AI ASSISTED CODING

LAB TEST – 02

NAME : Neha Shazneen

ROLL NO : 2403A510A1

BATCH : 05

DEPARTMENT : CSE

SET A

A.1 - Compute per-player average from logs (AI completion)

Prompt :

You are integrating a sports analytics telemetry service where each player emits periodic measures as CSV lines: id,timestamp,velocity. Due to flaky connectivity, some lines may be truncated or contain non-numeric values. Ops needs a quick aggregation for dashboards and alert thresholds.

Task :

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

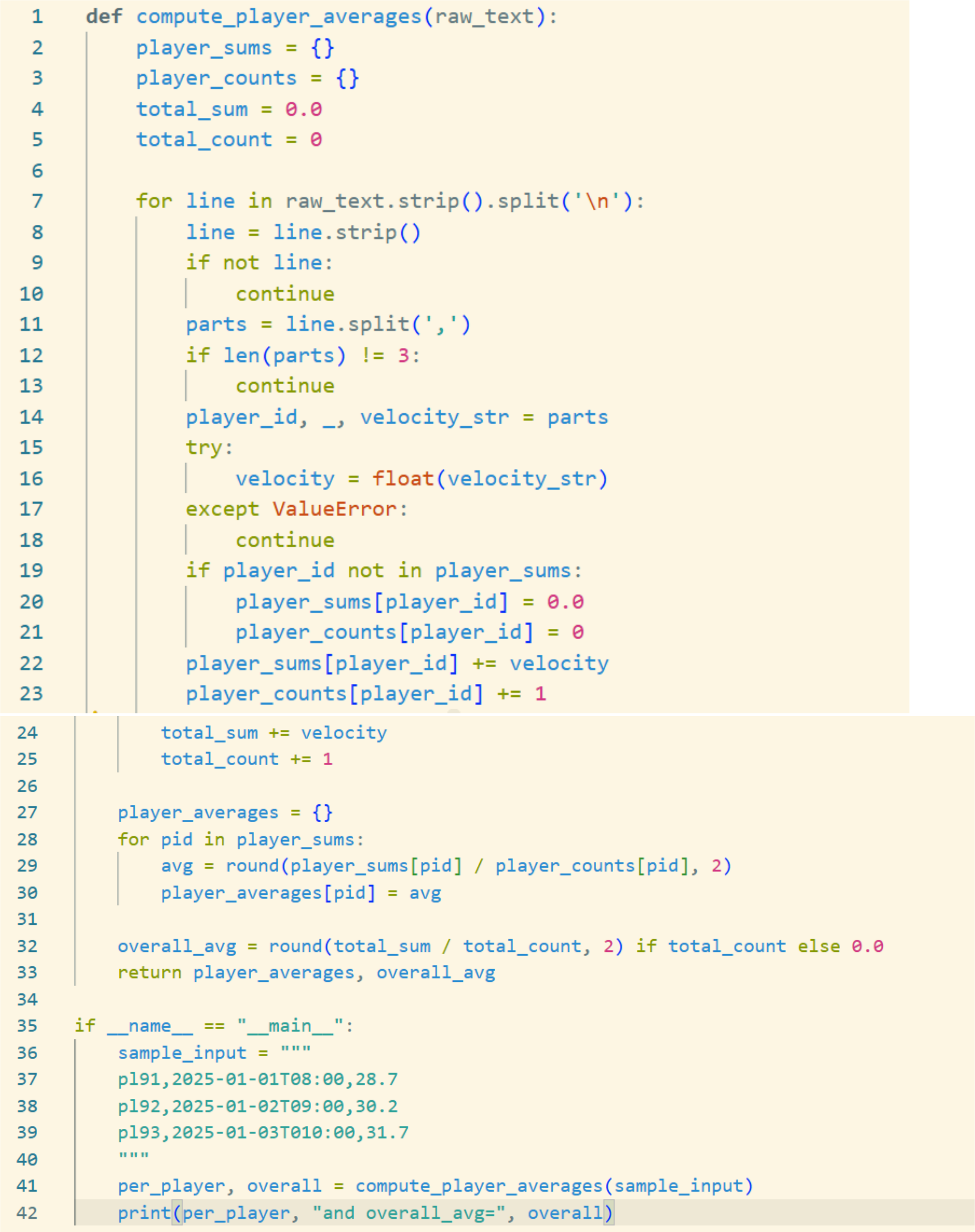
Data & Edge Cases:

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

Constraints & Notes:

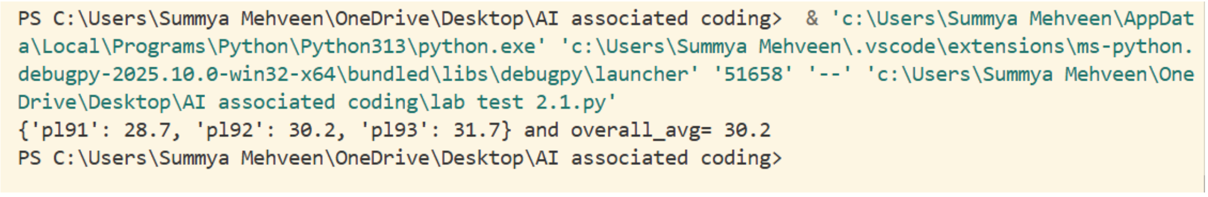
- Prefer O(n) pass; ignore lines that cannot be split into three fields or have non-numeric metric.
- Round averages to 2 decimals.

Code Generated :



```python
def compute_player_averages(raw_text):
    player_sums = {}
    player_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
        player_id, _, velocity_str = parts
        try:
            velocity = float(velocity_str)
        except ValueError:
            continue
        if player_id not in player_sums:
            player_sums[player_id] = 0.0
            player_counts[player_id] = 0
        player_sums[player_id] += velocity
        player_counts[player_id] += 1
        total_sum += velocity
        total_count += 1

    player_averages = {}
    for pid in player_sums:
        avg = round(player_sums[pid] / player_counts[pid], 2)
        player_averages[pid] = avg

    overall_avg = round(total_sum / total_count, 2) if total_count else 0.0
    return player_averages, overall_avg

if __name__ == "__main__":
    sample_input = """
    pl91,2025-01-01T08:00,28.7
    pl92,2025-01-02T09:00,30.2
    pl93,2025-01-03T010:00,31.7
    """
    per_player, overall = compute_player_averages(sample_input)
    print(per_player, "and overall_avg=", overall)
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  & 'c:\Users\Summya Mehveen\AppDat
a\Local\Programs\Python\Python313\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.
debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '51658' '--' 'c:\Users\Summya Mehveen\One
Drive\Desktop\AI associated coding\lab test 2.1.py'
{'pl91': 28.7, 'pl92': 30.2, 'pl93': 31.7} and overall_avg= 30.2
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

The provided Python function compute_player_averages correctly parses multiline CSV telemetry data, computes per-player average velocities, and calculates the overall average velocity. It robustly skips malformed or non-numeric lines, ensuring only valid data is aggregated. The output matches the expected format: a dictionary of player averages (rounded to 2 decimals) and an overall average. The sample input produces the correct result, demonstrating the function handles edge cases as required.

A.2 - Implement MatchStats with add/remove/summary (AI completion)

Prompt :

A microservice in the sports analytics platform maintains a small in-memory structure to track values keyed by identifier (e.g., order IDs, sensor IDs). Engineers want a minimal class to add, remove, and summarize current values for quick health checks.

Task :

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

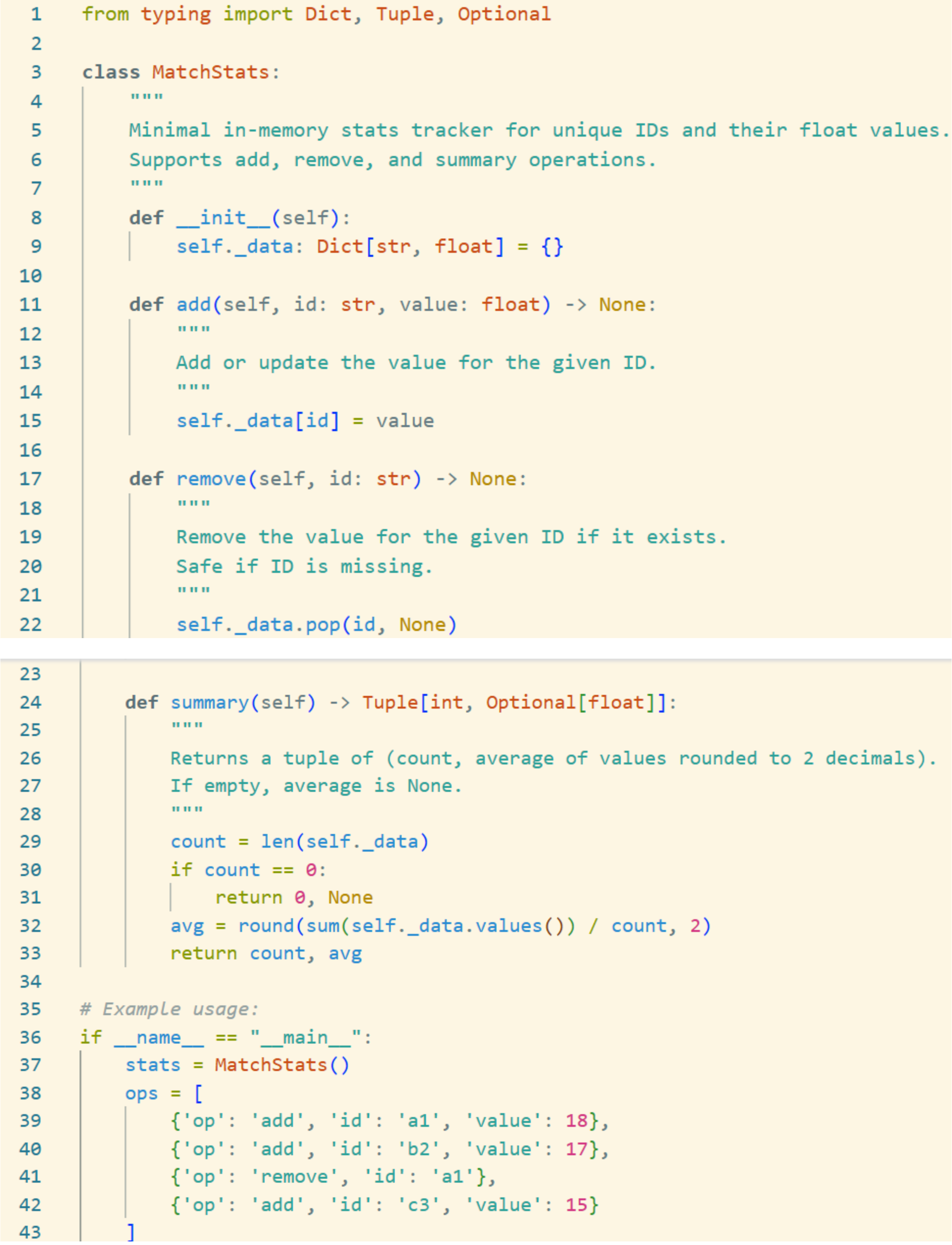
Data & Edge Cases:

- 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.

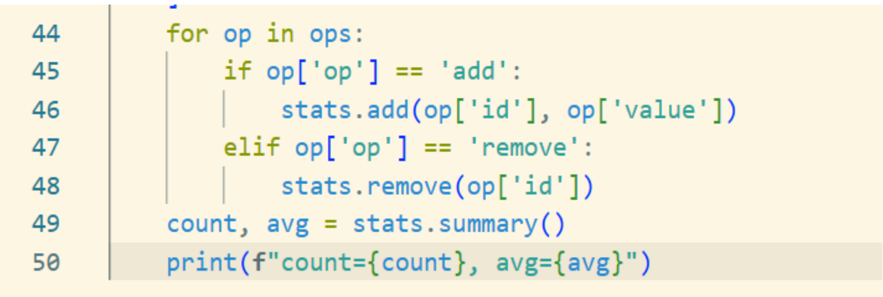
Constraints & Notes:

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

Code Generated :

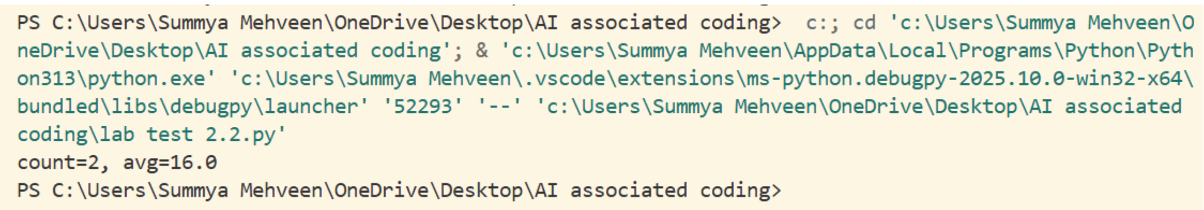


```python
from typing import Dict, Tuple, Optional

class MatchStats:
    """
    Minimal in-memory stats tracker for unique IDs and their float values.
    Supports add, remove, and summary operations.
    """
    def __init__(self):
        self._data: Dict[str, float] = {}

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

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

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

# Example usage:
if __name__ == "__main__":
    stats = MatchStats()
    ops = [
        {'op': 'add', 'id': 'a1', 'value': 18},
        {'op': 'add', 'id': 'b2', 'value': 17},
        {'op': 'remove', 'id': 'a1'},
        {'op': 'add', 'id': 'c3', 'value': 15}
    ]
```



```python
    for op in ops:
        if op['op'] == 'add':
            stats.add(op['id'], op['value'])
        elif op['op'] == 'remove':
            stats.remove(op['id'])
    count, avg = stats.summary()
    print(f"count={count}, avg={avg}")
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\O
neDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Pyth
on313\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\
bundled\libs\debugpy\launcher' '52293' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated
coding\lab test 2.2.py'
count=2, avg=16.0
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

The MatchStats class efficiently manages a dictionary of unique IDs and their float values, supporting O(1) add, remove, and summary operations.

- Adding an ID overwrites its value if it already exists.
- Removing a missing ID is safe and does not raise an exception.
- The summary() method returns the correct count and the average (rounded to 2 decimals), or None for the average if the store is empty.

The provided example demonstrates correct behavior: after the sequence of operations, the count is 2 and the average is 16.0, matching the sample output and requirements.