SOFTWARE ENGINEERING
LAB – WEEK 5

Name: Navyata Venkatesh

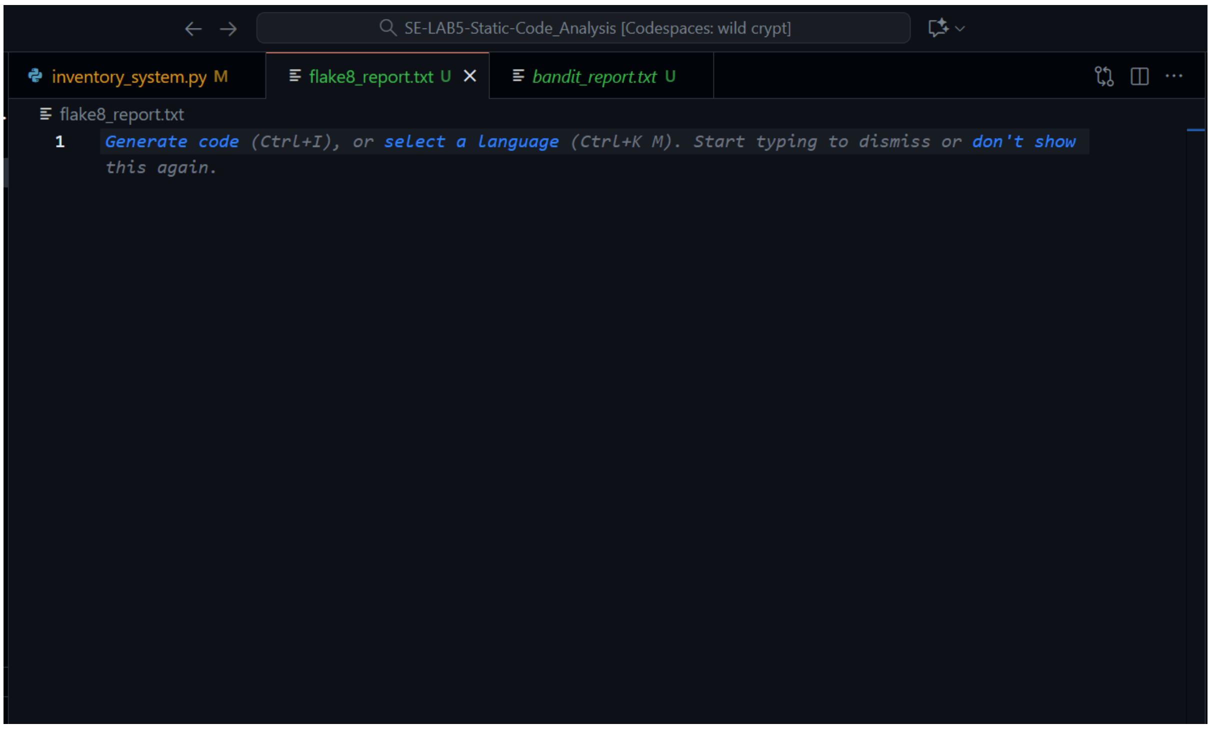
SRN: PES2UG23CS375

Section: F

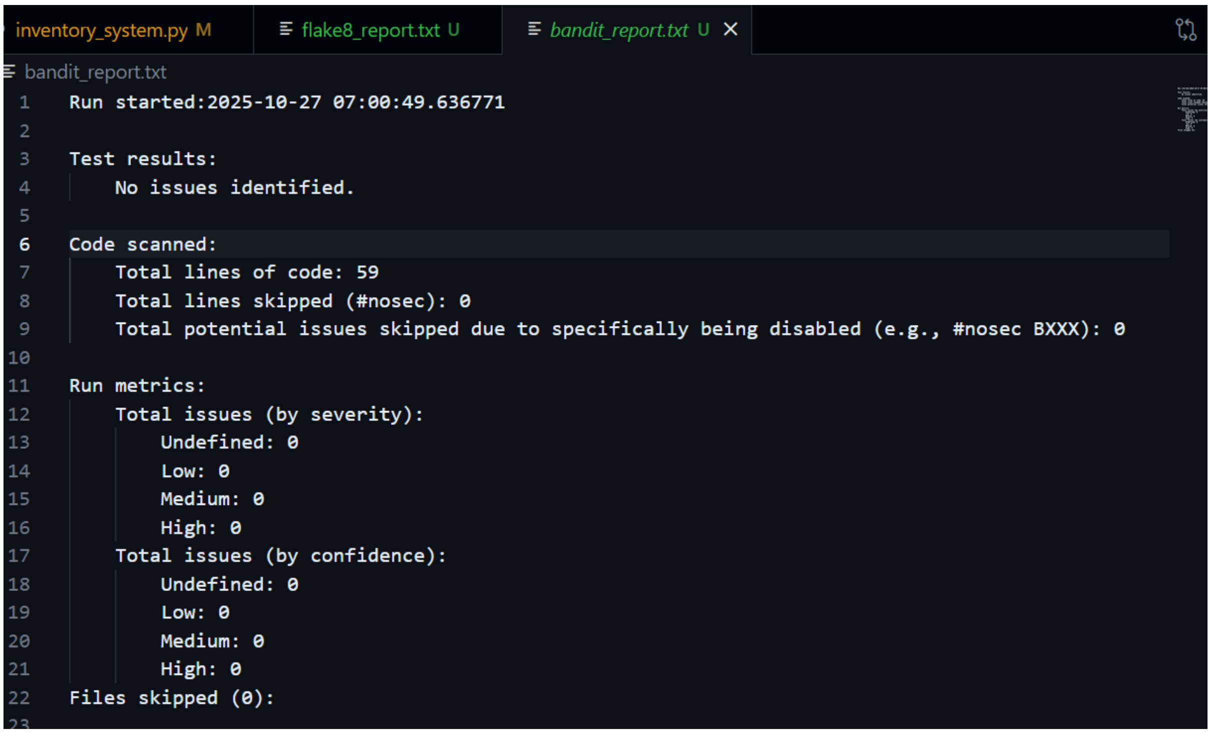
Issue	Type	Line(s)	Description	Fix Approach
Mutable default argument	Bug	8	Function uses [] as default parameter	Change default to None and initialize inside function.
Bare except	Bug	19	Using pass to catch all exceptions	Use specific exceptions like- except KeyError
Use of eval()	Secu rity	59	eval() can execute arbitrary code	Remove eval() and replace with alternatives like dictionary mapping or with ast.literal_eval()
Missing with for file operations	Best Pract ice	28,34	The files opened without context manager or encoding	Open the file using open() function
Unused import	Clea nup	2	Import present int code but not used	Remove unused imports
Function name not in snake case	Style	8,14,22 ,25,31, 36,41,4 8	Function names do not follow good practice naming conventions	Rename the function names with proper naming conventions

Using manual open() and close()	Code Quality	26,32	Using manual open() cold cause leaks if exceptions occur	Replace manual open() with “with open() as f”
String formatting	Style	16	Old % formatting used	Use F-string instead
Re-mentioning Global	Warning	37	Re-mentioning a global variable as global variable again	Remove the line
Removing extra spaces	style	37	Extra spaces around operators	Remove the extra spaces
Use. items() instead	Code quality	53,60	Loop uses indices	Replace with stock_data.items()
Requires input validation	reliability	9	Invalid inputs when given could cause runtime errors	Correct the code such that invalid input is not allowed
Missing extra space between function definitions	Style	(before every function)	Not proper spacing between functions	Insert 2 blank lines between the functions

Screenshots:



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

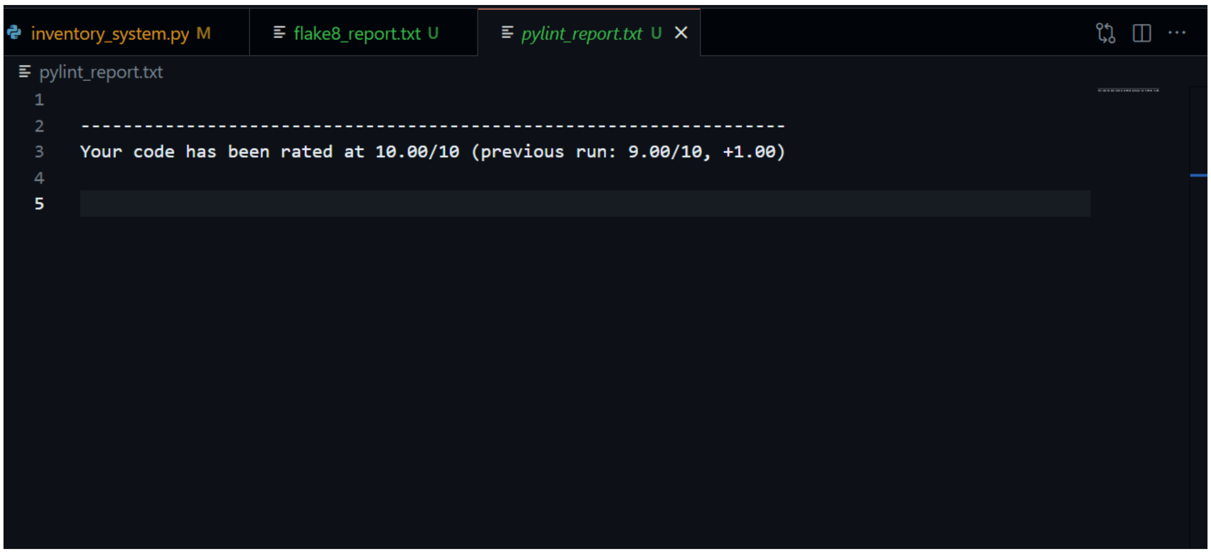


```
Run started:2025-10-27 07:00:49.636771

Test results:
    No issues identified.

Code scanned:
    Total lines of code: 59
    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):
```



```
-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.00/10, +1.00)
```

Fixed Code:

```
'''Inventory Management System'''
import json
from datetime import datetime


# Global variable
stock_data = {}


def add_item(item="default", qty=0, logs=None):
    ''' Add an item to inventory and increase its quantity'''
    if logs is None:
        logs = []
    if not isinstance(item, str) or not isinstance(qty, int):
        print("Invalid item name or quantity type.")
        return
    stock_data[item] = stock_data.get(item, 0) + qty
    logs.append(f"{datetime.now()}: Added {qty} of {item}")
```

```
def remove_item(item, qty):
    '''Remove the required item and decrease its quantity'''
    try:
        stock_data[item] -= qty
        if stock_data[item] <= 0:
            del stock_data[item]
    except KeyError:
        pass

def get_qty(item):
    '''Print quantity of a particular item'''
    return stock_data[item]

def load_data(file="inventory.json"):
    '''Loading the json file'''
    with open(file, "r", encoding="utf-8") as f:
        data = json.load(f)
    stock_data.update(data)

def save_data(file="inventory.json"):
    '''Saving the json file with changes'''
    with open(file, "w", encoding="utf-8") as f:
        json.dump(stock_data, f)
```

```
def print_data():
    '''Printing a report of the inventory'''
    print("Items Report")
    for item, qty in stock_data.items():
        print(item, "->", qty)

def check_low_items(threshold=5):
    '''Checking which items are below threshold and displaying them'''
    result = []
    for item, qty in stock_data.items():
        if qty < threshold:
            result.append(item)


    return result

def main():
    '''Main Function'''
    add_item("apple", 10)
    add_item("banana", -2)
    add_item(123, "ten")
    remove_item("apple", 3)
    remove_item("orange", 1)
    print("Apple stock:", get_qty("apple"))
    print("Low items:", check_low_items())
    save_data()
    load_data()
    print_data()
    print("demo completed")

main()
```

Reflections:

1. The easiest issues to fix were the formatting and whitespace errors reported by Flake8, such as missing blank lines, unnecessary spaces, and trailing whitespace. This was because they are just styling errors. The hardest issues were related to using the eval() function and improper file handling, since they required changed the working of the code.

2. Yes, there was a minor false positive from Pylint suggesting that the variable used inside an exception block was unused but this variable was intentionally kept for debugging but wasn't used later, so the warning didn't indicate an actual problem. Hence to avoid this, the variable was also removed.

3. I would integrate static analysis tools like Pylint, Flake8, and Bandit into a Continuous Integration (CI) pipeline so that every commit is automatically checked for code quality and security issues. I would also run these tools before committing changes in my local IDE to catch basic syntax, formatting, or security problems early. This improves code quality overall.

4. After applying the fixes, the code became cleaner, more readable, and easier to maintain. Functions were better structured with proper spacing, consistent naming, and documentation strings. Security and reliability also improved especially after removing the use of eval() and using safer file-handling practices. Overall, the static analysis process helped the code align more closely with Python’s best practices and standards.