION CALL
ADD ASSET VALUES TO ASSET OBJECT TO HANDLE TRACKING THEIR TOTALS, GIVING US OUR PROFIT AND LOSS
IF ORDERID EXISTS IN ORDER DICTIONARY
TRUE
ADD TO ORDER OBJECT
ORDERID.ADD(ASSET, BASEASSET, ASSETSIZE, BASEASSETSIZE)
FALSE
CREATE ORDER OBJECT
ORDER(ASSET, BASEASSET, ASSETSIZE, BASEASSETSIZE, BASEVALUE)
IF ORDER OBJECTS BASEVALUE IS 0
FALSE
PASS
TRUE
ORDERID.ADD (BASEVALUE)
BASSET FUNCTION
BASSET
CALLS API TO GET THE USD VALUE OF THE BASE ASSET AT TIME OF TRADE
PARAMS
ASSET_OR_BASE :: BOOL
NAME :: STRING
SIZE :: FLOAT
DATE :: INT
BIZ_TYPE :: STRING
BIZTYPE "DEPOSIT" OR "WITHDRAWAL"?
FALSE
ASSET OR BASE ASSET?
ASSET
BASE ASSET
CALL API
BASEVALUE = GETPRICES(NAME,DATE)
TRUE
CALL API
BASEVALUE = GETPRICES(NAME,DATE)
CALCULATE USD VALUE OF ORDER FOR DEPOSIT TRANSACTIONS
BASESIZE = SIZE * BASEVALUE
IS ASSET BEING PROCESSED AN ASSET OR BASE ASSET?
ASSET
BASE ASSET
RETURN
NAME
"USD"
SIZE
BASESIZE
BASEVALUE
RETURN
"USD"
NAME
BASESIZE
SIZE
BASEVALUE
RETURN
NONE
NAME
0
SIZE
BASEVALUE
RETURN
ASSETNAME
NONE (for no base asset)
ASSETSIZE
0 (no size for base asset)
0 (for no base value)


GLOBAL VARIABLES
FILE_NAME :: STRING
PRICES :: DICTIONARY
CLIENT :: API CALL :: RETURNS LIST
RUN ME
PARAMS
NAME :: STRING
DATE :: INT
LOCAL VARIABLES
IGNORED ASSETS :: LIST
Dates are returned as a multiple of 60 (because KuCoin saves price history every 60 seconds, so our date must match that
IF NAME NOT IN IGNORED_ASSETS
TRUE
FALSE
RETURN 1
DATE :: FUNCTION CALL
ROUNDS DATE UP TO THE NEAREST MULTIPLE OF 60
PRICE = FUNCTION CALL FIND_PRICE(NAME, DATE)
FIND_PRICE
PARAMS
NAME :: STRING
DATE :: INT
SYMBOL = F"{NAME}-USDT"
END_AT = DATE * 60 * 60 * 24
END_AT IS NUMBER OF SECONDS IN A DAY
IF NAME IN STORED PRICES
FALSE
CREATE DICTIONARY FOR ASSET NAME WITHIN PRICES DICTIONARY
TRUE
ID DATE IN STORED PRICES[NAME][DATE]
FALSE
GET_DATA(SYMBOL, DATE, END_AT, NAME
TRUE
RETURN PRICE
GET_DATA
PARAMS
SYMBOL :: STRING
START :: INT
END :: INT
NAME :: STRING
WHILE NO ERROR IS RETURNED
TRY
API_DATA = API CALL
RETURNS
LIST
NO ERROR RETURNED
IS ERROR "TOO MANY REQUESTS"?
TRUE
PRINT
ERROR MESSAGE
LOOP
FOR 20 SECONDS
PRINT
"RETRYING"
TRUE
FALSE
PARSE API_DATA WITH PARSE_PRICES FUNCTION
PARSE_PRICES(API_DATA)
PRINT ERROR MESSAGE
SAVE ERROR MESSAGE IN LOG FILE
RETURN
1.010101
This value is a placeholder value to keep the program running, it can be updated manually in the exported csv's
PARSE_PRICES
PARAMS
API_DATA :: LIST
START :: INT
NAME :: STRING
LOCAL VARIABLE
OHLC_TO_RETURN :: FLOAT
FOR LOOP
FOR EACH ITEM IN API_DATA
OHLC = ROUND(I[1] + I[2] + I[3] + I[4] / 4),4)
DATE = INT(I[0])
STORE PRICES IN PRICE LIST
PRICES[NAME][STR(DATE)] = OHLC
TRUE
IF DATE == START
TRUE
OHLC_TO_RETURN = OHLC
IS THERE MORE DATA TO PARSE
IS OHLC_TO_RETURN NOT 0
TRUE
FALSE
PRINT ERROR
SAVE ERROR IN
ERROR_LOG
RETURN
1.001001001
(placeholder value)
RETURN
OHLC_TO_RETURN

Link to Presentation:

https://youtu.be/wPfuvu49w3U