

SE LAB-5

Medha Raj

PES2UG23CS334

Code of inventory_system.py

```
import json
import logging
from datetime import datetime
from typing import Dict, List, Optional

# Inventory data storage
STOCK_DATA: Dict[str, int] = {}

def add_item(item: str = "default", qty: int = 0,
             logs: Optional[List[str]] = None) -> None:
    """Add an item and quantity to the inventory, with validation."""
    if logs is None:
        logs = []

    if not isinstance(item, str) or not item.strip():
        logging.error("add_item: invalid 'item' type or empty name:
        %r", item)
        raise ValueError("item must be a non-empty string")

    if not isinstance(qty, int) or qty < 0:
        logging.error("add_item: invalid 'qty': %r", qty)
        raise ValueError("qty must be a non-negative integer")

    STOCK_DATA[item] = STOCK_DATA.get(item, 0) + qty
    timestamp = datetime.now().isoformat()
    log_entry = f"{timestamp}: Added {qty} of {item}"
    logs.append(log_entry)
    logging.info(log_entry)

def remove_item(item: str, qty: int) -> None:
    """Remove quantity of an item if present in inventory."""
    if not isinstance(item, str) or not item.strip():
        logging.error("remove_item: invalid 'item': %r", item)
```

```

        raise ValueError("item must be a non-empty string")

    if not isinstance(qty, int) or qty <= 0:
        logging.error("remove_item: invalid 'qty': %r", qty)
        raise ValueError("qty must be a positive integer")

    if item not in STOCK_DATA:
        logging.warning("remove_item: item %s not found.", item)
        return

    current = STOCK_DATA[item]
    if current <= qty:
        del STOCK_DATA[item]
        logging.info("Removed %d of %s; item deleted.", qty, item)
    else:
        STOCK_DATA[item] = current - qty
        logging.info("Removed %d of %s; new qty=%d.", qty, item,
STOCK_DATA[item])

def get_qty(item: str) -> int:
    """Return quantity of an item; 0 if not found."""
    if not isinstance(item, str) or not item.strip():
        logging.error("get_qty: invalid 'item': %r", item)
        raise ValueError("item must be a non-empty string")

    return STOCK_DATA.get(item, 0)

def load_data(file_path: str = "inventory.json") -> None:
    """Load inventory data safely from a JSON file."""
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            content = file.read().strip()
            if not content:
                logging.warning("load_data: file %s is empty.",
file_path)

                STOCK_DATA.clear()
                return

            data = json.loads(content)
            if not isinstance(data, dict):

```

```

        raise ValueError("inventory file must contain a JSON
object")

    STOCK_DATA.clear()
    STOCK_DATA.update(data)
    logging.info("Inventory loaded from %s", file_path)
except FileNotFoundError:
    logging.warning("File %s not found. Start in empty inventory.",
file_path)
    STOCK_DATA.clear()
except json.JSONDecodeError as err:
    logging.error("Failed to parse JSON in %s: %s", file_path, err)
    raise

def save_data(file_path: str = "inventory.json") -> None:
    """Save inventory data to a JSON file."""
    try:
        with open(file_path, "w", encoding="utf-8") as file:
            json.dump(STOCK_DATA, file, indent=2, ensure_ascii=False)
        logging.info("Inventory saved to %s", file_path)
    except OSError as err:
        logging.error("Failed to write to %s: %s", file_path, err)
        raise

def print_data() -> None:
    """Log the current inventory report."""
    logging.info("Items Report:")
    for name, qty in STOCK_DATA.items():
        logging.info("%s -> %d", name, qty)

def check_low_items(threshold: int = 5) -> List[str]:
    """Return items with quantity below a given threshold."""
    if not isinstance(threshold, int) or threshold < 0:
        logging.error("check_low_items: invalid threshold: %r",
threshold)
        raise ValueError("threshold must be a non-negative integer")

    return [name for name, qty in STOCK_DATA.items() if qty <
threshold]

```

```

def main() -> None:
    """Main execution for inventory demo."""
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(message)s",
    )

    try:
        add_item("apple", 10)
        add_item("banana", 2)
    except ValueError as err:
        logging.warning("main: skipping invalid add_item: %s", err)

    try:
        add_item(123, "ten") # invalid input to test validation
    except ValueError as err:
        logging.warning("main: invalid add_item input: %s", err)

    remove_item("apple", 3)
    remove_item("orange", 1)

    logging.info("Apple stock: %d", get_qty("apple"))
    logging.info("Low items: %s", check_low_items())

    try:
        save_data()
        load_data()
        print_data()
    except Exception as err:
        logging.error("main: error during save/load: %s", err)

if __name__ == "__main__":
    main()

```

Screenshots:

```
bandit_report.txt
1 Run started:2025-10-27 07:09:36.808509
2
3 Test results:
4   No issues identified.
5
6 Code scanned:
7   Total lines of code: 113
8   Total lines skipped (#nosec): 0
9   Total potential issues skipped due to specifically being disabled (e.g., #nosec BXXX): 0
10
11 Run metrics:
12   Total issues (by severity):
13     Undefined: 0
14     Low: 0
15     Medium: 0
16     High: 0
17   Total issues (by confidence):
18     Undefined: 0
19     Low: 0
20     Medium: 0
21     High: 0
22 Files skipped (0):
23
```

```
flake8_report.txt
1 Generate code (Ctrl+I), or select a language (Ctrl+K M). Start typing to dismiss or don't show this again.
```

```
pylint_report.txt
1 ***** Module inventory_system
2 inventory_system.py:1:0: C0114: Missing module docstring (missing-module-docstring)
3 inventory_system.py:143:11: W0718: Catching too general exception Exception (broad-exception-caught)
4
5 -----
6 Your code has been rated at 9.80/10 (previous run: 9.80/10, +0.00)
7
8
```

Issue Table:

no.	Issue	Type	Approx. Line(s)	Description	Fix approach
1	Mutable default argument (logs=[])	Bug / Style	addItem definition	Default list shared between calls leads to surprising shared state.	Change default to None and initialize inside function.
2	Bare except: in removeItem	Security / Reliability	removeItem	Catches all exceptions (including KeyboardInterrupt, SystemExit), hides errors.	Catch specific exceptions (KeyError, TypeError) and log the error.
3	eval("print('eval used')") usage	Security (High)	main()	eval is dangerous and unnecessary.	Remove eval; replace with safe logging or direct function call.
4	No input validation in addItem and removeItem	Bug / Reliability	addItem, removeItem	Functions accept wrong types (addItem(123, "ten")), negative quantities.	Add type and value checks; raise ValueError or log and return.
5	Unsafe file I/O (open/read/write without with)	Style / Reliability	loadData, saveData	Files not closed on exceptions.	Use with open(...) and handle FileNotFoundError / JSONDecodeError / IOError.
6	getQty may raise KeyError	Bug	getQty	Calling getQty for missing item raises KeyError and crashes.	Return 0 or use dict.get and document behavior.
7	Prints used instead of logging; no logging config	Style / Maintainability	multiple	Print statements used; no logging config for module.	Configure logging in main; use logging instead of prints and f-strings.

no.	Issue	Type	Approx. Line(s)	Description	Fix approach
8	Global mutable stock_data used without encapsulation	Design	top of file	Global state may cause tests/side effects.	Keep global but access consistently; consider returning state or wrapping in a class (not done here to keep minimal changes).

REFLECTION QUESTIONS

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

Easiest: Style and formatting issues flagged by Flake8 were the simplest to fix. Problems like long lines (E501), missing blank lines, and string formatting were straightforward because the feedback was explicit and the corrections had no effect on logic.

Hardest: Logic-related or design-related fixes, such as adding input validation, replacing the bare except: with specific exception types, and configuring logging correctly were more challenging. These required understanding how the program worked and ensuring that changes didn't break functionality.

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

There were minor Pylint warnings (e.g., suggesting to remove "global variables" or to rename constants) that didn't represent real errors for this small lab program. In this context, those were false positives because the global dictionary STOCK_DATA was intentionally kept simple for the exercise, and changing it to a class would have added unnecessary complexity.

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

Integrate Pylint, Flake8, and Bandit into a pre-commit hook so code is automatically checked before each commit. Add these tools to a Continuous Integration (CI) pipeline (e.g., GitHub Actions) so that every pull request runs the checks automatically. Use VS Code extensions for Pylint or Flake8 to get

real-time linting feedback during local development, catching problems before pushing code.

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

The code is now cleaner, safer, and more maintainable:

- Replacing `eval()` and bare `except:` improved security and reliability.
- Adding input validation prevents invalid data from corrupting the inventory.
- Using logging instead of `print()` improved traceability and debugging.
- Refactoring long lines and adding docstrings improved readability and made the code PEP 8 compliant.

Overall, the program feels more professional and production-ready, with clear error messages, safer operations, and consistent style.