**Python 技术管理规范**

**（共40条）**

**2022141461032左骐玮**

**A、强制级规则（Must）**

说明：违反此级别规则的代码不得提交至主分支，CI/CD 流程需拦截此类问题

**缩进与格式**

必须使用 4 个空格缩进（禁止混合 Tab 与空格）

单行代码长度不超过 120 字符（PEP8 建议 79 字符，团队放宽至 120）

二元运算符前换行时，运算符需置于新行开头（如result = (a + b

- c \* d)）

**命名规范**

变量 / 函数 / 方法名使用小写 + 下划线（max\_value/calculate\_average）

类名使用大驼峰（class DataProcessor）

私有成员（类 / 模块内）以单下划线开头（\_private\_method），保护成员（子类可见）以双下划线开头（\_\_protected\_attr）

常量名全大写 + 下划线（MAX\_RETRY\_TIMES = 3）

**导入规范**

禁止使用通配符导入（from module import \*）

导入需按以下顺序分组（每组空一行）：标准库→第三方库→项目内模块

同一模块的多行导入需使用圆括号包裹（如from module import (

func1, func2

)）

**异常处理**

捕获异常时必须指定具体异常类型（禁止裸except:）

禁止直接忽略异常（except KeyError: pass需添加注释说明合理性）

自定义异常需继承Exception基类（禁止直接继承BaseException）

**注释与文档**

公共函数 / 类 / 模块必须编写文档字符串（Docstring），格式遵循 Google 风格（见附录 A）

复杂逻辑（如算法、条件分支）必须添加行内注释（注释与代码间隔 2 空格）

弃用的代码 / 函数必须添加@deprecated装饰器并标注替代方案（需安装deprecation库）

**类型提示**

公共函数的参数 / 返回值必须标注类型（如def add(a: int, b: int) -> int:）

容器类型需使用泛型标注（如def process(items: list[str]) -> dict[str, float]）

可选参数需明确Optional类型（如def connect(host: str, port: Optional[int] = None)）

**依赖与版本**

项目必须通过requirements.txt或pyproject.toml声明依赖，禁止直接使用pip install安装未记录的包

第三方库版本需指定明确范围（如numpy>=1.24,<1.26），禁止使用\*通配符

禁止将测试 / 开发依赖（如pytest）与生产依赖混合声明

**版本控制**

提交代码前必须运行pylint（得分≥8）和mypy（无错误），结果需附在提交信息中

分支命名必须符合规范：功能分支feature/xxx，修复分支fix/xxx，发布分支release/v1.2

合并请求（MR）必须关联 Jira 任务号（如[SPM-123] 优化用户认证逻辑）

**测试要求**

新增功能必须配套单元测试（覆盖率≥80%），CI 流程需阻断覆盖率不达标提交

测试用例必须独立（禁止依赖外部状态，使用pytest.fixture管理测试上下文）

接口测试需覆盖成功 / 失败 / 边界场景（如test\_login\_success/test\_login\_invalid\_password/test\_login\_empty\_input）

**基础规范**

禁止使用eval()/exec()执行动态代码（除非经技术委员会审批）

字符串拼接必须使用f-string（Python 3.6+），禁止%格式化或+拼接长字符串

文件编码必须声明为# -\*- coding: utf-8 -\*-（顶部首行）

**代码安全**

禁止硬编码敏感信息（如 API 密钥、数据库密码），必须通过环境变量（os.getenv）或配置中心（如pydantic-settings）获取，CI/CD 流程需扫描并拦截明文密钥提交。

**函数 / 类设计**

公共类必须显式定义\_\_init\_\_方法的参数类型提示（如class User: def \_\_init\_\_(self, id: int, name: str): ...），且参数顺序需遵循 “必填参数在前，可选参数在后” 原则。

**模块结构**

包（package）内必须包含\_\_init\_\_.py文件（即使内容为空），用于明确标识 Python 包结构，避免导入时的路径解析错误。

**配置管理**

全局配置（如数据库地址、日志级别）必须通过独立的config模块集中管理（示例：from app.config import settings），禁止分散在多个业务文件中定义。

**日志规范**

生产环境日志必须包含时间戳、日志级别、模块名、线程 / 进程 ID，格式需通过logging.Formatter统一配置（如"%(asctime)s [%(levelname)s] %(module)s:%(thread)d - %(message)s"）。

**数据库操作**

SQL 语句必须使用参数化查询（如cursor.execute("SELECT \* FROM users WHERE id = %s", (user\_id,))），禁止通过字符串拼接生成 SQL，防止 SQL 注入攻击。

**API 设计**

HTTP 接口响应必须包含统一格式（示例：{"code": 200, "msg": "success", "data": ...}），禁止直接返回原始数据（如数据库查询结果）或无结构的字典。

**并发处理**

多线程 / 协程代码中，对共享资源（如全局变量、文件句柄）的访问必须使用threading.Lock（多线程）或asyncio.Lock（协程）加锁保护，禁止无锁操作。

**代码审查**

合并请求（MR）必须至少有 2 名非提交者的团队成员审批通过，禁止自审自合并（管理员权限账户需额外审计）。

**文档管理**

修改公共接口（如函数、类方法）时，必须同步更新对应的文档字符串（Docstring）或外部文档（如 Swagger/Postman 集合），否则视为未完成开发任务。

**B、推荐级规则（Should）**

说明：未遵守时需在代码评审中说明原因，鼓励逐步改进

**代码结构**

单个函数长度不超过 50 行（含空行 / 注释），超过时需拆分

类的方法数不超过 15 个（违反时需检查是否符合单一职责原则）

循环嵌套不超过 3 层（超过时建议用生成器或重构逻辑）

**性能优化**

优先使用内置函数（如map()/filter()）或生成器（(x for x in iterable)）替代显式循环

对高频调用函数添加lru\_cache缓存（需评估内存开销）

涉及大数据量处理时，优先使用pandas/numpy向量化操作替代逐行处理

**日志与调试**

使用logging模块替代print()调试（级别需符合：DEBUG→开发，INFO→运行状态，WARNING→潜在问题，ERROR→功能失效）

关键操作（如数据库写入、接口调用）需记录上下文（如logging.info("用户%d提交订单，金额%.2f", user\_id, amount)）

生产环境禁止输出DEBUG级别日志（通过logging.basicConfig(level=logging.INFO)配置）

**类型与检查**

推荐使用TypedDict定义复杂字典结构（如User = TypedDict("User", {"id": int, "name": str})）

推荐为私有函数添加类型提示（提升代码自文档性）

推荐每周运行mypy --strict检查（发现潜在类型问题）

**协作与维护**

推荐使用pre-commit钩子自动执行代码格式化（如black）和静态检查

推荐为复杂逻辑编写README.md说明设计思路（如算法选择、性能权衡点）

推荐每季度更新依赖库（使用pip-review或pip-tools检查新版本）

**设计模式**

复杂业务逻辑（如支付方式切换、消息通知类型）推荐使用工厂模式 / 策略模式解耦，避免大量if-else分支（示例：通过PaymentFactory动态创建支付处理器）。

**代码重构**

重复代码（同一逻辑≥3 次出现）推荐提取为公共函数 / 类（DRY 原则），并添加单元测试验证重构后的功能一致性。

**测试策略**

关键业务路径（如用户注册→登录→下单）推荐添加端到端（E2E）测试，使用pytest-bdd或selenium模拟用户操作流程。

**依赖升级**

推荐每月使用pip-audit检查依赖库的安全漏洞（示例：pip-audit --strict），对高风险漏洞（CVSS≥7.0）需 48 小时内修复。

**代码可读性**

布尔变量推荐使用明确前缀命名（如is\_valid/has\_permission/needs\_approval），避免flag/status等模糊命名。

**错误处理**

接口层推荐捕获业务异常并转换为友好的错误信息（示例：{"code": 400, "msg": "用户名已存在"}），禁止直接返回 Python 原生异常信息（如KeyError: 'username'）。

**监控集成**

核心功能（如支付接口、数据同步）推荐添加性能监控（使用prometheus或opentelemetry记录响应时间、错误率），关键指标需设置告警阈值。

**代码可维护性**

长期维护的项目推荐添加CONTRIBUTING.md文档，说明开发流程（如如何运行测试、提交 MR）、编码规范（如分支命名）和协作规则。

**C、允许级规则（May）**

说明：根据项目场景灵活选择，需在团队内保持一致

**格式灵活性**

允许单行导入超过 120 字符时使用反斜杠换行（如from very\_long\_module\_name import very\_long\_function\_name as func）

允许在列表 / 字典初始化时使用尾随逗号（如items = [1, 2, 3, ]）

**命名扩展**

允许短生命周期变量使用缩写（如循环变量i/j，临时变量tmp\_df）

允许测试用例使用test\_xxx（pytest 风格）或XxxTest（unittest 风格）命名

**注释风格**

允许文档字符串使用 Google 风格（示例见附录 A）或 reST 风格（需团队统一）

允许行内注释使用中文（如业务逻辑）或英文（如技术术语）

**依赖管理**

允许使用poetry替代pip管理依赖（需提交poetry.lock）

允许在requirements.txt中使用-r base.txt拆分依赖（按环境 / 功能分组）

**测试扩展**

允许使用pytest-mock替代unittest.mock（需团队掌握 mock 技巧）

允许集成测试跳过部分用例（标注@pytest.mark.skip(reason="外部服务不可用")）

**工具选择**

允许使用flake8替代pylint进行代码检查（需在.flake8配置文件中同步团队规则，如max-line-length=120）。

**版本号规则**

允许根据项目类型选择版本号格式：

通用项目推荐使用语义化版本（SemVer）：主版本.次版本.修订号（如v2.1.3）；

内部工具类项目允许使用日期版本：年份.月份.修订号，需团队内统一。