General Style

Indentation: Use 4 spaces per indentation level.

Maximum Line Length: Limit all lines to a maximum of 79 characters.

Blank Lines: Use blank lines to separate functions and classes, and within functions to indicate logical sections.

Imports:

- Imports should usually be on separate lines.
- Group imports in the following order: standard library imports, related third-party imports, local application/library specific imports.

Naming Convention

Variables: Use lowercase words separated by underscores (snake_case).

Constants: Use uppercase words separated by underscores (UPPERCASE).

Classes: Use the CapWords convention (also known as CamelCase).

Functions/Methods: Use lowercase words separated by underscores (snake_case).

Modules/Packages: Use short, lowercase names. Underscores can be used if it improves readability.

Private Members: Use a single leading underscore (_) to indicate a private member.

Variable Definitions

Global Variables: Avoid using global variables so as not to have functions interfere with one another.

Object Variables: Define variables that belong to every instance of an object as an object method.

Docstrings and Comments

Docstrings: Use triple quotes (""") for docstrings. Every module, class, and function should have a docstring describing what it does.

```
def function_name(parameters):
    """
    Brief description of the function.

    Longer explanation if necessary.

    Args:
        parameters (type): Description of parameters.

    Returns:
        type: Description of return value.
    """
    pass
```

Comments: Use comments to explain why something is done, not how. Place comments on their own line if possible and use a single # followed by a space.

```
# This is a comment
x = x + 1  # Increment x
```

Code Layout

Spacing: Use spaces around operators and after commas. Do not use spaces inside parentheses, brackets, or braces.

```
a = (b + c) * (d - e)
```

Blocks: Use consistent and clear block structures. Always use colons (:) after statements that introduce an indented block.

Error Handling

Exceptions: Use exceptions for error handling. Always use specific exceptions rather than a generic Exception. Provide informative messages.

```
try:
   # Code that may raise an exception
except SpecificException as e:
   # Handle the exception
   print(f"Error: {e}")
```

Sources

PEP 8 – Style Guide for Python Code