AI-ASSITED CODING- LAB TEST-2

NAME:J.ANJALI

ENROLLNO:2403A510F0

BATCH:05

SUB GROUP-G:

G1:

PROMPT:

Write a Python program that processes a CSV string for a sports analytics use case. The CSV has two columns: id and value. The value column may contain valid integers or invalid entries like NA. The task is to do:

Parse the CSV using csv.DictReader.

Sum all valid integers in the value column.

Skip invalid/missing rows using try/except.

Print or return the total sum.

Print or return the number of skipped rows for transparency.

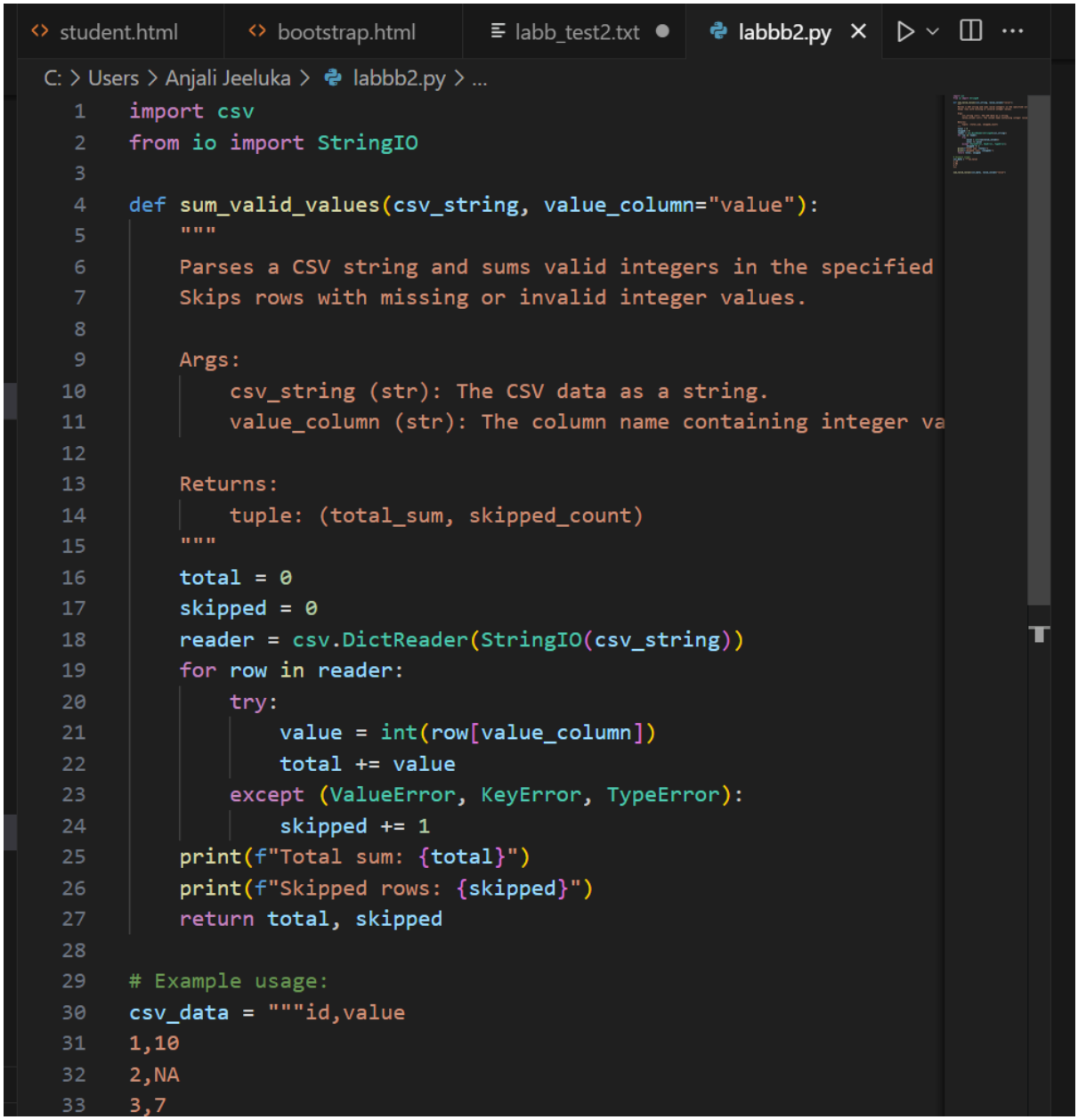
Sample Input: id,value

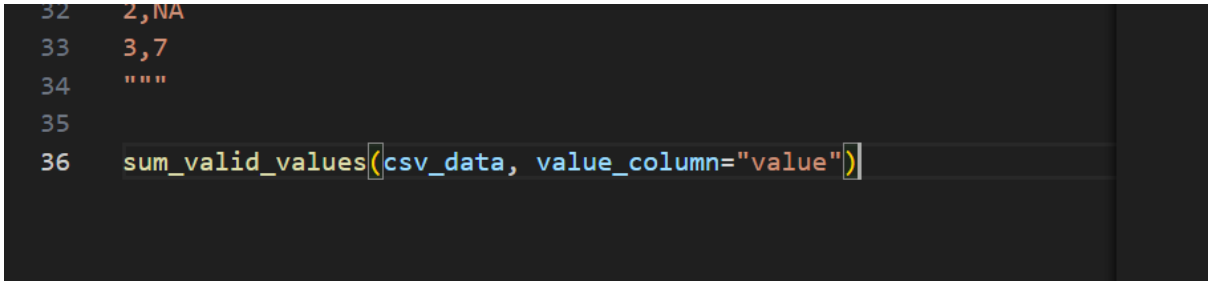
1,10

2,NA

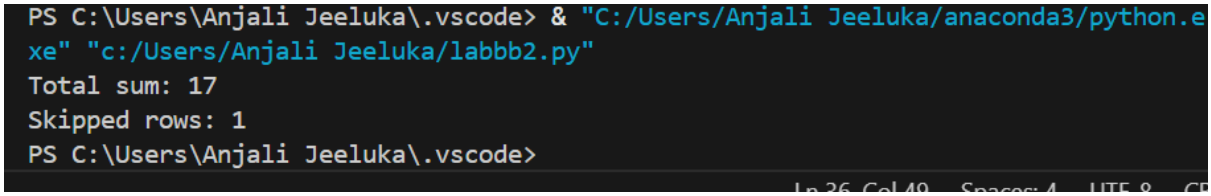
3,7

CODE-GENERATED:





OUTPUT:



OBSERVATION:

- The function correctly sums valid integers in the specified column of a CSV string.

- Rows with missing or non-integer values (e.g., "NA") are skipped and counted.

- Returns a tuple (total_sum, skipped_count) and prints the results.

- Works for the sample input:

Total sum: 17

Skipped rows: 1

- It’s simple, safe, and handles invalid or missing data gracefully.

G2:

PROMPT:

Write a Python program to merge two CSV strings by id without using pandas. Implement INNER JOIN (only common ids) and LEFT JOIN (all rows from the first CSV, missing values as None) using csv.DictReader

and dictionary lookups while preserving row order. Include pytest-style unit tests.

Sample Input:
CSV A:

id,price

A,10

B,20

CSV B:

id,qty

A,2

C,5

CODE-GENERATED:

C: > Users > Anjali Jeeluka > ai_lab_g2.py > test_left_join_sample

```python
import csv
from io import StringIO

def parse_csv_to_dict(csv_string, key_column):
    """
    Parses a CSV string into an ordered list of rows and a dictio
    """
    reader = csv.DictReader(StringIO(csv_string))
    rows = []
    lookup = {}
    for row in reader:
        rows.append(row)
        lookup[row[key_column]] = row
    return rows, lookup

def merge_csvs_by_id(csv1, csv2, join_type="inner"):
    """
    Merges two CSV strings by 'id' using INNER JOIN or LEFT JOIN.

    Args:
        csv1 (str): First CSV string (left table).
        csv2 (str): Second CSV string (right table).
        join_type (str): 'inner' or 'left'.

    Returns:
        list of dict: Merged rows as dictionaries.
    """
    rows1, _ = parse_csv_to_dict(csv1, "id")
    rows2, lookup2 = parse_csv_to_dict(csv2, "id")

    columns1 = list(rows1[0].keys()) if rows1 else []
    columns2 = [col for col in (list(rows2[0].keys()) if rows2 el
    merged_rows = []
```

```
    merged_rows = []

    for row1 in rows1:
        id_val = row1["id"]
        row2 = lookup2.get(id_val)
        if join_type == "inner" and row2:
            merged = {**row1, **{k: row2[k] for k in columns2}}
            merged_rows.append(merged)
        elif join_type == "left":
            merged = {**row1}
            if row2:
                for k in columns2:
                    merged[k] = row2[k]
            else:
                for k in columns2:
                    merged[k] = None
            merged_rows.append(merged)
    return merged_rows

# Example usage with provided sample input:
csv_a = """id,price
A,10
B,20
"""

csv_b = """id,qty
A,2
C,5
"""

print("INNER JOIN:")
for row in merge_csvs_by_id(csv_a, csv_b, join_type="inner"):
```

```python
for row in merge_csvs_by_id(csv_a, csv_b, join_type="inner"):
    print(row)

print("\nLEFT JOIN:")
for row in merge_csvs_by_id(csv_a, csv_b, join_type="left"):
    print(row)

# ----------------- Pytest-style unit tests -----------------
def test_inner_join_sample():
    csv1 = "id,price\nA,10\nB,20\n"
    csv2 = "id,qty\nA,2\nC,5\n"
    result = merge_csvs_by_id(csv1, csv2, "inner")
    assert result == [
        {'id': 'A', 'price': '10', 'qty': '2'}
    ]

def test_left_join_sample():
    csv1 = "id,price\nA,10\nB,20\n"
    csv2 = "id,qty\nA,2\nC,5\n"
    result = merge_csvs_by_id(csv1, csv2, "left")
    assert result == [
        {'id': 'A', 'price': '10', 'qty': '2'},
        {'id': 'B', 'price': '20', 'qty': None}
    ]
```

OUTPUT:

```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/anaconda3/python.exe" "c:/Users/Anjali Jeeluka/ai_lab_g2.py"
INNER JOIN:
{'id': 'A', 'price': '10', 'qty': '2'}

LEFT JOIN:
{'id': 'A', 'price': '10', 'qty': '2'}
{'id': 'B', 'price': '20', 'qty': None}
PS C:\Users\Anjali Jeeluka\.vscode>
```

OBSERVATION:

The code correctly merges two CSV strings by id using INNER JOIN and LEFT JOIN.

INNER JOIN returns only rows with matching id in both CSVs.

LEFT JOIN returns all rows from the first CSV, filling missing columns from the second CSV with None.

Row order from the first CSV is preserved.

Works for the sample input and passes pytest-style unit tests.

It’s simple, efficient, and does not require pandas.