LAB TEST-2

AI ASSISTED CODING

**2403A51238**

**Chandana**

**QUESTION: M.1 — [S18M1] Stable sort employees by dept asc, salary desc**

**Context:**

Payroll audit sorting in real estate listings platform.

 Your Task:

Sort by dept asc, salary desc; re-emit CSV.

 Data & Edge Cases:

name,dept,salary rows.

 AI Assistance Expectation:

AI for csv.DictReader/Writer keys.

Constraints & Notes:

Stable within dept.

Sample Input

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

Sample Output

Raj,Eng,120  
Abi,Eng,110  
Maya,HR,90

Acceptance Criteria: Stable, correct

**1)PROMPT TO AI:**

"Subgroup M M.1 — [S18M1] Stable sort employees by dept asc, salary desc

Context: Payroll audit sorting in real estate listings platform.

Your Task: Sort by dept asc, salary desc; re-emit CSV.

Data & Edge Cases: name,dept,salary rows.

Constraints & Notes: Stable within dept.

Sample Input

name,dept,salary

Raj,Eng,120

Maya,HR,90

Abi,Eng,110

Sample Output

Raj,Eng,120

Abi,Eng,110

Maya,HR,90

Acceptance Criteria: Stable, correct"

**2)solution.py:**

"""

solution.py

Stable sort employees by dept asc, salary desc

"""

def sort\_employees(data):

"""

Sorts employees by dept asc & salary desc.

Input: list of dicts with keys: name, dept, salary

Output: sorted list

"""

return sorted(data, key=lambda r: (r["dept"], -int(r["salary"])))

if \_\_name\_\_ == "\_\_main\_\_":

# Sample input

employees = [

{"name": "Raj", "dept": "Eng", "salary": "120"},

{"name": "Maya", "dept": "HR", "salary": "90"},

{"name": "Abi", "dept": "Eng", "salary": "110"},

]

# Run sorting

result = sort\_employees(employees)

# Print output

for r in result:

print(f"{r['name']},{r['dept']},{r['salary']}")

**OUTPUT FOR solution.py:**

Raj,Eng,120

Abi,Eng,110

Maya,HR,90

**3)tests.py**

import unittest

from solution import sort\_employees

class TestSortEmployees(unittest.TestCase):

def test\_sorting(self):

employees = [

{"name": "Raj", "dept": "Eng", "salary": "120"},

{"name": "Maya", "dept": "HR", "salary": "90"},

{"name": "Abi", "dept": "Eng", "salary": "110"},

]

expected = [

{"name": "Raj", "dept": "Eng", "salary": "120"},

{"name": "Abi", "dept": "Eng", "salary": "110"},

{"name": "Maya", "dept": "HR", "salary": "90"},

]

result = sort\_employees(employees)

self.assertEqual(result, expected)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**OUTPUT FOR tests.py:**

.

----------------------------------------------------------------------

Ran 1 test in 0.001s

OK

**4)Docstrings & Inline Comments:**

Docstrings:

- Module-level docstring: purpose, usage.

- sort\_employees(): explains inputs, behavior, and assumptions.

Inline comments:

- Explain reading/writing CSV with csv.DictReader/DictWriter.

- Key sort line: rows.sort(key=lambda r: (r["dept"], -int(r["salary"]))) with comment about stability and -int for descending salary.

**5)README.md:**

# M.1 [S18M1] - Stable Sort Employees

## Approach

- Store employees as list of dicts.

- Use sorted() with key=(dept asc, salary desc).

- Print results in CSV format.

## Assumptions

- Salary is integer.

- Dept sorted lexicographically (A-Z).

- Stable sort is guaranteed by Python.

## Complexity

- Sorting: O(n log n)

## Run Program

python solution.py

## Run Tests

python -m unittest tests.py

**6)Debugging / Refactor Note (Before / After):**

Before:

- Unsure about stability, considered using `sorted()` vs `list.sort()` and worry about accidental re-orders.

- Initial naive solutions attempted manual grouping by dept then sorting each group.

After:

- Final: used Python's list.sort() which is stable by design.

- Key line: rows.sort(key=lambda r: (r["dept"], -int(r["salary"])))

- This simplified code, improved correctness and preserved input order for ties.

**QUESTION: M.2 — [S18M2] Process movement commands Context:**

**Context:**

Grid movement simulator in real estate listings platform.

**Your Task:**

Parse N/E/S/W steps to final (x,y).

 Data & Edge Cases:

Ignore invalid tokens.

 AI Assistance Expectation:

AI to scaffold parsing; add invalid cases.

 Constraints & Notes:

Return (x,y).

Sample Input

['N2', 'E1', 'S1', 'E2']

Sample Output

(3,1)

Acceptance Criteria: Validates tokens

**1)Prompt to AI:**

"M.2 — [S18M2] Process movement commands

Context: Grid movement simulator in real estate listings platform.

Your Task: Parse N/E/S/W steps to final (x,y) coordinates.

Data & Edge Cases: Ignore invalid tokens.

Constraints & Notes: Return (x,y).

Sample Input: ['N2', 'E1', 'S1', 'E2']

Sample Output: (3,1)

"

**2)solution.py**

"""

solution.py

Process grid movement commands and return final (x, y) position.

"""

def process\_movements(commands):

"""

Takes a list of movement strings like 'N2', 'E1', etc.

Ignores invalid tokens.

Returns final coordinates (x, y) starting from (0, 0).

"""

x, y = 0, 0

for cmd in commands:

if len(cmd) < 2:

continue # invalid token

direction = cmd[0].upper()

try:

steps = int(cmd[1:])

except ValueError:

continue # invalid number, skip

if direction == 'N':

y += steps

elif direction == 'S':

y -= steps

elif direction == 'E':

x += steps

elif direction == 'W':

x -= steps

else:

continue # invalid direction

return (x, y)

if \_\_name\_\_ == "\_\_main\_\_":

# Sample input

commands = ['N2', 'E1', 'S1', 'E2']

result = process\_movements(commands)

print(result)

**OUTPUT FOR solution.py:**

(3,1)

**3)tests.py**

import unittest

from solution import process\_movements

class TestProcessMovements(unittest.TestCase):

def test\_basic(self):

commands = ['N2', 'E1', 'S1', 'E2']

expected = (3,1)

self.assertEqual(process\_movements(commands), expected)

def test\_invalid\_tokens(self):

commands = ['N2', 'X5', 'E1', 'S1', 'abc', 'E2']

expected = (3,1)

self.assertEqual(process\_movements(commands), expected)

def test\_empty\_list(self):

commands = []

expected = (0,0)

self.assertEqual(process\_movements(commands), expected)

def test\_negative\_steps(self):

commands = ['N2', 'S-1', 'E1']

# 'S-1' is valid, so y decreases by -1 (effectively increases)

expected = (1,3)

self.assertEqual(process\_movements(commands), expected)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**OUTPUT for tests.py:**

....

----------------------------------------------------------------------

Ran 4 tests in 0.002s

OK

**4) Docstrings & Inline Comments:**

**Docstrings:**

"""

solution.py

Process grid movement commands and return final (x, y) position.

Functions:

- process\_movements(commands): Parses a list of movement commands and returns

the final coordinates (x, y) starting from (0,0).

Usage:

- Provide a list of strings like ['N2', 'E1', 'S1', 'E2'].

- Invalid tokens (wrong direction or non-integer steps) are ignored.

- Returns a tuple: (x, y).

"""

**Inline Comments**:

x, y = 0, 0 # start coordinates

for cmd in commands:

if len(cmd) < 2:

continue # skip invalid tokens too short to be meaningful

direction = cmd[0].upper() # first letter is direction, case-insensitive

try:

steps = int(cmd[1:]) # convert remaining string to integer

except ValueError:

continue # skip tokens with non-integer steps

# Update coordinates based on direction

if direction == 'N':

y += steps

elif direction == 'S':

y -= steps

elif direction == 'E':

x += steps

elif direction == 'W':

x -= steps

else:

continue # ignore any other invalid direction

return (x, y) # final coordinates after processing all commands

**5)README.md:**

# M.2 [S18M2] - Grid Movement Simulator

## Approach

- Initialize (x, y) = (0, 0)

- Parse each command: first letter = direction, remaining = steps

- Ignore invalid tokens or invalid numbers

- Update coordinates based on N/S/E/W

- Return final coordinates as tuple

## Assumptions

- Steps are integers

- Direction letters are case-insensitive

- Invalid tokens ignored

## Complexity

- Time: O(n) for n commands

- Space: O(1)

## Run Program

python solution.py

## Run Tests

python -m unittest tests.py

**6)Debugging / Refactor Note (Before / After):**

Before:

- Initially tried parsing movement commands without validation.

- Program failed or gave errors when tokens were invalid or steps were not integers.

After:

- Added checks for:

\* Token length < 2 → ignored

\* Invalid direction letters → ignored

\* Non-integer step values → ignored

- Now handles all invalid cases gracefully.

- Program correctly returns final (x, y) coordinates for valid commands.