Python Coding Conventions

A concise guide to basic rules for writing clear, consistent Python code across the team.

---

1. File Organization & Imports

- Keep each `.py` file focused on a single purpose.
- **Top of file**:
    1. Module docstring (short description).
    2. Standard library imports.
    3. Third-party imports.
    4. Local application imports.
- **Blank lines**: Use a blank line between each import group.
- **Line length**: Wrap lines at **79 characters** (maximum 99 in special cases).

```
"""module docstring"""

import os
import sys

import numpy as np
from PIL import Image

from .utils import helper_function
```

---

2. Naming Conventions

Entity	Style	Example
Variables	`snake_case`	`train_loader`
Functions	`snake_case`	`load_data()`
Classes	`PascalCase`	`ImageClassifier`
Constants	`UPPER_SNAKE_CASE`	`DEFAULT_BATCH_SIZE`
Modules (files)	`snake_case.py`	`data_utils.py`
Packages (folders)	lowercase, no symbols	`models/`

---

3. Formatting & Style

- **Indentation**: 4 spaces per level (no tabs).
- **Spaces around operators**: Yes.
- **No trailing whitespace**.

- **Blank lines**:
    - 2 blank lines between top-level function and class definitions.
    - 1 blank line between methods in a class.

```
# Good:

def foo():
    pass


class Bar:
    def method_one(self):
        pass

    def method_two(self):
        pass
```

---

4. Docstrings & Comments

- **All public functions and classes** should have a docstring summarizing:
    - What it does.
    - Key arguments and return values (brief).
- **Use triple-quotes** (`"""Docstring."""`).
- **Inline comments** sparingly, only to explain non-obvious code.
- **TODO/FIXME** tags for known issues, with a short description.

```
def preprocess(image_path: str) -> np.ndarray:
    """
    Load an image file and convert it to a normalized NumPy array.

    Args:
        image_path: Path to the image file.

    Returns:
        A (H, W, C) array of floats in [0, 1].
    """
    # Read and normalize
    arr = np.array(Image.open(image_path)) / 255.0
    return arr
```

---

5. Error Handling

- **Raise exceptions** for invalid inputs or unrecoverable errors.
- **Catch only** specific exceptions when you can handle them meaningfully.
- **Don’t** use bare `except:`; always specify the exception class.

```
try:
    result = compute(data)
except ValueError as e:
    logger.error(f"Invalid data: {e}")
    raise
```

By following these basic rules, our codebase will stay clean, consistent, and maintainable.