

Code Review and Debugging with AI

Part A checks five functions against contracts, Part B develops an IPv4 review from independent tests to correction, and Part C records linting, recursive frames and command-line work. Preserve sources and observed evidence.

Due: Homework 3 · Tue 17 Nov, 23:59

Goals

- Check a function's result, order, boundaries and mutation against its contract
- Separate API, behavior, performance and readability questions
- Design independent checks and keep candidates, failures and revisions
- Interpret lint diagnostics and inspect recursive call frames
- Declare assistance honestly

The three parts are available together. Suggested time: about 40 minutes for A, 35 for B and 15 for C. A contributes 60% of this week's homework score and B 40%. C records completion and is relevant to the final exam. Platform totals are raw automatic points; manual notes remain for the TA. A submitted/pending note is not an awarded manual mark.

This lab is not the solo assignment

Lab08 follows [Level 1](#): AI assistance is allowed with an actual source, adopted scope and verification record. Independent work and instructor-candidate use can be declared honestly. [Solo assignment 1](#) remains **Level 0** for both ordinary and adjacent-peg Hanoi, worth 5% of the semester grade. The recursion exercise here uses classroom factorial, not the solo implementation.

Submit this week's work through [HW3](#), which covers weeks 7–9. Its deadline comes from the shared calendar and is displayed in GMT+8. The header's earliest-week Level 0 badge does

not replace per-week instructions: week seven remains independent work and week eight follows this page.

The week-eight package and cards are prepared locally, but not synchronized to the server. Use HW3's week-eight entries once published. Until then retain the local files, do not misuse a week-seven entry and do not describe saving locally as an online submission.

Working method

Use the Python 3.13 installation, VS Code and terminal already prepared; Windows students use their existing WSL environment. Ask a TA about setup issues instead of reinstalling the operating system.

Candidates below are constructed teaching examples, not actual model transcripts.

Determine the required behavior before examining the code. There need not be exactly one defect. Correct-but-expensive or hard-to-read code can still deserve improvement without inventing a functional failure.

The five A functions can be saved separately or in one file, uploaded to each relevant card. Complete the AI declaration attached to each code submission; use the separate text cards for the review evidence. Do not expose credentials, private files or another student's code in records or screenshots.

Part A — Contracts, corrections and preserved behavior

Five code problems are worth 4 automatic points each, totaling 20. Four review-note questions carry 1 manual point each. Part A's raw maximum is **24**.

8-1 The second-largest distinct value

Implement `second_largest(nums: list[int]) -> int`. The list has at most 1000 integers and **at least two distinct values**. Return the second-largest distinct integer, without changing the input. Fewer distinct values are outside this contract; do not invent a None or exception convention.

Candidate:

```
def second_largest(nums: list[int]) -> int:
    ordered = sorted(nums, reverse=True)
    return ordered[1]
```

```
second_largest([3, 1, 2])    -> 2
second_largest([5, 5, 3])    -> 3
second_largest([-8, -1, -3]) -> -3
```

A sorted position is not a distinct-value rank. Recheck repeated maxima, negatives and unchanged input after the correction. The result is an int, not a boolean replacement.

Code: `w08-second-largest`. Evidence: question 1 on the Part A card.

8-2 Title-case API review

Implement `title_case(s: str) -> str`, matching Python 3.13's `str.title()` exactly. Keep whitespace and punctuation and return a new string. Input length is at most 200. Natural-language conventions for articles or acronyms are not added requirements.

Candidate:

```
def title_case(s: str) -> str:
    return s.capitalize_words()
```

The individual `title()` method was not demonstrated in class; use L08's documentation-checking method and the [string method reference](#). Compare `capitalize()` with `title()` and confirm that a plausible name is not evidence an API exists.

```
title_case("hello python")    -> "Hello Python"
title_case("")                -> ""
title_case(" python AND c ") -> " Python And C "
title_case("they're here")    -> "They'Re Here"
```

The apostrophe result describes the Python API, not advice for writing English titles.

Code: `w08-title-case` . Documentation and observed cases: question 2 on the A card.

8-3 Duplicate detection and work

Implement `has_duplicate(items: list[int]) -> bool`, detecting repeated integer values without modifying the list. Length is at most 1000; empty and one-element lists return False. Return an actual bool.

Candidate:

```
def has_duplicate(items: list[int]) -> bool:
    for i in range(len(items)):
        for j in range(len(items)):
            if i != j and items[i] == items[j]:
                return True
    return False
```

```
has_duplicate([])          -> False
has_duplicate([7])         -> False
has_duplicate([3, 1, 2])   -> False
has_duplicate([3, 1, 3])   -> True
```

The original behavior can be correct. Reduce repeated work using familiar tools such as L06's set, preserving results and mutation behavior.

For n distinct integers, count evaluations of `i != j` separately from integer-equality comparisons. These are source expressions, not CPU instructions. Large timing runs are unnecessary, and nesting alone is not a defect. Automatic tests check behavior, not a proof of complexity; the TA reviews the explanation.

Code: `w08-has-duplicate` . Analysis and actual checks: question 3 on the A card.

8-4 Character frequencies and readability

Implement `count_chars(s: str) -> dict[str, int]`, counting Python characters, including case, whitespace and punctuation. Input length is at most 500. Return a new

empty dictionary for empty input. Results must be independent across calls and must not change when another returned dictionary is modified.

Candidate:

```
def count_chars(s):
    d = {}
    for i in range(len(s)):
        c = s[i]
        if c in d:
            d[c] = d[c] + 1
        else:
            d[c] = 1
    return d
```

```
count_chars("aba") -> {"a": 2, "b": 1}
count_chars("AaA") -> {"A": 2, "a": 1}
count_chars("")    -> {}
```

Dictionary key order is not graded. Frequencies are positive int values. Count characters, not UTF-8 bytes; do not automatically combine multiple code points into a visual glyph.

The original behavior need not be wrong. Improve names/traversal and check equivalence; missing type hints or length alone is not a PEP 8 or logic defect.

Code: `w08-count-chars`. Readability and behavior evidence: question 4 on the A card.

8-5 Largest adjacent gap

Implement `max_gap(nums: list[int]) -> int`. Sort by value, return the largest adjacent difference and keep the input unchanged. Length is at most 1000. Fewer than two elements return integer 0, not False.

Candidate:

```
def max_gap(nums: list[int]) -> int:
    ordered = sorted(nums)
    gaps = [ordered[i + 1] - ordered[i]
            for i in range(len(ordered) - 1)]
    return max(gaps)
```

```
max_gap([3, 9, 1, 15]) -> 6
max_gap([]) -> 0
max_gap([7]) -> 0
max_gap([2, 2, 2]) -> 0
max_gap([-3, -1, -10]) -> 7
```

Fix short-input behavior from the contract, then recheck ordinary, repeated and negative values plus mutation. A list comprehension or ordinary loop is enough; generator expressions are not needed.

Code: `w08-max-gap`. Record this function's source, change and tests in its code-submission declaration.

Part A review notes

Use `w08-a-review-notes` for four questions, one manual point each:

1. Distinct ranking: original counterexample, expected/observed behavior, correction and mutation check.
2. Title API: documentation, method distinctions and a real punctuation/whitespace case.
3. Duplicate detection: original behavior, the two counts, revised state and verification.
4. Frequencies: behavior already correct, naming/traversal changes and empty/case/repeated-call checks.

Support each answer with code, inputs or observations. Do not classify every candidate as "cannot run", or substitute length for review substance.

Part B — One IPv4 review, four steps

The corrected program is worth 8 automatic points and four evidence questions 1 manual point each. Part B's raw maximum is **12**.

B-1 Contract and independent cases

Implement `is_valid_ipv4(s: str) -> bool` for string inputs, using this strict dotted-decimal contract:

- exactly four parts separated by `.`;
- ASCII digits `0–9` only, each value 0–255;
- a single `0` is allowed, other leading-zero forms are not;
- reject empty parts, signs and surrounding/internal whitespace;
- return a bool for every string, not an exception for an invalid form.

Design at least eight cases first, with three valid and five invalid strings. Save expectations and reasons before inspection. Consider lexical forms as well as numeric ranges.

Use the taught string operations, loops and conditions rather than a ready-made validator such as `ipaddress`. This is not a regex exercise; regular expressions come in L12. Automatic cases check outward behavior, while the method is reviewed manually.

B-2 Candidate and evidence

If an online AI is available, preserve the actual tool/model, request and original code. Say if the model is unknown. If the tool is unavailable or you prefer not to use it, use the instructor candidate below and disclose no online AI use; a paid account is not required.

Run your cases and record returns or exceptions, relating them to the contract. Independent cases reduce implementation-driven expectations; code inspection can suggest more cases. Neither ordering guarantees every omission will be found.

▼ Instructor candidate: finish B-1's independent cases before inspection

This is a constructed teaching candidate with review issues, not a complete correct solution or an actual model transcript:

```
def is_valid_ipv4(s: str) -> bool:
    parts = s.split(".")
    if len(parts) != 4:
        return False
    for part in parts:
        if not part.isdigit():
            return False
        if not 0 <= int(part) <= 255:
            return False
    return True
```

B-3 Correction and regression

Try a correction yourself before further discussion. If the candidate passes your initial suite, explain the coverage and add checks instead of inventing a failure.

Retain both versions. Recheck ordinary input, 0/255, part count, leading zeros, non-ASCII digits, signs and whitespace. Return a bool; None's falsiness is not a substitute for False.

Upload the complete corrected code to `w08-b-is-valid-ipv4`. Its submission note identifies source, changes and actual verification; the detailed candidate/test record belongs on the B text card.

B-4 Evidence and declaration

Use `w08-b-ipv4-notes` for the four steps:

1. Contract and at least eight independent cases with expectations.
2. Actual source, original implementation and observed outcomes.
3. Corrections, revised code and regressions.
4. Whole-lab declaration and the limits of the finite suite.

Identify actual tools/models, adopted scope, manual changes, real verification and independent work. Truthful no-AI and instructor-candidate work are equally acceptable. Invent no conversations, models or test results. Do not send credentials, tokens or Solo assignment 1 code to a tool.

Part C — Linting, frames and command-line records

Use `w08-c-tools-notes`; all three questions are worth 0 and completion is registered by the TA. The card accepts **text, not image uploads**. Keep screenshots locally for inspection, listing their filenames and observations. A local filename does not mean the image reached the platform.

C-1 Ruff diagnostics

Use the VS Code extension or the course Python environment:

```
python3 -m pip install ruff
ruff check .
```

Record the version, your own paths/rules, actual counts and categories, and explain three unfamiliar diagnostics and decisions. Supplement with the practice file if needed rather than inventing warnings:

```
import math

def f(l):
    unused = 1
    return missing_name + len(l)
```

```
ruff check --select F401,F821,F841,E741 practice.py
```

The command selects these rules explicitly, not as a claim about defaults. No diagnostics does not prove behavior. Understand a fix before applying it. If pip rejects an externally managed environment, use the course interpreter or extension, not administrator forcing; L10 develops virtual environments.

C-2 Recursive frames

Use classroom factorial, not the solo assignment:

```
def factorial(n: int) -> int:
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(3))
```

1. Enable Python and Python Debugger and select Python 3.13. Break on the recursive-call line.
2. Start through Run and Debug or F5, stepping into calls until n=1.
3. Select each active frame and record its local n; the outer factorial(3) retains n=3.
4. Save a stack/variables screenshot locally, list the filename and relate it to L07's indented trace.
5. Try Over, Into, Out and Continue, recording each effect.

Use Fn or toolbar actions where bindings differ. If GUI tools are temporarily unavailable, keep a real print trace and disclose incomplete screenshot work. Do not fabricate a screenshot; the platform does not automatically verify that it exists.

C-3 Command-line recap

Retain the six actual command/output records: home `cs1602/lab08` directory, three files `a.py/b.py/c.txt`, `ls -l`, `a.py` containing `print("hello")`, a count of prior `.py` regular files, and literal `TODO/print(` searches in your own code.

```
mkdir -p ~/cs1602/lab08
cd ~/cs1602/lab08
touch a.py b.py c.txt
ls -l
# Use the editor to write print("hello") into a.py first.
python3 a.py

# Replace the path with a directory containing only your Labs 1-7 code.
find /path/to/labs1-7 -type f -name '*.py' | wc -l
grep -rn -F -e 'TODO' -e 'print(' /path/to/labs1-7
```

Do not include the newly created Lab08 files in the prior-lab count. Filenames have no newlines. `ls -l | wc -l` also counts non-file lines such as total, so it is not an exact file count. Search only your own source.

Command-line reference

Earlier material remains available for lookup; reinstalling Linux is not this lab's task.

```
pwd
ls
ls -l
ls -a
ls -lh
cd /path/to/dir
cd ..
cd ~
cd -
mkdir mydir
mkdir -p a/b/c
touch file.txt
cp a.txt b.txt
cp -r dir1 dir2
mv a.txt b.txt
rm -i scratch.txt
rm -ri scratch-dir
```

Completion depends on the shell. touch creates a missing empty file or updates an existing timestamp without emptying contents. Check copy/move targets before overwriting. rm bypasses the recycle bin; use only your practice files, not system-directory deletion commands. Windows setup may use `wsl --install` under supported conditions with administrator privileges and a possible reboot, not guaranteed instant availability.

```
cat file.txt
head file.txt
head -n 20 file.txt
tail file.txt
tail -f log.txt
less file.txt
wc -l file.txt
grep 'error' log.txt
grep -i 'error' log.txt
grep -n 'error' log.txt
grep -r 'TODO' .
find . -name '*.py'
find . -name '*.py' -size +10k
grep '2026' data.txt | wc -l
history | grep git
```

Pipes connect standard output to the next command. In less, q exits and / searches. Ctrl+C stops tail -f.

```
python3 hello.py
python3 -c 'print(2**10)'
python3 -m pip --version
which python3
man ls
```

`python3 -m pip` uses that interpreter. L11 introduces the testing framework, which can be launched with `python3 -m pytest` once installed. Later reference commands are retained below; processes are developed in week nine and permissions in week eleven:

```
ps aux
ps aux | grep python
top
df -h
echo $PATH
chmod +x script.sh
```

The grep pipeline can include grep itself, so it is not an exact process count. Ctrl+C interrupts foreground work.

Marks and submission checklist

Raw points for this week: A automatic 20 + manual 4 = 24; B automatic 8 + manual 4 = 12. C records completion. The TA combines them as:

Weekly homework score = 60 * A_earned/24 + 40 * B_earned/12

The denominators include manual work. Automatic marks alone are not a complete weekly score before the notes are reviewed. More raw judge points do not increase the semester weighting.

After publication, use HW3’s week-eight entries:

Work	Problem ID
Five A functions	w08-second-largest , w08-title-case , w08-has-duplicate , w08-count-chars , w08-max-gap
Four A review records	w08-a-review-notes
Corrected IPv4 code	w08-b-is-valid-ipv4
Four B records and declaration	w08-b-ipv4-notes
Three tool records	w08-c-tools-notes

- ☐ Five A contracts checked, including mutation; each code note declares actual assistance
- ☐ Four A answers contain counterexamples, documentation or analysis and regressions
- ☐ B retains independent cases, original/revised code and outcomes in their separate entries

- ☐ C records actual diagnostics, frames and six command steps, disclosing incomplete GUI work
- ☐ Local files saved; no misuse of an old entry before publication; every new card checked afterward
- ☐ Pending manual work is not treated as an awarded mark
- ☐ Solo assignment 1 remains Level 0, using its published deadline