PIZENITH

SOFTWARE DEVELOPMENT BEST PRACTICES

Jan 2025

Version 1.0

Python: Best Practice with Code Checklist

Prepared For

Public

Prepared By

Pizenith Technologies LLP

Jan 20, 2025

General Use Case

INDEX

Best Practices

Best practices in coding refer to a set of guidelines and techniques that aim to improve code quality, maintainability, and efficiency. Here are some widely followed best practices across programming languages:

1. **Write Readable Code:**
    - Write code that is easy to understand and follow. Use meaningful variable and function names, and write comments to explain what the code does.

2. **Use Version Control:**
    - Use version control systems, such as Git, to manage changes made to code over time. This allows you to revert to a previous version of the code if needed and makes it easier to collaborate with others.

3. **Write Unit Tests:**
    - Write automated tests to validate the behavior of your code. This helps to catch bugs early in the development process and makes it easier to maintain the code over time.

4. **Follow Coding Standards:**
    - Follow coding standards, such as the PEP 8 Style Guide for Python or the Google Java Style Guide, to write consistent and readable code.

5. **Refactor Code Regularly:**
    - Regularly review and refactor your code to remove redundant or complex code. This helps to improve code readability and maintainability and makes it easier to add new features.

6. **Document Code:**

- Document your code by writing comments and documentation. This makes it easier for others to understand the code and for you to remember how it works in the future.

7. **Use Error Handling:**
    - Use error handling techniques, such as exceptions, to handle errors and unexpected conditions in your code.

8. **Avoid Hard-Coding Values:**
    - Avoid hard-coding values, such as file paths or API keys, in your code. Instead, use configuration files or environment variables to store these values.

9. **Use Appropriate Data Structures:**
    - Use appropriate data structures, such as arrays, lists, or dictionaries, to store and manipulate data. This helps to improve code performance and readability.

10. **Keep Code Modular:**
    - Write code that is modular, meaning it is divided into small, self-contained units. This makes it easier to maintain and reuse code and makes it easier to test individual parts of the code.

11. **Use Named Tuples Instead of Dictionaries:**
    - Named tuples are a lightweight alternative to dictionaries and provide a way to store data in a named, ordered format. They are faster than dictionaries and use less memory, making them a good choice for data-intensive applications.

12. **Avoid Global Variables:**
    - Global variables can make code difficult to maintain and test, as changes to one part of the code can affect other parts. Instead, use function arguments and return values to pass data between parts of the code.

13. **Use List Comprehensions:**

- List comprehension provides a concise and readable way to create lists and are faster than traditional loops. Use them instead of loops whenever possible.

14. Use Generators:

- Generators are a way to produce a sequence of values one at a time, on demand. They are more memory-efficient than lists, as they only store the current value, not the entire sequence.

15. Use Virtual Environments:

- Virtual environments provide a way to isolate the dependencies of your Python project from other projects. This makes it easier to manage dependencies and ensures that your code will run the same way on different machines.

16. Use SOLID Principles:

- The SOLID principles are guidelines for writing maintainable and scalable software. They aim to reduce dependencies so that engineers can change one area of software without impacting others. These principles make designs easier to understand, maintain, and extend.

Code Quality Tools

Code quality tools are used to analyze and improve the quality of code in software development. Here are some of the most popular code quality tools in Python:

Pylint

- A static code analysis tool that checks for errors, coding standard violations, and best practices. It provides a detailed report that includes information about code complexity, performance, security issues, and more.

Flake8

- A linter that combines the functionality of Pyflakes, Pycodestyle, and the McCabe complexity checker. It analyzes code for coding standards, syntax errors, and other Issues.

Bandit

- A security-focused code analysis tool that checks for common security issues, such as cross-site scripting (XSS) vulnerabilities, SQL injection, and hardcoded secrets. It supports custom plugins, making it a powerful tool for checking code quality and security in Python applications.

Pre-commit

- A Python-based software development tool that allows you to manage and maintain your project's Git hooks. Git hooks are scripts that run automatically before or after certain Git events, such as committing code or pushing to a remote repository. With pre-commit, you can define a set of hooks for your project, making it easier to enforce best practices and keep your codebase consistent.
- More info: [Pre-commit](https://pre-commit.com/#plugins)

Black

- A popular Python code formatting tool that enforces a strict and uniform code style by automatically reformatting code to follow the PEP 8 style guide. It's fast, uncomplicated, and highly opinionated, promoting consistency and reducing manual intervention.Isort
- A Python library for sorting imports in a standardized and customizable way, improving the readability and maintainability of code. It can be combined with other code quality tools and integrated into development environments.

Testing Libraries and Methods

Python has a rich ecosystem of testing libraries and tools that make it easier to write and run tests for your code. Here are some popular Python testing libraries and methods:

Unittest

- A built-in testing library in Python that provides a framework for writing and running tests. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

Pytest

- A popular third-party testing library designed to make it easier to write and run tests. It has a simple and intuitive API, supports fixtures and plugins, and features like parallel testing and detailed test reporting.

Assert Statement

- The assert statement is a simple way to write tests in Python. You can use the assert statement to check that an expression is true and raise an exception if it is not. This can be used to write simple tests or as a starting point for more complex testing frameworks.

Code Review Checklist

- Are all new packages used included in `requirements.txt`?
- Does the code pass all lint checks?
- Do functions use type hints, and are there any type hint errors?
- Is the code readable and does it use Pythonic constructs wherever possible?

Sample Code Repository

The repository below is available on Pizenith Public Github and contains a multi-

module Python project with best practices and standards implemented. Steps to run and explore are present in the readme.

- Repo Link
(https://github.com/pizenith-technologies/python_best_practices)