**Code Structure**

**a. Modular Code**

* Break your code into modules and packages.
* Each module should have a clear responsibility.
* Avoid large, monolithic scripts.

**b. Naming Conventions**

* Use **snake\_case** for variables, functions, and method names.
* Use **PascalCase** for class names.
* Constants should be in **ALL\_CAPS**.
* Make names meaningful and descriptive (e.g., calculate\_total is better than calc).

**Code Formatting**

**a. PEP 8**

* Follow the [PEP 8](https://www.python.org/dev/peps/pep-0008/) style guide.
* Use an IDE or tool like flake8 or black to enforce style standards.

**b. Indentation**

* Use 4 spaces per indentation level, not tabs.

**c. Line Length**

* Limit all lines to a maximum of 79 characters. For docstrings and comments, limit lines to 72 characters.

**d. Blank Lines**

* Use blank lines to separate functions and class definitions, and to organize code into logical sections.
* Two blank lines between top-level functions and classes.

**Comments and Documentation**

**a. Code Comments**

* Write comments to explain **why** something is done, not **what** the code does.
* Use inline comments sparingly.
* Use block comments to explain more complex code.

**b. Docstrings**

* Write docstrings for all public modules, functions, classes, and methods.
* Follow [PEP 257](https://www.python.org/dev/peps/pep-0257/) for docstring conventions.

**4. Error Handling**

**a. Exceptions**

* Use exceptions to handle errors, not return values like -1 or None.
* Catch specific exceptions rather than using a generic except block.

**Security**

**a. Input Validation**

* Validate user inputs and avoid trusting external data directly.
* Use regular expressions or validation libraries to sanitize inputs.

**b. Avoid Hardcoding Secrets**

* Never hardcode sensitive information like API keys, passwords, or secrets in the codebase.
* Use environment variables or secret management tools like AWS Secrets Manager.

**Performance Considerations**

**a. Efficient Data Structures**

* Choose appropriate data structures (e.g., lists, sets, dictionaries) for better performance.
* Avoid unnecessary complexity or over-engineering.

**b. Profiling**

* Use profiling tools like cProfile or timeit to identify performance bottlenecks.

**c. Lazy Loading**

* Use lazy loading and generators when working with large data sets to optimize memory usage.