**A PEP 8.**

Esta PEP começa indo diretamente ao ponto, com a seguinte citação:

A má consistência é o pesadelo das pequenas mentes.

Segundo Guido, um código é lido mais vezes do que é escrito, e para facilitar a leitura dos códigos foi criado este guia. Um guia de estilo é tudo sobre consistência, mas há casos em que essa consistência não é aplicável e, cabe a você escolher quando segui-lo.

**Tabs ou Espaços?**

Espaços são preferíveis, use tabulações somente se for para permanecer consistente com o restante do código. Python 3 desabilita a mistura de tabulações e espaços para indentação, códigos em Python 2 devem ser convertidos para o uso de espaços exclusivamente.

**Indentação.**

Python é uma linguagem indentada, isto é, o espaçamento no começo de cada linha é importante para a execução do código, o guia define que, esta indentação deve ser de 4 espaços. Caso seja necessário a quebra de linha dentro de parênteses, colchetes e chaves, é necessário que o primeiro argumento da segunda linha esteja alinhado com o parênteses de abertura, é possível também colocar todos os argumentos em novas linhas e acrescentar um nível de indentação. Em funções, os parâmetros devem ter um nível de indentação à mais que as instruções do bloco.

# Válido

foo = funcao\_qualquer(

variavel1, variavel2,

variavel3, variavel4)

# Válido

foo = funcao\_qualquer(variavel1, variavel2,

variavel3, variavel4)

# Válido

def funcao(variavel1, variavel2,

variavel3, variavel4):

pass

# Inválido, não dá pra distinguir a indentação

def funcao(variavel1, variavel2,

variavel3, variavel4):

pass

# Inválido, argumentos na segunda linha são proibidos

# se não houver alinhamento vertical.

foo = funcao\_qualquer(variavel1, variavel2,

variavel1, variavel2)

As mesmas regras se aplicam a listas e tuplas:

minha\_lista = [

1, 2, 3

4, 5, 6

7, 8, 9

]

# Ou ainda

minha\_lista = [

1, 2, 3

4, 5, 6

7, 8, 9

]

A PEP não cobre casos em que a condição de um if-else é quebrada em múltiplas linhas, mas sugere que seja colocado um comentário entre o if e as instruções ou ainda, colocar indentação extra na segunda linha:

# Sem indentação extra, causa conflito visual.

if (isto and

outro):

faca()

# Coloque um comentário, diminui o conflito visual.

if (isto and

outro):

# Já que ambos são verdadeiros, execute.

faca()

# Indentação extra an segunda linha.

if (isto

and outro):

faca()

**Limitação no Tamanho das Linhas.**

Limite todas as linhas a no máximo 79 colunas. Para blocos de texto longos, docstrings e comentários, limite à 72 caracteres. Limitar as colunas é útil em editores de texto, possibilitando a abertura de múltiplas abas para edição, porém, estes limites são negociáveis entre a equipe.

Para quebrar uma expressão em mais linhas, utilize a contra-barra '\' ou coloque a expressão entre parênteses.

with open("path/to/file/one/file1.txt") as file\_one, \

open("path/to/file/two/file2.txt") as file\_two:

file\_two.write(file\_one.read())

**A Linha deve quebrar antes ou depois dos operadores?**

Seguindo a indicação de Donald Knuth, em seu livro *Computers and Typesetting,* sempre quebre a linha antes do operador, para facilitar a leitura.

Seguindo as tradições dos matemáticos:

lucro = (salario\_bruto

+ (dividendo - dividendo\_qualificado)

- deducao\_ira

- juros\_emprestimo\_estudantil

)

**Linhas Brancas.**

Envolva, funções de alto nível e definições de classe, com duas linhas brancas. Definições de métodos dentro de uma classe com uma linha branca. Linhas extras podem ser utilizadas para separar grupos de funções relacionadas e, dentro de funções, para indicar seções lógicas. Linhas podem ser omitidas entre vários códigos de linha única relacionados.

**Codificação.**

Use sempre UTF-8 em Python 3, e ASCII em Python 2.

Arquivos utilizando ASCII em Python 2, ou UTF-8 em Python não devem declarar a codificação.

**Importações.**

Imports devem ser feitos em linhas únicas, por exemplo:

# Correto:

import os

import sys

# Incorreto:

import sys, os

No entanto, é permitido importar vários objetos e funções de uma mesma classe:

from subprocess import Popen, PIPE

Imports devem estar sempre no topo do arquivo, sucedendo comentários de módulo e docstrings. Sendo agrupados na seguinte ordem:

* Bibliotecas Padrão.
* Bibliotecas Externas.
* Bibliotecas Locais.

Imports absolutos são recomendados, pois são mais legíveis e retornam mensagens de erro melhores.

import mypkg.sibling

from mypkg import sibling

from mypkg.sibling import example

Imports com asterísco devem ser evitados, pois não deixam claro o que está no escopo, confundindo leitores e ferramentas automáticas.

**Variáveis mágicas.**

Devem ser postas depois do docstring do módulo e, antes das importações, exceto as importações de \_\_future\_\_.

"""Este é um módulo de exemplo, este módulo faz coisas."""

from \_\_future\_\_ import barry\_as\_FLUFL

\_\_all\_\_ = ['a', 'b', 'c']

\_\_version\_\_ = '0.1'

\_\_author\_\_ = 'Comandante Umbiggles'

import os

import sys

**Espaços em branco em expressões.**

Evite espaços em branco em excesso nos seguintes casos:

* Imediatamente dentro de parênteses, colchetes e chaves.

# Válido.

compra(hamburguer[0], {ovos: 2})

# Inválido.

compra( hamburguer[ 0 ], { ovos: 2 } )

* Entre vírgulas e parênteses de fechamento.

# Válido

foo = (0,)

# Inválido

bar = (0, )

* Imediatamente antes de vírgulas, dois-pontos e ponto e vírgula:

# Válido

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

# Inválido

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

* No entanto, em um corte de listas o sinal dois-pontos age como um operador binário, e deve ter uma quantidade de espaços iguais em ambos os lados(tratando-o como operador de menor prioridade). Em cortes estendidos ambos os sinais devem ter o mesmo espaçamento, exceto quando o parâmetro é omitido, então o espaço também deve ser.

# Casos Corretos.

presunto[1:9], presunto[1:9:3], presunto, presunto[1::3], presunto[1:9:]

presunto[minimo:maximo], ham[minimo:maximo:], ham[minimo::passo]

presunto[minimo+espacamento : maximo+espacamento]

presunto[: funcao\_maximo(x) : funcao\_passo(x)], presunto[:: funcao\_passo(x)]

presunto[minimo + espacamento : maximo + espacamento]

# Casos Inválidos.

presunto[minimo + espacamento:maximo + espacamento]

presunto[1: 9], presunto[1 :9], presunto[1:9 :3]

presunto[minimo : : maximo]

presunto[ : maximo]

* Imediatamente antes da abertura de um parênteses que começa a lista de argumentos da chamada de uma função.

# Correto.

compra(1)

# Incorreto.

compra (1)

* Imediatamente antes da abertura de um parênteses que inicia uma indexação ou corte.

# Válido.

dicio[chave] = lista[indice]

# Inválido.

dicio [chave] = lista [indice]

* Mais de um espaço ao redor de um operador de definição(=) ou qualquer outro operador para alinhá-lo a outros.

# Correto.

x = 1

y = 2

variavel\_longa = 3

# Incorreto.

x = 1

y = 2

variavel\_longa = 3

* Evite espaços no final das linhas. Por exemplo, uma contra-barra seguida por um espaço e uma quebra de linha, não funcionará como o esperado.
* Sempre envolva operadores binários por espaços.
* Caso utilize operadores de diferentes prioridades, considere acrescentar espaços ao redor daqueles de menor prioridade.

# Correto.

i = i + 1

enviado += 1

x = x\*2 - 1

hipot = x\*x + y\*y

c = (a+b) \* (a-b)

# Errado.

i=i+1

enviado +=1

x = x \* 2 - 1

hipot = x \* x+y \* y

c = (a + b) \* (a - b)

* Não use espaços ao redor do operador de atribuição(=) quando este indicar parâmetros, ou valores padrão de parâmetros, de funções, a menos que utilize anotações de argumento.

# Correto

def func(y, x=1):

return magica(a=y,b=x)

# Correto com anotação de argumento

def func(y, x: int = 1);

pass

# Incorreto

def func(y, x = 1):

return magica(a = y,b = x)

* Expressões compostas(múltiplas expressões em uma única linha) são desencorajadas.

# Não faça

if a == b: faca()

faca\_um(); faca\_dois(); faca\_tres()

* Apesar de estar ok utilizar if/for/while pequenos juntos em um única linha, nunca faça isso com expressões multi-clausulas. Também evite agrupar linhas tão grandes.

if foo == 'blah': faz\_qualquer\_coisa()

for x in lista: total += x

while t < 10: t = demore()

**Nomes de variáveis e funções.**

Há várias convenções de nomes aceitas pela PEP 8, mas a mais comum é a snake\_case, onde as palavras são separadas por sublinha(\_) e a completamente em caixa baixa. É comum ainda que se comece o nome de uma variável com sublinha, indicando que esta variável é de uso interno(como se fosse privada, mas Python não possui encapsulamento, mantenha em mente que isto é apenas uma convenção, ainda é possível acessar estas variáveis).

**Nomes de Modulos e Pacotes.**

A PEP encoraja que sejam utilizados nomes curtos, em caixa baixa. Sublinha pode ser utilizado para promover a legibilidade, exceto em nomes de pacotes.

**Nomes de Classes.**

Nomes de classe normalmente seguem o padrão CamelCase, com a primeira letra de cada palavra em caixa alta.

**Comentários.**

Comentários contraditórios mais atrapalham do que ajudam. Sempre mantenha os comentários atualizados com o código.

Comentários devem ser sentenças completas, com a primeira letra maiúscula, a menos que a sentença comece com um identificador.

Comentários em blocos geralmente consistem em vários parágrafos terminados em ponto final, cada parágrafo deve ser separado por uma linha com um único sinal de tralha(#). Coloque dois espaços após o ponto-final de cada parágrafo, exceto o parágrafo final.

Garanta que seus comentários são concisos e coesos e, dê preferência a escrever comentários em inglês, exceto quando tiver a certeza que estrangeiros não lerão seu código.

Os comentários também devem ter indentação alinhada àquilo a que se referem.

Comentários inline devem ter pelo menos 2 espaços de distância do código da linha. E são desnecessários caso sejam óbvios.

# Inútil.

x = x + 1 # Incrementa x

# Útil.

x = x + 1 # Compensa a borda