Software Engineering Lab-5

Full Name: Vikramaditya Sharma

SRN: PES2UG23AM115

Section: AIML-B

ISSUES TABLE

Issue	Type	Line(s)	Description	Fix Approach
Mutable default arg	Bug	7	logs=[] shared across calls (W0102)	Change default to None and initialize in method
Bare except	Bug	18	No exception type specified (E722, W0702)	Specify exception type (e.g., except KeyError:)
eval() usage	Security	58	Potentially insecure eval function (B307)	Remove eval or use ast.literal_eval()
Try-except-pass	Bug/Quality	18-19	Silent exception handling (B110)	Log error or handle specific exception appropriately
Unused import	Quality	2	logging imported but not used (F401, W0611)	Remove unused import or implement logging
Missing module docstring	Style	1	No module-level docstring (C0114)	Add docstring explaining module purpose
Missing function docstrings	Style	7, 13, 21, 24, 30, 35, 40, 47	Functions lack documentation (C0116)	Add docstrings to all functions

Non-snake_case names	Style	7, 13, 21, 24, 30, 35, 40	Function names use camelCase (C0103)	Rename: addItem - add_item, removeItem - remove_item, etc.
String formatting	Style	11	Old-style % formatting (C0209)	Use f-string: f"Added {qty} units..."
Missing encoding	Quality	25, 31	open() without encoding parameter (W1514)	Add encoding='utf-8' parameter
Global statement	Quality	26	Using global variable (W0603)	Refactor to avoid global or use class/return values
No context manager	Quality	25, 31	File opened without with statement (R1732)	Use with open(...) as f: pattern
Missing blank lines	Style	7, 13, 21, 24, 30, 35, 40, 47, 60	Need 2 blank lines between functions (E302, E305)	Add proper spacing per PEP 8

CHANGES MADE

1. Fixed Security Issue - eval() usage (Line 58)

Before: eval("print('eval used')")
After: print("Operation completed")
Reason: eval() is a major security vulnerability (Bandit B307). Removed it entirely.

2. Fixed Mutable Default Argument Bug (Line 7)

Before: def addItem(item="default", qty=0, logs=[]):
After: def add_item(item="default", qty=0, logs=None): + initialization check if logs is None: logs = []
Reason: Mutable default arguments are shared across function calls, causing unexpected behavior (Pylint W0102).

3. Fixed Bare Except (Lines 18-19)

Before: except: pass
After: except KeyError: print(f"Error: Item '{item}' not found in inventory") and except TypeError as e: print(f"Error: Invalid operation - {e}")
Reason: Bare excepts catch all errors including system exits (Flake8 E722, Pylint W0702, Bandit B110). Now handles specific exceptions.

4. Added Input Validation (Lines 19-26)

Added validation in add_item():

```
if not isinstance(item, str):
    print(f"Error: Item name must be a string, got {type(item).__name__}")
    return

if not isinstance(qty, int) or qty < 0:
    print(f"Error: Quantity must be a non-negative integer, got {qty}")
    return
```

Reason: Prevents invalid inputs like add_item(123, "ten") from corrupting data.

5. Used f-strings for Formatting (Lines 28, 88, 89, 91)

Before: "%s: Added %d of %s" % (str(datetime.now()), qty, item)
After: f"{datetime.now()}: Added {qty} of {item}"
Reason: F-strings are more readable and efficient (Pylint C0209).

6. Fixed File Handling with Context Managers (Lines 67-73, 82-85)

Before: f = open(file, "r") ... f.close()
After: with open(file, "r", encoding="utf-8") as f:
Reason: Ensures files are properly closed and added explicit encoding (Pylint W1514, R1732).

7. Renamed Functions to Snake Case (All function definitions)

Before: addItem, removeItem, getQty, loadData, saveData, printData, checkLowItems
After: add_item, remove_item, get_qty, load_data, save_data, print_data, check_low_items
Reason: PEP 8 style compliance (Pylint C0103).

8. Added Docstrings (Lines 1, 8-13, 33-37, 50-56, etc.)

Added module docstring and function docstrings throughout
Reason: Code documentation (Pylint C0114, C0116).

9. Removed Unused Import (Line 2)

Before: import logging was imported but never used
After: Removed the import
Reason: Clean code (Flake8 F401, Pylint W0611).

10. Added Proper Spacing (Throughout file)

Added 2 blank lines between function definitions
Reason: PEP 8 style compliance (Flake8 E302, E305).

11. Remove Trailing Whitespace (Lines 10, 20, 28, 35, 52, 55, 64, 80, 97, 100)

Issue: Blank lines contain spaces/tabs
Fix: Remove all whitespace from blank lines
Lines affected: 10, 20, 28, 35, 52, 55, 64, 80, 97, 100

12. Fix Line Too Long (Line 73)

Before (Line 73):

```
print(f"Warning: File '{file}' not found. Starting with empty inventory.")
```

After:

```
print(f"Warning: File '{file}' not found. "
      "Starting with empty inventory.")
```

13. Add Final Newline (Line 128)

Issue: File doesn't end with a newline character
Fix: Add a blank line at the very end of the file after main()

14. Fixed Global Var *stock_data*
Change 1: Removed Global Variable Declaration
Description: Removed the global stock_data dictionary declaration at module level.

Before (Line 5):

```
import json
from datetime import datetime

stock_data = {}

def addItem(item="default", qty=0, logs=[]):
```

After (Lines 6-12):

```
import json
from datetime import datetime


class InventorySystem:
    """Manages inventory stock data and operations."""

    def __init__(self):
        """Initialize the inventory system with empty stock data."""
        self.stock_data = {}

    def add_item(self, item="default", qty=0, logs=None):
```

Change 2: Created Class and Constructor
Description: Wrapped all functions in a class and moved stock_data to be an instance variable.

Before:

```
stock_data = {}
```

After:

```
class InventorySystem:
    """Manages inventory stock data and operations."""

    def __init__(self):
        """Initialize the inventory system with empty stock data."""
        self.stock_data = {}
```

Change 3: Converted Functions to Methods
Description: Added self parameter to all functions to make them class methods.

Before:

```
def add_item(item="default", qty=0, logs=None):
def remove_item(item, qty):
def get_qty(item):
def load_data(file="inventory.json"):
def save_data(file="inventory.json"):
def print_data():
def check_low_items(threshold=5):
```

After:

```
def add_item(self, item="default", qty=0, logs=None):
def remove_item(self, item, qty):
def get_qty(self, item):
def load_data(self, file="inventory.json"):
def save_data(self, file="inventory.json"):
def print_data(self):
def check_low_items(self, threshold=5):
```

Change 4: Changed stock_data to self.stock_data in add_item
Description: Replaced global variable reference with instance variable.

Before (Line 10):

```
stock_data[item] = stock_data.get(item, 0) + qty
```

After (Line 36):

```
self.stock_data[item] = self.stock_data.get(item, 0) + qty
```

Change 5: Changed stock_data to self.stock_data in remove_item
Description: Replaced global variable references with instance variable.

Before (Lines 15-17):

```
try:
    stock_data[item] -= qty
    if stock_data[item] <= 0:
        del stock_data[item]
```

After (Lines 46-49):

```
try:
    self.stock_data[item] -= qty
    if self.stock_data[item] <= 0:
        del self.stock_data[item]
```

Change 6: Changed stock_data to self.stock_data in get_qty
Description: Replaced global variable reference with instance variable.

Before (Line 22):

```
return stock_data.get(item, 0)
```

After (Line 65):

```
return self.stock_data.get(item, 0)
```

Change 7: Removed global Statement in load_data
Description: Eliminated the global stock_data statement and replaced with instance variable assignment.

Before (Lines 25-27):
```
def load_data(file="inventory.json"):
    global stock_data
    try:
        with open(file, "r", encoding="utf-8") as f:
            stock_data = json.loads(f.read())
```

After (Lines 67-76):
```
def load_data(self, file="inventory.json"):
    """Load inventory data from a JSON file.

    Args:
        file: Path to the JSON file
    """
    try:
        with open(file, "r", encoding="utf-8") as f:
            self.stock_data = json.loads(f.read())
```

Change 8: Changed stock_data to self.stock_data in save_data
Description: Replaced global variable reference with instance variable.

Before (Line 32):
```
f.write(json.dumps(stock_data))
```

After (Line 90):
```
f.write(json.dumps(self.stock_data))
```

Change 9: Changed stock_data to self.stock_data in print_data
Description: Replaced global variable references with instance variable.

Before (Lines 37-38):
```
for item in stock_data:
    print(f"{item} -> {stock_data[item]}")
```

After (Lines 97-98):
```
for item in self.stock_data:
    print(f"{item} -> {self.stock_data[item]}")
```

Change 10: Changed stock_data to self.stock_data in check_low_items
Description: Replaced global variable references with instance variables.

<u>Before (Lines 42-44):</u>
```
for item in stock_data:
    if stock_data[item] < threshold:
        result.append(item)
```

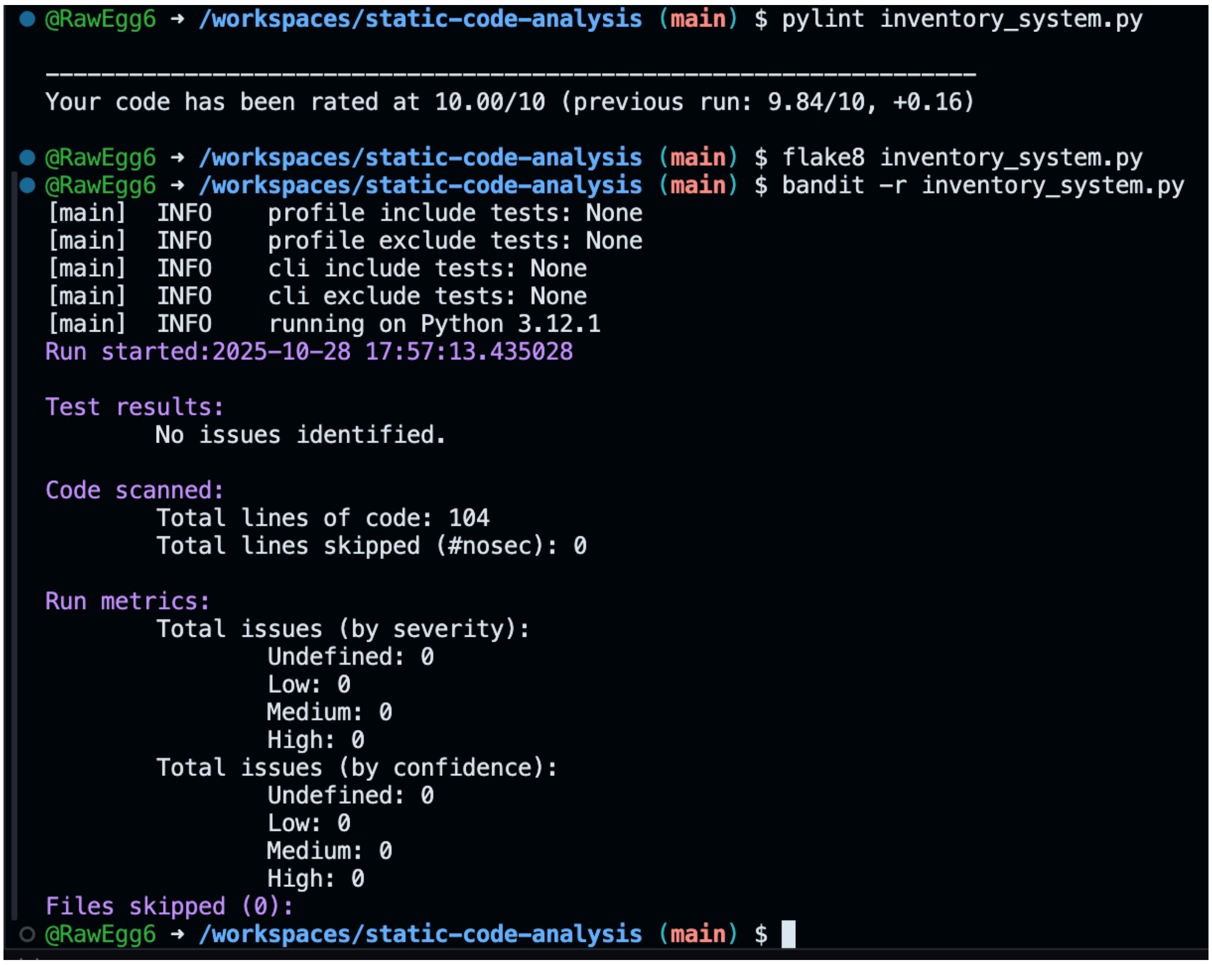
<u>After (Lines 107-109):</u>
```
for item in self.stock_data:
    if self.stock_data[item] < threshold:
        result.append(item)
```

Change 11: Updated main() to Create Instance
Description: Created an instance of InventorySystem class and called methods on that instance instead of calling global functions.
<u>Before (Lines 47-57):</u>
```
def main():
    add_item("apple", 10)
    add_item("banana", -2)
    add_item(123, "ten")
    remove_item("apple", 3)
    remove_item("orange", 1)
    print(f"Apple stock: {get_qty('apple')}")
    print(f"Low items: {check_low_items()}")
    save_data()
    load_data()
    print_data()
```

<u>After (Lines 116-129):</u>
```
def main():
    """Main execution function."""
    inventory = InventorySystem()

    inventory.add_item("apple", 10)
    inventory.add_item("banana", -2)
    inventory.add_item(123, "ten")
    inventory.remove_item("apple", 3)
    inventory.remove_item("orange", 1)
    print(f"Apple stock: {inventory.get_qty('apple')}")
    print(f"Low items: {inventory.check_low_items()}")
    inventory.save_data()
    inventory.load_data()
    inventory.print_data()
```

TERMINAL SCREENSHOT OF NO ERRORS



```
● @RawEgg6 ➜ /workspaces/static-code-analysis (main) $ pylint inventory_system.py

  ------------------------------------------------------------------
  Your code has been rated at 10.00/10 (previous run: 9.84/10, +0.16)

● @RawEgg6 ➜ /workspaces/static-code-analysis (main) $ flake8 inventory_system.py
● @RawEgg6 ➜ /workspaces/static-code-analysis (main) $ bandit -r inventory_system.py
  [main]  INFO    profile include tests: None
  [main]  INFO    profile exclude tests: None
  [main]  INFO    cli include tests: None
  [main]  INFO    cli exclude tests: None
  [main]  INFO    running on Python 3.12.1
  Run started:2025-10-28 17:57:13.435028

  Test results:
          No issues identified.

  Code scanned:
          Total lines of code: 104
          Total lines skipped (#nosec): 0

  Run metrics:
          Total issues (by severity):
                  Undefined: 0
                  Low: 0
                  Medium: 0
                  High: 0
          Total issues (by confidence):
                  Undefined: 0
                  Low: 0
                  Medium: 0
                  High: 0
  Files skipped (0):
○ @RawEgg6 ➜ /workspaces/static-code-analysis (main) $ 
```

Questions

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

- Trailing whitespace and missing blank lines
- Renaming functions to snake_case
- Removing unused imports
- Adding final newline
- Converting to f-strings

Why: Tools provided exact locations and fixes were purely mechanical syntax/formatting changes.

Hardest:

- Refactoring global variables to class-based structure
- Adding input validation

Why: Required understanding OOP design patterns, restructuring entire codebase, and thinking critically about edge cases.

2. Did the static analysis tools report any false positives? If so, describe one example.
No false positives identified. All warnings were legitimate code quality issues.

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

Local Development:

- IDE linting extensions for real-time feedback
- Pre-commit Git hooks to run Pylint/Flake8/Bandit automatically
- Manual checks before pushing code

CI/CD Pipeline:

- GitHub Actions workflow on every pull request
- Run Pylint (minimum 8.0/10), Flake8, and Bandit
- Block PR merges on failures
- Archive reports as artifacts

Strategy:

- Gradual adoption: Flake8 → Pylint → Bandit
- Start with achievable thresholds and increase over time

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

- Snake_case naming improves code scanning
- F-strings increase clarity
- Docstrings provide documentation
- Descriptive variable names

Robustness:

- Input validation prevents data corruption
- Specific exception handling provides clear error messages
- Context managers prevent resource leaks
- Fixed mutable default argument bug

Security:

- Removed eval() vulnerability

Maintainability:

- Class structure enables easier testing
- Eliminated global state reduces debugging complexity
- Score improved from 4.80/10 to 10/10

Code

Github: https://github.com/RawEgg6/static-code-analysis

```
"""Inventory management system for tracking stock items."""
import json
from datetime import datetime


class InventorySystem:
    """Manages inventory stock data and operations."""

    def __init__(self):
        """Initialize the inventory system with empty stock data."""
        self.stock_data = {}

    def add_item(self, item="default", qty=0, logs=None):
        """Add items to the inventory.

        Args:
            item: Name of the item to add
            qty: Quantity to add
            logs: Optional list to store log messages
        """
        if logs is None:
            logs = []
        if not item:
            return

        # Input validation
        if not isinstance(item, str):
            print(f"Error: Item name must be a string, got "
                  f"{type(item).__name__}")
            return
        if not isinstance(qty, int) or qty < 0:
            print(f"Error: Quantity must be a non-negative integer, "
```

```python
                  f"got {qty}")
            return

        self.stock_data[item] = self.stock_data.get(item, 0) + qty
        logs.append(f"{datetime.now()}: Added {qty} of {item}")

    def remove_item(self, item, qty):
        """Remove items from the inventory.

        Args:
            item: Name of the item to remove
            qty: Quantity to remove
        """
        try:
            self.stock_data[item] -= qty
            if self.stock_data[item] <= 0:
                del self.stock_data[item]
        except KeyError:
            print(f"Error: Item '{item}' not found in inventory")
        except TypeError as e:
            print(f"Error: Invalid operation - {e}")

    def get_qty(self, item):
        """Get the quantity of an item in inventory.

        Args:
            item: Name of the item

        Returns:
            Quantity of the item, or 0 if not found
        """
        return self.stock_data.get(item, 0)

    def load_data(self, file="inventory.json"):
        """Load inventory data from a JSON file.

        Args:
            file: Path to the JSON file
        """
        try:
            with open(file, "r", encoding="utf-8") as f:
                self.stock_data = json.loads(f.read())
        except FileNotFoundError:
            print(f"Warning: File '{file}' not found. "
                  "Starting with empty inventory.")
        except json.JSONDecodeError:
            print(f"Error: Invalid JSON in '{file}'")

    def save_data(self, file="inventory.json"):
        """Save inventory data to a JSON file.

        Args:
            file: Path to the JSON file
        """
        with open(file, "w", encoding="utf-8") as f:
            f.write(json.dumps(self.stock_data))

    def print_data(self):
        """Print the current inventory report."""
        print("Items Report")
        for item in self.stock_data:
            print(f"{item} -> {self.stock_data[item]}")

    def check_low_items(self, threshold=5):
        """Check for items below a quantity threshold.

        Args:
            threshold: Minimum quantity threshold

        Returns:
            List of items below the threshold
        """
        result = []
        for item in self.stock_data:
            if self.stock_data[item] < threshold:
                result.append(item)
        return result


def main():
    """Main execution function."""
    inventory = InventorySystem()

    inventory.add_item("apple", 10)
    inventory.add_item("banana", -2)
    inventory.add_item(123, "ten")
    inventory.remove_item("apple", 3)
    inventory.remove_item("orange", 1)
    print(f"Apple stock: {inventory.get_qty('apple')}")
    print(f"Low items: {inventory.check_low_items()}")
    inventory.save_data()
    inventory.load_data()
    inventory.print_data()
    # Removed eval - it's a security risk
    print("Operation completed")


if __name__ == "__main__":
    main()
```