eNoah Coding Standards - Python

Table of Contents

1. Naming Conventions

1.1 Class Names (PascalCase)

Class names should be nouns in PascalCase with the first letter of each word capitalized.

python

```python
# Goodclass UserProfile:

    pass

class DatabaseConnection:

    pass

class HtmlParser:

    pass

# Badclass user_profile:    # snake_case

    pass

class userProfile:     # camelCase
```

```
    pass
```

1.2 Method and Function Names (snake_case)

Methods and functions should be verbs in snake_case, all lowercase with underscores.

```
python

# Gooddef calculate_user_score():

    pass

def get_html_content():

    pass

def is_valid_input():

    pass

# Baddef CalculateUserScore():    # PascalCase

    pass

def calculateUserScore():    # camelCase

    pass
```

1.3 Variable Names (snake_case)

Variable names should be descriptive and in snake_case.

```
python
```

```python
# Good

user_name = "John"

max_retry_count = 3

is_authenticated = True

# Bad

userName = "John"       # camelCase

MAXRETRYCOUNT = 3       # all caps

isAuthenticated = True # camelCase
```

1.4 Constants (UPPER_SNAKE_CASE)

Constants should be all uppercase with underscores.

```
python

# Good

MAX_CONNECTIONS = 100

DEFAULT_TIMEOUT = 30

API_BASE_URL = "https://api.example.com"

# Bad

maxConnections = 100

default_timeout = 30
```

```
ApiBaseUrl = "https://api.example.com"
```

1.5 Module Names

Module names should be short, lowercase, and avoid underscores when possible.

```
python

# Goodimport jsonimport databaseimport config_loa
der

# Badimport JSONParserimport DataBaseConnectionim
port configLoader
```

1.6 Private Conventions

Use leading underscore for private variables and methods.

```
python

class UserManager:

    def __init__(self):

        self._internal_cache = {}

        self.public_data = []



    def _validate_user(self, user):

        # Private method

        pass
    def get_user(self, user_id):

        # Public method

        pass
```

2. Code Structure

2.1 Imports

Group imports in this order: standard library, third-party, local imports.

python

```
# Standard libraryimport osimport sysfrom datetime import datetime, timedelta

# Third-partyimport requestsfrom django.db import models

# Local applicationfrom myapp.utils import helper_functionfrom .models import UserModel
```

2.2 Indentation

Use 4 spaces for indentation (never tabs).

python

```
# Gooddef process_data(data):

    if data is not None:
        for item in data:

            if item.is_valid():

                item.process()



    return True

# Bad (using tabs or 2 spaces)def process_data(da
ta):

  if data is not None:     # 2 spaces

    for item in data:

        if item.is_valid():   # mixed

            item.process()
```

2.3 Line Length

Maximum line length should be 79 characters (88 if using Black formatter).

python

```
# Good

result = (very_long_variable_name +

          another_long_variable_name +

          final_variable_name)
```

```
# Use parentheses for line continuation

long_function_call(

    parameter_one,

    parameter_two,

    parameter_three)
```

2.4 Blank Lines

Use blank lines to separate logical sections.

```
python

import osimport sys



class DataProcessor:

    """Class for processing data."""



    def __init__(self, config):

        self.config = config

        self.data = []



    def load_data(self, source):
        # Implementation

        pass



    def process_data(self):

        # Implementation

        pass



def helper_function():

    """Standalone helper function."""

    pass
```

3. Documentation

3.1 Docstrings

Use Google-style or NumPy-style docstrings consistently.

python

```
def calculate_area(radius):

    """Calculate the area of a circle.
    Args:

        radius (float): The radius of the circle
in meters.



    Returns:

        float: The area of the circle in square m
eters.



    Raises:

        ValueError: If radius is negative.

        TypeError: If radius is not a number.

    """

    if not isinstance(radius, (int, float)):

        raise TypeError("Radius must be a number")

    if radius < 0:

        raise ValueError("Radius cannot be negati
ve")



    return 3.14159 * radius ** 2
```

```python
class UserManager:

    """Manages user operations and data.



    Attributes:

        users (list): List of active users.

        max_users (int): Maximum allowed users.

    """



    def __init__(self, max_users=100):

        """Initialize UserManager.



        Args:

            max_users (int, optional): Maximum us
ers allowed. Defaults to 100.

        """

        self.users = []

        self.max_users = max_users
```

3.2 Comments

Use comments to explain why, not what. Keep comments up to date.

python

```
# Good# Calculate exponential moving average to reduce noise

ema = (current_value * smoothing) + (previous_ema * (1 - smoothing))

# Bad# Set x to 5

x = 5

# Temporary workaround for API rate limiting - remove after Q4 2024def fetch_data_with_retry():

    # Implementation

    pass
```

3.3 Type Hints

Use type hints for better code clarity and IDE support.

python

```
from typing import List, Dict, Optional, Union

def process_user_data(

    users: List[Dict[str, Union[str, int]]],
    timeout: Optional[int] = None) -> bool:

    """Process user data with type safety."""

    if timeout is None:

        timeout = 30



    return len(users) > 0



class DataProcessor:

    def __init__(self, config: Dict[str, str]) ->
 None:

        self.config = config

        self._cache: Dict[str, List] = {}
```

4. Code Formatting

4.1 If-Else Statements

```
python

# Goodif user.is_authenticated and user.has_permi
ssion('read'):

    display_content()elif user.is_anonymous:
```

```
    show_login_prompt()else:

    show_access_denied()

# Avoid complex nested ifsif (condition_one and

    condition_two and

    condition_three):

    do_something()
```

4.2 Loops

```
python

# Goodfor index, item in enumerate(items):

    if item.is_valid():

        process_item(item)

    else:

        log_invalid_item(index, item)

# Using list comprehensions appropriately

valid_items = [item for item in items if item.is_
valid()]

# Avoid using range(len()) when possiblefor i in
range(len(items)):  # Bad

    process(items[i])
```

```
for item in items:  # Good

    process(item)
```

4.3 Exception Handling

```
python

# Goodtry:

    response = requests.get(url, timeout=10)

    response.raise_for_status()

    return response.json()except requests.excepti
ons.Timeout:

    logger.error("Request timeout for %s", url)

    return Noneexcept requests.exceptions.HTTPErr
or as e:

    logger.error("HTTP error %s for %s", e.respon
se.status_code, url)

    return Noneexcept Exception as e:

    logger.exception("Unexpected error: %s", e)

    return None

# Custom exceptionsclass ValidationError(Exceptio
n):

    """Exception raised for validation errors."""
    def __init__(self, message: str, field: str =
 None):

        self.message = message

        self.field = field

        super().__init__(self.message)
```

4.4 Class Structure

```
python

class DatabaseConnection:

    """Manages database connections."""



    # Class constants

    DEFAULT_TIMEOUT = 30

    MAX_RETRIES = 3



    def __init__(self, connection_string: str):

        self.connection_string = connection_strin
g

        self._connection = None
        self._is_connected = False

    def connect(self) -> bool:
        """Establish database connection."""
        try:
            self._connection = create_connection
(self.connection_string)
            self._is_connected = True
            return True
        except ConnectionError:
            logger.error("Failed to connect to da
tabase")
            return False

    def disconnect(self) -> None:
        """Close database connection."""
        if self._connection:
            self._connection.close()
            self._is_connected = False
```

5. Best Practices

5.1 Magic Numbers

Avoid magic numbers - use named constants.

```python
# Badif status == 1:

    process_active_user()elif status == 2:

    process_inactive_user()

# Good

USER_ACTIVE = 1

USER_INACTIVE = 2

USER_SUSPENDED = 3

if status == USER_ACTIVE:

    process_active_user()elif status == USER_INAC
TIVE:

    process_inactive_user()
```

5.2 Security Considerations

```python
# Avoid eval() and exec()# Bad
```

```
result = eval(user_input)

# Good - use safe alternativesimport asttry:

    result = ast.literal_eval(user_input)except
(SyntaxError, ValueError):

    result = None

# Safe file handlingimport os

# Bad - path traversal vulnerability

file_path = user_provided_path

# Good - validate and sanitize pathsdef safe_open
_file(user_path, base_directory):

    full_path = os.path.abspath(os.path.join(base
_directory, user_path))

    if not full_path.startswith(base_directory):

        raise SecurityError("Path traversal attem
pt detected")



    return open(full_path, 'r')
```

5.3 Performance Guidelines

```
python

# Use generators for large datasetsdef read_large
_file(filename):
    with open(filename, 'r') as file:

        for line in file:

            yield line.strip()

# Avoid unnecessary function calls in loops# Badf
or i in range(len(data)):

    if is_valid(data[i]):  # Function call each i
teration

        process(data[i])

# Good

valid_items = [item for item in data if item.is_v
alid()]for item in valid_items:

    process(item)

# Use built-in functions when possible# Bad

total = 0for number in numbers:

    total += number

# Good

total = sum(numbers)
```

6. Tooling and Automation

6.1 Required Tools

```
ini
```

```
# pyproject.toml[tool.black]line-length = 88targe
t-version = ['py38']


[tool.isort]profile = "black"multi_line_output =
3


[tool.mypy]python_version = "3.8"warn_return_any
= truewarn_unused_configs = true


[tool.flake8]max-line-length = 88extend-ignore =
"E203,W503"exclude = ".git,__pycache__,build,dist
"
```

6.2 Pre-commit Configuration

```
yaml
```

```
# .pre-commit-config.yamlrepos:

  - repo: https://github.com/pre-commit/pre-commi
t-hooks

    rev: v4.4.0

    hooks:

      - id: trailing-whitespace

      - id: end-of-file-fixer

      - id: check-yaml

      - id: check-added-large-files
```

```yaml
  - repo: https://github.com/psf/black

    rev: 23.3.0

    hooks:

      - id: black



  - repo: https://github.com/pycqa/isort

    rev: 5.12.0

    hooks:

      - id: isort



  - repo: https://github.com/pycqa/flake8

    rev: 6.0.0

    hooks:

      - id: flake8
```

8. Testing Standards

```
python
```

```
# tests/test_models.pyimport unittestfrom src.my_
package.core.models import User

class TestUserModel(unittest.TestCase):

    """Test cases for User model."""



    def setUp(self):

        """Set up test fixtures."""

        self.user_data = {

            'username': 'testuser',

            'email': 'test@example.com'

        }



    def test_user_creation(self):

        """Test user creation with valid data."""

        user = User(**self.user_data)

        self.assertEqual(user.username, 'testuser
')

        self.assertEqual(user.email, 'test@exampl
e.com')
```

```python
    def test_user_invalid_email(self):

        """Test user creation with invalid email 
raises error."""

        with self.assertRaises(ValueError):

            User(username='test', email='invalid-
email')

if __name__ == '__main__':

    unittest.main()
```

These Python coding standards ensure consistency, readability, and maintainability across all Python projects at eNoah, following the same professional structure as your PHP standards document.

QMS/SDM/UG/CODING STANDARDS-PYTHON