PEP8 - общепринятый стиль кода на языке Python

Весь код на языке программирования Python пишется в соответствии с общепринятым стилем, который изложен в документе PEP8. Соответствие какому-то единому стандарту необходимо для повышения читаемости и понимаемости кода как самим автором, так и другими программистами.

1 Внешний вид кода

1.1 Отступы

Во-первых, оформление блоков кода делается 2 или 4 пробелами. Хотя возможно использование табуляции для проставления отступов, но этот способ является менее предпочтительным. Стоит заметить, что Python 3 не допускает использование табуляции и пробелов одновременно.

Во-вторых, есть два способа выравнивания элементов, обернутых в скобки (круглые, квадратные и фигурные). Первое – вертикальное выравнивание, второе – использование висячего отступа. Следует помнить, что во втором случае на первой линии не должно быть аргументов, а последующие строки должны выравниваться с одинаковым отступом.

Правильно:

```
# Вертикальное выравнивание
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
# Использование висячего отступа
# Добавлены пробелы для отделения аргументов от блока кода
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

Неправильно:

```
# Аргументы на первой линии запрещены,
# если не используется вертикальное выранивание
foo = long_function_name(var_one, var_two,
```

```
    var_three, var_four)

# Требуется больший отступ для выделения висячего отступа
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Опционально:

```
# Нет необходимости в большем количестве отступов
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

Закрывающие скобки в многострочных конструкциях могут находиться под первым непробельным символом последней строки списка либо быть под первым символом строки, начинающей многострочную конструкцию.

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )

my_another_list = [
    1, 2, 3,
    4, 5, 6,
]
result = another_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

1.2 Максимальная длина строки

Длина строки не должна превышать 79 символов. Строки документации и комментариев ограничиваются 72 символами. Для этого используют переносы строк внутри скобок. Однако, бинарные операторы и некоторые конструкции требуют использование обратного слеша для явного указания, что используется многострочная конструкция.

```
# Такие конструкции требуют использование обратного слеша
with open('/path/to/some/file/you/want/to/read') as file_1, \
```

```
        open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
sum_of_elements = value1 + \
    value2
```

С другой стороны, если выражение обернуто в скобки, то бинарные операторы уже не требуют обратного слеша.

```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

1.3 Пустые строки

Функции верхнего уровня и определения классов двумя пустыми строками. Методы внутри класса разделяются одной пустой строкой. Можно использовать пустые строки для логического разделения кода.

1.4 Кодировка

Кодировка Python должна быть UTF-8. Файлы не должны иметь объявления кодировки.

Начиная с версии Python 3.0 в стандартной библиотеке действует следующее соглашение: все идентификаторы обязаны содержать только ASCII символы, и означать английские слова везде, где это возможно (во многих случаях используются сокращения или неанглийские технические термины). Кроме того, строки и комментарии тоже должны содержать лишь ASCII символы. Исключения составляют: (a) test case, тестирующий не-ASCII особенности программы, и (б) имена авторов. Авторы, чьи имена основаны не на латинском алфавите, должны транслитерировать свои имена в латиницу.

Проектам с открытым кодом для широкой аудитории также рекомендуется использовать это соглашение.

1.5 Импорты

Каждый импорт, как правило, должен быть на отдельной строке.

Правильно:

```
import os
import sys
```

Неправильно:

```
import sys, os
```

В то же время, можно писать так:

```
from subprocess import Popen, PIPE
```

Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант.

Импорты должны быть сгруппированы в следующем порядке:

1. импорты из стандартной библиотеки
2. импорты сторонних библиотек
3. импорты модулей текущего проекта

Вставляйте пустую строку между каждой группой импортов.

2 Пробелы

2.1 Избегайте пробелов в следующих ситуациях

Непосредственно внутри круглых, квадратных или фигурных скобок.

Правильно:

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

Неправильно:

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

Непосредственно перед запятой, точкой с запятой или двоеточием:

Правильно:

```
if x == 4: print(x, y); x, y = y, x
```

Неправильно:

```
if x == 4 : print(x , y) ; x , y = y , x
```

Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции:

Правильно:

```
spam(1)
```

Неправильно:

```
spam (1)
```

Сразу перед открывающей скобкой, после которой следует индекс или срез:

Правильно:

```
dict['key'] = list[index]
```

Неправильно:

```
dict ['key'] = list [index]
```

Избегайте пробелов в конце строки. Например, из-за оставленного пробела после обратного слеша следующая строка может не считаться как продолжение предыдущей.

2.2 Вместе с бинарными операторами

Всегда окружайте эти бинарные операторы одним пробелом с каждой стороны: присваивания (=, +=, -= и другие), сравнения (==, <, >, !=, <>, <=, >=, in, not in, is, is not), логические (and, or, not).

Если используются операторы с разными приоритетами, попробуйте добавить пробелы вокруг операторов с самым низким приоритетом. Используйте свои собственные суждения, однако, никогда не используйте более одного пробела, и всегда используйте одинаковое количество пробелов по обе стороны бинарного оператора.

Правильно:

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Неправильно:

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Не используйте пробелы вокруг знака =, если он используется для обозначения именованного аргумента или значения параметров по умолчанию.

Правильно:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

Неправильно:

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

Не используйте составные инструкции (несколько команд в одной строке).

Правильно:

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

Неправильно:

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

3 Комментарии

Комментарии, противоречащие коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код!

Комментарии должны являться законченными предложениями. Если комментарий — фраза или предложение, первое слово должно быть написано с большой буквы, если только это не имя переменной, которая начинается с маленькой буквы (никогда не изменяйте регистр переменной!).

Если комментарий короткий, можно опустить точку в конце предложения. Блок комментариев обычно состоит из одного или более абзацев, составленных из полноценных предложений, поэтому каждое предложение должно оканчиваться точкой.

Ставьте два пробела после точки в конце предложения.

Программисты, которые не говорят на английском языке, пожалуйста, пишите комментарии на английском, если только вы не уверены на 120%, что ваш код никогда не будут читать люди, не знающие вашего родного языка.

3.1 Блоки комментариев

Блок комментариев обычно объясняет код (весь, или только некоторую часть), идущий после блока, и должен иметь тот же отступ, что и сам код. Каждая строчка такого блока должна начинаться с символа # и одного пробела после него (если только сам текст комментария не имеет отступа).

Абзацы внутри блока комментариев разделяются строкой, состоящей из одного символа #.

3.2 Inline комментарии

Старайтесь реже использовать подобные комментарии.

Такой комментарий находится в той же строке, что и инструкция. Inline комментарии должны отделяться по крайней мере двумя пробелами от инструкции. Они должны начинаться с символа # и одного пробела.

Комментарии в строке с кодом не нужны и только отвлекают от чтения, если они объясняют очевидное. Не пишите вот так:

```
x = x + 1                 # Increment x
```

Впрочем, такие комментарии иногда полезны:

```
x = x + 1                 # Compensate for border
```

3.3 Строки документации

Пишите документацию для всех публичных модулей, функций, классов, методов. Строки документации необязательны для приватных методов, но лучше написать, что делает метод. Комментарий нужно писать после строки с **def**.

PEP 257 объясняет, как правильно и хорошо документировать. Заметьте, очень важно, чтобы закрывающие кавычки стояли на отдельной строке. А еще лучше, если перед ними будет ещё и пустая строка, например:

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.

"""
```

Для однострочной документации можно оставить закрывающие кавычки на той же строке.

4 Соглашение по именованию

Соглашения по именованию переменных в Python немного туманны, поэтому их список никогда не будет полным — тем не менее, ниже мы приводим список рекомендаций, действующих на данный момент. Новые модули и пакеты должны быть написаны согласно этим стандартам, но если в какой-либо уже существующей библиотеке эти правила нарушаются, предпочтительнее писать в едином с ней стиле.

- Имена модулей должны записываться коротко, маленькими буквами. Подчеркивания допускаются, если это улучшает читаемость. В именах пакетов предпочтительнее не использовать подчеркивание.
- Имена классов пишутся в виде много идущих подряд слов, каждое с заглавной буквы. Если в названии есть аббревиатура, то она пишется заглавными буквами.
- Исключения тоже являются классами.
- Имена функций пишутся с маленькой буквы, между словами используют подчеркивание.
- Стиль именования функций, когда слова пишутся подряд с заглавной буквы, кроме первого слова, допустим только в тех местах, где уже преобладает такой стиль, для сохранения совместимости.
- Имена переменных или открытых атрибутов класса состоят из маленьких букв, слова разделяются символами подчеркивания.
- Имена глобальных констант состоят из заглавных букв, слова разделяются символами подчеркивания.

Например:

- mymodule – модуль или пакет
- UserClassName – класс
- ExceptionsAreAlsoClasses – исключение
- function_name – функция или метод
- notDesiredFunctionName – менее предпочтительное имя функции
- variable_name – переменная или открытый атрибут класса
- GLOBAL_CONSTANT – глобальаня константа

4.1 Символы подчеркивания

Кроме разделения слов внутри названий символы подчеркивания могут выполнять другие роли:

- Один символ подчеркивания в начале атрибута или метода говорит о том, что этот атрибут или этот метод для каких-то внутренних нужд.
- Один символ подчеркивания в конце названия используется, чтобы избежать конфликта с каким-либо зарезервированным словом.

- Два символа подчеркивания ставятся в начале названий скрываемых методов и атрибутов.
- По два символа подчеркивания в начале и в конце названия используются у так называемых «магических» методов. Эти методы описаны в документации, и изобретать свои не следует.

4.2 Общие замечания

Никогда не используйте символы l (маленькая латинская буква «эль»), O (заглавная латинская буква «о») или I (заглавная латинская буква «ай») как однобуквенные идентификаторы. В некоторых шрифтах эти символы неотличимы от цифр один и нуль. Если очень нужно l, пишите вместо неё заглавную L.

Имена должны содержать только символы ASCII и означать только английские слова.

Если имя – часть API, то оно должно быть согласовано со стилем кода интерфейса, а не реализации.

Имена, которые видны пользователю как часть открытого API, должны следовать конвенциям, которые отражают использование, а не реализацию.

Список литературы

Guido Van Rossum, Barry Warsaw и Nick Coghlan (2001). *PEP 8 – Style Guide for Python Code*. URL: `https://www.python.org/dev/peps/pep-0008/` (дата обр. 15.10.2019).