Python Coding Guidelines

1. FORMATTING & INDENTATION

- Use 4 spaces per indentation level.
- Keep lines under 79 characters (PEP8 standard).
- Use blank lines to separate functions, classes, and logical sections.
- Avoid unnecessary whitespace inside parentheses, brackets, or braces.

2. NAMING CONVENTIONS

- Variables & functions: lowercase_with_underscores
- Classes: CapitalizedWords (PascalCase)
- Constants: UPPER_CASE_WITH_UNDERSCORES
- Modules and packages: short, all-lowercase names
- Avoid single-character names (except for counters like i, j).

3. CODE STRUCTURE & ORGANIZATION

- Each file should have a clear purpose (module-level cohesion).
- Imports at the top, one per line, grouped logically:
  1. Standard libraries
  2. Third-party libraries
  3. Local application imports
- Use __main__ guard for executable scripts:

```
if __name__ == "__main__": main()
```

4. PROGRAMMING PRACTICES

- Use meaningful variable names (avoid x, temp, data unless obvious).
- Avoid hard-coded values; use constants or config files.

- Use functions and classes to encapsulate logic.
- Avoid deep nesting; use early returns instead.
- Handle exceptions gracefully using try-except-finally.
- Add docstrings for all modules, classes, and methods.
- Keep functions small and focused (Single Responsibility Principle).
- Use comments only when necessary - prefer self-explanatory code.

5. DOCUMENTATION & COMMENTS

- Use triple quotes for docstrings (PEP257).
- Include parameters, return types, and brief descriptions.
- Example:

```
def add(a, b):
    """Return the sum of two numbers."""
    return a + b
```

- Write inline comments sparingly; they should explain 'why', not 'what'.

6. VERSION CONTROL & STYLE CONSISTENCY

- Follow PEP8 for Python style guidelines.
- Use tools like pylint, flake8, or black for formatting.
- Commit frequently with meaningful messages.
- Avoid committing commented-out code or print statements.

7. SECURITY & PERFORMANCE

- Validate all user inputs.
- Close files and database connections properly.
- Avoid using eval() and exec() unless absolutely necessary.
- Optimize algorithms and use built-in functions when possible.
- Use logging instead of print() for tracking runtime behavior.