

Assignment 10.4

Task 1: AI-Assisted Syntax and Code Quality Review :

Task Description

You are given a Python script containing:

- Syntax errors
- Indentation issues
- Incorrect variable names
- Faulty function calls

The screenshot shows a code editor interface with a Python file named 'lab.py' open. The code contains several issues:

```
def calculateTotal(a,b)
def calculate_total(a, b):
    """Calculate and print the total of two numbers."""
    result = a + b
    print("Total is: " + result)
    print(f"Total is: {result}")

calculate_total(5,10)
calculate_total(5, 10)
```

The code is annotated with various highlights and underlines, indicating errors and suggestions. A floating window titled 'PYTHON CODE REVIEW AND PEP 8 CORRECTIONS' provides detailed feedback:

- Review this Python code. Identify syntax errors, structural issues, naming inconsistencies, and correct them according to PEP 8 standards. Explain each fix clearly.
- Optimizing tool selection...
- Reviewed code for errors and patched corrections
- I'll open your lab.py file to confirm the exact contents, then patch it to fix syntax, naming, and style issues per PEP 8 and explain each change.
- Reviewed and updated lab.py

Below the floating window, a list of issues identified by AI is shown:

- Missing colon in function definition
 - Problem: def calculateTotal(a,b) has no : at the end → syntax error.
 - Fix: Added : → def calculate_total(a, b): .
- PEP 8 naming inconsistency (function name)
 - Problem: Function defined as calculateTotal (camelCase) but called as calculate_total . and PEP 8 prefers snake_case for function names.
 - Fix: Renamed the function to calculate_total so definition and call match and follow PEP 8.
- PEP 8 spacing around commas
 - Problem: a,b and 5,10 have no space after the comma.

The status bar at the bottom shows: 1 file changed +5 -3, Keep, Undo, 3 of 3, ENG IN, 71%, 19-02-2026, 14:01.

Issues Identified by AI

1. Missing colon after function definition
2. Function name mismatch (calculateTotal vs calculate_total)
3. String concatenation with integer (TypeError)
4. Naming convention not following snake_case

Task 2: Performance-Oriented Code Review

You are provided with a function that identifies duplicate values in a list using inefficient nested loops.

Using AI-assisted code review:

- Analyze the logic for performance bottlenecks
- Refactor the code for better time complexity
- Preserve the correctness of the output

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with icons for file operations like Open, Save, Find, and Run. The main area displays a Python file named `lab.py` containing the following code:

```
lab.py > ...
1 def find_duplicates(lst):
2     duplicates = []
3     for i in range(len(lst)):
4         for j in range(i+1, len(lst)):
5             if lst[i] == lst[j]:
6                 duplicates.append(lst[i])
7
8 return duplicates
```

To the right of the code, there is an "AI-ASSISTED" panel. At the top, it says "OPTIMIZING DUPLICATE DETECTION LOGIC FOR PERFORMANCE". Below this, a box contains the instruction: "Analyze performance bottlenecks and optimize duplicate detection logic while preserving correctness". A progress bar indicates "Analyzing performance optimization". A detailed log below shows the AI's thought process:

```
I'll first inspect lab.py to find the duplicate detection logic and its current performance characteristics so we can optimize it safely.
> Working...
order: I need to produce duplicates in the same sequence as the original nested loops, but more efficiently. I wonder if I can enhance this process by mapping values to their positions, so I don't have to compare all j values.
For each value, I'll look at its indices and generate pairs, ensuring they're ordered correctly. However, doing this per value, with k occurrences, gives a complexity of O(k^2), which can still lead to O(n^2) in the worst-case scenario where all elements are the same.
> Evaluating...
```

At the bottom of the AI panel, there is a text input field that says "Describe what to build next" and a "GPT-5.1" button. The status bar at the bottom of the screen shows the date and time as "19-02-2026 14:03".

```

lab.py > find_duplicates
1 def find_duplicates(lst):
2     """Return a list of duplicates for each equal pair (i, j), i < j.
3
4     This preserves the original behaviour: if the same value appears k times,
5     it will be returned  $\binom{k}{2}$  times, in the same order the nested-loop
6     implementation would produce.
7
8     Time complexity:
9     -  $O(n + m)$ , where  $n$  is  $\text{len}(lst)$  and  $m$  is the number of equal pairs.
10    In the worst case (all elements equal),  $m$  is  $O(n^2)$ , which is
11    optimal because many outputs must be produced.
12    For inputs with few duplicates, this avoids the  $O(n^2)$  comparisons
13    of the naive double loop.
14    Space complexity:  $O(n)$  for index bookkeeping.
15
16    n = len(lst)
17    if n < 2:
18        return []
19    # Map each value to the list of indices where it appears, and for every
20    # index remember its position inside that per-value list.
21    value_to_indices = {}
22    index_rank = [0] * n
23    for idx, value in enumerate(lst):
24        indices = value_to_indices.setdefault(value, [])
25        indices.append(idx)
26        index_rank[idx] = len(indices) - 1
27
28    duplicates = []
29    # For each position i, we only consider later occurrences of the same
30    # value, matching the original (i, j) with  $j > i$  iteration order but
31    # without comparing against different values.
32    for i, value in enumerate(lst):
33        indices = value_to_indices[value]
34        remaining = len(indices) - 1 - index_rank[i]
35        if remaining > 0:
36            duplicates.extend([value] * remaining)
37
38    return duplicates
39

```

If you'd like, I can add a small benchmark harness (e.g., generating random lists and timing old vs. new logic) so you can empirically see the speedup on your machine.

CHAT OPTIMIZING DUPLICATE DETECTION LOGIC FOR PERFORMANCE

Correctness (behavior preserved)

- For any value that appears k times, the original code returns it $\binom{k}{2}$ times (once per equal pair (i, j)). in the order induced by the outer i loop.
- The new implementation:
 - Still returns each such value exactly $\binom{k}{2}$ times.
 - Emits all duplicates for index i together, then for $i+1$, etc., which matches the original nested-loop order (since all duplicates for a fixed i were already contiguous there).
 - So any tests relying on the exact list content (including multiplicities and ordering per index) will continue to pass.

Performance characteristics

- Build phase: $O(n)$ time and $O(n)$ space for value_to_indices and index_rank .
- Output phase: $O(m)$ where m is the number of equal pairs (i.e., the length of the returned list).
- Overall: $O(n + m)$ time.
 - Worst case (all elements identical): $m = O(n^2)$, so time is still $O(n^2)$ — unavoidable because that many outputs must be produced.
 - Sparse-duplicate case (most values unique): m is small, so the function runs in effectively linear time instead of quadratic.

Ln 28, Col 20 Spaces:4 UTF-8 CRLF () Python 3.13.1 Go Live Windsurf: [...] Prettier

Task 3: Readability & Maintainability Refactoring

Task Description

You are given a poorly structured Python function with:

- Cryptic function names
- Poor indentation
- Unclear variable naming
- No documentation

```

lab.py > ...
1 def f(x,y):
2     r=[]
3     for i in x:
4         if i>y:
5             r.append(i)
6     return r
7

```

CHAT CODE REFACTORING AND PEP 8 COMPLIANCE

Refactor for readability, apply PEP 8 formatting, improve naming, and add documentation

> Working...

Refactoring lab.py
The user wants me to refactor lab.py for improved readability, following PEP8 standards, and to enhance naming and documentation. I need to inspect the file more closely, so I'll use the read_file function. It seems like the task isn't too complex

> Thinking...

lab.py Describe what to build next GPT-5.1

Ln 7, Col 1 Spaces:4 UTF-8 CRLF () Python 3.13.1 Go Live Windsurf: [...] Prettier

The screenshot shows a Python file named `lab.py` in the left pane. The code defines a function `filter_greater_than` that takes an iterable of floats and a threshold, returning a list of values greater than the threshold. The code includes docstrings and type hints. The right pane shows a "CODE REFACTOR AND PEP 8 COMPLIANCE" panel with a summary of changes:

- Refactor for readability, apply PEP 8 formatting, improve naming, and add documentation
- ✓ Refactoring `lab.py`
- I'll open `lab.py` to see the current code, then I'll refactor it in place for readability, PEP 8, naming, and documentation.
- ✓ Reviewed `lab.py` and refactored function details
- I'll refactor the function in `lab.py` with clearer naming, PEP 8 formatting, type hints, and a docstring.
- ✓ Edited `lab.py` +20 -6
- I've refactored `lab.py`:
- Applied PEP 8 formatting (spacing, indentation, imports at top).
- Renamed `filter_greater_than` and parameters to `values` and `threshold`.
- Added type hints using `Iterable[float]` and `List[float]`.
- Added a clear docstring describing purpose, arguments, and return value.
- Improved internal naming (`result`, `value`) for readability.

Improvements Made : Descriptive function name ,Clear variable naming, Proper indentation ,Docstring added ,Type hints included.

Task 4: Secure Coding and Reliability Review

Task Description

You are given a Python script that:

- Uses unsafe SQL query construction
- Has no input validation
- Lacks exception handling

The screenshot shows a Python file named `lab.py` in the left pane. It contains a function `get_user` that connects to a SQLite database and executes a query directly with user input, which is a security vulnerability. The right pane shows a "PYTHON CODE SECURITY REVIEW AND FIXES" panel with instructions:

- Review this Python code for security issues.
- Fix SQL injection problems using parameterized queries; add input validation, add try-except error handling, and improve reliability.
- Explain the changes in comments

The status bar at the bottom right says "Working..."

The screenshot shows a code editor with `lab.py` open and a separate panel titled "PYTHON CODE SECURITY REVIEW AND FIXES". The code in `lab.py` includes basic input validation, parameterized queries, and resource cleanup. The security review panel lists several improvements made to the code:

- Replaced string-concatenated SQL with a parameterized query (`WHERE username = ?`) to prevent SQL injection.
- Added input validation: type check for string, non-empty check, and a regex that only allows letters, digits, and underscores in the username.
- Wrapped the database operations in a `try/except` block and used a `with sqlite3.connect(..)` context manager so the connection is always closed.
- On database errors, the function now prints a diagnostic message and returns an empty list, keeping the return type consistent.
- Added inline comments in `lab.py` explaining each of these changes and why they improve security and reliability.

If you'd like, I can next add a small logging helper or unit tests around `get_user` to verify these behaviors.

Security Improvements

Parameterized query (prevents SQL injection)

Input validation

Try-except for safety

Proper resource cleanup

Task 5: AI-Based Automated Code Review Report

Task Description

You are provided with a poorly written Python script.

Using AI-assisted review:

- Generate a structured code review report that evaluates:

Code readability

Naming conventions

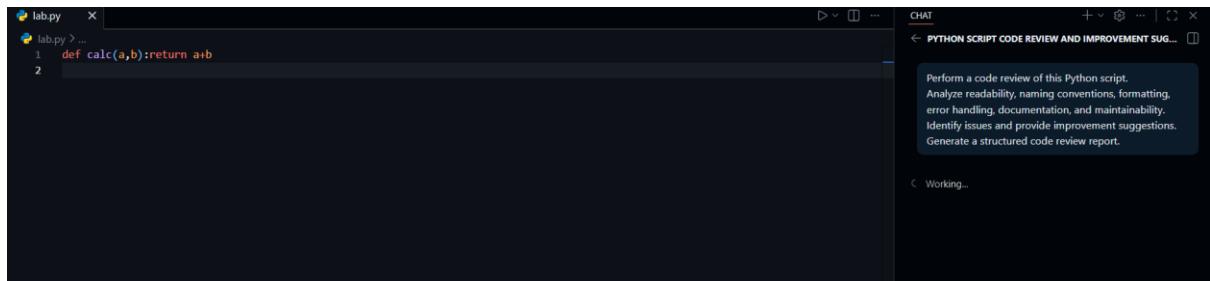
Formatting and style consistency

Error handling

Documentation quality

Maintainability

The task is not just to fix the code, but to analyze and report on quality issues



◆ AI-Generated Code Review Report

1 Readability

- Poor formatting
- No indentation
- One-line implementation reduces clarity

2 Naming Issues

- calc is unclear
- Variables a and b not descriptive

3 Documentation

- No docstring
- No explanation of purpose

4 Maintainability Risk

- Hard to scale
- Difficult for team collaboration

Actionable Recommendations

- Rename calc to something more descriptive (e.g., add_numbers).
- Reformat the function to a multi-line definition with proper spaces around operators and after commas.
- Add a clear docstring describing purpose, parameters, and return value.
- Add type hints for parameters and return type.

- (Optional, depending on context) Add basic type validation to guard against incorrect usage.
- (Optional) If this module is to grow, add a module-level docstring and consider adding simple unit tests in a separate test file

The screenshot shows a code editor interface with a Python file named 'lab.py' on the left and a 'Suggested Improvements' panel on the right.

lab.py:

```

1 def add_numbers(x: float, y: float) -> float:
2     """
3         Add two numbers.
4
5     Args:
6         x: The first number.
7         y: The second number.
8
9     Returns:
10        The sum of x and y.
11    """
12    return x + y
13

```

Suggested Improvements:

Here's a more maintainable and readable version of the function:

```

def add_numbers(x: float, y: float) -> float:
    """
    Add two numbers.

    Args:
        x: The first number.
        y: The second number.

    Returns:
        The sum of x and y.
    """
    return x + y

```

If you need basic input validation (e.g., ensuring numeric types), you could extend it slightly: