Unit 6 Seminar

Title: Exploring Linters to Support Testing in Python

Questions:

The following questions will be discussed during this week's seminar. These questions are provided in the '**testing-with-python**' zip file with instructions provided in the '**Secure Software Development**' PDF file. These activities should be completed on your chosen Jupyter Notebook workspace.

Question 1

Run styleLint.py.

What happens when the code is run? Can you modify this code for a more favourable outcome? What amendments have you made to the code?

Question 2

pip install pylint

Run

pylint

on pylintTest.py

Review each of the code errors returned. Can you correct each of the errors identified by pylint?

Before correcting the code errors, save the pylintTest.py file with a new name (it will be needed again in the next question).

Question 3

pip install flake8

Run

flake8

on pylintTest.py

Review the errors returned. In what way does this error message differ from the error message returned by pylint?

Run flake8 on metricTest.py. Can you correct each of the errors returned by flake8? What amendments have you made to the code?

Question 4

pip install mccabe

Run

mccabe

on sums.py. What is the result?

Run

mccabe

on sums2.py. What is the result?

What are the contributors to the cyclomatic complexity in each piece of code?

Activity

Select one or more of the tools installed above and use it/them to test the code your team has created as part of the summative assessment. You should demonstrate your tests (and share your results) during the seminar. Discuss the need to change the code based on the output from the test tools/linters.

You will also have the opportunity to discuss your team's progress during the seminar.

Response:

Question 1: Run styleLint.py

The code for the factorial function in styleLint.py has an indentation issue, and running it may not work as expected. The docstring is not properly aligned, and the function lacks appropriate spacing, which could result in an error or unfavorable formatting.

Amendments to the code:

Corrected Code:

```
def factorial(n):
    """Return the factorial of n."""
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Modifications made:

- The docstring has been properly aligned and formatted.
- Improved the readability by adding spaces around operators.

- The function will now work correctly when called.

Question 2: Run Pylint on pylintTest.py

1. **Errors returned by Pylint:** After running Pylint on the given pylintTest.py, errors include:

    - **Deprecation**: raw_input is used instead of input. Since the code was written for Python 2, it needs to be updated for Python 3.
    - **Indentation issues**: Indentation and formatting inconsistencies.
    - **Unused variable warnings** and **improper naming conventions**.

    Corrected Code:

```
import string

shift = 3
choice = input("Would you like to encode or decode? ")  # Fixed raw_input to input
word = input("Please enter text: ")  # Fixed raw_input to input
letters = string.ascii_letters + string.punctuation + string.digits
encoded = ''

if choice == "encode":
    for letter in word:
        if letter == ' ':
            encoded += ' '
        else:
            x = letters.index(letter) + shift
            encoded += letters[x % len(letters)]  # Prevent index overflow
elif choice == "decode":  # Fixed indentation and conditional
    for letter in word:
        if letter == ' ':
            encoded += ' '
        else:
            x = letters.index(letter) - shift
            encoded += letters[x % len(letters)]

print(encoded)

```

Changes:

- Fixed indentation.
- Updated raw_input to input for Python 3.
- Improved handling of string indexing to avoid index overflow.
- Moved the second condition (decode) out of the inner loop for proper flow.

Question 3: Run Flake8

1. **Flake8 on pylintTest.py:** Running Flake8 on pylintTest.py returns errors related to PEP8 compliance such as:

    - Line length exceeding 79 characters.
    - Improper spacing around operators.
    - Missing docstrings.

2. **Flake8** is more focused on style and formatting compared to Pylint, which checks for both style and logical errors.

3. **Flake8 on metricTest.py:** Running Flake8 on metricTest.py shows similar PEP8-related issues such as:

    - Improper variable naming.
    - Long lines without breaks.
    - Missing docstrings in some functions.

4. **Amendments made**:

    - Ensure proper indentation.
    - Break long lines into shorter ones.
    - Add missing docstrings for clarity.

Question 4: Run McCabe on sums.py and sums2.py

1. **McCabe on sums.py**: Cyclomatic complexity returned by McCabe would typically be 1 because there is only a single path (no branches or conditionals).

2. **McCabe on sums2.py**: McCabe will return a higher cyclomatic complexity for sums2.py because of the use of conditionals (if-else statements), which contribute to multiple independent paths in the code. The result might be higher than 1, depending on the number of conditional branches.

Activity: Testing Team's Code with Linters

Final Discussion: Based on the Output

- **Code Changes Based on Output**:

    - **Pylint** may suggest improving the logic, structure, and docstring coverage.
    - **Flake8** will encourage PEP8 compliance for readability.
    - **McCabe** will highlight areas of high cyclomatic complexity that may need to be simplified for better maintainability and security.

- **How the Changes Improve Security**:
    - Fixing unused variables, improving code readability, and reducing complexity will make the code easier to maintain, audit, and debug, reducing the risk of security vulnerabilities.
    - Simplifying complex code (based on McCabe's results) makes it less error-prone and easier to test, helping prevent logical flaws that attackers could exploit.