SE Lab-5

Name: Nikitha Devaraj

SRN: PES2UG23CS388

Repository link:

https://github.com/PES2UG23CS388/SE-LAB5-Static-Code_Analysis

Final code:

```
"""
A simple inventory management system module.

This module allows for adding, removing, and querying item quantities,
as well as loading from and saving to a JSON file.
"""

import json
from datetime import datetime
from typing import Dict, List  # Used for type hinting

# No global variables; data is passed as arguments.


def add_item(inventory: Dict[str, int], item: str, qty: int) -> None:
    """
    Add a specified quantity of an item to the inventory.
```

```
    Args:
        inventory (Dict[str, int]): The inventory dictionary to modify.
        item (str): The name of the item to add.
        qty (int): The quantity to add.
    """
    # Added type checking for robustness
    if not isinstance(item, str) or not isinstance(qty, int):
        print(f"Error: Invalid types for item ({type(item)}) or qty ({type(qty)}).")
        return

    inventory[item] = inventory.get(item, 0) + qty
    # Use f-strings for cleaner output
    print(f"{datetime.now()}: Added {qty} of {item}")


def remove_item(inventory: Dict[str, int], item: str, qty: int) -> None:
    """
    Remove a specified quantity of an item from the inventory.

    Args:
        inventory (Dict[str, int]): The inventory dictionary to modify.
        item (str): The name of the item to remove.
        qty (int): The quantity to remove.
    """
    if item not in inventory:
```

```
        print(f"Warning: Item '{item}' not in stock. Cannot remove.")
        return

    try:
        inventory[item] -= qty
        if inventory[item] <= 0:
            del inventory[item]
            print(f"Removed all of item '{item}'.")
        else:
            print(f"Removed {qty} of '{item}'. New total: {inventory[item]}")
    # Catch specific errors
    except (TypeError, ValueError):
        print(f"Error: Invalid quantity '{qty}' for item '{item}'.")


def get_qty(inventory: Dict[str, int], item: str) -> int:
    """
    Get the current quantity of a specific item.

    Args:
        inventory (Dict[str, int]): The inventory dictionary.
        item (str): The name of the item to query.

    Returns:
        int: The quantity of the item, or 0 if not found.
    """
```

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



def load_data(filename: str = "inventory.json") -> Dict[str, int]:
    """
    Load inventory data from a JSON file.

    Args:
        filename (str): The name of the file to load.

    Returns:
        Dict[str, int]: The loaded inventory dictionary.
    """
    try:
        # Renamed 'f' to 'file_handle' to satisfy Pylint (C0103)
        with open(filename, "r", encoding="utf-8") as file_handle:
            data = json.load(file_handle)
            if not isinstance(data, dict):
                print("Error: Data in file is not a dictionary. Starting fresh.")
                return {}
            return data
    except FileNotFoundError:
        print(f"Warning: Data file '{filename}' not found. Starting fresh inventory.")
        return {}
    except json.JSONDecodeError:
        print(f"Error: Could not decode JSON from '{filename}'. Starting fresh.")
```

```
        return {}


def save_data(inventory: Dict[str, int], filename: str = "inventory.json") -> None:
    """
    Save the current inventory data to a JSON file.

    Args:
        inventory (Dict[str, int]): The inventory dictionary to save.
        filename (str): The name of the file to save to.
    """
    try:
        # Renamed 'f' to 'file_handle' to satisfy Pylint (C0103)
        with open(filename, "w", encoding="utf-8") as file_handle:
            json.dump(inventory, file_handle, indent=4)
        print(f"Inventory saved to {filename}.")
    # Renamed 'e' to 'error' to satisfy Pylint (C0103)
    except IOError as error:
        print(f"Error saving data to {filename}: {error}")


def print_data(inventory: Dict[str, int]) -> None:
    """
    Print a report of all items and their quantities.

    Args:
```

```
    inventory (Dict[str, int]): The inventory to print.
    """
    print("\n--- Items Report ---")
    if not inventory:
        print("Inventory is empty.")
    else:
        for item, qty in inventory.items():
            print(f"{item} -> {qty}")
    print("----------\n")


def check_low_items(inventory: Dict[str, int], threshold: int = 5) -> List[str]:
    """
    Return a list of items with quantities below the threshold.

    Args:
        inventory (Dict[str, int]): The inventory to check.
        threshold (int): The low-stock threshold.

    Returns:
        List[str]: A list of item names that are low in stock.
    """
    # Using a list comprehension is considered "Pythonic"
    return [item for item, qty in inventory.items() if qty < threshold]
```

```
def main() -> None:
    """
    Main function to run the inventory management program.
    """
    stock_data = load_data()

    add_item(stock_data, "apple", 10)
    add_item(stock_data, "banana", 2)
    add_item(stock_data, 123, "ten")  # Safely handled by type check
    remove_item(stock_data, "apple", 3)
    remove_item(stock_data, "orange", 1)  # Safely handled

    print(f"Apple stock: {get_qty(stock_data, 'apple')}")

    low_items = check_low_items(stock_data)
    print(f"Low items: {low_items}")

    print_data(stock_data)
    save_data(stock_data)
    print("Program finished.")


# Standard __name__ == "__main__" guard
if __name__ == "__main__":
    main()
# THIS IS THE FINAL LINE. MAKE SURE THERE IS ONE EMPTY LINE AFTER THIS COMMENT.
```

```python
"""
A simple inventory management system module.

This module allows for adding, removing, and querying item quantities,
as well as loading from and saving to a JSON file.
"""

import json
from datetime import datetime
from typing import Dict, List  # Used for type hinting

# No global variables; data is passed as arguments.


def add_item(inventory: Dict[str, int], item: str, qty: int) -> None:
    """
    Add a specified quantity of an item to the inventory.

    Args:
        inventory (Dict[str, int]): The inventory dictionary to modify.
        item (str): The name of the item to add.
        qty (int): The quantity to add.
    """
    # Added type checking for robustness
    if not isinstance(item, str) or not isinstance(qty, int):
        print(f"Error: Invalid types for item ({type(item)}) or qty ({type(qty)}).")
        return

    inventory[item] = inventory.get(item, 0) + qty
    # Use f-strings for cleaner output
```

```
        print(f"{datetime.now()}: Added {qty} of {item}")


def remove_item(inventory: Dict[str, int], item: str, qty: int) -> None:
        """
        Remove a specified quantity of an item from the inventory.

        Args:
            inventory (Dict[str, int]): The inventory dictionary to modify.
            item (str): The name of the item to remove.
            qty (int): The quantity to remove.
        """
        if item not in inventory:
            print(f"Warning: Item '{item}' not in stock. Cannot remove.")
            return

        try:
            inventory[item] -= qty
            if inventory[item] <= 0:
                del inventory[item]
                print(f"Removed all of item '{item}'.")
            else:
                print(f"Removed {qty} of '{item}'. New total: {inventory[item]}")
        # Catch specific errors
        except (TypeError, ValueError):
            print(f"Error: Invalid quantity '{qty}' for item '{item}'.")


def get_qty(inventory: Dict[str, int], item: str) -> int:
        """
        Get the current quantity of a specific item.

```

```
        Args:
            inventory (Dict[str, int]): The inventory dictionary.
            item (str): The name of the item to query.

        Returns:
            int: The quantity of the item, or 0 if not found.
        """
        return inventory.get(item, 0)


def load_data(filename: str = "inventory.json") -> Dict[str, int]:
        """
        Load inventory data from a JSON file.

        Args:
            filename (str): The name of the file to load.

        Returns:
            Dict[str, int]: The loaded inventory dictionary.
        """
        try:
            # Renamed 'f' to 'file_handle' to satisfy Pylint (C0103)
            with open(filename, "r", encoding="utf-8") as file_handle:
                data = json.load(file_handle)
                if not isinstance(data, dict):
                    print("Error: Data in file is not a dictionary. Starting fresh.")
                    return {}
                return data
        except FileNotFoundError:
            print(f"Warning: Data file '{filename}' not found. Starting fresh inventory.")
            return {}
```

```
        except json.JSONDecodeError:
            print(f"Error: Could not decode JSON from '{filename}'. Starting fresh.")
            return {}


def save_data(inventory: Dict[str, int], filename: str = "inventory.json") -> None:
    """
    Save the current inventory data to a JSON file.

    Args:
        inventory (Dict[str, int]): The inventory dictionary to save.
        filename (str): The name of the file to save to.
    """
    try:
        # Renamed 'f' to 'file_handle' to satisfy Pylint (C0103)
        with open(filename, "w", encoding="utf-8") as file_handle:
            json.dump(inventory, file_handle, indent=4)
        print(f"Inventory saved to {filename}.")
    # Renamed 'e' to 'error' to satisfy Pylint (C0103)
    except IOError as error:
        print(f"Error saving data to {filename}: {error}")


def print_data(inventory: Dict[str, int]) -> None:
    """
    Print a report of all items and their quantities.

    Args:
        inventory (Dict[str, int]): The inventory to print.
    """
    print("\n--- Items Report ---")

    if not inventory:
        print("Inventory is empty.")
    else:
        for item, qty in inventory.items():
            print(f"{item} -> {qty}")
    print("----------------------\n")


def check_low_items(inventory: Dict[str, int], threshold: int = 5) -> List[str]:
    """
    Return a list of items with quantities below the threshold.

    Args:
        inventory (Dict[str, int]): The inventory to check.
        threshold (int): The low-stock threshold.

    Returns:
        List[str]: A list of item names that are low in stock.
    """
    # Using a list comprehension is considered "Pythonic"
    return [item for item, qty in inventory.items() if qty < threshold]


def main() -> None:
    """
    Main function to run the inventory management program.
    """
    stock_data = load_data()

    add_item(stock_data, "apple", 10)
    add_item(stock_data, "banana", 2)
    add_item(stock_data, 123, "ten")  # Safely handled by type check
```

```
154         add_item(stock_data, "apple", 10)
155         add_item(stock_data, "banana", 2)
156         add_item(stock_data, 123, "ten")  # Safely handled by type check
157         remove_item(stock_data, "apple", 3)
158         remove_item(stock_data, "orange", 1)  # Safely handled
159
160         print(f"Apple stock: {get_qty(stock_data, 'apple')}")
161
162         low_items = check_low_items(stock_data)
163         print(f"Low items: {low_items}")
164
165         print_data(stock_data)
166         save_data(stock_data)
167         print("Program finished.")
168
169
170     # Standard __name__ == "__main__" guard
171     if __name__ == "__main__":
172         main()
173     # THIS IS THE FINAL LINE. MAKE SURE THERE IS ONE EMPTY LINE AFTER THIS COMMENT.
```

Code Execution:

```
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $ pylint inventory_system.py > pylint_report.txt
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $ bandit -r inventory_system.py > bandit_report.txt
[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
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $ flake8 inventory_system.py > flake8_report.txt
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $
0
```

```
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $ python inventory_system.py
2025-10-27 16:48:33.751992: Added 10 of apple
2025-10-27 16:48:33.752019: Added 2 of banana
Error: Invalid types for item (<class 'int'>) or qty (<class 'str'>).
Removed 3 of 'apple'. New total: 28
Warning: Item 'orange' not in stock. Cannot remove.
Apple stock: 28
Low items: []

--- Items Report ---
apple -> 28
banana -> 8
--------------------

Inventory saved to inventory.json.
Program finished.
@PES2UG23CS388 ➜/workspaces/SE-LAB5-Static-Code_Analysis (main) $
```

bandit_report.txt

```
Run started:2025-10-27 16:46:57.050718

Test results:
    No issues identified.

Code scanned:
    Total lines of code: 126
    Total lines skipped (#nosec): 0
    Total potential issues skipped due to specifically being disabled (e.g., #nosec BXXX): 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):

```

flake8_report.txt

```
inventory_system.py:26:80: E501 line too long (84 > 79 characters)
inventory_system.py:88:80: E501 line too long (81 > 79 characters)
inventory_system.py:92:80: E501 line too long (86 > 79 characters)
inventory_system.py:95:80: E501 line too long (81 > 79 characters)
inventory_system.py:99:80: E501 line too long (83 > 79 characters)
inventory_system.py:133:80: E501 line too long (80 > 79 characters)

```

Issue	Type	Line(s) (Original)	Description	Fix Approach
Use of eval()	Vulnerability / Security	67	The eval() function can execute arbitrary text as code, creating a major security vulnerability.	Removed the eval() call entirely as it served no function for the inventory system.
Mutable Default Arg	Bug	8	The list logs=[] in addItem was a mutable default argument, meaning it	Removed the logs parameter completely, as it was unused and print() was used for

Issue	Type	Line(s) (Original)	Description	Fix Approach
			would be shared across all function calls.	logging instead.
Bare `except`	Bug / Bad Practice	18	`except:` catches *all* errors, including system-level ones, and silently hides bugs like `KeyError` or `TypeError`.	Replaced the bare `except:` with a specific `try...except (TypeError, ValueError):` to catch invalid quantity types. Added a separate `if item not in inventory:` check to handle `KeyError` gracefully.
Use of Global Variable	Bad Practice	5, 27, 32	Using `global stock_data` makes the program's state hard to track and maintain.	Refactored the code to remove the global variable. `stock_data` is now created in `main()` and passed as an argument to all functions that need it.
No Input Validation	Bug / Robustness	8, 14, 48	Functions `addItem` and `removeItem` would crash if given invalid data types (e.g., `addItem(123, "ten")`).	Added `isinstance()` checks to `add_item` to validate input types. The fix to `remove_item`'s `except` block also helps catch this.

Issue	Type	Line(s) (Original)	Description	Fix Approach
Improper File Closing	Bad Practice / Bug	26, 28, 31, 33	Manually calling `f.open()` and `f.close()` can lead to resource leaks if an error occurs before `f.close()` is run.	Replaced the manual calls with the `with open(...) as file_handle:` context manager, which automatically and safely closes the file.
Missing File Handling	Bug / Robustness	25	The `loadData` function would crash with a `FileNotFoundError` if the `inventory.json` file didn't exist.	Wrapped the file logic in `load_data` inside a `try...except FileNotFoundError:` block to handle the case gracefully and return a new, empty dictionary.
Inconsistent Naming	Style	8, 14, 21, etc.	Function names used `camelCase` (e.g., `addItem`), which violates Python's PEP 8 `snake_case` convention.	Renamed all functions to `snake_case` (e.g., `add_item`, `remove_item`, `load_data`) for PEP 8 compliance.
Unused Import	Style	2	The `logging` module was imported but never used, adding unnecessary clutter.	Removed the `import logging` line.
Missing Docstrings	Style	All	The module and all functions lacked	Added a PEP 257 compliant module-level docstring and

Issue	Type	Line(s) (Original)	Description	Fix Approach
			docstrings, making the code hard to understand.	a docstring to every function explaining its purpose, Args, and Returns.

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

- **Answer:**
    - **Easiest:** The easiest fixes were removing the **eval() call** and the **unused import logging**. These were simple one-line deletions that instantly improved security and code cleanliness.
    - **Hardest:** The hardest fix was **refactoring the code to remove the global stock_data variable**. This was a structural change that required modifying the function signature of almost every function to pass the inventory dictionary as a parameter.

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

- **Answer:**
    - No, the tools did not report any significant false positives. All the high-priority warnings (like eval(), broad-except, and mutable-default-arg) pointed to real bugs or security risks. Even the low-priority style warnings (like invalid-name for camelCase) were technically correct according to the PEP 8 standard.

How would you integrate static analysis tools into your actual software development workflow? Consider continuous integration (CI) or local development practices.

- **Answer:**

- I would integrate them in two key places:
    1. **Local Development:** As a plugin for my code editor (like VS Code) to get real-time linting and error-checking as I type.
    2. **Continuous Integration (CI):** As a required step in a CI/CD pipeline (like GitHub Actions). This would automatically run Pylint, Bandit, and Flake8 on every git push and block any code that fails the quality checks from being merged.

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

- **Answer:**
    - **Robustness:** The code is far more robust. Removing the global variable makes the program's state predictable, and replacing broad-except with specific error handling prevents silent bugs and crashes.
    - **Readability:** The code is much easier to read. The new snake_case naming is consistent, and the comprehensive docstrings explain what each function does, its arguments, and its return value.
    - **Security:** The most critical improvement was removing the eval() function, which eliminated a major security vulnerability.