**Python 技术管理文档规范**

**一、基础规范**

**a. 强制**

1. 所有Python文件必须使用 UTF-8 编码。
2. 每个模块文件必须有文件头注释，说明作者、时间、功能。
3. 缩进必须统一使用4个空格，不得混用Tab。
4. 每个函数之间必须空一行，类之间空两行。
5. 行宽限制为不超过120列，超过需换行。
6. 所有换行符使用 Unix 风格（\n），禁止使用 Windows 风格（\r\n）。
7. 顶级定义必须靠左缩进，禁止嵌套定义模块级函数或类。
8. 所有模块必须添加 if \_\_name\_\_ == "\_\_main\_\_" 判断保护入口。

**b. 推荐**

1. 同级导入使用单行，跨级导入使用多行（组间空一行）。
2. 避免使用 ; 一行多个语句。
3. 建议使用括号自动换行，少用 \。
4. 每个文件尽量控制在1000行以内，函数不超过100行。

**c. 允许**

1. \_\_init\_\_.py 可空但推荐注明用途（如用于标记包结构）。

**二、命名规范**

**a. 强制**

1. 模块名使用小写字母+下划线，禁止使用大写。
2. 类名使用首字母大写的驼峰式（PascalCase）。
3. 函数、变量名使用小写+下划线（snake\_case）。
4. 常量使用大写字母加下划线命名（如：MAX\_RETRIES）。
5. 私有变量/函数使用单下划线开头（如 \_process()）。
6. 不得覆盖内置名称（如 list、str、id 等）。

**b. 推荐**

1. 变量命名应语义明确，避免 a, b, c 等无意义名。
2. 类中属性使用 self.<name> 形式定义，避免直接暴露全局变量。
3. 异常类名以 Error 结尾，如 InputError。

**c. 允许**

1. 特殊情况可使用简写（如db\_conn、cfg），需在注释中说明。

**三、注释规范**

**a. 强制**

1. 所有公共函数、类必须使用文档字符串（docstring）。
2. 中文注释必须为简体中文，统一风格。
3. 所有临时代码必须带 TODO 或 FIXME 标记，后接处理人和时间。
4. 不得留下无效注释（如 # 变量定义）。

**b. 推荐**

1. 使用Google、PEP257或NumPy风格的docstring。
2. 复杂算法/条件判断需块级注释说明。
3. 建议为每个模块的主流程添加流程注释。

**c. 允许**

1. 可使用行尾注释，但仅用于补充变量说明，不宜过长。

**四、模块与包结构**

**a. 强制**

1. 模块结构应体现层次性，如 api/, core/, utils/ 等。
2. 同一包中不得存在多个入口类。
3. 每个模块必须职责单一，不得功能混杂。

**b. 推荐**

1. 包结构不宜过深，一般控制在3级以内。
2. 将通用工具方法抽离至 utils 模块。

**c. 允许**

1. 可根据业务将项目拆分为多个子模块，使用 namespace 组织结构。

**五、函数与类设计**

**a. 强制**

1. 函数必须有明确的输入/输出，参数应合理命名。
2. 类不得超过500行代码，建议进行职责拆分。
3. 所有类的构造函数（\_\_init\_\_）必须初始化关键属性。
4. 不得使用 \*args 和 \*\*kwargs 捕获所有参数后随意处理。

**b. 推荐**

1. 函数参数不超过5个，过多建议封装成对象。
2. 类之间应使用组合而非继承来复用逻辑。
3. 对外接口函数应加入类型提示（type hints）。

**c. 允许**

1. 可使用装饰器封装公共逻辑，如日志、权限校验。

**六、异常处理**

**a. 强制**

1. 所有异常必须明确捕获类型，不得使用 except:。
2. 异常日志必须记录完整堆栈信息（traceback）。
3. 不得抛出 BaseException、SystemExit、KeyboardInterrupt等非业务异常。

**b. 推荐**

1. 使用自定义异常类提升可读性。
2. 对于关键操作使用 try...except...finally 保证资源释放。

**c. 允许**

1. 可使用上下文管理器（with）封装异常处理。

**七、测试与调试**

**a. 强制**

1. 所有核心逻辑必须有对应单元测试。
2. 不得在生产代码中使用 print 进行调试。
3. 所有测试代码需放在 tests/ 目录中，遵循统一命名规范。

**b. 推荐**

1. 使用 pytest 进行单元测试。
2. 单元测试覆盖率应不低于80%。
3. 每次提交前应运行测试并通过。

**c. 允许**

1. 可使用 unittest.mock 模拟外部依赖测试。

**八、日志管理**

**a. 强制**

1. 禁止使用 print() 输出日志，统一使用 logging 模块。
2. 日志不得包含用户敏感信息（如密码、身份证、IP）。
3. 所有错误日志必须记录时间、级别、模块名。

**b. 推荐**

1. 日志等级建议：DEBUG < INFO < WARNING < ERROR < CRITICAL。
2. 日志格式应统一（建议包含时间、级别、来源、信息）。

**c. 允许**

1. 可根据模块功能设置子 logger。

**九、安全规范**

**a. 强制**

1. 不得在代码中硬编码密码、Token、密钥等敏感信息。
2. 严禁使用 eval()、exec() 等不安全函数。
3. 所有数据库/外部接口调用必须进行异常与重试处理。

**b. 推荐**

1. 输入输出必须验证合法性（如长度、字符集、类型等）。
2. 所有web项目应进行XSS/SQL注入防护。

**c. 允许**

1. 可使用配置中心/环境变量动态加载敏感配置。

**十、性能优化与并发**

**a. 强制**

1. 不得频繁创建/销毁大对象，应复用资源。
2. 大批量处理必须使用生成器或异步机制避免阻塞。

**b. 推荐**

1. 建议使用 lru\_cache、functools 提高函数性能。
2. IO密集任务建议使用 asyncio/aiohttp。

**c. 允许**

1. 可使用Cython/Numpy等工具加速特定任务。

**十一、开发与部署管理**

**a. 强制**

1. 所有代码提交必须通过 pre-commit 检查。
2. 禁止提交 .pyc/\_\_pycache\_\_/.env 等文件。
3. 所有依赖必须写入 requirements.txt 或 pyproject.toml 中。

**b. 推荐**

1. 使用版本控制工具（如Git）进行规范化分支开发（如feature/、bugfix/）。
2. 推荐使用Docker封装部署Python项目。