# Monthly Bill Generator - Solution Explanation

## Problem Overview

The problem requires us to develop a function that generates a monthly bill based on a list of items. Each item has various attributes including an item code, rate, quantity, and a date range during which it’s active. The main challenge is to properly calculate charges for items that are active during the target month and group similar items together.

## My Approach

When I first looked at this problem, I broke it down into a few key steps:

1. Figure out which items are actually active in the target month
2. Calculate how much to charge for each active item
3. Group similar items together
4. Add everything up for the final bill

### Active Item Detection

For the first step, I needed to determine if an item is active during the target month. This is basically checking if the item’s date range overlaps with our target month. An item is considered active if: - Its end date is on or after the first day of the target month, AND - Its start date is on or before the last day of the target month

Here’s how I implemented this:

# Skip items that aren't active in our target month  
if end < first\_day or start > last\_day:  
 continue

### Calculating Charges

Once I know an item is active, I have to figure out exactly how many days in the month it’s active for. This gets a bit tricky because an item might only be active for part of the month. I calculate this by finding the intersection between the item’s date range and the target month:

# Find the actual billing period for this month  
start\_date = max(start, first\_day)  
end\_date = min(end, last\_day)  
  
# Calculate how many days this item is active  
active\_days = (end\_date - start\_date).days + 1

Then I prorate the charge based on the number of active days:

# Calculate prorated amount  
daily\_rate = (rate \* qty) / days\_in\_month  
amount = daily\_rate \* active\_days

### Grouping Logic

The requirements said we need to group items with the same: - Item code - Rate - Billing period

So I created a unique key for each combination:

key = (item["item\_code"], rate, billing\_period)

And then added quantities and amounts for items with the same key:

if key in grouped:  
 grouped[key]["qty"] += qty  
 grouped[key]["amount"] += amount  
else:  
 grouped[key] = {  
 "qty": qty,  
 "amount": amount  
 }

### Handling Data Inconsistencies

One thing I noticed in the sample data was that some fields like rate and qty were sometimes strings and sometimes numbers. I handled this by explicitly converting them:

rate = float(item["rate"])  
qty = int(item["qty"])

## Edge Cases I Considered

1. **Zero-rate items**: I excluded items with a rate of 0 since they don’t contribute to the bill.
2. **Different month lengths**: I made sure my code handles months with different numbers of days correctly.
3. **Data type inconsistencies**: As mentioned above, I handled the mixed string/number values.
4. **Items active for partial months**: I correctly prorated charges for items that aren’t active for the entire month.

## Testing

I tested my solution with the provided sample data for November 2024. My code successfully: - Identified all active items - Grouped them according to the specified criteria - Calculated the correct prorated amounts - Generated the bill with the expected format

## Conclusion

This was an interesting problem that required careful attention to detail, especially regarding date calculations and data handling. My solution efficiently processes the input data and produces a properly formatted bill with grouped line items and the correct total revenue.

If I were to improve this further, I might add more robust error handling for edge cases like invalid dates or missing fields in the input data.