一、代码风格规范：PEP 8

官方指南，核心建议包括：

✅ 缩进：

- 使用 4 个空格缩进，不要用 tab。

✅ 命名风格：

- **变量/函数名**： `snake_case` ，如： `calculate_total()`
- **类名**： `PascalCase` ，如： `CustomLogger`
- **常量名**： `ALL_CAPS` ，如： `MAX_RETRIES`
- 模块文件名小写加下划线，如： `data_loader.py`

✅ 空格使用：

```
# 正确
x = 1
y = x * (x + 2)

# 错误
x=1
y = x*(x+2)
```

✅ 每行最多 79 字符，文档字符串最多 72 字符。

✅ 注释规范：

- 单行注释用 `#` ，与内容之间加一个空格。
- 文档字符串用 `"""` 包裹，适用于模块、函数、类。

二、项目目录结构（通用模板）

```
your_project/
├── your_package/             # 项目主代码
│   ├── __init__.py
│   ├── module1.py
│   └── module2.py
├── tests/                    # 单元测试
│   ├── __init__.py
│   └── test_module1.py
├── scripts/                  # 命令行工具（可选）
├── docs/                     # 文档目录（可选）
├── requirements.txt          # 依赖列表
├── setup.py                  # 安装/发布配置
├── pyproject.toml            # 现代构建配置（推荐）
├── README.md                 # 项目说明
└── .gitignore
```

三、文档规范

- 每个模块、函数、类都写清楚 **用途、参数、返回值**。
- 推荐使用 Google 或 NumPy 风格的 docstring，例如：

```python
def add(a: int, b: int) -> int:
    """Return the sum of two integers.

    Args:
        a (int): First number.
        b (int): Second number.

    Returns:
        int: The sum of a and b.
    """
    return a + b
```

四、依赖管理

- 使用 `requirements.txt` 明确列出依赖（`pip freeze > requirements.txt`）。
- 更推荐用 `poetry` 或 `pipenv` 管理依赖和虚拟环境。
- 使用 `.gitignore` 排除虚拟环境文件夹。

五、测试规范

- 所有功能要有测试，使用 `unittest` / `pytest` 都可以。
- 测试文件命名：`test_*.py`
- 测试目录结构与主项目结构保持一致。

六、版本控制（Git）规范

- 遵循 Conventional Commits：

  ```
  feat: 添加用户注册功能
  fix: 修复登录 bug
  docs: 修改 README 说明
  ```

- 不把 `.pyc`、虚拟环境、数据文件上传。

七、类型注解（推荐）

- 使用 Python 的 `typing` 提升可读性与静态检查能力。

```python
from typing import List

def get_names(users: List[str]) -> List[str]:
    ...
```

八、其他建议

- 使用 `logging` 替代 `print`。
- 遵守 “函数只做一件事” 原则。
- 每个模块 < 500 行，每个函数 < 50 行。
- 使用 `.env` 管理环境变量（如数据库密码）。
- 代码中不要硬编码路径、密码、API 密钥。

如果你是做团队项目或者希望进大厂，**建议你强制使用这些规范**。可以使用自动化工具帮助你：

目标	工具
格式化代码	`black`，`yapf`
静态检查	`flake8`，`pylint`
类型检查	`mypy`
安装依赖	`poetry`，`pipenv`
自动测试	`pytest`