Ticker Stream

Endpoint

Create endpoints `/tickerStreamPart1` and `/tickerStreamPart2` that accepts a JSON payload over `POST` described below.

General Input

A comma-separated-value (CSV) stream of ticks in the format:

```
timestamp,ticker,quantity,price
```

For the purpose of this challenge:

1. `timestamp` will just be `hh:mm`, you can also treat this as a `string`.
2. `price` is any positive decimal value greater than `0.0`, we only need to handle 1 decimal place for input and output.

Part One

Input

At `/tickerStreamPart1`:

```
{
  "stream": ["timestamp,ticker,quantity,price", "timestamp,ticker,quantity,price"]
}
```

Output

```
{
  "output": ["explained,below", "explained,below"]
}
```

Aggregate the stream by time in chronological order, with each output record in the format:

```
timestamp,ticker1,cumulativeQuantity1,cumulativeNotional1,ticker2,cumu
```

Example: `00:00,A,5,5.5,B,4,4.4`

The group `ticker,cumulativeQuantity,cumulativeNotional` repeats for each ticker with a tick at the timestamp. Tickers should be sorted alphabetically as well.

The `notional` is the product of `quantity` and `price` at each tick, and the `cumulativeNotional` is the running sum of `notional` values for each ticker up till the timestamp.

Part Two

Input

At `/tickerStreamPart2`:

```
{
  "stream": ["timestamp,ticker,quantity,price", "timestamp,ticker,quantity,price"],
  "quantityBlock": 5
}
```

Output

```
{
  "output": ["explained,below", "explained,below"]
}
```

Aggregate the stream by time in chronological order, but this time each output record is 'delayed' by only reporting cumulative quantities in multiples (1 or more) of `quantityBlock`. The tickers reported at each timestamp should continue to be unique.

Take note that if only a portion of the current tick is applied for reporting the next quantity block, the `notional` calculation should factor the correct quantity, with the leftover quantity effectively 'hidden' from the true `cumulativeNotional`.

Example:

```
quantityBlock: 5

Input:
```

```
[
   "00:06,A,1,5.6",
   "00:05,A,1,5.6",
   "00:00,A,1,5.6",
   "00:02,A,1,5.6",
   "00:03,A,1,5.6",
   "00:04,A,1,5.6"
]
```

```
Result:

[
   "00:05,A,5,28.0"
]
```

`"00:06,A,1,5.6"` is not included in the output as the cumulative quantity is not a multiple of `quantityBlock`, which is `5`.

Scoring

Each evaluation attempt will give a score of `125` if the output is correct, else `0`.

There will be eight evaluation attempts, summing to a maximum of `1000` before that raw score is divided by 10 (rounding down).