Criterion C Development

**Use of existing tools (why, how, for every techniques)**

**Adequate for the task**

**（criteria和technique的对应关系）**

**Module organization**

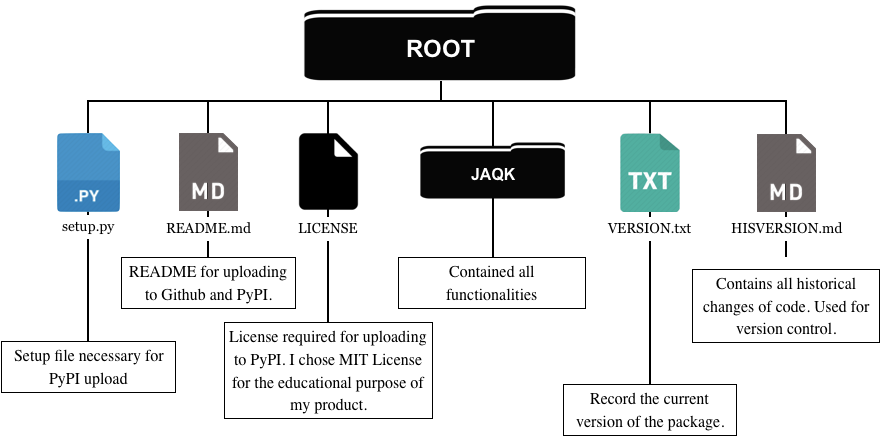
****

Figure 1: Organization of my product

This is what I have for uploading my module to PyPI. **JAQK** contains all functionalities (structure described in Criterion B). Other files are auxiliary for PyPI upload. They ensure the module can be accessed via “pip install JAQK” in command line.

**Organization of JAQK:**

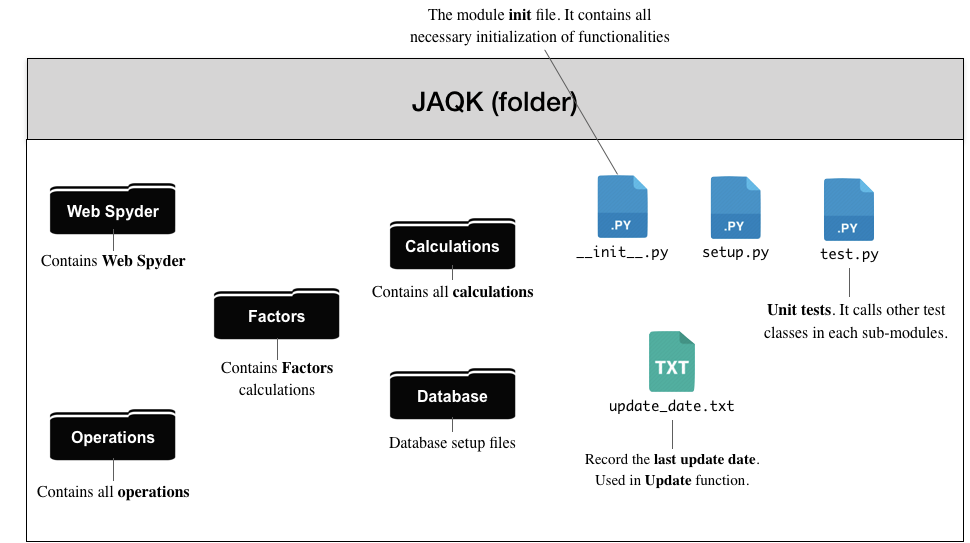


Figure 2: Organization of folder “JAQK” (where all functionalities locate)

Figure above is what Jerry will use in his PC’s Python IDE. **\_\_init\_\_.py** imports all functionalities in my module. High independency among modules ensure high extensibilities and low maintenance and version control work, and **Unit tests** further ensure integrity of all functionalities after code changes. (extensibility)

**Examples of Techniques used**

**1. Setup**

1.1 Covert file formats for PyPI uploading (‘.py’ to ‘.csv’)

1.2 GUI component for choosing a directory

**2. Web Spyder**

2.1 Asynchronous (non-blocking) main loop

2.2 Coroutine for **Getter**

2.3 HTML parsing (CSS selection)

2.4 Date calculations

2.5 Set operations

2.6 Read excel file into a CSV sheet

**3. Factors**

3.1 Using dictionary to reduce I/O

**4. Calculations**

4.1 Storing intermediate results to speed up program (cache)

**5. Operations**

5.1 Relative path

5.2 Regular expression for formatting

5.3 NumPy vectorization

5.4 API translation

**6. Unit tests**

**1. Setup**

**1.1 Convert file formats for PyPI uploading**

Existing tool used: pandas

I used pandas to convert database initialization sheets from ‘.py’ to ‘.csv’. Pandas can automatically read comma separated values in those ‘.py’ files and display them as sheets, so I can then save these sheets with CSV (‘.csv’ postfix) format.

There are four groups of ‘.py’ files I need to convert. For each one, I first create folder in the GUI chosen setup path, then read original ‘.py’ files. Then convert and save them using pandas, and delete original ‘.py’ files to free memory spaces.



Figure 3: One of four example of converting ‘.py’ to ‘.csv’ sheets

**1.2 GUI component for choosing a directory**

Existing tool used: PySimpleGUI

Jerry will use this GUI component to choose the setup path he wants, because it simplifies Jerry’s work of passing extra parameters and it simplifies my work in coding. Also GUI provides more accurate setup paths and can prevent possible mistakes (wrong paths, wrong folders, etc.). I use PySimpleGUI (a higher level wrap up of Tkinter) to simply the code work since a) I don’t need any advanced features in Tkinter and b) the GUI doesn’t need to be beautiful or very complicated.

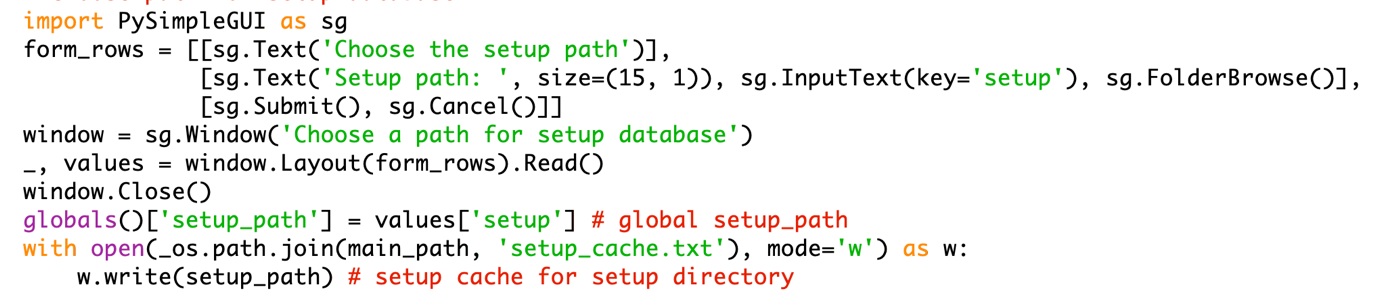


Figure 3: Example of GUI component for choosing a setup folder

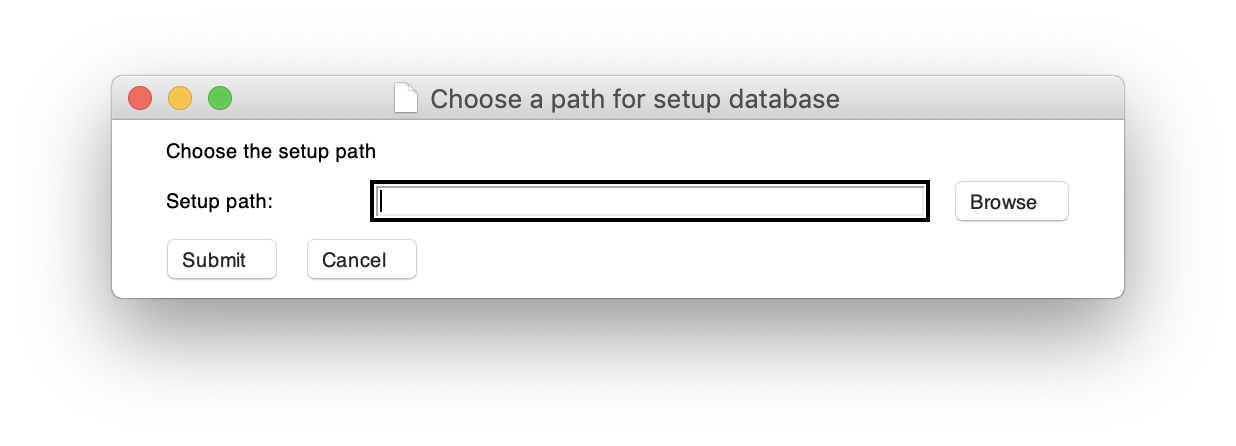


Figure 4: GUI layout for choosing path

**2. Web Spyder**

**2.1 Asynchronous (non-blocking) main loop**

Existing tool used: async

Using asynchronous main loop, **Getter** doesn’t need to wait for each request session before going next; instead, it performs other tasks during the waiting until the request is finished. Just like CPU scheduling, asynchronous main loop enables full usage of internet bandwidth, which very effectively speed up **Web Spyder**. Batches ensures that asynchronous main loop doesn’t burden computing power and internet bandwidth. I use “async” module in Python (internal) for this because it’s the easiest and most effective choice and it supports asynchronous request coroutine (explained later).

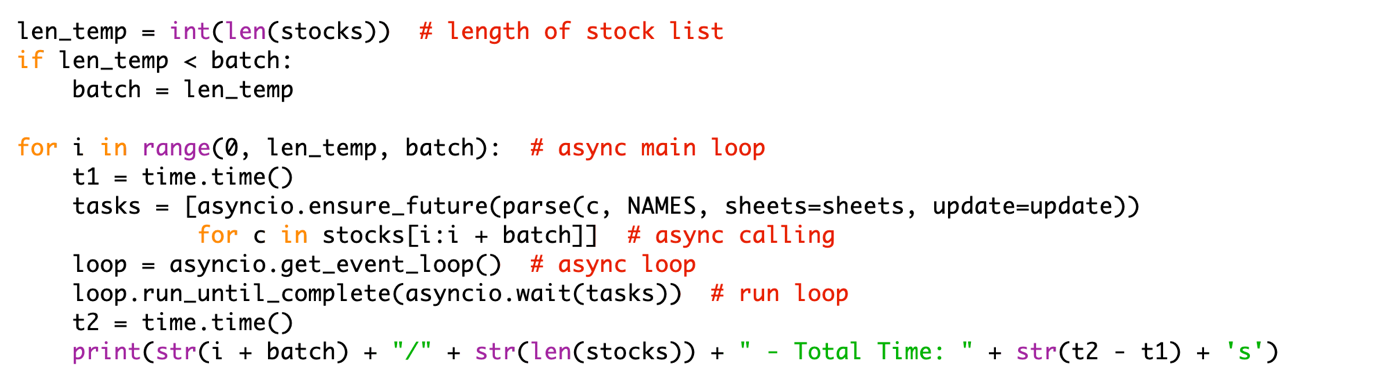


Figure 3: Asynchronous main loop

**2.2 Coroutine for Getter**

Existing tool used: aiohttp

Coroutine is executed by asynchronous main loop described above. Coroutine doesn’t need locks in multiprocessing, so it uses much fewer computer power than multiprocessing. Coroutine shares one only process which eliminates conflicting variables problem and the need of process lock, so it is much more efficient. In Web Spyder, internet requests take over 1 second to respond, so when using together with asynchronous main loop, the 1-seconnd requests can be awaited before the next coroutine starts. **（性能上的评估，用于没用的对比速度）**

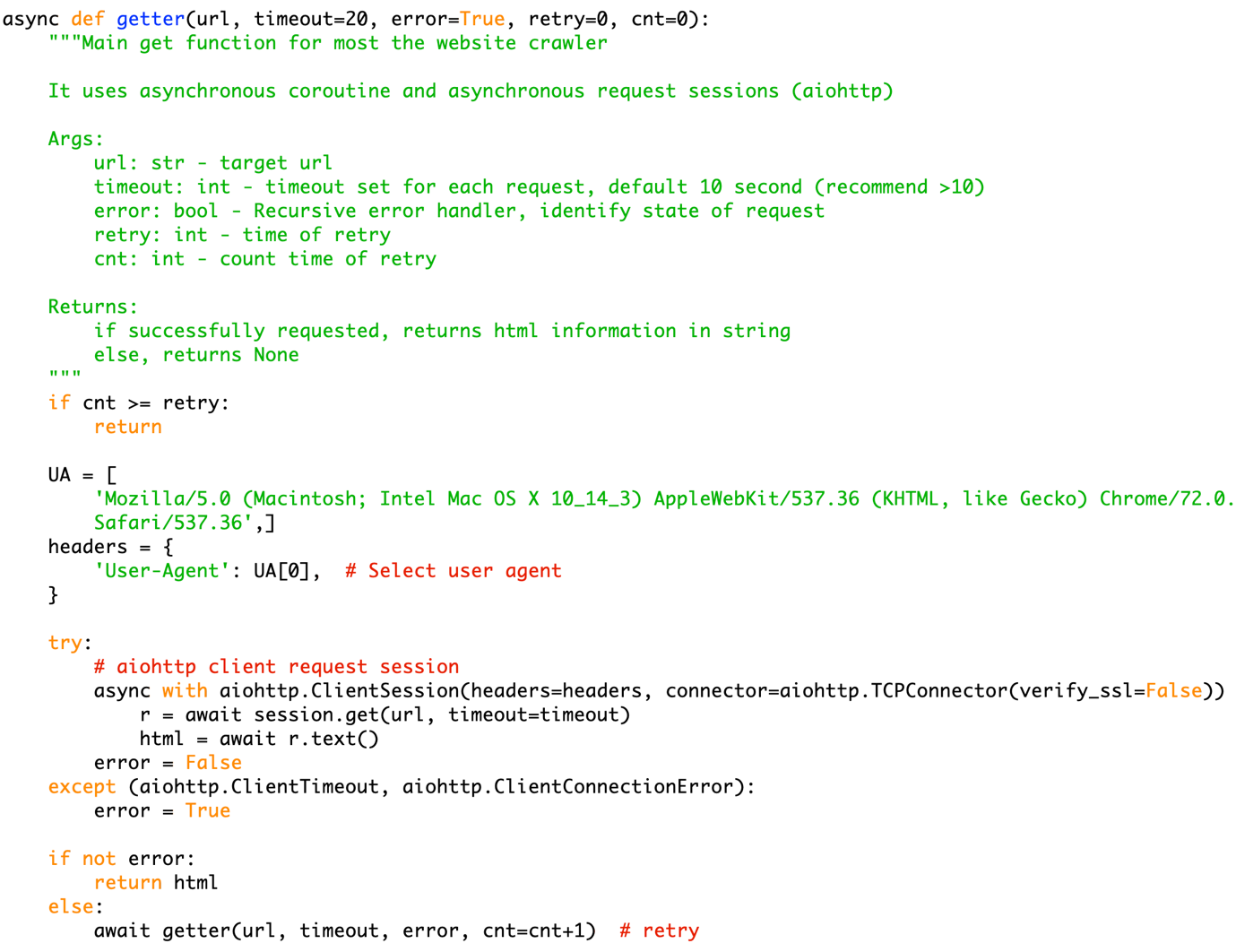


Figure 4: Example of coroutine request

“async” is a mark for asynchronous coroutine, and “aiohttp” is Python module for internet requests coroutine. “await” symbolizes awaiting coroutine to finish.

**2.3 HTML parsing (CSS selection)**

Existing tool used: pyquery

I used Python module “pyquery” for parsing. It enables efficient CSS selection. CSS selection has easier syntax rules than XPath. “pyquery” is more powerful than other alternative such as “beautifulsoup”, and syntax is easier to use. HTML parsing enables me to extract data from HTML texts collected by **Getter**. Each webpage has a unique parser and parsing rule, below is one example.

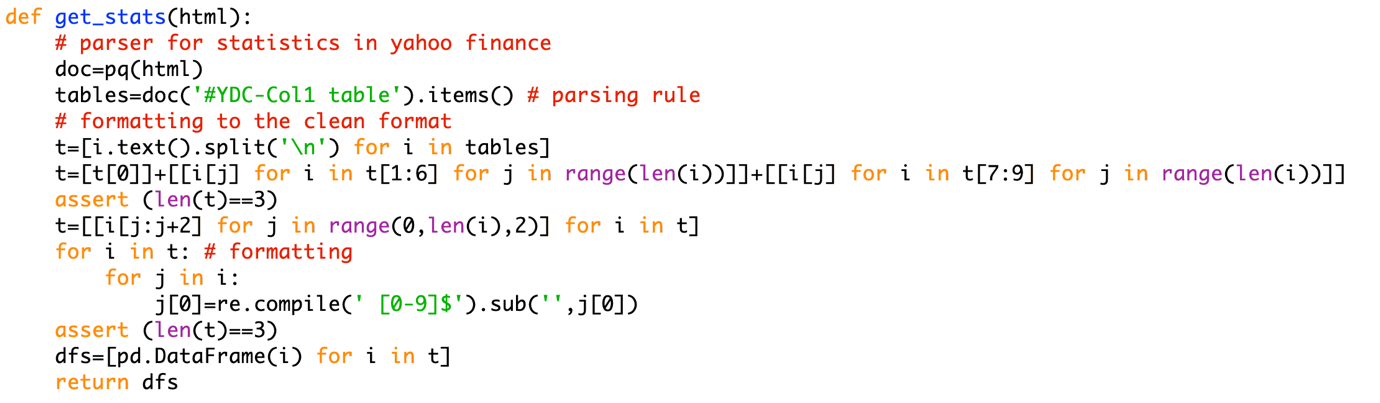


Figure 6: Example of one parser

Each webpage has a unique parser and parsing rule, above is one example.

**2.4 Date calculations**

Existing tool used: datetime

When updating database, the client needs the dates between today and last update in order to find stocks that need to be updated. I use “datetime” to compute these dates, which simplifies the calculation. An example is below:

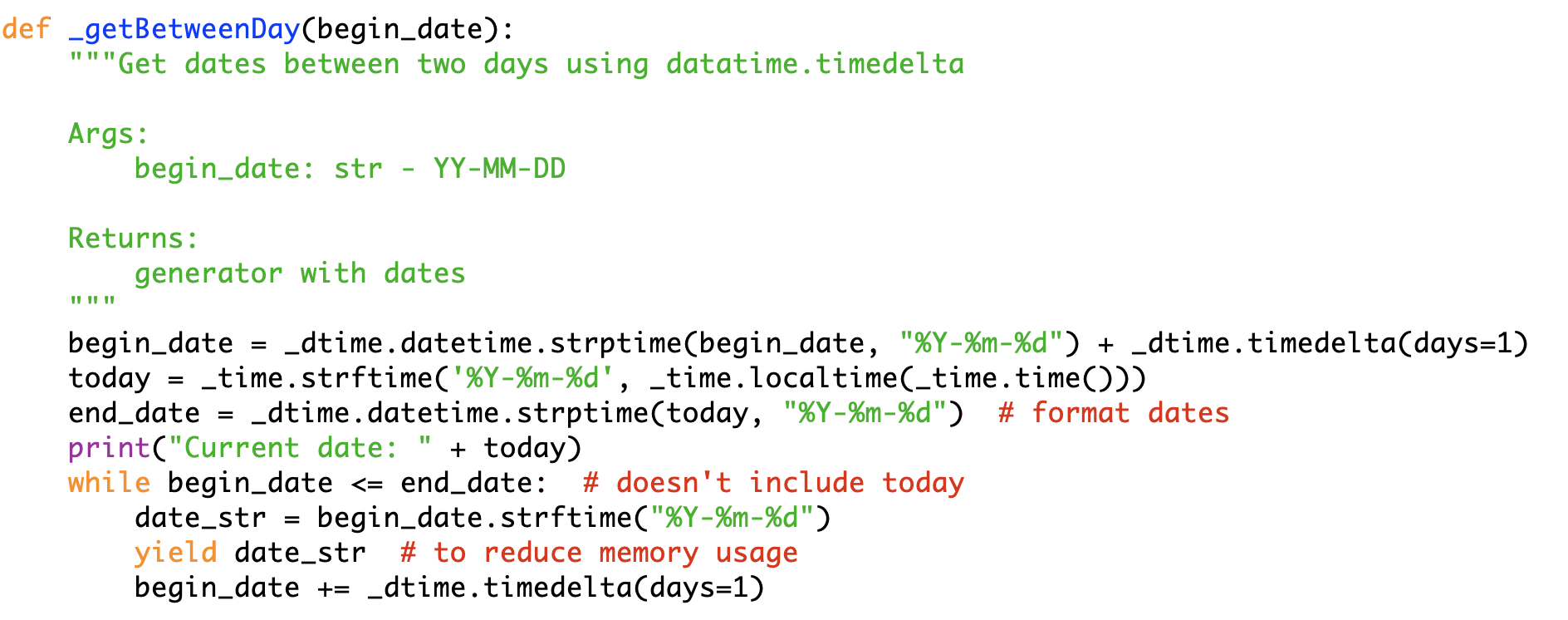


Figure 7: Example of operations with dates

**2.5 Set operations**

Set operations enable me to find companies that need to be updated, so the **Update** only updates companies needed by Jerry. （**Set 操作了什么，intersection是什么，set operation需要date的计算）** Python’s set type has internal optimization, so the work of finding intersections is very efficient in time and memory usage. It also simplifies code work that only calling set.intersection() will do the work.

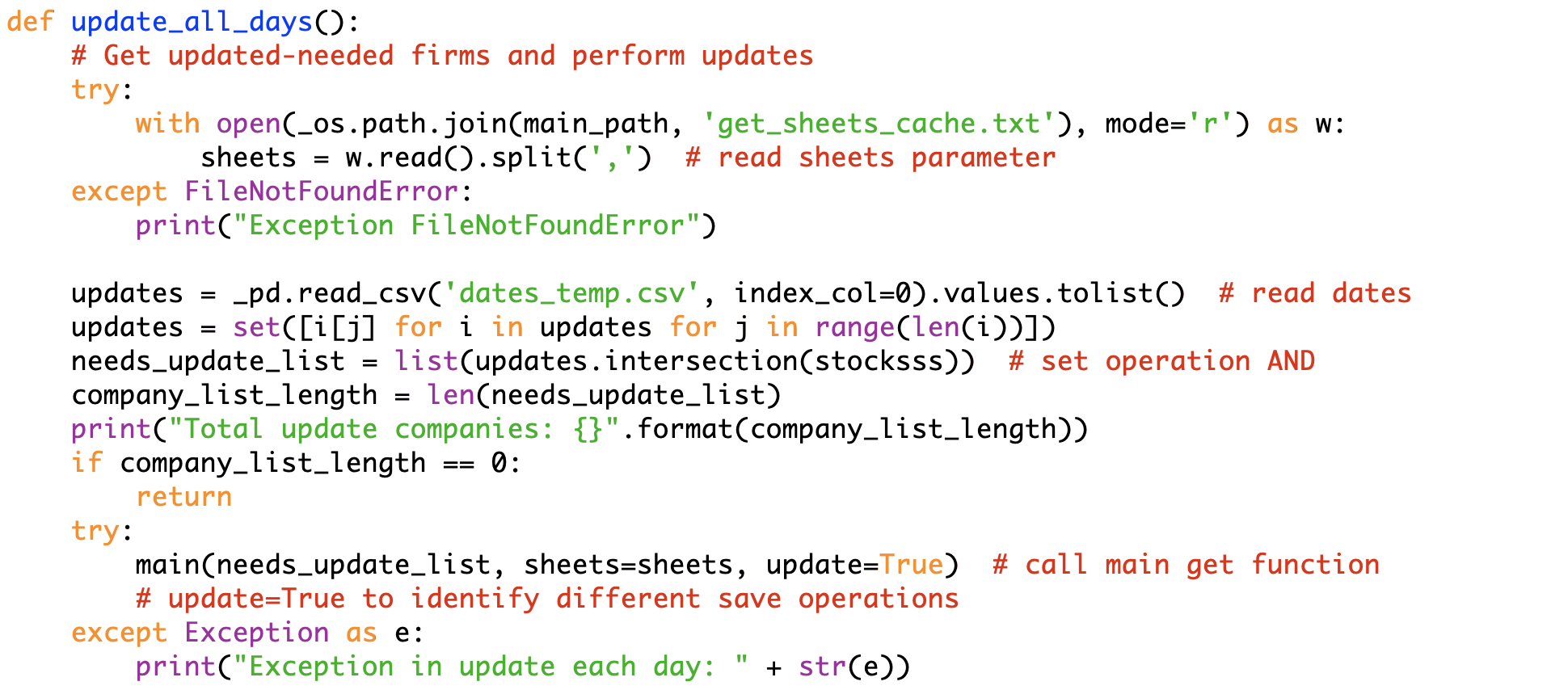


Figure 10: Example of a set operation

**2.6 Read excel file into a CSV sheet**

Existing tool used: pyopenxl

Reading a excel is different from reading a CSV sheet that excel files may have multiple sheets, but pandas can’t recognize that （**pandas做得到）**, so using pandas along can’t address the problem. Pyopenxl can recognize and read all sheets, and with pandas, all sheets will be merged into one CSV sheet.

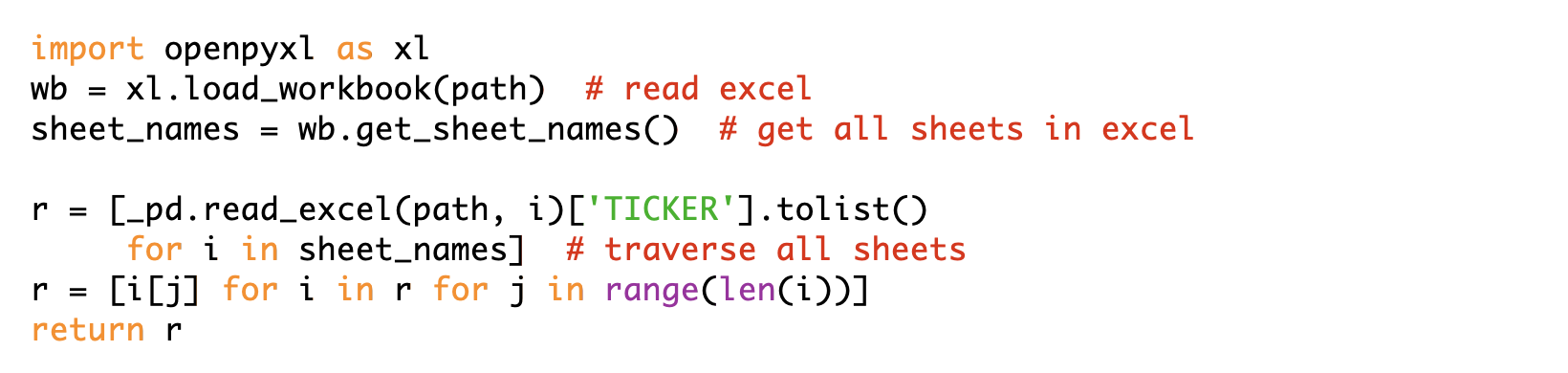
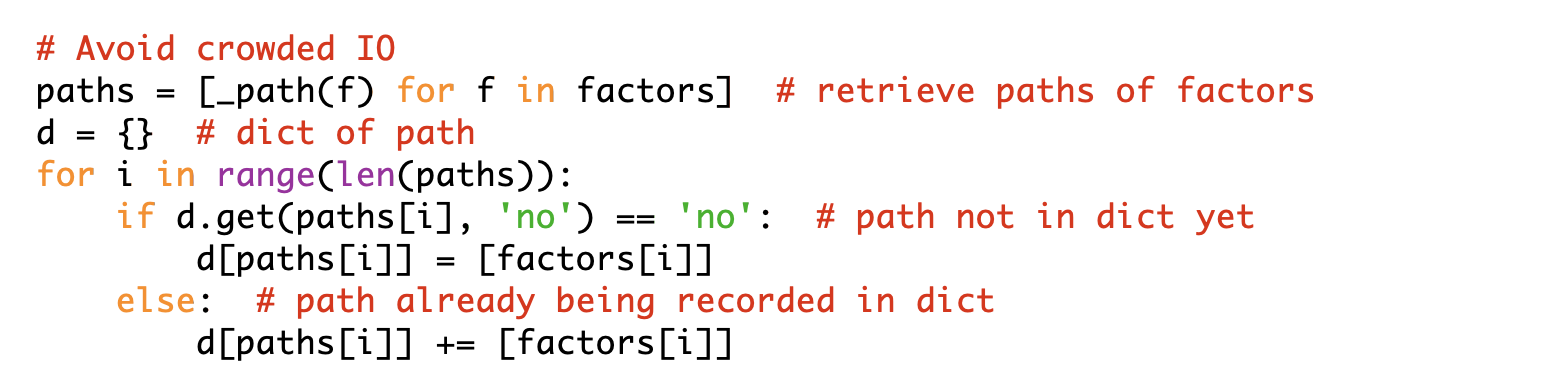


Figure 11: Example of reading a excel file

**3. Factors**

**3.1 Using dictionary to reduce I/O**

When collecting factors from data sheets, if without any special processing, **Open** would open a data sheet for each time a factor within is required. This wastes lots of time and memory. Alternatively, I group factors into a dictionary with format {sheet\_name1: [factor1, factor2, … ], sheet\_name2: [factor3, factor4], …}, **Such as Total Revenue, Gross Profit (factor是什么)**. This reduces opening sheets when there are lots of factors to locate. **Mapping的关系，**

 Figure 11: Example of using dictionary to reduce times of opening data sheets

**4. Calculations**

**4.1 Storing intermediate results to speed up program**

Calculations sometimes need to open all companies’ data sheets before giving out result, which usually take more than 2 seconds. Using complicated algorithms increase code work significantly. But using .csv sheets to store previous calculated data and previous results of each calculations can effectively speed up calculations because future calls of calculations could directly read the result from these .csv sheets. Example below use .csv sheets to speed up calculations.

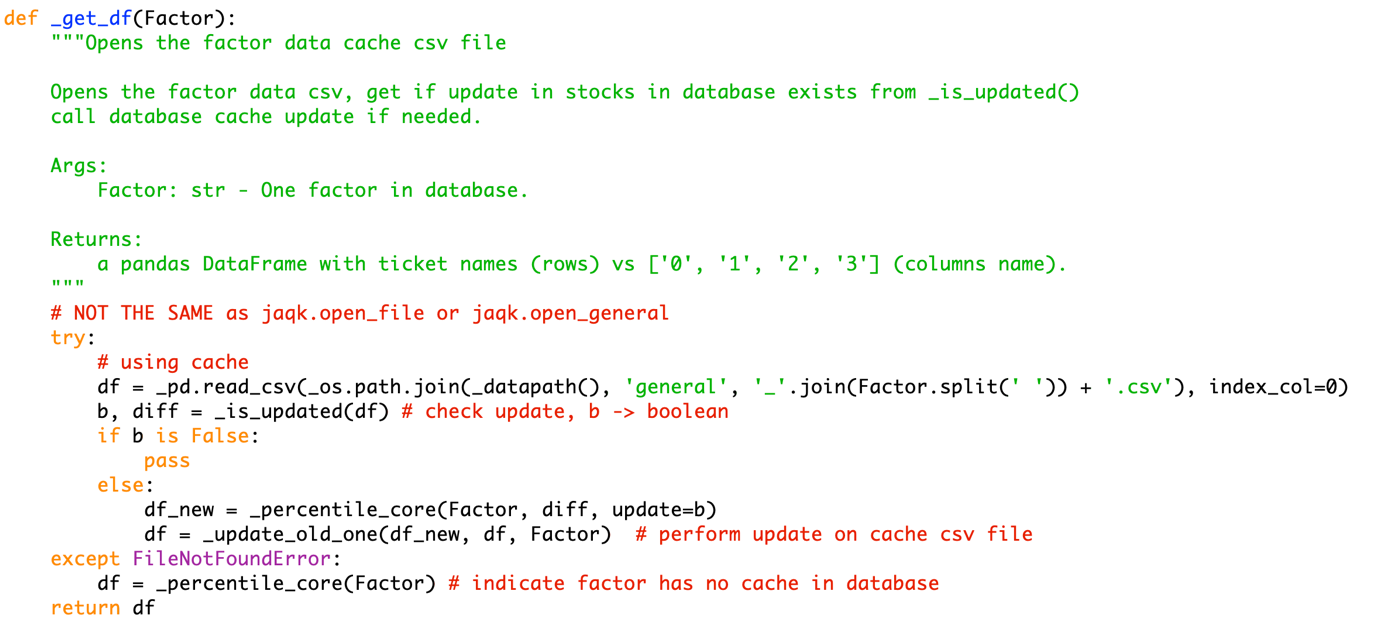


Figure 20: Example code of using .csv sheets cache for speeding up **Calculations**

I also use .txt files to store variables that need to be used for most functions in the module. “setup path” and “last update date” are two example of such usage. Using global variables may create errors when being used in a module or cross file usage. By using a .txt file, functions can obtain these parameters whenever needed without any possibility or error.

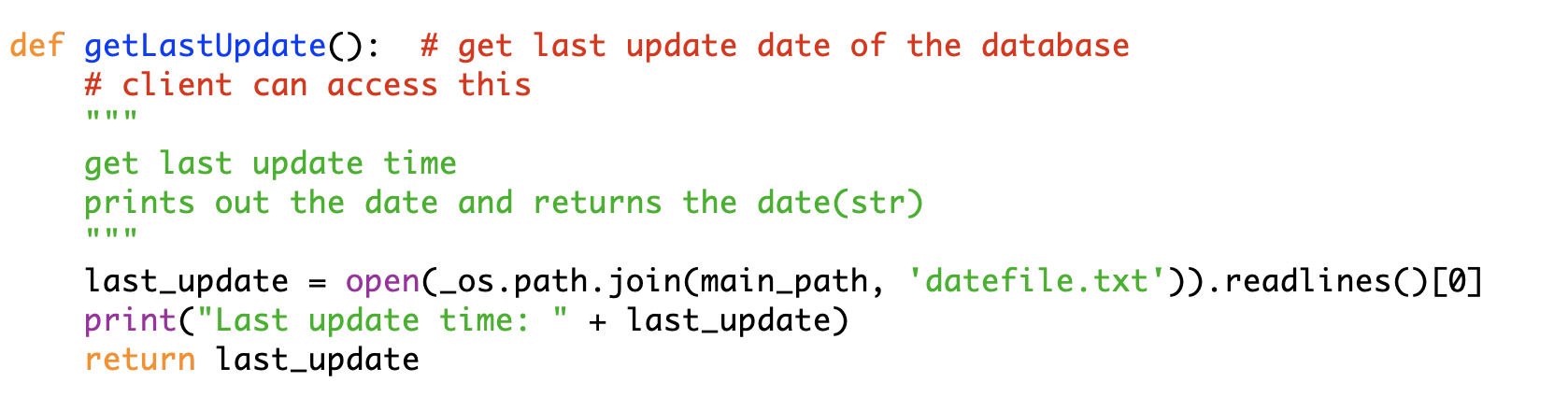


Figure 21: Example code of using a .txt file to retrieve parameter: last update date

**5. Operations**

**5.1 Relative path**

Absolute path has huge problems when the module is used in other’s PCs. Relative path ensures calling correct module and saving to correct paths. Relative path is constructed using os.path.join.

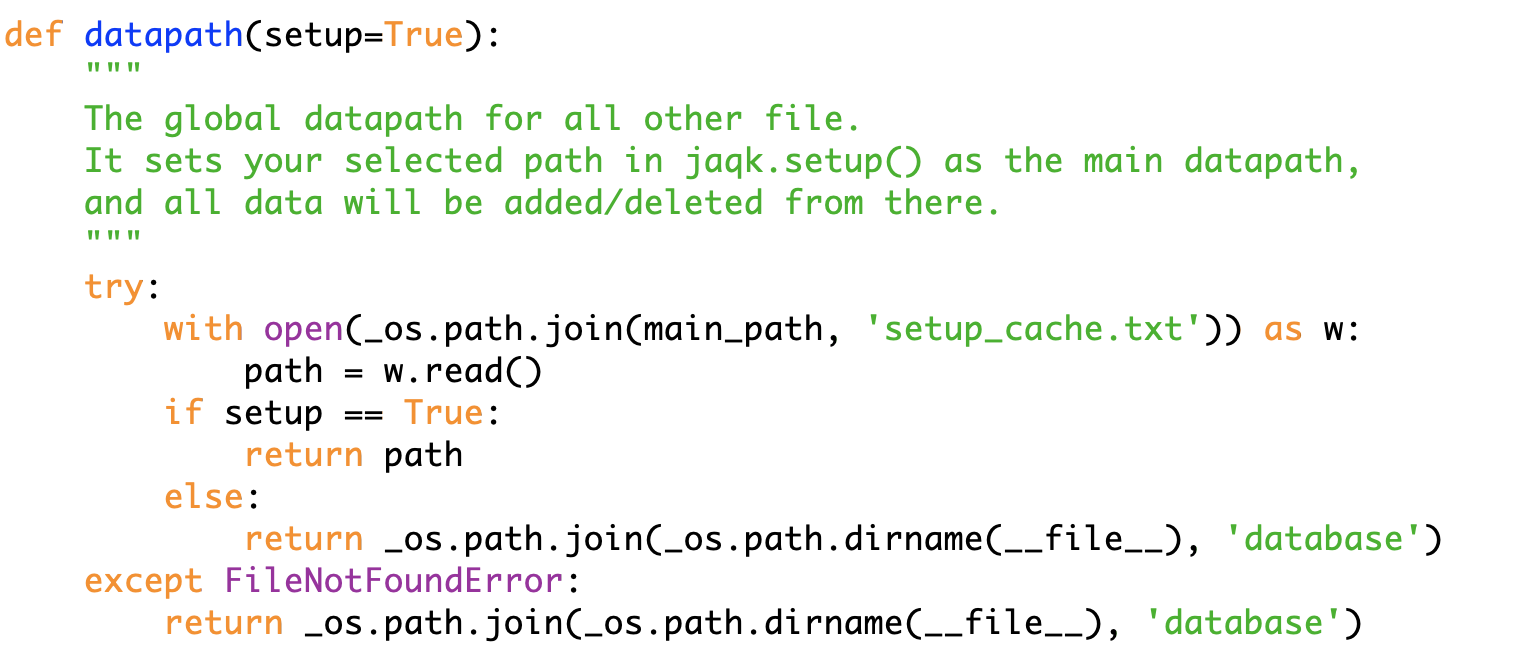


Figure 22: Example code of using relative path for path control

**5.2 Regular expression for formatting**

Existing tool used: re (python module for regular expression)

In formatting, I use regular expressions to locate decimals and money signs in numbers and substitute such characters with correct values. Regular expression can precisely and efficiently locate characters wanted with very concise syntax. Example below shows using regular expression to detect characters wanted from uncleaned numbers.

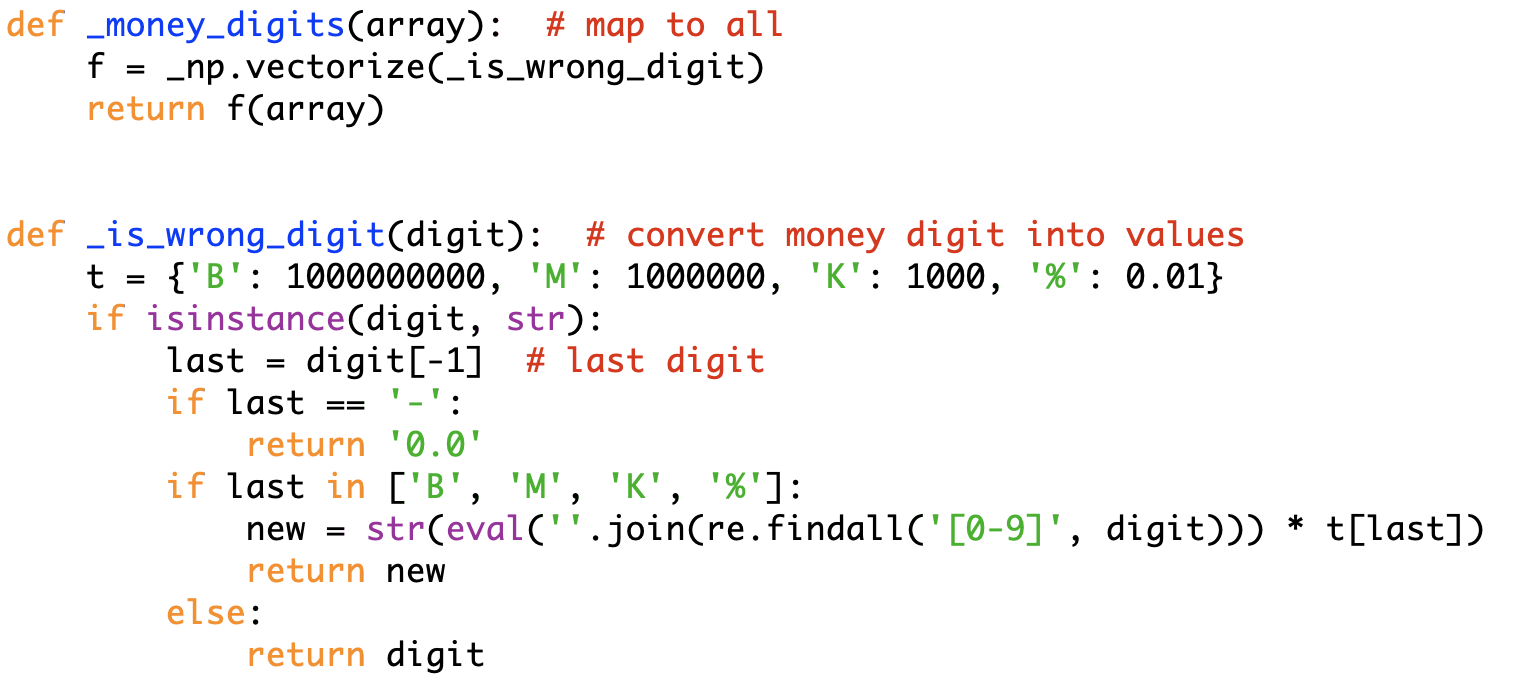


Figure 24: Example code of locating characters wanted using RE

**5.3 NumPy vectorization**

Existing tool used: numpy.vectorize

Data sheets contain 2-D data, so applying formatting rules to these numbers can either be done with two “for” loops that traverse all numbers and perform updates, or using some vectorization that map the formatting rule to the entire sheet. NumPy vectorization maps formatting rules to all data in a sheet. It’s much faster than two “for” loops and require fewer coding work. Plus, numpy vectorization map formatting rules to all data in an array, no matter the dimension of the array, so I don’t need extra code to deal with different input dimensions. **（加速的比较）**

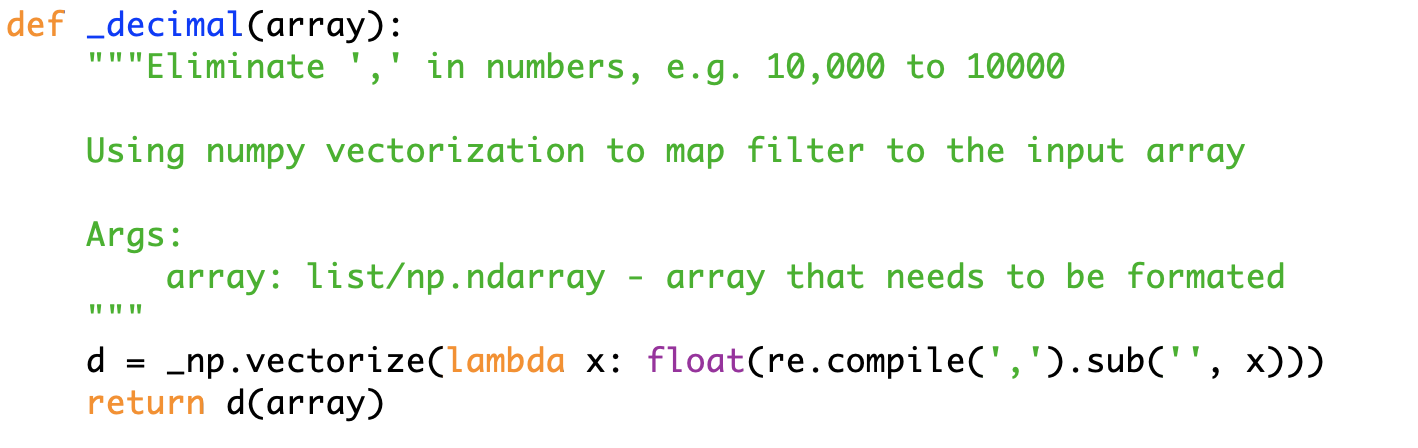


Figure 25: Example code of formatting (operations.Format.py)

**5.4 Baidu Translate API**

Existing tool used: Baidu Translate API

This is designed for the client to get a brief understanding of the company he is looking into. It uses translation API from BAIDU and English descriptions from Yahoo Finance. Baidu API is free and has very high accuracy (tested) and respond speed, so it’s the adequate technique for translating descriptions.



Figure 26: Example code of using Baidu Translation API

**5.5 List / dictionary comprehensions**

Python has list and dictionary comprehensions that increases processing speed and simplifies development. They make development more compatible to human thinking. I use them (with high priority than normal for / while loops) very often to speed up and shortens the code. **(why fast, HOW FAST)**

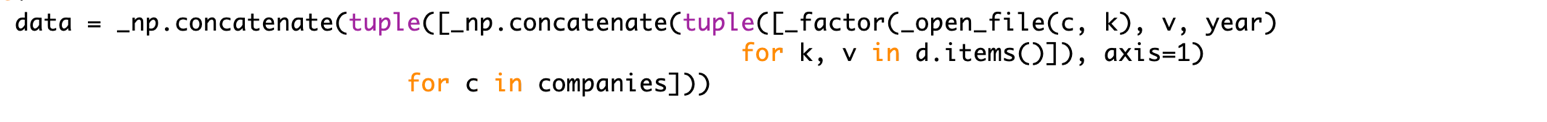


Figure 32: Example code of list comprehension

However, it’s easy to overuse them with the thought of making code fancy. I use them with cautious and to the correct extent.

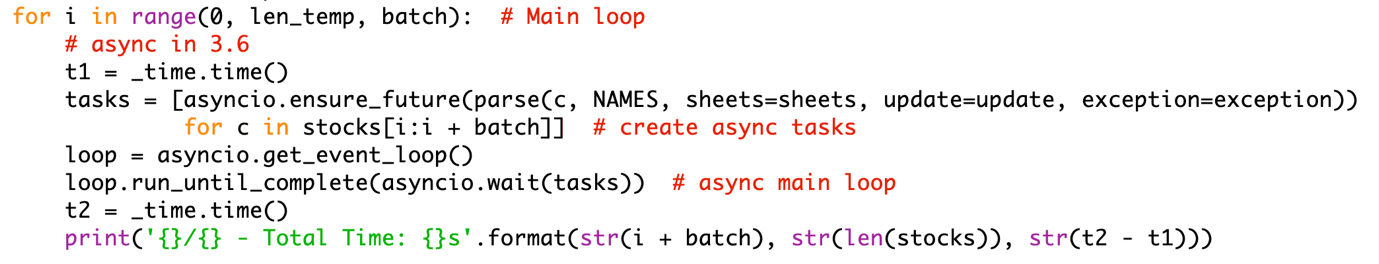


Figure 33: Example code of using for loop rather than comprehensions

In example above, although I can put all codes within the “for” loop into a ‘exec()’ call, I use for loop for better readability.

**6. Unit test**

Existing tool used: uinttest

Using Python internal unittest library, I perform unittest on all functionalities to further ensure the integrity of functionalities after possible code changes. Each sub-module in my product has its unittest script with testing case, and a unittest main function wraps them up and perform testing. Unit test enables extensibility of my module, since I won’t need to worry if future optimization of addition of functionalities may compromise current functionalities. Plus, unittest in Python supports testing of error message, so I can also test with incorrect parameters to make sure right error message pops out. **（和B中test plan对应）(5个module，sub-module)**

Example of unittesting:



Figure 29: Example code of unit tests in stocks (stocks.test\_stocks.py)



Figure 30: Example code of unit tests for Save operations (operations.test\_operations.py)

Above are two examples of 17 tests I have. They ensure every functionalities have correct output, correct errors raised, correct operations, and so on. This enables extensibility for future development.

Individual unit testing rules are collected in main test function that runs the test cases. This makes unit testing more efficient.

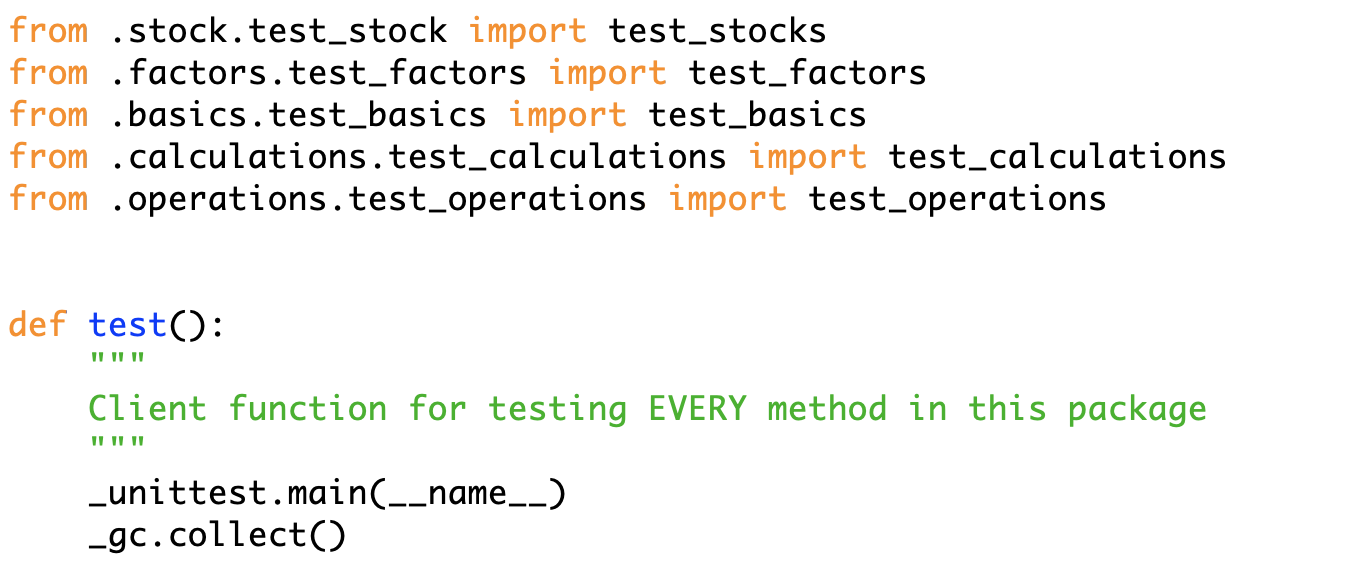


Figure 31: Example code of unit testing main function (test2.py)