Coding Standard for Exam Office Management System

What is Coding Standard

Coding standards are guidelines and best practices that help ensure code is written in a consistent, readable, and maintainable way. They can cover various aspects of programming, including syntax, structure, naming conventions, and documentation. Adhering to coding standards helps teams collaborate more effectively and makes the codebase easier to understand and manage over time.

Name of Coding Standard

The followed coding standard is `Python Enhancement Proposal 8 (PEP8)`

Naming Convention

The following are the currently recommended naming convention.

1. Avoid These Names

Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.

Example:

```python
# Avoid using 'l', 'O', or 'I' as single-character variable names
# Good variable names
user_name = "Alice"
file_name = "data.txt"
```

2. Variable Name

Variable names must be written in `snake_case`.

Example:

```python
# Variable names should be in snake_case
total_amount = 100
user_age = 25
```

3. Package and Module Names

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.

Example:

```
# Module name with underscores for readability
# module_name.py

# Package name without underscores
# mypackage/
```

4. Name of Class

Class names should normally use the `PascalCase` convention. The naming convention for functions may be used instead in cases where the interface is documented and used primarily as a callable.

Example:

```
class MyClass:
    def __init__(self, value):
        self.value = value

class MyCallableClass:
    def __call__(self, arg):
        return arg * 2
```

5. Exception Names

The class naming convention applies here. However, you should use the suffix “Error” on your exception names.

Example:

```
class CustomError(Exception):
    pass

class FileNotFoundError(Exception):
    pass
```

6. Function Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Example:

```python
def calculate_total(amount, tax_rate):
    return amount + (amount * tax_rate)

def fetch_data_from_server(url):
    pass
```

7. Function and Method Arguments

Always use `self` for the first argument to instance methods. Always use `cls` for the first argument to class methods.

Example:

```python
class MyClass:
    def instance_method(self, value):
        self.value = value

    @classmethod
    def class_method(cls, value):
        cls.value = value
```

8. Method Names and Instance Variables

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability. Use one leading underscore only for non-public methods and instance variables. To avoid name clashes with subclasses, use two leading underscores to invoke Python's name mangling rules.

Example:

```python
class MyClass:
    def __init__(self):
        self.public_variable = 1
        self._protected_variable = 2
        self.__private_variable = 3

    def public_method(self):
        pass

    def _protected_method(self):
        pass

    def __private_method(self):
        pass
```

9. Constants

Constants are usually defined on a module level and written in all capital letters with underscores separating words.

Example:

```
MAX_CONNECTIONS = 100
PI = 3.14159
DEFAULT_TIMEOUT = 60
```

Coding Convention

The following are the currently recommended coding convention.

1. Indentation

The guideline suggests using 4 spaces per indentation level.

2. Tabs or Spaces

Tabs should be used solely to remain consistent with code that is already indented with tabs. Python disallows mixing tabs and spaces for indentation.

3. Maximum Line Length

Limit all lines to a maximum of 79 characters.

4. Imports

The import statement, just like any other statement or keyword in Python, should be used and added to the code properly following the best practices. Let's see them one by one:

a. Multiple Imports

Multiple imports should usually be on separate lines. For example:

```
import numpy
import pandas
import matplotlib
```

b. Always on the Top

Imports are always put at the top of the file:

- After any module comments and docstrings.
- Before module globals and constants.

Example:

```
# import the numpy module
import numpy
```

c. Import Modules in an Order

A good practice is to import modules in the following order:

- Standard library modules – e.g. `sys`, `os`, `getopt`, `re`.
- Third-party library modules – e.g. `ZODB`, `PIL.Image`, etc.
- Locally developed modules.

d. Absolute Imports

Absolute imports are recommended, as they are usually more readable and tend to be better performed if the import system is incorrectly configured.

Example:

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

e. Wildcard Imports

Wildcard imports (`from module import *`) should be avoided. Avoid wildcard imports since they make it unclear which names are present in the namespace, confusing both readers and many automated tools.

5. Whitespace in Expressions and Statements

Avoid unnecessary whitespace in the following situations:

a. Between a Trailing Comma

```
# Correct:
a = (0,)

# Wrong:
b = (0, )
```

b. Immediately Before a Comma, Semicolon, or Colon

```
# Correct:
if a == 5: print(a, b); a, b = b, a

# Wrong:
if a == 5 : print(a , b) ; a , b = b , a
```

c. Immediately Before the Open Parenthesis That Starts the Argument List of a Function Call

```
# Correct:
demo()

# Wrong:
demo ()
```

d. Immediately Before the Open Parenthesis That Starts an Indexing or Slicing

```
# Correct:
dct['key'] = lst[index]

# Wrong:
dct ['key'] = lst [index]
```

Comments

1. Inline Comments

Use inline comments sparingly and make sure they explain why the code does something, not what it does.
Place inline comments on the same line, separated by two spaces.

```
x = x + 1 _# Increment x by 1_
```

2. Block Comments

Use block comments to explain a section of code.
Start each line with a # and a single space, aligning with the indentation level of the code.

```
_# This block calculates the average
# from a list of scores._
total = sum(scores)
average = total / len(scores)
```

3. Docstrings

Use triple quotes (""") for docstrings in functions, classes, and modules.
Docstrings should explain the purpose of the function, method, or class.

```
def greet(name):
"""
This function takes a name as input and returns a greeting string.
"""
return f"Hello, {name}!"
```

Reference

Here are the references:

PEP 8 -- Style Guide for Python Code
Coding Standards - Style Guide for Python Programs
DataCamp- PEP-8 Tutorial: Code Standards in Python