SE Lab 5: Static Code Analysis

NAME: V LAKSHITHA
SRN: PES2UG23CS665
SECTION: CSE-K

ISSUE TABLE:

Issue	Type	Description	Fix Approach
Type Error on qty addition	Runtime Type Error	Adding string quantity to int caused unsupported operand error	Added strict input validation inadd_itemto ensureqtyis an integer before addition.
Path configuration for PostgreSQL	Configuration Issue	psqlcommand not found in shell PATH	Exported PostgreSQL binary path to PATH in.zshrcto resolve command not found errors.
Bare except and catch-all in remove	Bad Practice	Using broad except masked underlying issues and reduced code clarity	Replaced bare except with specificKeyErrorexception handling inremove_item.
Mutable default argument (logs=[])	Python anti-pattern	Mutable default argument led to unexpected shared list between calls	Changed default tologs=Noneand created a new list inside the function.
Dangerous use ofeval	Security Risk	Use ofevalon code input posed security vulnerabilities	Removed all use ofevaland avoided executing arbitrary code strings.
File open/close withoutwith	Resource Leak Risk	File objects were not always closed properly	Usedwithstatement for file operations to ensure proper resource management.
Error-prone JSON deserialization	Error Handling	JSON file could be missing or corrupted	Added exception handling forFileNotFoundErrorandjson.JSONDecodeErrorinload_data.
Negative quantity allowed	Logical Bug	Negative quantities allowed in add_item causing negative stock	Added validation to reject negative quantities inadd_itemwith error message.
No error when removing nonexistent item	User Feedback Missing	Removing missing item failed silently	Added specific error message for missing item inremove_item.

Issue	Type	Description	Fix Approach
Invalid type inputs not validated	Stability Issue	Inputs like numeric item or string qty caused errors	Added type checks and validation with error messages inadd_itemandremove_item.
Code style and naming inconsistency	Code Quality	Inconsistent naming conventions and style	Followed PEP8 style including function names, variable naming, and added docstrings for readability and maintainability.
Logs argument handling	Functional Bug	Logs list shared as default mutable argument	Fixed with None default and inside assignment; Controlled log appending only on valid operations.
Silently ignoring invalid inputs	Debugging Difficulty	Early code ignored invalid input silently reducing error visibility	Added explicit print statements for invalid input detection to assist debugging and raise code quality.
Incomplete low stock item reporting	Feature Improvement	No clear method to identify low stock items	Addedcheck_low_itemsfunction to generate list of stock below threshold.
Inefficient JSON loading and dumping	Performance	Usedjson.loads(f.read())andf.write(json.dumps())	Replaced withjson.load(f)andjson.dump(stock_data, f)for efficient and idiomatic file operations.

REFLECTION:

1. **Which issues were the easiest to fix, and which were the hardest? Why?**
   The easiest issue to fix was the mutable default arguments and replacing bare except clauses because they were straightforward fixes requiring small code changes and standard Python best practices.
   The hardest to fix was input validation for type and logical correctness I had to repeatedly keep adding required checks across multiple functions without breaking existing behavior.

2. **Did the static analysis tools report any false positives? If so, describe one example.**
   The static analysis tools flagged the missing docstrings or line length as big issues, which despite being warnings, may not impact runtime correctness and could be subjective.
   example: flagging single-quoted strings or ordering of imports that had no effect on runtime.

3. **How would you integrate static analysis tools into your actual software development workflow? Consider continuous integration (CI) or local development practices.**
   I would integrate static analysis tools in the following ways:
   Incorporate linters (like pylint, flake8) and security scanners in CI pipelines to automatically check code quality on every push.
   - Use pre-commit hooks locally to catch issues before commit, enforcing consistent style and static correctness early.
   - Integrate automated tests alongside static checks for comprehensive validation.

4. **What tangible improvements did you observe in the code quality, readability, or potential robustness after applying the fixes?**
   - Code readability increased markedly with consistent naming, spacing, and added docstrings.
   - Robustness improved via input validation and error handling, reducing runtime crashes and undefined behavior.
   - Security was strengthened by eliminating eval and using safer file operations.
   - Easier maintenance and collaboration due to better structured and documented codebase.