Monthly Billing System in Python

# 1. Overview

This document describes a Python function that creates monthly billing reports for rented equipment. The function figures out how much to charge for each item by checking how many days it was in use during the billing month

# Required Libraries

The following Python libraries are used:

- datetime: For handling date parsing and arithmetic.

- collections.defaultdict: For grouping line items by item and period.

- calendar: For calculating number of days in the month.

# 3. Function Logic

1. **Set Up the Billing Period:**  
    The function starts by parsing the target\_month (in the format YYYY-MM) and determines the exact start and end dates of that month. This defines the window for calculating active rentals.
2. **Process Each Rental Item:**  
    For every item in the list, it reads the rental’s start\_date and stop\_date. If the stop date comes before the start date, the item is skipped to avoid invalid data.
3. **Calculate Active Usage:**  
    It figures out how many days the item was actually in use during the target month, even if the rental began before or ended after that month. Only the days within the month are counted.
4. **Prorate Charges:**  
    Based on the number of active days, the function calculates a prorated charge for that item. However, the quantity of items rented is treated as a fixed number—it’s not adjusted based on days.
5. **Group and Summarize Data:**  
    Items are grouped together based on shared characteristics like item\_code, rental rate, and billing\_period. For each group, the total amount and total quantity are calculated.
6. **Finalize the Bill:**  
    Before generating the final report, all amounts are rounded for clarity. The function then compiles everything into a structured bill, including the total revenue for the month.

# · Rent Calculation Logic

To calculate prorated rent for an item within the target month:

• `total\_days` = Total rental duration from start\_date to stop\_date (inclusive)

• `active\_days` = Number of days the item was active within the target month

• `daily\_amount` = total amount / total\_days

• `prorated\_amount` = daily\_amount \* active\_days

Note: Quantity remains unchanged and is added directly to the grouped total.

# 5. Sample Function Code

def generate\_monthly\_bill(item\_list: list, target\_month: str) -> dict:  
 from datetime import datetime  
 from collections import defaultdict  
 import calendar  
  
 month\_start = datetime.strptime(target\_month + "-01", "%Y-%m-%d")  
 days\_in\_month = calendar.monthrange(month\_start.year, month\_start.month)[1]  
 month\_end = month\_start.replace(day=days\_in\_month)  
  
 grouped = defaultdict(lambda: {"qty": 0.0, "amount": 0.0})  
 total\_revenue = 0.0  
  
 for item in item\_list:  
 try:  
 item\_start = datetime.strptime(item["start\_date"], "%Y-%m-%d")  
 item\_stop = datetime.strptime(item["stop\_date"], "%Y-%m-%d")  
  
 if item\_stop < item\_start:  
 continue  
  
 active\_start = max(month\_start, item\_start)  
 active\_end = min(month\_end, item\_stop)  
  
 if active\_start > active\_end:  
 continue  
  
 total\_days = (item\_stop - item\_start).days + 1  
 active\_days = (active\_end - active\_start).days + 1  
  
 qty = float(item["qty"])  
 amount = float(item["amount"])  
 rate = float(item["rate"])  
  
 daily\_amount = amount / total\_days  
 prorated\_amount = daily\_amount \* active\_days  
  
 billing\_period = f"{active\_start.date()} to {active\_end.date()}"  
 group\_key = (item["item\_code"], rate, billing\_period)  
  
 grouped[group\_key]["qty"] += qty  
 grouped[group\_key]["amount"] += prorated\_amount  
  
 except (ValueError, TypeError, ZeroDivisionError):  
 continue  
  
 line\_items = []  
 for (item\_code, rate, billing\_period), data in grouped.items():  
 qty\_rounded = round(data["qty"], 2)  
 amount\_rounded = round(data["amount"], 2)  
  
 line\_items.append({  
 "item\_code": item\_code,  
 "rate": rate,  
 "qty": qty\_rounded,  
 "amount": amount\_rounded,  
 "billing\_period": billing\_period  
 })  
  
 total\_revenue += amount\_rounded  
  
 return {  
 "line\_items": line\_items,  
 "total\_revenue": round(total\_revenue, 2)  
 }

# 6 .Output:

The output of sample testcase given the problem statement for january 2024 is as shown in Fig 01

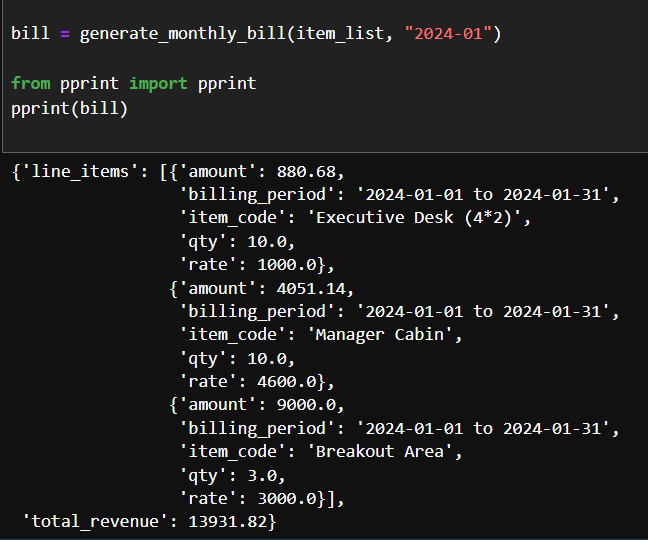


Fig 1 : output of sample testcase