QUANTEDGE CAPITAL
DEVELOPER INTERVIEW

SCHEDULE

PART I: PROGRAMMING EXERCISE (180 MINUTES)

TASK OVERVIEW

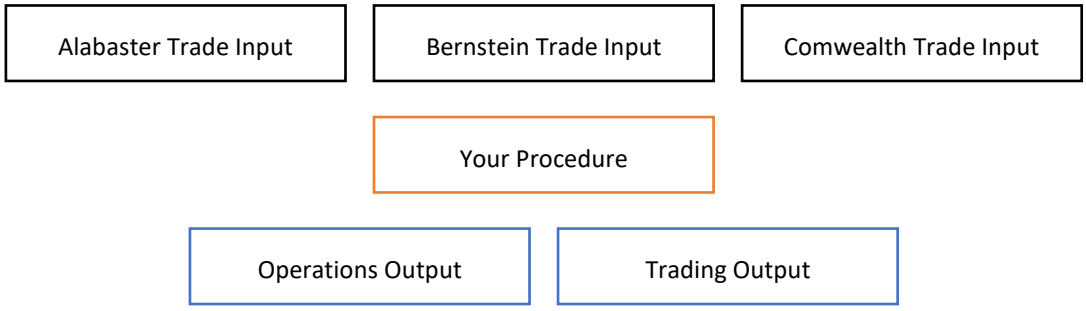
SCENARIO

Quantedge has just began trading a new line of derivative products called Zero Coupon Gross Domestic Product Swaps (GDPS). We trade this product with several brokers - Alabaster, Bernstein, and Comwealth - who provide us with trade reconciliation data at the end of each trading day. This data must be used by both Operations and Trading the following morning for a myriad of business-integral purposes from signal generation to portfolio valuation.

Your goal is to design a procedure in Python that:

- Takes as input trade reconciliation files from these multiple brokers
- Provides as output data files that meet each business line's requirements

An illustration of the different components is as follows:



These requirements are further detailed in the Specification section below.

SPECIFICATION

After the market closes on each day, brokers provide us with trade reconciliation files:

- input-alabaster/alabaster_20160101_tradelist.csv
- input-bernstein/bernstein_TradeInputList_160101.csv
- input-comwealth/trades_comwealth_jan2016.csv

These files contain fields that describe the following information:

- Security (str): a unique security identifier
- Date (str): date of observation
- Price (float): price of each security
- Quantity (float): position in each security
- Beta (float): end of day risk measure of each security

Note that brokers provide this information in different formats:

- Some brokers provide dates in different formats
- Some brokers name their columns differently
- Some brokers may not have all columns present
- Some brokers may have more than the necessary columns present
- Some brokers may quote quantities in different units

Also note that brokers have different identifiers for securities:

- A mapping table between different identifiers is provided
- Assume that Security is the master identifier

The following information may also be useful:

- Some securities may be held with only a subset of the brokers
- The brokers' reconciliation files only contain entries for securities that we have positions in

Business lines have the following requirements for their data files:

> **Operations**: Operations must identify price mismatches between different brokers in order to follow up on valuation discrepancies. Thus they require a table indexed by Security with the set of columns Price_[broker], Quantity_[broker] and Beta_[broker] for each broker.
>
> - one row for each Security for which there is a Price mismatch among any of the brokers
> - a column Date containing the day's date in the format YYYY-MM-DD
> - columns Price_[broker] for each broker which contain the Price from that broker
> - columns Quantity_[broker] for each broker which contain the Quantity from that broker
> - columns Beta_[broker] for each broker which contains the Beta from that broker
>
> **Trading**: Trading requires aggregate data for use the next day, which will be used to help make trading decisions. Thus they require a table indexed by Security with the columns AveragePrice, TotalQuantity and MaxBeta.
>
> - one row for each Security in the union of all unique securities across brokers
> - a column Date containing the day's date in the format YYYY-MM-DD
> - a column AveragePrice which contains the average Price across brokers
> - a column TotalQuantity which contains the sum of Quantity across brokers
> - a column MaxBeta which contains the maximum Beta across brokers

Before market opens the following day, we expect these outputs in the following folders:

- output-operations/output.csv
- output-trading/output.csv

This procedure should run from the command line using:

```
python main.py
```

The procedure should not require any manual intervention on a daily basis. You have complete liberty to make any implementation choices that you think will best solve the problem, but must respect the output requirements.

GRADING

In this scenario, the procedure will be run every day, will be shared with many team members, and over time may need to accommodate new input brokers and output business lines. Thus the following grading criteria will be used:

- **Clarity**: well-reasoned modularity, variable naming, documentation quality, overall coherence
- **Correctness**: strict adherence to output specifications, handling of missing values
- **Robustness**: handling of corner cases, exception handling
- **Design**: appropriate use of object oriented and / or functional design patterns, scalability

All four aspects will play a part in the evaluation of your code. There will be a verbal question and answer session after the programming exercise that will focus on these grading criteria.

RESOURCES

Feel free to consult any online resources and documents, including but not limited to Google, StackOverflow and official Python documentation.

TASK CLARIFICATIONS

Please ask us any questions you may have with regards to the programming exercise. You may do so at any point during the exercise.

IMPLEMENTATION

Implement the aforementioned requirements.

PART II: VERBAL EXERCISE (40 MINUTES)

CLARITY (10 MINUTES)

In a live trading day when things are moving quickly, we may be in a situation where we need a peer – even one not very well versed in programming – to debug your code on the fly. Consider a meeting in which you are explaining your solution to a group who are familiar with the scenario but are not familiar with programming.

1. Walk us through your code, explaining the overall design and touching on a few examples on how idiosyncrasies are handled. The use a whiteboard with diagrams is encouraged, and the use of jargon is discouraged.

CORRECTNESS (10 MINUTES)

It is not enough to provide the correct output given inputs that are error-free, as in the real-world broker files may contain a myriad of missing values. Error handling in situations where brokers provide missing prices, quantities or betas is not easy, and there may be outstanding corner cases that must be tested.

2. How are missing values currently handled in your procedure in each of the five columns? Are there any cases in which behavior is not well defined? How might this be improved?

ROBUSTNESS (10 MINUTES)

There is often a big gap between prototype code and production-ready code. Given the time constraint it is inevitable that your code may not be perfect and production-ready at present, and that is okay. What is important is that the team understands the existing shortcomings of the code, and knows how to improve.

3. What are the weakest areas of the code you have written that given the chance, you would like to improve before this code is put into production?

A common difficulty of munging broker data is that the file format need not stay constant over time. Even if your code works for this particular given set of inputs, the format of such inputs might suddenly change – perhaps columns get renamed or reordered, or perhaps dates representations get modified, or perhaps some days involved only a subset of brokers, all of which may cause errors during processing. In this way, non-stationary input data schemas are quite pernicious.

4. What would happen in your current code if a broker suddenly changed the format of his file? Name two realistic examples, and how your code is affected in each case. Is this behavior desirable? How might it be improved?

DESIGN (10 MINUTES)

In the future, we may wish to expand the scope of this procedure. Consider the case in which we wish to add 5 more trading partners who will each provide us with reconciliation files, and at the same time add 2 more internal data consumers to include not only operations and trading desks, but also research and risk desks.

5. How would you need to modify your code to accommodate these new brokers and new data consumers? If you were given more time what higher level architecture and design choices might you make differently to accommodate such changes in a scalable fashion?