Python Code Quality Tools

Introduction

Code quality refers to the overall quality of the code. It takes into consideration things like:

- Readability
- Maintainability
- Efficiency
- Adherence to best practices

Why do we focus on quality?

- Code management (reducing bugs and errors)
- Facilitate collaboration with other team members
- Easing maintenance
- Easing future updates
- Improve reliability
- Improve software and team performance

Consequences of poor code quality

- Technical debt
- Increased development costs
- Decreased productivity over time

How do we create coding standards?

- They are agreed by a team or community
- They are chosen because a group/community agrees that they lead to more readable and maintainable code

Examples of Coding Standards

- Consistent naming conventions
- Proper code organization and modularization
- Effective use of comments and documentation
- Following the Don’t Repeat Yourself (DRY) principle
- Writing testable code

PEP - Python Enhancement Proposals

PEPs are design documents that:

- Provide information to the Python community
- Describe new features for Python or its processes
- Serve as announcement tools for proposing new features
- Facilitate communication within the Python community
- Document decisions

PEP 8: Style Guide for Python Code

PEP 8 provides coding conventions for the Python standard library.

- One of the most popular coding standards
- Many projects follow PEP 8 to maintain consistency across the Python ecosystem

PEP 8: Key Points

- Indentation: 4 spaces
- Maximum line length:
  - 79 characters for code
  - 72 for comments and docstrings

Naming conventions:

- Functions, variables, attributes: `lowercase_with_underscores`
- Classes: `CapitalizedWords`
- Constants: `ALL_CAPS_WITH_UNDERSCORES`

PEP 8: Coding Structure

- Imports: Separate lines, grouped (standard library, third-party, local)
- Whitespace: Blank lines to separate functions, classes, and larger blocks
- Comments:
  - Use inline comments sparingly
  - Write docstrings for all public modules, functions, classes, and methods

Linters and Their Importance

A linter is a tool that analyzes source code. It’s like Grammarly for code.

- Detects programming errors, bugs, and stylistic errors
- Does not **execute** the code

What Linters Find

- Syntax errors
- Unused variables or imports
- Violations of coding standards
- Potential logical errors or bugs
- Security vulnerabilities

Why Use Linters?

- Early detection of errors
- Consistency
- Time saver
- Code quality improvement
- Integration with CI/CD
- Help developers learn best practices and improve coding skills

Popular Python Linters

Pylint

```
pip install pylint
pylint your_file.py
```

Black

```
pip install black
black your_file.py
```

Ruff

```
pip install ruff
ruff check your_file.py
```

Comparing Pylint, Black, and Ruff

- Pylint: Comprehensive linter, checks style and logic
- Black: Opinionated code formatter, focuses on consistent style
- Ruff: Fast linter, combines multiple tools, customizable

Additional Resources

- PEP 8 – Style Guide for Python Code
- Pylint Documentation
- Black Documentation
- Ruff Documentation

Thank You!

Any questions?