# **Name: Laxman Srivastava**

**SRN: PES1UG23CS326**

**SECTION: F**

Repo link: [laxmanclo/se-lab5](https://github.com/laxmanclo/se-lab5)

# 

# 

# 

# 

# 

# **Lab 5 Reflection: Static Code Analysis**

## **1. Which issues were the easiest to fix, and which were the hardest? Why?**

Easiest to fix:

* String formatting (changing %s to f-strings) - just syntax change
* File handling with context managers (with open()) - straightforward pattern
* Removing the eval() statement - just delete dangerous code

Hardest to fix:

* Mutable default arguments - needed to understand WHY logs=[] is bad (Python creates the list once at function definition, not each call)
* Deciding which specific exceptions to catch - had to understand what errors each function could actually raise (KeyError, FileNotFoundError, etc.)
* Adding proper type validation without making code too verbose

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

Yes, Pylint flagged the logging import as unused (W0611), but this might be intentional if the code is meant to use logging in future development. In a real project, we'd either implement logging or remove the import.

Another borderline case: Pylint warned about global variable usage (W0603), but for this simple inventory system, using a global stock\_data dict is actually a reasonable design choice. In production, we'd use a class instead.

## **3. How would you integrate static analysis tools into your actual software development workflow?**

Local Development:

* Configure VS Code/PyCharm to run Flake8 automatically on save
* Run Pylint before committing code
* Use pre-commit hooks to automatically check code before git commits

CI/CD Pipeline:

* Add GitHub Actions workflow that runs all three tools on every pull request
* Fail the build if Bandit finds high-severity security issues
* Generate code quality reports and track improvements over time
* Set minimum quality scores (e.g., Pylint score must be > 8.0)

Team Standards:

* Create .pylintrc, .flake8 config files to enforce team conventions
* Review tool reports in code reviews
* Make passing static analysis a requirement before merging

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

Security Improvements:

* Removed eval() - eliminated code injection vulnerability
* Fixed bare exceptions - system won't silently fail on critical errors

Reliability Improvements:

* Fixed mutable default argument bug - logs parameter now works correctly
* Added error handling for missing files and invalid data
* Type validation prevents crashes from wrong input types

Readability Improvements:

* Added docstrings - now clear what each function does
* Used f-strings - logging messages are easier to read
* Better variable names - item, quantity instead of i
* Context managers make file handling obvious

Maintainability:

* Specific exceptions make debugging easier
* Proper documentation helps new developers understand code
* Following PEP 8 style makes code consistent with Python standards