Руководство по стилю кода Python в соответствии с PEP8

1. Оформление кода

1.1 Отступы

- **Используйте 4 пробела на уровень отступа**:
  - Не используйте табуляцию или смешение табуляции с пробелами.
  - Настройте текстовый редактор так, чтобы при нажатии клавиши табуляции вставлялись 4 пробела.

1.2 Максимальная длина строки

- **Ограничьте все строки максимальной длиной в 79 символов**:
  - Для блоков текста (например, docstring или комментариев) ограничьте строки 72 символами.
  - Используйте обратные косые черты или скобки для переноса длинных строк.

1.3 Пустые строки

- **Разделяйте определения функций и классов верхнего уровня двумя пустыми строками**.
- **Определения методов внутри класса должны быть разделены одной пустой строкой**.
- **Обрамляйте код верхнего уровня в функции или классе двумя пустыми строками**.

1.4 Импорты

- **Импорты, как правило, должны быть на отдельных строках**:

```
# Правильно:
import os
import sys

# Неправильно:
import sys, os
```

- **Инструкции импорта должны располагаться в начале файла, сразу после любых комментариев модуля и docstring.**
- **Импорты должны быть сгруппированы в следующем порядке**:
    1. Импорты стандартной библиотеки.
    2. Импорты сторонних библиотек.
    3. Импорты, специфичные для приложения/библиотеки.
    - **Пример**:

```
import os
import sys

import requests

from mymodule import myfunction
```

1.5 Пробелы в выражениях и операторах

- **Избегайте лишних пробелов в следующих случаях**:
    - Внутри круглых, квадратных скобок или фигурных скобок.
    - Между запятой и закрывающей скобкой.
    - Перед запятой, точкой с запятой или двоеточием.
    - **Правильно**:

```
spam(ham[1], {eggs: 2})
```

    - **Неправильно**:

```
spam( ham[ 1 ], { eggs: 2 } )
```

- **Всегда окружайте бинарные операторы пробелами с обеих сторон** (например, оператор присваивания (`=`), операторы сравнения (`==`), арифметические операторы и т.д.).
    - **Правильно**:

```
x = y + z
```

    - **Неправильно**:

```
x=y+z
```

2. Комментарии

2.1 Блочные комментарии

- **Блочные комментарии, как правило, относятся к коду, следующему за ними, и имеют тот же уровень отступа, что и код.**
- **Каждая строка блочного комментария начинается с `#` и одного пробела** (если это не отступленный текст внутри комментария).

2.2 Встроенные комментарии

- **Используйте встроенные комментарии экономно.**
- **Встроенный комментарий — это комментарий на той же строке, что и инструкция.**
- **Отделяйте встроенные комментарии от инструкции как минимум двумя пробелами:**

```
x = x + 1  # Увеличить x на 1
```

2.3 Строки документации (Docstrings)

- **Пишите строки документации для всех публичных модулей, функций, классов и методов.**
- **Docstrings должны описывать действие метода, а также все аргументы и возвращаемые значения.**
    - **Пример:**

```
def multiply(a: int, b: int) -> int:
    """Умножает два числа и возвращает результат.

    Аргументы:
        a (int): Первое число.
        b (int): Второе число.

    Возвращает:
        int: Результат умножения a и b.
```

```
    """
    return a * b
```

3. Соглашения по наименованию

3.1 Имена переменных

- **Используйте `snake_case` для имен переменных и функций.**

```
my_variable = 10
def my_function():
    pass
```

3.2 Имена функций и переменных

- **Имена функций должны быть в нижнем регистре, слова разделяются подчеркиванием** для улучшения читаемости.

3.3 Имена классов

- **Используйте `CamelCase` для имен классов:**

```
class MyClass:
    pass
```

3.4 Константы

- **Константы должны объявляться на уровне модуля и записываться в `UPPERCASE_WITH_UNDERSCORES`.**

```
MAX_CONNECTIONS = 100
```

4. Рекомендации по программированию

4.1 Избегайте прямого сравнения с `True`, `False` или `None`

- **Всегда используйте `if foo is None:`, а не `if foo == None:`.**
- **Для булевых значений используйте контекстное сравнение:**

```
if not x:  # Правильно
if x is False:  # Неправильно
```

4.2 Обработка исключений

- **Используйте исключения в ситуациях, когда программа должна вести себя иначе, чем обычно:**
  - Предпочитайте использование блоков `try/except`, а не явную проверку на наличие исключения.
  - **Пример:**

```
try:
    value = my_list[index]
except IndexError:
    handle_exception()
```

5. Организация кода

5.1 Структура модуля

- **Следуйте этому порядку при организации содержимого модуля:**
  1. Docstring модуля.
  2. Импорты на уровне модуля.
  3. Константы и глобальные переменные.
  4. Классы и функции.

5.2 Основная функция

- **При написании скрипта включите основную функцию и защитите точку входа с помощью `if __name__ == "__main__":`.**

```
def main():
    pass

if __name__ == "__main__":
    main()
```