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:

```python
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']}")
```

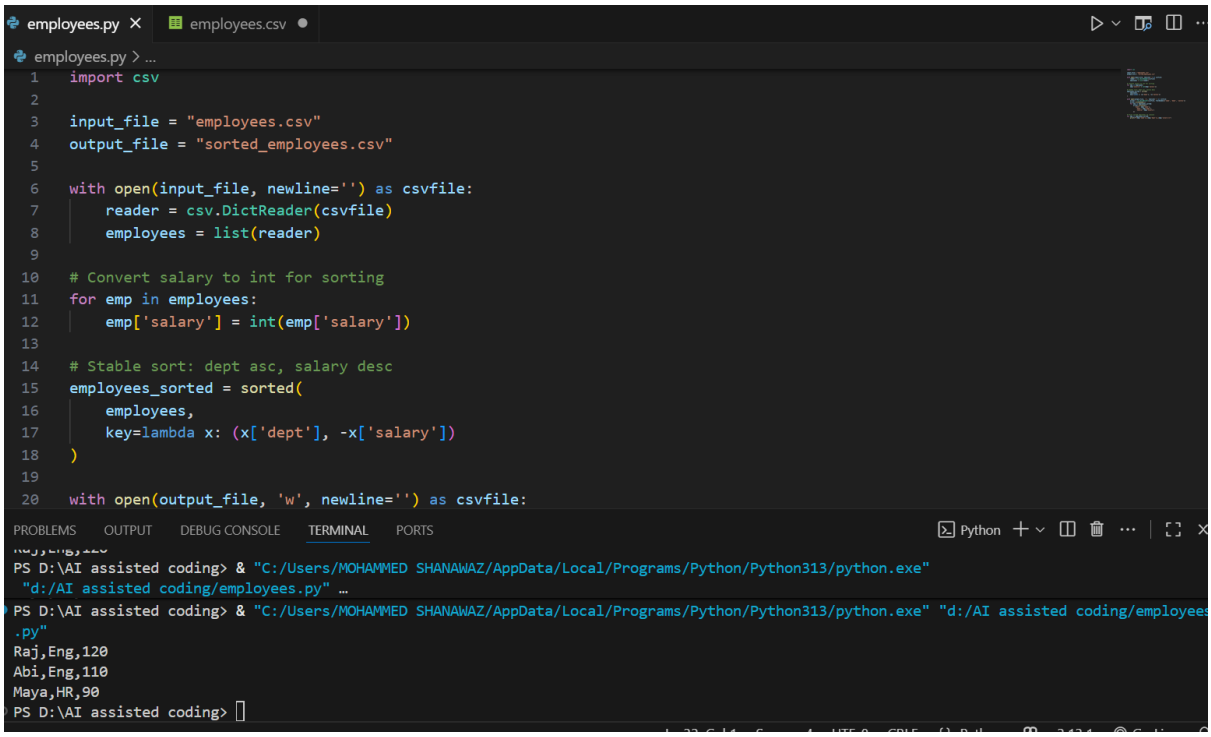
Input:

Employees.csv file:

```csv
name,dept,salary

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

Output:



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:

```python
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)

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