STAT243 Lecture 4 Good Practices (Reduced)

1 Good Coding Practices

1.1 Code Syntax and Style

1.1.1 General Rules

- **参考标准**：遵循 Python 的官方风格指南 PEP 8。
- **文件头部信息**：用注释说明作者、日期、用途、Python 版本、主要依赖库。
- **Docstrings**：为公共模块、类、函数编写文档字符串；非公共项使用行内注释。
- **缩进**：每层 4 个空格（避免使用制表符）。
- **空白与可读性**：
  - 操作符两侧加空格；
  - 函数参数与元素之间加空格；
  - 使用空行分隔逻辑块；
  - 使用括号分组长表达式以便换行与注释。

> ⚠ **Remark**
>
> 避免超过 **79 字符的代码行** 和 **72 字符的注释行**。
> 可用 `ruff format` 或 `black` 自动格式化。

- **命名约定（PEP8）**：
  - 类名 → `UpperCamelCase`
  - 函数、变量 → `snake_case`
  - 非公共方法 → `_leading_underscore`
  - 函数名使用动词：如 `calc_mean()`、`compute_error()`。
- **注释原则**：
  - 说明复杂逻辑或魔法常数；
  - 简洁明确，完整句式；

1.1.2 Linting

- **Linting 定义**：自动检测代码风格与潜在错误的工具。
- **推荐工具**：`ruff`（轻量快速）或 `black`。

Shell

```
pip install ruff
ruff check test.py   ## 检查语法
ruff format test.py  ## 自动格式化
```

`ruff` 自动修复空格、缩进与注释风格。

1.2 Assertions, Exceptions, and Testing

1.2.1 Exceptions

- **异常（Exception）**：运行时错误的处理机制。
- 可通过 `raise` 主动抛出异常：

Python

```python
def myfun(val):
    if val <= 0:
        raise ValueError("`val` should be positive")
```

- **try-except 结构**：捕获并处理异常。

Python

```python
try:
    with open("file.txt", "r") as f:
        text = f.read()
except Exception as err:
    print(f"{err}\nCheck path: {os.getcwd()}")
    return None
```

- **re-raise 异常**：

Python

```python
try:
    requests.get(url)
except Exception as err:
    print("Problem accessing URL.")
    raise
```

1.2.2 Assertions

- **断言（assert）** 用于验证程序状态是否符合预期：

Python

```python
number = -42
assert number > 0, f"Expected positive, got {number}"
```

若条件不满足，抛出 `AssertionError`。

- **常用断言形式**：

Python

```python
assert x in y
assert isinstance(x, float)
assert all(arr)
```

1.2.3 Testing

- **单元测试（Unit Tests）**：验证单个函数的行为。
- 推荐使用 `pytest`：

Python

```python
import pytest
import dummy

def test_numeric():
    assert dummy.add_one(3) == 4

def test_bad_input():
```

```
    with pytest.raises(TypeError):
        dummy.add_one('hello')
```

运行测试：

Shell

```
pytest test_dummy.py
```

---

2 Debugging and Recommendations for Avoiding Bugs

2.1 防止与捕获错误的技巧

2.1.1 Defensive Programming（防御式编程）

- 检查函数输入的有效性。
- 提供合理默认值。
- 对输入/输出进行范围验证。
- 用 `assert`、`try` 或 `raise` 抛出带信息的错误。

2.1.2 try/except 捕获运行时错误

2.1.3 维度保持（Dimensionality Handling）

2.1.4 避免使用全局变量