SE LAB-5

VIVEK TUTIKA

PES2UG23CS707

SECTION-“K”

Issue	Type	Line(s)	Description	Fix Approach
Mutable default-arg	Bug	12	logs=[] shared across calls	Change default to None and initialize in method
Missing final newline	Style / Formatting	142/173	File ends without a single final newline or has missing/extra final newline.	Ensure file ends with exactly one newline character (add or remove extra blank lines).
Missing function docstrings	Documentation error	18, 41, 80, 84, 120, 133, 139, 154	Top-level functions lack docstrings.	Add concise docstrings for each function describing purpose, args, returns.
Open without encoding	formatting	87, 122	Files opened without explicit encoding.	Open files with encoding, open(file, "r", encoding="utf-8").

JSON load/save robustness	functional	load_data / save_data	Potential JSON errors, invalid types, reassigning global dict.	Use try/except for JSONDecodeError, validate types, convert values to int, update existing dict (clear/update) instead of reassign.
Line too long	Non functional/formatting	Many lines	Lines exceed 79-char limit.	Wrap long lines (split logging args, use implicit concatenation or variables) to <=79 chars.
addItem ignores logging module	functional	31	Uses log.appen	Use inbuilt module logging.info no parameter
Behavioral bug: invalid calls / type errors	functional	main / addItem call	Calls passing non-int quantities or wrong names remained in code.	Validate inputs (convert qty to int), fix example calls in main to use correct types and names.
Use of global statement	bug	85	global used to reassign module-level dict.	Avoid global reassignments: mutate existing dict (clear()/update()) or return new dict to caller.

REFLECTION-

1. Easiest vs hardest fixes

- Easiest: formatting/style (final newline, trailing blank lines, blank-line spacing, wrapping long lines).
- Hardest: syntax and functional fixes (filling empty function bodies, JSON load/save robustness, avoiding global reassignments). They require understanding intent, adding logic/error handling, and risk changing runtime behavior.

2. Static analysis false positives

- (C0116) for very small, self-explanatory helper functions can be noisy — it's a maintainability rule rather than an actual bug.

3. How to integrate linters into workflow

- Local: Editor integration + pre-commit hooks (pre-commit config running flake8/pylint/black/isort) so issues are caught before commit.
- CI: Add a pipeline step (GitHub Actions / GitLab CI) that runs tests and linters on PRs; fail the build for errors, warn for lower-severity rules.
- Practices: separate style auto-fixes (black/isort) from stricter quality checks (pylint/flake8); keep config files (.pylintrc, .flake8) in repo to avoid surprises; run linters in containers to match dev environment.

4. Tangible improvements after fixes

- Code is parseable and runnable (syntax errors removed).
- Fewer runtime failure modes: JSON errors handled, file encoding specified, qty inputs validated.
- Better maintainability: consistent snake_case naming, docstrings, and logging messages.
- Higher static-analysis scores and clearer, more uniform style (easier code review and onboarding).
- Generally none critical in this run. One minor example: missing-function-docstring