Python 技术管理规范文档

概述

本文档参考Google Python Style Guide等业界标准，制定了Python开发的技术管理规范。规范分为三个级别：

- **强制（MUST）**：必须遵守的规则，违反将导致代码审查失败
- **推荐（SHOULD）**：强烈建议遵守的规则，有助于提高代码质量
- **允许（MAY）**：可选的规则，根据项目具体情况决定

一、代码风格规范

强制规则

1. 缩进使用4个空格

```
# 正确
def function():
    return True

# 错误：使用tab或其他空格数
def function():
    return True
```

2. 行长度不超过88个字符

```
# 正确
result = some_function_with_a_long_name(
    argument_one, argument_two, argument_three
)

# 错误：超过88个字符
result = some_function_with_a_long_name(argument_one, argument_two,
argument_three, argument_four)
```

3. 必须使用UTF-8编码

```
# 文件开头（如果包含非ASCII字符）
# -*- coding: utf-8 -*-
```

4. 导入语句必须分行写

```
```

```python
# 正确
import os
import sys
from typing import List, Dict

# 错误
import os, sys
```

5. 字符串引号保持一致性

- 项目内统一使用单引号或双引号
- 三引号用于文档字符串

推荐规则

6. 按照PEP8标准格式化代码

- 使用black或类似工具自动格式化

7. 合理使用空行分隔代码块

```python
# 推荐
class MyClass:
    """类文档字符串."""

    def __init__(self):
        """初始化方法."""
        self.value = 0

    def method(self):
        """实例方法."""
        return self.value
```

8. 运算符周围添加空格

```python
# 推荐
x = y + z
result = function(a=1, b=2)

# 不推荐
x=y+z
result = function(a=1,b=2)
```

允许规则

9. 可以使用行尾注释，但要适度

```
x = x + 1  # 增加计数器
```

二、命名规范

强制规则

10. 类名使用大驼峰命名

```
# 正确
class UserAccount:
    pass

# 错误
class user_account:
    pass
```

11. 函数和变量使用小写加下划线

```
# 正确
def get_user_info():
    pass

user_name = "john"

# 错误
def getUserInfo():
    pass

userName = "john"
```

12. 常量使用全大写加下划线

```
# 正确
MAX_CONNECTION_COUNT = 100
API_BASE_URL = "https://api.example.com"
```

13. 私有属性和方法使用单下划线前缀

```
class MyClass:
    def __init__(self):
        self._private_attr = 0
```

```
    def _private_method(self):
        pass
```

推荐规则

14. 使用有意义的变量名

```
# 推荐
user_count = len(users)
is_valid = check_validation(data)

# 不推荐
n = len(users)
flag = check_validation(data)
```

15. 避免使用单字母变量名（除了循环计数器）

```
# 推荐
for index, item in enumerate(items):
    process(item)

# 可接受
for i in range(10):
    print(i)
```

允许规则

16. 可以使用缩写，但要保持一致性

```
# 允许
def calc_total_price():
    pass
```

三、文档和注释规范

强制规则

17. 所有公共模块、类、函数必须有文档字符串

```
def calculate_area(radius: float) -> float:
    """计算圆的面积.

    Args:
        radius: 圆的半径
```

```
    Returns:
        圆的面积

    Raises:
        ValueError: 当半径为负数时
    """
    if radius < 0:
        raise ValueError("半径不能为负数")
    return 3.14159 * radius ** 2
```

18. 文档字符串必须使用三重双引号

```
# 正确
def function():
    """这是文档字符串."""
    pass

# 错误
def function():
    '''这是文档字符串.'''
    pass
```

推荐规则

19. 使用Google风格的文档字符串格式

- Args: 参数说明
- Returns: 返回值说明
- Raises: 异常说明

20. 为复杂的业务逻辑添加注释

```
# 使用二分查找算法提高搜索效率
def binary_search(arr, target):
    # 实现代码
    pass
```

21. 注释应该解释"为什么"而不是"是什么"

```
# 推荐：解释原因
time.sleep(1)  # 避免API调用频率过高被限制

# 不推荐：重复代码
time.sleep(1)  # 休眠1秒
```

允许规则

22. 可以使用TODO注释标记待完成的工作

```
# TODO: 添加输入验证
def process_data(data):
    pass
```

四、类型提示规范

强制规则

23. 公共API必须使用类型提示

```
from typing import List, Dict, Optional

def process_users(users: List[Dict[str, str]]) -> Optional[str]:
    """处理用户列表."""
    pass
```

24. 导入类型提示模块必须在typing部分

```
from typing import List, Dict, Union
```

推荐规则

25. 为所有函数参数和返回值添加类型提示

```
def get_user_age(user_id: int) -> int:
    return 25
```

26. 使用Union类型处理多种可能的类型

```
def format_id(user_id: Union[int, str]) -> str:
    return str(user_id)
```

允许规则

27. 可以使用类型别名简化复杂类型

```

```

```
UserDict = Dict[str, Union[str, int]]

def process_user(user: UserDict) -> None:
    pass
```

五、错误处理规范

强制规则

28. 不得使用裸露的except语句

```
# 正确
try:
    risky_operation()
except SpecificException as e:
    handle_error(e)

# 错误
try:
    risky_operation()
except:
    pass
```

29. 必须处理可预见的异常

```
def divide(a: float, b: float) -> float:
    if b == 0:
        raise ValueError("除数不能为零")
    return a / b
```

30. 资源必须正确释放

```
# 正确：使用上下文管理器
with open('file.txt', 'r') as f:
    content = f.read()

# 正确：手动释放资源
try:
    f = open('file.txt', 'r')
    content = f.read()
finally:
    f.close()
```

推荐规则

31. 使用自定义异常类型

```python
class ValidationError(Exception):
    """数据验证错误."""
    pass

def validate_email(email: str) -> None:
    if '@' not in email:
        raise ValidationError(f"无效的邮箱地址: {email}")
```

32. 异常信息应该清晰且有用

```python
# 推荐
raise ValueError(f"用户ID {user_id} 不存在")

# 不推荐
raise ValueError("错误")
```

允许规则

33. 可以使用断言进行调试检查

```python
def calculate_square(x: float) -> float:
    assert x >= 0, "输入值必须非负"
    return x ** 2
```

六、性能和最佳实践

强制规则

34. 避免在循环中进行重复计算

```python
# 正确
length = len(items)
for i in range(length):
    process(items[i])

# 错误
for i in range(len(items)):  # 每次都计算长度
    process(items[i])
```

35. 使用列表推导式替代简单的循环

```
# 正确
squares = [x**2 for x in range(10)]

# 不推荐
squares = []
for x in range(10):
    squares.append(x**2)
```

推荐规则

36. 使用生成器处理大型数据集

```
def read_large_file(file_path: str):
    """逐行读取大文件."""
    with open(file_path, 'r') as f:
        for line in f:
            yield line.strip()
```

37. 使用缓存避免重复计算

```
from functools import lru_cache

@lru_cache(maxsize=128)
def expensive_function(n: int) -> int:
    # 耗时计算
    return result
```

38. 优先使用内置函数和标准库

```
# 推荐：使用内置函数
total = sum(numbers)

# 不推荐：手动实现
total = 0
for num in numbers:
    total += num
```

允许规则

39. 可以使用第三方库提高开发效率

```
import requests  # HTTP库
import pandas as pd  # 数据处理库
```

七、测试规范

强制规则

40. 所有公共函数必须有单元测试

```
import unittest

class TestCalculator(unittest.TestCase):
    def test_add(self):
        result = add(2, 3)
        self.assertEqual(result, 5)

    def test_divide_by_zero(self):
        with self.assertRaises(ValueError):
            divide(10, 0)
```

推荐规则

41. 测试文件名以test_开头

```
test_calculator.py
test_user_service.py
```

42. 使用描述性的测试方法名

```
def test_user_creation_with_valid_email_succeeds(self):
    pass

def test_user_creation_with_invalid_email_raises_error(self):
    pass
```

允许规则

43. 可以使用pytest等第三方测试框架

```
import pytest

def test_calculation():
    assert add(2, 3) == 5
```

八、安全规范

强制规则

44. 永远不要在代码中硬编码敏感信息

```
# 错误
API_KEY = "sk-1234567890abcdef"

# 正确
import os
API_KEY = os.getenv('API_KEY')
```

推荐规则

45. 验证所有外部输入

```
def process_user_input(user_input: str) -> str:
    if not isinstance(user_input, str):
        raise TypeError("输入必须是字符串")

    if len(user_input) > 1000:
        raise ValueError("输入长度超出限制")

    return user_input.strip()
```

九、项目结构规范

推荐规则

46. 使用标准的项目结构

```
project/
├── src/
│   └── package/
│       ├── __init__.py
│       └── module.py
├── tests/
│   └── test_module.py
├── requirements.txt
├── setup.py
└── README.md
```

47. 每个包必须包含__init__.py文件

48. 使用requirements.txt管理依赖

```
requests==2.28.1
pandas>=1.5.0
pytest>=7.0.0
```

十、工具和配置

推荐规则

49. 使用代码格式化工具

- 推荐使用black进行代码格式化
- 配置pre-commit钩子自动格式化

50. 使用静态分析工具

- 使用pylint或flake8进行代码检查
- 使用mypy进行类型检查
