

# AI ASSISTED CODING LAS TEST-2

2403A51276

Mohammed Ibtahim Hussain

## 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.

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

### Prompt Given:

I need a Python program that reads a CSV file of employees with columns name, dept, salary, sorts the data by department (ascending) and salary (descending), and writes the result to a new CSV. The sort must be stable. Please give me:

- A complete Python script.
- Step-by-step instructions for running it in Windows Command Prompt (with spaces in folder names).
- Example input CSV and the exact expected sorted output.
- Edge case handling if the CSV is missing the dept or salary column.
- A version that lets me run with just python script.py without needing to type file names.

## Initial csv file:

```
employees.csv X move_parser.py sorted.csv sort_emp.py
employees.csv
1 name,dept,salary
2 Raj,Eng,120
3 Maya,HR,90
4 Abi,Eng,110
```

## Code Given by AI:

```
employees.csv move_parser.py sorted.csv sort_emp.py
sort_emp.py > ...
1 import csv
2 import argparse
3 import sys
4
5 EXPECTED_KEYS = ['name', 'dept', 'salary']
6
7 def parse_args():
8     p = argparse.ArgumentParser()
9     p.add_argument('input_csv', help='Path to input CSV (must contain name,dept,salary header)')
10    p.add_argument('output_csv', help='path to output CSV (rows will be written there)')
11    p.add_argument('--with-header', action='store_true', help='Include header in output file')
12    p.add_argument('--on-invalid-salary', choices=['zero','error'], default='zero',
13                  help='If salary is not a valid integer: treat as 0 or raise an error')
14    return p.parse_args()
15
16 def load_and_normalize(input_path, on_invalid_salary='zero'):
17     rows = []
18     with open(input_path, newline='', encoding='utf-8') as f:
19         reader = csv.DictReader(f)
20         if reader.fieldnames is None:
21             raise ValueError("Input CSV has no header row.")
22         # normalize header names -> find required columns (case-insensitive)
23         hdr_map = {h.strip().lower(): h for h in reader.fieldnames}
24         missing = [k for k in EXPECTED_KEYS if k not in hdr_map]
25         if missing:
26             raise ValueError(f"Missing required columns in input CSV: {missing}. Found headers: {reader.fieldnames}")
27         # original header mapping (exact original header names)
28         name_h = hdr_map['name']
29         dept_h = hdr_map['dept']
30         salary_h = hdr_map['salary']
31
32         for idx, raw in enumerate(reader):
33             # read the three fields in a normalized way
34             name = (raw.get(name_h) or '').strip()
35             dept = (raw.get(dept_h) or '').strip()
36             salary_raw = (raw.get(salary_h) or '').strip()
37             # parse salary to int
38             try:
39                 salary_int = int(salary_raw)
40             except Exception:
41                 if on_invalid_salary == 'error':
42                     raise ValueError(f"Invalid salary value on row {idx+2}: {salary_raw!r}")
43                 salary_int = 0
44             rows.append({
45                 'name': name,
46                 'dept': dept,
47                 'salary': salary_raw, # keep original text for writing
48                 '_salary_int': salary_int, # parsed int for sorting
49                 '_idx': idx # original position for stable tie-break
50             })
51     return rows
52
53 def stable_sort_rows(rows):
54     # Key: dept asc, salary desc (-_salary_int), then original index for deterministic ties
55     return sorted(rows, key=lambda r: (r['dept'], -r['_salary_int'], r['_idx']))
56
57 def write_output(rows, output_path, with_header=False):
58     # We will output columns in order name,dept,salary to match the task
59     with open(output_path, 'w', newline='', encoding='utf-8') as f:
60         writer = csv.writer(f, lineterminator='\n')
61         if with_header:
62             writer.writerow(EXPECTED_KEYS)
63         for r in rows:
64             writer.writerow([r['name'], r['dept'], r['salary']])
65
```

```
employees.csv  move_parser.py  sorted.csv  sort_emp.py
sort_emp.py > ...
16 def load_and_normalize(input_path, on_invalid_salary='zero'):
46     'dept': dept,
47     'salary': salary_raw, # keep original text for writing
48     'salary_int': salary_int, # parsed int for sorting
49     '_idx': idx # original position for stable tie-break
50 })
51 return rows
52
53 def stable_sort_rows(rows):
54     # Key: dept asc, salary desc (-_salary_int), then original index for deterministic ties
55     return sorted(rows, key=lambda r: (r['dept'], -r['_salary_int'], r['_idx']))
56
57 def write_output(rows, output_path, with_header=False):
58     # We will output columns in order name,dept,salary to match the task
59     with open(output_path, 'w', newline='', encoding='utf-8') as f:
60         writer = csv.writer(f, lineterminator='\n')
61         if with_header:
62             writer.writerow(EXPECTED_KEYS)
63         for r in rows:
64             writer.writerow([r['name'], r['dept'], r['salary']])
65
66 def main():
67     args = parse_args()
68     rows = load_and_normalize(args.input_csv, on_invalid_salary=args.on_invalid_salary)
69     sorted_rows = stable_sort_rows(rows)
70     write_output(sorted_rows, args.output_csv, args.with_header)
71     print(f"Wrote {len(sorted_rows)} sorted rows to {args.output_csv}")
72
73 if __name__ == '__main__':
74     main()
75
```

## Output:

New CSV file created after sorting them in order.

```
employees.csv  move_parser.py  sorted.csv  sort_emp.py
sorted.csv
1  name,dept,salary
2  Raj,Eng,120
3  Abi,Eng,110
4  Maya,HR,90
5
```

## 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).

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

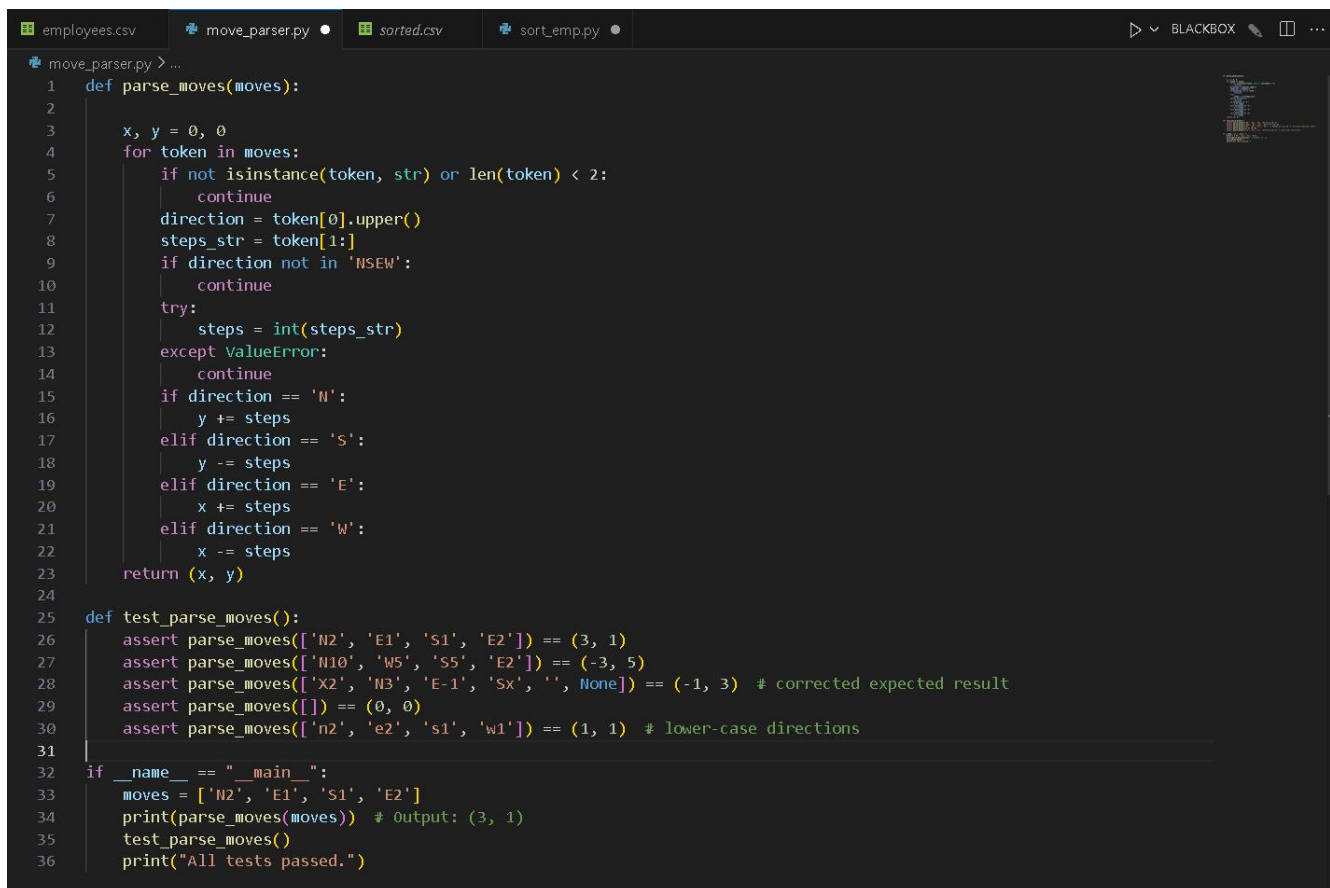
(3,1) Acceptance Criteria: Validates tokens

## Prompt Given:

I need a Python script that simulates grid movements. The input is a list of tokens like ['N2', 'E1', 'S1', 'E2']. Each token is a direction (N/E/S/W, case-insensitive) followed by a positive integer. Invalid tokens should be ignored. The script should compute the final (x,y) coordinate starting from (0,0). Please provide:

- A complete Python script that works in Windows Command Prompt.
- Step-by-step instructions for saving and running the script.
- Sample runs with both valid and invalid tokens, with expected outputs.
- Edge case handling for inputs like N0, n5, W, empty list, or non-string tokens.
- A version where I can run `python move.py N2 E1 S1 E2` from the command line."

## Code Given by AI:



```
1 def parse_moves(moves):
2
3     x, y = 0, 0
4     for token in moves:
5         if not isinstance(token, str) or len(token) < 2:
6             continue
7         direction = token[0].upper()
8         steps_str = token[1:]
9         if direction not in 'NSEW':
10            continue
11        try:
12            steps = int(steps_str)
13        except ValueError:
14            continue
15        if direction == 'N':
16            y += steps
17        elif direction == 'S':
18            y -= steps
19        elif direction == 'E':
20            x += steps
21        elif direction == 'W':
22            x -= steps
23    return (x, y)
24
25 def test_parse_moves():
26     assert parse_moves(['N2', 'E1', 'S1', 'E2']) == (3, 1)
27     assert parse_moves(['N10', 'W5', 'S5', 'E2']) == (-3, 5)
28     assert parse_moves(['X2', 'N3', 'E-1', 'Sx', '', None]) == (-1, 3) # corrected expected result
29     assert parse_moves([]) == (0, 0)
30     assert parse_moves(['n2', 'e2', 's1', 'w1']) == (1, 1) # lower-case directions
31
32 if __name__ == "__main__":
33     moves = ['N2', 'E1', 'S1', 'E2']
34     print(parse_moves(moves)) # Output: (3, 1)
35     test_parse_moves()
36     print("All tests passed.")
```

## Output:



```
[Done] exited with code=1 in 0.117 seconds

[Running] python -u "c:\Users\96653\OneDrive\Desktop\AI LAB TEST 2\move_parser.py"
(3, 1)
All tests passed.
```