# Name : Hinge Chanakya Kumar

## Enrollment no: 2403A51T02

## BATCH – 1 CSE CSE

## Set – O

# Question Paper Set A13

AI-Assisted Coding Exam — Python (1 hour)  
- You may either (a) write the exact prompt to have your AI assistant generate code, or (b) write the code yourself. If you use AI, submit the prompt and the final code you executed.  
- Use VS Code Copilot / Gemini / CursorAI. Credit your prompts.  
- Prefer Python standard library; write clear, tested, well-documented code.  
- Each subgroup (A–O) has two tasks. Attempt all. Medium complexity, use-case based.  
- Syllabus pillars:  
 A–C: AI completion (classes, loops, conditionals)  
 D–E: Debugging with AI  
 F–G: TDD with AI  
 H–I: Documentation & Code Review  
 J–K: Code Quality & Performance  
 L–M: Files/CSV & Regex  
 N–O: Algorithms & Data Structures

Deliverables for each question  
1) If using AI: the exact prompt you issued. If manual: note 'manual' and a brief design reason.  
2) solution.py  
3) tests.py (unittest/pytest; write tests first for TDD items)  
4) Docstrings & inline comments (AI-assisted allowed)  
5) Short README.md (approach, assumptions, complexity, run tests)  
6) For debugging/refactor: brief before/after note

Question Paper Set A13

AI-Assisted Coding Exam — Python (1 hour)

- You may either (a) write the exact prompt to have your AI assistant generate code, or (b) write the code yourself. If you use AI, submit the prompt and the final code you executed.

- Use VS Code Copilot / Gemini / CursorAI. Credit your prompts.

- Prefer Python standard library; write clear, tested, well-documented code.

- Each subgroup (A–O) has two tasks. Attempt all. Medium complexity, use-case based.

- Syllabus pillars:

A–C: AI completion (classes, loops, conditionals)

D–E: Debugging with AI

F–G: TDD with AI

H–I: Documentation & Code Review

J–K: Code Quality & Performance

L–M: Files/CSV & Regex

N–O: Algorithms & Data Structures

Deliverables for each question

1) If using AI: the exact prompt you issued. If manual: note 'manual' and a brief design reason.

2) solution.py

3) tests.py (unittest/pytest; write tests first for TDD items)

4) Docstrings & inline comments (AI-assisted allowed)

5) Short README.md (approach, assumptions, complexity, run tests)

6) For debugging/refactor: brief before/after note

## Subgroup A

### A.1 — [S13A1] Compute per-tractor average from logs (AI completion)

Scenario (agritech):

Context:

You are integrating a agritech telemetry service where each tractor emits periodic measures as CSV lines: `id,timestamp,soil\_moist`. Due to flaky connectivity, some lines may be truncated or contain non-numeric values. Ops needs a quick aggregation for dashboards and alert thresholds.

Your Task:

Write a Python function to parse the raw text (multiple lines) and compute per-tractor averages of `soil\_moist`. Return a dict {id: avg} and separately compute an overall average.

Data & Edge Cases:

Input contains newlines, optional leading/trailing spaces, and may include malformed rows. Timestamps are ISO-8601 but not needed for math.

AI Assistance Expectation:

Use AI code completion to scaffold the loop, dictionary accumulation (sum and count), and exception handling for malformed rows.

Constraints & Notes:

Prefer O(n) pass; ignore lines that cannot be split into three fields or have non-numeric metric; round averages to 2 decimals.

Sample Input

tr131,2025-01-01T08:00,32.7  
tr132,2025-01-02T09:00,34.2  
tr133,2025-01-03T010:00,35.7

Sample Output

{'tr131': 32.7, 'tr132': 34.2, 'tr133': 35.7} and overall\_avg=34.2

Acceptance Criteria: Correct averages per ID; overall average reported; malformed lines skipped

### A.2 — [S13A2] Implement YieldTracker with add/remove/summary (AI completion)

Scenario (agritech):

Context:

A microservice in the agritech platform maintains a small in-memory structure to track values keyed by identifier (e.g., order IDs, sensor IDs). Engineers want a minimal class to add, remove, and summarize current values for quick health checks.

Your Task:

Implement a `YieldTracker` class with methods `add(id: str, value: float)`, `remove(id: str)`, and `summary() -> tuple[int, float|None]` returning (count, average).

Data & Edge Cases:

IDs are unique keys. Re-adding the same ID overwrites its value. Removing a missing ID should be safe (no exception). For an empty store, average is None.

AI Assistance Expectation:

Ask AI to generate the class skeleton with docstrings and type hints, then refine method behavior and add a quick usage example.

Constraints & Notes:

Keep state in a dict; O(1) per operation; return rounded average to 2 decimals (when non-empty).

Sample Input

[{'op': 'add', 'id': 'a1', 'value': 22}, {'op': 'add', 'id': 'b2', 'value': 17}, {'op': 'remove', 'id': 'a1'}, {'op': 'add', 'id': 'c3', 'value': 19}]

Sample Output

count=2, avg=18.0

Acceptance Criteria: Handles add/remove; correct count and average; safe on missing IDs

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

## Subgroup B

### B.1 — [S13B1] Apply surge/penalty rules (conditionals)

Scenario (agritech):

Context:

Pricing in the agritech app uses a base per-km rate and time-based surge after business peaks. Product wants a deterministic calculator for receipts and audits.

Your Task:

Implement a fare function: fare = km \* base\_per\_km \* surgeMultiplier, where surge applies strictly after 18:00 local time.

Data & Edge Cases:

Input is a list of rides with `time` as HH:MM (24h) and `km` as float. Edge case: exactly at 18:00 should be treated as non-surge for 18:00:00; after 18:00 (e.g., 18:01) surges.

AI Assistance Expectation:

Prompt AI to outline parsing HH:MM, applying conditionals, and rounding to 2 decimals; then implement and write a quick test.

Constraints & Notes:

No external libraries; round each fare to 2 decimals; do not mutate input.

Sample Input

[{'time': '08:00', 'km': 3.0}, {'time': '18:30', 'km': 5.0}]

Sample Output

[66.0, 132.0]

Acceptance Criteria: Correct surge threshold and rounding

### B.2 — [S13B2] Debug rolling mean (off-by-one)

Scenario (agritech):

Context:

A team in agritech noticed off-by-one bugs in a rolling KPI computation (moving averages) that undercount windows.

Your Task:

Use AI to identify the bug and fix the window iteration so all valid windows are included.

Data & Edge Cases:

For xs=[13, 14, 15, 16] and w=2, number of windows should be len(xs)-w+1.

AI Assistance Expectation:

Ask AI to add a failing test first, propose the minimal fix, and verify with the sample.

Constraints & Notes:

Guard invalid w (<=0 or >len(xs)); preserve O(n\*w) simple solution.

Sample Input

xs=[13, 14, 15, 16], w=2  
Buggy code:  
  
def rolling\_mean(xs, w):  
 sums = []  
 for i in range(len(xs)-w):  
 window = xs[i:i+w]  
 sums.append(sum(window)/w)  
 return sums

Sample Output

[13.5, 14.5, 15.5]

Acceptance Criteria: All valid windows included; passes tests; no index errors

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

## Subgroup C

### C.1 — [S13C1] Debug de-duplication (case-insensitive)

Scenario (agritech):

Context:

Customer contact lists in the agritech CRM contain duplicates differing only by case (e.g., 'A@x.com' vs 'a@x.com').

Your Task:

Write a function that returns the first occurrence of each email (case-insensitive) while preserving the original order.

Data & Edge Cases:

Input: list of emails. Normalize for comparison using lowercase; keep the original cased value for output.

AI Assistance Expectation:

Use AI to spot the bug (reinitializing `seen` in a loop) and propose a corrected, stable algorithm.

Constraints & Notes:

Include unit tests covering: ['A@x.com','a@x.com','B@y.com'] -> ['A@x.com','B@y.com']

Sample Input

['A@x.com', 'a@x.com', 'B@y.com']

Sample Output

['A@x.com', 'B@y.com']

Acceptance Criteria: Preserves first occurrence order; case-insensitive matching

### C.2 — [S13C2] TDD: slugify titles

Scenario (agritech):

Context:

Content titles in the agritech CMS must become SEO-friendly slugs for URLs.

Your Task:

Design tests first for slugify(text) then implement: lowercase, remove non-alnum except hyphen, spaces->hyphen, collapse multiple hyphens, trim hyphens.

Data & Edge Cases:

Test punctuation, multiple spaces, and boundary hyphens.

AI Assistance Expectation:

Use AI to generate parameterized tests (pytest) and then implement a regex-based slugify.

Constraints & Notes:

Return correct slugs for provided samples.

Sample Input

['Hello World!', 'AI & You', 'Set13-C2']

Sample Output

['hello-world', 'ai-you', 'set13-C2']

Acceptance Criteria: All tests pass; edge cases covered

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

## Subgroup D

### D.1 — [S13D1] TDD: increment version suffix

Scenario (agritech):

Context:

File versioning in the agritech data pipeline uses a `\_vNN` suffix before the extension.

Your Task:

Create tests and implement bump\_version(name) that adds or increments `\_vNN` with zero-padding.

Data & Edge Cases:

Handle names with and without existing suffix; preserve original extension.

AI Assistance Expectation:

Use AI to propose regex and test cases for edge names like `report\_v9.csv`, `summary.csv`.

Constraints & Notes:

Preserve original extension and base name.

Sample Input

['report\_v1.csv', 'summary.csv', 'log\_v09.txt']

Sample Output

['report\_v02.csv', 'summary\_v01.csv', 'log\_v10.txt']

Acceptance Criteria: Correct zero-padding; extension preserved

### D.2 — [S13D2] Generate docstrings and usage examples

Scenario (agritech):

Context:

Data analysts in agritech normalize metrics to [0,1] for comparability.

Your Task:

Add Google-style docstrings and handle the edge-case where all scores are equal (avoid divide-by-zero).

Data & Edge Cases:

Empty lists return empty; if max==min, return zeros of the same length.

AI Assistance Expectation:

Use AI to draft docstrings with Args/Returns/Examples and generate unit tests for edge-cases.

Constraints & Notes:

Add tests demonstrating the m==n case.

Sample Input

def normalize(scores):  
 m = max(scores); n = min(scores)  
 return [(x-n)/(m-n) for x in scores]

Sample Output

Docstring includes Args/Returns/Examples; guard for m==n

Acceptance Criteria: Doc quality and guard confirmed by tests

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

## Subgroup E

## ****P.1 — [S01P1] Validate and Mask Credit Card Numbers****

### ****Scenario (E-commerce):****

**Context:**  
In an e-commerce platform, customer credit card numbers are stored temporarily before being passed to a payment gateway. For security, only the **last 4 digits** should be visible to support staff during troubleshooting.

### ****Your Task:****

* Write a function that:
  1. Validates if a given string is a **16-digit numeric credit card number**.
  2. Returns a **masked version** where only the last 4 digits are visible, and the rest are replaced with \*.

### ****Data & Edge Cases:****

* Input may contain non-numeric characters → reject as invalid.
* Input shorter or longer than 16 digits → reject as invalid.
* Return "Invalid Card" in invalid cases.

### ****AI Assistance Expectation:****

* Use AI to scaffold:
  + Regex check for 16-digit numeric input.
  + String slicing and masking logic.
* Generate tests for valid/invalid card numbers.

### ****Constraints & Notes:****

* Do not use external libraries; stick to built-in Python.
* Ensure the function is reusable for lists of card numbers.

### ****Sample Input:****

card = "1234567812345678"

### ****Sample Output:****

\*\*\*\*\*\*\*\*\*\*\*\*5678

### ****Acceptance Criteria:****

* Returns masked string correctly for valid 16-digit inputs.
* Returns "Invalid Card" for invalid inputs.
* Handles multiple test cases in a list of cards.

### E.2 — [S13E2] Refactor nested loops to dict aggregation

Scenario (agritech):

Context:

A legacy agritech script uses verbose loops to aggregate key-value tuples.

Your Task:

Refactor to a pythonic aggregation using dict.get or collections.defaultdict with type hints.

Data & Edge Cases:

Input example: [('a',1),('b',2),('a',3)] -> {'a':4,'b':2}.

AI Assistance Expectation:

Ask AI for refactor suggestions, then apply and ensure behavior parity via tests.

Constraints & Notes:

Type hints for function signatures required.

Sample Input

data=[('a',1),('b',2),('a',3)]

Sample Output

{'a':4,'b':2}

Acceptance Criteria: Behavior unchanged; improved readability

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

## Subgroup F

## ****F.1 — [S13F1] Refactor duplicate discount logic****

### ****Scenario (Agritech E-commerce):****

**Context:**  
An agritech marketplace applies seasonal discounts differently for seeds, fertilizers, and tools. Currently, each product type has **copy-pasted discount logic**, which makes the code hard to maintain.

### ****Your Task:****

* Refactor the discount calculation into a **single reusable function**.
* Ensure that product type determines the discount percentage.
* AI should assist in:
  1. Identifying duplicate logic.
  2. Suggesting a clean function with parameters.
  3. Adding documentation and meaningful names.

### ****Buggy / Duplicate Code (Given):****

def seed\_discount(price):

return price - price \* 0.1

def fert\_discount(price):

return price - price \* 0.15

def tool\_discount(price):

return price - price \* 0.05

### ****Data & Edge Cases:****

* Discounts: seeds → 10%, fertilizers → 15%, tools → 5%.
* Invalid category should raise ValueError.
* Must handle floating-point prices safely.

### ****AI Assistance Expectation:****

* Use AI to generate a **refactored single function**:
* def apply\_discount(price: float, category: str) -> float: ...
* AI should also generate **doctest or unit tests** for correctness.

### ****Constraints & Notes:****

* DRY (Don’t Repeat Yourself) principle must be applied.
* Include type hints in final function signature.
* Add inline docstring with discount table.

### ****Sample Input:****

print(apply\_discount(100, "fertilizer"))

### ****Sample Output:****

85.0

### ****Acceptance Criteria:****

* No duplicate discount functions.
* Handles all categories correctly.
* Raises error for unknown category.
* Clean, documented, testable code.

### F.2 — [S13F2] Optimize membership checks

Scenario (agritech):

Context:

A streaming job in agritech checks if IDs are in a large corpus.

Your Task:

Optimize membership checks by converting corpus to a set once, then mapping stream to booleans.

Data & Edge Cases:

corpus=[1,2,3,4,5]; stream=[2,5,9] -> [True, True, False].

AI Assistance Expectation:

Ask AI to suggest complexity improvements and micro-bench ideas.

Constraints & Notes:

Return list[bool] aligned to stream order.

Sample Input

corpus=[1,2,3,4,5], stream=[2,5,9]

Sample Output

[True, True, False]

Acceptance Criteria: Uses set; correct booleans

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

## Subgroup G

### G.1 — [S13G1] Sum CSV column ignoring bad rows

Scenario (agritech):

Context:

Ad-hoc CSV exports in agritech contain missing/invalid numeric fields.

Your Task:

Sum the 'value' column as int, skipping invalid rows, and report total (print skipped count optional).

Data & Edge Cases:

id,value

1,10

2,NA

3,7 -> 17 with one skipped.

AI Assistance Expectation:

Use AI to draft robust CSV parsing with try/except and tests.

Constraints & Notes:

Print or return count of skipped rows for transparency.

Sample Input

id,value  
1,10  
2,NA  
3,7

Sample Output

17

Acceptance Criteria: Skips invalid rows; correct total

### G.2 — [S13G2] Merge two CSVs by id

Scenario (agritech):

Context:

Two CSVs in agritech must be merged by 'id' for reporting.

Your Task:

Implement inner and left joins without pandas, following SQL semantics.

Data & Edge Cases:

A:id,price & B:id,qty -> inner join has only common ids; left join keeps all A with None for missing B.

AI Assistance Expectation:

Use AI to outline dict-building and join logic; write unit tests for both joins.

Constraints & Notes:

No external deps; stable output order preferred.

Sample Input

id,price  
A,10  
B,20  
---  
id,qty  
A,2  
C,5

Sample Output

inner=[('A',10,2)], left=[('A',10,2),('B',20,None)]

Acceptance Criteria: Correct join behavior

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

## Subgroup H

### H.1 — [S13H1] Extract hashtags and mentions

Scenario (agritech):

Context:

Moderation in the agritech app needs hashtag and mention extraction.

Your Task:

Use regex to extract @mentions and #hashtags (case-insensitive) and return lowercase lists.

Data & Edge Cases:

Punctuation around tags should be ignored.

AI Assistance Expectation:

Ask AI for a robust regex and tests covering multiple tags.

Constraints & Notes:

Return mentions and hashtags lists; lowercase.

Sample Input

Hello @alice check #AI and #Python with @Bob

Sample Output

mentions=['alice','bob'], hashtags=['ai','python']

Acceptance Criteria: Lowercased; ignores punctuation

### H.2 — [S13H2] Shortest path on weighted graph (Dijkstra)

Scenario (agritech):

Context:

Routing decisions in the agritech graph need shortest paths for small, weighted graphs.

Your Task:

Implement Dijkstra from a source node 'A' to all nodes using a priority queue.

Data & Edge Cases:

Use adjacency dict with positive weights.

AI Assistance Expectation:

Prompt AI to outline the algorithm steps and edge relaxation pattern.

Constraints & Notes:

Return dict of distances with 0 for source.

Sample Input

{'A':{'B':1,'C':4},'B':{'C':2,'D':5},'C':{'D':1},'D':{}}

Sample Output

{'A':0,'B':1,'C':3,'D':4}

Acceptance Criteria: Correct distances; stable for positive weights

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

## Subgroup I

### I.1 — [S13I1] Top-3 frequent words

Scenario (agritech):

Context:

Basic text analytics in agritech requires most frequent terms for summaries.

Your Task:

Return the top-3 words by frequency, breaking ties lexicographically.

Data & Edge Cases:

Normalize to lowercase, split on spaces; ignore punctuation for simplicity (optional).

AI Assistance Expectation:

Use AI to propose Counter/ sorting approach and tie-breaking mechanics.

Constraints & Notes:

Stable ordering by (-count, word).

Sample Input

to be or not to be that is the question

Sample Output

[('to', 2), ('be', 2), ('is', 1)]

Acceptance Criteria: Tie-breaking lexicographically

## ****I.2 — [S13I2] Generate Docstrings for Crop Yield Function****

### ****Scenario (Agritech):****

**Context:**  
An agritech analytics team has a function that calculates **average crop yield per acre**, but it lacks documentation. The function works but is not self-explanatory. New developers find it confusing.

### ****Your Task:****

* Write code **average crop yield per acre function, that took** total\_yield and acres as input and return **average crop yield per acre value.**
* Use AI assistance to generate a **clear, structured docstring** for the given function.
* Ensure the docstring covers:
  + **Parameters** (types and meaning)
  + **Return type**
  + **Example usage**
  + **Notes on edge cases**

### ****Data & Edge Cases:****

* total\_yield = 500, acres = 50 → 10
* Division by zero should be highlighted as a possible error.
* Input values should be numeric.

### ****AI Assistance Expectation:****

* Use AI to propose a **PEP-257 compliant docstring**.
* AI should also suggest **type hints** (float for return type).
* Generate **doctests** inside the docstring.

### ****Constraints & Notes:****

* Do not modify function logic, only add documentation & type hints.
* Keep docstring concise but informative.

### ****Sample Input:****

print(avg\_yield\_per\_acre(500, 50))

### ****Sample Output:****

10.0

### ****Acceptance Criteria:****

* Function includes a proper docstring.
* Type hints added:
* def avg\_yield\_per\_acre(total\_yield: float, acres: float) -> float
* Doctest runs correctly.
* Reviewed for readability and clarity.

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

## Subgroup J

### J.1 — [S13J3] Normalize sensor readings

**Scenario (agritech):**  
**Context:** Sensor data logs in agritech are stored as CSV-like text. Each line contains sensor\_id, timestamp, value. For downstream ML, values must be normalized.

**Your Task:**  
Parse the text into structured data, compute **z-score normalization** for each sensor’s values independently (per sensor\_id), and return results as a dict mapping sensor\_id → list of (timestamp, z\_value).

**Data & Edge Cases:**

* Input text may have blank lines.
* Each sensor\_id should be normalized independently.
* If a sensor has only one value, return z=0 for all its rows.

**AI Assistance Expectation:**  
Ask AI to suggest pandas/numpy vs pure Python approaches, discuss numerical stability, and test edge cases.

**Constraints & Notes:**

* Output as dict[str, list[tuple[str, float]]].
* Round z-scores to 3 decimals for consistency.

**Sample Input**

s1,2025-01-01T10:00,10

s1,2025-01-01T11:00,20

s2,2025-01-01T10:30,100

s2,2025-01-01T11:30,100

**Sample Output**

{

's1': [('2025-01-01T10:00', -1.0), ('2025-01-01T11:00', 1.0)],

's2': [('2025-01-01T10:30', 0.0), ('2025-01-01T11:30', 0.0)]

}

**Acceptance Criteria:**

* Correct per-sensor z-score calculation.
* Handles one-value case with zeros.
* Ignores blank lines safely.

### J.2 — [S13J4] Detect overlapping irrigation schedules

**Scenario (agritech):**  
**Context:** Smart irrigation systems schedule watering by time slots per field. Overlaps between schedules can cause overwatering or pressure issues.

**Your Task:**  
Given a list of schedules (field\_id, start\_time, end\_time in ISO format), detect **pairs of fields with overlapping watering times**.

**Data & Edge Cases:**

* Input is a list of dicts.
* Times are inclusive at start, exclusive at end.
* Return list of tuples (fieldA, fieldB) where overlap occurs.
* If no overlaps, return empty list.
* Ensure each pair is unique (no duplicates, order sorted lexicographically).

**AI Assistance Expectation:**  
Ask AI to propose interval overlap detection strategies (sorting by start, sweep-line, or O(n²) check depending on data size).

**Constraints & Notes:**

* Return list[tuple[str,str]].
* Assume all times in same timezone (no conversion needed).

**Sample Input**

[

{'field': 'F1', 'start': '2025-01-01T08:00', 'end': '2025-01-01T10:00'},

{'field': 'F2', 'start': '2025-01-01T09:30', 'end': '2025-01-01T11:00'},

{'field': 'F3', 'start': '2025-01-01T11:00', 'end': '2025-01-01T12:00'}

]

**Sample Output**

[('F1', 'F2')]

**Acceptance Criteria:**

* Correct detection of overlaps.
* Unique, sorted pairs.
* Works for multiple overlaps or none.

## Subgroup K

### K.1 — [S13K1] Rotate NxN matrix 90° clockwise

Scenario (agritech):

Context:

A agritech UI component rotates square glyphs; engineers want an in-place matrix rotation utility.

Your Task:

Rotate an NxN matrix 90° clockwise, preferably in-place, with coverage for 1x1 and 2x2.

Data & Edge Cases:

Example 3x3 shown in sample.

AI Assistance Expectation:

Use AI to outline layer-by-layer swaps or transpose+reverse approach; add tests.

Constraints & Notes:

Include tests for small N.

Sample Input

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Sample Output

[[7, 4, 1], [8, 5, 2], [9, 6, 3]]

Acceptance Criteria: In-place behavior correct

### K.2 — [S13K2] Compute added/removed lines

Scenario (agritech):

Context:

Change review in agritech needs a function to show added/removed lines between versions.

Your Task:

Given `old` and `new` lists of lines, return (added, removed) preserving the display order.

Data & Edge Cases:

No duplicates in outputs; do not modify input.

AI Assistance Expectation:

Ask AI for an approach using sets but keep stable ordering via list comprehensions.

Constraints & Notes:

Do not show unchanged items.

Sample Input

old=['a','b','c'], new=['b','c','d']

Sample Output

added=['d'], removed=['a']

Acceptance Criteria: Stable ordering; correct diff

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

## Subgroup L

### L.1 — [S13L1] Stub external API for tests

Scenario (agritech):

Context:

A currency conversion in agritech must be testable without network.

Your Task:

Inject a rate-fetch function into `convert(amount, ccy)` and stub it in tests.

Data & Edge Cases:

When fetch\_rate('USD')=83.0, convert(10,'USD')=830.0.

AI Assistance Expectation:

Use AI to suggest dependency injection or monkeypatch patterns.

Constraints & Notes:

Keep convert() pure w.r.t external IO.

Sample Input

def convert(amount, ccy): return amount \* fetch\_rate(ccy)

Sample Output

convert(10,'USD') with rate 83.0 => 830.0

Acceptance Criteria: No network; reproducible tests

### L.2 — [S13L2] Flatten nested JSON with dot keys

Scenario (agritech):

Context:

Configs in agritech arrive as nested JSON; downstream needs flattened keys.

Your Task:

Flatten nested dict to dot-separated keys; for lists, use [index] notation.

Data & Edge Cases:

Example provided.

AI Assistance Expectation:

Ask AI to propose a recursive function and tests with dict+list combos.

Constraints & Notes:

Return a new flat dict.

Sample Input

{'user': {'id': 1, 'name': 'Ana'}, 'meta': {'lang': 'en'}}

Sample Output

{'user.id':1,'user.name':'Ana','meta.lang':'en'}

Acceptance Criteria: Handles nested dicts and lists

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

## Subgroup M

### M.1 — [S13M1] Stable sort employees by dept asc, salary desc

Scenario (agritech):

Context:

HR exports in agritech require deterministic sorting for payroll audits.

Your Task:

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

Data & Edge Cases:

name,dept,salary rows provided.

AI Assistance Expectation:

Use AI to outline csv.DictReader/Writer usage and key composition.

Constraints & Notes:

Stable sort within department by salary desc.

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 and correct ordering

### M.2 — [S13M2] Validate Palindrome Function with Tests

**Scenario (AI-assisted debugging):**  
In an agritech text-analysis module, engineers need a function to check whether a given string is a palindrome (ignoring case and non-alphanumeric characters). A previous intern wrote a buggy version of the function.

**Your Task:**

1. Use AI-assisted coding tools (e.g., VSCode Copilot, Gemini, CursorAI) to:
   * Debug and correct the provided function.
   * Write **at least 5 unit tests** (positive, negative, edge cases).
2. Ensure the function passes all your test cases.

**Buggy Starter Code (given to students):**

def is\_palindrome(text: str) -> bool:

cleaned = ''.join(ch for ch in text if ch.isalpha())

return cleaned == cleaned[::-1]

**Data & Edge Cases:**

* Input may contain spaces, punctuation, and digits.
* Must be case-insensitive.
* Empty string should return True.

**Deliverables:**

* Corrected Python function.
* A test suite (you may use pytest or Python’s built-in unittest).
* Explanation of what bug(s) you fixed.

**Sample Input / Output:**

Input: "A man, a plan, a canal: Panama"

Output: True

Input: "hello"

Output: False

**Acceptance Criteria:**

* Handles mixed case + non-alphanumeric characters.
* Passes at least 5 meaningful test cases (normal, edge, tricky cases).
* Code must be readable and AI-generated documentation (docstring/comments) included.

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

## Subgroup N

### N.1 — [S13N1] Validate Sorting Algorithm with AI-Generated Test Cases

**Scenario (AgriTech Analytics):**  
An agritech platform has a custom function that sorts lists of integers. Engineers want to ensure the function works correctly across edge cases, but manually writing all test cases is tedious. The task is to **use AI-assisted tools to generate a robust test suite**.

**Your Task:**

1. Review the provided sorting function for correctness:

def custom\_sort(arr):

# naive implementation

for i in range(len(arr)):

for j in range(i+1, len(arr)):

if arr[i] > arr[j]:

arr[i], arr[j] = arr[j], arr[i]

return arr

1. Use AI-assisted tools (VSCode Copilot, Gemini, CursorAI) to:
   * Suggest **edge cases**, **randomized inputs**, and **special inputs**.
   * Generate **at least 7 automated test cases** covering:
     + Empty list
     + Single element
     + Already sorted list
     + Reverse sorted list
     + List with duplicates
     + Large numbers
     + Negative numbers
2. Correct the function if AI highlights bugs or inefficiencies.

**Constraints & Notes:**

* Test suite should assert correct order.
* AI may assist in generating code for test cases or using pytest/unittest.
* Document reasoning for each generated test case.

**Sample Input / Output Examples:**

| **Input** | **Expected Output** |
| --- | --- |
| [] | [] |
| [1] | [1] |
| [2,1,3] | [1,2,3] |
| [5,5,2,2] | [2,2,5,5] |
| [10,-1,0] | [-1,0,10] |

**Acceptance Criteria:**

* Function sorts correctly for all AI-generated test cases.
* Test cases are **meaningful and cover edge conditions**.
* Code and test suite are **clean, documented, and reproducible**.

### N.2 — [S13N2] Compute simple sentiment score

Scenario (agritech):

Context:

A lightweight agritech feedback analyzer uses a small lexicon to score sentiment.

Your Task:

Compute total sentiment using: good=+1, great=+2, bad=-1.

Data & Edge Cases:

Tokenize by spaces; strip simple punctuation.

AI Assistance Expectation:

Use AI to draft tests and implement a simple tokenizer.

Constraints & Notes:

Sum over tokens and return integer.

Sample Input

good product with bad packaging but great value

Sample Output

2

Acceptance Criteria: Correct total; punctuation stripped

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

## Subgroup O

### O.1 — [S13O1] Point-in-polygon (ray casting)

Scenario (agritech):

Context:

Geofencing in agritech requires checking if points lie within polygonal regions.

Your Task:

Implement ray-casting point-in-polygon; treat points on edges as inside.

Data & Edge Cases:

Square example provided; return boolean list for queries.

AI Assistance Expectation:

Ask AI to outline crossing-count logic and edge handling.

Constraints & Notes:

Return list[bool] aligning to input order.

Sample Input

poly=[(0,0),(4,0),(4,4),(0,4)], pts=[(2,2),(5,5)]

Sample Output

[True, False]

Acceptance Criteria: Edges counted as inside

**Prompt:** So the senior is agritech where I need to implement ray-casting point-in-polygon; treat points on edges as inside in python code with Square example provided; return boolean list for queries and return list[bool] aligning to input order Sample Input should be

poly=[(0,0),(4,0),(4,4),(0,4)], pts=[(2,2),(5,5)]

Sample Output and should be

[True, False]

Acceptance Criteria: Edges counted as inside

### O.2 — [S13O2] Compute rolling median (w=3)

Scenario (agritech):

Context:

A agritech monitoring job computes rolling medians (w=3) for anomaly detection.

Your Task:

Return the median for each sliding window; prefer an efficient approach.

Data & Edge Cases:

Example provided; for large n, two-heaps or bisect approach acceptable.

AI Assistance Expectation:

Use AI to propose options and complexity; implement simple but efficient enough method.

Constraints & Notes:

Include tests for short lists and exact window size.

Sample Input

[1, 3, 2, 5, 4]

Sample Output

[2, 3, 4]

Acceptance Criteria: Efficient and correct

**Prompt:**  So the Scenario is agritech where I need to create A agritech monitoring job computes rolling medians (w=3) for anomaly detection code where we have to add return the median for each sliding window; prefer an efficient approach and Data & Edge Cases where I need provide Example; for large n, two-heaps or bisect approach acceptable so here we Include tests for short lists and exact window size with

Sample Input

[1, 3, 2, 5, 4]

And, Sample Output

[2, 3, 4]

Acceptance Criteria: Efficient and correct