Electron Corp – Python Internship Tasks (FULL DOCUMENTATION)

Company:

Electron Corp is an IT & Cybersecurity company focused on building secure, scalable, and maintainable software systems.

This task set is designed to evaluate:
- Python fundamentals
- Algorithmic thinking
- Data structures
- Edge case handling
- Code clarity and documentation
- Ability to follow specifications

Language:

Python 3.x (standard library only unless stated otherwise)

Difficulty Progression:

Tasks are ordered from easiest to hardest.

TASK 1 — Merge and Sort Two Lists

Description:

Write a function that merges two lists of integers and returns a new list sorted in ascending order.

Function Signature:

```
merge_and_sort(a: list[int], b: list[int]) -> list[int]
```

Rules:

- Do not modify the original lists
- The result must be a new list
- Sorting must be numeric

Edge Cases:

- One or both lists empty
- Negative values
- Duplicate values

Example:

Input:
```
a = [5, 3, 8]
b = [1, 7, 4]
```

Output:
```
[1, 3, 4, 5, 7, 8]
```

Test Cases:

1) [3,1,2], [5,4] → [1,2,3,4,5]
2) [], [3,2,1] → [1,2,3]
3) [-3,-1], [-2,0] → [-3,-2,-1,0]
4) [1,1,2], [2,3] → [1,1,2,2,3]
5) [100], [] → [100]

==
TASK 2 — Palindrome Checker
==

Description:

Determine whether a given string is a palindrome.
A palindrome reads the same forwards and backwards.

Rules:

- Ignore case
- Ignore non-alphanumeric characters
- Empty string is a palindrome

Function Signature:
----
is_palindrome(text: str) -> bool

Example:

"A man, a plan, a canal: Panama" → True

Test Cases:
----
1) "racecar" → True
2) "RaceCar" → True
3) "Hello" → False
4) "" → True
5) "12321" → True
6) "No 'x' in Nixon" → True
7) "abc123" → False

==
TASK 3 — Word Occurrence Counter
==

Description:

Count how many times each word appears in a string.
Words are separated by whitespace.
Punctuation must be preserved.

Function Signature:
----
count_words(text: str) -> dict[str, int]

Rules:

- Case-sensitive
- Multiple spaces should be ignored

Example:

"hello world hello"

→ {"hello": 2, "world": 1}

Test Cases:
----
1) "apple banana apple" → {"apple":2,"banana":1}
2) "word" → {"word":1}
3) "" → {}
4) "a a   b" → {"a":2,"b":1}
5) "Hello hello" → {"Hello":1,"hello":1}

===
TASK 4 — Difference From Average
===

Description:

Given a list of numbers, calculate the difference between each number
and the average of the list.

Formula:
----
difference = value - average

Function Signature:
----
diff_from_average(data: list[float]) -> list[float]

Rules:

- Round results to 2 decimal places
- Empty list returns empty list

Example:

Input: [55,95,62,36,48]
Output: [-4.20, 35.80, 2.80, -23.20, -11.20]

Test Cases:
----
1) [10,10,10] → [0.00,0.00,0.00]
2) [1.5,2.25] → [-0.38,0.38]
3) [] → []
4) [100] → [0.00]

===
TASK 5 — Find Duplicate Elements
===

Description:

Return all values that appear more than once in a list.
The result must be sorted and contain unique values only.

Function Signature:
----
find_duplicates(data: list) -> list

Rules:

- Preserve original data types
- Output must be sorted

Example:

[1,2,3,2,1] → [1,2]

Test Cases:
----
1) [4,5,6,5,4,7] → [4,5]
2) [8,9,10] → []
3) [] → []
4) [2,2,2] → [2]
5) ["a","b","a","b"] → ["a","b"]

==
TASK 6 — Validate Parentheses
==

Description:

Check whether a string containing brackets is valid.

Valid pairs:

(), {}, []

Rules:

- Brackets must close in correct order
- Empty string is valid

Function Signature:
----
is_valid_brackets(s: str) -> bool

Example:

"{[]}" → True

Test Cases:
----
1) "()" → True
2) "()[]{}" → True
3) "(]" → False
4) "([)]" → False
5) "{[]}" → True
6) "" → True
7) "(((((" → False

==
TASK 7 — Reconstruct the Coffee Line
==

Description:

Carrol was first in line.
Each person remembers how many people stood between them and Carrol.

Input:

A list mem where mem[i] indicates how many people were between person (i+1)
and Carrol.

Output:

The reconstructed order (excluding Carrol) or [] if impossible.

Function Signature:
----
reconstruct_line(mem: list[int]) -> list[int]

Rules:

- Person IDs are 1..n
- If multiple solutions exist, return any
- Return [] if no valid solution

Example:

Input: [1,2,0]
Output: [3,1,2]

Test Cases:
----
1) [1,2,0] → [3,1,2]
2) [1,0,1] → []
3) [0] → [1]
4) [0,0] → []
5) [2,0,1,0] → []

==
TASK 8 — Pickaxe Trading Optimization
==

Description:

Decide which pickaxe to buy and which to sell to obtain the strongest
possible pickaxe within budget constraints.

Function Signature:
----
trade_pickaxe(materials, store, inventory, budget) -> tuple | None

Rules:

- Buy only if strictly stronger than current best
- Sell weakest pickaxes first
- Cannot sell unknown or worthless items
- Budget must never go below zero

Test Cases:
----
1) Provided example → ("Platinum", {"Copper"}, 25)
2) Inventory empty, exact budget → buy strongest
3) Only unsellable items → None
4) Multiple sales required → valid result

==
TASK 9 — Chess Move Executor
==

Description:

Apply a sequence of chess moves to a board and return the final board.

Rules:

- Moves are guaranteed valid
- Must support:
  - Normal moves
  - Captures
  - Castling
  - Pawn promotion
  - En-passant

Function Signature:
----
apply_moves(board, moves) -> board

Test Cases:
----
1) Provided example
2) Castling only
3) Promotion with capture
4) En-passant scenario

==
TASK 10 — REST-like Log Analyzer (MINI PROJECT)
==

Goal:

Build a Python program that analyzes HTTP-style logs and generates a JSON summary.

Required Output:

- Total requests
- Requests per endpoint
- Average response time per endpoint
- Top 5 slowest endpoints

Log Format:

```
<TIMESTAMP> <METHOD> <PATH> <STATUS> <RESPONSE_TIME_MS>
```

Example:

```
2025-01-01T10:00:00Z GET /api/login 200 134
```

Deliverables:

- CLI program
- JSON output file
- Unit tests
- Integration tests
- Documentation

Django?

Django is NOT required.
Optional for bonus points if API endpoints are exposed.

Testing:

- pytest for unit tests
- CLI integration tests
- Sample log files

Evaluation:

Correctness (50%)
Code Quality (25%)
Testing (15%)
Extras (10%)

END OF DOCUMENT