SE – LAB – 5

NAME: UMAR AHMED RAYADURG

SRN: PES2UG23CS662

Issue	Type	Line(s)	Description	Fix Approach
Mutable default arg	Bug	12	logs=[] shared across calls	Changed tologs=None, initialized in function
Bare except	Bug/Sec	32	Used except: pass; hides errors	Catch KeyError only, print informative error
Dangerous eval	Security	74	Use of eval, risk of code injection	Removed eval; replaced with a simple print
No input validation	Bug	12	add_item accepted wrong types, caused crash	Added isinstance type checks with error print
Function naming	Style	all	Not using snake_case function names	Renamed all functions to snake_case style
Two blank lines	Style	many	Not enough blank lines between functions	Added two blank lines per PEP8

Reflection:

Which issues were the easiest to fix, and which were the hardest? Why?

Easiest: Removing the dangerous eval call and unused imports. These were simple one-line deletions.

Medium: Changing function names to snake_case (needed updating all references).

Hardest: Implementing proper input validation and resolving the mutable default argument, since it required changes to function signatures and logic. Fixing all spacing for blank lines was a bit tedious due to many small corrections across the file.

Did the static analysis tools report any false positives? If so, describe one example.

There were no significant false positives. Some style warnings (e.g., Pylint's global usage) are sometimes unavoidable in a single-file script, but are generally fair.

How would you integrate static analysis tools into your actual software development workflow?

I would add static analysis tools like Pylint, Flake8, and Bandit to my IDE and configure them in pre-commit hooks or CI (Continuous Integration) pipelines. This way, all code is checked before being merged or deployed, maintaining standards consistently within a team.

What tangible improvements did you observe in the code quality, readability, or potential robustness after applying the fixes?

The code is much more readable, properly documented, safely handles exceptions, and follows standard Python style. It is robust against invalid input, secure from dangerous patterns, and easier for collaborators to maintain.