Summary of Google's Python Style Guide

The Google Python Style Guide (PyGuide) outlines best practices for writing clean, readable, and maintainable Python code, primarily for internal use at Google but applicable broadly. It emphasizes consistency, simplicity, and tools like pylint for enforcement. The guide is divided into **Language Rules** (semantics and idioms) and **Style Rules** (formatting and conventions). Below is a structured summary of key sections.

1. Background

- Python is Google's primary dynamic language.
- Use auto-formatters (e.g., Black or Pyink) and a provided Vim settings file to resolve formatting debates.
- Focus on "dos and don'ts" for programs, assuming Python 3.6+ compatibility.

2. Python Language Rules

These cover semantic choices and advanced features.

- **Linting**: Run pylint on all code with Google's pylintrc. Suppress warnings sparingly with # pylint: disable=.... Delete unused arguments and explain why (e.g., # Unused by vikings.). Avoid _ prefixes for unused vars to preserve named argument usability.
- **Imports**: Import modules/packages, not individuals. Use import x for modules, from x import y for specifics, and aliases (as z) only for conflicts, long names, or standards (e.g., numpy as np). Avoid relative imports; use absolute paths. Group and sort imports: future, standard lib, third-party, local.
- **Packages**: Use full pathnames (e.g., from doctor.who import jodie) to avoid sys.path ambiguities.
- **Exceptions**: Raise built-ins like ValueError. Custom exceptions end in "Error" and inherit appropriately. Avoid assert for runtime checks (use only in tests). Minimize try/except blocks; avoid bare except. Use finally for cleanup.
- **Mutable Global State**: Avoid it; use module-level constants (in CAPS_WITH_UNDER). Mark internals with _ and access via functions.
- **Nested/Local Classes & Functions**: Allowed for closures over locals (not self/cls). Use _ for private module functions instead of deep nesting.
- **Comprehensions & Generators**: Use for simple cases; prioritize readability over brevity. Avoid complex/nested ones. For generators, document "Yields:" and handle resources.
- **Default Iterators**: Prefer built-ins like for key in dict or if x in list.
- **Lambdas**: Limit to one-liners; prefer operator module or comprehensions over map/filter.
- **Conditional Expressions**: OK for simple ternaries; keep each part on one line.
- **Default Arguments**: Avoid mutable defaults (e.g., []); use None and initialize inside.
- **Properties**: For cheap, unsurprising access; avoid for overrides or passthroughs.
- **True/False Evaluations**: Use implicit booleans (e.g., if foo:); check None explicitly. Avoid == False.
- **Lexical Scoping**: Allowed, but watch for shadowing.
- **Decorators**: Use sparingly (e.g., @property); test them. Avoid staticmethod unless API-required.
- **Threading**: Use queue.Queue and threading.Condition; don't assume atomicity.
- **Power Features**: Avoid metaclasses, __del__, etc., for readability.

- **Modern Python**: Use __future__ imports for compatibility.
- **Type Annotations**: Required for public APIs and complex code; check with pytype. Prefer built-ins (e.g., list[str]) and from __future__ import annotations.

3. Python Style Rules

These focus on formatting and code structure.

- **Semicolons**: Never use to end lines or chain statements.
- **Line Length**: 80 characters max. Use parentheses for continuation; exceptions for URLs/imports.
- **Parentheses**: Minimal use; optional for tuples/line breaks.
- **Indentation**: 4 spaces only; align wraps vertically or with 4-space hangs. Use trailing commas for multi-line sequences.
- **Blank Lines**: 2 between top-level defs; 1 between methods/after docstrings.
- **Whitespace**: No inside (),[],{}; spaces around operators/after commas. No trailing spaces.
- **Shebang**: #!/usr/bin/env python3 for executables.
- **Comments & Docstrings**: Docstrings for modules/functions/classes (Google format: summary, Args, Returns, etc.). Block comments for complex logic; inline for clarity. Use TODO(username): for hacks.
- **Strings**: Prefer f-strings or format(); join() for loops. Use """ for multi-line with textwrap.dedent(). Precise, grep-friendly error messages.
- **Resources**: Always close files/sockets with with or contextlib.closing().
- **Imports Formatting**: Top of file, grouped/sorted (future > stdlib > third-party > local).
- **Statements**: One per line; no if/else or try/except chaining.
- **Getters/Setters**: get_foo()/set_foo() for complex access; prefer properties for simple.
- **Naming**: Descriptive, no abbreviations/types in names. Conventions:
    - Functions/vars/modules: lower_with_under
    - Classes/exceptions: CapWords
    - Constants: CAPS_WITH_UNDER
    - Private: _leading_
    - Files: lower_with_under.py
- **Main**: Use if __name__ == '__main__': main() for scripts.
- **Function Length**: Aim for ~40 lines; split if complex.

This guide promotes readable, testable code while leveraging Python's strengths. For full details, refer to the original document.

Source : https://google.github.io/styleguide/pyguide.html