AI ASSISTED CODING
LAB EXAM-2

SHIVA SAI.B
2403A51257
BATCH-11

Subgroup A:

A.1 — [S18A1] Compute per-probe average from logs (AI completion)

Context:

You are integrating a real estate listings platform telemetry service where each probe emits periodic measures as CSV lines: `id,timestamp,ppm`. Some lines may be truncated or contain non-numeric values. Ops needs a quick aggregation for dashboards and alert thresholds.

Your Task:

Write a Python function to parse the raw text (multiple lines) and compute per-probe averages of `ppm`. Return a dict {id: avg} and separately compute an overall average.

Data & Edge Cases:

Input contains newlines, optional leading/trailing spaces, and may include malformed rows. Timestamps are ISO-8601 but not needed for math.

AI Assistance Expectation:

Use AI code completion to scaffold the loop, dictionary accumulation (sum and count), and exception handling for malformed rows.

Constraints & Notes:

Prefer O(n) pass; ignore lines that cannot be split into three fields or have non-numeric metric; round averages to 2 decimals.

CODE:

C: > Users > bavir > OneDrive > Desktop > exam.py > ...

```python
def compute_probe_averages(raw_text):
    probe_sums = {}
    probe_counts = {}
    total_sum = 0.0
    total_count = 0

    for line in raw_text.strip().split('\n'):
        line = line.strip()
        if not line:
            continue
        parts = line.split(',')
        if len(parts) != 3:
            continue
        probe_id, _, ppm_str = parts
        try:
            ppm = float(ppm_str)
        except ValueError:
            continue
        probe_sums[probe_id] = probe_sums.get(probe_id, 0.0) + ppm
        probe_counts[probe_id] = probe_counts.get(probe_id, 0) + 1
        total_sum += ppm
        total_count += 1

    averages = {pid: round(probe_sums[pid] / probe_counts[pid], 2) for pid in probe_sums}
    overall_avg = round(total_sum / total_count, 2) if total_count else None
    return averages, overall_avg

# Example usage:
sample = """pr181,2025-02-01T08:00,22.5
pr182,2025-02-02T09:00,24.2
pr183,2025-02-03T10:00,25.9
"""

result, overall = compute_probe_averages(sample)
print(result)      # {'pr181': 22.5, 'pr182': 24.2, 'pr183': 25.9}
print("overall_avg=", overall)  # overall_avg=24.2
```

INPUT/OUTPUT:

```
PS C:\Users\bavir> & "C:/Program Files/Python313/python.exe" c:/Users/bavir/OneDrive/Desktop/exam.py
{'pr181': 22.5, 'pr182': 24.2, 'pr183': 25.9}
overall_avg= 24.2
PS C:\Users\bavir>
```

WHAT THIS DOES:

This code defines a function compute_probe_averages that processes raw CSV telemetry data for real estate probes. Here's what it does:

1. Initializes dictionaries and counters to track sums and counts per probe, as well as overall totals.
2. Loops through each line of the input text, stripping whitespace and skipping empty lines.

3. Splits each line by commas and checks for exactly three fields (probe ID, timestamp, ppm value).
4. Attempts to convert the ppm value to a float; if conversion fails (malformed or non-numeric), the line is skipped.
5. Accumulates sums and counts for each probe and overall.
6. Calculates per-probe averages (rounded to 2 decimals) and the overall average.
7. Returns a dictionary of probe averages and the overall average.
8. The sample usage demonstrates the function with example data and prints the results.

Malformed or incomplete lines are ignored, ensuring robust aggregation.

Summary:

The already handles all the specified edge cases and constraints:

- **Newlines and spaces:** It strips each line and skips empty ones.
- **Malformed rows:** It checks for exactly three fields and uses try/except to ignore non-numeric ppm values.
- **Timestamps:** It ignores the timestamp field.
- **O(n) pass:** It processes each line once in a single loop.
- **Rounding:** Averages are rounded to 2 decimals.
- **Dictionary accumulation:** Uses dictionaries for sum and count per probe.

Summary:
Your implementation meets all requirements for edge cases, constraints, and performance. No changes are needed.

A.2 — [S18A2] Implement ListingWindow with add/remove/summary (AI completion)

Context:

A microservice in the real estate listings platform platform maintains a small in-memory structure to track values keyed by identifier (e.g., alert IDs, machine IDs, listing IDs). Engineers want a minimal class to add, remove, and summarize current values for quick health checks.

Your Task:

Implement a `ListingWindow` class with methods `add(id: str, value: float)`, `remove(id: str)`, and `summary() -> tuple[int, float|None]` returning (count, average).

Data & Edge Cases:

IDs are unique keys. Re-adding the same ID overwrites its value. Removing a missing ID should be safe (no exception). For an empty store, average is None.

AI Assistance Expectation:

Ask AI to generate the class skeleton with docstrings and type hints, then refine method behavior and add a quick usage example.

Constraints & Notes:

Keep state in a dict; O(1) per operation; return rounded average to 2 decimals (when non-empty).

Code:

C: > Users > bavir > OneDrive > Desktop > exam2.py > ListingWindow > summary

```python
from typing import Dict, Tuple, Optional

class ListingWindow:
    """
    Minimal in-memory tracker for values keyed by unique IDs.
    Supports add, remove, and summary operations for health checks.
    """
    def __init__(self):
        self._store: Dict[str, float] = {}

    def add(self, id: str, value: float) -> None:
        """
        Add or update the value for a given ID.
        """
        self._store[id] = value

    def remove(self, id: str) -> None:
        """
        Remove the value for a given ID. Safe if ID is missing.
        """
        self._store.pop(id, None)

    def summary(self) -> Tuple[int, Optional[float]]:
        """
        Returns a tuple of (count, average value rounded to 2 decimals).
        If empty, average is None.
        """
        count = len(self._store)
        if count == 0:
            return (0, None)
        avg = round(sum(self._store.values()) / count, 2)
        return (count, avg)

# Usage example:
if __name__ == "__main__":
    lw = ListingWindow()
    lw.add('a1', 12)
    lw.add('b2', 17)
    lw.remove('a1')
    lw.add('c3', 9)
    print(lw.summary())  # Output: (2, 13.0)
```

INPUT/OUPUT:

Here's how the code addresses each requirement:

- IDs are unique keys. Re-adding overwrites:
  The internal state is a dictionary (self. store: Dict[str, float]). Adding with the same ID updates the value, overwriting any previous entry.
- Removing a missing ID is safe:
  The remove method uses pop(id, None), which does nothing if the ID isn't present—no exception is raised.
- Empty store, average is None:
  In summary, if the store is empty (count == 0), it returns (0, None).
- Class skeleton, docstrings, type hints:
  The class and methods have docstrings and type hints for clarity and maintainability.
- State in dict; O(1) operations:
  All operations (add, remove, summary) use dictionary methods, which are O(1).
- Average rounded to 2 decimals:
  The average is computed and rounded using round(..., 2).
- **Usage example:**
  The code includes a sample usage in the __main__ block, demonstrating add/remove/summary