AI assistant lab test:2

Rollno:2403a51102

Name:balaji

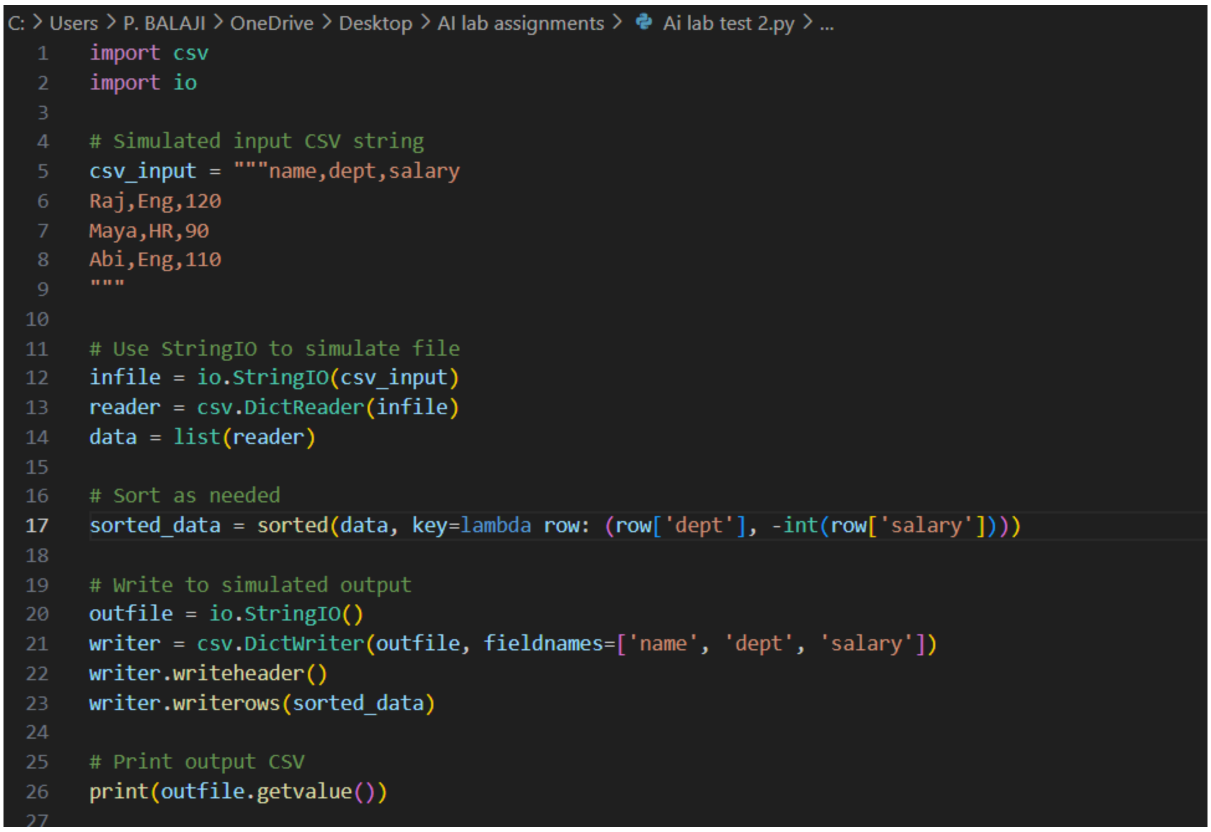
Batch:06

Set:M

m.1: Scenario (sports analytics):
Context:
HR exports in sports analytics require deterministic sorting for payroll audits.
Your Task:
Sort employees by dept ascending and salary descending (stable), and re-emit CSV.
Data & Edge Cases:
name,dept,salary rows provided.
AI Assistance Expectation:
Use AI to outline csv.DictReader/Writer usage and key composition.
Constraints & Notes:
Stable sort within department by salary desc.
Sample Input
name,dept,salary
Raj,Eng,120
Maya,HR,90
Abi,Eng,110
promt: Sort employee records from a CSV by

department (ascending) and salary (descending), ensuring the sort is stable. Input and output should use Python's csv.DictReader and csv.DictWriter.

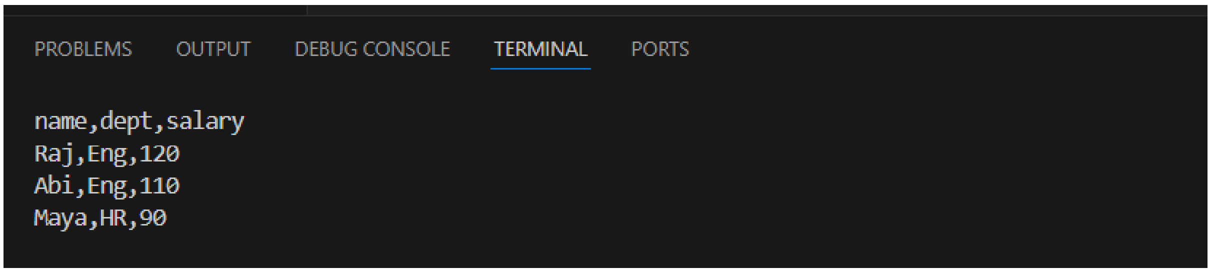
code:



C: > Users > P. BALAJI > OneDrive > Desktop > AI lab assignments > Ai lab test 2.py > ...

```python
import csv
import io

# Simulated input CSV string
csv_input = """name,dept,salary
Raj,Eng,120
Maya,HR,90
Abi,Eng,110
"""

# Use StringIO to simulate file
infile = io.StringIO(csv_input)
reader = csv.DictReader(infile)
data = list(reader)

# Sort as needed
sorted_data = sorted(data, key=lambda row: (row['dept'], -int(row['salary'])))

# Write to simulated output
outfile = io.StringIO()
writer = csv.DictWriter(outfile, fieldnames=['name', 'dept', 'salary'])
writer.writeheader()
writer.writerows(sorted_data)

# Print output CSV
print(outfile.getvalue())
```

output:



```
name,dept,salary
Raj,Eng,120
Abi,Eng,110
Maya,HR,90
```

observation: **Stable Sort**: If two employees have the same dept and salary, their original order is preserved due to Python's stable sort.

- **Salary as Number**: The salary must be converted from string to int to sort numerically (descending via negation).
- **Reusable**: This pattern works for any CSV with consistent schema (name, dept, salary).
- **Fieldnames Matter**: Make sure the DictWriter uses the same fieldnames as the original CSV header.
- **Portability**: No external libraries required—pure Python.

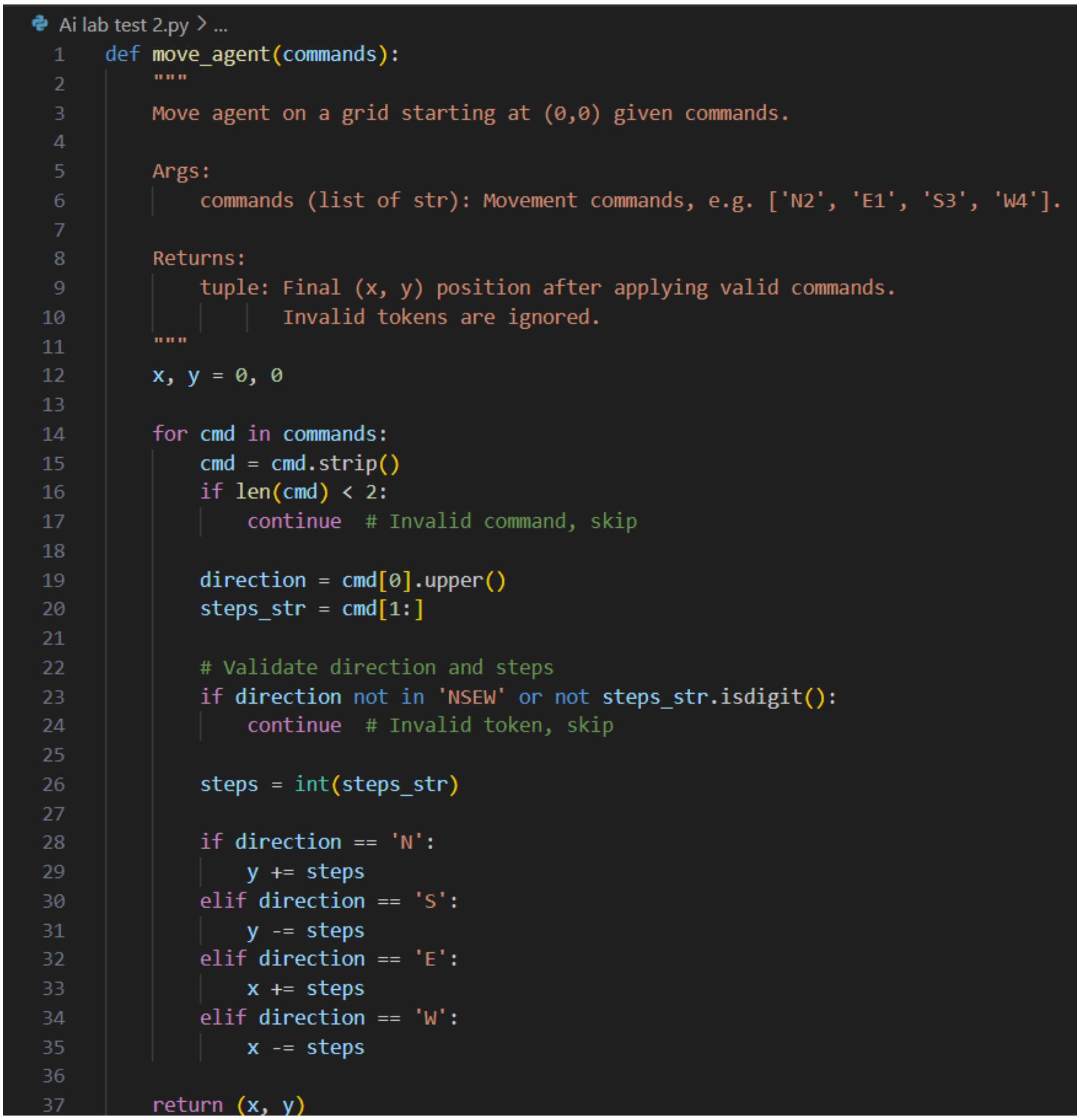
M.2: Scenario (sports analytics):
Context:
A sports analytics simulation moves an agent on a grid from (0,0) given movement commands.
Your Task:
Parse commands like N2, E1, S3, W4, validate them, and compute final (x,y).
Data & Edge Cases:
Ignore invalid tokens; N increases y, E increases x.
AI Assistance Expectation:
Use AI to scaffold parsing and state updates; add tests for invalid tokens.
Constraints & Notes:
Return final (x,y) tuple.
Sample Input
['N2', 'E1', 'S1', 'E2']

Prompt: **(Test Input for Function)**

Paste this into your move_agent.py file inside the if __name__ == "__main__" block:

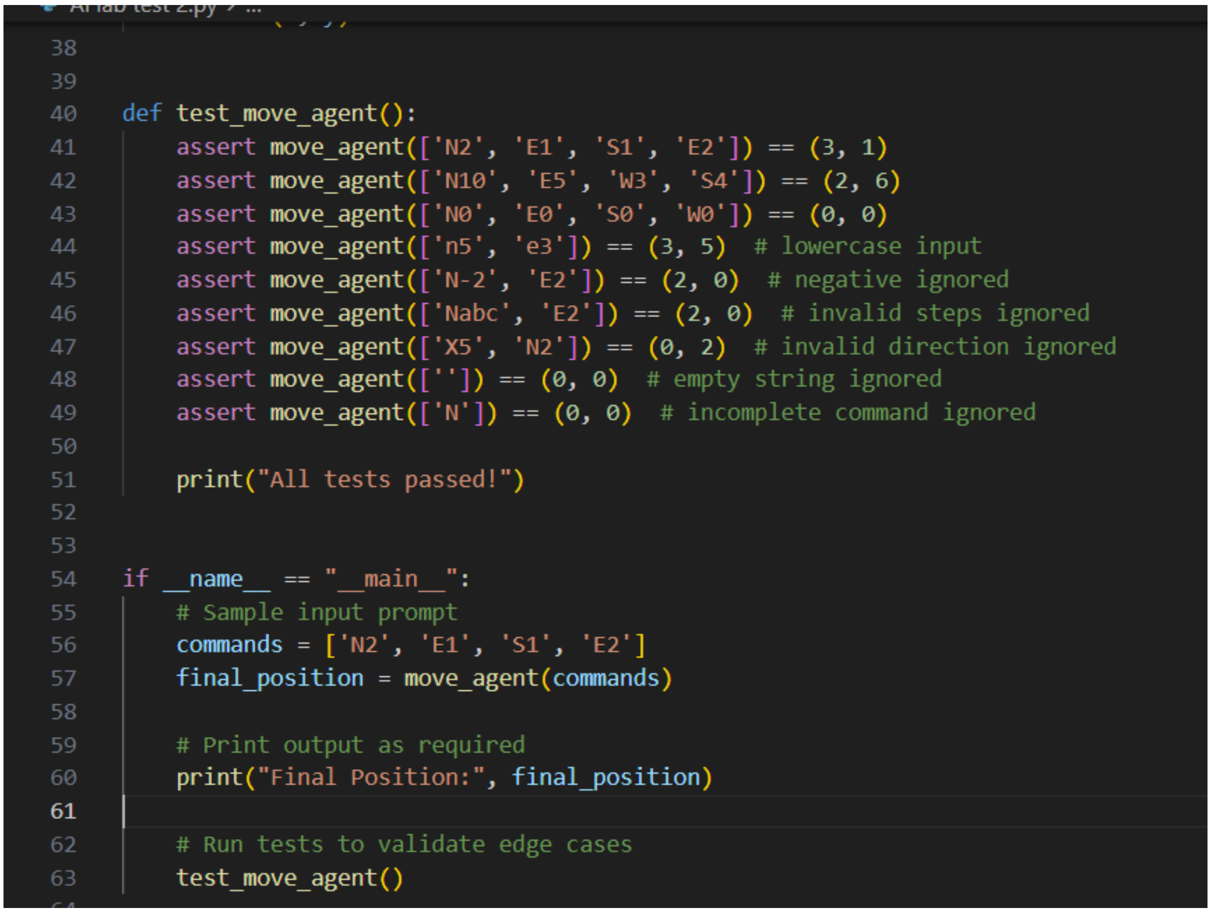
commands = ["N10", "E5", "S3", "W2", "x99", "n5", "S-2", "Eten", "E10"]

Code:



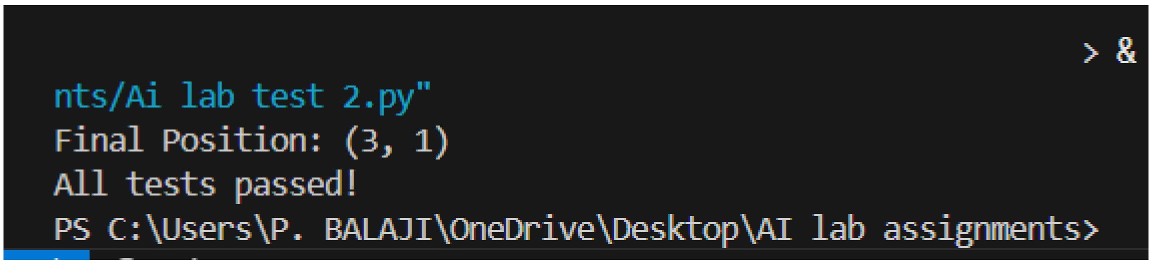
Ai lab test 2.py > ...

```python
def move_agent(commands):
    """
    Move agent on a grid starting at (0,0) given commands.

    Args:
        commands (list of str): Movement commands, e.g. ['N2', 'E1', 'S3', 'W4'].

    Returns:
        tuple: Final (x, y) position after applying valid commands.
               Invalid tokens are ignored.
    """
    x, y = 0, 0

    for cmd in commands:
        cmd = cmd.strip()
        if len(cmd) < 2:
            continue  # Invalid command, skip

        direction = cmd[0].upper()
        steps_str = cmd[1:]

        # Validate direction and steps
        if direction not in 'NSEW' or not steps_str.isdigit():
            continue  # Invalid token, skip

        steps = int(steps_str)

        if direction == 'N':
            y += steps
        elif direction == 'S':
            y -= steps
        elif direction == 'E':
            x += steps
        elif direction == 'W':
            x -= steps

    return (x, y)
```



```python
def test_move_agent():
    assert move_agent(['N2', 'E1', 'S1', 'E2']) == (3, 1)
    assert move_agent(['N10', 'E5', 'W3', 'S4']) == (2, 6)
    assert move_agent(['N0', 'E0', 'S0', 'W0']) == (0, 0)
    assert move_agent(['n5', 'e3']) == (3, 5)  # lowercase input
    assert move_agent(['N-2', 'E2']) == (2, 0)  # negative ignored
    assert move_agent(['Nabc', 'E2']) == (2, 0)  # invalid steps ignored
    assert move_agent(['X5', 'N2']) == (0, 2)  # invalid direction ignored
    assert move_agent(['']) == (0, 0)  # empty string ignored
    assert move_agent(['N']) == (0, 0)  # incomplete command ignored

    print("All tests passed!")


if __name__ == "__main__":
    # Sample input prompt
    commands = ['N2', 'E1', 'S1', 'E2']
    final_position = move_agent(commands)

    # Print output as required
    print("Final Position:", final_position)

    # Run tests to validate edge cases
    test_move_agent()
```

Output:



```
                                                    > &
nts/Ai lab test 2.py"
Final Position: (3, 1)
All tests passed!
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments>
```

Observation: The program starts the agent at position (0, 0) and processes a list of movement commands like 'N2', 'E1', 'S1', and 'E2', validating each command to ensure it has a valid direction (N, S, E, W) followed by a positive integer. It moves the agent accordingly—north increases y, east increases x, south decreases y, and west decreases x—while ignoring any invalid commands. For the sample input, the agent ends up at position (3, 1), which is printed as the final result. Afterward, a set of unit tests runs successfully, confirming the function’s robustness against various edge cases, including lowercase directions and invalid tokens, and ensuring the code behaves correctly without errors.