def safe\_float(value):

"""Convert to float, return None if invalid."""

try:

return float(value)

except ValueError:

return None

def format\_currency(amount):

return f"Rs. {int(amount):,}"

try:

sales = []

with open("sales\_data.txt", "r") as f:

for line in f:

try:

tid, product, category, price, qty, salesman = line.strip().split(",")

price = safe\_float(price.strip())

qty = safe\_float(qty.strip())

if price is None or qty is None:

print(f"Skipping invalid line: {line.strip()}")

continue

sales.append({

"id": tid.strip(),

"product": product.strip(),

"category": category.strip(),

"price": price,

"qty": qty,

"salesman": salesman.strip()

})

except ValueError:

print(f"Skipping malformed line: {line.strip()}")

# Step 2: calculate revenue per salesman

revenue\_dict = {}

for s in sales:

revenue\_dict[s["salesman"]] = revenue\_dict.get(s["salesman"], 0) + (s["price"] \* s["qty"])

# Step 3: filter salesmen with revenue > 50,000

filtered = dict(filter(lambda item: item[1] > 50000, revenue\_dict.items()))

# Step 4: map into required structure

mapped = list(map(

lambda kv: {"salesman": kv[0], "total\_revenue": format\_currency(kv[1])},

filtered.items()

))

# Step 6: sort by revenue (descending)

mapped.sort(key=lambda x: int(x["total\_revenue"].replace("Rs. ", "").replace(",", "")), reverse=True)

# Step 5: write to file

with open("top\_salesmen.txt", "w") as f:

for entry in mapped:

f.write(f"{entry['salesman']}, {entry['total\_revenue']}\n")

print("Top salesmen written to top\_salesmen.txt")

except FileNotFoundError:

print("Error: sales\_data.txt not found!")