PEP8 - стиль кода в языке Python

Введение

Этот документ описывает соглашение о том, как писать код для языка python.

Этот документ создан на основе рекомендаций Гуидо ван Россума с добавлениями от Барри.

Ключевая идея Гуидо такова: код читается намного больше раз, чем пишется. Собственно, рекомендации о стиле написания кода направлены на то, чтобы улучшить читабельность кода и сделать его согласованным между большим числом проектов. В идеале, весь код будет написан в едином стиле, и любой сможет легко его прочесть.

Внешний вид кода

Отступы

Используйте 4 пробела на один уровень отступа. В старом коде, который вы не хотите трогать, можно продолжить пользоваться 8 пробелами для отступа.

Табуляция или пробелы?

Никогда не смешивайте символы табуляции и пробелы.

Самый распространенный способ отступов — пробелы. На втором месте — отступы только с использованием табуляции. Код, в котором используются и те, и другие типы отступов, должен быть исправлен так, чтобы отступы в нем были расставлены только с помощью пробелов.

В новых проектах для отступов мы настоятельно рекомендуем использовать пробелы. К тому же, многие редакторы позволяют легко делать отступы.

Максимальная длина строки

Ограничьте максимальную длину строки 79 символами.

Пока еще существует немало устройств, где длина строки равна 80 символам; к тому же, ограничив ширину окна 80 символами, мы сможем расположить несколько окон рядом друг с другом. Автоматический перенос строк на таких устройствах нарушит форматирование, и код будет труднее понять. Так что, пожалуйста, ограничьте длину строки 79 символами, и 72 символами в случае длинных блоков текста (строки документации или комментарии).

Пустые строки

Отделяйте функции (верхнего уровня, не функции внутри функций) и определения классов двумя пустыми строчками.

Используйте (без энтузиазма) пустые строки в коде функций, чтобы отделить друг от друга логические части.

Кодировки (PEP 263)

Код ядра python всегда должен использовать ASCII или Latin-1 кодировку (также известную как ISO-8859-1). Начиная с версии python 3.0, предпочтительной является кодировка UTF-8 (смотрите PEP 3120).

Import-секции

Импортирование разных модулей должно быть на разных строчках, например:

правильно:
```
import os
import sys
```
неправильно:
```
import os, sys
```
В то же время, можно писать вот так:
```
from subprocess import Popen, PIPE
```

Импортирование всегда нужно делать сразу после комментариев к модулю и строк документации, перед объявлением глобальных переменных и констант.

Группируйте импорты в следующем порядке:

- импорты стандартной библиотеки
- импорты сторонних библиотек
- импорты модулей текущего проекта

Вставляйте пустую строку между каждой группой импортов.

Пробелы в выражениях и инструкциях

Избегайте использования пробелов в следующих ситуациях:

1. Сразу после или перед скобками (обычными, фигурными и квадратными)
```
можно:
    spam(ham[1], {eggs: 2})

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

2. Сразу перед запятой, точкой с запятой, двоеточием:
```
if x == 4: print x, y; x, y = y, x

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

3. Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции:
```
spam(1)

spam (1)
```

4. Сразу перед открывающей скобкой, после которой следует индекс или срез:
```
dict['key'] = list[index]

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

5. Использование более одного пробела вокруг оператора присваивания (или любого другого) для того, чтобы выровнять его с другим таким же оператором на соседней строке:
```
x = 1
y = 2
long_variable = 3

x             = 1
y             = 2
long_variable = 3
```

Прочие рекомендации:

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

2. Ставьте пробелы вокруг арифметических операций.

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

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

3. Не используйте пробелы для отделения знака =, когда он употребляется для обозначения аргумента-ключа (keyword argument) или значения параметра по умолчанию.

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

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

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

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

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

Комментарии

Комментарии, которые противоречат 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.

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.

Блок 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 лучше отделять строкой, состоящей из одного символа #.

Комментарии в строке с кодом

Старайтесь реже использовать подобные комментарии.

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

```
x = x + 1                 # Увеличиваем X на один
```

Впрочем, иногда такие комментарии полезны:

```
x = x + 1                  # Место для рамки окна
```

Имена

Соглашения об именах переменных в python немного туманны, поэтому их список никогда не будет полным — тем не менее, ниже мы приводим список рекомендаций, действующих на данный момент.

Описание: Стили имен

Существует много разных стилей. Поможем вам распознать, какой стиль именования используется, независимо от того, для чего он используется.

Обычно различают следующие стили:

- b (одиночная маленькая буква)
- B (одиночная заглавная буква)
- lowercase (слово в нижнем регистре)
- lower_case_with_underscores (слова из маленьких букв с подчеркиваниями)
- UPPERCASE (заглавные буквы)
- UPPERCASE_WITH_UNDERSCORES (слова из заглавных букв с подчеркиваниями)
- CapitalizedWords (слова с заглавными буквами, или CapWords, или CamelCase [5]). Замечание: когда вы используете аббревиатуры в таком стиле, пишите все буквы аббревиатуры заглавными — HTTPServerError лучше, чем HttpServerError.
- mixedCase (отличается от CapitalizedWords тем, что первое слово начинается с маленькой буквы)
- Capitalized_Words_With_Underscores (слова с заглавными буквами и подчеркиваниями — уродливо!)

Стили имен

Имена, которых следует избегать

Никогда не используйте символы l (малелькая латинская буква «эль»), O (заглавная латинская буква «о») или I (заглавная латинская буква «ай») как однобуквенные идентификаторы.

В некоторых шрифтах эти символы неотличимы от цифры один и нуля (и символа вертикальной палочки, — прим. перев.) Если очень нужно использовать l имена, пишите вместо неё заглавную L.

Имена модулей и пакетов

Модули должны иметь короткие имена, состоящие из маленьких букв. Можно использовать и символы подчеркивания, если это улучшает читабельность. То же, за исключением символов подчеркивания, относится и к именам пакетов.

Имена классов

Все имена классов должны следовать соглашению CapWords почти без исключений. Классы внутреннего использования могут начинаться с символа подчеркивания.

Имена глобальных переменных

Будем надеяться, что такие имена используются только внутри одного модуля. Руководствуйтесь теми же соглашениями, что и для имен функций.

Имена функций

Имена функций должны состоять из маленьких букв, а слова разделяться символами подчеркивания — это необходимо, чтобы увеличить читабельность.

Аргументы функций и методов

Если имя аргумента конфликтует с зарезервированным ключевым словом python, обычно лучше добавить в конец имени символ подчеркивания, чем исказить написание слова или использовать аббревиатуру. Таким образом, print_ лучше, чем prnt.

Общие рекомендации

- Код должен быть написан так, чтобы не зависеть от разных реализация языка (PyPy, Jython, IronPython, Pyrex, Psyco и пр.).
- Пользуйтесь ''.startswith() и ''.endswith() вместо обработки частей строк (string slicing) для проверки суффиксов или префиксов. startswith() и endswith() выглядят чище и порождают меньше ошибок. Например:

```
да: if foo.startswith('bar'):

нет: if foo[:3] == 'bar':
```

- Для последовательностей (строк, списков, кортежей) можно использовать тот факт, что пустая последовательность есть false:

```
да: if not seq:
    if seq:

нет: if len(seq):
     if not len(seq):
```

- Не сравнивайте логические типы с True и False с помощью ==:

```
да: if greeting:
нет: if greeting == True:
```

Copyright