技术管理专项作业训练

a.强制

1. 所有代码块必须统一使用 4 个空格缩进，禁止使用 Tab。
2. 每行代码不得超过 80 个字符，超出需换行处理。
3. 所有源代码文件必须使用 UTF-8 编码。
4. 变量、函数、类命名必须遵循 PEP8 命名规范：
5. 函数、变量名：小写字母+下划线（如：get_data()）；
6. 类名：首字母大写驼峰（如：DataProcessor）。
7. 所有函数、类必须添加 docstring 注释，说明用途、参数和返回值。
8. 禁止一行写多个语句（每行仅允许一个逻辑语句）。
9. 禁止在生产代码中使用 print()、pdb 等调试语句。
10. 禁止裸 except:，必须明确捕获具体异常类型（如 exceptValueError:）。
11. 禁止使用魔法数字，应使用具名常量代替（如 TIMEOUT=30）。
12. 所有代码提交前必须通过静态代码检查工具（如 flake8、pylint）。
13. 生产代码中禁止硬编码敏感信息（如密码、密钥、APIToken 等）。
14. 所有代码合并前必须通过单元测试，且测试覆盖率不低于 80%。
15. 所有关键操作（如数据库、网络请求）必须有异常处理机制。
16. 源代码不得直接存放于根目录，需放入 src/或模块包中。
17. except 块中必须包含处理逻辑，禁止空处理。
18. 禁止在函数或循环体内导入模块，应统一放置于文件开头。
19. 禁止使用高风险函数 eval()、exec()。
20. 不得直接修改第三方库源码，如需扩展，应使用封装或继承。
21. 所有代码合并必须经至少 1 人 CodeReview 通过。
22. 项目必须使用 Git 进行版本控制，提交信息需规范（如 feat:/fix:/docs:等前缀）。

b.推荐

1. 尽量使用类型注解（TypeHints）提升可读性与工具支持。
2. 类的私有属性应以下划线_开头（如_internal_buffer）。
3. 尽量避免函数参数过多，建议不超过 5 个，可使用数据类封装。
4. 复杂表达式应拆分为多步赋值，提高可读性。
5. 使用列表推导式/生成器表达式时保持简洁，避免多层嵌套。
6. 优先使用内置迭代器与内置函数（如 enumerate()、zip()、map()）提高性能。
7. 函数长度建议不超过 40 行，超过应逻辑拆分。
8. 推荐使用 f-string 格式化字符串，简洁且高效。
9. 使用 with 语句管理资源（如文件、锁），自动处理关闭与异常。
10. 模块、包目录应包含__init__.py 文件，确保其为可导入包。
11. 单元测试应与被测模块同名，统一放置于 tests/目录下。
12. 日志记录应使用 logging 模块，避免输出到标准输出。
13. 尽量避免全局变量，必要时应通过配置文件或类属性集中管理。
14. 代码中如有 TODO、FIXME，应注明责任人和预期完成时间。
15. 减少循环嵌套层级，建议不超过两层。
16. 推荐使用虚拟环境（venv、conda）管理项目依赖。

17. 项目需包含 README.md，清晰说明安装、使用与测试方式。
18. 推荐使用 requirements.txt 或 pyproject.toml 管理依赖项。
19. 推荐引入 CI 工具（如 GitHubActions、GitLabCI）自动化测试与部署。
20. 推荐使用 black、isort、ruff 等工具实现代码自动格式化与风格统一。

c.允许

1. 允许适度使用 lambda 表达式，但应控制逻辑复杂度。
2. 允许在函数内部定义嵌套函数，但应避免嵌套过深。
3. 允许使用装饰器封装通用逻辑，但需注释清楚其作用。
4. 允许使用第三方库扩展标准库功能，前提是评估依赖风险。
5. 允许不同模块风格略有差异，但需团队达成共识并记录约定。
6. 允许使用 multiprocessing、threading 或 async 异步框架提升性能，但必须经充分测试。
7. 允许自定义异常类型用于细粒度错误管理。
8. 性能瓶颈处允许使用 Cython、Numba 等工具进行加速。
9. 类型注解中允许使用高级类型（如 Any、Union、Optional）提升灵活性。
10. 允许使用 enum 模块定义常量集，提高可读性。
11. 项目文档允许使用 Markdown、reStructuredText 等格式。
12. 对外 API/类库接口允许编写用法示例，提高可用性。
13. 特殊情况下允许违背部分推荐规范，但需记录理由并经团队评审同意。