In this task, you will create a simple mechanism for allocating resources named “locates”. Locates are requested by multiple clients. Those requests need to be aggregated and requested from an external API, which we do not control, that returns a list of **approved locates**. We should then distribute the approved locates to the clients that requested them. The approved locates may not fully cover the request. In this case, you will have to find a proper way to distribute the partial result between the requesting clients.

To simplify things, the requested locates are received from a CSV file of the following format:

`client_name, symbol, number_of_locates_requested`

For example, a file can be:

```
Client1, ABC, 300
Client2, QQQ, 100
Client2, ABC, 200
Client3, TTT, 100
```

Notice that each combination of client and symbol may only appear once.
Also, `number_of_locates_requested` is always a multiple of 100.

The request from the server is represented by the provided function:

`request_locates(requested_locates : dict[str, int]) -> dict[str, int]:`

Where both requested locates and returned result (**“approved locates**”) is a map from Symbols to int (the server is unaware of the requesting clients).

So, the expected flow of your function should be:

1. **Read all requests from the CSV file.**
2. **Aggregate all the requests for each symbol.**
3. **Call `request_locates` with the aggregated result.**
4. **Distribute the result to the clients.**

For simplicity's sake, the "distribution" can be done by printing out the result or writing it to a file in the same format as the received request (**`client_name, symbol, number_of_locates_approved`**).

<u>Distributing results to clients:</u>

The **`request_locates`** function may return only a subset of the requested locates. For example, when requesting:

```
{“SymbolA” : 500, “SymbolB” : 400, “SymbolC”: 700}
```
It may return:
```
{“SymbolA” : 500, “SymbolB” : 345}
```

Your logic will then have to distribute this result to the clients in a way you conceive as fair. Use the following guidelines:

1. **If only part of the request was approved, the distribution of the results should be proportional to the requested sum.**
2. **Since clients use the locates in chunks of 100, any chunk not divisible by 100 is less useful. So, for example, giving 200 to one client and 110 to another is better than giving 180 to the first and 130 to the second (even if the second distribution is more “fair”) since it provides 3 “round” chunks of 100.**
3. **All approved locates should be distributed.**
4. **No client should receive more than requested.**

Important: you may add additional considerations as you see fit. Try to consider as many edge cases as possible. If you see specific scenarios which require more complex logic to resolve, feel free to write down what they are and general ideas to resolve them without doing the full implementation.

For any questions please email:
dana.t@qspark.co
Or
nimrod.s@qspark.co