Question Paper Set A1

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: Code Quality & Performance

  H–I: Documentation & Code Review
  J–K: Code Quality & Performance
  L–M: Debugging with AI
  N–O: AI completion (classes, loops, conditionals)

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 A1

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: AI completion (classes, loops, conditionals)

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 — [S01A1] Compute per-device average from logs (AI completion)

Scenario (e-commerce):

Context:

You are integrating a e-commerce telemetry service where each device emits periodic measures as CSV lines: `id,timestamp,tempC`. 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-device averages of `tempC`. 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

```
de11,2025-01-01T08:00,20.7
de12,2025-01-02T09:00,22.2
de13,2025-01-03T010:00,23.7
```

Sample Output

```
{'de11': 20.7, 'de12': 22.2, 'de13': 23.7} and overall_avg=22.2
```

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

A.2 — [S01A2] Implement OrderBook with add/remove/summary (AI completion)

Scenario (e-commerce):

Context:

A microservice in the e-commerce 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 `OrderBook` 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': 10}, {'op': 'add', 'id': 'b2', 'value': 17}, {'op': 'remove', 'id': 'a1'}, {'op': 'add', 'id': 'c3', 'value': 7}]
```

Sample Output

```
count=2, avg=12.0
```

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

---

Subgroup B

B.1 — [S01B1] Apply surge/penalty rules (conditionals)

Scenario (e-commerce):

Context:

Pricing in the e-commerce 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

```
[30.0, 60.0]
```

Acceptance Criteria: Correct surge threshold and rounding

B.2 — [S01B2] Debug rolling mean (off-by-one)

Scenario (e-commerce):

Context:

A team in e-commerce 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=[1, 2, 3, 4] 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=[1, 2, 3, 4], 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

```
[1.5, 2.5, 3.5]
```

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

---

Subgroup C

C.1 — [S01C1] Debug de-duplication (case-insensitive)

Scenario (e-commerce):

Context:

Customer contact lists in the e-commerce 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 — [S01C2] TDD: slugify titles

Scenario (e-commerce):

Context:

Content titles in the e-commerce 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', 'Set1-C2']
```

Sample Output

```
['hello-world', 'ai-you', 'set1-C2']
```

Acceptance Criteria: All tests pass; edge cases covered

Subgroup D

D.1 — [S01D1] TDD: increment version suffix

Scenario (e-commerce):

Context:

File versioning in the e-commerce 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 — [S01D2] Generate docstrings and usage examples

Scenario (e-commerce):

Context:

Data analysts in e-commerce 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

---

TASK_1:

Prompt-1:

def normalize(scores):

m = max(scores); n = min(scores)

return [(x-n)/(m-n) for x in scores] . Add docstring for this code. it should take list as input. if input is empty ,return empty list. if minimum and maximum is same in ist return a ist of zeroes.I want you to rewrite the code according to the instructions I gave.

Code:

Task-01.py > ...

```python
import unittest
import re

def bump_version(name):
    # Handle known double extensions (.tar.gz, .tar.bz2, .tar.xz, etc.)
    double_exts = [".tar.gz", ".tar.bz2", ".tar.xz"]
    ext = ""
    base = name

    for dext in double_exts:
        if name.endswith(dext):
            base = name[: -len(dext)]
            ext = dext
            break
    else:
        # Otherwise, split at last dot
        if "." in name:
            base, single_ext = name.rsplit(".", 1)
            ext = "." + single_ext
        else:
            base, ext = name, ""

    # Look for version suffix in base
    match = re.search(r"(.*)_v(\d+)$", base)
    if match:
        prefix, version = match.groups()
        new_version = int(version) + 1
        width = max(len(version), 2)  # preserve padding (min 2)
        return f"{prefix}_v{new_version:0{width}d}{ext}"
    else:
        return f"{base}_v01{ext}"
```

```
# ---------- Unit Tests ----------
class TestBumpVersion(unittest.TestCase):
    def test_add_version(self):
        self.assertEqual(bump_version("summary.csv"), "summary_v01.csv")
        self.assertEqual(bump_version("data.txt"), "data_v01.txt")
        self.assertEqual(bump_version("file"), "file_v01")

    def test_increment_version(self):
        self.assertEqual(
        self.assertEqual(bump_version("log_v09.txt"), "log_v10.txt")
        self.assertEqual(bump_version("notes_v99.md"), "notes_v100.md")

    def test_edge_cases(self):
        self.assertEqual(bump_version("archive_v01.tar.gz"), "archive_v02.tar.gz")
        self.assertEqual(bump_version("archive_v9.tar.bz2"), "archive_v10.tar.bz2")
        self.assertEqual(bump_version("my.file_v2.csv"), "my.file_v03.csv")
        self.assertEqual(bump_version("noext_v1"), "noext_v02")
        self.assertEqual(bump_version("justfile"), "justfile_v01")
        self.assertEqual(bump_version("complex.name_v09.backup"), "complex.name_v10.backup")
        self.assertEqual(bump_version("multi.part.name.csv"), "multi.part.name_v01.csv")
        self.assertEqual(bump_version("multi.part.name_v01.csv"), "multi.part.name_v02.csv")
        self.assertEqual(bump_version("bigfile_v0009.csv"), "bigfile_v0010.csv")
```

(function) def bump_version(name: Any) -> str

```
        self.assertEqual(bump_version("bigfile_v0009.csv"), "bigfile_v001


# ---------- Demo Run ----------
if __name__ == "__main__":
    files = [
        "my.file_v2.csv",
        "archive_v01.tar.gz",
        "archive_v9.tar.bz2",
        "report_v1.csv",
        "summary.csv",
        "file"
    ]
    print("Sample Input → Output")
    for f in files:
        print(f" {f:30} → {bump_version(f)}")

    print("\nRunning unit tests...\n")
    unittest.main(exit=False)
```

Output:

```
OK
PS C:\Users\Namitha\OneDrive\Desktop\Lab test-2> & C:/Users/Namitha/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/Namitha/OneD
ve/Desktop/Lab test-2/Task-01.py"
Sample Input → Output
 my.file_v2.csv               → my.file_v03.csv
 archive_v01.tar.gz           → archive_v02.tar.gz
 archive_v9.tar.bz2           → archive_v10.tar.bz2
 report_v1.csv                → report_v02.csv
 summary.csv                  → summary_v01.csv
 file                         → file_v01

Running unit tests...

...
----------------------------------------------------------------------
Ran 3 tests in 0.000s
```

TASK_2:

Prompt-2: Write TDD for a function bump_version(name) that increments or adds a _vNN version suffix before the file extension. Handle cases with/without existing suffix, preserve extension, and use zero-padded 2-digit numbers. Example: report_v1.csv → report_v02.csv, summary.csv → summary_v01.csv, log_v09.txt → log_v10.txt. Include regex-based solution and edge-case tests."

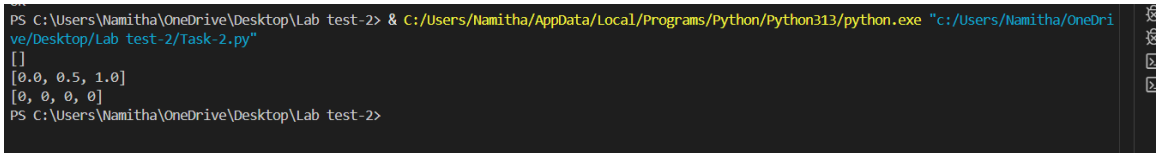
Code:

```
sk-2.py ×  Task-01.py                                   Launch
Task-2.py > normalize
def normalize(scores):
    """
    Normalizes a list of numerical scores to the range [0, 1].

    Args:
        scores (list): List of numerical values to normalize.

    Returns:
        list: List of normalized values. Returns an empty list if input is empty.
              If all values are the same, returns a list of zeroes.
    """
    if not scores:
        return []
    m = max(scores)
    n = min(scores)
    if m == n:
        return [0 for _ in scores]
    return [(x - n) / (m - n) for x in scores]
print(normalize([ ])) # Example usage
print(normalize([10,20,30])) # Example usage
print(normalize([1,1,1,1 ])) # Example usage
```

Output:



E.1 — [S01E1] Inventory Stock Mismatch

Scenario (Error Debugging):
Context: An e-commerce platform tracks stock counts, but a bug prevents correct subtraction after sales.

Buggy Code:

```
def update_stock(stock, sold):
    for item in sold:
        stock[item] = stock[item] + sold[item]    # BUG: should subtract
    return stock

stock = {"Shoes": 10, "Socks": 20}
sold = {"Shoes": 2, "Socks": 5}
print(update_stock(stock, sold))
```

Expected Output:

```
{'Shoes': 8, 'Socks': 15}
```

Actual Output:

```
{'Shoes': 12, 'Socks': 25}
```

AI Assistance Expectation:

- Use AI to identify the logical bug.
- Correct subtraction logic and validate with test data.

Deliverables:

- Corrected function.
- Explanation of the bug fix.
- Test case results.

E.2 — [S01E2] Division by Zero in Averages

Scenario (Error Debugging):
Context: A report generator calculates averages but sometimes crashes when given an empty list.

Buggy Code:

```
def avg(scores):
    return sum(scores) / len(scores)

print(avg([90, 80, 100]))
print(avg([]))   # CRASH
```

Expected Output:

```
90.0
No scores available
```

Actual Output:

```
ZeroDivisionError: division by zero
```

AI Assistance Expectation:

- Use AI to suggest conditional error handling.
- Add try/except or explicit check for `len(scores) == 0`.

Deliverables:

- Fixed `avg()` function.
- Test cases with empty and non-empty lists.
- Explanation of fix.

Subgroup F — AI-Assisted Code Review

F.1 — [S01F1] Inefficient Prime Checker

Scenario (Code Review):
Context: A student wrote code to check if a number is prime, but it's inefficient.

Buggy Code:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Task:

- Use AI-assisted code review to suggest improvements.
- Optimize loop range, variable naming, and add docstring.

Sample Input/Output:

```
Input: 29 → Output: True
Input: 12 → Output: False
```

Acceptance Criteria:

- Efficient up to √n.
- Clear variable names.
- Includes docstring.

F.2 — [S01F2] Readability in Student Grade Script

Scenario:
The following script calculates averages but is unreadable and unmaintainable.

Buggy Code:

```
def f(l):
    s = 0
    for i in l:
        s += i
    return s/len(l)
```

Task:

- Use AI to review code and refactor for readability.
- Add type hints, descriptive names, error handling for empty list.

Expected Output:

```
Input: [80, 90, 100] → Output: 90.0
```

Acceptance Criteria:

- Self-explanatory function name.
- Handles empty input gracefully.

Subgroup G

G.1 — [S01G1] Sum CSV column ignoring bad rows

Scenario (e-commerce):

Context:

Ad-hoc CSV exports in e-commerce 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 — [S01G2] Merge two CSVs by id

Scenario (e-commerce):

Context:

Two CSVs in e-commerce 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 — [S01H1] Extract hashtags and mentions

Scenario (e-commerce):

Context:

Moderation in the e-commerce 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 — [S01H2] Shortest path on weighted graph (Dijkstra)

Scenario (e-commerce):

Context:

Routing decisions in the e-commerce 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 — [S01I1] Generate Documentation for Inventory Management

Scenario (e-commerce):
A junior developer wrote a Python function that updates product stock after an order is placed. However, the function lacks documentation, making it hard to maintain.

```
def update_stock(product_id, qty, stock_dict):
    if product_id in stock_dict:
        stock_dict[product_id] -= qty
        if stock_dict[product_id] < 0:
            stock_dict[product_id] = 0
    return stock_dict
```

Your Task:

- Use AI to generate **comprehensive docstrings and inline comments**.
- Ensure the docstring specifies parameters, return type, and edge cases (like negative stock).

Data & Edge Cases:

- Stock must never go negative.
- If `product_id` not in stock → no update.

AI Assistance Expectation:

- Use AI to propose a PEP 257-style docstring.
- AI should add inline comments explaining **why** conditions exist, not just **what** they do.

Sample Input

```
update_stock("P01", 3, {"P01": 5})
```

Sample Output

```
{"P01": 2}
```

Acceptance Criteria:

- Function retains same logic.
- Generated documentation is clear, professional, and AI-assisted.
- Inline comments add readability without redundancy.

I.2 — [S01I2] AI-Assisted Code Review for Payment Validation

Scenario (fintech/e-commerce):
A developer wrote the following function to validate payment details. It works but may not meet quality or security standards.

```
def validate_payment(card, exp, cvv):
    if len(card) == 16 and len(str(cvv)) == 3:
        return True
    else:
        return False
```

Your Task:

- Perform an **AI-assisted code review**.
- Suggest improvements regarding:
  - Readability
  - Code quality
  - Security best practices (don't log sensitive data, avoid weak checks).
- Rewrite the function with improvements, using AI to scaffold.

Data & Edge Cases:

- Expiry date validation missing.
- `card` type not enforced (should be digits only).
- Should gracefully handle invalid inputs (e.g., `None`, wrong length).

AI Assistance Expectation:

- Ask AI for review suggestions.
- Apply fixes (e.g., regex for card digits, structured return values).

Sample Input

```
validate_payment("1234567890123456", "12/25", 123)
```

Sample Output

```
True
```

Acceptance Criteria:

- Reviewed code has improved readability & error handling.
- AI-proposed improvements are documented (before → after).
- No sensitive data exposed in logs or messages.

Subgroup J

J.1 — [S01J1] Document & Improve Order Discount Function

Scenario (e-commerce):
A developer implemented a discount function for orders. The code works but is undocumented and not easily understandable by other team members.

```
def apply_discount(price, discount):
    final = price - (price * discount/100)
    if final < 0:
        final = 0
    return final
```

Your Task:

- Use AI to generate a **docstring (PEP 257 style)**.
- Add inline comments for clarity.
- Use AI to suggest improvements in **type safety** and **input validation** (e.g., handling negative discount or non-numeric inputs).

Data & Edge Cases:

- Discount should be between 0 and 100.
- Price should never be negative.

AI Assistance Expectation:

- Ask AI to propose improved type annotations.
- AI should generate professional, human-readable documentation.

Sample Input

```
apply_discount(200, 10)
```

Sample Output

```
180.0
```

Acceptance Criteria:

- Code logic unchanged (except validation improvements).
- AI-generated documentation is clear and maintainable.
- Inline comments explain "why" not just "what".

Deliverables:

- Documented & refactored function in `.py` file.
- AI prompt + generated documentation snippet.

J.2 — [S01J2] Code Review for User Login Check

Scenario (web platform):
The following function checks whether a user is allowed to log in. It was written quickly and lacks quality and security considerations.

```
def can_login(username, password, db):
    if username in db and db[username] == password:
        return True
    else:
        return False
```

Your Task:

- Perform an **AI-assisted code review**.
- Identify risks:
  - Password stored in plain text.
  - No input validation.
  - Logic readability issues.
- Propose improvements with AI:
  - Use `hashlib` (or other library) for password checks.
  - Improve code readability with early return.
  - Document function clearly.

Data & Edge Cases:

- `username` missing from db.
- Empty strings for credentials.
- Invalid types (`None`, numbers).

AI Assistance Expectation:

- Ask AI to propose review comments.
- Use AI to generate a refactored, documented version.

Sample Input

```
db = {"alice": "secret123"}
can_login("alice", "secret123", db)
```

Sample Output

```
True
```

Acceptance Criteria:

- Reviewed code improves readability & security.
- AI-generated documentation added.
- Clear before → after record of review suggestions.

Deliverables:

- Original + reviewed function.
- AI review comments captured (as report).
- Final refactored function with docstring.

Subgroup K

K.1 — [S01K1] Rotate NxN matrix 90° clockwise

Scenario (e-commerce):

Context:

A e-commerce 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 — [S01K2] Merge Two Sorted Lists

Scenario (e-commerce):
Context:
In the backend of an e-commerce order management system, two sorted lists of order IDs (integers) are generated by different microservices. To prepare a consolidated report, engineers need to merge them into a single sorted list.

Your Task:
Write a function to merge two sorted lists into one sorted list. Aim for **O(n + m)** time complexity, where `n` and `m` are the lengths of the lists.

Data & Edge Cases:

- Lists may be of unequal length.
- Either list can be empty.
- Must handle duplicates correctly.

AI Assistance Expectation:
Use AI to explain the **two-pointer technique** or equivalent merging logic. Provide unit tests for normal and edge cases.

Constraints & Notes:

- Avoid built-in sorting of the concatenated list.
- Ensure stability (relative order of duplicates preserved).

Sample Input:

```
List A = [1, 3, 5]
List B = [2, 4, 6]
```

Sample Output:

```
[1, 2, 3, 4, 5, 6]
```

Acceptance Criteria:

- Correct merged list.
- Efficient (linear time).
- Handles empty and duplicate cases.

Subgroup L

Sub Group L — Files / CSV & Regex

L.1 — [S01L1] Extract Emails from Log File

Scenario (E-commerce):
Context:
An e-commerce company stores customer support chat logs in plain text files. The team needs to extract all valid email addresses for follow-up.

Your Task:
Write a function that:

1. Reads a log file line by line.
2. Uses **regex** to extract all valid emails.
3. Returns them as a **unique sorted list**.

Data & Edge Cases:

- Lines may contain multiple emails.
- Some text might look like an email but isn't valid.
- File may be empty.

AI Assistance Expectation:
Use AI to suggest a **regex pattern** for matching emails and provide test cases.

Sample Input (file contents):

```
Support ticket: user1@example.com
Invalid: user@@example.com
Contact: admin@shop.com, helpdesk@shop.com
```

Sample Output:

```
["admin@shop.com", "helpdesk@shop.com", "user1@example.com"]
```

Acceptance Criteria:

- Only valid emails captured.
- Deduplicated and sorted output.

L.2 — [S01L2] Summarize Orders from CSV

Scenario (E-commerce):
Context:
Orders are stored in a CSV file with fields: `order_id, customer, amount`. The finance team wants a quick **summary of total sales per customer**.

Your Task:
Write a program to:

1. Read the CSV file.
2. Use a dictionary to accumulate total amounts per customer.
3. Print results in sorted order by customer name.

Data & Edge Cases:

- Handle empty CSV.
- Skip malformed rows (missing fields).
- Large file efficiency.

AI Assistance Expectation:
Use AI to propose efficient **CSV parsing** and robust error handling. Provide tests for small and large datasets.

Sample Input (CSV):

```
order_id,customer,amount
1,Alice,120
2,Bob,90
3,Alice,60
```

Sample Output:

```
Alice: 180
Bob: 90
```

Acceptance Criteria:

- Aggregation correct.
- Skips malformed lines.
- Works for thousands of rows.

Subgroup M

M.1 — [S01M1] Stable sort employees by dept asc, salary desc

Scenario (e-commerce):

Context:

HR exports in e-commerce 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 — [S01M2] Debug Shopping Cart Total

Scenario (E-commerce):
Context:
The finance report shows wrong cart totals.

Buggy Code (given):

```
def cart_total(cart):
    total = 0
    for price, qty in cart.items():
        total += price * qty
    return total
```

Issue:
Here `cart` is defined as `{"apple": (2, 3), "banana": (1, 5)}` where each tuple = `(price, qty)`.

Your Task:

- Debug the loop to correctly unpack tuples.

Sample Input:

```
cart = {"apple": (2, 3), "banana": (1, 5)}
print(cart_total(cart))
```

Expected Output:

```
11
```

Acceptance Criteria:

- Handles price/quantity tuples correctly.
- Works for empty cart (returns 0).

Subgroup N — Code Privacy

N.1 — [S01N1] Mask sensitive data in logs

Scenario (code privacy):
Context: During debugging in an e-commerce platform, logs may accidentally expose sensitive information like emails or phone numbers.

Your Task:
Scan log lines and replace sensitive fields (emails, phone numbers) with masked versions (`user@***.com`, `***-***-1234`).

Data & Edge Cases:

- Input: log lines as text.
- Replace **all occurrences**, not just first.
- Ignore if no sensitive info.

AI Assistance Expectation:
Use regex with Python's `re.sub()` for masking. AI scaffolds common regex patterns.

Constraints & Notes:
Must not expose raw sensitive values in output logs.

Sample Input:

```
User Raj logged in with email raj.kumar@example.com and phone
9876543210
```

Sample Output:

```
User Raj logged in with email raj.kumar@***.com and phone ***43210
```

Acceptance Criteria:
All sensitive fields masked, log readability preserved.

N.2 — [S01N2] Redact PII from CSV exports

Scenario (code privacy):
Context: HR exports employee data in CSVs, but for sharing externally, PII (name, email, phone) must be redacted.

Your Task:
Read CSV with `csv.DictReader`, replace sensitive columns with `REDACTED`, and re-emit CSV.

Data & Edge Cases:

- Input may contain multiple sensitive columns.
- Non-sensitive fields must remain intact.

AI Assistance Expectation:
Show `DictReader/DictWriter` scaffolding, redaction logic.

Constraints & Notes:
Consistent redaction, headers unchanged.

Sample Input:

```
name,email,dept,salary
Raj,raj@example.com,Eng,120
Maya,maya@example.com,HR,90
```

Sample Output:

```
name,email,dept,salary
REDACTED,REDACTED,Eng,120
REDACTED,REDACTED,HR,90
```

Acceptance Criteria:
All PII redacted; non-PII untouched.

Subgroup O — AI Completion (Classes, Loops, Conditionals)

O.1 — [S01O1] Shopping Cart with Discounts

Scenario (AI completion):
Context: An e-commerce app needs a shopping cart class with support for discount rules.

Your Task:

- Create a `ShoppingCart` class.
- Methods: `add_item(name, price, qty)`, `apply_discount(percent)`, `total()`.
- Ensure discount applies to the grand total only once.

AI Assistance Expectation:

- Use AI completion for class boilerplate, constructor, and method stubs.
- Add conditionals to prevent discount below 0%.

Sample Input:

```
cart = ShoppingCart()
cart.add_item("Shoes", 2000, 1)
cart.add_item("Socks", 200, 3)
cart.apply_discount(10)
print(cart.total())
```

Sample Output:

```
2600
```

Acceptance Criteria:

- Handles multiple items.
- Discount correctly applied.

Deliverables:

- Prompt for AI completion.
- Final class `ShoppingCart`.
- Test script with 3+ items.

O.2 — [S01O2] Generate Multiplication Tables

Scenario (AI completion):
Context: A training platform needs dynamic generation of multiplication tables for learners.

Your Task:

- Write a function `generate_tables(n, upto)` that prints multiplication tables up to `n`, each table going till `upto`.
- Use nested loops.
- Validate inputs (`n > 0`, `upto > 0`).

AI Assistance Expectation:

- Use AI to complete nested loops.
- Insert input validation conditionals.

Sample Input:

```
generate_tables(2, 3)
```

Sample Output:

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
```

Acceptance Criteria:

- Correct multiplication results.
- Neatly formatted output.

Deliverables:

- Prompt for AI completion.
- Python script `tables.py`.
- Example run screenshots.