NAME:- DONTHI MEGHANA

ROLL NO:- 2403A510D9

BATCH:- 05

COURSE NAME:- AI ASSISTED CODING

Subgroup E
E.1 — [S09E1] Generate README from comments
Scenario (sports analytics):
Context:
A small sports analytics utility module needs a README for onboarding and CI.
Your Task:
From inline comments, produce a README with sections: Overview, Setup, Usage, Tests,
Limitations; include one CLI example.
Data & Edge Cases:
Comments mention module name and functions: parse, validate, export.
AI Assistance Expectation:
Use AI to draft the README and refine wording for clarity.
Constraints & Notes:
Ensure each section is present and concise.
Sample Input
module: sports analytics utilities
functions: parse, validate, export
Sample Output
README with 5 sections and example
Acceptance Criteria: Includes example CLI invocation

Prompt:
"Generate a concise README in Markdown for a small sports analytics utility module. The README must include the following sections:

1. Overview — Briefly explain what the module does (based on inline comments such as module name and functions: parse, validate, export).
2. Setup — Simple installation or environment setup instructions.
3. Usage — Show how to run the module from the command line with at least one CLI example.
4. Tests — Instructions on how to test the functions using sample data.
5. Limitations — List any known constraints like supported formats, assumptions, or lack of error handling.

Input will be inline comments (e.g., # module: sports analytics utilities, # functions: parse, validate, export). Output must be a structured README in Markdown that is clear, professional, and concise."

CODE:-

LAB.E.1.py > ...

```python
"""
module: sports analytics utilities
functions: parse, validate, export

parse: Parses raw sports data files into structured format.
validate: Validates parsed data for consistency and correctness.
export: Exports validated data to CSV or JSON for downstream analysis.
"""

Zencoder
def parse(input_file):
    """Parse raw sports data from input_file and return structured data."""
    # ...implementation...
    pass

Zencoder
def validate(data):
    """Validate structured sports data for consistency and correctness."""
    # ...implementation...
    pass

Zencoder
def export(data, output_file, fmt="csv"):
    """Export validated data to output_file in specified format (csv or json)."""
    # ...implementation...
    pass

# README Generation Prompt:
# From the above comments, generate a README with the following sections:
# Overview, Setup, Usage, Tests, Limitations. Include one CLI example.

# --- README.md (AI-generated) ---
README = """
# Sports Analytics Utilities

## Overview
This module provides utility functions for sports analytics, including parsing raw data, validating for consistency,

## Setup
No special setup required. Ensure Python 3.x is installed.

## Usage
Import the module and use the provided functions:

```python
from LAB.E.1 import parse, validate, export

data = parse('raw_data.txt')
if validate(data):
    export(data, 'output.csv', fmt='csv')
```
```

Keep Undo 10 of

LAB.E.1.py > validate

```
## Usage
Import the module and use the provided functions:

```python
from LAB.E.1 import parse, validate, export

data = parse('raw_data.txt')
if validate(data):
    export(data, 'output.csv', fmt='csv')
```

CLI Example
```sh
python LAB.E.1.py raw_data.txt output.csv --format csv
```

Tests
Test each function with sample data to ensure correct parsing, validation, and export. Use assert statements or a te

Limitations
- Only basic CSV and JSON export supported.
- Assumes well-formed input files.
- No advanced error handling or logging.
"""

if __name__ == "__main__":
    print(README)
```

Keep Undo 10 of

OUTPUT:-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Administrator\OneDrive\ai> & C:/Python313/python.exe c:/Users/Administrator/OneDrive/ai/LAB.E.1.py

# Sports Analytics Utilities

## Overview
This module provides utility functions for sports analytics, including parsing raw data, validating for consistency, and expor
ting results for further analysis.

## Setup
No special setup required. Ensure Python 3.x is installed.

## Usage
Import the module and use the provided functions:

```python
from LAB.E.1 import parse, validate, export

data = parse('raw_data.txt')
if validate(data):
    export(data, 'output.csv', fmt='csv')
```

CLI Example
```sh
python LAB.E.1.py raw_data.txt output.csv --format csv
```

Tests
Test each function with sample data to ensure correct parsing, validation, and export. Use assert statements or a test framewo
rk.

Limitations
```

```
### CLI Example
```sh
python LAB.E.1.py raw_data.txt output.csv --format csv
```

Tests
Test each function with sample data to ensure correct parsing, validation, and export. Use assert statements or a test framewo
rk.

Limitations
- Only basic CSV and JSON export supported.
- Assumes well-formed input files.
- No advanced error handling or logging.

PS C:\Users\Administrator\OneDrive\ai>
```

OBSERVATIONS:-

□ Module Context

- The module is for *sports analytics utilities*.
- It provides three main functions: parse, validate, and export.
- Purpose: to process sports-related raw data and export it in usable formats.

□ Inline Comments → README Mapping

- Inline comments give minimal info (module name + functions).
- The task is to expand these into a professional README with five sections.
- AI should refine the wording for clarity while staying concise.

□ Structure Requirements

- Must include: **Overview, Setup, Usage, Tests, Limitations**.
- A **CLI example** is mandatory under *Usage*.
- Should feel like onboarding documentation for a small utility.

□ Edge Cases & Constraints

- Only limited comments are available → AI needs to *infer* and expand meaning.
- Keep wording concise (onboarding + CI use case).

- Limitations should mention: basic format support, assumptions about input, lack of advanced error handling.

? Expected Output

- A polished **README.md** in Markdown.
- Clear section headers.
- One concrete CLI invocation example (e.g., python LAB.E.1.py raw_data.txt output.csv --format csv).
- Professional, easy-to-read wording

E.2 — [S09E2] Refactor nested loops to dict aggregation
Scenario (sports analytics):

- **Context:**
  A legacy sports analytics script uses verbose loops to aggregate key-value tuples.
  Your Task:
  Refactor to a pythonic aggregation using dict.get or collections.defaultdict with type hints.
  Data & Edge Cases:
  Input example: [('a',1),('b',2),('a',3)] -> {'a':4,'b':2}.
  AI Assistance Expectation:
  Ask AI for refactor suggestions, then apply and ensure behavior parity via tests.
  Constraints & Notes:
  Type hints for function signatures required.
  Sample Input
  data=[('a',1),('b',2),('a',3)]
  Sample Output
  {'a':4,'b':2}
  Acceptance Criteria: Behavior unchanged; improved readability
- **Sample Input**
  data=[('a',1),('b',2),('a',3)]
  Sample Output
  {'a':4,'b':2}
  Acceptance Criteria: Behavior unchanged; improved readability

PROMPT:-

Prompt:

"Refactor a legacy Python script that uses nested loops to aggregate key-value tuples into a dictionary. Replace verbose loops with a more Pythonic solution using either `dict.get` *or* `collections.defaultdict`*. Ensure function signatures include type hints. Input format: list of* `(str, int)` *tuples. Example input:* `[('a',1), ('b',2), ('a',3)]`*. Expected output:* `{'a':4, 'b':2}`*. Provide the refactored function, explain improvements in readability and efficiency, and include a simple test to confirm behavior parity with the original version."*

CODE:-

Welcome | LAB.E.1.py | LAB.E.2.py

LAB.E.2.py > ...

```python
from typing import List, Tuple, Dict
from collections import defaultdict

Zencoder
def aggregate_scores(data: List[Tuple[str, int]]) -> Dict[str, int]:
    # Pythonic aggregation using defaultdict
    agg = defaultdict(int)
    for k, v in data:
        agg[k] += v
    return dict(agg)

# Test for behavior parity
if __name__ == "__main__":
    data = [('a', 1), ('b', 2), ('a', 3)]
    result = aggregate_scores(data)
    print(result)  # Output: {'a': 4, 'b': 2}

```

OUTPUT:-

```
PS C:\Users\Administrator\OneDrive\ai> & C:/Python313/python.exe c:/Users/Administrator/OneDrive/ai/LAB.E.2.py
{'a': 4, 'b': 2}
PS C:\Users\Administrator\OneDrive\ai>
```

🔍 Observations

1. **Context & Goal**
   - The legacy script uses **nested loops** → inefficient and verbose.
   - Task is to rewrite it using **dict aggregation** (`dict.get` or `collections.defaultdict`).
   - This improves readability and performance.
2. **Data Example**
   - Input: `[('a',1),('b',2),('a',3)]`
   - Output: `{'a':4,'b':2}`
   - Confirms aggregation by **summing values** for the same key.
3. **Constraints & Notes**
   - Must **preserve behavior** (output unchanged).
   - Function signatures should use **type hints** → improves clarity for onboarding & CI.
   - Output dict should have summed values, no duplicates.
4. **Edge Cases**
   - Empty list → `{}`
   - All keys unique → no aggregation needed.
   - All keys same → single key with total sum.
   - Large input → efficiency matters, hence `defaultdict` preferred.
5. **Expected AI Assistance**
   - Suggest refactor from loops to Pythonic style.
   - Provide **readable code snippet** with type hints.
   - Include at least **one test case** to confirm parity.

***THE END ***