Coding Standards

1. Consistency

- Ensure a uniform coding style throughout the entire codebase to improve readability and maintainability.
- Use consistent naming conventions, indentation, and formatting throughout the codebase.

---

2. Naming Conventions

2.1 Variables

- Use **snake_case** for variable names (all lowercase with underscores).
- Choose descriptive and meaningful names that convey the variable's purpose.
- Avoid single-letter or ambiguous names unless used in small scopes (e.g., loop counters).
- Leading underscore can indicate internal or "private" variables.

Example:

```
# Descriptive and uses snake_case
total_price = 49.99
```

2.2 Constants

- Define constants at the module level.
- Use **UPPERCASE_SNAKE_CASE** for constants to indicate immutability by convention.

- Python does not enforce immutability but follow this convention for clarity.

  Example:

```python
# All uppercase for constants
MAX_ATTEMPTS = 3
```

2.3 Functions

- Use **snake_case** for function names.
- Function names should be **action-oriented** and descriptive, representing what the function does.

  Example:

```python
# Descriptive function name in snake_case
def send_notification(user_id):
    # Sends a notification to the given user_id
    pass
```

2.4 Classes

- Use **CamelCase** for class names.
- Class names should start with an uppercase letter and be nouns or noun phrases.

  Example:

```python
# CamelCase for class name
class OrderHistory:
    pass
class UserProfile:
    pass
```

2.5 Packages & Modules

- Use **short, all-lowercase** names for packages and modules.
- Avoid underscores in package names to maintain compatibility across OSs.
- Module names may use underscores if necessary for readability.

Example:

```python
# CamelCase for class name
class OrderHistory:
    pass
class UserProfile:
    pass
```

---

3. Comments & Documentation

- Use **docstrings** (""" ... """) to describe modules, classes, methods, and functions.
- Docstrings should clearly explain the purpose, arguments, and return values.
- Use inline comments sparingly, only when the code's intent is not obvious.
- Avoid obvious comments that repeat what the code does.

Example:

```python
def calculate_discount(price, percentage):
    """
    Calculate the discount amount for a given price and percentage.

    Args:
        price (float): The original price.
        percentage (float): Discount percentage.

    Returns:
        float: The discount amount.
    """
    # Ensure percentage is within 0-100
    if not 0 <= percentage <= 100:
        raise ValueError("Percentage must be between 0 and 100.")
    return price * (percentage / 100)
```

---

4. Formatting & Indentation

- Use **4 spaces** per indentation level; no tabs.
- Limit lines to **79 characters** to improve readability (PEP 8).
- Use blank lines to separate logical sections of code:
  - Two blank lines before top-level functions and classes.
  - One blank line between methods inside a class.

Example:

```python
def display_user(name):
    # Good formatting with 4 spaces per indent
    if name:
        print(f"Hello, {name}!")
    else:
        print("No user name provided.")

```

---

5. Error Handling

- Use **try-except** blocks to gracefully handle exceptions.
- Prefer **specific exceptions** over broad ones (except ValueError rather than except Exception).
- Optionally, raise meaningful exceptions with informative messages.

Example:

```python
try:
    # Try to open a file
    with open("data.txt", "r") as file:
        content = file.read()
except FileNotFoundError as e:
    # Handle specific file not found error
    print("File not found:", e)
except Exception as e:
    # Handle any other exceptions
    print("An error occurred:", e)

```

---

6. Import Formatting

- Write **one import statement per line**.
- Organize imports in this order with a blank line between groups:
    1. Standard library imports
    2. Third-party imports
    3. Local application imports
- Use **absolute imports** rather than relative imports for clarity.

Example:

```python
# Standard library imports
import os
import sys

# Third-party imports
import requests

# Local imports
from myproject.utils import send_notification
```

---

7. URL Formatting

- In web applications or APIs, use **lowercase letters** in URLs.
- Separate words with **hyphens (-)** or **underscores (_)** to improve readability.
- Keep URLs concise and meaningful.

Example:

```
# Good URL formatting in web applications or API endpoints

# Using hyphens (preferred)
"https://myapi.com/user-profile"
"https://myapi.com/update-order-status"

# Using underscores (allowed, but less common)
"https://myapi.com/user_profile"
"https://myapi.com/update_order_status"
```

---

8. Template Style (HTML)

- Use consistent and readable indentation in templates (usually 2 or 4 spaces).
- Write clean, semantic HTML markup to improve accessibility and maintenance.

Example

```
<!-- Consistent and readable HTML template formatting -->
<!DOCTYPE html>
<html>
    <head>
        <title>User Dashboard</title>
    </head>
    <body>
        <h1>Welcome, User!</h1>
        <p>This is your dashboard.</p>
    </body>
</html>
```

---

9. Code Readability & Reusability

- Break complex tasks into smaller, reusable functions or methods.
- Use meaningful variable and function names; avoid cryptic abbreviations.
- Encapsulate reusable logic inside functions, classes, or modules to avoid duplication.

  Example:

```python
def get_full_name(first_name, last_name):
    # Break down logic into clear, small functions
    return f"{first_name} {last_name}"

def print_greeting(full_name):
    print(f"Hello, {full_name}!")
```

---

10. Code Reusability

- Encapsulate reusable code into functions, classes, or modules to avoid duplication

  Example:

```python
# Reusable function to calculate area
def calculate_area(length, width):
    # Multiplies length and width to find area of rectangle
    return length * width
```

---

11. Testing and Quality Assurance

- Write **unit tests** to ensure code correctness and prevent regressions.
- Use Python testing frameworks like unittest or pytest.
- Structure tests clearly with descriptive test method names.

- Encapsulate reusable code into functions, classes, or modules to avoid duplication

  Example:

```python
import unittest

# Unit test for calculate_area function
class TestAreaCalculation(unittest.TestCase):
    def test_calculate_area(self):
        # Checks if area calculation is correct for 5 x 10
        self.assertEqual(calculate_area(5, 10), 50)
```

---

12. Security

- Always **sanitize and validate user inputs** to prevent injection and other vulnerabilities.
- Follow best practices for secure coding, especially in web and API development.

  Example:

```python
# Validate user input to avoid injection or logic errors
try:
    age = int(user_input)
    if age < 0 or age > 120:
        raise ValueError("Invalid age range.")
except ValueError as e:
    print("Input error:", e)

# Flask example: sanitize input to prevent XSS
from flask import request, escape

@app.route("/submit", methods=["POST"])
def submit():
    username = escape(request.form['username'])  # Prevent script injection
    return f"Hello, {username}!"
```

13. References & Resources

1. **PEP 8 – Python Style Guide**
   The official Python style guide that defines conventions for code layout and formatting.
   https://peps.python.org/pep-0008/
2. **Google Python Style Guide**
   Google's internal style rules, helpful for writing large-scale, maintainable Python code.
   https://google.github.io/styleguide/pyguide.html
3. **CKAN Python Contribution Guide**
   Community-maintained style and contribution guide for open-source Python projects.
   https://docs.ckan.org/en/2.9/contributing/python.html
4. **Python Coding Standards by Zenesys**
   Best practices and coding guidelines used in real-world enterprise development.
   https://www.zenesys.com/python-coding-standards-best-practices
5. **JU SWE Wiki – Updated Coding Standard (JU_CSE-27)**
   Customized standard used by JU High Fives for consistent academic and team projects.
   https://github.com/JU-CSE-27/swewiki/blob/master/resources/Updated_coding-standard.pdf