# Python 技术管理文档

## 一、强制规则

1. 代码缩进必须使用 4 个空格，禁止使用制表符（Tab）。
2. 每行代码长度不得超过 80 个字符，超过时应合理换行。
3. 模块、类、函数、变量等命名必须使用小写字母和下划线组合的方式，如user\_name。
4. 类名使用大写字母开头的驼峰命名法，如UserProfile。
5. 所有 Python 文件必须以 UTF-8 编码保存。
6. 在导入模块时，必须按照标准库、第三方库、自定义库的顺序依次导入，且每组导入之间需空一行。
7. 禁止使用通配符导入模块，如from module import \*。
8. 条件语句、循环语句、函数定义等必须使用冒号（:）开启新的代码块。
9. 函数和类的定义前后必须空两行。
10. 必须为公共函数和类编写文档字符串（Docstring），说明功能、参数、返回值等信息。
11. 比较操作中，常量必须放在比较运算符的右侧，如if x == 10: 。
12. 异常处理时，必须捕获具体的异常类型，禁止使用except:捕获所有异常。
13. 字符串拼接必须使用str.join()方法，而不是使用+运算符（在循环拼接字符串场景）。
14. 所有全局变量必须在模块顶部定义，且命名使用全大写字母和下划线组合，如MAX\_COUNT。
15. 函数参数默认值必须为不可变对象，避免因可变对象默认值导致的意外行为。
16. 布尔表达式中，禁止将非布尔值直接作为条件，如if some\_list: 需改为if len(some\_list) > 0: 。
17. 字典和列表推导式必须控制复杂度，嵌套层级不能超过 2 层。
18. 必须使用with语句来处理文件、数据库连接等资源，确保资源正确释放。
19. 所有模块必须包含\_\_all\_\_变量，明确导出的内容。
20. 代码中禁止出现硬编码的路径、端口号等配置信息，应使用配置文件或环境变量。

## 二、推荐规则

1. 在定义类时，推荐使用@dataclass装饰器来简化类的定义和初始化，如@dataclass class Person: 。
2. 推荐使用logging模块进行日志记录，而不是直接打印输出。
3. 对于复杂的条件判断，推荐使用if-elif-else结构，提高代码可读性。
4. 推荐使用typing模块进行类型注解，提高代码的可维护性和可读性。
5. 当函数返回多个值时，推荐使用元组或namedtuple，而不是多个单独的返回语句。
6. 推荐使用enum.Enum来定义枚举类型，增强代码的语义性和可读性。
7. 在使用循环时，推荐使用for循环遍历可迭代对象，而不是使用while循环（在明确可迭代场景）。
8. 推荐在模块顶部添加版权声明和作者信息。
9. 对于较长的表达式，推荐使用括号来提高可读性，如result = ((a + b) \* (c - d)) / e 。
10. 推荐使用functools.lru\_cache装饰器对频繁调用且结果不常变的函数进行缓存，提高性能。
11. 当处理集合类型数据时，推荐使用集合操作符（如&、|）来替代循环实现交集、并集等操作。
12. 推荐使用contextlib.contextmanager装饰器自定义上下文管理器，简化资源管理代码。
13. 对于较大的项目，推荐使用pytest进行单元测试，并遵循测试用例命名规范（如test\_function\_name）。
14. 推荐使用flake8、pylint等工具进行代码静态检查，及时发现潜在问题。
15. 当处理日期和时间时，推荐使用datetime模块中的相关类和方法，避免使用字符串进行复杂的日期时间计算。
16. 推荐使用pathlib模块来处理文件路径，提高跨平台兼容性。
17. 在编写 SQL 语句时，推荐使用参数化查询，避免 SQL 注入风险。
18. 对于常量定义，推荐使用typing.Final进行类型标注，明确其不可变性。
19. 推荐在项目中使用requirements.txt文件来管理第三方库依赖，并使用pip freeze > requirements.txt命令更新。
20. 当代码中存在魔法数字（Magic Number）时，推荐将其定义为常量，并使用有意义的命名。

## 三、允许规则

1. 可以使用pass语句作为占位符，用于暂时未实现功能的函数、类或代码块。
2. 允许在必要时使用assert语句进行调试，但在生产环境中应确保assert语句不会影响程序逻辑（可通过命令行参数控制assert行为）。
3. 允许使用global关键字在函数内部访问和修改全局变量，但应谨慎使用，避免代码难以理解和维护。
4. 可以使用yield关键字定义生成器函数，以实现延迟计算和节省内存。
5. 允许使用lambda表达式定义简短的匿名函数，但应确保其功能简单明了，不影响代码可读性。
6. 可以使用try-except-else结构，在except块捕获异常，在else块处理无异常时的逻辑。
7. 允许在代码中使用print语句进行简单的调试输出，但在正式发布的代码中应删除或替换为日志记录。
8. 可以根据具体需求选择使用列表、元组、集合或字典等数据结构。
9. 允许使用import as语句为导入的模块或对象指定别名，以简化代码书写，如import numpy as np 。
10. 可以使用super()函数调用父类的方法，实现方法重写和继承。
11. 允许在类中定义\_\_init\_\_方法以外的其他特殊方法（如\_\_str\_\_、\_\_repr\_\_），以提供自定义的对象表示和输出。
12. 可以使用if \_\_name\_\_ == "\_\_main\_\_"语句来定义模块的主执行入口，方便模块的测试和运行。
13. 允许在代码中使用importlib模块动态导入模块，实现更灵活的代码结构。
14. 可以根据项目需求选择使用合适的 Python 版本和第三方库。
15. 允许在代码中使用注释来解释复杂的逻辑或算法，但注释应简洁明了，避免过多冗余。
16. 可以使用zip函数同时遍历多个可迭代对象。
17. 允许使用filter和map函数对可迭代对象进行过滤和转换操作。
18. 可以在代码中使用for-else结构，在循环正常结束时执行else块中的代码（未通过break跳出循环）。
19. 允许使用async和await关键字编写异步代码，提高程序的并发性能。
20. 可以根据项目实际情况，对代码进行适当的重构和优化，以提高代码质量和可维护性。