Coding Standards (2)

---

1. Consistency:

- Consistent coding style ensures readability and maintainability across the codebase. Ensure uniformity in coding style across the project. Use consistent naming conventions, indentation, and formatting throughout the codebase.

---

2. Naming Conventions:

2.1 Variables:

- Use snake_case for variable names. Start with a lowercase letter or an underscore.
- Choose descriptive and meaningful names that convey the purpose of the variable.

Example:

```
# Good variable names
user_name = "John Doe"
total_students = 100
_internal_variable = 42  # Leading underscore indicates it's private/internal
```

2.2 Constants:

- Constants are usually defined at the module level and should be named using UPPERCASE_SNAKE_CASE.
- Python doesn't have true constants, but using this naming convention indicates that a variable should be treated as a constant and not changed.

Example:

```
# Constants (pseudo-constants in Python)
MAXIMUM_VALUE = 100
PI = 3.14159
```

2.3 Functions:

- Use snake_case for function names.
- Choose descriptive names that represent the action performed by the function.

Example:

```
# Good function names
def calculate_area(length, width):
    return length * width

def format_text(text):
    # Function to format text
    Pass
```

2.4 Classes:

- Use CamelCase for class names.
- Start class names with an uppercase letter.

Example:

```
# Good class names
class RectangleShape:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

2.5 Packages and Modules:

- Use short, all-lowercase names for packages.
- Modules should also have short, all-lowercase names.
- Avoid using underscores in package names to ensure compatibility across operating systems.

Example:

```
# Good package/module names
import mypackage.my_module
```

---

3. Comments and Documentation:

- Use docstrings to describe modules, functions, classes, and methods.
- Use comments to clarify complex logic or provide additional context.

Example:

```
def calculate_area(length, width):
    """Calculate the area of a rectangle.

    Args:
        length (int): The length of the rectangle.
        width (int): The width of the rectangle.

    Returns:
        int: The calculated area.
    """
    return length * width
```

---

4. Formatting and Indentation:

- Use 4 spaces for indentation.
- Limit lines to 79 characters for better readability.

Example:

```
# Good indentation and formatting
if condition:
    # Code block
    pass
```

---

5. Error Handling:

- Utilize try-except blocks to handle exceptions gracefully.
- Raise specific exceptions when needed.

Example:

```
try:
    # Code that might raise an error
except ValueError as e:
    # Handle specific error
    print("ValueError occurred:", e)
except Exception as e:
    # Handle other exceptions
    print("An error occurred:", e)
```

---

6. Import Formatting:

- Use separate lines for each import statement.
- Group imports in the following order: standard library imports, third-party library imports, and local application imports.
- Use absolute imports instead of relative imports when importing within the same project.

Example:

```
# Good import formatting
# Standard library imports
import os
import sys

# Third-party library imports
import pandas as pd
import numpy as np

# Local application imports
from mypackage import module_name
```

---

7. URL Formatting:

- For URLs in web applications or API endpoints, use lowercase letters and separate words with hyphens or underscores.
- Keep URLs concise, descriptive, and meaningful.

Example:

```
# Good URL formatting
# Using hyphens to separate words
https://example.com/user-profile
https://api.example.com/get-data

# Using underscores to separate words
https://example.com/user_profile
https://api.example.com/get_data
```

---

8. Template Style:

- Use consistent and readable HTML markup in templates.
- Utilize appropriate indentation to maintain clarity.

Example:

```
<!-- Good template style -->
<html>
    <head>
        <title>My Website</title>
    </head>
    <body>
        <h1>Welcome to my website!</h1>
        <p>This is a paragraph of text.</p>
    </body>
</html>
```

---

9. Code Readability:

- Break down complex tasks into smaller functions or methods.
- Use meaningful variable names and avoid cryptic abbreviations.

Example:

```
def calculate_area(length, width):
    return length * width

def display_result(result):
    print("The result is:", result)
```

---

10. Code Reusability:

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

Example:

```
# Reusable function to calculate area
def calculate_area(length, width):
    return length * width
```

---

11. Testing and Quality Assurance:

- Write unit tests using Python testing frameworks (unittest, pytest, etc.) to ensure code quality.

Example:

```
import unittest

class TestAreaCalculation(unittest.TestCase):
    def test_calculate_area(self):
        self.assertEqual(calculate_area(5, 10), 50)
```

---

12. Security:

- Sanitize and validate inputs to prevent security vulnerabilities, especially in user inputs or web-related functionalities.

References:

1. https://docs.ckan.org/en/2.9/contributing/python.html
2. https://www.tutorialspoint.com/coding-standards-style-guide-for-python-programs
3. https://peps.python.org/pep-0008/
4. https://github.com/JU-CSE-27/swe-wiki/blob/master/resources/Updated_coding-standard.pdf
5. https://www.zenesys.com/python-coding-standards-best-practices