Python 技术管理规范文档

文档目的：本规范旨在为Python项目提供一套统一、清晰、可维护的代码编写标准。通过遵循这些规范，团队可以提升代码可读性，减少潜在错误，降低协作与维护成本。

规范级别定义:

- **a. 强制 (Mandatory)**: 必须无条件遵守的规则。违反这些规则的代码将被视为不合格，不允许提交到主干分支。
- **b. 推荐 (Recommended)**: 强烈建议遵守的最佳实践。在有充分理由并经过团队同意的情况下可以有例外，但应尽可能遵循。
- **c. 允许 (Permitted)**: 提供灵活性，表示某些做法是被接受的，开发者可以根据具体场景自行选择。

---

1. 命名规范 (Naming Conventions)

级别	规则ID	规则描述
强制	N-1	变量、函数、方法名使用小写字母和下划线（`snake_case`）。示例：`user_name`，`calculate_sum()`。
强制	N-2	类名使用帕斯卡命名法（`PascalCase`）。示例：`class UserProfile:`，`class DatabaseConnection:`。
强制	N-3	常量名使用大写字母和下划线（`UPPER_SNAKE_CASE`）。示例：`MAX_CONNECTIONS = 10`。
强制	N-4	模块名应为简短、全小写的名称。如果需要，可以使用下划线。示例：`db_utils.py`。
强制	N-5	类的内部变量（不希望外部直接访问）以单个下划线开头。示例：`self._internal_data`。
推荐	N-6	避免使用单个字母作为变量名（除了在循环或数学表达式中，如`i`，`j`，`k`，`x`，`y`），应使用有意义的名称。
推荐	N-7	函数命名应体现其功能，最好是动词或动宾短语。示例：`get_user_info()`，`delete_record()`。

2. 代码格式化 (Code Formatting)

级别	规则ID	规则描述
强制	F-1	使用4个空格作为一级缩进。禁止使用Tab，或Tab与空格混用。
强制	F-2	在二元运算符两侧各留一个空格。示例：`x = y + 1`。
强制	F-3	逗号、分号、冒号后要留一个空格。示例：`my_list = [1, 2, 3]`。
推荐	F-4	每行代码的长度不应超过88个字符（与代码格式化工具`black`的默认配置保持一致）。
推荐	F-5	使用括号、方括号或花括号进行隐式行连接，而不是使用反斜杠`\`进行显式连接。
推荐	F-6	import语句应分行书写。示例：`import os`，`import sys`，而不是`import os, sys`。
推荐	F-7	import语句应按照“标准库、第三方库、本地应用”的顺序分组，组间用一个空行隔开。
推荐	F-8	在函数和类的定义之间使用两个空行。
推荐	F-9	在类中，方法之间使用一个空行。
允许	F-10	允许在不影响可读性的情况下，在列表、字典或函数参数的末尾添加一个多余的逗号（trailing comma）。这有助于版本控制。

3. 注释与文档字符串 (Comments & Docstrings)

级别	规则ID	规则描述
强制	C-1	所有公开的模块、函数、类和方法都必须有文档字符串（docstring）。
强制	C-2	文档字符串使用三个双引号 `"""Docstring goes here"""` 包围。
强制	C-3	文档字符串第一行应为该对象的简要概述，以句号结尾。
推荐	C-4	采用Google风格的文档字符串，清晰地描述参数（Args）、返回值（Returns）和可能抛出的异常（Raises）。
推荐	C-5	注释应言之有物，解释“为什么”这样做，而不是“做了什么”。代码本身应该清晰地说明“做了什么”。
推荐	C-6	对于复杂的代码块，在其前添加块注释（以 `#` 和一个空格开头）。
允许	C-7	允许在代码行后使用行内注释，但它应与代码至少隔开两个空格。

4. 语言特性与用法 (Language Features & Usage)

级别	规则ID	规则描述
强制	L-1	使用 `with` 语句处理文件、锁等需要明确关闭的资源。
强制	L-2	使用 `is` 或 `is not` 来比较单例对象，如 `None`, `True`, `False`。示例：`if my_var is None:`。
强制	L-3	捕获异常时，应指定具体的异常类型，而不是裸露的 `except:`。
强制	L-4	不要使用可变类型（如 `list`, `dict`）作为函数定义的默认参数。
推荐	L-5	推荐使用f-strings（`f"{var}"`）进行字符串格式化，它比 `%` 格式化和 `str.format()` 更简洁、高效。
推荐	L-6	对于所有新编写的代码，推荐使用类型提示（Type Hinting）。
推荐	L-7	优先使用列表推导式（List Comprehensions）来创建列表，而不是 `map()` 和 `filter()`，因为它通常更具可读性。
推荐	L-8	使用 `in` 关键字来检查成员是否存在。示例：`if item in my_list:`。
推荐	L-9	使用 `startswith()` 和 `endswith()` 来检查字符串的前缀和后缀，而不是进行切片比较。
推荐	L-10	使用 `dict.get()` 或 `collections.defaultdict` 来处理可能不存在的字典键。
允许	L-11	允许使用 `_` 作为丢弃不用的变量名。示例：`for _, value in my_dict.items():`。
允许	L-12	在确保性能不是瓶颈的情况下，允许为了代码的可读性而牺牲微小的性能。

5. 项目结构与依赖管理 (Project Structure & Dependency Management)

级别	规则ID	规则描述
强制	P-1	每个项目都必须使用独立的虚拟环境（如 venv, conda）。
强制	P-2	项目的所有依赖项必须在 `requirements.txt` 或 `pyproject.toml` 文件中明确声明。
强制	P-3	不要在版本控制系统中提交虚拟环境目录、`__pycache__` 目录或 `.pyc` 文件。使用 `.gitignore` 文件进行排除。
推荐	P-4	推荐使用 `src` 布局，将所有源代码放在一个 `src` 目录中。
推荐	P-5	配置文件（如数据库连接信息）应与代码分离，不要硬编码在代码中。

6. 测试 (Testing)

级别	规则ID	规则描述
强制	T-1	提交到主干分支的关键功能和Bug修复必须附带相应的单元测试。
推荐	T-2	推荐使用 `pytest` 作为测试框架。
推荐	T-3	测试代码应与源代码分开存放，通常放在项目根目录下的 `tests/` 目录中。

自动化工具：为了更好地执行以上规范，推荐在CI/CD流程中集成自动化工具，如 `black` 用于代码格式化，`flake8` 或 `pylint` 用于代码质量检查，`isort` 用于import排序。