

Ai Assisted Coding

Lab Exam-2

Sub Group- M

Name: Syed Akbarpasha

Enroll no:2403a52093

Batch=04

M.1:

Task:

Sort employees by dept ascending and salary descending (stable), and re-emit CSV.

Prompt:

Write a Python script that reads employee data from a CSV file with columns name, dept, and salary.

Use `csv.DictReader` to parse the input and `csv.DictWriter` to output the sorted data. Sort the employees by department in ascending order and, within each department, by salary in descending order. Ensure the sort is stable. Output the sorted data as CSV

Code:

```
import csv

input_file = "employees.csv"
output_file = "sorted_employees.csv"

with open(input_file, newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    employees = list(reader)

# Convert salary to int for sorting
for emp in employees:
    emp['salary'] = int(emp['salary'])

# Stable sort: dept asc, salary desc
employees_sorted = sorted(
    employees,
    key=lambda x: (x['dept'], -x['salary'])
)

with open(output_file, 'w', newline='') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames=['name', 'dept', 'salary'])
    writer.writeheader()
    for emp in employees_sorted:
        writer.writerow({
            'name': emp['name'],
            'dept': emp['dept'],
            'salary': emp['salary']
        })

# Print sorted employees to console
for emp in employees_sorted:
    print(f"{emp['name']},{emp['dept']},{emp['salary']}")
```

Input:

Employees.csv file:

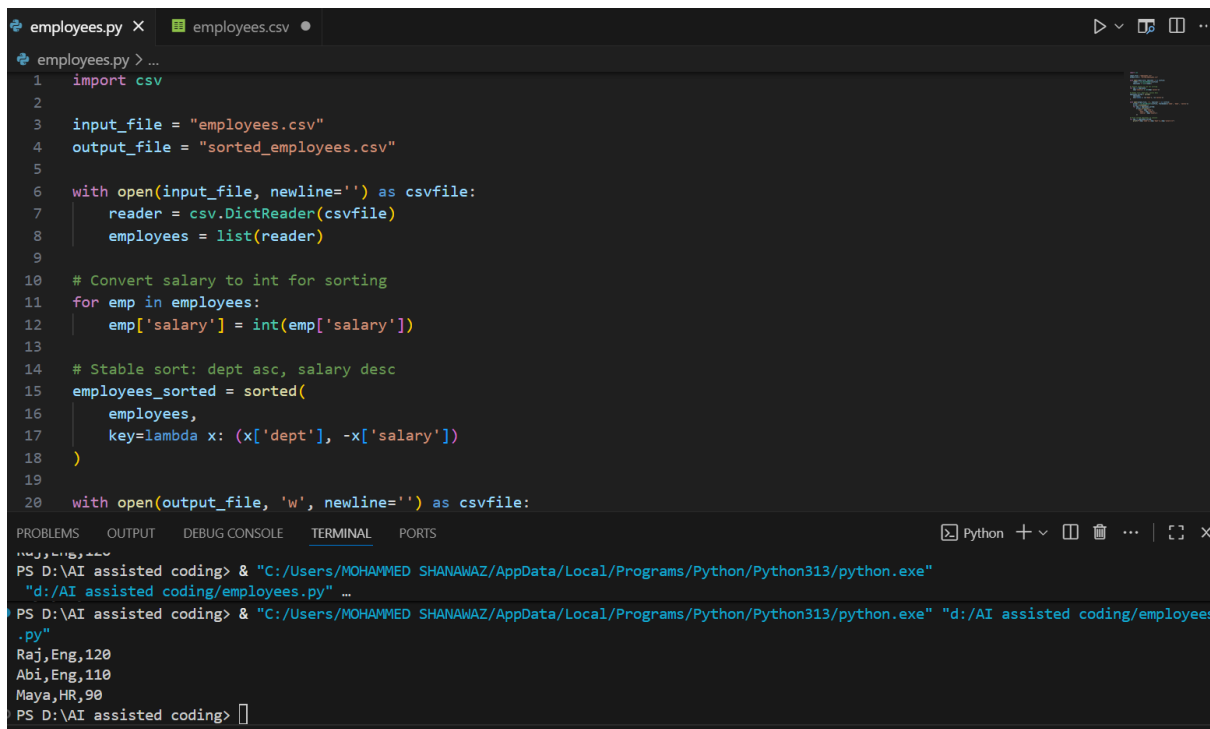
name,dept,salary

Raj,Eng,120

Maya,HR,90

Abi,Eng,110

Output:



The screenshot shows a VS Code editor with a Python script named `employees.py` and a terminal window. The script reads a CSV file `employees.csv`, sorts it by department (ascending) and salary (descending), and writes the result to `sorted_employees.csv`. The terminal shows the command to run the script and the resulting output.

```
employees.py x employees.csv
employees.py > ...
1 import csv
2
3 input_file = "employees.csv"
4 output_file = "sorted_employees.csv"
5
6 with open(input_file, newline='') as csvfile:
7     reader = csv.DictReader(csvfile)
8     employees = list(reader)
9
10 # Convert salary to int for sorting
11 for emp in employees:
12     emp['salary'] = int(emp['salary'])
13
14 # Stable sort: dept asc, salary desc
15 employees_sorted = sorted(
16     employees,
17     key=lambda x: (x['dept'], -x['salary'])
18 )
19
20 with open(output_file, 'w', newline='') as csvfile:
```

```
PS D:\AI assisted coding> & "C:/Users/MOHAMMED SHANAWAZ/AppData/Local/Programs/Python/Python313/python.exe"
"d:/AI assisted coding/employees.py" ...
PS D:\AI assisted coding> & "C:/Users/MOHAMMED SHANAWAZ/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI assisted coding/employees
.py"
Raj,Eng,120
Abi,Eng,110
Maya,HR,90
PS D:\AI assisted coding>
```

Observation:

The provided Python script efficiently sorts employee records from a CSV file by department in ascending order and salary in descending order, using a stable sort to preserve the original order for salary ties. It leverages `csv.DictReader` for input and `csv.DictWriter` for output, ensuring the header is retained and the output format matches the input. This approach is

suitable for payroll audits requiring deterministic and reproducible sorting.

M.2:

Task:

Parse commands like N2, E1, S3, W4, validate them, and compute final (x,y)

Prompt:

Write a Python function that takes a list of movement commands (e.g., ['N2', 'E1', 'S1', 'E2']) and computes the final position (x, y) of an agent starting at (0, 0) on a grid. Each command consists of a direction (N, E, S, W) followed by a positive integer. Ignore invalid tokens. N increases y, E increases x, S decreases y, and W decreases x. Return the final (x, y) tuple. Add tests to ensure invalid tokens are ignored.

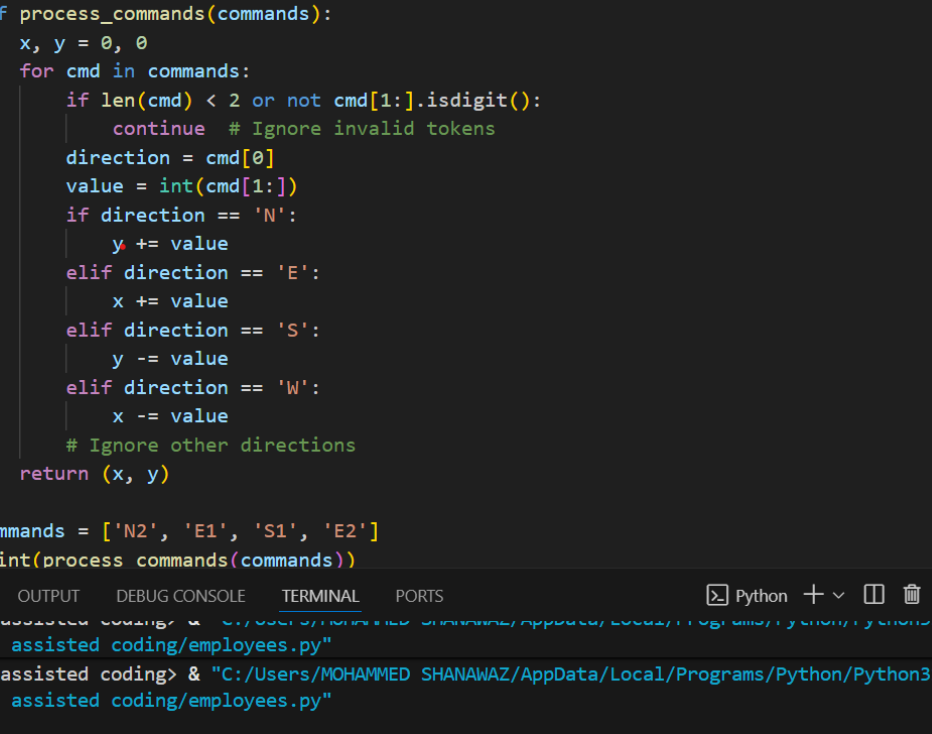
Code:

```
def process_commands(commands):
    x, y = 0, 0
    for cmd in commands:
        if len(cmd) < 2 or not cmd[1:].isdigit():
            continue # Ignore invalid tokens
        direction = cmd[0]
        value = int(cmd[1:])
        if direction == 'N':
            y += value
        elif direction == 'E':
            x += value
        elif direction == 'S':
            y -= value
        elif direction == 'W':
            x -= value
        # Ignore other directions
    return (x, y)
```

Input:

```
commands = ['N2', 'E1', 'S1', 'E2']
print(process_commands(commands))
```

Output:



The image shows a VS Code editor window with a file named `employees.py`. The script defines a function `process_commands` that takes a list of commands and processes them based on direction (N, E, S, W) and value. The terminal shows the script being executed, resulting in the output `(3, 1)`.

```

employees.py
1 def process_commands(commands):
2     x, y = 0, 0
3     for cmd in commands:
4         if len(cmd) < 2 or not cmd[1:].isdigit():
5             continue # Ignore invalid tokens
6         direction = cmd[0]
7         value = int(cmd[1:])
8         if direction == 'N':
9             y += value
10        elif direction == 'E':
11            x += value
12        elif direction == 'S':
13            y -= value
14        elif direction == 'W':
15            x -= value
16        # Ignore other directions
17    return (x, y)
18
19 commands = ['N2', 'E1', 'S1', 'E2']
20 print(process_commands(commands))

```

```

PS D:\AI assisted coding> & "C:/Users/MOHAMMED SHANAWAZ/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI assisted coding/employees.py"
(3, 1)
PS D:\AI assisted coding> & "C:/Users/MOHAMMED SHANAWAZ/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI assisted coding/employees.py"
(3, 1)
PS D:\AI assisted coding>

```

Observation:

The function must correctly parse and validate each command, updating the agent's position only for valid tokens. It should ignore any malformed or invalid commands, ensuring robustness. The final output should reflect the cumulative effect of all valid movements.

