1. Code Layout:
    - Consistent Indentation: Use a consistent and clear indentation style (e.g., spaces or tabs) to make the code visually organized and easy to read.
    - Organized Directory Structure: Keep your project organized with a logical directory structure. Group related code files into directories, such as "controllers," "models," "views," and "utilities."
    - Clear Naming Conventions: Use meaningful and descriptive names for variables, functions, and classes. Follow a naming convention like camelCase, snake_case, or PascalCase for consistency.
    - Commenting: Add comments to explain the purpose of functions, classes, and complex sections of code. Use docstrings for functions and classes to provide documentation.
    - Consistent Code Style: Follow a consistent code style guide or linting rules (e.g., PEP 8 for Python) to maintain uniformity in your code.
2. Readability:
    - Modularization: Break down your code into small, reusable modules or functions. Each function should have a single responsibility, making it easier to understand.
    - Avoid Magic Numbers and Strings: Replace magic numbers and strings with named constants or variables with descriptive names to improve code readability.
    - Use White Space: Use blank lines and spaces to separate code blocks, making it easier to follow the logical flow of your code.
    - Error Handling: Implement clear and informative error handling with descriptive error messages to help identify issues quickly.
    - Avoid Deep Nesting: Limit the level of nesting in your code to prevent excessive indentation, which can make code harder to read.
3. Reusability:
    - DRY (Don't Repeat Yourself): Identify common patterns in your code and refactor them into reusable functions or classes. This minimizes redundancy and makes maintenance easier.
    - Create Utility Functions: Extract commonly used code snippets into utility functions that can be imported and reused throughout the application.
    - Separate Concerns: Use the concept of separation of concerns to keep different aspects of your application (e.g., data access, business logic, and presentation) in separate modules or classes.
    - Implement Interfaces and Abstract Classes: In object-oriented languages, use interfaces and abstract classes to define contracts that can be implemented or extended by multiple classes.

- Dependency Injection: Use dependency injection to provide components with their dependencies, making it easier to swap out implementations and improve testability.