Lab 5: Static Code Analysis

Objective

To enhance Python code quality, security, and style by utilizing **static analysis tools** (Pylint, Bandit, and Flake8) to detect and rectify common programming issues.

Duration:

90 minutes

Software Requirements

- **GitHub Account**
- **Web Browser** (for GitHub Codespaces)
- **inventory_system.py** (provided file, which will be within your Codespaces environment)

Learning Outcomes

By the end of this lab, you will be able to:

- **Understand the purpose and scope of static code analysis.**
- **Identify and resolve common Python programming issues**, including mutable default arguments, overly broad exceptions, and insecure coding patterns.
- **Effectively use Pylint, Bandit, and Flake8** to analyze Python code.
- **Interpret static analysis reports** and apply fixes based on issue severity.
- **Reflect on the contribution of static analysis** to overall code quality and maintainability.

Note: Students may be randomly called for a presentation after completing the lab. Please be prepared to discuss your code, bugs you found, and how you fixed them.

Introduction

In modern software development, ensuring code quality, security, and maintainability is paramount. **Static code analysis** is a crucial technique that involves examining source code without executing it to find potential bugs, vulnerabilities, and style violations. This lab will guide you through using three industry-standard Python static analysis tools:

- **Pylint**: Your "strict code reviewer" for **code quality and logical errors**. It flags issues like unused variables, poor practices, and potential design flaws.
- **Flake8**: Your "grammar checker" for Python code, enforcing **PEP 8 style guidelines**. It combines checks for formatting, whitespace, line length, and syntax issues.
- **Bandit**: Your "app's security guard" for Python, identifying **common security vulnerabilities**. It detects dangerous functions and insecure coding patterns.

You'll perform this lab entirely within **GitHub Codespaces**, a cloud-based development environment that provides all necessary tools without local setup.

Steps:

Option1: Step 1: Setup Your GitHub Codespaces Environment (15 minutes) (Not mandatory)

1. **Access the Lab Repository:**
    - Navigate to the GitHub repository provided for this lab (your instructor will provide this link).
    - Click the green **"Code"** button.
    - Select the **"Codespaces"** tab.
    - Click **"Create codespace on main"** (or similar option if a specific branch is indicated).
    - Wait for your Codespace to initialize. This may take a minute or two as it sets up a virtual development environment for you.
2. **Inspect `inventory_system.py`:**
    - Once your Codespace loads, you'll see a file explorer on the left. Locate and open `inventory_system.py`.
    - Familiarize yourself with the code's structure and functionality.
    - Run the program from the terminal within Codespaces to observe its initial behavior: `python inventory_system.py`

3. **Install Static Analysis Tools:**
    - The Codespace environment should have `pip` pre-installed. In the Codespace terminal, install the required tools:
    - `pip install flake8 bandit pylint`

Option 2:

Step 1: Create a new repository on github.
Step 2: Clone the repository locally and start working on the local machine.

Step 3: Install all the necessary tools.

```
pip install flake8 bandit pylint
```

Step 4: Run the program from the terminal within Codespaces to observe its initial behavior:
`python inventory_system.py`

Confirm Tool Installation:

- Verify that each tool is correctly installed by checking its version:

```
pylint --version

bandit --version

flake8 --version
```

Step 2: Run Static Analysis Tools (20 minutes)

Now that your environment is set up, run each analysis tool against `inventory_system.py`. The output will be redirected to text files for easier review.

1. **Run Pylint:** `pylint inventory_system.py > pylint_report.txt`
2. **Run Bandit:** `bandit -r inventory_system.py > bandit_report.txt`
3. **Run Flake8:** `flake8 inventory_system.py > flake8_report.txt`
4. You should now see `pylint_report.txt`, `bandit_report.txt`, and `flake8_report.txt` in your file explorer. Open them to view the reports.

Step 3: Identify and Document Issues (15 minutes)

Review the generated reports and cross-reference them with the common issues outlined for each tool. Use the provided table structure to document your findings. You may need to refer to the official documentation for Pylint, Bandit, and Flake8 if you need more context on specific error codes or warnings.

Example Known Issue Table:

Issue	Type	Line(s)	Description	Fix Approach
Mutable default arg	Bug	12	`logs=[]` shared across calls	Change default to `None` and initialize in method
... (Your findings)	...	...	...	...

Step 4: Fix Issues (30 minutes)

Based on your documented issues, begin fixing the problems in inventory_system.py.

1. **Prioritize:** Focus on high and medium severity issues first.
2. **Fix at least 4 issues** from your identified list.
3. **Suggested fixes to consider:**
    - Replace except: with specific exception types.
    - Use **f-strings** for cleaner logging/output.
    - Implement **input validation** for functions that take user input.
    - Properly configure logging in the main execution block.
4. **Verify Fixes:** After making changes, **rerun the static analysis tools** (as in Step 2) and check the new reports (pylint_report.txt, bandit_report.txt, flake8_report.txt) to confirm that the issues you've addressed no longer appear. Repeat this process until your fixes are verified.

- **Pro Tip:** An extra 2 marks will be awarded for fixing ALL issues reported by the tools!

Reflection (10 minutes)

Answer the following questions concisely (using paragraphs or bullet points) based on your experience during this lab. Add your answers to a new file named reflection.md in your Codespace.

1. **Which issues were the easiest to fix, and which were the hardest? Why?**
2. **Did the static analysis tools report any false positives?** If so, describe one example.
3. **How would you integrate static analysis tools into your actual software development workflow?** Consider continuous integration (CI) or local development practices.
4. **What tangible improvements did you observe in the code quality, readability, or potential robustness after applying the fixes?**

Deliverables:

Submission: Github repository link .

- **The repository should have 6 files in total.**
    - **Given inventory (inventory_system.py) file**
    - **Updated clean inventory file (ex: cleaned_inventory_system.py)**
    - **All the 3 reports generated wrt to the updated clean inventory file.**
    - **One readme.md file.**
        - **The readme.md file should contain complete known issue table, reflection questions and answers.**